3-е издание

Android

ПРОГРАММИРОВАНИЕ для ПРОФЕССИОНАЛОВ

Б. ФИЛЛИПС, К. СТЮАРТ,
К. МАРСИКАНО

Big nerd ranch

3rd Edition

Android Programming

The Big Nerd Ranch Guide

Bill Phillips, Chris Stewart, & Kristin Marsicano

Б. Филлипс, К. Стюарт, К. Марсикано

Android

ПРОГРАММИРОВАНИЕ для ПРОФЕССИОНАЛОВ

3-е издание

ПИТЕР®

Санкт-Петербург • Москва • Екатеринбург • Воронеж
Нижний Новгород • Ростов-на-Дону
Самара • Минск

2017

ББК 32.973.2-018.2
УДК 004.451
Ф53

Филлипс Б., Стюарт К., Марсикано К.

Ф53 Android. Программирование для профессионалов. 3-е изд. — СПб.: Питер, 2017. — 688 с.: ил. — (Серия «Для профессионалов»).

ISBN 978-5-4461-0413-0

Когда вы приступаете к разработке приложений для Android — вы как будто оказываетесь в чужой стране: даже зная местный язык, на первых порах всё равно чувствуете себя некомфортно. Такое впечатление, что все окружающие знают что-то такое, чего вы никак не понимаете. И даже то, что вам уже известно, в новом контексте оказывается попросту неправильным.

Третье издание познакомит вас с интегрированной средой Android Studio, которая сильно облегчает разработку приложений. Вы не только изучите основы программирования, но и узнаете о возможностях самых распространенных версий Android; новых инструментах, таких как макеты с ограничениями и связывание данных; модульном тестировании; средствах доступности; архитектурном стиле MVVM; локализации; новой системе разрешений времени выполнения. Все учебные приложения были спроектированы таким образом, чтобы продемонстрировать важные концепции и приемы программирования под Android и дать опыт их практического применения.

12+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.973.2-018.2
УДК 004.451

Права на издание получены по соглашению с Pearson Education Inc. Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

Информация, содержащаяся в данной книге, получена из источников, рассматриваемых издательством как надежные. Тем не менее, имея в виду возможные человеческие или технические ошибки, издательство не может гарантировать абсолютную точность и полноту приводимых сведений и не несет ответственности за возможные ошибки, связанные с использованием книги.

ISBN 978-0134706054 англ.
ISBN 978-5-4461-0413-0

© 2017 Big Nerd Ranch, LLC
© Перевод на русский язык ООО Издательство «Питер», 2017
© Издание на русском языке, оформление ООО Издательство «Питер», 2017
© Серия «Для профессионалов», 2017

Краткое содержание

Оглавление

Посвящается старому проигрывателю на моем рабочем столе. Благодарю тебя за то, что ты был со мной все это время. Обещаю скоро купить тебе новую иглу.

Б. Ф.

Посвящается моему отцу Дэвиду, научившему меня тому, как важно упорно работать, и моей матери Лизе, которая требовала, чтобы я всегда поступал правильно.

К. С.

Посвящается моему отцу Дэйву Вадасу — это он убедил меня стать программистом. Моя мать Джоан Вадас подбадривала меня на всех этапах жизненного пути (а также напоминала, что от просмотра серии «Золотых девочек» жизнь всегда становится лучше).

К. М.

Благодарности

К третьему изданию книги мы уже привыкли говорить то, что скажем сейчас. Тем не менее сказать об этом нужно: книги создаются не одними авторами. Своим существованием они обязаны многочисленными помощникам и коллегам, без которых мы не смогли бы представить и написать весь этот материал.

- Спасибо Брайану Харди (Brian Hardy), которому вместе с Биллом хватило смелости воплотить в реальность самое первое издание этой книги. Начав с нуля, Брайан и Билл сделали замечательный проект.
- Мы благодарны нашим коллегам-преподавателям и участникам группы разработки Android Эндрю Лунсфорду (Andrew Lunsford), Болоту Керимбаеву (Bolot Kerimbaev), Брайану Гарднеру (Brian Gardner), Дэвиду Гринхальгу (David Greenhalgh), Джошу Скину (Josh Skeen), Мэтту Комптону (Matt Compton), Полу Тернеру (Paul Turner) и Рашаду Куртону (Rashad Cureton). (Скоро, Рашад... Уже скоро.) Спасибо за их терпение в преподавании незавершенных рабочих материалов, за их бесценные предложения и исправления. Многим людям за всю жизнь не встретиться с такой талантливой и интересной командой. Благодаря таким людям повседневная работа на Big Nerd Ranch приносит радость.
- Отдельное спасибо Эндрю, который проработал всю книгу и обновил все снимки экранов Android Studio. Мы ценим его скрупулезность, внимание к деталям и сарказм.
- Спасибо Заку Саймону (Zack Simon), невероятно талантливому и деликатному дизайнеру Big Nerd Ranch. Мы и не знали, что Зак обновил памятку, которая прилагается к книге. Если она вам понравится, свяжитесь с Заком и скажите ему об этом сами. Но мы также поблагодарим Зака прямо здесь: спасибо, Зак!
- Спасибо Кар Лун Вону (Kar Loong Wong) за переработку экрана со списком преступлений. Чем больше нам помогает Кар, тем лучше выглядят приложения в книге. Спасибо, Кар!
- Благодарим Марка Далримпла (Mark Dalrymple), который проверил наше описание ограничений макетов на точность и полноту. Если вы когда-нибудь встретите Марка, попросите его проверить ваше описание ограничений макетов. У него это отлично получается! А если у вас такого описания еще нет — попросите его сделать зверюшку из воздушного шарика.
- Спасибо Аарону Хиллегассу (Aaron Hillegass). В практическом плане эта книга была бы невозможной без Big Nerd Ranch — компании, основанной Аароном.
- Наш руководитель проекта Элизабет Холадей (Elizabeth Holaday). Знаменитый Уильям С. Берроуз иногда создавал свои произведения весьма оригинальным способом: он разделял свою работу на части, подбрасывал их в воз-

дух и публиковал то, что получится. Без такого сильного руководителя, как Лиз, наша суматоха и простодушный энтузиазм заставили бы нас прибегнуть к этому способу. Она помогала нам сосредоточиться на том, что действительно интересует читателей, добиться четкости и ясности формулировок.

- Спасибо нашему выпускающему редактору Анне Бентли (Anna Bentley) и корректору Симоне Пэймент (Simone Payment) за то, что они сгладили многие шероховатости нашего изложения.
- Крис Лопер (Chris Loper) из *IntelligentEnglish.com* спроектировал и создал печатную и электронную версии книги. Это инструментарий DocBook также существенно упростил нашу жизнь.

Остается лишь поблагодарить наших студентов. Между нами существует обратная связь: мы преподаем материал, они высказывают свое мнение. Без этой обратной связи книга не появилась бы на свет, и не была бы переиздана. Если книги Big Nerd Ranch действительно отличаются от других (как мы надеемся), то именно благодаря этой обратной связи. Спасибо вам!

Изучение Android

Начинающему программисту Android предстоит основательно потрудиться. Изучать Android — все равно что жить в другой стране: даже если вы говорите на местном языке, на первых порах вы все равно не чувствуете себя как дома. Такое впечатление, что все окружающие понимают что-то такое, чего вы еще не усвоили. И даже то, что уже известно, в новом контексте оказывается попросту неправильным.

У Android существует определенная культура. Носители этой культуры общаются на Java, но знать Java недостаточно. Чтобы понять Android, необходимо изучить много новых идей и приемов. Когда оказываешься в незнакомой местности, полезно иметь под рукой путеводитель.

Здесь на помощь приходим мы. Мы, сотрудники Big Nerd Ranch, считаем, что каждый программист Android должен:

- *писать* приложения для Android;
- *понимать*, что он пишет.

Этот учебник поможет вам в достижении обеих целей. Мы обучали тысячи профессиональных программистов Android. Мы проведем вас по пути разработки нескольких приложений Android, описывая новые концепции и приемы по мере надобности. Если на пути нам встретятся какие-то трудности, если что-то покажется слишком сложным или нелогичным, мы постараемся объяснить, почему все именно так и не иначе.

Такой подход позволит вам с ходу применить полученные сведения — вместо того, чтобы, накопив массу теоретических знаний, разбираться, как же их использовать на практике. Перевернув последнюю страницу, вы будете обладать опытом, необходимым для дальнейшей работы в качестве Android-разработчика.

Предварительные условия

Для работы с этой книгой читатель должен быть знаком с языком Java, включая такие концепции, как классы и объекты, интерфейсы, слушатели, пакеты, внутренние классы, анонимные внутренние классы и обобщенные классы.

Без знания этих концепций вы почувствуете себя в джунглях начиная со второй страницы. Лучше начните с вводного учебника по Java и вернитесь к этой книге после его прочтения. Сейчас имеется много превосходных книг для начинающих; подберите нужный вариант в зависимости от своего опыта программирования и стиля обучения.

Если вы хорошо разбираетесь в концепциях объектно-ориентированного программирования, но успели малость подзабыть Java, скорее всего, все будет нормально. Мы приводим краткие напоминания о некоторых специфических возможностях Java (таких, как интерфейсы и анонимные внутренние классы). Держите учебник по Java наготове на случай, если вам понадобится дополнительная информация во время чтения.

Что нового в третьем издании?

В третьем издании в распоряжении разработчика появилась пара новых инструментов: макеты с ограничениями (и связанный с ними редактор) и связывание данных. Также добавились новые главы, посвященные модульному тестированию, средствам доступности, архитектурному стилю MVVM и локализации. Ближе к концу книги также был добавлен материал, посвященный новой системе разрешений времени выполнения. Наконец, в книге появились новые упражнения и врезки «Для любознательных», а многие из существующих были переработаны.

Как работать с книгой

Эта книга не справочник. Мы старались помочь в преодолении начального барьера, чтобы вы могли извлечь максимум пользы из существующих справочников и пособий. Книга основана на материалах пятидневного учебного курса в Big Nerd Ranch. Соответственно предполагается, что вы будете читать ее с самого начала. Каждая глава базируется на предшествующем материале, и пропускать главы не рекомендуется.

На наших занятиях студенты прорабатывают эти материалы, но в обучении также участвуют и другие факторы — специальное учебное помещение, хорошее питание и удобная доска, группа заинтересованных коллег и преподаватель, отвечающий на вопросы.

Желательно, чтобы ваша учебная среда была похожа на нашу. В частности, вам стоит хорошенько высыпаться и найти спокойное место для работы. Следующие факторы тоже сыграют положительную роль:

- Создайте учебную группу с друзьями или коллегами.
- Выделяйте время, когда вы будете заниматься исключительно чтением книги.
- Примите участие в работе форума книги на сайте *forums.bignerdranch.com*.
- Найдите специалиста по Android, который поможет вам в трудный момент.

Структура книги

В этой книге мы напишем восемь приложений для Android. Два приложения очень просты, и на их создание уходит всего одна глава. Другие приложения часто

оказываются более сложными, а самое длинное приложение занимает 13 глав. Все приложения спроектированы так, чтобы продемонстрировать важные концепции и приемы и дать опыт их практического применения.

GeoQuiz — в первом приложении мы исследуем основные принципы создания проектов Android, активности, макеты и явные интенты.

CriminalIntent — самое большое приложение в книге, предназначено для хранения информации о проступках ваших коллег по офису. Вы научитесь использовать фрагменты, интерфейсы «главное-детализированное представление», списковые интерфейсы, меню, камеру, неявные интенты и многое другое.

BeatBox — наведите ужас на своих врагов и узнайте больше о фрагментах, воспроизведении мультимедийного контента, архитектуре MVVM, связывании данных, тестировании, темах и графических объектах.

NerdLauncher — нестандартный лаунчер, раскроет тонкости работы системы интентов и задач.

PhotoGallery — клиент Flickr для загрузки и отображения фотографий из общедоступной базы Flickr. Приложение демонстрирует работу со службами, многопоточное программирование, обращения к веб-службам и т. д.

DragAndDraw — в этом простом графическом приложении рассматривается обработка событий касания и создание нестандартных представлений.

Sunset — в этом «игрушечном» приложении вы создадите красивое представление заката над водой, а заодно освоите тонкости анимации.

Locatr — приложение позволяет обращаться к сервису Flickr за изображениями окрестностей вашего текущего местонахождения и отображать их на карте. Вы научитесь пользоваться сервисом геопозиционирования и картами.

Упражнения

Многие главы завершаются разделом с упражнениями. Это ваша возможность применить полученные знания, покопаться в документации и отработать навыки самостоятельного решения задач.

Мы настоятельно рекомендуем выполнять упражнения. Сойдя с проторенного пути и найдя собственный способ, вы закрепите учебный материал и приобретете уверенность в работе над собственными проектами.

Если же вы окажетесь в тупике, вы всегда сможете обратиться за помощью на форум *forums.bignerdranch.com*.

А вы любознательны?

В конце многих глав также имеется раздел «Для любознательных». В нем приводятся углубленные объяснения или дополнительная информация по темам, представленным в главах. Содержимое этих разделов не является абсолютно необходимым, но мы надеемся, что оно покажется вам интересным и полезным.

Стиль программирования

Существует два ключевых момента, в которых наши решения отличаются от повсеместно встречающихся в сообществе Android.

Мы используем анонимные классы для слушателей. В основном это дело вкуса. На наш взгляд, код получается более стройным. Реализация метода слушателя размещается непосредственно там, где вы хотите ее видеть. В высокопроизводительных приложениях анонимные внутренние классы могут создать проблемы, но в большинстве случаев они работают нормально.

После знакомства с фрагментами в главе 7 мы используем их во всех пользовательских интерфейсах. Фрагменты не являются абсолютно необходимыми, но, на наш взгляд, это ценный инструмент в арсенале любого Android-разработчика. Когда вы освоитесь с фрагментами, работать с ними несложно. Фрагменты имеют очевидные преимущества перед активностями, включая гибкость при построении и представлении пользовательских интерфейсов, так что дело того стоит.

Типографские соглашения

Все листинги с кодом и разметкой XML выводятся моноширинным шрифтом. Код или разметка XML, которые вы должны ввести, выделяется жирным шрифтом. Код или разметка XML, которые нужно удалить, перечеркиваются. Например, в следующей реализации метода вызов `makeText(…)` удаляется, а вместо него добавляется вызов `checkAnswer(true)`:

```
@Override
public void onClick(View v) {
    ~~Toast.makeText(QuizActivity.this, R.string.incorrect_toast,~~
                ~~Toast.LENGTH_SHORT).show();~~
    checkAnswer(true);
}
```

Версии Android

В этой книге программирование для Android рассматривается для всех распространенных версий Android. На момент написания книги это версии Android 4.4 (KitKat) — Android 7.1 (Nougat). И хотя старые версии Android продолжают занимать некоторую долю рынка, на наш взгляд, хлопоты по поддержке этих версий не оправдываются. За дополнительной информацией о поддержке версий Android, предшествующих 4.4, обращайтесь к предыдущим изданиям книги. Второе издание было ориентировано на Android 4.1 и выше, а первое — на Android 2.3 и выше.

Даже после выхода новых версий Android приемы, изложенные в книге, будут работать благодаря политике обратной совместимости Android (подробности см. в главе 6). На сайте *forums.bignerdranch.com* будет публиковаться информация об изменениях, а также комментарии по поводу использования материала книги с последними версиями.

Необходимые инструменты

Прежде всего вам понадобится Android Studio — интегрированная среда разработки для Android-программирования, созданная на базе популярной среды IntelliJ IDEA.

Установка Android Studio включает в себя:

- Android SDK — последнюю версию Android SDK.
- Инструменты и платформенные средства Android SDK — средства отладки и тестирования приложений.
- Образ системы для эмулятора Android, который позволяет создавать и тестировать приложения на различных виртуальных устройствах.

На момент написания книги среда Android Studio находилась в активной разработке и часто обновлялась. Учтите, что ваша версия Android Studio может отличаться от описанной в книге. За информацией об отличиях обращайтесь на сайт *forums.bignerdranch.com*.

Загрузка и установка Android Studio

Пакет Android Studio доступен на сайте разработчиков Android по адресу *developer.android.com/sdk/*.

Возможно, вам также придется установить пакет Java Development Kit (JDK8), если он еще не установлен в вашей системе; его можно загрузить на сайте *www.oracle.com*.

Если у вас все равно остаются проблемы, обратитесь по адресу *developer.android.com/sdk/* за дополнительной информацией.

Загрузка старых версий SDK

Android Studio предоставляет SDK и образ эмулируемой системы для последней платформы. Однако не исключено, что вы захотите протестировать свои приложения в более ранних версиях Android.

Компоненты любой платформы можно получить при помощи Android SDK Manager. В Android Studio выполните команду Tools ▸ Android ▸ SDK Manager. (Меню Tools отображается только при наличии открытого проекта. Если вы еще не создали проект, SDK Manager также можно вызвать с экрана Android Setup Wizard; в разделе Quick Start выберите вариант Configure ▸ SDK Manager, как показано на рис. 1.)

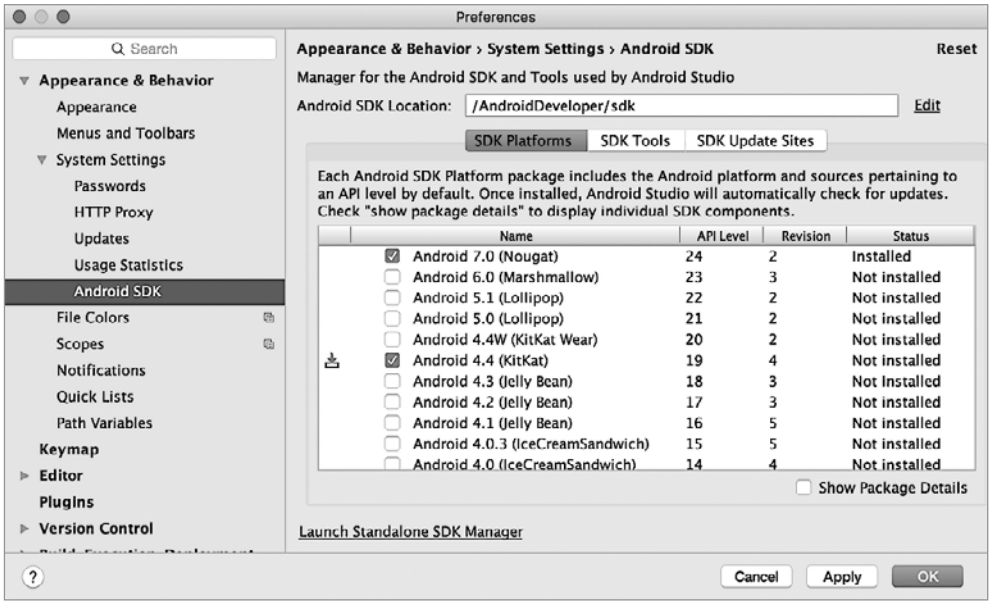


Рис. 1. Android SDK Manager

Выберите и установите каждую версию Android, которая будет использоваться при тестировании. Учтите, что загрузка этих компонентов может потребовать времени.

Android SDK Manager также используется для загрузки новейших выпусков Android — например, новой платформы или обновленных версий инструментов.

Физическое устройство

Эмулятор удобен для тестирования приложений. Тем не менее он не заменит реальное устройство на базе Android для оценки быстродействия. Если у вас имеется физическое устройство, мы рекомендуем время от времени использовать его в ходе работы над книгой.

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу электронной почты *comp@piter.com* (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства *http://www.piter.com* вы найдете подробную информацию о наших книгах.

1 Первое приложение Android

В первой главе даются новые концепции и составляющие, необходимые для построения приложений Android. Не беспокойтесь, если к концу главы что-то останется непонятным, — это нормально. Мы еще вернемся к этим концепциям в последующих главах и рассмотрим их более подробно.

Приложение, которое мы построим, называется GeoQuiz. Оно проверяет, насколько хорошо пользователь знает географию. Пользователь отвечает на вопрос, нажимая кнопку True или False, а GeoQuiz мгновенно сообщает ему результат.

На рис. 1.1 показан результат нажатия кнопки True.

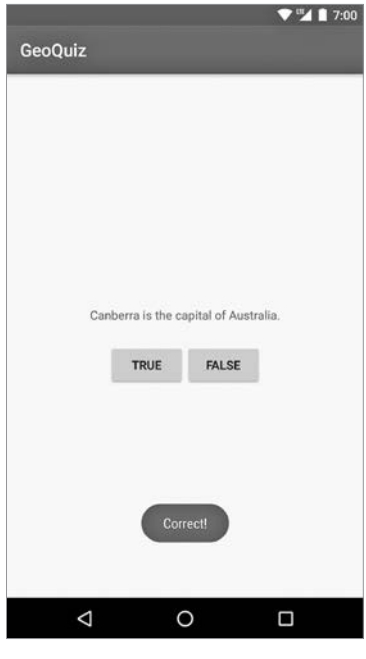


Рис. 1.1. А вы знаете географию страны антиподов?

Основы построения приложения

Приложение GeoQuiz состоит из *активности* (activity) и *макета* (layout):

- *Активность* представлена экземпляром `Activity` — классом из Android SDK. Она отвечает за взаимодействие пользователя с информацией на экране.

  Чтобы реализовать функциональность, необходимую приложению, разработчик пишет субклассы `Activity`. В простом приложении бывает достаточно одного субкласса; в сложном приложении их может потребоваться несколько.

  GeoQuiz — простое приложение, поэтому в нем используется всего один субкласс `Activity` с именем `QuizActivity`. Класс `QuizActivity` управляет пользовательским интерфейсом, изображенным на рис. 1.1.

- *Макет* определяет набор объектов пользовательского интерфейса и их расположение на экране. Макет формируется из определений, написанных на языке XML. Каждое определение используется для создания объекта, выводимого на экране (например, кнопки или текста).

  Приложение GeoQuiz включает файл макета с именем activity_quiz.xml. Разметка XML в этом файле определяет пользовательский интерфейс, изображенный на рис. 1.1.

Отношения между `QuizActivity` и activity_quiz.xml изображены на рис. 1.2.

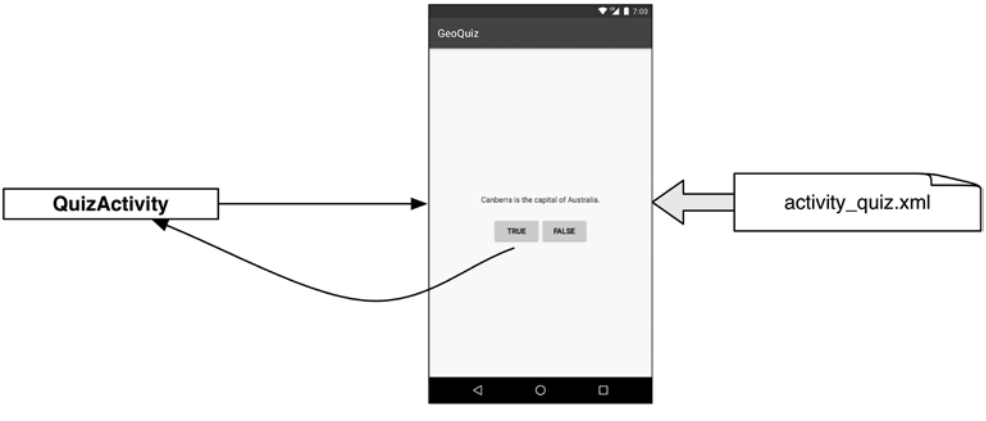


Рис. 1.2. QuizActivity управляет интерфейсом, определяемым в файле activity_quiz.xml

Учитывая все сказанное, давайте построим приложение.

Создание проекта Android

Работа начинается с создания *проекта Android*. Проект Android содержит файлы, из которых состоит приложение. Чтобы создать новый проект, откройте Android Studio.

Если Android Studio запускается на вашем компьютере впервые, на экране появляется диалоговое окно с приветствием (рис. 1.3).

Рис. 1.3. Добро пожаловать в Android Studio

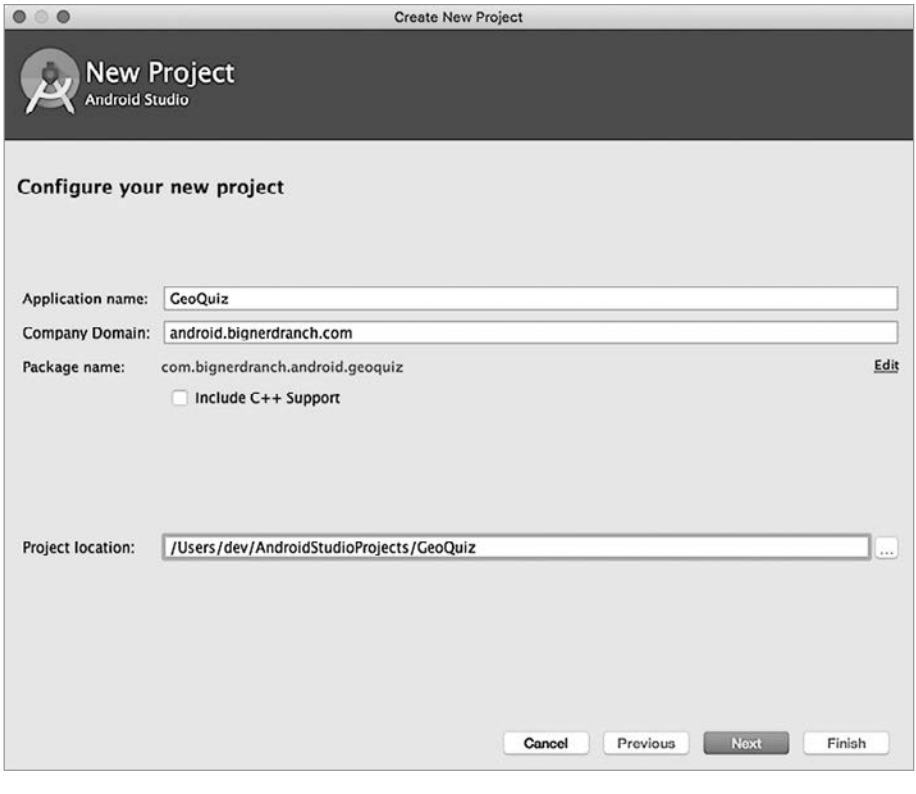


Рис. 1.4. Создание нового проекта

Выберите в диалоговом окне команду Start a new Android Studio project. Если диалоговое окно не отображается при запуске, значит, ранее вы уже создавали другие проекты. В таком случае выполните команду File ▸ New Project....

На экране появится мастер создания проекта (рис. 1.4.) На первом экране мастера введите имя приложения GeoQuiz (см. рис. 1.4). В поле Company Domain введите строку *android.bignerdranch.com*; сгенерированное имя пакета (Package Name) при этом автоматически поменяется на com.bignerdranch.android.geoquiz. В поле Project location введите любую папку своей файловой системы на свое усмотрение.

Обратите внимание: в имени пакета используется схема «обратного DNS», согласно которой доменное имя вашей организации записывается в обратном порядке с присоединением суффиксов дополнительных идентификаторов. Эта схема обеспечивает уникальность имен пакетов и позволяет различать приложения на устройстве и в Google Play.

Щелкните на кнопке Next. На следующем экране можно ввести дополнительную информацию об устройствах, которые вы намерены поддерживать. Приложение GeoQuiz будет поддерживать только телефоны, поэтому установите только флажок Phone and Tablet. Выберите в списке минимальную версию SDK API 19: Android 4.4 (KitKat) (рис. 1.5). Разные версии Android более подробно рассматриваются в главе 6.

Рис. 1.5. Определение поддерживаемых устройств

Щелкните на кнопке Next.

Следующим этапом вам будет предложено выбрать шаблон первого экрана GeoQuiz (рис. 1.6). Выберите простейший шаблон: пустую активность (Empty Activity) и щелкните на кнопке Next.

(Инструментарий Android регулярно обновляется, поэтому окно мастера на вашем компьютере может несколько отличаться от моего. Обычно это не создает проблем; принимаемые решения остаются практически неизменяемыми. Если ваше окно не имеет ничего общего с изображенным, значит, инструментарий изменился более радикально. Без паники. Загляните на форум книги *forums.bignerdranch.com*, и мы поможем вам найти обновленную версию.)

В последнем диалоговом окне мастера введите имя субкласса активности `QuizActivity` (рис. 1.7). Обратите внимание на суффикс `Activity` в имени класса. Его присутствие не обязательно, но это очень полезное соглашение, которое стоит соблюдать.

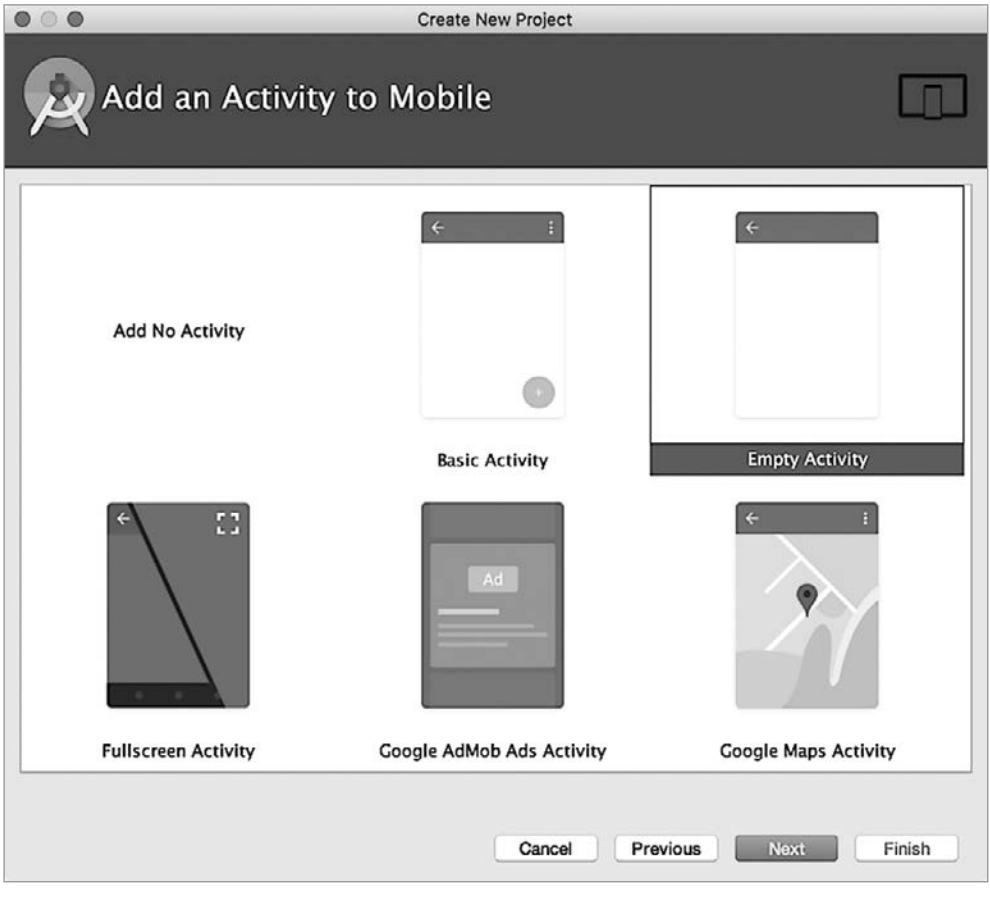


Рис. 1.6. Выбор типа активности

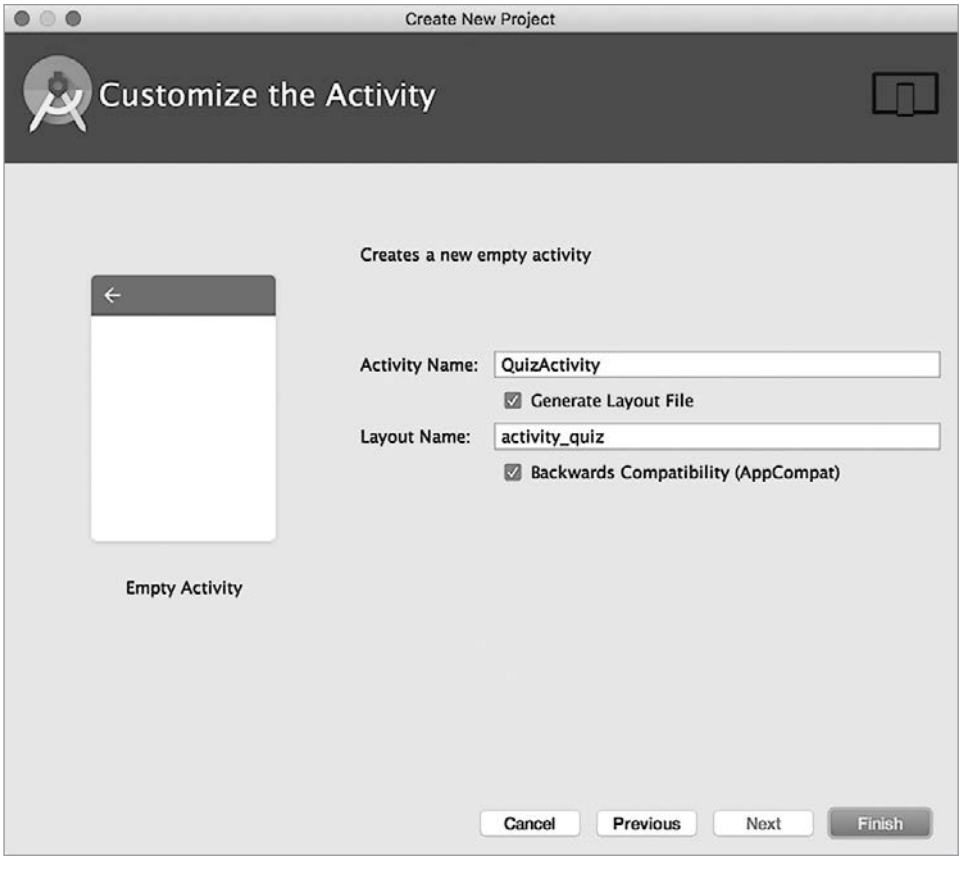


Рис. 1.7. Настройка новой активности

Оставьте флажок Generate Layout File установленным. Имя макета автоматически заменяется на `activity_quiz` в соответствии с переименованием активности. Имя макета записывается в порядке, обратном имени активности; в нем используются символы нижнего регистра, а слова разделяются символами подчеркивания. Эту схему формирования имен рекомендуется применять как для макетов, так и для других ресурсов, о которых вы узнаете из дальнейших глав.

Если в вашей версии Android Studio на экране присутствуют другие элементы управления, оставьте их в исходном состоянии. Щелкните на кнопке Finish. Android Studio создает и открывает новый проект.

Навигация в Android Studio

Среда Android Studio открывает ваш проект в окне, изображенном на рис. 1.8.

Различные части окна проекта называются *панелями*.

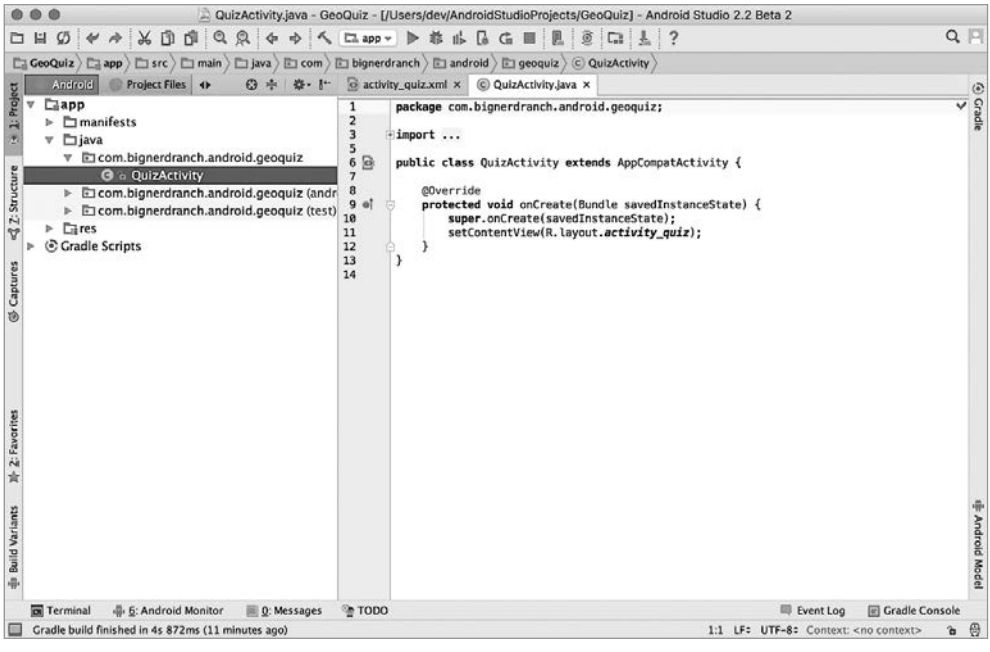


Рис. 1.8. Окно нового проекта

Слева располагается *окно инструментов Project*. В нем выполняются операции с файлами, относящимися к вашему проекту.

В середине находится панель *редактора*. Чтобы вам было проще приступить к работе, Android Studio открывает в редакторе файл activity_quiz.xml.

Видимостью различных панелей можно управлять, щелкая на их именах на полосе кнопок инструментов у левого, правого или нижнего края экрана. Также для многих панелей определены специальные комбинации клавиш. Если полосы с кнопками не отображаются, щелкните на серой квадратной кнопке в левом нижнем углу главного окна или выполните команду View ▸ Tool Buttons.

Построение макета пользовательского интерфейса

Откройте файл /res/layout/activity_quiz.xml. (Если в редакторе выводится изображение, щелкните на вкладке Text в нижней части панели, чтобы вернуться к разметке XML.)

В настоящее время файл activity_quiz.xml определяет разметку для активности по умолчанию. Такая разметка часто изменяется, но XML будет выглядеть примерно так, как показано в листинге 1.1.

Листинг 1.1. Разметка активности по умолчанию (activity_quiz.xml)

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_quiz"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context="com.bignerdranch.android.geoquiz.QuizActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"/>
</RelativeLayout>
```

Макет активности по умолчанию определяет два *виджета* (widgets): `RelativeLayout` и `TextView`.

Виджеты — это структурные элементы, из которых составляется пользовательский интерфейс. Виджет может выводить текст или графику, взаимодействовать с пользователем или размещать другие виджеты на экране. Кнопки, текстовые поля, флажки — все это разновидности виджетов.

Android SDK включает множество виджетов, которые можно настраивать для получения нужного оформления и поведения. Каждый виджет является экземпляром класса `View` или одного из его субклассов (например, `TextView` или `Button`).

На рис. 1.9 показано, как выглядят на экране виджеты `RelativeLayout` и `TextView`, определенные в листинге 1.1.

Впрочем, это не те виджеты, которые нам нужны. В интерфейсе `QuizActivity` задействованы пять виджетов:

- вертикальный виджет `LinearLayout`;
- `TextView`;
- горизонтальный виджет `LinearLayout`;
- две кнопки `Button`.

На рис. 1.10 показано, как из этих виджетов образуется интерфейс QuizActivity.

Теперь нужно определить эти виджеты в файле activity_quiz.xml.

В окне инструментов Project найдите каталог app/res/layout, раскройте его содержимое и откройте файл activity_quiz.xml. Внесите изменения, представленные в листинге 1.2. Разметка XML, которую нужно удалить, выделена перечеркиванием, а добавляемая разметка XML — жирным шрифтом. Эти обозначения будут использоваться во всей книге.

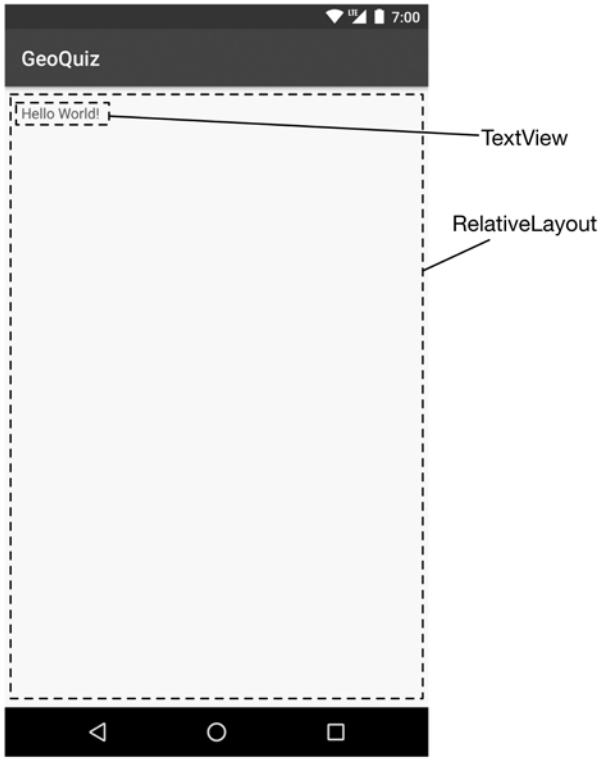


Рис. 1.9. Виджеты по умолчанию на экране

Не беспокойтесь, если смысл вводимой разметки остается непонятным; скоро вы узнаете, как она работает. Но будьте внимательны: разметка макета не проверяется, и опечатки рано или поздно вызовут проблемы.

В трех строках, начинающихся с `android:text`, будут обнаружены ошибки. Пока не обращайте внимания, мы их скоро исправим.

Листинг 1.2. Определение виджетов в XML (activity_quiz.xml)

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".QuizActivity">

    <TextView
       android:text="@string/hello_world"
```

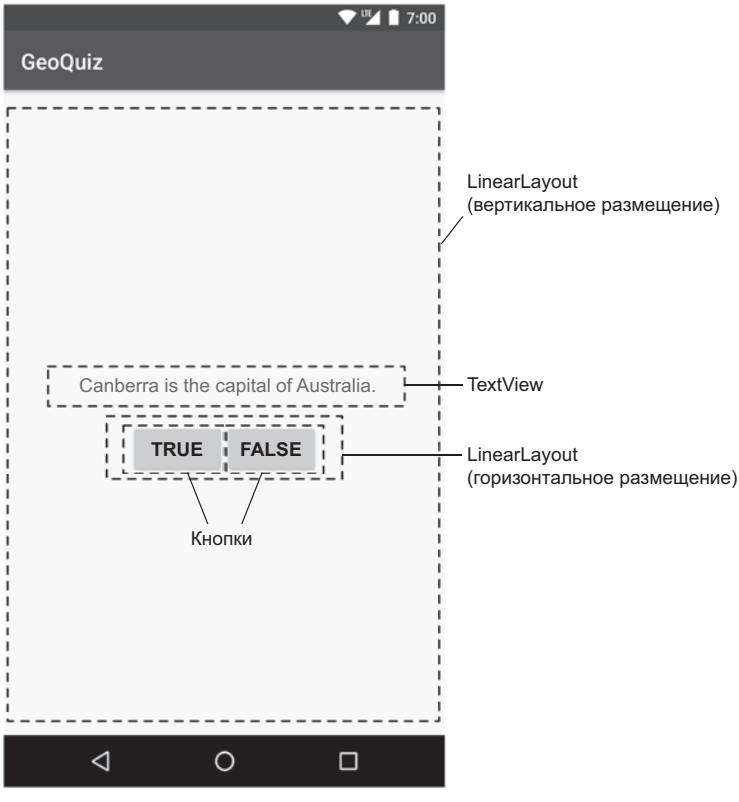


Рис. 1.10. Запланированное расположение виджетов на экране

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</RelativeLayout>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:gravity="center"
  android:orientation="vertical" >

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="24dp"
    android:text="@string/question_text" />

  <LinearLayout
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal" >

    <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/true_button" />

    <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/false_button" />

  </LinearLayout>

</LinearLayout>
```

Сравните XML с пользовательским интерфейсом, изображенным на рис. 1.10. Каждому виджету в разметке соответствует элемент XML. Имя элемента определяет тип виджета.

Каждый элемент обладает набором *атрибутов* XML. Атрибуты можно рассматривать как инструкции по настройке виджетов.

Чтобы лучше понять, как работают элементы и атрибуты, полезно взглянуть на разметку с точки зрения иерархии.

Иерархия представлений

Виджеты входят в иерархию объектов `View`, называемую *иерархией представлений*. На рис. 1.11 изображена иерархия виджетов для разметки XML из листинга 1.2.

Корневым элементом иерархии представлений в этом макете является элемент `LinearLayout`. В нем должно быть указано пространство имен XML ресурсов Android *http://schemas.android.com/apk/res/android*.

`LinearLayout` наследует от субкласса `View` с именем `ViewGroup`. Виджет `ViewGroup` предназначен для хранения и размещения других виджетов. `LinearLayout` используется в тех случаях, когда вы хотите выстроить виджеты в один столбец или строку. Другие субклассы `ViewGroup` — `FrameLayout`, `TableLayout` и `RelativeLayout`.

Если виджет содержится в *ViewGroup*, он называется *потомком* (child) `ViewGroup`. Корневой элемент `LinearLayout` имеет двух потомков: `TextView` и другой элемент `LinearLayout`. У `LinearLayout` имеется два собственных потомка `Button`.

Атрибуты виджетов

Рассмотрим некоторые атрибуты, используемые для настройки виджетов.

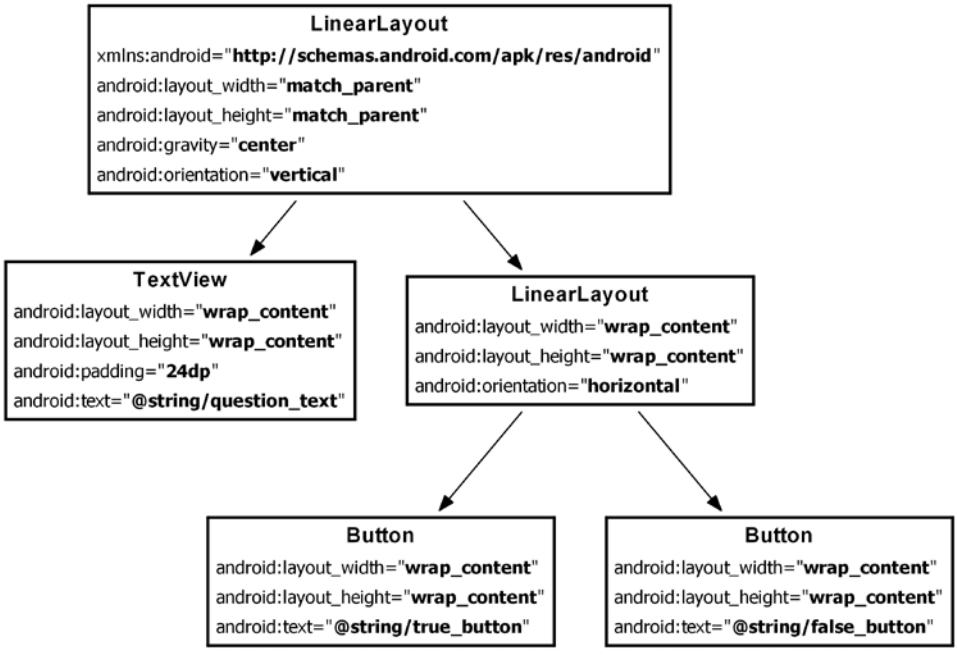


Рис. 1.11. Иерархия виджетов и атрибутов

android:layout_width и android:layout_height

Атрибуты `android:layout_width` и `android:layout_height`, определяющие ширину и высоту, необходимы практически для всех разновидностей виджетов. Как правило, им задается значение `match_parent` или `wrap_content`:

- `match_parent` — размеры представления определяются размерами родителя;
- `wrap_content` — размеры представления определяются размерами содержимого.

(Иногда в разметке встречается значение `fill_parent`. Это устаревшее значение эквивалентно `match_parent`.)

В корневом элементе `LinearLayout` атрибуты ширины и высоты равны `match_parent`. Элемент `LinearLayout` является корневым, но у него все равно есть родитель — представление Android для размещения иерархии представлений вашего приложения.

У других виджетов макета ширине и высоте задается значение `wrap_content`. На рис. 1.10 показано, как в этом случае определяются их размеры.

Виджет `TextView` чуть больше содержащегося в нем текста из-за атрибута `android:padding="24dp"`. Этот атрибут приказывает виджету добавить заданный отступ вокруг содержимого при определении размера, чтобы текст вопроса не соприкасался с кнопкой. (Интересуетесь, что это за единицы — `dp`? Это пикселы, не зависящие от плотности (density-independent pixels), о которых будет рассказано в главе 9.)

android:orientation

Атрибут `android:orientation` двух виджетов `LinearLayout` определяет, как будут выстраиваться потомки — по вертикали или горизонтали. Корневой элемент `LinearLayout` имеет вертикальную ориентацию; у его потомка `LinearLayout` горизонтальная ориентация.

Порядок определения потомков устанавливает порядок их отображения на экране. В вертикальном элементе `LinearLayout` потомок, определенный первым, располагается выше остальных. В горизонтальном элементе `LinearLayout` первый потомок является крайним левым. (Если только на устройстве не используется язык с письменностью справа налево, например арабский или иврит; в таком случае первый потомок будет находиться в крайней правой позиции.)

android:text

Виджеты `TextView` и `Button` содержат атрибуты `android:text`. Этот атрибут сообщает виджету, какой текст должен в нем отображаться.

Обратите внимание: значения атрибутов представляют собой не строковые литералы, а ссылки на *строковые ресурсы*.

Строковый ресурс — строка, находящаяся в отдельном файле XML, который называется *строковым файлом*. Виджету можно назначить фиксированную строку (например, `android:text="True"`), но так делать не стоит. Лучше размещать строки в отдельном файле, а затем ссылаться на них, так как использование строковых ресурсов упрощает локализацию.

Строковые ресурсы, на которые мы ссылаемся в activity_quiz.xml, еще не существуют. Давайте исправим этот недостаток.

Создание строковых ресурсов

Каждый проект включает строковый файл по умолчанию с именем strings.xml.

Откройте файл res/values/strings.xml. В шаблон уже включен один строковый ресурс. Добавьте три новые строки для вашего макета.

Листинг 1.3. Добавление строковых ресурсов (strings.xml)

```
<resources>
    <string name="app_name">GeoQuiz</string>
    <string name="question_text">Canberra is the capital of Australia.</string>
    <string name="true_button">True</string>
    <string name="false_button">False</string>
</resources>
```

(В зависимости от версии Android Studio файл может содержать другие строки. Не удаляйте их — это может породить каскадные ошибки в других файлах.)

Теперь по ссылке `@string/false_button` в любом файле XML проекта GeoQuiz вы будете получать строковый литерал `"False"` на стадии выполнения.

Если в файле activity_quiz.xml оставались ошибки, связанные с отсутствием строковых ресурсов, они должны исчезнуть. (Если ошибки остались, проверьте оба файла — возможно, где-то допущена опечатка.)

Строковый файл по умолчанию называется strings.xml, но ему можно присвоить любое имя на ваш выбор. Проект может содержать несколько строковых файлов. Если файл находится в каталоге res/values/, содержит корневой элемент `resources` и дочерние элементы `string`, ваши строки будут найдены и правильно использованы приложением.

Предварительный просмотр макета

Макет готов, и его можно просмотреть в графическом конструкторе (рис. 1.12).

Рис. 1.12. Предварительный просмотр в графическом конструкторе макетов (activity_quiz.xml)

Прежде всего убедитесь в том, что файлы сохранены и не содержат ошибок. Затем вернитесь к файлу activity_quiz.xml и откройте панель Preview (если она еще не открыта) при помощи вкладки в правой части редактора.

От разметки XML к объектам View

Как элементы XML в файле activity_quiz.xml превращаются в объекты `View`? Ответ на этот вопрос начинается с класса `QuizActivity`.

При создании проекта GeoQuiz был автоматически создан субкласс `Activity` с именем `QuizActivity`. Файл класса `QuizActivity` находится в каталоге app/java (в котором хранится Java-код вашего проекта).

В окне инструментов Project откройте каталог app/java, а затем содержимое пакета com.bignerdranch.android.geoquiz. Откройте файл QuizActivity.java и просмотрите его содержимое (листинг 1.4).

Листинг 1.4. Файл класса QuizActivity по умолчанию (QuizActivity.java)

```
package com.bignerdranch.android.geoquiz;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class QuizActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_quiz);
    }
}
```

(Интересуетесь, что такое `AppCompatActivity`? Это субкласс, наследующий от класса Android `Activity` и обеспечивающий поддержку старых версий Android. Более подробная информация об `AppCompatActivity` приведена в главе 13.)

(Если вы не видите все директивы `import`, щелкните на знаке + слева от первой директивы `import`, чтобы раскрыть список.)

Файл содержит один метод `Activity`: `onCreate(Bundle)`.

(Если ваш файл содержит методы `onCreateOptionsMenu(Menu)` и `onOptionsItemSelected(MenuItem)`, пока не обращайте на них внимания. Мы вернемся к теме меню в главе 13.)

Метод `onCreate(Bundle)` вызывается при создании экземпляра субкласса активности. Такому классу нужен пользовательский интерфейс, которым он будет управлять. Чтобы предоставить классу активности его пользовательский интерфейс, следует вызвать следующий метод `Activity`:

```
public void setContentView(int layoutResID)
```

Этот метод *заполняет* (inflates) макет и выводит его на экран. При заполнении макета создаются экземпляры всех виджетов в файле макета с параметрами, определяемыми его атрибутами. Чтобы указать, какой именно макет следует заполнить, вы передаете идентификатор ресурса макета.

Ресурсы и идентификаторы ресурсов

Макет представляет собой *ресурс*. Ресурсом называется часть приложения, которая не является кодом, — графические файлы, аудиофайлы, файлы XML и т. д.

Ресурсы проекта находятся в подкаталоге каталога app/res. В окне инструментов Project видно, что файл activity_quiz.xml находится в каталоге res/layout/. Строковый файл, содержащий строковые ресурсы, находится в res/values/.

Для обращения к ресурсу в коде используется его *идентификатор ресурса*. Нашему макету назначен идентификатор ресурса `R.layout.activity_quiz`.

Чтобы просмотреть текущие идентификаторы ресурсов проекта GeoQuiz, необходимо сначала изменить режим представления проекта. По умолчанию Android Studio использует режим представления Android (рис. 1.13). В этом режиме истинная структура каталогов проекта Android скрывается, чтобы вы могли сосредоточиться на тех файлах и папках, которые чаще всего нужны программисту.

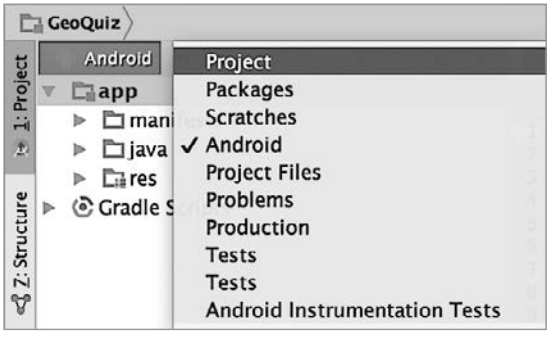


Рис. 1.13. Изменение режима представления проекта

Найдите раскрывающийся список в верхней части окна инструментов Project и выберите вместо режима Android режим Project. В этом режиме файлы и папки проекта представлены в своем фактическом состоянии.

Чтобы просмотреть ресурсы приложения GeoQuiz, раскройте содержимое каталога app/build/generated/source/r/debug. В этом каталоге найдите имя пакета своего проекта и откройте файл R.java из этого пакета. Поскольку этот файл генерируется процессом сборки Android, вам не следует его изменять, о чем деликатно предупреждает надпись в начале файла.

Возможно, внесение изменений в ресурсы не приведет к моментальному обновлению этого файла. Android Studio поддерживает скрытый файл R.java, который используется при построении вашего кода. Файл R.java из листинга 1.5 был сгенери-

рован для вашего приложения перед его установкой на устройстве или эмуляторе. Вы увидите, что файл будет обновлен при запуске приложения.

Листинг 1.5. Текущие идентификаторы ресурсов GeoQuiz

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 *
 ...
 */
package com.bignerdranch.android.geoquiz;

public final class R {
    public static final class anim {
        ...
    }
    ...

    public static final class id {
        ...
    }
    public static final class layout {
        ...
        public static final int activity_quiz=0x7f030017;
    }
    public static final class mipmap {
        public static final int ic_launcher=0x7f030000;
    }
    public static final class string {
        ...
        public static final int app_name=0x7f0a0010;
        public static final int false_button=0x7f0a0012;
        public static final int question_text=0x7f0a0014;
        public static final int true_button=0x7f0a0015;
    }
}
```

Файл R.java может быть довольно большим; в листинге 1.5 значительная часть его содержимого не показана.

Теперь понятно, откуда взялось имя `R.layout.activity_quiz` — это целочисленная константа с именем `activity_quiz` из внутреннего класса `layout` класса `R`.

Строкам также назначаются идентификаторы ресурсов. Мы еще не ссылались на строки в коде, но эти ссылки обычно выглядят так:

```
setTitle(R.string.app_name);
```

Android генерирует идентификатор ресурса для всего макета и для каждой строки, но не для отдельных виджетов из файла activity_quiz.xml. Не каждому виджету нужен идентификатор ресурса. В этой главе мы будем взаимодействовать лишь с двумя кнопками, поэтому идентификаторы ресурсов нужны только им.

Прежде чем генерировать идентификаторы ресурсов, переключитесь обратно в режим представления Android. В книге мы будем работать в этом режиме, но ничто не мешает вам использовать режим Project, если вы этого хотите.

Чтобы сгенерировать идентификатор ресурса для виджета, включите в определение виджета атрибут `android:id`. В файле activity_quiz.xml добавьте атрибут `android:id` для каждой кнопки.

Листинг 1.6. Добавление идентификаторов кнопок (activity_quiz.xml)

```
<LinearLayout ... >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp"
        android:text="@string/question_text" />

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <Button
            android:id="@+id/true_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/true_button" />

        <Button
            android:id="@+id/false_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/false_button" />

    </LinearLayout>

</LinearLayout>
```

Обратите внимание: знак + присутствует в значениях `android:id`, но не в значениях `android:text`. Это связано с тем, что мы *создаем* идентификаторы, а на строки только *ссылаемся*.

Подключение виджетов к программе

Теперь, когда кнопкам назначены идентификаторы ресурсов, к ним можно обращаться в `QuizActivity`. Все начинается с добавления двух переменных.

Введите следующий код в QuizActivity.java (листинг 1.7). (Не используйте автозавершение; введите его самостоятельно.) После сохранения файла выводятся два сообщения об ошибках.

Листинг 1.7. Добавление полей (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {

    private Button mTrueButton;
    private Button mFalseButton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_quiz);
    }
}
```

Сейчас мы исправим ошибки, а пока обратите внимание на префикс `m` у имен двух полей (переменных экземпляров). Этот префикс соответствует схеме формирования имен Android, которая будет использоваться в этой книге.

Наведите указатель мыши на красные индикаторы ошибок. Они сообщают об одинаковой проблеме: «Не удается разрешить символическое имя `Button`» (Cannot resolve symbol 'Button').

Чтобы избавиться от ошибок, следует импортировать класс `android.widget.Button` в QuizActivity.java. Введите следующую директиву импортирования в начале файла:

```
import android.widget.Button;
```

А можно пойти по простому пути и поручить эту работу Android Studio. Нажмите Option+Return (или Alt+Enter), и волшебство IntelliJ придет вам на помощь. Новая директива `import` теперь появляется под другими директивами в начале файла. Этот прием часто бывает полезным, если ваш код работает не так, как положено. Экспериментируйте с ним почаще!

Ошибки должны исчезнуть. (Если они остались, поищите опечатки в коде и XML.)

Теперь вы можете подключить свои виджеты-кнопки. Процедура состоит из двух шагов:

- получение ссылок на заполненные объекты `View`;
- назначение для этих объектов слушателей, реагирующих на действия пользователя.

Получение ссылок на виджеты

В классе активности можно получить ссылку на заполненный виджет, для чего используется следующий метод `Activity`:

```
public View findViewById(int id)
```

Метод получает идентификатор ресурса виджета и возвращает объект `View`.

В файле QuizActivity.java (листинг 1.8) по идентификаторам ресурсов ваших кнопок можно получить заполненные объекты и присвоить их полям. Учтите, что возвращенный объект `View` перед присваиванием необходимо преобразовать в `Button`.

Листинг 1.8. Получение ссылок на виджеты (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {

    private Button mTrueButton;
    private Button mFalseButton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_quiz);

        mTrueButton = (Button) findViewById(R.id.true_button);
        mFalseButton = (Button) findViewById(R.id.false_button);
    }
}
```

Назначение слушателей

Приложения Android обычно *управляются событиями* (event-driven). В отличие от программ командной строки или сценариев, такие приложения запускаются и ожидают наступления некоторого события, например нажатия кнопки пользователем. (События также могут инициироваться ОС или другим приложением, но события, инициируемые пользователем, наиболее очевидны.)

Когда ваше приложение ожидает наступления конкретного события, мы говорим, что оно «прослушивает» данное событие. Объект, создаваемый для ответа на событие, называется *слушателем* (listener). Такой объект реализует *интерфейс слушателя* данного события.

Android SDK поставляется с интерфейсами слушателей для разных событий, поэтому вам не придется писать собственные реализации. В нашем случае прослушиваемым событием является «щелчок» на кнопке, поэтому слушатель должен реализовать интерфейс `View.OnClickListener`.

Начнем с кнопки TRUE. В файле QuizActivity.java включите следующий фрагмент кода в метод `onCreate(Bundle)` непосредственно после присваивания.

Листинг 1.9. Назначение слушателя для кнопки TRUE (QuizActivity.java)

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_quiz);

        mTrueButton = (Button)findViewById(R.id.true_button);
        mTrueButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Пока ничего не делает, но скоро будет!
            }
        });
```

```
        mFalseButton = (Button) findViewById(R.id.false_button);
    }
}
```

(Если вы получили ошибку «View cannot be resolved to a type», воспользуйтесь комбинацией Option+Return (Alt+Enter) для импортирования класса `View`.)

В листинге 1.9 назначается слушатель, информирующий о нажатии виджета `Button` с именем `mTrueButton`. Метод `setOnClickListener(OnClickListener)` получает в аргументе слушателя, а конкретнее — объект, реализующий `OnClickListener`.

Анонимные внутренние классы

Слушатель реализован в виде *анонимного внутреннего класса*. Возможно, синтаксис не очевиден; просто запомните: все, что заключено во внешнюю пару круглых скобок, передается `setOnClickListener(OnClickListener)`. В круглых скобках создается новый безымянный класс, вся реализация которого передается вызываемому методу:

```
mTrueButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Пока ничего не делает, но скоро будет!
    }
});
```

Все слушатели в этой книге будут реализованы в виде анонимных внутренних классов. В этом случае реализация методов слушателя находится непосредственно там, где вы хотите ее видеть, а мы избегаем затрат ресурсов на создание именованного класса, который будет использоваться только в одном месте.

Так как анонимный класс реализует `OnClickListener`, он должен реализовать единственный метод этого интерфейса — `onClick(View)`. Мы пока оставили реализацию `onClick(View)` пустой, и компилятор не протестует. Интерфейс слушателя требует, чтобы метод `onClick(View)` был реализован, но не устанавливает никаких правил относительно того, *как именно* он будет реализован.

(Если ваши знания в области анонимных внутренних классов, слушателей или интерфейсов оставляют желать лучшего, полистайте учебник по Java перед тем, как продолжать, или хотя бы держите справочник под рукой.)

Назначьте аналогичного слушателя для кнопки FALSE.

Листинг 1.10. Назначение слушателя для кнопки FALSE (QuizActivity.java)

```
mTrueButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Пока ничего не делает, но скоро будет!
    }
});
```

```
    mFalseButton = (Button) findViewById(R.id.false_button);
    mFalseButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // Пока ничего не делает, но скоро будет!
        }
    });
}
```

Уведомления

Пора заставить кнопки делать что-то полезное. В нашем приложении каждая кнопка будет выводить на экран временное *уведомление* (toast) — короткое сообщение, которое содержит какую-либо информацию для пользователя, но не требует ни ввода, ни действий. Наши уведомления будут сообщать пользователю, правильно ли он ответил на вопрос (рис. 1.14).

Для начала вернитесь к файлу strings.xml и добавьте строковые ресурсы, которые будут отображаться в уведомлении.

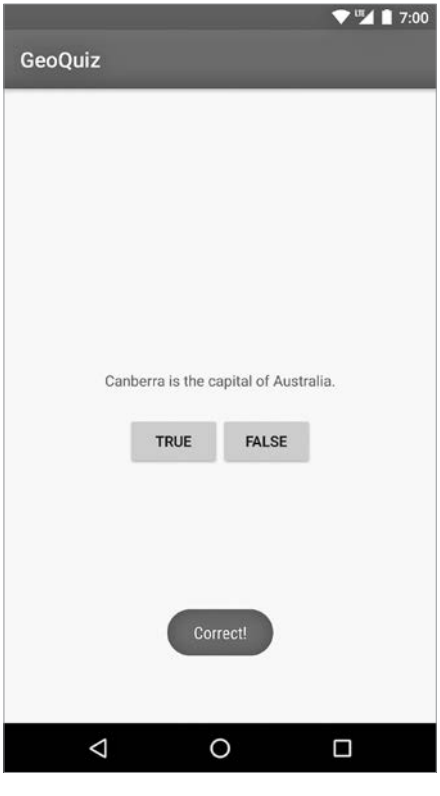


Рис. 1.14. Уведомление с информацией для пользователя

Листинг 1.11. Добавление строк уведомлений (strings.xml)

```
<resources>
    <string name="app_name">GeoQuiz</string>
    <string name="question_text">Canberra is the capital of Australia.</string>
    <string name="true_button">True</string>
    <string name="false_button">False</string>
    <string name="correct_toast">Correct!</string>
    <string name="incorrect_toast">Incorrect!</string>
</resources>
```

Уведомление создается вызовом следующего метода класса `Toast`:

```
public static Toast makeText(Context context, int resId, int duration)
```

Параметр `Context` обычно содержит экземпляр `Activity` (`Activity` является субклассом `Context`). Во втором параметре передается идентификатор ресурса строки, которая должна выводиться в уведомлении. Параметр `Context` необходим классу `Toast` для поиска и использования идентификатора ресурса строки. Третий параметр обычно содержит одну из двух констант `Toast`, определяющих продолжительность пребывания уведомления на экране.

После того как объект уведомления будет создан, вызовите `Toast.show()`, чтобы уведомление появилось на экране.

В классе `QuizActivity` вызов `makeText(...)` будет присутствовать в слушателе каждой кнопки (листинг 1.12). Вместо того чтобы вводить все вручную, попробуйте добавить эти вызовы с использованием функции автозавершения среды Android Studio.

Автозавершение

Автозавершение экономит много времени, так что с ним стоит познакомиться пораньше.

Начните вводить новый код из листинга 1.12. Когда вы доберетесь до точки после класса `Toast`, на экране появится подсказка со списком методов и констант класса `Toast`.

Чтобы выбрать одну из рекомендаций, используйте клавиши ↑ и ↓. (Если вы не собираетесь использовать автозавершение, просто продолжайте печатать. Без нажатия клавиши Tab, клавиши Return/Enter или щелчка на окне подсказки подстановка не выполняется.)

Выберите в списке рекомендаций метод `makeText(Context context, int resID, int duration)`. Механизм автозавершения добавляет полный вызов метода.

Задайте параметры метода `makeText` так, как показано в листинге 1.12.

Листинг 1.12. Создание уведомлений (QuizActivity.java)

```
mTrueButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
```

```
        Toast.makeText(QuizActivity.this,
                       R.string.correct_toast,
                       Toast.LENGTH_SHORT).show();
            // Пока ничего не делает, но скоро будет!
    }
});
mFalseButton = (Button) findViewById(R.id.false_button);
mFalseButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Toast.makeText(QuizActivity.this,
                       R.string.incorrect_toast,
                       Toast.LENGTH_SHORT).show();
            // Пока ничего не делает, но скоро будет!
    }
});
```

В вызове `makeText(…)` экземпляр `QuizActivity` передается в аргументе `Context`. Но, как и следовало ожидать, просто передать `this` нельзя. В этом месте кода мы определяем анонимный класс, где `this` обозначает `View.OnClickListener`.

Благодаря использованию автозавершения вам не придется ничего специально делать для импортирования класса `Toast`. Когда вы соглашаетесь на рекомендацию автозавершения, необходимые классы импортируются автоматически.

Сохраните внесенные изменения. Посмотрим, как работает новое приложение.

Выполнение в эмуляторе

Для запуска приложений Android необходимо устройство — физическое или *виртуальное*. Виртуальные устройства работают под управлением эмулятора Android, включенного в поставку средств разработчика.

Чтобы создать виртуальное устройство Android (AVD, Android Virtual Device), выполните команду Tools ▸ Android ▸ AVD Manager. Когда на экране появится окно AVD Manager, щелкните на кнопке +Create Virtual Device... в левой части этого окна.

Открывается диалоговое окно с многочисленными параметрами настройки виртуального устройства. Для первого AVD выберите эмуляцию устройства Nexus 5X, как показано на рис. 1.15. Щелкните на кнопке Next.

На следующем экране выберите образ системы, на основе которого будет работать эмулятор. Выберите эмулятор x86 Nougat и щелкните на кнопке Next (рис. 1.16). (Возможно, перед этим вам придется выполнить необходимые действия для загрузки компонентов эмулятора.)

Наконец, просмотрите и при необходимости измените свойства эмулятора (впрочем, свойства существующего эмулятора также можно изменить позднее). Пока присвойте своему эмулятору имя, по которому вы сможете узнать его в будущем, и щелкните на кнопке Finish (рис. 1.17).

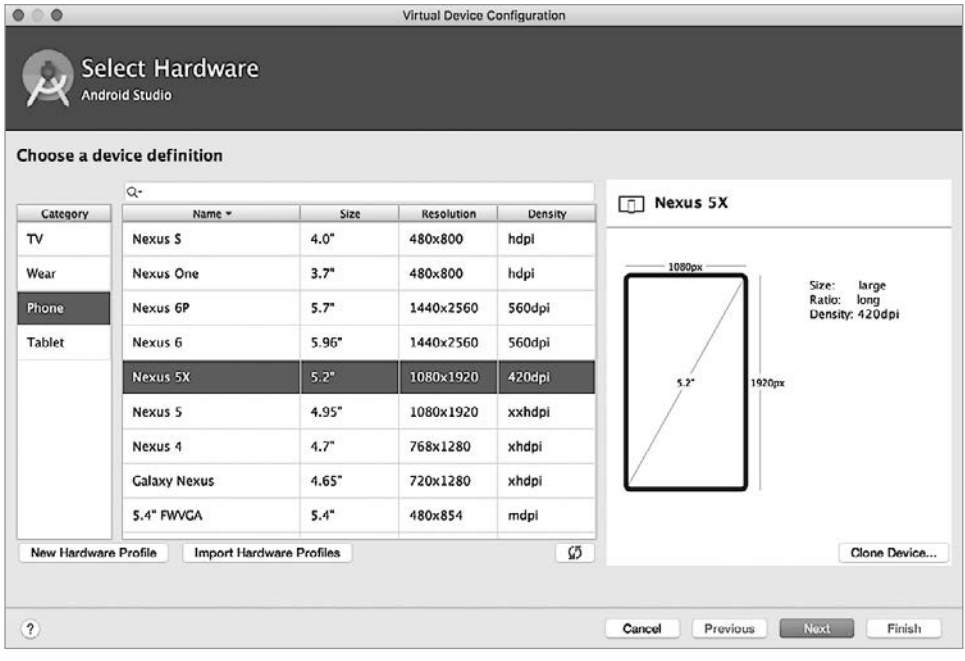


Рис. 1.15. Выбор виртуального устройства

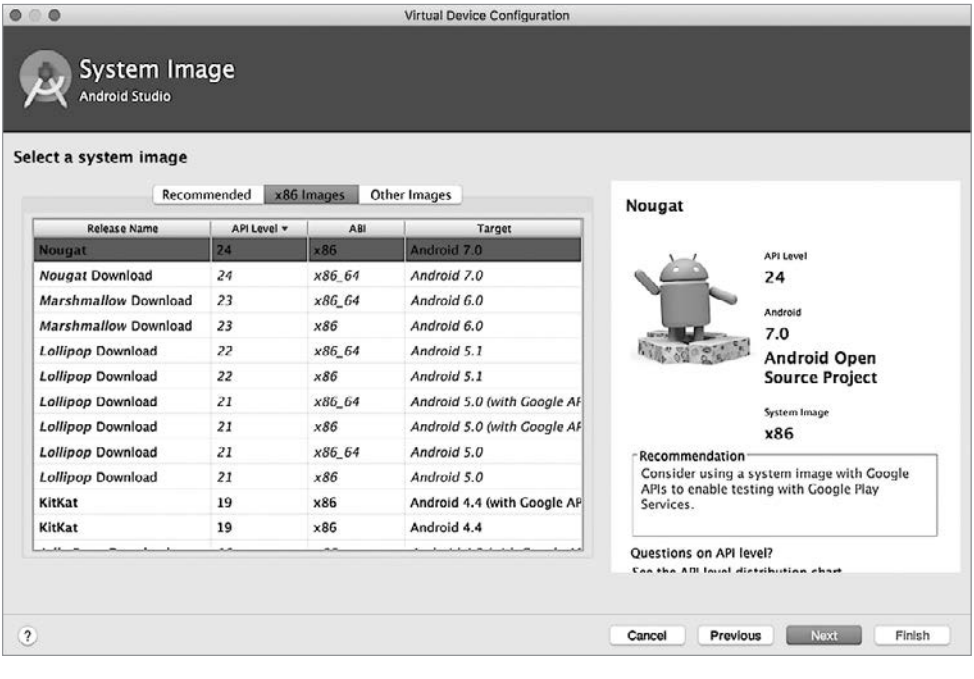


Рис. 1.16. Выбор образа системы

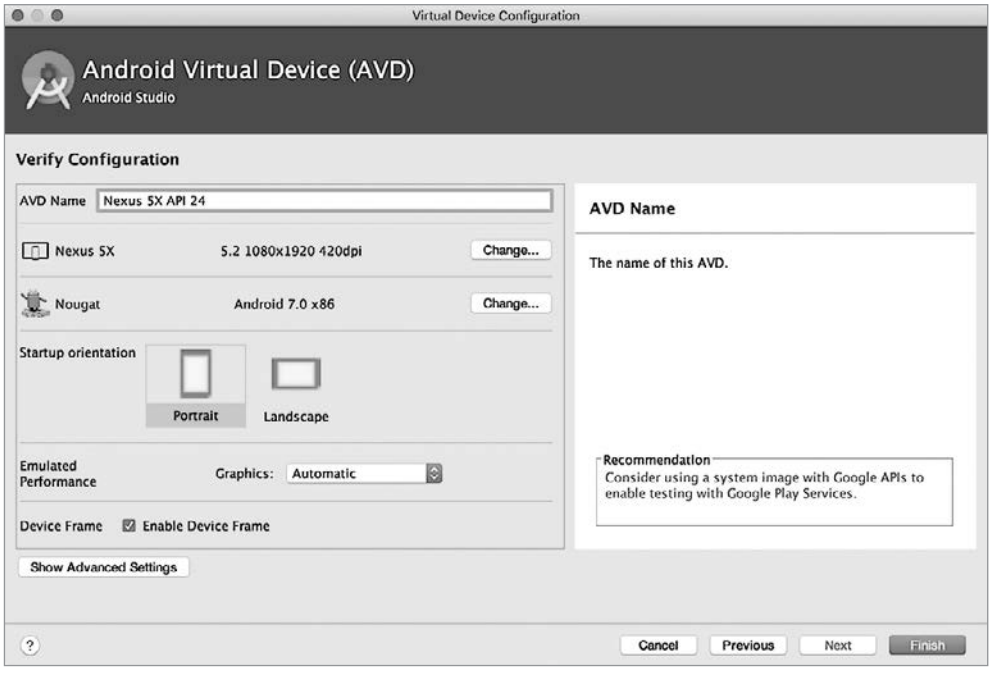


Рис. 1.17. Настройка свойств эмулятора

Когда виртуальное устройство будет создано, в нем можно запустить приложение GeoQuiz. На панели инструментов Android Studio щелкните на кнопке Run (зеленый символ «воспроизведение») или нажмите Ctrl+R. Android Studio находит созданное виртуальное устройство, запускает его, устанавливает на нем пакет приложения и запускает приложение.

Возможно, на запуск эмулятора потребуется некоторое время, но вскоре приложение GeoQuiz запустится на созданном вами виртуальном устройстве. Понажимайте кнопки и оцените уведомления.

Если при запуске GeoQuiz или нажатии кнопки происходит сбой, в нижней части представления LogCat панели Android DDMS выводится полезная информация. (Если представление LogCat не открылось автоматически при запуске GeoQuiz, его можно открыть при помощи кнопки Android Monitor в нижней части окна Android Studio.) Ищите исключения в журнале; они будут выделяться красным цветом (рис. 1.18).

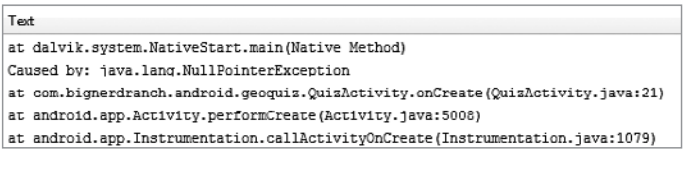


Рис. 1.18. Исключение NullPointerException в строке 21

Сравните свой код с кодом в книге и попытайтесь найти источник проблемы. Затем снова запустите приложение. (LogCat и процесс отладки более подробно рассматриваются в следующих двух главах.)

Не закрывайте эмулятор; не стоит ждать, пока он загружается при каждом запуске.

Вы можете остановить приложение кнопкой Back (треугольник, обращенный влево; в старых версиях Android использовалось изображение U-образной стрелки), а затем снова запустить приложение из Android Studio, чтобы протестировать изменения.

Эмулятор полезен, но тестирование на реальном устройстве дает более точные результаты. В главе 2 мы запустим приложение GeoQuiz на физическом устройстве, а также расширим набор вопросов по географии.

Для любознательных: процесс построения приложений Android

Вероятно, у вас уже накопились неотложные вопросы относительно того, как организован процесс построения приложений Android. Вы уже видели, что Android Studio строит проект автоматически в процессе его изменения, а не по команде. Во время построения инструментарий Android берет ваши ресурсы, код и файл AndroidManifest.xml (содержащий метаданные приложения) и преобразует их в файл .apk. Полученный файл подписывается отладочным ключом, что позволяет запускать его в эмуляторе. (Чтобы распространять файл .apk среди пользователей, необходимо подписать его ключом публикации. Дополнительную информацию об этом процессе можно найти в документации разработчика Android по адресу *developer.android.com/tools/publishing/preparing.html*.)

Как содержимое activity_quiz.xml преобразуется в объекты `View` в приложении? В процессе построения утилита aapt (Android Asset Packaging Tool) компилирует ресурсы файлов макетов в более компактный формат. Откомпилированные ресурсы упаковываются в файл .apk. Затем, когда метод `setContentView(…)` вызывается в методе `onCreate(…)` класса `QuizActivity`, `QuizActivity` использует класс `LayoutInflater` для создания экземпляров всех объектов `View`, определенных в файле макета (рис. 1.19).

(Также классы представлений можно создать на программном уровне в классе активности вместо определения их в XML. Однако отделение презентационной логики от логики приложения имеет свои преимущества, главное из которых — использование изменений конфигурации, встроенных в SDK; мы подробнее поговорим об этом в главе 3.)

За дополнительной информацией о том, как работают различные атрибуты XML и как происходит отображение представлений на экране, обращайтесь к главе 9.

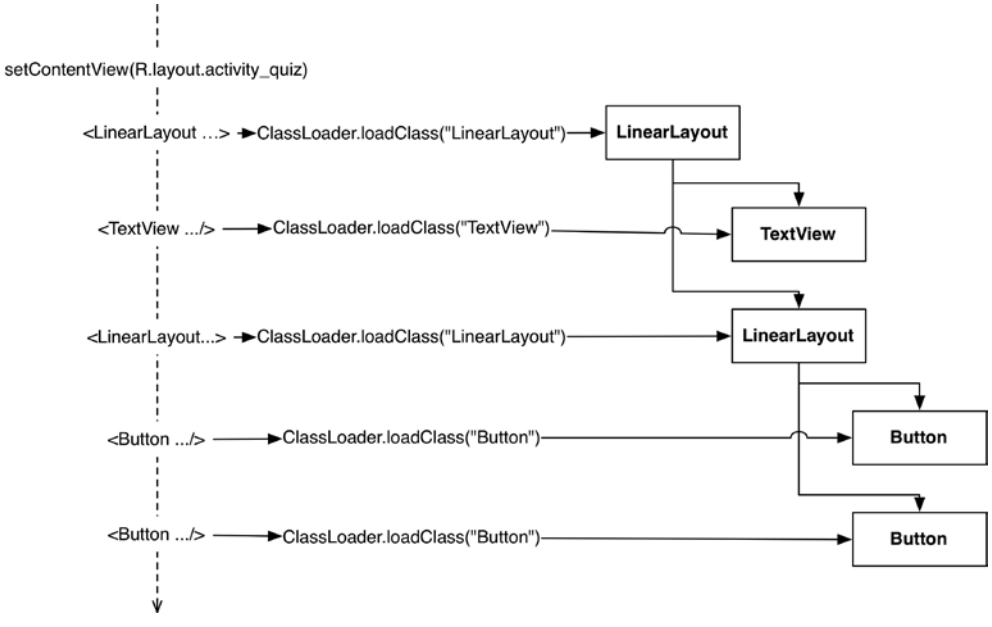


Рис. 1.19. Заполнение activity_quiz.xml

Средства построения программ Android

Все программы, которые мы запускали до настоящего времени, исполнялись из среды Android Studio. Этот процесс интегрирован в среду разработки — он вызывает стандартные средства построения программ Android (такие, как aapt), но сам процесс построения проходит под управлением Android Studio.

Может оказаться, что вам по каким-то своим причинам потребуется провести построение за пределами среды Android Studio. Для этого проще всего воспользоваться программой командной строки. В современной системе построения приложений Android используется программа Gradle.

(Вы и сами поймете, представляет ли эта информация для вас интерес. Если нет, просто бегло просмотрите ее, не беспокоясь о том, зачем это нужно и что делать, если что-то не работает. Рассмотрение всех тонкостей использования командной строки выходит за рамки книги.)

Чтобы использовать Gradle в режиме командной строки, перейдите в каталог своего проекта и выполните следующую команду:

```
$ ./gradlew tasks
```

В системе Windows команда выглядит немного иначе:

```
> gradlew.bat tasks
```

Команда выводит список задач, которые можно выполнить. Та задача, которая вам нужна, называется installDebug. Запустите ее следующей командой:

```
$ ./gradlew installDebug
```

Или в системе Windows:

```
> gradlew.bat installDebug
```

Команда устанавливает приложение на подключенное устройство, но не запускает его — вы должны сделать это вручную.

Упражнения

Упражнения в конце главы предназначены для самостоятельной работы. Некоторые из них просты и рассчитаны на повторение материала главы. Другие, более сложные упражнения потребуют логического мышления и навыков решения задач.

Трудно переоценить важность упражнений. Они закрепляют усвоенный материал, повышают уверенность в обретенных навыках и сокращают разрыв между изучением программирования Android и умением самостоятельно писать программы для Android.

Если у вас возникнут затруднения с каким-либо упражнением, сделайте перерыв, потом вернитесь и попробуйте еще раз. Если и эта попытка окажется безуспешной, обратитесь на форум книги *forums.bignerdranch.com*. Здесь вы сможете просмотреть вопросы и решения, полученные от других читателей, а также опубликовать свои вопросы и решения.

Чтобы избежать случайного повреждения текущего проекта, мы рекомендуем создать копию и практиковаться на ней.

В файловом менеджере компьютера перейдите в корневой каталог своего проекта. Скопируйте папку GeoQuiz и вставьте новую копию рядом с оригиналом (в macOS воспользуйтесь функцией дублирования). Переименуйте новую папку и присвойте ей имя GeoQuiz Challenge. В Android Studio выполните команду File ▸ Import Project.... В окне импорта перейдите к папке GeoQuiz Challenge и нажмите кнопку OK. Скопированный проект появляется в новом окне готовым к работе.

Упражнение: настройка уведомления

Измените уведомление так, чтобы оно отображалось в верхней, а не в нижней части экрана. Для изменения способа отображения уведомления воспользуйтесь методом `setGravity` класса `Toast`. Выберите режим `Gravity.TOP`. За дополнительной информацией обращайтесь к документации разработчика по адресу *developer.android.com/reference/android/widget/Toast.html#setGravity(int, int, int)*.

2 Android и модель MVC

В этой главе мы обновим приложение GeoQuiz и включим в него дополнительные вопросы (рис. 2.1).

Рис. 2.1. Больше вопросов!

Для этого в проект GeoQuiz будет добавлен класс `Question`. Экземпляр этого класса инкапсулирует один вопрос с ответом «да/нет».

Затем мы создадим массив объектов `Question`, с которым будет работать класс `QuizActivity`.

Создание нового класса

В окне инструментов Project щелкните правой кнопкой мыши на пакете com.bignerdranch.android.geoquiz и выберите команду New ▸ Java Class. Введите имя класса `Question` и щелкните на кнопке OK (рис. 2.2).

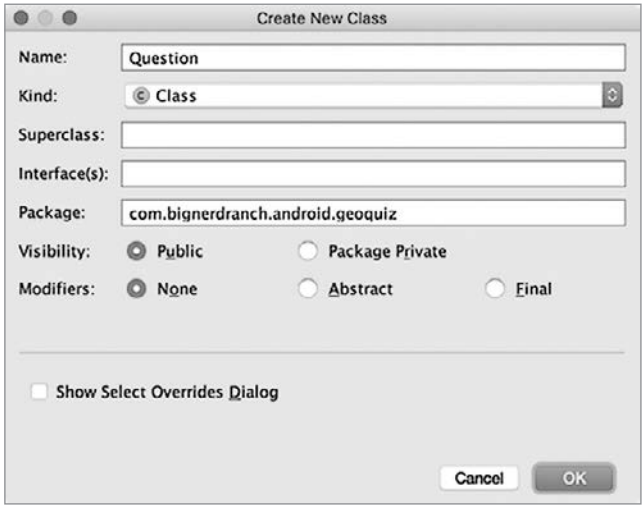


Рис. 2.2. Создание класса Question

Добавьте в файл Question.java два поля и конструктор:

Листинг 2.1. Добавление класса Question (Question.java)

```
public class Question {

    private int mTextResId;
    private boolean mAnswerTrue;

    public Question(int textResId, boolean answerTrue) {
        mTextResId = textResId;
        mAnswerTrue = answerTrue;
    }
}
```

Класс `Question` содержит два вида данных: текст вопроса и правильный ответ (да/нет).

Почему поле `mTextResID` объявлено с типом `int`, а не `String`? В нем будет храниться идентификатор ресурса (всегда `int`) строкового ресурса с текстом вопроса. Мы займемся созданием этих строковых ресурсов в следующем разделе.

Для переменных необходимо определить `get-` и `set`-методы. Вводить их самостоятельно не нужно — проще приказать Android Studio сгенерировать реализации.

Генерирование get- и set-методов

Прежде всего следует настроить Android Studio на распознавание префикса `m` в полях классов.

Откройте окно настроек Android Studio (меню Android Studio на Mac, команда File ▸ Settings в системе Windows). Откройте раздел Editor, затем раздел Code Style. Выберите категорию Java и перейдите на вкладку Code Generation.

В таблице Naming найдите строку Field (рис. 2.3) и в поле Naming Prefix введите префикс m для полей. Затем добавьте префикс s для статических полей в строке Static field. (В проекте GeoQuiz префикс `s` не используется, но он пригодится в будущих проектах.)

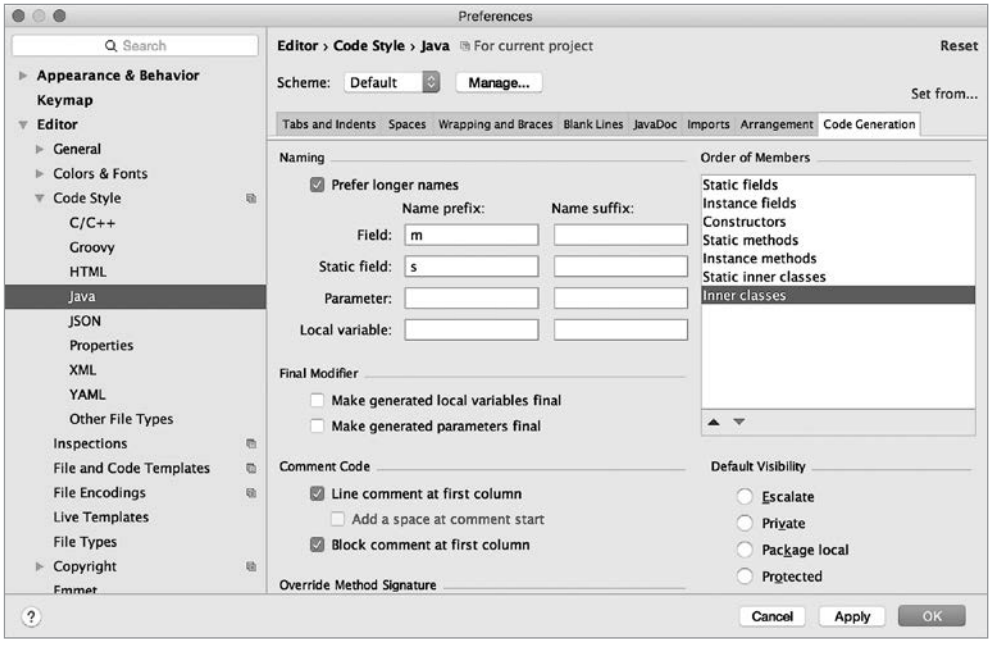


Рис. 2.3. Настройка стиля оформления кода Java

Щелкните на кнопке OK.

Зачем мы задавали эти префиксы? Если теперь приказать Android Studio сгенерировать `get`-метод для `mTextResID`, среда сгенерирует методы с именем `getTextResID()` вместо `getMTextResID()` и `isAnswerTrue()` вместо `isMAnswerTrue()`.

Вернитесь к файлу Question.java, щелкните правой кнопкой мыши после конструктора и выберите команду Generate... ▸ Getter And Setter. Выберите поля mTextResID и mAnswerTrue, затем щелкните на кнопке OK, чтобы сгенерировать `get`- и `set`-метод для каждой переменной. Результат представлен в листинге 2.2.

Листинг 2.2. Сгенерированные get- и set-методы (Question.java)

```
public class Question {

    private int mTextResId;
    private boolean mAnswerTrue;
    ...
    public int getTextResId() {
        return mTextResId;
    }

    public void setTextResId(int textResId) {
        mTextResId = textResId;
    }

    public boolean isAnswerTrue() {
        return mAnswerTrue;
    }

    public void setAnswerTrue(boolean answerTrue) {
        mAnswerTrue = answerTrue;
    }
}
```

Класс `Question` готов. Вскоре мы внесем изменения в `QuizActivity` для работы с `Question`, но для начала посмотрим, как фрагменты GeoQuiz будут работать вместе.

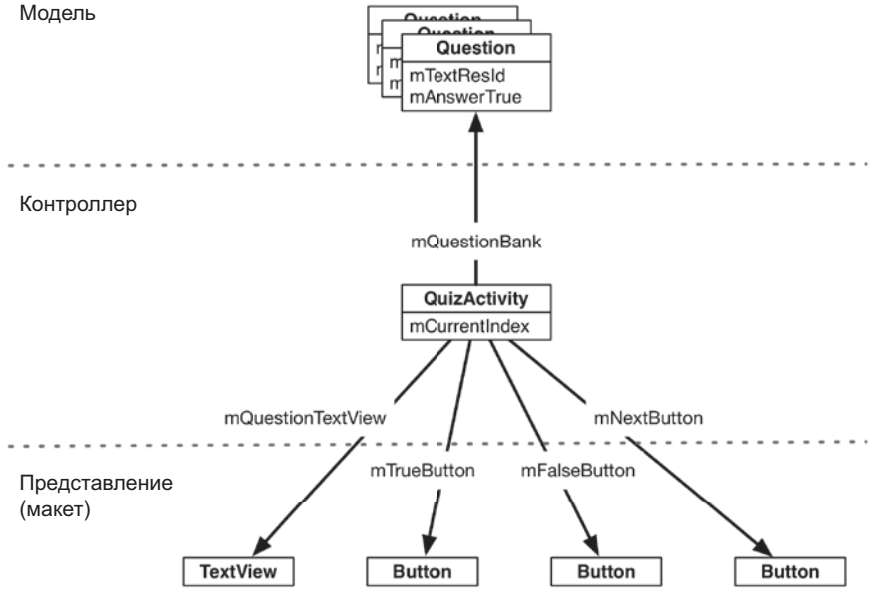


Рис. 2.4. Диаграмма объектов GeoQuiz

Класс `QuizActivity` должен создать массив объектов `Question`. В процессе работы он взаимодействует с `TextView` и тремя виджетами `Button` для вывода вопросов и предоставления обратной связи на ответы пользователя. Отношения между этими объектами изображены на рис. 2.4.

Архитектура «Модель-Представление-Контроллер» и Android

Вероятно, вы заметили, что объекты на рис. 2.4 разделены на три области: «Модель», «Контроллер» и «Представление». Приложения Android строятся на базе архитектуры, называемой «Модель-Представление-Контроллер», или сокращенно MVC (Model-View-Controller). Согласно канонам MVC, каждый объект приложения должен быть *объектом модели*, *объектом представления* или *объектом контроллера*.

- *Объект модели* содержит данные приложения и «бизнес-логику». Классы модели обычно проектируются для *моделирования* сущностей, с которыми имеет дело приложение, — пользователь, продукт в магазине, фотография на сервере, вопрос «да/нет» и т. д. Объекты модели ничего не знают о пользовательском интерфейсе; их единственной целью является хранение данных и управление ими.

  В приложениях Android классы моделей обычно создаются разработчиком для конкретной задачи. Все объекты модели в вашем приложении составляют его *уровень модели*.

  Уровень модели GeoQuiz состоит из класса `Question`.

- *Объекты представлений* умеют отображать себя на экране и реагировать на ввод пользователя, например касания. Простейшее правило: если вы видите что-то на экране, значит, это представление.

  Android предоставляет широкий набор настраиваемых классов представлений. Разработчик также может создавать собственные классы представлений. Объекты представления в приложении образуют *уровень представления*.

  В GeoQuiz уровень представления состоит из виджетов, заполненных по содержимому файла activity_quiz.xml.

- *Объекты контроллеров* связывают объекты представления и модели; они содержат «логику приложения». Контроллеры реагируют на различные события, инициируемые объектами представлений, и управляют потоками данных между объектами модели и уровнем представления.

  В Android контроллер обычно представляется субклассом `Activity`, `Fragment` или `Service`. (Фрагменты рассматриваются в главе 7, а службы — в главе 28.)

  Уровень контроллера GeoQuiz в настоящее время состоит только из класса `QuizActivity`.

На рис. 2.5 показана логика передачи управления между объектами в ответ на пользовательское событие, такое как нажатие кнопки. Обратите внимание: объек-

ты модели и представлений не взаимодействуют друг с другом напрямую; в любом взаимодействии участвуют «посредники»-контроллеры, получающие сообщения от одних объектов и передающие инструкции другим.

Рис. 2.5. Взаимодействия MVC при получении ввода от пользователя

Преимущества MVC

Приложение может обрастать функциональностью до тех пор, пока не станет слишком сложным для понимания. Разделение кода на классы упрощает проектирование и понимание приложения в целом; разработчик мыслит в контексте классов, а не отдельных переменных и методов.

Аналогичным образом разделение классов на уровни модели, представления и контроллера упрощает проектирование и понимание приложения; вы можете мыслить в контексте уровней, а не отдельных классов.

Приложение GeoQuiz нельзя назвать сложным, но преимущества разделения уровней видны и здесь. Вскоре мы обновим уровень представления GeoQuiz и добавим в него кнопку NEXT. При этом нам совершенно не нужно помнить о только что созданном классе `Question`.

MVC также упрощает повторное использование классов. Класс с ограниченными обязанностями лучше подходит для повторного использования, чем класс, который пытается заниматься всем сразу.

Например, класс модели `Question` ничего не знает о виджетах, используемых для вывода вопроса «да/нет». Это упрощает использование `Question` в приложении для разных целей. Например, если потребуется вывести полный список всех вопросов, вы можете использовать тот же объект, который используется для вывода всего одного вопроса.

Обновление уровня представления

После теоретического знакомства с MVC пора переходить к практике: обновим уровень представления GeoQuiz и включим в него кнопку NEXT.

В Android объекты уровня представления обычно заполняются на основе разметки XML в файле макета. Весь макет GeoQuiz определяется в файле activity_quiz.xml. В него следует внести изменения, представленные на рис. 2.6. (Для экономии места на рисунке не показаны атрибуты виджетов, оставшихся без изменений.)

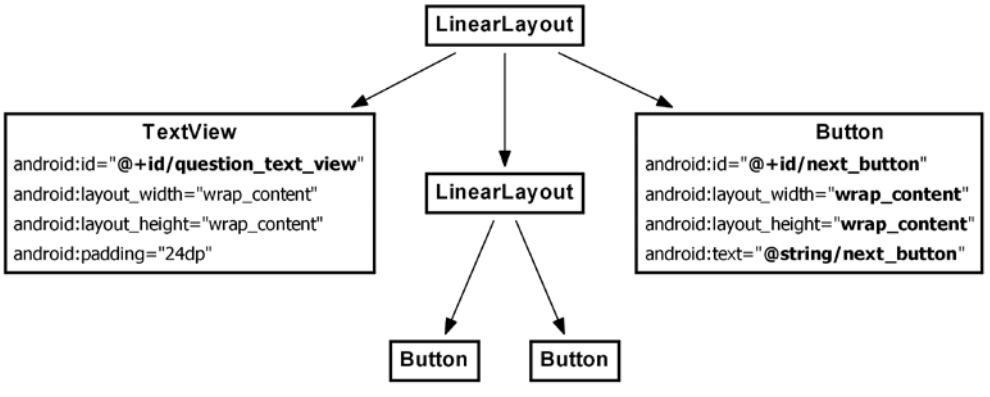


Рис. 2.6. Новая кнопка!

Итак, на уровне представления необходимо внести следующие изменения:

- Удалите атрибут `android:text` из `TextView`. Жестко запрограммированный текст вопроса не должен присутствовать в определении.
- Назначьте `TextView` атрибут `android:id`. Идентификатор ресурса необходим виджету для того, чтобы мы могли задать его текст в коде `QuizActivity`.
- Добавьте новый виджет `Button` как потомка корневого элемента `LinearLayout`.

Вернитесь к файлу activity_quiz.xml и выполните все перечисленные действия.

Листинг 2.3. Новая кнопка и изменения в TextView (activity_quiz.xml)

```
<LinearLayout ... >
    <TextView
        android:id="@+id/question_text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp"
        android:text="@string/question_text" />

    <LinearLayout ... >
        ...
    </LinearLayout>
```

```
    <Button
        android:id="@+id/next_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/next_button" />

</LinearLayout>
```

На экране появится знакомая ошибка с сообщением об отсутствующем строковом ресурсе.

Вернитесь к файлу res/values/strings.xml. Переименуйте `question_text` и добавьте строку для новой кнопки.

Листинг 2.4. Обновленные строки (strings.xml)

```
<string name="app_name">GeoQuiz</string>
<string name="question_text">Canberra is the capital of Australia.</string>
<string name="question_australia">Canberra is the capital of Australia.</string>
<string name="true_button">True</string>
<string name="false_button">False</string>
<string name="next_button">Next</string>
<string name="correct_toast">Correct!</string>
```

Пока файл strings.xml открыт, добавьте строки с вопросами по географии, которые будут предлагаться пользователю.

Листинг 2.5. Добавление строк вопросов (strings.xml)

```
<string name="question_australia">Canberra is the capital of Australia.</string>
<string name="question_oceans">The Pacific Ocean is larger than
  the Atlantic Ocean.</string>
<string name="question_mideast">The Suez Canal connects the Red Sea
  and the Indian Ocean.</string>
<string name="question_africa">The source of the Nile River is in Egypt.</string>
<string name="question_americas">The Amazon River is the longest river
  in the Americas.</string>
<string name="question_asia">Lake Baikal is the world\'s oldest and deepest
  freshwater lake.</string>
...
```

Обратите внимание на использование служебной последовательности `\'` для включения апострофа в последнюю строку. В строковых ресурсах могут использоваться все стандартные служебные последовательности, включая `\n` для обозначения новой строки.

Вернитесь к файлу activity_quiz.xml и ознакомьтесь с изменениями макета в графическом конструкторе.

Пока это все, что относится к уровню представления GeoQuiz. Пора связать все воедино в классе контроллера `QuizActivity`.

Обновление уровня контроллера

В предыдущей главе в единственном контроллере GeoQuiz — `QuizActivity` — не происходило почти ничего. Он отображал макет, определенный в файле activity_quiz.xml, назначал слушателей для двух кнопок и организовывал выдачу уведомлений.

Теперь, когда у нас появились дополнительные вопросы, классу `QuizActivity` придется приложить дополнительные усилия для связывания уровней модели и представления GeoQuiz.

Откройте файл QuizActivity.java. Добавьте переменные для `TextView` и новой кнопки `Button`. Также создайте массив объектов `Question` и переменную для индекса массива.

Листинг 2.6. Добавление переменных и массива Question (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {

    private Button mTrueButton;
    private Button mFalseButton;
    private Button mNextButton;
    private TextView mQuestionTextView;

    private Question[] mQuestionBank = new Question[] {
        new Question(R.string.question_australia, true),
        new Question(R.string.question_oceans, true),
        new Question(R.string.question_mideast, false),
        new Question(R.string.question_africa, false),
        new Question(R.string.question_americas, true),
        new Question(R.string.question_asia, true),
    };

    private int mCurrentIndex = 0;
    ...
```

Программа несколько раз вызывает конструктор `Question` и создает массив объектов Question.

(В более сложном проекте этот массив создавался бы и хранился в другом месте. Позднее мы рассмотрим более правильные варианты хранения данных модели. А пока для простоты массив будет создаваться в контроллере.)

Мы собираемся использовать `mQuestionBank`, `mCurrentIndex` и методы доступа `Question` для вывода на экран серии вопросов.

Начнем с получения ссылки на `TextView` и задания тексту виджета вопроса с текущим индексом.

Листинг 2.7. Подключение виджета TextView (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_quiz);

        mQuestionTextView = (TextView) findViewById(R.id.question_text_view);
        int question = mQuestionBank[mCurrentIndex].getTextResId();
        mQuestionTextView.setText(question);

        mTrueButton = (Button) findViewById(R.id.true_button);
        ...
    }
}
```

Сохраните файлы и проверьте возможные ошибки. Запустите программу GeoQuiz. Первый вопрос из массива должен отображаться в виджете `TextView`.

Теперь разберемся с кнопкой NEXT. Получите ссылку на кнопку, назначьте ей слушателя `View.OnClickListener`. Этот слушатель будет увеличивать индекс и обновлять текст `TextView`.

Листинг 2.8. Подключение новой кнопки (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        mFalseButton.setOnClickListener(new View.OnClickListener() {
            ...
            }
        });

        mNextButton = (Button) findViewById(R.id.next_button);
        mNextButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
                int question = mQuestionBank[mCurrentIndex].getTextResId();
                mQuestionTextView.setText(question);
            }
        });
    }
}
```

Обновление переменной `mQuestionTextView` осуществляется в двух разных местах. Лучше выделить этот код в закрытый метод, как показано в листинге 2.9. Далее останется лишь вызвать этот метод в слушателе `mNextButton` и в конце `onCreate(Bundle)` для исходного заполнения текста в представлении активности.

Листинг 2.9. Инкапсуляция в методе updateQuestion() (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {
    ...
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        ...
        mQuestionTextView = (TextView) findViewById(R.id.question_text_view);
        int question = mQuestionBank[mCurrentIndex].getTextResId();
        mQuestionTextView.setText(question);
        ...
        mNextButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
                int question = mQuestionBank[mCurrentIndex].getTextResId();
                mQuestionTextView.setText(question);
                updateQuestion();
            }
        });

        updateQuestion();
    }

    private void updateQuestion() {
        int question = mQuestionBank[mCurrentIndex].getTextResId();
        mQuestionTextView.setText(question);
    }
}
```

Запустите GeoQuiz и протестируйте новую кнопку NEXT.

Итак, с вопросами мы разобрались, пора обратиться к ответам. В текущем состоянии приложение GeoQuiz считает, что на все вопросы ответ должен быть положительным; исправим этот недостаток. И снова мы реализуем закрытый метод для инкапсуляции кода, вместо того чтобы вставлять одинаковый код в двух местах.

Сигнатура метода, который будет добавлен в `QuizActivity`, выглядит так:

```
private void checkAnswer(boolean userPressedTrue)
```

Метод получает логическую переменную, которая указывает, какую кнопку нажал пользователь: TRUE или FALSE. Ответ пользователя проверяется по ответу текущего объекта `Question`. Наконец, после определения правильности ответа, метод создает уведомление для вывода соответствующего сообщения.

Включите в файл QuizActivity.java реализацию `checkAnswer(boolean)`, приведенную в листинге 2.10.

Листинг 2.10. Добавление метода checkAnswer(boolean) (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
    }

    private void updateQuestion() {
```

```
        int question = mQuestionBank[mCurrentIndex].getTextResId();
        mQuestionTextView.setText(question);
    }

    private void checkAnswer(boolean userPressedTrue) {
        boolean answerIsTrue = mQuestionBank[mCurrentIndex].isAnswerTrue();

        int messageResId = 0;

        if (userPressedTrue == answerIsTrue) {
            messageResId = R.string.correct_toast;
        } else {
            messageResId = R.string.incorrect_toast;
        }

        Toast.makeText(this, messageResId, Toast.LENGTH_SHORT)
            .show();
    }
}
```

Включите в слушателя кнопки вызова `checkAnswer(boolean)`, как показано в листинге 2.11.

Листинг 2.11. Вызов метода checkAnswer(boolean) (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        mTrueButton = (Button) findViewById(R.id.true_button);
        mTrueButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(QuizActivity.this,
                               R.string.correct_toast,
                               Toast.LENGTH_SHORT).show();
                checkAnswer(true);
            }
        });

        mFalseButton = (Button) findViewById(R.id.false_button);
        mFalseButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(QuizActivity.this,
                               R.string.incorrect_toast,
                               Toast.LENGTH_SHORT).show();
                checkAnswer(false);
            }
        });
        ...
    }
    ...
}
```

Программа GeoQuiz снова готова к работе. Давайте запустим ее на реальном устройстве.

Запуск на устройстве

В этом разделе мы займемся настройкой системы, устройства и приложения для выполнения GeoQuiz на физическом устройстве.

Подключение устройства

Прежде всего подключите устройство к системе. Если разработка ведется на Mac, ваша система должна немедленно распознать устройство. Системе Windows может потребоваться установка драйвера *adb* (Android Debug Bridge). Если Windows не может найти драйвер *adb*, загрузите его с сайта производителя устройства.

Настройка устройства для разработки

Чтобы вы могли тестировать приложения на своем устройстве, необходимо разрешить для устройства отладку USB.

Флажок Developer options по умолчанию не отображается. Чтобы включить его, откройте меню Settings ▸ About Tablet/Phone и нажмите Build Number семь раз. После этого вернитесь в меню Settings, найдите раздел Developer options и установите флажок USB debugging.

Настройки серьезно различаются между устройствами. Если у вас возникнут проблемы с включением отладки на вашем устройстве, обратитесь за помощью по адресу *developer.android.com/tools/device.html*.

Чтобы убедиться в том, что устройство было успешно опознано, откройте режим представления Devices. Для этого проще всего выбрать панель Android Monitor в нижней части окна Android Studio. На панели должен находиться раскрывающийся список подключенных устройств (рис. 2.7). В списке должны присутствовать как AVD, так и физическое устройство.

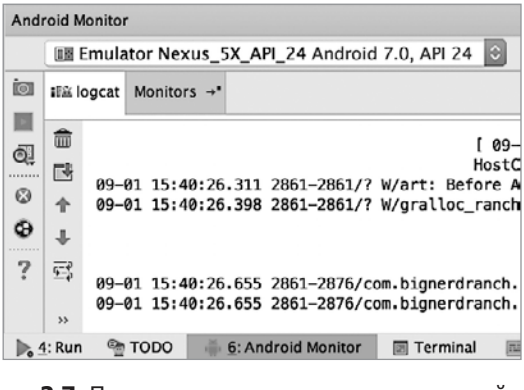


Рис. 2.7. Просмотр списка подключенных устройств

Если у вас возникнут трудности с распознаванием устройства, прежде всего убедитесь в том, что устройство включено и для него включена отладка USB.

Если устройство так и не появилось в режиме представления Devices, дополнительную информацию можно найти на сайте разработчиков Android. Начните со страницы *developer.android.com/tools/device.html* или обратитесь на форум книги *forums.bignerdranch.com* за дополнительной информацией о решении проблемы.

Запустите GeoQuiz так, как это делалось ранее. Android Studio предложит выбор между запуском на виртуальном устройстве и на физическом устройстве, подключенном к системе. Выберите физическое устройство; GeoQuiz запустится на выбранном устройстве.

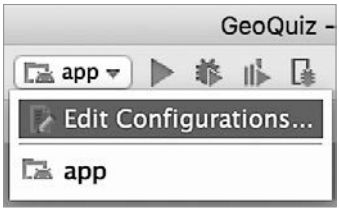


Рис. 2.8. Конфигурации запуска

Если Android Studio не предлагает выбрать устройство, а GeoQuiz запускается в эмуляторе, проверьте описанную ранее процедуру и убедитесь в том, что устройство подключено. Затем проверьте правильность конфигурации запуска; чтобы изменить параметры конфигурации, выберите раскрывающийся список в верхней части окна (рис. 2.8).

Выберите в списке строку Edit Configurations; откроется новое окно с подробной информацией о текущей конфигурации (рис. 2.9).

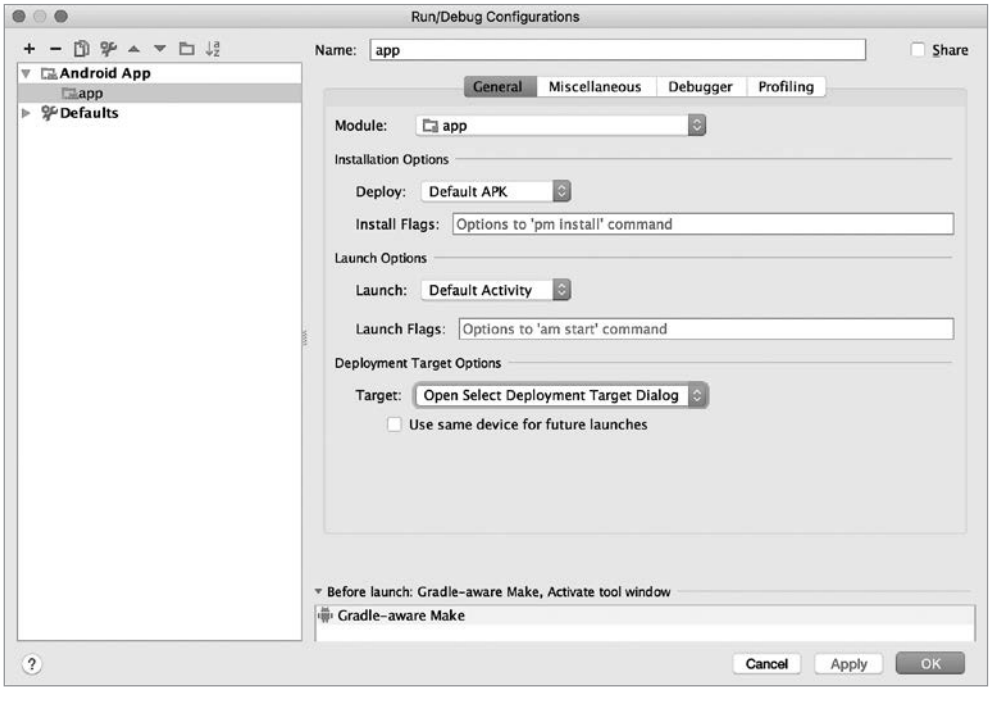


Рис. 2.9. Свойства конфигурации запуска

Выберите на левой панели строку app и убедитесь в том, что в разделе Deployment Target Options выбран вариант Open Select Deployment Target Dialog, а флажок Use same device for future launches не установлен. Щелкните на кнопке OK и запустите приложение заново. На этот раз вам будет предложено выбрать устройство для запуска.

Добавление значка

Приложение GeoQuiz работает, но пользовательский интерфейс смотрелся бы более привлекательно, если бы на кнопке NEXT была изображена стрелка, обращенная направо.

Изображение такой стрелки можно найти в файле решений этой книги (*www.bignerdranch.com/solutions/AndroidProgramming3e.zip*). Файл решений представляет собой набор проектов Android Studio для всех глав книги. Загрузите файл и откройте каталог 02_MVC/GeoQuiz/app/src/main/res. Найдите в нем подкаталоги drawable-hdpi, drawable-mdpi, drawable-xhdpi и drawable-xxhdpi.

Суффиксы имен каталогов обозначают экранную плотность пикселов устройства.

mdpi	Средняя плотность (~160 dpi)
hdpi	Высокая плотность (~240 dpi)
xhdpi	Сверхвысокая плотность (~320 dpi)
xxhdpi	Сверхсверхвысокая плотность (~480 dpi)

(Также существуют другие категории устройств, включая `ldpi` и `xxxhdpi`, но в решениях они не используются.)

Каждый каталог содержит два графических файла: arrow_right.png и arrow_left.png. Эти файлы адаптированы для плотности пикселов, указанной в имени каталога.

Мы включим в решения GeoQuiz все файлы изображений. При запуске ОС выбирает файл изображения, наиболее подходящий для конкретного устройства, на котором выполняется приложение. Следует учитывать, что дублирование изображений увеличивает размер приложения. В данном примере это не создает серьезных проблем, потому что GeoQuiz — очень простое приложение.

Если приложение выполняется на устройстве, экранная плотность которого не соответствует ни одному признаку в именах каталогов, Android автоматически масштабирует изображение к подходящему размеру. Благодаря этому обстоятельству не обязательно предоставлять изображения для всех категорий плотности. Для сокращения размера приложения можно сосредоточиться на одной или нескольких категориях высокой плотности и выборочно оптимизировать графику для уменьшенного разрешения, когда автоматическое масштабирование Android создает артефакты на устройствах с низким разрешением.

(Альтернативные варианты дублирования изображений с разной плотностью и каталог mipmap рассматриваются в главе 23.)

Добавление ресурсов в проект

Следующим шагом станет включение графических файлов в ресурсы приложения GeoQuiz.

Проверьте, чтобы в окне инструментов Project отображалось представление Project (выберите в раскрывающемся списке в верхней части окна инструментов Project строку Project, как показано на рис. 1.13 в главе 1). Раскройте содержимое узла GeoQuiz/app/src/main/res. В нем должны присутствовать папки с именами mipmap-hdpi и mipmap-xhdpi, показанные на рис. 2.10. (Также в списке присутствуют другие папки; пока не обращайте на них внимания.)

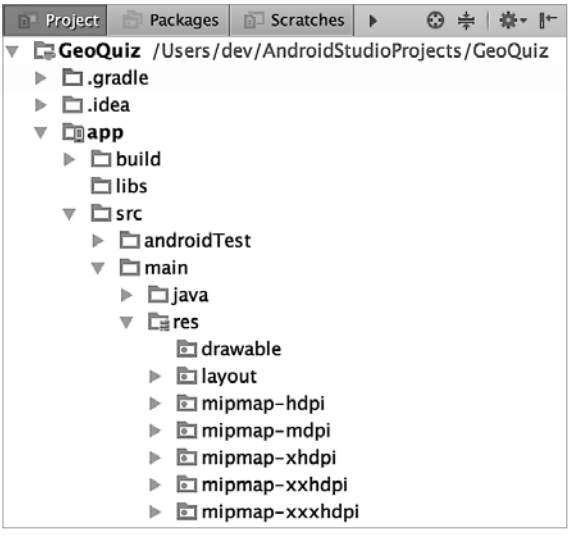


Рис. 2.10. Проверка существования каталогов drawable

В файле решения выделите и скопируйте четыре каталога, упоминавшихся ранее: drawable-hdpi, drawable-mdpi, drawable-xhdpi и drawable-xxhdpi. В Android Studio вставьте скопированные каталоги в каталог app/src/main/res. В результате должны появиться четыре каталога для разной плотности, каждый из которых содержит файлы arrow_left.png и arrow_right.png (рис. 2.11).

Переключив окно инструментов Project обратно в режим Android, вы увидите сводку добавленных файлов (рис. 2.12).

Процесс включения графики в приложение чрезвычайно прост. Любому файлу .png, .jpg или .gif, добавленному в папку res/drawable, автоматически назначается идентификатор ресурса. (Учтите, что имена файлов должны быть записаны в нижнем регистре и не могут содержать пробелов.)

Эти идентификаторы ресурсов не уточняются плотностью пикселов, так что вам не нужно определять плотность пикселов экрана во время выполнения; просто используйте идентификатор ресурса в коде. При запуске приложения ОС автоматически выберет изображение, подходящее для конкретного устройства.

Рис. 2.11. Изображения стрелок в каталогах drawable проекта GeoQuiz

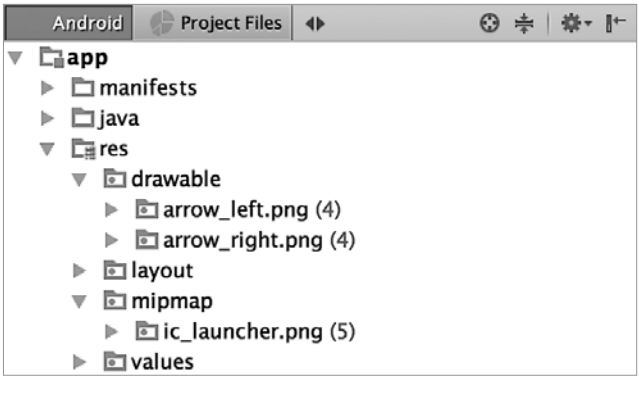


Рис. 2.12. Сводка изображений со стрелками в каталогах drawable приложения GeoQuiz

Система ресурсов Android более подробно рассматривается в главе 3 и далее. А пока давайте заставим работать нашу стрелку.

Ссылки на ресурсы в XML

Для ссылок на ресурсы в коде используются идентификаторы ресурсов. Но мы хотим настроить кнопку NEXT так, чтобы в определении макета отображалась стрелка. Как включить ссылку на ресурс в разметку XML?

Да почти так же, только с немного измененным синтаксисом. Откройте файл activity_quiz.xml и добавьте два атрибута в определение виджета `Button`.

Листинг 2.12. Включение графики в кнопку NEXT (activity_quiz.xml)

```
<LinearLayout ... >
    ...
    <LinearLayout ... >
        ...
    </LinearLayout>

    <Button
        android:id="@+id/next_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/next_button"
        android:drawableRight="@drawable/arrow_right"
        android:drawablePadding="4dp" />

</LinearLayout>
```

В ресурсах XML вы ссылаетесь на другой ресурс по его типу и имени. Ссылка на строку начинается с префикса `@string/`. Ссылка на графический ресурс начинается с префикса `@drawable/`.

Имена ресурсов и структура каталогов res более подробно рассматриваются начиная с главы 3.

Сохраните приложение GeoQuiz, запустите его и насладитесь новым внешним видом кнопки. Протестируйте ее и убедитесь, что кнопка работает точно так же, как прежде.

Рис. 2.13. Управление поворотом

Впрочем, в программе GeoQuiz скрывается ошибка. Во время выполнения GeoQuiz нажмите кнопку NEXT, чтобы перейти к следующему вопросу, а затем поверните устройство. (Если программа выполняется в эмуляторе, щелкните на кнопке поворота налево или направо на плавающей панели инструментов (рис. 2.13).)

После поворота мы снова видим первый вопрос. Почему это произошло и как исправить ошибку?

Ответы на эти вопросы связаны с жизненным циклом активности — этой теме посвящена глава 3.

Упражнение. Добавление слушателя для TextView

Кнопка NEXT удобна, но было бы неплохо сделать так, чтобы пользователь мог переходить к следующему вопросу простым нажатием на виджете `TextView`.

Подсказка. Для `TextView` можно использовать слушателя `View.OnClickListener`, который использовался с `Button`, потому что класс `TextView` также является производным от `View`.

Упражнение. Добавление кнопки возврата

Добавьте кнопку для возвращения к предыдущему вопросу. Пользовательский интерфейс должен выглядеть примерно так, как показано на рис. 2.14.

Рис. 2.14. Теперь с кнопкой возврата!

Это очень полезное упражнение. В нем вам придется вспомнить многое из того, о чем говорилось в двух предыдущих главах.

Упражнение. От Button к ImageButton

Возможно, пользовательский интерфейс будет смотреться еще лучше, если на кнопках будут отображаться только значки, как на рис. 2.15.

Рис. 2.15. Кнопки только со значками

Для этого оба виджета должны относиться к типу `ImageButton` (вместо обычного `Button`).

Виджет `ImageButton` является производным от `ImageView`, в отличие от виджета Button, производного от `TextView`. Диаграммы их наследования изображены на рис. 2.16.

Атрибуты `text` и `drawable` кнопки NEXT можно заменить одним атрибутом `ImageView`:

```
<Button ImageButton
  android:id="@+id/next_button"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/next_button"
  android:drawableRight="@drawable/arrow_right"
  android:drawablePadding="4dp"
  android:src="@drawable/arrow_right"
  />
```

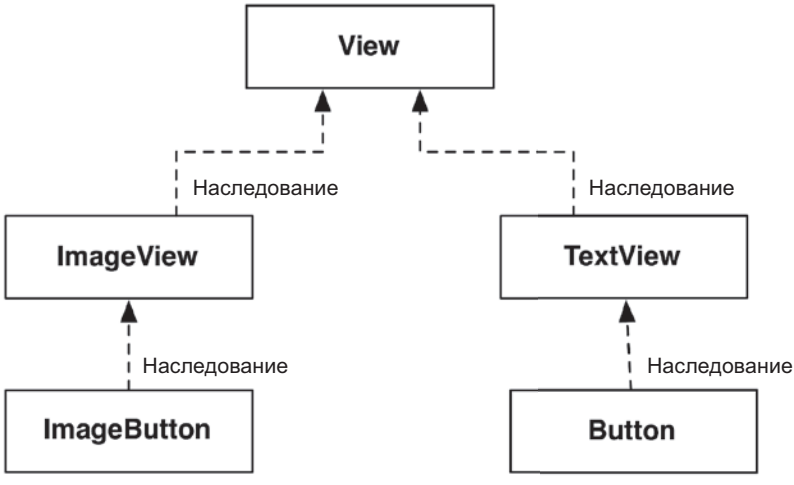


Рис. 2.16. Диаграмма наследования ImageButton и Button

Конечно, вам также придется внести изменения в `QuizActivity`, чтобы этот класс работал с `ImageButton`.

После того как вы замените эти кнопки на кнопки `ImageButton`, Android Studio выдаст предупреждение об отсутствии атрибута `android:contentDescription`. Этот атрибут обеспечивает доступность контента для читателей с ослабленным зрением. Строка, заданная этому атрибуту, читается экранным диктором (при включении соответствующих настроек в системе пользователя).

Наконец, добавьте атрибут `android:contentDescription` в каждый элемент `ImageButton`.

3 Жизненный цикл активности

Много ли пользы от приложения, которое сбрасывается в исходное состояние при повороте устройства? В конце главы 2 выяснилось, что приложение возвращается к первому вопросу при каждом повороте устройства (независимо от того, какой вопрос отображался до поворота). В этой главе вы научитесь решать ужасную — и чрезвычайно распространенную — «проблему поворота». Но чтобы решить ее, необходимо ознакомиться с азами жизненного цикла активности.

У каждого экземпляра `Activity` имеется жизненный цикл. Во время этого жизненного цикла активность переходит между четырьмя возможными состояниями: выполнение, приостановка, остановка и несуществование. Для каждого перехода у `Activity` существует метод, который оповещает активность об изменении состояния. На рис. 3.1 изображен жизненный цикл активности, состояния и методы.

На рис. 3.1 для каждого состояния указано, существует ли экземпляр активности в памяти, видим ли он для пользователя, и находится ли он на переднем плане (получает ли ввод от пользователя). Сводка этой информации приведена в табл. 3.1.

Таблица 3.1. Состояния активности

Состояние	Находится в памяти?	Видима ли для пользователя?	На переднем плане?
Активность не существует	Нет	Нет	Нет
Активность остановлена	Да	Нет	Нет
Активность приостановлена	Да	Да/частично*	Нет
Активность выполняется	Да	Да	Да

(* В зависимости от обстоятельств приостановленная активность может быть видима полностью или частично. Эта тема рассматривается более подробно в разделе «Анализ жизненного цикла активности на примере».)

Состояние выполнения представляет активность, с которой в настоящий момент взаимодействует пользователь. В любой момент времени на устройстве может быть только одна активность, находящаяся в состоянии выполнения.

Субклассы `Activity` могут использовать методы, представленные на рис. 3.1, для выполнения необходимых действий во время критических переходов жизненного

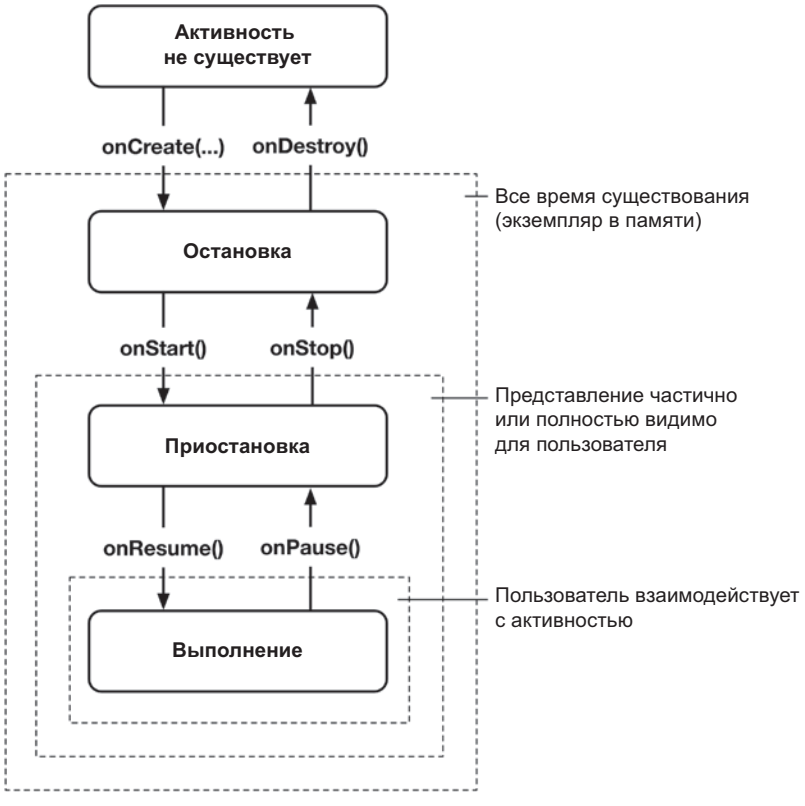


Рис. 3.1. Диаграмма состояния Activity

цикла активности. Эти методы часто называются *методами обратного вызова жизненного цикла*.

Один из таких методов вам уже знаком — это метод `onCreate(Bundle)`. ОС вызывает этот метод после создания экземпляра активности, но до его отображения на экране.

Как правило, активность переопределяет `onCreate(…)` для подготовки пользовательского интерфейса:

- заполнение виджетов и их вывод на экран (вызов `setContentView(int)`);
- получение ссылок на заполненные виджеты;
- назначение слушателей виджетам для обработки взаимодействия с пользователем;
- подключение к внешним данным модели.

Важно понимать, что вы никогда не вызываете `onCreate(…)` или другие методы жизненного цикла `Activity` в своих приложениях: вы переопределяете их в субклассах активности, а Android вызывает их в нужный момент времени (в зависимости от того, что делает пользователь и что происходит в системе) для того, чтобы оповестить приложение об изменении состояния.

Регистрация событий жизненного цикла Activity

В этом разделе мы переопределим методы жизненного цикла, чтобы отслеживать основные переходы жизненного цикла `QuizActivity`. Реализации ограничиваются регистрацией в журнале сообщения о вызове метода. Эти сообщения помогут вам наблюдать за изменениями состояния `QuizActivity` во время выполнения при выполнении действий пользователем.

Создание сообщений в журнале

В Android класс `android.util.Log` отправляет журнальные сообщения в общий журнал системного уровня. Класс `Log` предоставляет несколько методов регистрации сообщений. Следующий метод чаще всего встречается в этой книге:

```
public static int d(String tag, String msg)
```

Имя «`d`» означает «debug» (отладка) и относится к уровню регистрации сообщений. (Уровни `Log` более подробно рассматриваются в последнем разделе этой главы.) Первый параметр определяет источник сообщения, а второй — его содержимое.

Первая строка обычно содержит константу `TAG`, значением которой является имя класса. Это позволяет легко определять источник конкретного сообщения.

Откройте файл QuizActivity.java и добавьте константу `TAG` в `QuizActivity`:

Листинг 3.1. Добавление константы TAG (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {

    private static final String TAG = "QuizActivity";
    ...
}
```

Включите в `onCreate(…)` вызов `Log.d(…)` для регистрации сообщения.

Листинг 3.2. Включение команды регистрации сообщения в OnCreate(…) (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d(TAG, "onCreate(Bundle) called");
        setContentView(R.layout.activity_quiz);
        ...
    }
}
```

Теперь переопределите еще пять методов в `QuizActivity`; для этого добавьте следующие определения после `onCreate(Bundle)`:

Листинг 3.3. Переопределение методов жизненного цикла (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
    }

    @Override
    public void onStart() {
        super.onStart();
        Log.d(TAG, "onStart() called");
    }

    @Override
    public void onResume() {
        super.onResume();
        Log.d(TAG, "onResume() called");
    }

    @Override
    public void onPause() {
        super.onPause();
        Log.d(TAG, "onPause() called");
    }

    @Override
    public void onStop() {
        super.onStop();
        Log.d(TAG, "onStop() called");
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.d(TAG, "onDestroy() called");
    }
    ...
}
```

Обратите внимание на вызовы реализаций суперкласса перед регистрацией сообщений. Эти вызовы являются обязательными. Вызов реализации суперкласса должен располагаться в первой строке реализации переопределения любого метода обратного вызова.

Возможно, вы заметили аннотацию `@Override`. Она приказывает компилятору проследить за тем, чтобы класс действительно содержал переопределяемый метод. Например, компилятор сможет предупредить вас об опечатке в имени метода:

```
public class QuizActivity extends AppCompatActivity {

    @Override
    public void onCreat(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        ...
    }
    ...
}
```

Родительский класс `AppCompatActivity` не содержит метод `onCreat(Bundle)`, поэтому компилятор сообщает об ошибке. Это позволит вам исправить опечатку, вместо того чтобы догадываться об ошибке по странному поведению запущенного приложения.

Использование LogCat

Чтобы просмотреть системный журнал во время работы приложения, используйте LogCat — программу, включенную в инструментарий Android SDK.

При запуске GeoQuiz данные LogCat появляются в нижней части окна Android Studio (рис. 3.2). Если панель LogCat не видна, выберите панель Android Monitor в нижней части окна и перейдите на вкладку logcat.

Запустите GeoQuiz; на панели LogCat начнут быстро появляться сообщения. По умолчанию выводятся журнальные сообщения, сгенерированные с именем пакета вашего приложения. Вы увидите сообщения своей программы, а также некоторые системные сообщения.

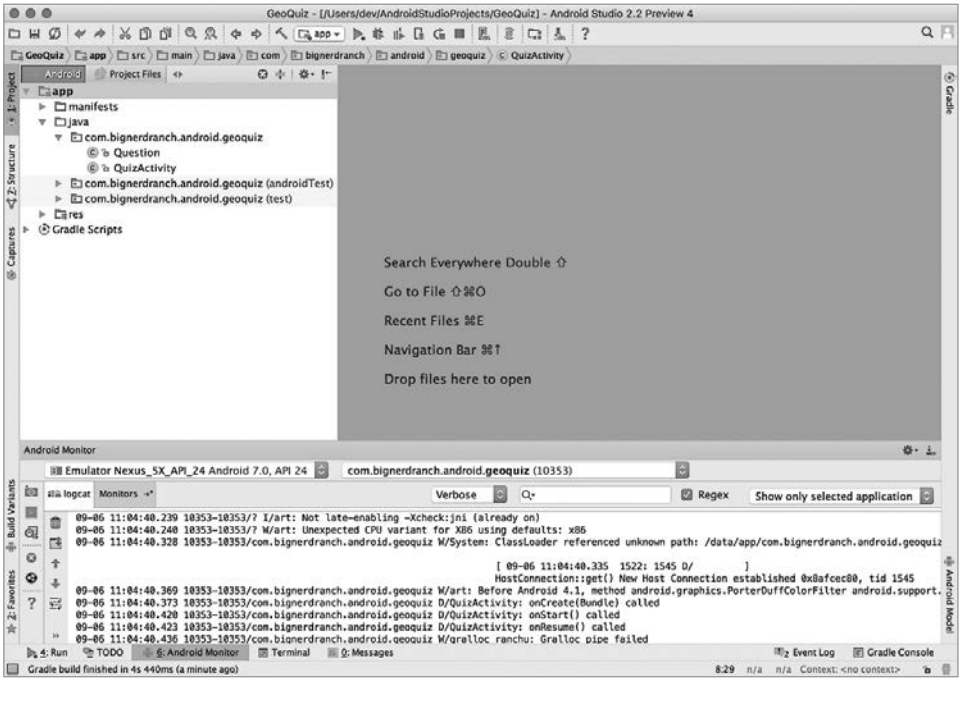


Рис. 3.2. Android Studio с выводом LogCat

Чтобы упростить поиск сообщений, можно отфильтровать вывод по константе TAG. В LogCat щелкните на раскрывающемся списке фильтра в правом верхнем углу панели. Обратите внимание на существующий фильтр, настроенный на вывод сообщений только от вашего приложения (Show only selected application). Если выбрать вариант No Filters, будут выводиться журнальные сообщения, сгенерированные в масштабах всей системы.

Выберите в раскрывающемся списке пункт Edit Filter Configuration. Нажмите кнопку + для создания нового фильтра. Введите имя фильтра QuizActivity и введите строку QuizActivity в поле by Log Tag: (рис. 3.3).

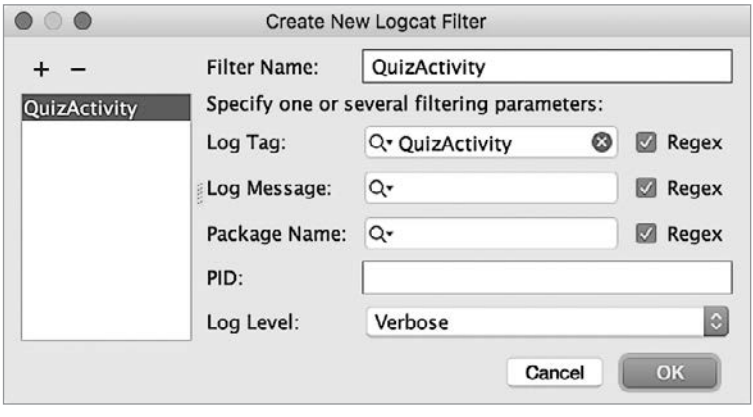


Рис. 3.3. Создание фильтра в LogCat

Щелкните на кнопке OK; после этого будут видны только сообщения с тегом QuizActivity (рис. 3.4).

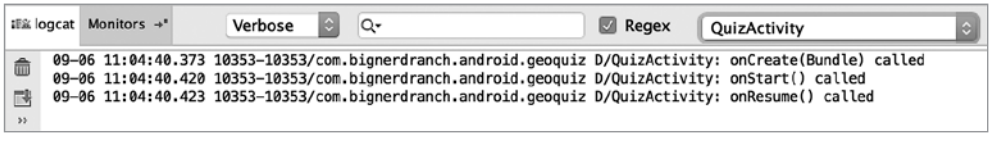


Рис. 3.4. При запуске GeoQuiz происходит создание, запуск и продолжение активности

Анализ жизненного цикла активности на примере

Как видите, после запуска GeoQuiz и создания исходного экземпляра `QuizActivity` были вызваны три метода жизненного цикла: `onCreate(Bundle)`, `onStart()` и `onResume()` (см. рис. 3.4). Экземпляр `QuizActivity` после этого находится в состоянии выполнения (он находится в памяти, виден пользователю и находится на переднем плане).

(Если вы не видите отфильтрованного списка, выберите фильтр QuizActivity из списка фильтров LogCat.)

А теперь немного поэкспериментируем. Нажмите на устройстве кнопку Back, а затем проверьте вывод LogCat. Активность приложения получила вызовы `onPause()`, `onStop()` и `onDestroy()` (рис. 3.5). Экземпляр `QuizActivity` после этого не существует (он не находится в памяти, не виден и, конечно, не находится на переднем плане).

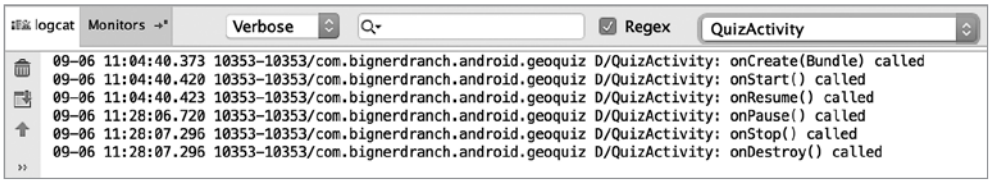


Рис. 3.5. Нажатие кнопки Back приводит к уничтожению активности

Нажимая кнопку Back, вы сообщаете Android: «Я завершил работу с активностью, и она мне больше не нужна». Android уничтожает активность и удаляет все следы ее существования из памяти, чтобы избежать неэффективного расходования ограниченных ресурсов устройства.

Перезапустите приложение GeoQuiz при помощи кнопки приложения GeoQuiz. Android создает новый экземпляр `QuizActivity` «с нуля» и вызывает методы `onCreate()`, `onStart()` и `onResume()`; `QuizActivity` переходит из состояния небытия в состояние выполнения.

Нажмите кнопку Home и проверьте вывод LogCat. На устройстве отображается домашний экран, а экземпляр `QuizActivity` становится невидимым. В каком состоянии сейчас находится `QuizActivity`? Информацию можно получить из LogCat. Ваша активность получила вызовы `onPause()` и `onStop()`, но не вызов `onDestroy()` (рис. 3.6).

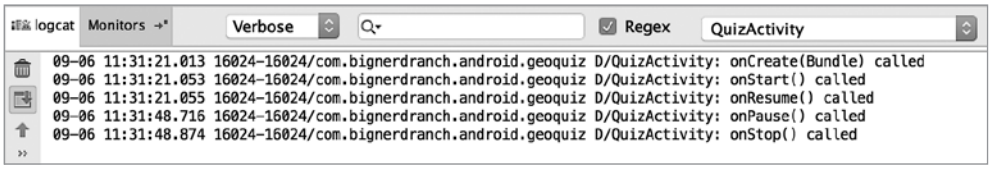


Рис. 3.6. Нажатие кнопки Home приводит к остановке активности

Нажатие кнопки Home сообщает Android: «Я сейчас займусь другим делом, но потом могу вернуться. Этот экран мне еще понадобится». Android приостанавливает, а в конечном итоге и останавливает активность. Это означает, что после нажатия Home ваш экземпляр `QuizActivity` пребывает в состоянии остановки (находится в памяти, не виден, не активен). Android делает это для того, чтобы быстро и легко перезапустить `QuizActivity` с того момента, на котором была прервана работа, если вы решите вернуться к GeoQuiz в будущем.

(Это не полное описание работы Home. Остановленные активности могут уничтожаться по усмотрению ОС — за более полным описанием обращайтесь к разделу «Снова о жизненном цикле активности».)

Вернитесь к приложению GeoQuiz, выбрав его в диспетчере задач. Для этого нажмите кнопку Recents рядом с кнопкой Home (рис. 3.7). На устройствах без кнопки Recents выполните долгое нажатие кнопки Home.

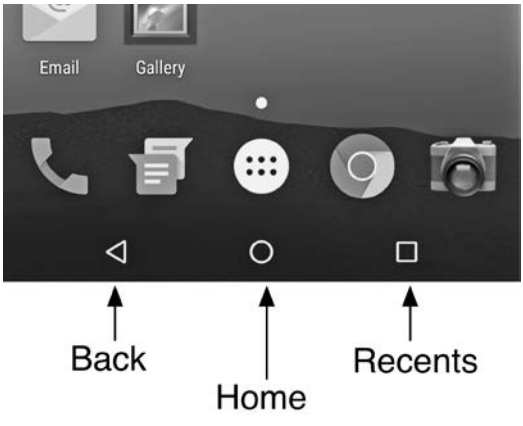


Рис. 3.7. Кнопки Back, Home и Recents

Каждая карточка в диспетчере задач представляет соой приложение, с которым пользователь взаимодействовал в прошлом (рис. 3.8). (В документации разработчика этот экран называется «сводным экраном», overview screen.)

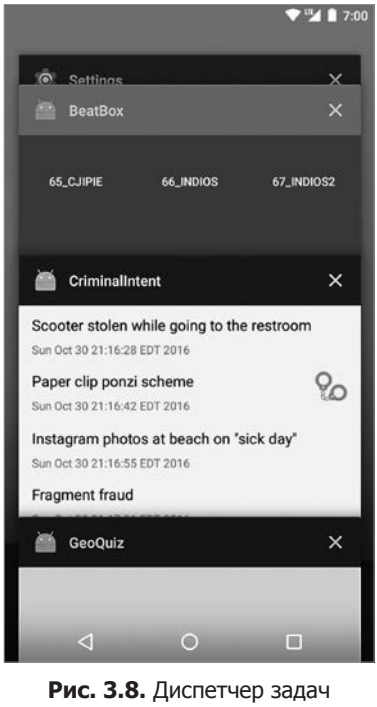


Рис. 3.8. Диспетчер задач

Нажмите на приложении GeoQuiz; `QuizActivity` заполнит экран.

Из данных Logcat следует, что активность получила вызовы `onStart()` и `onResume()`. Обратите внимание: метод `onCreate()` при этом *не вызывался*. Дело в том, что активность `QuizActivity` находилась в состоянии остановки после нажатия пользователем кнопки Home. Так как экземпляр активности все еще находится в памяти, создавать его не нужно. Вместо этого достаточно запустить активность (перевести ее в видимое состояние приостановки), а затем продолжить ее выполнение (перевести в состояние выполнения с получением ввода от пользователя).

Активность также может пребывать в приостановленном состоянии (т.е. быть видимой полностью или частично, но не быть активной). Такая ситуация может возникнуть тогда, когда поверх вашего приложения запускается новая активность с прозрачным фоном или активность, размеры которой меньше размеров экрана. Ситуация с полной видимостью встречается в многооконном режиме (поддерживаемом только в Android 6.0 Nougat и выше), когда пользователь работает с окном, не содержащим вашу активность, однако сама активность остается полностью видимой в другом окне.

Далее в этой книге мы будем переопределять различные методы жизненного цикла активности для решения реальных задач. При этом применение каждого метода будет рассматриваться более подробно.

Повороты и жизненный цикл активности

А теперь вернемся к ошибке, обнаруженной в конце главы 2. Запустите GeoQuiz, нажмите кнопку NEXT для перехода к следующему вопросу, а затем поверните устройство. (Чтобы имитировать поворот в эмуляторе, нажмите Ctrl + →/Ctrl + ←, или щелкните на кнопке поворота на панели инструментов.)

После поворота GeoQuiz снова выводит первый вопрос. Чтобы понять, почему это произошло, просмотрите вывод LogCat. Он выглядит примерно так, как показано на рис. 3.9.

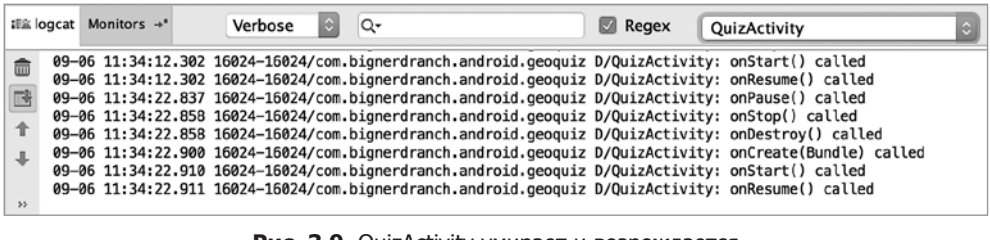


Рис. 3.9. QuizActivity умирает и возрождается

Когда вы поворачиваете устройство, экземпляр `QuizActivity`, который вы видели, уничтожается, и вместо него создается новый экземпляр. Снова поверните устройство — произойдет еще один цикл уничтожения и возрождения.

Из-за этого и возникает ошибка. Каждый раз, когда вы поворачиваете устройство, экземпляр `QuizActivity` полностью уничтожается. Значение, хранившееся

в переменной `mCurrentIndex` этого экземпляра, стирается из памяти. Таким образом, при повороте устройства приложение GeoQuiz «забывает», на какой вопрос вы смотрели, а после завершения поворота Android создает новый экземпляр `QuizActivity` с нуля. Переменная `mCurrentIndex` инициализируется (0) в вызове `onCreate(Bundle)`, а пользователь начинает с первого вопроса.

Вскоре мы исправим ошибку, но сначала подробнее разберемся, почему это происходит.

Конфигурации устройств и альтернативные ресурсы

Поворот приводит к изменению *конфигурации устройства*. Конфигурация устройства представляет собой набор характеристик, описывающих текущее состояние конкретного устройства. К числу характеристик, определяющих конфигурацию, относится ориентация экрана, плотность пикселов, размер экрана, тип клавиатуры, режим стыковки, язык и многое другое.

Как правило, приложения предоставляют альтернативные ресурсы для разных конфигураций устройств. Пример такого рода нам уже встречался: вспомните, как мы включали в проект несколько изображений стрелки для разной плотности пикселов.

Плотность пикселов является фиксированным компонентом конфигурации устройства; она не может измениться во время выполнения. Напротив, некоторые компоненты (такие, как ориентация) *могут* изменяться при выполнении. (Также возможны другие изменения конфигурации на стадии выполнения: доступность клавиатуры, язык, многооконный режим и т. д.)

При изменении конфигурации *во время выполнения* может оказаться, что приложение содержит ресурсы, лучше подходящие для новой конфигурации. По этой причине Android уничтожает активность, ищет ресурсы, которые лучше всего подходят для новой конфигурации, и заново строит экземпляр активности с этими ресурсами. Чтобы увидеть, как работает этот механизм, мы создадим альтернативный ресурс, который Android найдет и использует при изменении ориентации экрана.

Создание макета для альбомной ориентации

В окне инструментов Project щелкните правой кнопкой мыши на каталоге res и выберите команду New ▸ Android resource directory. Открывается окно со списками типов ресурсов и квалификаторами этих типов (наподобие изображенного на рис. 3.10). Выберите в раскрывающемся списке Resource type строку layout. Оставьте в списке Source set значение main.

Теперь необходимо выбрать способ уточнения ресурсов макета. Выберите в списке Available qualifiers строку Orientation и щелкните на кнопке >>, чтобы переместить значение Orientation в поле Chosen qualifiers.

Наконец, убедитесь в том, что в списке Screen Orientation выбрано значение Landscape (рис. 3.11). Проверьте, указано ли в поле Directory name имя каталога layout-land. Окно выглядит несколько экзотично, но единственная его цель — задание имени каталога. Щелкните на кнопке OK; Android Studio создаст папку res/layout-land/.

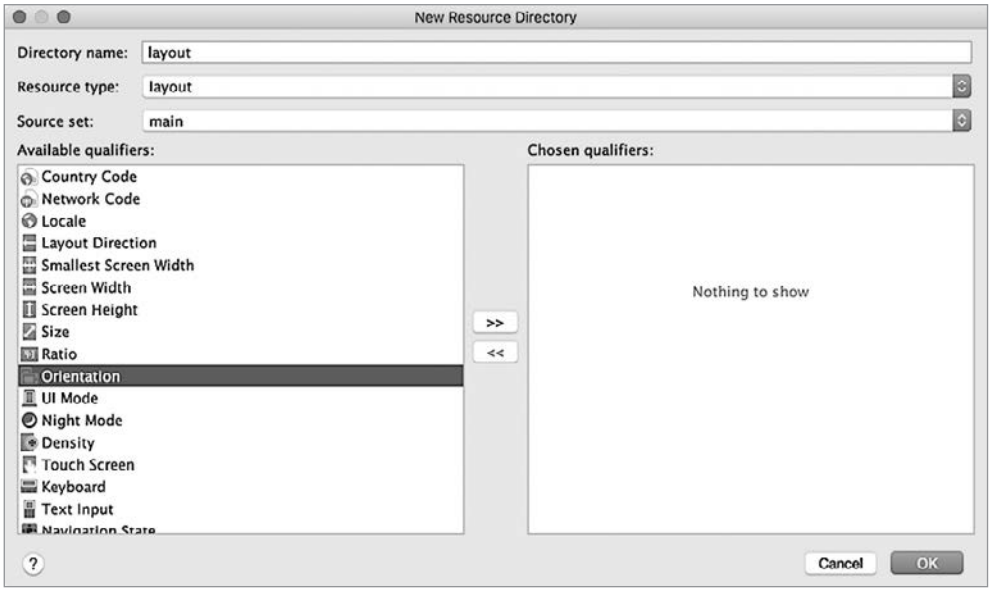


Рис. 3.10. Создание нового каталога ресурсов

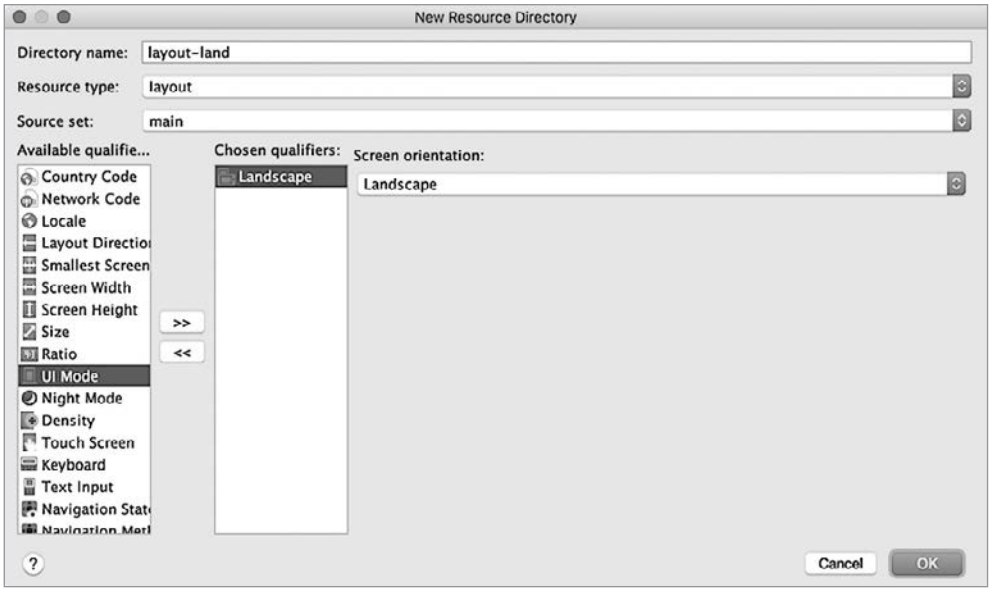


Рис. 3.11. Создание каталога res/layout-land

Суффикс -land — еще один пример конфигурационного квалификатора. По конфигурационным квалификаторам подкаталогов res Android определяет, какие ресурсы лучше всего подходят для текущей конфигурации устройства. Список кон-

фигурационных квалификаторов, поддерживаемых Android, и обозначаемых ими компонентов конфигурации устройств находится по адресу *developer.android.com/guide/topics/resources/providing-resources.html*.

Когда устройство находится в альбомной ориентации, Android находит и использует ресурсы в каталоге res/layout-land. В противном случае используются ресурсы по умолчанию из каталога res/layout/. Впрочем, на данный момент каталог res/layout-land не содержит ресурсов; этот недостаток нужно исправить.

Скопируйте файл activity_quiz.xml из res/layout/ в res/layout-land/. (Если в окне инструментов Project нет каталога res/layout-land/, выберите в раскрывающемся списке вариант Project для выхода из режима представления Android. Не забудьте вернуться в режим Android после выполнения операции. Также можно скопировать файл за пределами Android Studio в файловом менеджере или приложении командной строки.)

Теперь в приложении имеется два макета: макет для альбомной ориентации и макет по умолчанию. Оставьте имя файла без изменения. Два файла макетов должны иметь одинаковые имена, чтобы на них можно было ссылаться по одному идентификатору ресурса.

Теперь внесем некоторые изменения в альбомный макет, чтобы он отличался от макета по умолчанию. Сводка этих изменений представлена на рис. 3.12.

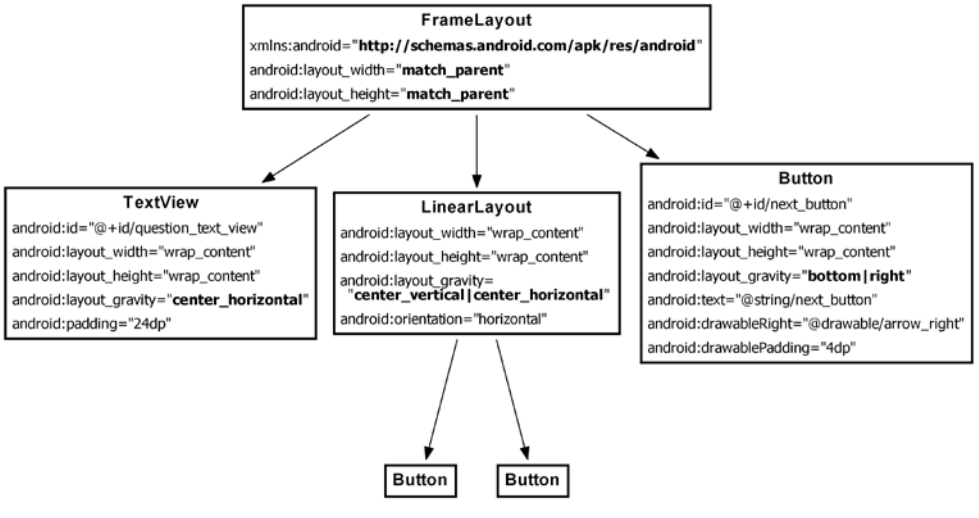


Рис. 3.12. Альтернативный макет для альбомной ориентации

Вместо `LinearLayout` будет использоваться `FrameLayout` — простейшая разновидность `ViewGroup` без определенного способа размещения потомков. В этом макете дочерние представления будут размещаться в соответствии с атрибутом `android:layout_gravity`.

Атрибут `android:layout_gravity` необходим виджетам `TextView`, `LinearLayout` и `Button`. Потомки `Button` виджета `LinearLayout` остаются без изменений.

Откройте файл layout-land/activity_quiz.xml и внесите необходимые изменения, руководствуясь рис. 3.12. Проверьте результат своей работы по листингу 3.4.

Листинг 3.4. Настройка альбомного макета (layout-land/activity_quiz.xml)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical" >

<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <TextView
        android:id="@+id/question_text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:padding="24dp" />

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical|center_horizontal"
        android:orientation="horizontal" >
      ...
    </LinearLayout>

    <Button
        android:id="@+id/next_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|right"
        android:text="@string/next_button"
        android:drawableRight="@drawable/arrow_right"
        android:drawablePadding="4dp"
        />

</LinearLayout>
</FrameLayout>
```

Снова запустите GeoQuiz. Поверните устройство в альбомную ориентацию, чтобы увидеть новый макет (рис. 3.13). Конечно, в программе используется не только новый макет, но и новый экземпляр `QuizActivity`.

Снова поверните устройство в книжную ориентацию, чтобы увидеть макет по умолчанию и другой экземпляр `QuizActivity`.

Рис. 3.13. QuizActivity в альбомной ориентации

Сохранение данных между поворотами

Android очень старается предоставить альтернативные ресурсы в нужный момент. Тем не менее уничтожение и повторное создание активностей при поворотах может создавать проблемы, как, например, в случае с возвратом к первому вопросу в приложении GeoQuiz.

Чтобы исправить эту ошибку, экземпляр `QuizActivity`, созданный после поворота, должен знать старое значение `mCurrentIndex`. Нам необходим механизм сохранения данных при изменении конфигурации времени выполнения (например, при поворотах). Одно из возможных решений заключается в переопределении метода Activity:

```
protected void onSaveInstanceState(Bundle outState)
```

Этот метод вызывается перед `onStop()`, кроме ситуации, в которой пользователь нажимает кнопку Back. (Вспомните: нажатие Back сообщает Android, что пользователь завершил работу с активностью, поэтому Android полностью стирает активность из памяти и не пытается сохранить данные для их повторного создания.)

Реализация по умолчанию `onSaveInstanceState(Bundle)` приказывает всем представлениям активности сохранить свое состояние в данных объекта `Bundle` — структуры, связывающей строковые ключи со значениями из ограниченного набора типов.

Тип `Bundle` нам уже встречался. Он передается методу `onCreate(Bundle)`:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ...
}
```

При переопределении `onCreate(Bundle)` вы вызываете реализацию `onCreate(Bundle)` суперкласса активности и передаете ей только что полученный объект `Bundle`. В реализации суперкласса сохраненное состояние представлений извлекается и используется для воссоздания иерархии представлений активности.

Переопределение onSaveInstanceState(Bundle)

Метод `onSaveInstanceState(Bundle)` можно переопределить так, чтобы он сохранял дополнительные данные в `Bundle`, а затем снова загружал их в `onCreate(Bundle)`. Именно так мы организуем сохранение значения `mCurrentIndex` между поворотами.

Для начала добавьте в константу QuizActivity.java, которая станет ключом в сохраняемой паре «ключ-значение».

Листинг 3.5. Добавление ключа для сохраняемого значения (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {

    private static final String TAG = "QuizActivity";
    private static final String KEY_INDEX = "index";

    private Button mTrueButton;
```

Теперь переопределим `onSaveInstanceState(…)` для записи значения `mCurrentIndex` в `Bundle` с использованием константы в качестве ключа.

Листинг 3.6. Переопределение onSaveInstanceState(…) (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {
    ...
    @Override
    protected void onPause() {
        ...
    }

    @Override
    public void onSaveInstanceState(Bundle savedInstanceState) {
        super.onSaveInstanceState(savedInstanceState);
        Log.i(TAG, "onSaveInstanceState");
        savedInstanceState.putInt(KEY_INDEX, mCurrentIndex);
    }

    @Override
    protected void onStop() {
        ...
    }
    ...
}
```

Наконец, в методе `onCreate(Bundle)` следует проверить это значение. Если оно существует, оно присваивается `mCurrentIndex`.

Листинг 3.7. Проверка сохраненных данных в onCreate(Bundle) (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d(TAG, "onCreate(Bundle) called");
        setContentView(R.layout.activity_quiz);

        if (savedInstanceState != null) {
            mCurrentIndex = savedInstanceState.getInt(KEY_INDEX, 0);
        }
        ...
    }
    ...
}
```

Запустите GeoQuiz и нажмите кнопку NEXT. Сколько бы поворотов устройства вы ни выполнили, вновь созданный экземпляр `QuizActivity` «вспоминает» текущий вопрос.

Учтите, что для сохранения в `Bundle` и восстановления из него подходят примитивные типы и классы, реализующие интерфейс `Serializable` или `Parcelable`. Впрочем, обычно сохранение объектов пользовательских типов в `Bundle` считается нежелательным, потому что данные могут потерять актуальность к моменту их извлечения. Лучше организовать для данных другой тип хранилища и сохранить в `Bundle` примитивный идентификатор.

Снова о жизненном цикле Activity

Переопределение `onSaveInstanceState(Bundle)` используется не только при поворотах или других изменениях конфигурации времени выполнения. Активность также может уничтожаться и в том случае, если пользователь занялся другими делами, а Android требуется освободить память (скажем, если пользователь нажал кнопку Home, решив посмотреть видеоролик или запустить игру).

С практической точки зрения ОС не освобождает память видимой (приостановленной или выполняемой) активности. Активность не помечается как разрешенная для уничтожения, пока не будет вызван метод `onStop()`, а выполнение активности не прекратится.

Впрочем, остановленная активность вполне подходит для уничтожения. Тем не менее беспокоиться не о чем. Если активность остановлена, значит, был вызван ее метод `onSaveInstanceState(Bundle)`. Таким образом, решение ошибки с поворотом также подходит для тех случаев, когда ОС уничтожает невидимую активность для освобождения памяти.

Как данные, сохраненные при вызове `onSaveInstanceState(Bundle)`, переживают уничтожение активности? При вызове `onSaveInstanceState(Bundle)` данные со-

храняются в объекте `Bundle`. Операционная система заносит объект `Bundle` в *запись активности* (activity record).

Чтобы понять, что собой представляет запись активности, добавим сохраненное состояние в схему жизненного цикла активности (рис. 3.14).

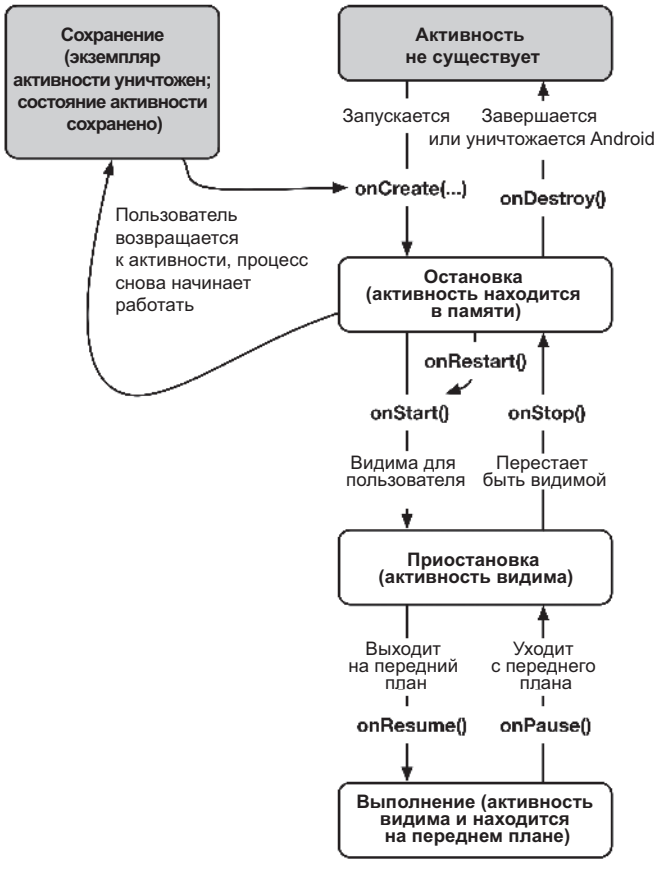


Рис. 3.14. Полный жизненный цикл активности

Когда ваша активность сохранена, объект `Activity` не существует, но объект записи активности «живет» в ОС. При необходимости операционная система может воссоздать активность по записи активности.

Следует учесть, что активность может перейти в сохраненное состояние без вызова `onDestroy()`. Однако вы всегда можете рассчитывать на то, что методы `onStop()` и `onSaveInstanceState(…)` были вызваны (если только на устройстве не возникли совсем уж критичные проблемы). Обычно метод `onSaveInstanceState(…)` переопределяется для сохранения небольших, временных состояний текущей активности данных в `Bundle`, а переопределение `onStop()` — для сохранения любых дол-

госрочных данных (например, редактируемого текста, потому что после возврата управления из этого метода активность может быть уничтожена в любой момент).

Когда же запись приложения пропадает? Когда пользователь нажимает кнопку Back, активность уничтожается раз и навсегда. При этом запись активности теряется. Записи активности также уничтожаются при перезагрузке.

Для любознательных: тестирование onSaveInstanceState(Bundle)

На момент написания книги сами активности не уничтожались при нехватке памяти. Вместо этого Android уничтожает в памяти весь процесс приложения, а вместе с ним и все находящиеся в памяти активности приложения. (Для каждого приложения создается отдельный процесс. Процессы приложений Android будут более подробно рассмотрены в разделе «Для любознательных: процессы и задачи» в главе 24.)

Процессы, содержащие активности переднего плана (выполняемые) и (или) видимые (приостановленные) имеют более высокий приоритет, чем другие процессы. Когда ОС потребуется освободить ресурсы, сначала выбираются низкоприоритетные процессы. С практической точки зрения процесс, содержащий видимую активность, не будет уничтожен ОС. Если процесс переднего плана будет уничтожен, это означает, что с устройством возникли какие-то серьезные проблемы (и, вероятно, уничтожение вашего приложения станет наименее серьезной из проблем пользователя).

Переопределяя `onSaveInstanceState(Bundle)`, необходимо убедиться в том, что состояние сохраняется и восстанавливается так, как предполагалось. Повороты тестируются легко; к счастью, ситуации с нехваткой памяти тоже не создают проблем. В этом можно легко убедиться самостоятельно.

Найдите в списке приложений и запустите приложение Settings. Выберите категорию Development options (возможно, список нужно будет прокрутить, пока вы не найдете нужную категорию). В нее входит много разных настроек; установите флажок Don't keep activities (рис. 3.15).

Запустите приложение и нажмите кнопку Home. Это приведет к приостановке и остановке активности. Затем остановленная активность будет уничтожена так, как если бы ОС Android решила освободить занимаемую ею память. Восстановите приложение и посмотрите, было ли состояние сохранено так, как ожидалось. Обязательно отключите этот режим после завершения тестирования, так как он приводит к снижению быстродействия и некорректному поведению некоторых приложений.

Помните, что нажатие кнопки Back (вместо Home) всегда приводит к уничтожению активности независимо от того, установлен флажок в настройках разработчика или нет. Нажатие кнопки Back сообщает ОС, что пользователь завершил работу с активностью.

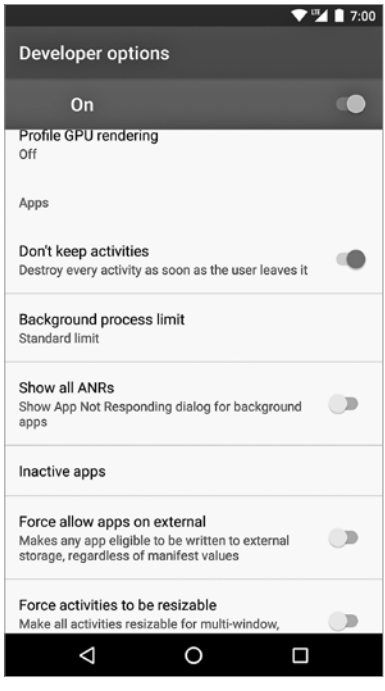


Рис. 3.15. Запрет сохранения активностей

Для любознательных: методы и уровни регистрации

Когда вы используете класс `android.util.Log` для регистрации сообщений в журнале, вы задаете не только содержимое сообщения, но и *уровень регистрации*, определяющий важность сообщения. Android поддерживает пять уровней регистрации (рис. 3.16). Каждому уровню соответствует свой метод класса `Log`. Регистрация данных в журнале сводится к вызову соответствующего метода `Log`.

Таблица 3.2. Методы и уровни регистрации

Уровень регистрации	Метод	Примечания
ERROR	Log.e(...)	Ошибки
WARNING	Log.w(...)	Предупреждения
INFO	Log.i(...)	Информационные сообщения
DEBUG	Log.d(...)	Отладочный вывод (может фильтроваться)
VERBOSE	Log.v(...)	Только для разработчиков

Каждый метод регистрации существует в двух вариантах: один получает строковый тег и строку сообщения, а второй получает эти же аргументы и экземпляр `Throwable`, упрощающий регистрацию информации о конкретном исключении, которое может быть выдано вашим приложением. В листинге 3.8 представлены примеры сигнатуры методов журнала. Для сборки строк сообщений используйте стандартные средства конкатенации строк Java или `String.format`, если их окажется недостаточно.

Листинг 3.8. Различные способы регистрации в Android

```
// Регистрация сообщения с уровнем отладки "debug"
Log.d(TAG, "Current question index: " + mCurrentIndex);

Question question;
try {
    question = mQuestionBank[mCurrentIndex];
} catch (ArrayIndexOutOfBoundsException ex) {
    // Регистрация сообщения с уровнем отладки "error"
    // вместе с трассировкой стека исключений
    Log.e(TAG, "Index was out of bounds", ex);
}
```

Упражнение. Предотвращение ввода нескольких ответов

После того как пользователь введет ответ на вопрос, заблокируйте кнопки этого вопроса, чтобы предотвратить возможность ввода нескольких ответов.

Упражнение. Вывод оценки

После того как пользователь введет ответ на все вопросы, отобразите уведомление с процентом правильных ответов. Удачи!

4 Отладка приложений Android

В этой главе вы узнаете, что делать, если в приложение закралась ошибка. В частности, вы научитесь пользоваться LogCat, Android Lint и отладчиком среды Android Studio.

Чтобы потренироваться в починке, нужно сначала что-нибудь сломать. В файле QuizActivity.java закомментируйте строку кода `onCreate(Bundle)`, в которой мы получаем `mQuestionTextView`.

Листинг 4.1. Из программы исключается важная строка (QuizActivity.java)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d(TAG, "onCreate(Bundle) called");
    setContentView(R.layout.activity_quiz);

    if (savedInstanceState != null) {
        mCurrentIndex = savedInstanceState.getInt(KEY_INDEX, 0);
    }

    // mQuestionTextView = (TextView)findViewById(R.id.question_text_view);

    mTrueButton = (Button)findViewById(R.id.true_button);
    mTrueButton.setOnClickListener(new View.OnClickListener() {
        ...
    });
    ...
}
```

Запустите GeoQuiz и посмотрите, что получится. На рис. 4.1 показано сообщение, которое выводится при сбое приложения. В разных версиях Android используются слегка различающиеся сообщения, но все они означают одно и то же.

Конечно, вы и так знаете, что случилось с приложением, но если бы не знали, было бы полезно взглянуть на приложение в другой перспективе.

Рис. 4.1. Сбой в приложении GeoQuiz

Исключения и трассировка стека

Откройте панель Android Monitor, чтобы понять, что же произошло. Прокрутите содержимое LogCat и найдите текст, выделенный красным шрифтом (рис. 4.2). Это стандартный отчет об исключениях `AndroidRuntime`.

Если информация об исключении отсутствует в LogCat, возможно, необходимо изменить фильтры LogCat: выберите в раскрывающемся списке фильтров вариант No Filters. С другой стороны, если данных в LogCat слишком много, также можно выбрать в списке Log Level значение Error, при котором отображаются только наиболее критичные сообщения. Также попробуйте поискать строку «FATAL EXCEPTION», которая приведет вас прямо к исключению, вызвавшему сбой приложения.

В отчете приводится исключение верхнего уровня и данные трассировки стека; затем исключение, которое привело к этому исключению, и *его* трассировка стека; и так далее, пока не будет найдено исключение, не имеющее причины.

Как правило, в написанном вами коде интерес представляет именно последнее исключение. В данном примере это исключение `java.lang.NullPointerException`. Строка непосредственно под именем исключения содержит начало трассировки стека. В ней указываются класс и метод, в котором произошло исключение, а также имя файла и номер строки кода. Щелкните на синей ссылке; Android Studio откроет указанную строку кода.

```
09-03 12:44:08.523    5458-5458/? E/AndroidRuntime: FATAL EXCEPTION: main
    Process: com.bignerdranch.android.geoquiz, PID: 5458
    java.lang.RuntimeException: Unable to start activity ComponentInfo{com.bignerdranch.android.ge
            at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2184)
            at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:2233)
            at android.app.ActivityThread.access$800(ActivityThread.java:135)
            at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1196)
            at android.os.Handler.dispatchMessage(Handler.java:102)
            at android.os.Looper.loop(Looper.java:136)
            at android.app.ActivityThread.main(ActivityThread.java:5001)
            at java.lang.reflect.Method.invokeNative(Native Method) <1 internal calls>
            at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:785)
            at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:601)
            at dalvik.system.NativeStart.main(Native Method)
     Caused by: java.lang.NullPointerException
            at com.bignerdranch.android.geoquiz.QuizActivity.updateQuestion(QuizActivity.java:35)
            at com.bignerdranch.android.geoquiz.QuizActivity.onCreate(QuizActivity.java:90)
            at android.app.Activity.performCreate(Activity.java:5231)
            at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1087)
            at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2148)
            at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:2233)
            at android.app.ActivityThread.access$800(ActivityThread.java:135)
            at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1196)
            at android.os.Handler.dispatchMessage(Handler.java:102)
            at android.os.Looper.loop(Looper.java:136)
            at android.app.ActivityThread.main(ActivityThread.java:5001)
            at java.lang.reflect.Method.invokeNative(Native Method)
            at java.lang.reflect.Method.invoke(Method.java:515) <3 more...>
```

Рис. 4.2. Исключения и трассировка стека в LogCat

В открывшейся строке программа впервые обращается к переменной `mQuestionTextView` в методе `updateQuestion()`. Имя исключения `NullPointerException` подсказывает суть проблемы: переменная не была инициализирована.

Раскомментируйте строку с инициализацией `mQuestionTextView`, чтобы исправить ошибку.

Помните: когда в вашей программе возникают исключения времени выполнения, следует искать последнее исключение в LogCat и первую строку трассировки стека со ссылкой на написанный вами код. Именно здесь возникает проблема, и именно здесь следует искать ответы.

Даже если сбой происходит на неподключенном устройстве, не все потеряно. Устройство сохраняет последние строки, выводимые в журнал. Длина и срок хранения журнала зависят от устройства, но обычно можно рассчитывать на то, что результаты будут храниться минимум 10 минут. Подключите устройство и выберите его на панели Devices. LogCat заполняется данными из сохраненного журнала.

Диагностика ошибок поведения

Проблемы с приложениями не всегда приводят к сбоям — в некоторых случаях приложение просто начинает некорректно работать. Допустим, пользователь нажимает кнопку NEXT, а в приложении ничего не происходит. Такие ошибки относятся к категории ошибок поведения.

В файле QuizActivity.java внесите изменение в слушателя `mNextButton` и закомментируйте код, увеличивающий `mCurrentIndex`.

Листинг 4.2. Из программы исключается важная строка (QuizActivity.java)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

   ...
    mNextButton = (Button)findViewById(R.id.next_button);
    mNextButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
            updateQuestion();
        }
    });
    ...
}
```

Запустите GeoQuiz и нажмите кнопку NEXT. Ничего не произойдет.

Эта ошибка коварнее предыдущей. Она не приводит к выдаче исключения, поэтому исправление ошибки не сводится к простому устранению исключения. Кроме того, некорректное поведение может быть вызвано разными причинами: то ли в программе не изменяется индекс, то ли не вызывается метод `updateQuestion()`.

Если вы не знаете причину происходящего, необходимо ее выяснить. Далее мы покажем два основных приема диагностики: сохранение трассировки стека и использование отладчика для назначения точки прерывания.

Сохранение трассировки стека

Включите команду сохранения отладочного вывода в метод `updateQuestion()` класса `QuizActivity`:

Листинг 4.3. Использование Exception (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {
    ...
    private void updateQuestion() {
        Log.d(TAG, "Updating question text", new Exception());
        int question = mQuestionBank[mCurrentIndex].getTextResId();
        mQuestionTextView.setText(question);
    }
```

Версия `Log.d` с сигнатурой `Log.d(String, String, Throwable)` регистрирует в журнале все данные трассировки стека — как уже встречавшееся ранее исключение `AndroidRuntime`. По данным трассировки стека вы сможете определить, в какой момент произошел вызов `updateQuestion()`.

При вызове `Log.d(String, String, Throwable)` вовсе не обязательно передавать перехваченное исключение. Вы можете создать новый экземпляр `Exception` и передать его методу без инициирования исключения. В журнале будет сохранена информация о том, где было создано исключение.

```
09-04 12:47:37.733  30612-30612/com.bignerdranch.android.geoquiz D/QuizActivity: Updating question text
    java.lang.Exception
            at com.bignerdranch.android.geoquiz.QuizActivity.updateQuestion(QuizActivity.java:34)
            at com.bignerdranch.android.geoquiz.QuizActivity.access$100(QuizActivity.java:12)
            at com.bignerdranch.android.geoquiz.QuizActivity$3.onClick(QuizActivity.java:83)
            at android.view.View.performClick(View.java:4438)
            at android.view.View$PerformClick.run(View.java:18422)
            at android.os.Handler.handleCallback(Handler.java:733)
            at android.os.Handler.dispatchMessage(Handler.java:95)
            at android.os.Looper.loop(Looper.java:136)
            at android.app.ActivityThread.main(ActivityThread.java:5001)
            at java.lang.reflect.Method.invokeNative(Native Method) <1 internal calls>
            at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:785)
            at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:601)
            at dalvik.system.NativeStart.main(Native Method)
```

Рис. 4.3. Результаты

Запустите приложение GeoQuiz, нажмите кнопку NEXT и проверьте вывод в LogCat (рис. 4.3).

Верхняя строка трассировки стека соответствует строке, в которой в журнале были зарегистрированы данные `Exception`. Следующая строка показывает, где был вызван `updateQuestion()`, — в реализации `onClick(View)`. Щелкните на ссылке в этой строке; откроется позиция, в которой была закомментирована строка с увеличением индекса заголовка. Но пока не торопитесь исправлять ошибку; сейчас мы найдем ее повторно при помощи отладчика.

Регистрация в журнале данных трассировки стека — мощный инструмент отладки, но он выводит довольно большой объем данных. Оставьте в программе несколько таких команд, и вскоре вывод LogCat превратится в невразумительную мешанину. Кроме того, по трассировке стека конкуренты могут разобраться, как работает ваш код, и похитить ваши идеи.

С другой стороны, в некоторых ситуациях нужна именно трассировка стека с информацией, показывающей, что делает ваш код. Если вы обратитесь за помощью на сайт *stackoverflow.com* или *forums.bignerdranch.com*, включите в вопрос трассировку стека. Информацию можно скопировать прямо из LogCat.

Прежде чем продолжать, удалите команду регистрации из QuizActivity.java.

Листинг 4.4. Прощай, друг (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {
    ...
    private void updateQuestion() {
        ~~Log.d(TAG, "Updating question text", new Exception());~~
        int question = mQuestionBank[mCurrentIndex].getTextResId();
        mQuestionTextView.setText(question);
    }
```

Установка точек прерывания

Попробуем найти ту же ошибку при помощи отладчика среды Android Studio. Мы установим *точку прерывания* в `updateQuestion()`, чтобы увидеть, был ли вызван этот метод. Точка прерывания останавливает выполнение программы в заданной позиции, чтобы вы могли в пошаговом режиме проверить, что происходит далее.

В файле QuizActivity.java вернитесь к методу `updateQuestion()`. В первой строке метода щелкните на серой полосе слева от кода. На месте щелчка появляется красный кружок; он обозначает точку прерывания (рис. 4.4).

```
private void updateQuestion() {
    int question = mQuestionStore[mCurrentIndex].getTextResId();
    mQuestionTextView.setText(question);
}
```

Рис. 4.4. Точка прерывания

Чтобы задействовать отладчик и активизировать точку прерывания, необходимо запустить приложение в отладочном режиме (в отличие от обычного запуска). Для этого щелкните на кнопке отладки (кнопка с зеленым жуком) рядом с кнопкой выполнения. Также можно выполнить команду меню Run ▸ Debug 'app'. Устройство сообщит, что оно ожидает подключения отладчика, а затем продолжит работу как обычно.

Когда приложение запустится и заработает под управлением отладчика, его выполнение будет прервано. Запуск GeoQuiz привел к вызову метода `QuizActivity.onCreate(Bundle)`, который вызвал `updateQuestion()`, что привело к срабатыванию точки прерывания.

На рис. 4.5 видно, что редактор открывает файл QuizActivity.java и выделяет строку с точкой прерывания, в которой было остановлено выполнение.

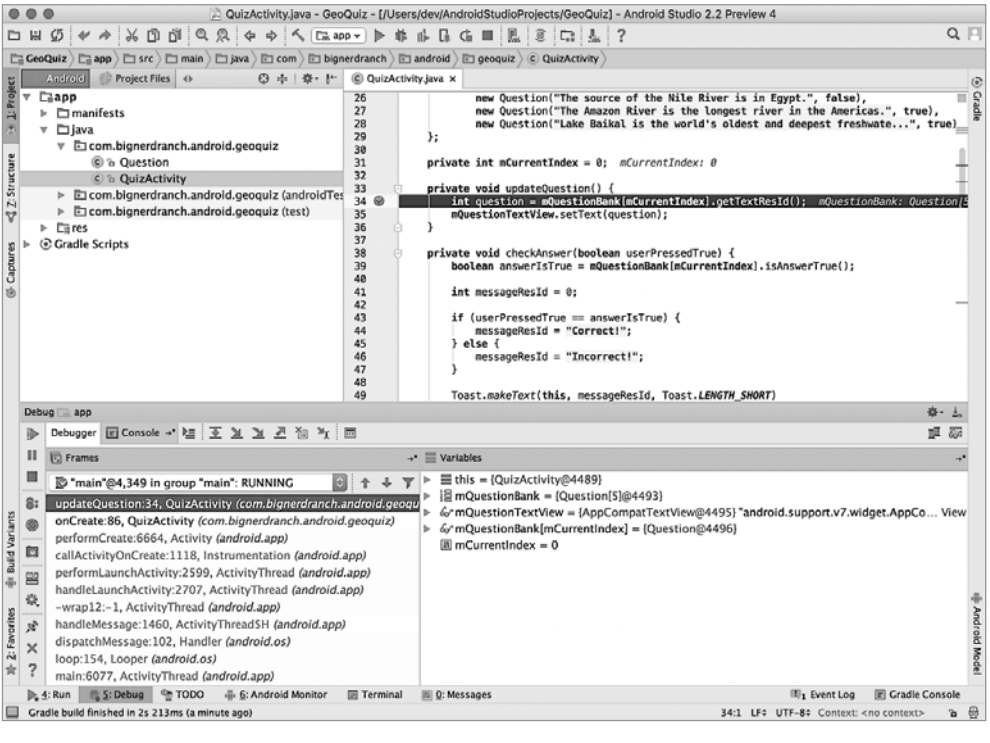


Рис. 4.5. Стоять!

В нижней части экрана теперь отображается панель Debug; она содержит области Frames и Variables (рис. 4.6).

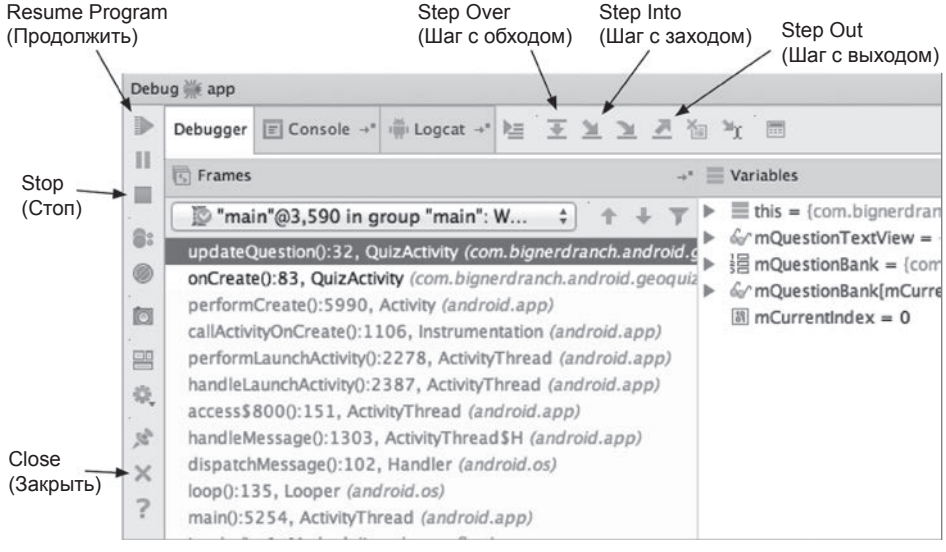


Рис. 4.6. Панель отладки

Кнопки со стрелками в верхней части панели используются для пошагового выполнения программы. Из трассировки стека видно, что метод `updateQuestion()` был вызван из `onCreate(Bundle)`. Так как нас интересует поведение кнопки NEXT, нажмите кнопку Resume, чтобы продолжить выполнение программы. Затем снова нажмите кнопку NEXT, чтобы увидеть, активизируется ли точка прерывания (она должна активизироваться).

Теперь, когда мы добрались до интересного момента выполнения программы, можно немного осмотреться. Представление Variables используется для просмотра текущих значений объектов вашей программы. На ней должны отображаться переменные, созданные в `QuizActivity`, а также еще одно значение: `this` (сам объект `QuizActivity`).

Переменную `this` можно было бы развернуть, чтобы увидеть все переменные, объявленные в суперклассе класса `QuizActivity` — `Activity`, в суперклассе класса `Activity`, в его суперклассе и т. д. Тем не менее пока мы ограничимся теми переменными, которые создали в программе.

Сейчас нас интересует только одно значение: `mCurrentIndex`. Прокрутите список и найдите в нем `mCurrentIndex`. Разумеется, переменная равна 0.

Код выглядит вполне нормально. Чтобы продолжить расследование, необходимо выйти из метода. Щелкните на кнопке шага с выходом Step Out.

Взгляните на панель редактора — управление передано слушателю `OnClickListener` кнопки `mNextButton`, в точку непосредственно после вызова `updateQuestion()`. Удобно, что и говорить.

Ошибку нужно исправить, но прежде чем вносить какие-либо изменения в код, необходимо прервать отладку приложения. Это можно сделать двумя способами: либо остановив программу, либо простым отключением отладчика. Чтобы остановить программу, щелкните на кнопке Stop на рис. 4.6. Вариант с отключением обычно проще: щелкните на кнопке Close, также обозначенной на рис. 4.6.

Верните `OnClickListener` в прежнее состояние.

Листинг 4.5. Возвращение к исходному состоянию (QuizActivity.java)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ...
    mNextButton = (Button)findViewById(R.id.next_button);
    mNextButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
            updateQuestion();
        }
    });
    ...
}
```

Мы рассмотрели два способа поиска проблемной строки кода: сохранение в журнале трассировки стека и установку точки прерывания в отладчике. Какой способ лучше? Каждый находит свои применения, и, скорее всего, один из них станет для вас основным.

Преимущество трассировки стека заключается в том, что трассировки из нескольких источников просматриваются в одном журнале. С другой стороны, чтобы получить новую информацию о программе, вы должны добавить новые команды регистрации, заново построить приложение, развернуть его и добраться до нужной точки. С отладчиком работать проще. Запустив приложение с подключенным отладчиком, вы сможете установить точку прерывания во время работы приложения, а потом поэкспериментировать для получения информации сразу о разных проблемах.

Прерывания по исключениям

Если вам недостаточно этих решений, вы также можете использовать отладчик для перехвата исключений. Вернитесь к методу `onCreate()` класса `QuizActivity` и снова закомментируйте строку кода, которая вызывала сбой приложения.

Листинг 4.6. Возвращение к нерабочей версии GeoQuiz (QuizActivity.java)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
    ...
    // mNextButton = (Button) findViewById(R.id.next_button);
```

```
    mNextButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
            updateQuestion();
        }
    });
    ...
}
```

Выполните команду Run ▸ View Breakpoints..., чтобы вызвать диалоговое окно прерываний по исключениям (рис. 4.7).

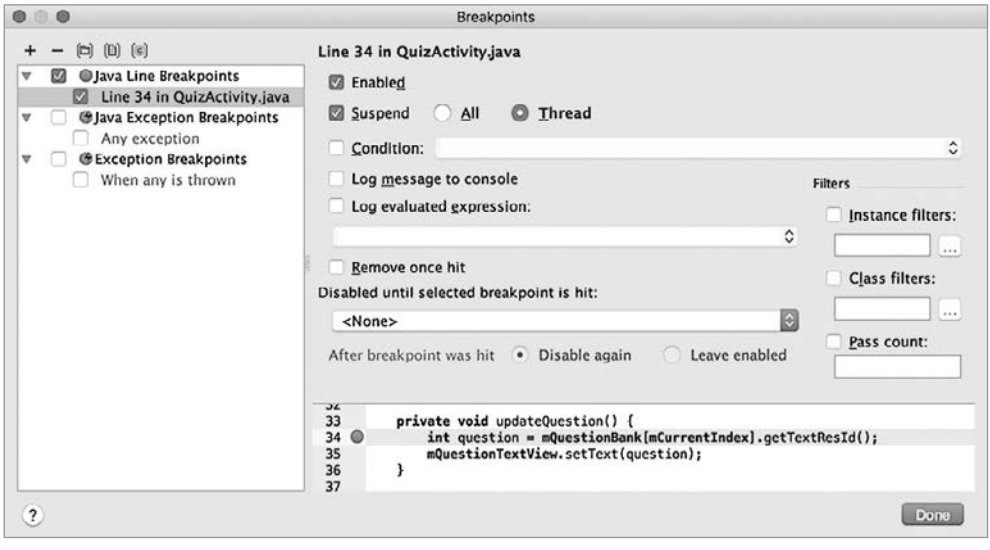


Рис. 4.7. Установка точки прерывания по исключению

В диалоговом окне перечислены все текущие точки прерывания. Удалите точку прерывания, добавленную ранее, — выделите ее и щелкните на кнопке со знаком «–».

Диалоговое окно также позволяет установить точку прерывания, которая срабатывает при инициировании исключения, где бы оно ни произошло. Прерывания можно ограничить только неперехваченными исключениями или же применить их как к перехваченным, так и к неперехваченным исключениям.

Щелкните на кнопке со знаком «+», чтобы добавить новую точку прерывания. Выберите в раскрывающемся списке строку Java Exception Breakpoints. Теперь можно выбрать тип перехватываемых исключений. Введите RuntimeException и выберите в списке предложенных вариантов RuntimeException (java.lang). `RuntimeException` является суперклассом `NullPointerException`, `ClassCastException` и других проблем времени выполнения, с которыми вы можете столкнуться, поэтому этот вариант удобен своей универсальностью.

Щелкните на кнопке Done и запустите GeoQuiz с присоединенным отладчиком. На этот раз отладчик перейдет прямо к строке, в которой было инициировано исключение, — замечательно.

Учтите, что, если эта точка прерывания останется включенной во время отладки, она может сработать на инфраструктурном коде или в других местах (на что вы не рассчитывали). Не забудьте отключить ее, если она не используется. Для удаления точки прерывания снова вернитесь к диалоговому окну Run ▸ View Breakpoints....

Отмените изменения из листинга 4.6 и верните приложение GeoQuiz в работоспособное состояние.

Особенности отладки Android

Как правило, процесс отладки в Android не отличается от обычной отладки кода Java. Тем не менее у вас могут возникнуть проблемы в областях, специфических для Android (например, ресурсы), о которых компилятор Java ничего не знает. В таких ситуациях на помощь приходит Android Lint.

Android Lint

Android Lint — *статический анализатор* кода Android. Статические анализаторы проверяют код на наличие дефектов, не выполняя его. Android Lint использует свое знание инфраструктуры Android для проверки кода и выявления проблем, которые компилятор обнаружить не может. Как правило, к рекомендациям Android Lint стоит прислушиваться.

В главе 6 вы увидите, как Android Lint выдает предупреждение о проблеме совместимости. Кроме того, Android Lint может выполнять проверку типов для объектов, определенных в XML. Попробуйте включить в QuizActivity следующую ошибку преобразования типов.

Листинг 4.7. Ошибка преобразования типа (QuizActivity.java)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d(TAG, "onCreate(Bundle) called");
    setContentView(R.layout.activity_quiz);
    ...
    mQuestionTextView = (TextView)findViewById(R.id.question_text_view);

    mTrueButton = (Button)findViewById(R.id.true_button);
    mTrueButton = (Button)findViewById(R.id.question_text_view);
    ...
}
```

Из-за указания неверного идентификатора ресурса код попытается преобразовать `TextView` в `Button` во время выполнения, что приведет к исключению неправильного преобразования типа. Компилятор Java не видит проблем в этом коде, но

Android Lint обнаружит ошибку еще до запуска приложения. Lint немедленно выделяет эту строку в коде, чтобы сообщить вам о наличии проблемы.

Вы можете запустить Lint вручную, чтобы получить список всех потенциальных проблем в приложении (в том числе и не столь серьезных, как эта). Выполните команду меню Analyze ▸ Inspect Code.... Вам будет предложено выбрать анализируемые части проекта. Выберите вариант Whole project и щелкните на кнопке OK. Android Studio запустит Lint, а также несколько других статических анализаторов кода.

После завершения анализа выводится список потенциальных проблем, разбитый на несколько категорий. Раскройте категорию Android Lint, чтобы просмотреть информацию Lint о вашем проекте (рис. 4.8).

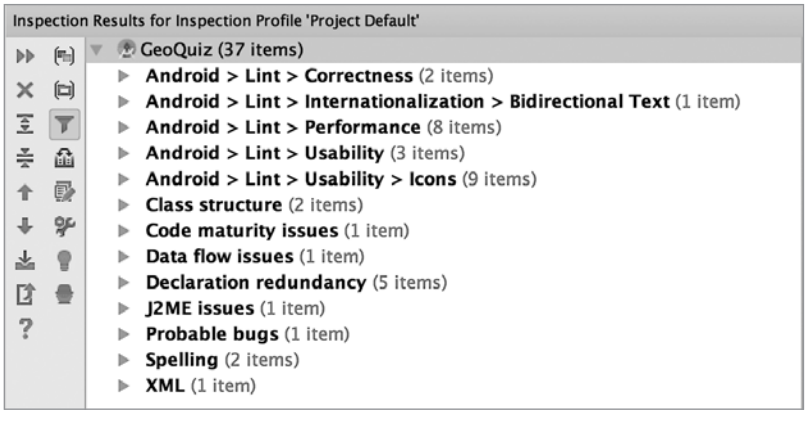


Рис. 4.8. Предупреждения Lint

Выберите проблему в списке, чтобы получить более подробную информацию и узнать ее местонахождение в проекте.

Ошибка несоответствия типов (Mismatched view type в категории Android ▸ Lint ▸ Correctness) уже известна: вы только что сами ее создали. Исправьте преобразование типа в `onCreate(Bundle)`.

Листинг 4.8. Исправление простой ошибки (QuizActivity.java)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d(TAG, "onCreate(Bundle) called");
    setContentView(R.layout.activity_quiz);

    mQuestionTextView = (TextView)findViewById(R.id.question_text_view);

    ~~mTrueButton = (Button)findViewById(R.id.question_text_view);~~
    mTrueButton = (Button)findViewById(R.id.true_button);
    ...
}
```

Запустите приложение GeoQuiz и убедитесь в том, что оно снова работает нормально.

Проблемы с классом R

Всем известны ошибки построения, которые возникают из-за ссылок на ресурсы до их добавления (или удаления ресурсов, на которые ссылаются другие файлы). Обычно повторное сохранение файла после добавления ресурса или удаления ссылки приводит к тому, что Android Studio проводит построение заново, а проблемы исчезают.

Однако иногда такие ошибки остаются или появляются из ниоткуда. Если вы столкнетесь с подобной ситуацией, попробуйте принять следующие меры.

Проверьте разметку XML в файлах ресурсов

Если файл R.java не был сгенерирован при последнем построении, то все ссылки на ресурс будут сопровождаться ошибками. Часто ошибки вызваны опечатками в разметке одного из файлов XML. Разметка макета не проверяется, поэтому среда не сможет привлечь ваше внимание к опечаткам в таких файлах. Если вы найдете ошибку и заново сохраните файл, код R.java будет сгенерирован заново.

Выполните чистку проекта

Выполните команду Build ▸ Clean Project. Android Studio строит проект заново, а результат построения нередко избавляется от ошибок. Глубокую чистку проекта полезно проводить время от времени.

Синхронизируйте проект с Gradle

Если вы вносите изменения в файл build.gradle, эти изменения необходимо синхронизировать с настройками построения вашего проекта. Выполните команду Tools ▸ Android ▸ Sync Project with Gradle Files. Android Studio строит проект заново с правильными настройками; при этом проблемы, связанные с изменением конфигурации Gradle, могут исчезнуть.

Запустите Android Lint

Обратите внимание на предупреждения, полученные от Android Lint. При запуске этой программы нередко выявляются совершенно неожиданные проблемы.

Если у вас все еще остаются проблемы с ресурсами (или иные проблемы), отдохните и просмотрите сообщения об ошибках и файлы макета на свежую голову. В запале легко пропустить ошибку. Также проверьте все ошибки и предупреждения Lint. При спокойном повторном рассмотрении сообщений нередко выявляются ошибки или опечатки.

Наконец, если вы зашли в тупик или у вас возникли другие проблемы с Android Studio, обратитесь к архивам *stackoverflow.com* или посетите форум книги по адресу *forums.bignerdranch.com*.

Упражнение. Layout Inspector

Для диагностики проблем с файлами макетов и интерактивного анализа визуализации макета на экране можно воспользоваться инструментом Layout Inspector. Убедитесь в том, что GeoQuiz выполняется в эмуляторе, и щелкните на значке Layout Inspector на левой панели окна Android Monitor (рис. 4.9). Далее вы сможете исследовать свойства своего макета, щелкая на элементах в представлении Layout Inspector.

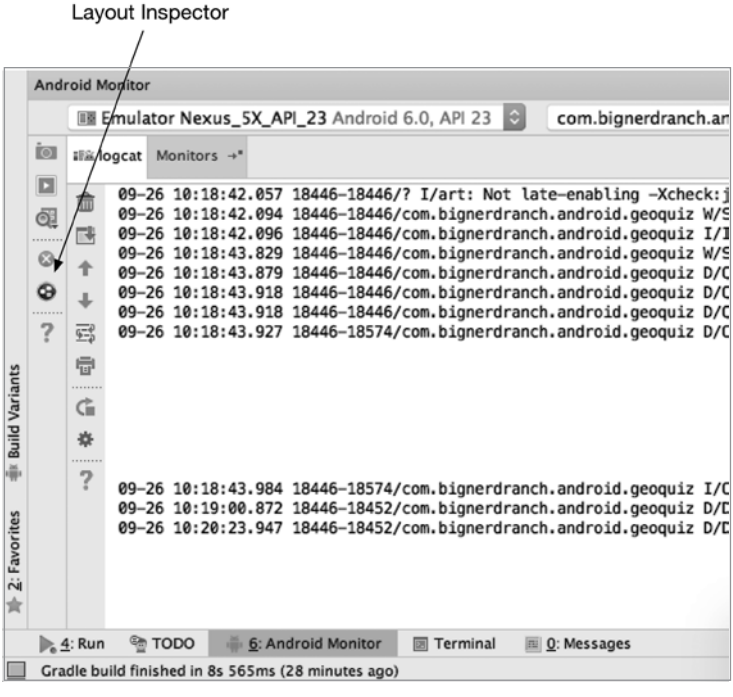


Рис. 4.9. Кнопка Layout Inspector

Упражнение. Allocation Tracker

Инструмент Allocation Tracker строит подробные отчеты о частоте и количестве вызовов выделения памяти в программе; эта информация помогает оптимизировать производительность приложения. В окне Android Monitor щелкните на кнопке Allocation Tracker (рис. 4.10).

Allocation Tracker начинает регистрировать операции выделения памяти при взаимодействии с приложением. После выполнения операции, которую вы хотите отслеживать, снова щелкните на кнопке, чтобы остановить отслеживание. На экране появляется отчет о выделении памяти (рис. 4.11).

В отчете приведено количество событий выделения памяти и размер каждого выделенного блока в байтах — в табличной и в визуальной форме. В верхней части окна инструментов можно выбрать тип отчета.

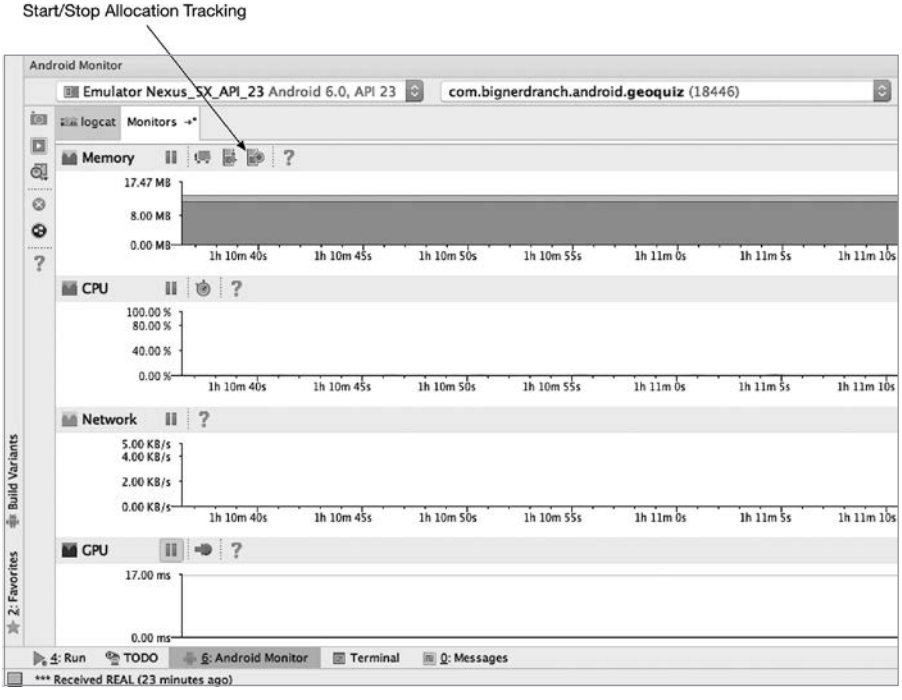


Рис. 4.10. Запуск Allocation Tracker

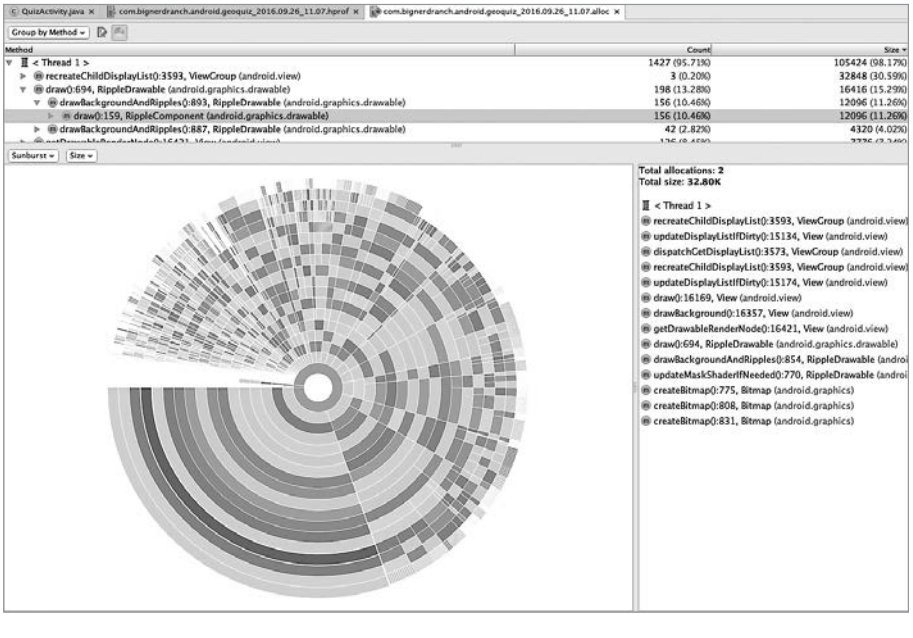


Рис. 4.11. Отчет Allocation Tracker

5 Вторая активность

В этой главе мы добавим в приложение GeoQuiz вторую активность. Как было сказано ранее, активность управляет информацией на экране; новая активность добавит в приложение второй экран, на котором пользователю будет предложено увидеть ответ на текущий вопрос.

Если пользователь решает просмотреть ответ, а затем возвращается к `QuizActivity` и отвечает на вопрос, он получает новое сообщение. Новая активность изображена на рис. 5.1.

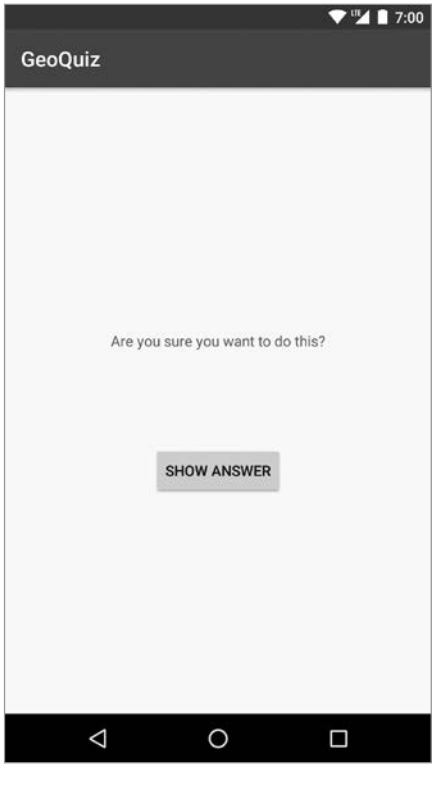


Рис. 5.1. CheatActivity позволяет подсмотреть ответ на вопрос

Если пользователь решает просмотреть ответ, а затем возвращается к `QuizActivity` и отвечает на вопрос, он получает новое сообщение (рис. 5.2).

Рис. 5.2. QuizActivity знает, что вы жульничаете

Почему эта задача является хорошим упражнением по программированию Android? Потому что вы научитесь:

- создавать новую активность и новый макет;
- запускать активность из другой активности (вы приказываете ОС создать экземпляр активности и вызвать его метод `onCreate(Bundle)`);
- передавать данные между родительской (запускающей) и дочерней (запущенной) активностью.

Подготовка к включению второй активности

В этой главе нам предстоит многое сделать. К счастью, часть рутинной работы будет выполнена за вас мастером New Activity среды Android Studio.

Но сначала откройте файл strings.xml и добавьте строки, необходимые для этой главы.

Листинг 5.1. Добавление строк (strings.xml)

```
<resources>
    ...
    <string name="incorrect_toast">Incorrect!</string>
    <string name="warning_text">Are you sure you want to do this?</string>
    <string name="show_answer_button">Show Answer</string>
    <string name="cheat_button">Cheat!</string>
    <string name="judgment_toast">Cheating is wrong.</string>
</resources>
```

Создание второй активности

При создании активности обычно изменяются по крайней мере три файла: файл класса Java, макет XML и манифест приложения. Но если эти файлы будут изменены некорректно, Android будет протестовать. Чтобы избежать возможных проблем, воспользуйтесь мастером New Activity среды Android Studio.

Запустите мастера New Activity: щелкните правой кнопкой мыши на пакете *com.bignerdranch.android.geoquiz* в окне инструментов Project и выберите команду New ▸ Activity ▸ Empty Activity (рис. 5.3).

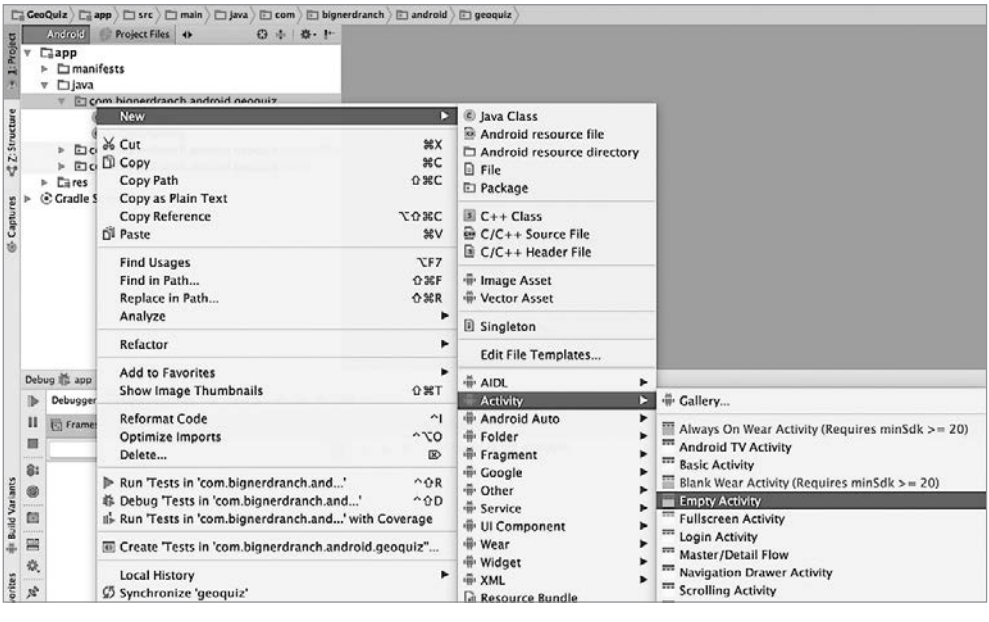


Рис. 5.3. Меню мастера New Activity

На экране появится диалоговое окно, изображенное на рис. 5.4. Введите в поле Activity Name строку CheatActivity — это имя вашего субкласса `Activity`. В поле Layout Name автоматически сгенерируется имя activity_cheat. Оно является базовым именем файла макета, создаваемого мастером.

Остальным полям можно оставить значения по умолчанию, но не забудьте убедиться в том, что имя пакета выглядит так, как ожидается. Оно определяет местонахождение CheatActivity.java в файловой системе. Щелкните на кнопке Finish, чтобы мастер завершил свою работу.

Теперь можно обратиться к пользовательскому интерфейсу. Снимок экрана в начале главы показывает, как должна выглядеть активность `CheatActivity`. На рис. 5.5 приведены определения виджетов.

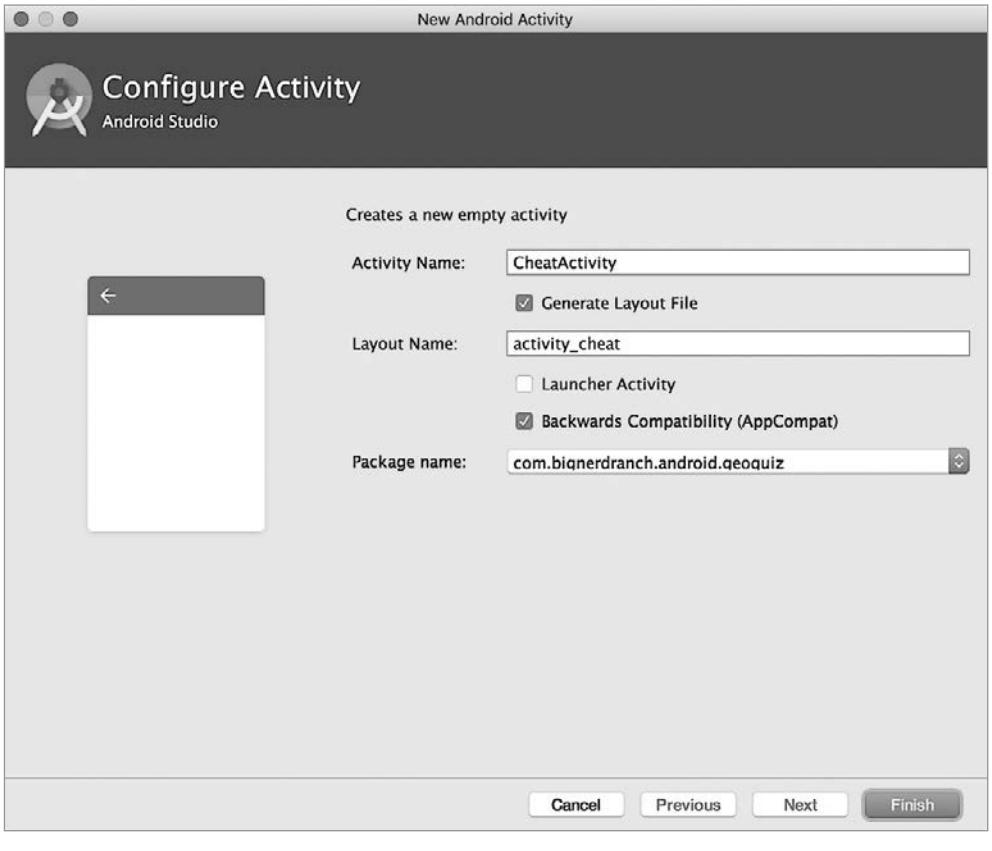


Рис. 5.4. Мастер New Blank Activity

Откройте файл activity_cheat.xml из каталога layout. Если этого не произошло, откройте файл самостоятельно и перейдите в режим представления Text (XML).

Попробуйте создать разметку XML для макета по образцу рис. 5.5. Замените простой макет элементом `LinearLayout`, и так далее по дереву представлений. После главы 9 в тексте вместо длинных фрагментов XML будут приводиться только схемы макетов вроде рис. 5.5, так что вам стоит освоить самостоятельное создание XML-макетов. Сравните свою работу с листингом 5.2.

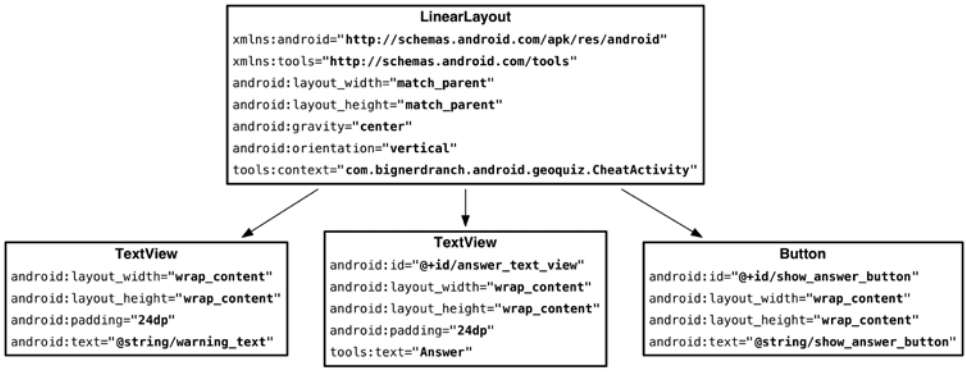


Рис. 5.5. Схема макета CheatActivity

Листинг 5.2. Заполнение макета второй активности (activity_cheat.xml)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              xmlns:tools="http://schemas.android.com/tools"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical"
              android:gravity="center"
              tools:context="com.bignerdranch.android.geoquiz.CheatActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp"
        android:text="@string/warning_text"/>

    <TextView
        android:id="@+id/answer_text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp"
        tools:text="Answer"/>

    <Button
        android:id="@+id/show_answer_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/show_answer_button"/>

</LinearLayout>
```

Обратите внимание на специальное пространство имен XML для `tools` и атрибута `tools:text` виджета `TextView`, в котором будет выводиться ответ. Это пространство имен позволяет переопределить любой атрибут виджета для того, чтобы он иначе отображался в режиме предварительного просмотра Android Studio. Так как у `TextView` есть атрибут `text`, вы можете присвоить ему фиктивное литеральное

значение, чтобы знать, как виджет будет выглядеть во время выполнения. Значение «Answer» никогда не появится в реальном приложении. Удобно, верно?

Мы не будем создавать для activity_cheat.xml альтернативный макет с альбомной ориентацией, однако есть возможность увидеть, как макет по умолчанию будет отображаться в альбомном режиме.

В окне предварительного просмотра найдите на панели инструментов над панелью предварительного просмотра кнопку, на которой изображено устройство с изогнутой стрелкой. Щелкните на этой кнопке, чтобы изменить ориентацию макета (рис. 5.6).

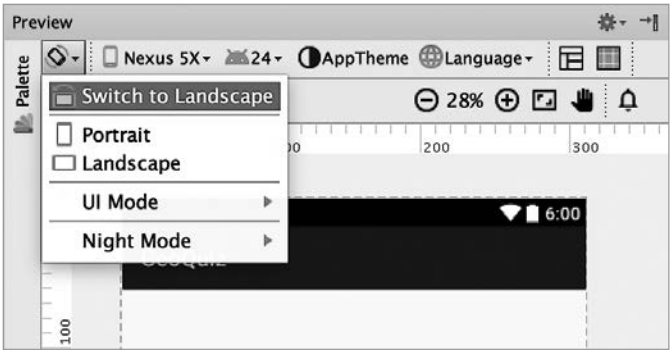


Рис. 5.6. Просмотр макета activity_cheat.xml в альбомной ориентации

Макет по умолчанию достаточно хорошо смотрится в обеих ориентациях, можно переходить к созданию субкласса активности.

Создание нового субкласса активности

В окне инструментов Project найдите пакет *com.bignerdranch.android.geoquiz* и откройте класс `CheatActivity` (он находится в файле CheatActivity.java).

Класс уже содержит простейшую реализацию `onCreate(Bundle)`, которая передает идентификатор ресурса макета, определенного в activity_cheat.xml, при вызове `setContentView(…)`.

Со временем функциональность метода `onCreate(Bundle)` в `CheatActivity` будет расширена. А пока посмотрим, что еще мастер New Activity сделал за вас: в манифест приложения было включено объявление `CheatActivity`.

Объявление активностей в манифесте

Манифест (manifest) представляет собой файл XML с метаданными, описывающими ваше приложение для ОС Android. Файл манифеста всегда называется AndroidManifest.xml и располагается в каталоге app/manifests вашего проекта.

В окне инструментов Project найдите и откройте файл AndroidManifest.xml. Также можно воспользоваться диалоговым окном Android Studio Quick Open — нажмите

Command+Shift+O (Ctrl+Shift+N) и начните вводить имя файла. Когда в окне будет предложен правильный файл, нажмите Return, чтобы открыть этот файл.

Каждая активность приложения должна быть объявлена в манифесте, чтобы она стала доступной для ОС.

Когда вы использовали мастер New Project для создания `QuizActivity`, мастер объявил активность за вас. Аналогичным образом мастер New Activity объявил `CheatActivity`, добавив разметку XML, выделенную в листинге 5.3.

Листинг 5.3. Объявление CheatActivity в манифесте (AndroidManifest.xml)

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bignerdranch.android.geoquiz" >

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <activity android:name=".QuizActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>

        </activity>
        <activity android:name=".CheatActivity">
        </activity>
    </application>

</manifest>
```

Атрибут `android:name` является обязательным. Точка в начале значения атрибута сообщает ОС, что класс этой активности находится в пакете, который задается атрибутом `package` в элементе `manifest` в начале файла.

Иногда встречается полностью уточненный атрибут `android:name` вида `android:name="com.bignerdranch.android.geoquiz.CheatActivity"`. Длинная форма записи идентична версии в листинге 5.3.

Манифест содержит много интересной информации, но сейчас мы хотим как можно быстрее наладить работу `CheatActivity`. Другие части манифеста будут рассмотрены позднее.

Добавление кнопки Cheat в QuizActivity

Итак, пользователь должен нажать кнопку в `QuizActivity`, чтобы вызвать на экран экземпляр `CheatActivity`. Следовательно, мы должны включить новые кнопки в layout/activity_quiz.xml и layout-land/activity_quiz.xml.

В макете по умолчанию добавьте новую кнопку как прямого потомка корневого элемента `LinearLayout`. Ее определение должно непосредственно предшествовать кнопке NEXT.

Листинг 5.4. Добавление кнопки Cheat! в макет по умолчанию (layout/activity_quiz.xml)

```
        ...
    </LinearLayout>

    <Button
        android:id="@+id/cheat_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/cheat_button"/>

    <Button
        android:id="@+id/next_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/next_button"
        android:drawableRight="@drawable/arrow_right"
        android:drawablePadding="4dp"/>

</LinearLayout>
```

В альбомном макете новая кнопка размещается внизу и по центру корневого элемента `FrameLayout`.

Листинг 5.5. Добавление кнопки CHEAT! в альбомный макет (layout-land/activity_quiz.xml)

```
        ...
    </LinearLayout>

    <Button
        android:id="@+id/cheat_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|center_horizontal"
        android:text="@string/cheat_button" />

    <Button
        android:id="@+id/next_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|right"
        android:text="@string/next_button"
        android:drawableRight="@drawable/arrow_right"
        android:drawablePadding="4dp" />

</FrameLayout>
```

Снова откройте файл QuizActivity.java. Добавьте переменную, получите ссылку и назначьте заглушку `View.OnClickListener` для кнопки CHEAT!.

Листинг 5.6. Подключение кнопки CHEAT! (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {
    ...
    private Button mNextButton;
    private Button mCheatButton;
    private TextView mQuestionTextView;
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        mNextButton = (Button) findViewById(R.id.next_button);
        mNextButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
                updateQuestion();
            }
        });

        mCheatButton = (Button)findViewById(R.id.cheat_button);
        mCheatButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Start CheatActivity
            }
        });

        updateQuestion();
    }
 ...
}
```

Теперь можно переходить к запуску `CheatActivity`.

Запуск активности

Чтобы запустить одну активность из другой, проще всего воспользоваться методом `startActivity`:

```
public void startActivity(Intent intent)
```

Напрашивается предположение, что `startActivity(Intent)` является статическим методом, который должен вызываться для запускаемого субкласса `Activity`. Тем не менее это не так. Когда активность вызывает `startActivity(Intent)`, этот вызов передается ОС.

Точнее, он передается компоненту ОС, который называется `ActivityManager`. `ActivityManager` создает экземпляр `Activity` и вызывает его метод `onCreate(Bundle)` (рис. 5.7).

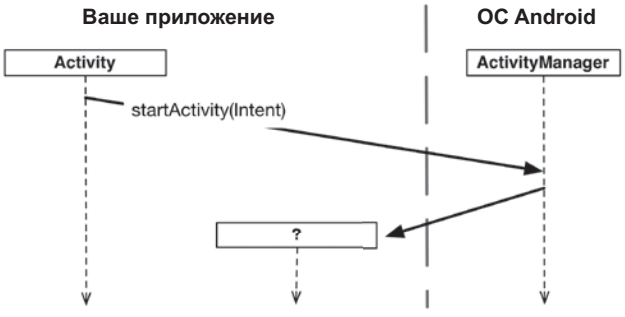


Рис. 5.7. Запуск активности

Откуда `ActivityManager` узнает, какую активность следует запустить? Эта информация передается в параметре `Intent`.

Передача информации через интенты

Интент (intent) — объект, который может использоваться *компонентом* для взаимодействия с ОС. Пока что из компонентов нам встречались только активности, но еще существуют службы (services), широковещательные приемники (broadcast receivers) и поставщики контента (content providers).

Интенты представляют собой многоцелевые средства передачи информации, а класс `Intent` предоставляет разные конструкторы в зависимости от того, для чего должен использоваться интент.

В данном случае интент сообщает `ActivityManager`, какую активность следует запустить, поэтому мы используем следующий конструктор:

```
public Intent(Context packageContext, Class<?> cls)
```

Объект `Class` задает класс активности, которая должна быть запущена `ActivityManager`. Аргумент `Context` сообщает `ActivityManager`, в каком пакете находится объект `Class` (рис. 5.8).

В слушателе `mCheatButton` создайте объект `Intent`, включающий класс `CheatActivity`, а затем передайте его при вызове `startActivity(Intent)` (листинг 5.7).

Листинг 5.7. Запуск CheatActivity (QuizActivity.java)

```
mCheatButton = (Button)findViewById(R.id.cheat_button);
mCheatButton.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        // Запуск CheatActivity
        Intent intent = new Intent(QuizActivity.this, CheatActivity.class);
        startActivity(intent);
    }
});
```

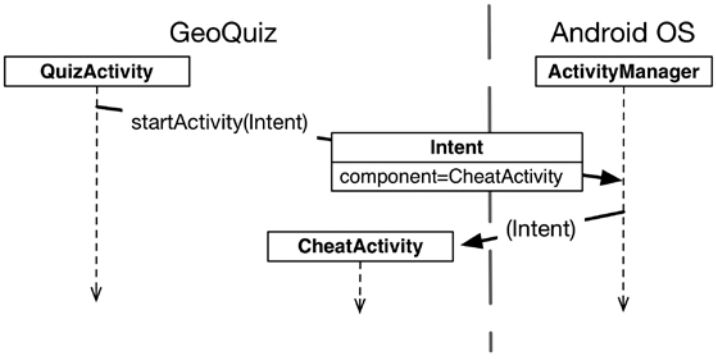


Рис. 5.8. Интент: передача ActivityManager информации о том, что нужно сделать

Прежде чем запустить активность, `ActivityManager` ищет в манифесте пакета объявление с именем, соответствующим заданному объекту `Class`. Если такое объявление будет найдено, активность запускается, все хорошо. Если объявление не найдено, выдается неприятное исключение `ActivityNotFoundException`, которое может привести к аварийному завершению приложения. Вот почему все активности должны объявляться в манифесте.

Запустите приложение GeoQuiz. Нажмите кнопку CHEAT!; на экране появится новая активность. Теперь нажмите кнопку BACK. Активность `CheatActivity` уничтожится, а приложение возвратится к `QuizActivity`.

Интенты явные и неявные

При создании объекта `Intent` с объектом `Context` и `Class` вы создаете *явный* (explicit) интент. Явные интенты используются для запуска активностей в приложениях.

Может показаться странным, что две активности внутри приложения должны взаимодействовать через компонент `ActivityManager`, находящийся вне приложения. Тем не менее такая схема позволяет активности одного приложения легко работать с активностью другого приложения.

Когда активность в вашем приложении должна запустить активность в другом приложении, вы создаете *неявный* (implicit) интент. Неявные интенты будут использоваться в главе 15.

Передача данных между активностями

Итак, в нашем приложении действуют активности `QuizActivity` и `CheatActivity`, и мы можем подумать о передаче данных между ними. На рис. 5.9 показано, какие данные будут передаваться между двумя активностями.

`QuizActivity` передает `CheatActivity` ответ на текущий вопрос при запуске `CheatActivity`.

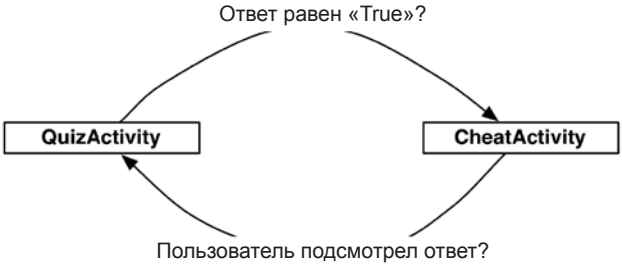


Рис. 5.9. Обмен данными между QuizActivity и CheatActivity

Когда пользователь нажимает кнопку BACK, чтобы вернуться к `QuizActivity`, экземпляр `CheatActivity` уничтожается. В последний момент он передает `QuizActivity` информацию о том, подсмотрел ли пользователь правильный ответ.

Начнем с передачи данных от `QuizActivity` к `CheatActivity`.

Дополнения интентов

Чтобы сообщить `CheatActivity` ответ на текущий вопрос, мы будем передавать значение

```
mQuestionBank[mCurrentIndex].isAnswerTrue();
```

Значение будет отправлено в виде *дополнения* (extra) объекта `Intent`, передаваемого `startActivity(Intent)`.

Дополнения представляют собой произвольные данные, которые вызывающая активность может передать вместе с интентом. Их можно рассматривать как аналоги аргументов конструктора, несмотря на то что вы не можете использовать произвольный конструктор с субклассом активности (Android создает экземпляры активностей и несет ответственность за их жизненный цикл). ОС направляет интент активности получателю, которая обращается к дополнению и извлекает данные (рис. 5.10).

Дополнение представляет собой пару «ключ-значение» наподобие той, которая использовалась для сохранения значения `mCurrentIndex` в `QuizActivity.onSaveInstanceState(Bundle)`.

Для включения дополнений в интент используется метод `Intent.putExtra(…)`, а точнее, метод

```
public Intent putExtra(String name, boolean value)
```

Метод `Intent.putExtra(…)` существует в нескольких разновидностях, но он всегда получает два аргумента. В первом аргументе всегда передается ключ `String`, а во втором — значение того или иного типа. Метод всегда возвращает сам объект `Intent`, так что при необходимости можно использовать цепочки из сцепленных вызовов.

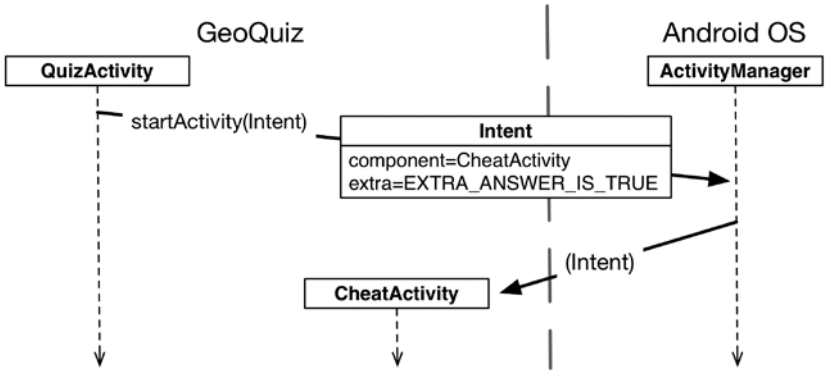


Рис. 5.10. Дополнения интентов: взаимодействие с другими активностями

Добавьте в CheatActivity.java ключ для дополнения.

Листинг 5.8. Добавление константы (CheatActivity.java)

```
public class CheatActivity extends AppCompatActivity {

    private static final String EXTRA_ANSWER_IS_TRUE =
        "com.bignerdranch.android.geoquiz.answer_is_true";
    ...
```

Активность может запускаться из нескольких разных мест, поэтому ключи для дополнений должны определяться в активностях, которые читают и используют их. Как видно из листинга 5.8, уточнение дополнения именем пакета предотвращает конфликты имен с дополнениями других приложений.

Теперь можно вернуться к `QuizActivity` и включить дополнение в интент, но это не лучший вариант. Нет никаких причин, по которым `QuizActivity` или любому другому коду приложения было бы нужно знать подробности реализации тех данных, которые `CheatActivity` ожидает получить в дополнении интента. Эту работу правильнее инкапсулировать в методе `newIntent(…)`.

Создайте этот метод в `CheatActivity`.

Листинг 5.9. Метод newIntent(...) (CheatActivity.java)

```
public class CheatActivity extends AppCompatActivity {

    private static final String EXTRA_ANSWER_IS_TRUE =
            "com.bignerdranch.android.geoquiz.answer_is_true";

    public static Intent newIntent(Context packageContext, boolean answerIsTrue) {
        Intent intent = new Intent(packageContext, CheatActivity.class);
        intent.putExtra(EXTRA_ANSWER_IS_TRUE, answerIsTrue);
        return intent;
    }
    ...
```

Этот статический метод позволяет создать объект `Intent`, настроенный дополнениями, необходимыми для `CheatActivity`. Логический аргумент `answerIsTrue` помещается в интент под закрытым именем с использованием константы `EXTRA_ANSWER_IS_TRUE`. Вскоре мы прочитаем это значение. Подобное использование метода `newIntent(…)` для субклассов активностей упрощает настройку интентов в других местах кода.

Раз уж речь зашла о других местах, используем этот метод в слушателе кнопки CHEAT!:

Листинг 5.10. Запуск CheatActivity.java с дополнением (QuizActivity.java)

```
mCheatButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Запуск CheatActivity
        Intent intent = new Intent(QuizActivity.this, CheatActivity.class);
        boolean answerIsTrue = mQuestionBank[mCurrentIndex].isAnswerTrue();
        Intent intent = CheatActivity.newIntent(QuizActivity.this, answerIsTrue);
        startActivity(intent);
    }
});
```

В нашей ситуации достаточно одного дополнения, но при необходимости можно добавить в `Intent` несколько дополнений. В этом случае добавьте в `newIntent(...)` дополнительные аргументы в соответствии с используемой схемой.

Для чтения значения из дополнения используется метод

```
public boolean getBooleanExtra(String name, boolean defaultValue)
```

Первый аргумент `getBooleanExtra(…)` содержит имя дополнения, а второй — ответ по умолчанию, если ключ не найден.

В `CheatActivity` прочитайте значение из дополнения в `onCreate(Bundle)` и сохраните его в переменной.

Листинг 5.11. Использование дополнения (CheatActivity.java)

```
public class CheatActivity extends AppCompatActivity {

    private static final String EXTRA_ANSWER_IS_TRUE =
            "com.bignerdranch.android.geoquiz.answer_is_true";

    private boolean mAnswerIsTrue;
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_cheat);

        mAnswerIsTrue = getIntent().getBooleanExtra(EXTRA_ANSWER_IS_TRUE, false);
    }
    ...
}
```

Обратите внимание: `Activity.getIntent()` всегда возвращает объект `Intent`, который был передан в `startActivity(Intent)`.

Наконец, добавьте в `CheatActivity` код, обеспечивающий использование прочитанного значения в виджете `TextView` ответа и кнопке SHOW ANSWER.

Листинг 5.12. Добавление функциональности подсматривания ответов (CheatActivity.java)

```
public class CheatActivity extends AppCompatActivity {
    ...
    private boolean mAnswerIsTrue;

    private TextView mAnswerTextView;
    private Button mShowAnswerButton;
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_cheat);

        mAnswerIsTrue = getIntent().getBooleanExtra(EXTRA_ANSWER_IS_TRUE, false);

        mAnswerTextView = (TextView) findViewById(R.id.answer_text_view);
        mShowAnswerButton = (Button) findViewById(R.id.show_answer_button);
        mShowAnswerButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (mAnswerIsTrue) {
                    mAnswerTextView.setText(R.string.true_button);
                } else {
                    mAnswerTextView.setText(R.string.false_button);
                }
            }
        });
    }
}
```

Код достаточно тривиален: мы задаем текст `TextView` при помощи метода `TextView.setText(int)`. Метод `TextView.setText(…)` существует в нескольких вариантах; здесь используется вариант, получающий идентификатор строкового ресурса.

Запустите приложение GeoQuiz. Нажмите кнопку CHEAT!, чтобы перейти к `CheatActivity`. Затем нажмите кнопку SHOW ANSWER, чтобы открыть ответ на текущий вопрос.

Получение результата от дочерней активности

В текущей версии приложения пользователь может беспрепятственно жульничать. Сейчас мы исправим этот недостаток; для этого `CheatActivity` будет сообщать `QuizActivity`, подсмотрел ли пользователь ответ.

Чтобы получить информацию от дочерней активности, вызовите следующий метод `Activity`:

```
public void startActivityForResult(Intent intent, int requestCode)
```

Первый параметр содержит тот же интент, что и прежде. Во втором параметре передается *код запроса* — определяемое пользователем целое число, которое передается дочерней активности, а затем принимается обратно родителем. Оно используется тогда, когда активность запускает сразу несколько типов дочерних активностей и ей необходимо определить, кто из потомков возвращает данные. `QuizActivity` запускает только один тип дочерней активности, но передача константы в коде запроса — полезная практика, с которой вы будете готовы к возможным будущим изменениям.

В классе `QuizActivity` измените слушателя `mCheatButton` и включите в него вызов `startActivityForResult(Intent, int)`.

Листинг 5.13. Вызов startActivityForResult(...) (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {

    private static final String TAG = "QuizActivity";
    private static final String KEY_INDEX = "index";
    private static final int REQUEST_CODE_CHEAT = 0;
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        mCheatButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Запуск CheatActivity
                boolean answerIsTrue = mQuestionBank[mCurrentIndex].isAnswerTrue();
                Intent intent = CheatActivity.newIntent(QuizActivity.this,
                                                        answerIsTrue);
                startActivity(intent);
                startActivityForResult(intent, REQUEST_CODE_CHEAT);
            }
        });
```

Передача результата

Существует два метода, которые могут вызываться в дочерней активности для возвращения данных родителю:

```
public final void setResult(int resultCode)
public final void setResult(int resultCode, Intent data)
```

Как правило, `resultCode` содержит одну из двух предопределенных констант: `Activity.RESULT_OK` или `Activity.RESULT_CANCELED`. (Также можно использовать другую константу `RESULT_FIRST_USER` как смещение при определении собственных кодов результатов.)

Назначение кода результата полезно в том случае, когда родитель должен выполнить разные действия в зависимости от того, как завершилась дочерняя активность.

Например, если в дочерней активности присутствуют кнопки OK и Cancel, то дочерняя активность может назначать разные коды результата в зависимости от того, какая кнопка была нажата. Родительская активность будет выбирать разные варианты действий в зависимости от полученного кода.

Вызов `setResult(…)` не является обязательным для дочерней активности. Если вам не нужно различать результаты или получать произвольные данные по интенту, просто разрешите ОС отправить код результата по умолчанию. Код результата всегда возвращается родителю, если дочерняя активность была запущена методом `startActivityForResult(…)`. Если метод `setResult(…)` не вызывался, то при нажатии пользователем кнопки BACK родитель получит код `Activity.RESULT_CANCELED`.

Возвращение интента

В нашей реализации дочерняя активность должна вернуть `QuizActivity` некоторые данные. Соответственно, мы создадим объект `Intent`, поместим в него дополнение, а затем вызовем `Activity.setResult(int, Intent)` для передачи этих данных `QuizActivity`.

Добавьте в `CheatActivity` константу для ключа дополнения и закрытый метод, который выполняет необходимую работу. Затем включите вызов этого метода в слушателя кнопки SHOW ANSWER.

Листинг 5.14. Назначение результата (CheatActivity.java)

```
public class CheatActivity extends AppCompatActivity {

    private static final String EXTRA_ANSWER_IS_TRUE =
        "com.bignerdranch.android.geoquiz.answer_is_true";
    private static final String EXTRA_ANSWER_SHOWN =
        "com.bignerdranch.android.geoquiz.answer_shown";
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        mShowAnswerButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (mAnswerIsTrue) {
                    mAnswerTextView.setText(R.string.true_button);
                } else {
                    mAnswerTextView.setText(R.string.false_button);
                }
                setAnswerShownResult(true);
            }
        });
    }
```

```
    private void setAnswerShownResult(boolean isAnswerShown) {
        Intent data = new Intent();
        data.putExtra(EXTRA_ANSWER_SHOWN, isAnswerShown);
        setResult(RESULT_OK, data);
    }
}
```

Когда пользователь нажимает кнопку SHOW ANSWER, `CheatActivity` упаковывает код результата и интент в вызове `setResult(int, Intent)`.

Затем, когда пользователь нажимает кнопку BACK для возвращения к `QuizActivity`, `ActivityManager` вызывает следующий метод родительской активности:

```
protected void onActivityResult(int requestCode, int resultCode, Intent data)
```

В параметрах передается исходный код запроса от `QuizActivity`, код результата и интент, переданный `setResult(int, Intent)`.

Последовательность взаимодействий изображена на рис. 5.11.

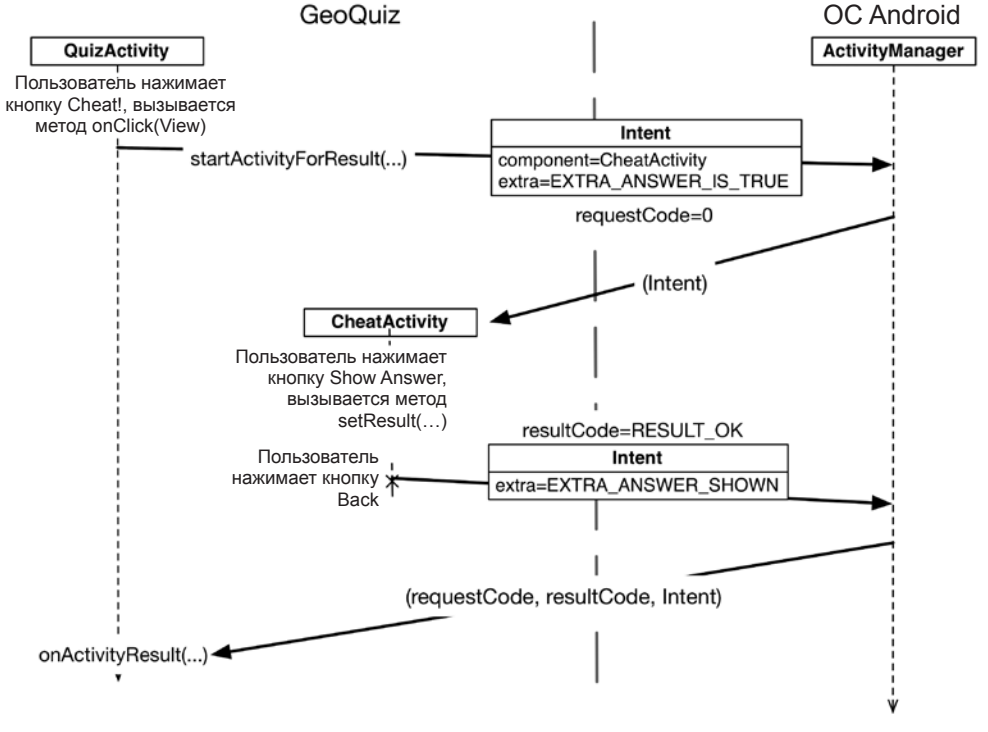


Рис. 5.11. Диаграмма последовательности для GeoQuiz

Остается последний шаг: переопределение `onActivityResult(int, int, Intent)` в `QuizActivity` для обработки результата. Но поскольку содержимое интента результата также относится к подробностям реализации `CheatActivity`, добавьте

еще один метод для упрощения декодирования дополнения к виду, который может использоваться `QuizActivity`.

Листинг 5.15. Декодирование интента результата (CheatActivity.java)

```
public static Intent newIntent(Context packageContext, boolean answerIsTrue) {
    Intent intent = new Intent(packageContext, CheatActivity.class);
    intent.putExtra(EXTRA_ANSWER_IS_TRUE, answerIsTrue);
    return intent;
}

public static boolean wasAnswerShown(Intent result) {
    return result.getBooleanExtra(EXTRA_ANSWER_SHOWN, false);
}

@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
}
```

Обработка результата

Добавьте в QuizActivity.java новую переменную для хранения значения, возвращаемого `CheatActivity`. Затем включите в переопределение `onActivityResult(…)` код его получения, проверки кода запроса и кода результата, чтобы быть уверенным в том, что они соответствуют ожиданиям. Как и в предыдущем случае, это полезная практика, которая упростит возможные будущие изменения.

Листинг 5.16. Реализация onactivityResult(…) (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {
    ...
    private int mCurrentIndex = 0;
    private boolean mIsCheater;
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (resultCode != Activity.RESULT_OK) {
            return;
        }
        if (requestCode == REQUEST_CODE_CHEAT) {
            if (data == null) {
                return;
            }
            mIsCheater = CheatActivity.wasAnswerShown(data);
        }
    }
    ...
}
```

Измените метод `checkAnswer(boolean)` в `QuizActivity`. Он должен проверять, подсмотрел ли пользователь ответ, и реагировать соответствующим образом.

Листинг 5.17. Изменение уведомления в зависимости от значения mIsCheater (QuizActivity.java)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
    mNextButton = (Button)findViewById(R.id.next_button);
    mNextButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
            mIsCheater = false;
            updateQuestion();
        }
    });
    ...
}
...
private void checkAnswer(boolean userPressedTrue) {
    boolean answerIsTrue = mQuestionBank[mCurrentIndex].isAnswerTrue();
    int messageResId = 0;
    if (mIsCheater) {
        messageResId = R.string.judgment_toast;
    } else {
        if (userPressedTrue == answerIsTrue) {
            messageResId = R.string.correct_toast;
        } else {
            messageResId = R.string.incorrect_toast;
        }
    }

    Toast.makeText(this, messageResId, Toast.LENGTH_SHORT)
        .show();
}
```

Запустите приложение GeoQuiz. Попробуйте подсмотреть ответ и понаблюдайте за тем, что произойдет.

Ваши активности с точки зрения Android

Давайте посмотрим, что происходит при переключении между активностями с точки зрения ОС. Прежде всего, когда вы щелкаете на приложении GeoQuiz в лаунчере, ОС запускает не приложение, а активность в приложении. А если говорить точнее, запускается *активность лаунчера* приложения. Для GeoQuiz активностью лаунчера является `QuizActivity`.

Когда мастер New Project создавал приложение GeoQuiz, класс `QuizActivity` был назначен активностью лаунчера по умолчанию. Статус активности лаунчера за-

дается в манифесте элементом `intent-filter` в объявлении `QuizActivity` (листинг 5.18).

Листинг 5.18. QuizActivity объявляется активностью лаунчера (AndroidManifest.xml)

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    ... >

    <application
        ... >

        <activity android:name=".QuizActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity android:name=".CheatActivity">
        </activity>
    </application>

</manifest>
```

Когда экземпляр `QuizActivity` окажется на экране, пользователь может нажать кнопку CHEAT!. При этом экземпляр `CheatActivity` запустится поверх `QuizActivity`. Эти активности образуют стек (рис. 5.12).

При нажатии кнопки BACK в `CheatActivity` этот экземпляр выводится из стека, а `QuizActivity` занимает свою позицию на вершине, как показано на рис. 5.12.

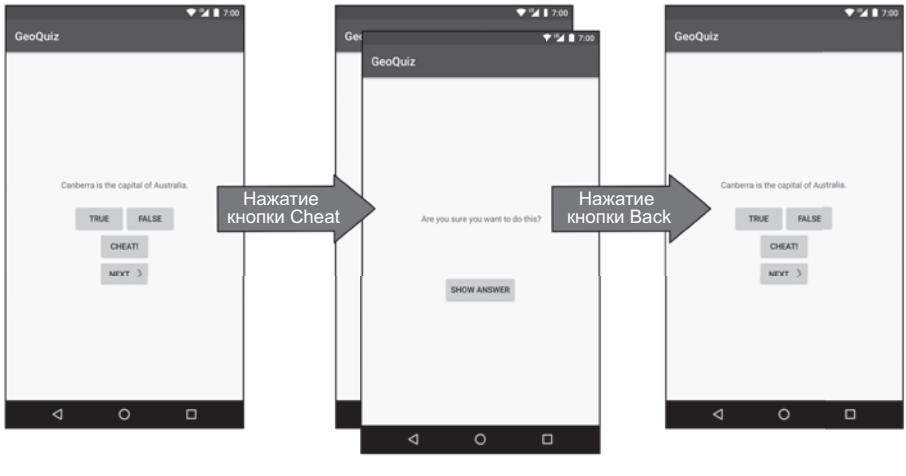


Рис. 5.12. Стек возврата GeoQuiz

Вызов `Activity.finish()` в `CheatActivity` также выводит `CheatActivity` из стека.

Если запустить GeoQuiz и нажать кнопку BACK в `QuizActivity`, то активность `QuizActivity` будет извлечена из стека, и вы вернетесь к последнему экрану, который просматривался перед запуском `GeoQuiz` (рис. 5.13).

Если вы запустили GeoQuiz из лаунчера, то нажатие кнопки BACK из `QuizActivity` вернет вас обратно (рис. 5.14).

Нажатие BACK в запущенном лаунчере вернет вас к экрану, который был открыт перед его запуском.

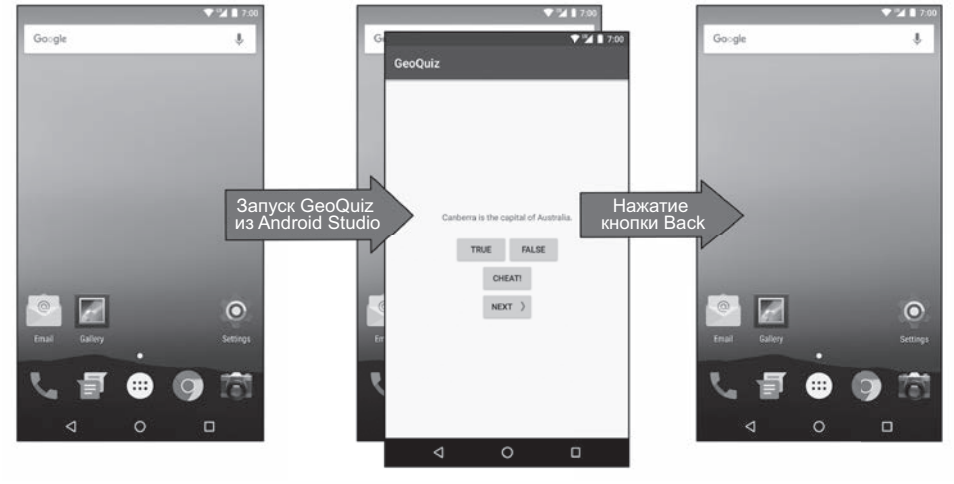


Рис. 5.13. Экран Home

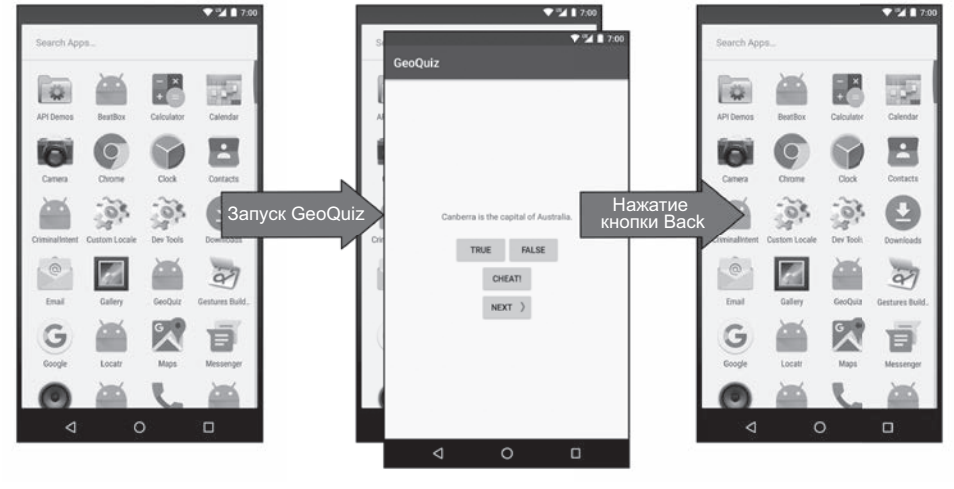


Рис. 5.14. Запуск GeoQuiz из лаунчера

Мы видим, что `ActivityManager` поддерживает *стек возврата* (back stack) и что этот стек не ограничивается активностями вашего приложения. Он совместно используется активностями всех приложений; это одна из причин, по которым `ActivityManager` участвует в запуске активностей и находится под управлением ОС, а не вашего приложения. Стек представляет использование ОС и устройства в целом, а не использование одного приложения.

(Как насчет кнопки Up? О том, как реализовать и настроить эту кнопку, будет рассказано в главе 13.)

Упражнение. Лазейка для мошенников

Мошенники никогда не выигрывают... Если, конечно, им не удастся обойти вашу защиту от мошенничества. А скорее всего, они так и сделают — именно потому, что они мошенники.

Приложение GeoQuiz содержит ряд «лазеек», которыми могут воспользоваться мошенники. В этом упражнении вы должны устранить эти лазейки. Основные дефекты приложения перечислены ниже по возрастанию сложности, от самых простых к самым сложным:

- Подсмотрев ответ, пользователь может повернуть `CheatActivity`, чтобы сбросить результат.
- После возвращения пользователь может повернуть `QuizActivity`, чтобы сбросить флаг `mIsCheater`.
- Пользователь может нажимать кнопку NEXT до тех пор, пока вопрос, ответ на который был подсмотрен, снова не появится на экране.

Удачи!

6 Версии Android SDK и совместимость

Теперь, после «боевого крещения» с приложением GeoQuiz, пора поближе познакомиться с разными версиями Android. Информация этой главы пригодится вам в следующих главах книги, когда мы займемся разработкой более сложных и реалистичных приложений.

Версии Android SDK

В табл. 6.1 перечислены версии SDK, соответствующие версии прошивки Android, и процент устройств, использующих их, по состоянию на декабрь 2016 года.

Таблица 6.1. Уровни API Android, версии прошивки и процент устройств

Уровень API	Кодовое название	Версия прошивки устройства	% использующих устройств
24	Nougat	7.0	0,4
23	Marshmallow	6.0	26,3
22	Lollipop	5.1	23,2
21		5.0	10,8
19	KitKat	4.4	24,0
18	Jelly Bean	4.3	1,9
17		4.2	6,4
16		4.1	4,5
15	Ice Cream Sandwich (ICS)	4.0.3, 4.0.4	1,2
10	Gingerbread	2.3.3–2.3.7	1,2
8	Froyo	2.2.x	0,1

(Версии Android, используемые менее чем на 0,1% устройств, в таблице не приводятся.)

За каждым выпуском с кодовым названием следуют инкрементные выпуски. Например, платформа Android 4.0 (API уровня 14) Ice Cream Sandwich была почти

немедленно заменена инкрементными выпусками, которые в конечном итоге привели к появлению Android 4.0.3 и 4.0.4 (API уровня 15).

Конечно, проценты из табл. 6.1 постоянно изменяются, но в них проявляется важная закономерность: устройства Android со старыми версиями не подвергаются немедленному обновлению или замене при появлении новой версии. По состоянию на декабрь 2016 года более 15% устройств все еще работали под управлением Jelly Bean или более ранней версии. Версия Android 4.3 (последнее обновление Jelly Bean) была выпущена в октябре 2013 года.

(Обновленные данные из таблицы доступны по адресу *developer.android.com/about/dashboards/index.html*.)

Почему на многих устройствах продолжают работать старые версии Android? В основном из-за острой конкуренции между производителями устройств Android и операторами сотовой связи. Операторы стремятся иметь возможности и телефоны, которых нет у других сетей. Производители устройств тоже испытывают давление — все их телефоны базируются на одной ОС, но им нужно выделяться на фоне конкурентов. Сочетание этого давления со стороны рынка и операторов сотовой связи привело к появлению многочисленных устройств со специализированными модификациями Android.

Устройство со специализированной версией Android не сможет перейти на новую версию, выпущенную Google. Вместо этого ему придется ждать совместимого «фирменного» обновления, которое может выйти через несколько месяцев после выпуска версии Google... А может вообще не выйти — производители часто предпочитают расходовать ресурсы на новые устройства, а не на обновление старых устройств.

Совместимость и программирование Android

Из-за задержек обновления в сочетании с регулярным выпуском новых версий совместимость становится важной проблемой в программировании Android. Чтобы привлечь широкую аудиторию, разработчики Android должны создавать приложения, которые хорошо работают на устройствах с разными версиями Android: KitKat, Lollipop, Marshmallow, Nougat и более современными версиями, а также на устройствах различных форм-факторов.

Поддержка разных размеров не столь сложна, как может показаться. Смартфоны выпускаются с экранами различных размеров, но система макетов Android хорошо приспосабливается к ним. С планшетами дело обстоит сложнее, но в этом случае на помощь приходят квалификаторы конфигураций (как будет показано в главе 17). Впрочем, для устройств Android TV и Android Wear (также работающих под управлением Android) различия в пользовательском интерфейсе настолько серьезны, что разработчику приходится переосмысливать организацию взаимодействия с пользователем и дизайн приложения.

Разумный минимум

Самой старой версией Android, поддерживаемой в примерах книги, является API уровня 19 (Kit Kat). Также встречаются упоминания более старых версий, но

мы сосредоточимся на тех версиях, которые считаем современными (API уровня 19+). Доля устройств с версиями Gingerbread, Ice Cream Sandwich и Jelly Bean падает от месяца к месяцу, и объем работы, необходимой для поддержки старых версий, перевешивает пользу от них.

Инкрементные выпуски не создают особых проблем с обратной совместимостью. Изменение основной версии — совсем другое дело. Работа, необходимая для поддержки только устройств 5.x, не так уж велика, но, если вам также необходимо поддерживать устройства 4.x, придется потратить время на анализ различий между версиями. К счастью, компания Google предоставила специальные библиотеки, которые упростят задачу. Вы узнаете о них в следующих главах.

Выпуск Honeycomb (Android 3.0) создал немало проблем для Android-разработчиков. Он стал поворотным моментом в развитии платформы, ознаменовавшим появление нового пользовательского интерфейса и новых архитектурных компонентов. Версия Honeycomb предназначалась только для планшетов, так что новые разработки получили массовое распространение лишь с выходом Ice Cream Sandwich. Версии, выпущенные после этого, были менее революционными.

Система Android помогает обеспечить обратную совместимость. Также существуют сторонние библиотеки, которые помогают в этом. С другой стороны, поддержание совместимости усложняет изучение программирования Android.

При создании проекта GeoQuiz в мастере нового приложения вы выбирали минимальную версию SDK (рис. 6.1). (Учтите, что в Android термины «версия SDK» и «уровень API» являются синонимами.)

Рис. 6.1. Еще не забыли?

Кроме минимальной поддерживаемой версии также можно задать *целевую версию* (target version) и *версию для построения* (build version). Давайте разберемся, чем различаются эти версии и как изменить их.

Все эти свойства задаются в файле build.gradle в модуле app. Версия для построения задается исключительно в этом файле. Минимальная и целевая версии SDK задаются в файле build.gradle, но используются для замены или инициализации значений из файла AndroidManifest.xml.

Откройте файл build.gradle, находящийся в модуле app. Обратите внимание на значения `compileSdkVersion`, `minSdkVersion` и `targetSdkVersion`.

Листинг 6.1. Конфигурация построения (app/build.gradle)

```
compileSdkVersion 25
buildToolsVersion "25.0.0"

defaultConfig {
    applicationId "com.bignerdranch.android.geoquiz"
    minSdkVersion 19
    targetSdkVersion 25
    ...
}
```

Минимальная версия SDK

Значение `minSdkVersion` определяет нижнюю границу, за которой ОС отказывается устанавливать приложение.

Выбирая API уровня 19 (KitKat), вы разрешаете Android устанавливать GeoQuiz на устройствах с версией KitKat и выше. Android откажется устанавливать GeoQuiz на устройстве, допустим, с системой Jelly Bean.

Снова обращаясь к табл. 6.1, мы видим, почему API 19 является хорошим выбором для минимальной версии SDK: в этом случае приложение можно будет устанавливать более чем на 80% используемых устройств.

Целевая версия SDK

Значение `targetSdkVersion` сообщает Android, для какого уровня API *проектировалось* ваше приложение. Чаще всего в этом поле указывается новейшая версия Android.

Когда следует понижать целевую версию SDK? Новые выпуски SDK могут изменять внешний вид вашего приложения на устройстве и даже поведение ОС «за кулисами». Если ваше приложение уже спроектировано, убедитесь в том, что в новых версиях оно работает так, как должно. За информацией о возможных проблемах обращайтесь к документации по адресу *developer.android.com/reference/android/os/Build.VERSION_CODES.html*. Далее либо измените свое приложение, чтобы оно работало с новым поведением, либо понизьте целевую версию SDK.

Понижение целевой версии SDK гарантирует, что приложение будет работать с внешним видом и поведением целевой версии, в которой оно хорошо работало. Этот параметр существует для обеспечения совместимости с новыми версиями Android, так как все изменения последующих версий игнорируются до увеличения `targetSdkVersion`.

Версия SDK для компиляции

Последний параметр SDK обозначен в листинге 6.1 именем `compileSdkVersion`. В файле AndroidManifest.xml этот параметр отсутствует. Если минимальная и целевая версии SDK помещаются в манифест и сообщаются ОС, версия SDK для построения относится к закрытой информации, известной только вам и компилятору.

Функциональность Android используется через классы и методы SDK. Версия SDK, используемая для построения, также называемая *целью построения* (build target), указывает, какая версия должна использоваться при построении вашего кода. Когда Android Studio ищет классы и методы, на которые вы ссылаетесь в директивах импорта, версия SDK для построения определяет, в какой версии SDK будет осуществляться поиск.

Лучшим вариантом цели построения является новейший уровень API (на момент написания книги 25, Nougat). Тем не менее при необходимости цель построения существующего приложения можно изменить, например, при выпуске очередной версии Android, чтобы вы могли использовать новые методы и классы, появившиеся в этой версии.

Все эти параметры — минимальную версию SDK, целевую версию SDK, версию SDK для построения — можно изменить в файле build.gradle, однако следует помнить, что изменения в этом файле проявятся только после синхронизации проекта с измененными файлами. Выберите команду Tools ▸ Android ▸ Sync Project with Gradle Files. Проект строится заново с обновленными параметрами.

Безопасное добавление кода для более поздних версий API

Различия между минимальной версией SDK и версией SDK для построения создают проблемы совместимости, которые необходимо учитывать. Например, что произойдет в GeoQuiz при выполнении кода, рассчитанного на версию SDK после минимальной версии KitKat (API уровня 19)? Если установить такое приложение и запустить его на устройстве с KitKat, произойдет сбой.

Раньше тестирование в подобных ситуациях было сущим кошмаром. Однако благодаря совершенствованию Android Lint проблемы, порожденные вызовом нового кода на старых устройствах, теперь успешно обнаруживаются. Если вы используете код версии, превышающей минимальную версию SDK, Android Lint сообщит об ошибке построения.

Весь код текущей версии GeoQuiz совместим с API уровня 19 и ниже. Добавим код API уровня 21 (Lollipop) и посмотрим, что произойдет.

Откройте файл CheatActivity.java. Включите в метод `OnClickListener` кнопки SHOW ANSWER следующий фрагмент кода для создания круговой анимации на время сокрытия кнопки:

Листинг 6.2. Добавление кода анимации (CheatActivity.java)

```
mShowAnswerButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (mAnswerIsTrue) {
            mAnswerTextView.setText(R.string.true_button);
        } else {
            mAnswerTextView.setText(R.string.false_button);
        }
        setAnswerShownResult(true);

        int cx = mShowAnswerButton.getWidth() / 2;
        int cy = mShowAnswerButton.getHeight() / 2;
        float radius = mShowAnswerButton.getWidth();
        Animator anim = ViewAnimationUtils
                .createCircularReveal(mShowAnswerButton, cx, cy, radius, 0);
        anim.addListener(new AnimatorListenerAdapter() {
            @Override
            public void onAnimationEnd(Animator animation) {
                super.onAnimationEnd(animation);
                mShowAnswerButton.setVisibility(View.INVISIBLE);
            }
        });
        anim.start();
    }
});
```

Метод `createCircularReveal` создает объект `Animator` по нескольким параметрам. Сначала вы задаете объект `View`, который будет скрываться или отображаться в ходе анимации. Затем указывается центральная позиция, начальный и конечный радиус анимации. Кнопка SHOW ANSWER скрывается, поэтому радиус изменяется в диапазоне от ширины кнопки до 0.

Перед запуском только что созданной анимации назначается слушатель, который помогает узнать о завершении анимации. После завершения выводится ответ, а кнопка скрывается.

Наконец, анимация запускается, и начинается круговая анимация. Тема анимации подробно рассматривается в главе 32.

Класс `ViewAnimationUtils` и его метод `createCircularReveal` были добавлены в Android SDK в API level 21, поэтому при попытке выполнения этого кода на устройстве с более низкой версией произойдет сбой.

После ввода кода в листинге 6.2 Android Lint немедленно выдает предупреждение о том, что выполнение кода небезопасно при вашей минимальной версии SDK. Если предупреждение не появилось, вы можете вручную активизировать Lint ко-

мандой Analyze ▸ Inspect Code.... Так как для построения используется API level 21, у компилятора не возникнет проблем с этим кодом. С другой стороны, Android Lint знает минимальную версию SDK и будет протестовать.

Сообщения об ошибках будут выглядеть примерно так: «Call requires API level 21 (current min is 19)». Запуск кода с этим предупреждением возможен, но Lint знает, что это небезопасно.

Как избавиться от ошибок? Первый способ — поднять минимальную версию SDK до 21. Однако тем самым вы не столько решаете проблему совместимости, сколько обходите ее. Если ваше приложение не может устанавливаться на устройствах с API уровня 19 и более старых устройствах, то проблема совместимости исчезает.

Другое, более правильное решение — заключить код более высокого уровня API в условную конструкцию, которая проверяет версию Android на устройстве.

Листинг 6.3. Предварительная проверка версии Android на устройстве (CheatActivity.java)

```
mShowAnswerButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (mAnswerIsTrue) {
            mAnswerTextView.setText(R.string.true_button);
        } else {
            mAnswerTextView.setText(R.string.false_button);
        }
        setAnswerShownResult(true);

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
            int cx = mShowAnswerButton.getWidth() / 2;
            int cy = mShowAnswerButton.getHeight() / 2;
            float radius = mShowAnswerButton.getWidth();
            Animator anim = ViewAnimationUtils
                    .createCircularReveal(mShowAnswerButton, cx, cy, radius, 0);
            anim.addListener(new AnimatorListenerAdapter() {
                @Override
                public void onAnimationEnd(Animator animation) {
                    super.onAnimationEnd(animation);
                    mShowAnswerButton.setVisibility(View.INVISIBLE);
                }
            });
            anim.start();
        } else {
            mShowAnswerButton.setVisibility(View.INVISIBLE);
        }
    }
});
```

Константа `Build.VERSION.SDK_INT` определяет версию Android на устройстве. Она сравнивается с константой, соответствующей Lollipop. (Коды версий доступны по адресу *developer.android.com/reference/android/os/Build.VERSION_CODES.html*.)

Теперь код анимации будет выполняться только в том случае, если приложение работает на устройстве с API уровня 21 и выше. Код стал безопасным для API уровня 19, Android Lint не на что жаловаться.

Запустите GeoQuiz на устройстве с версией Lollipop и выше и проверьте, как работает новая анимация.

Также можно запустить GeoQuiz на устройстве Jelly Bean или KitKat (виртуальном или физическом). Анимация в этом случае не отображается, но вы можете убедиться в том, что приложение работает нормально.

Документация разработчика Android

Ошибки Android Lint сообщают, к какому уровню API относится несовместимый код. Однако вы также можете узнать, к какому уровню API относятся конкретные классы и методы, — эта информация содержится в документации разработчика Android.

Постарайтесь поскорее научиться работать с документацией. Информация Android SDK слишком обширна, чтобы держать ее в голове, а с учетом регулярного появления новых версий разработчик должен знать, что нового появилось, и уметь пользоваться этими новшествами.

Документация разработчика Android — превосходный и обширный источник информации. Ее главная страница находится по адресу *developer.android.com*. Документация делится на три части: проектирование (Design), разработка (Develop) и распространение (Distribute). В разделе проектирования описаны паттерны и принципы проектирования пользовательского интерфейса приложений. В разделе разработки содержится основная документация и учебные материалы. В разделе распространения объясняется, как готовить и публиковать приложения в Google Play или через механизм открытого распространения. Все это стоит просмотреть, когда у вас появится возможность.

Раздел разработки дополнительно разбит на семь секций.

Training	Учебные модули для начинающих и опытных разработчиков, включая загружаемый код примеров
API Guides	Тематические описания компонентов приложений, функций и полезных приемов
Reference	Гипертекстовая документация по всем классам, методам, интерфейсам, константам атрибутов и т. д. в SDK, с возможностью поиска
Samples	Примеры кода, демонстрирующие использование API
Android Studio	Информация об интегрированной среде Android Studio
Android NDK	Описания и ссылки на инструментарий NDK (Native Development Kit), позволяющий писать код на C и C++
Google Services	Информация о собственных API компании Google, включая Google Maps и Google Cloud Messaging

Для работы с документацией не обязательно иметь доступ к Интернету. Откройте в файловой системе каталог, в который были загружены SDK; в нем расположен каталог docs с полной документацией.

Чтобы определить, к какому уровню API относится `ViewAnimationUtils`, проведите поиск этого метода при помощи строки поиска в правом верхнем углу браузера. Вы получите результаты из нескольких категорий. Вам нужен результат из справочной секции; чтобы отфильтровать информацию, воспользуйтесь фильтром в левой части окна.

Выберите первый результат; откроется справочная страница с описанием класса `ViewAnimationUtils` (рис. 6.2). В верхней части страницы находятся ссылки на разные секции.

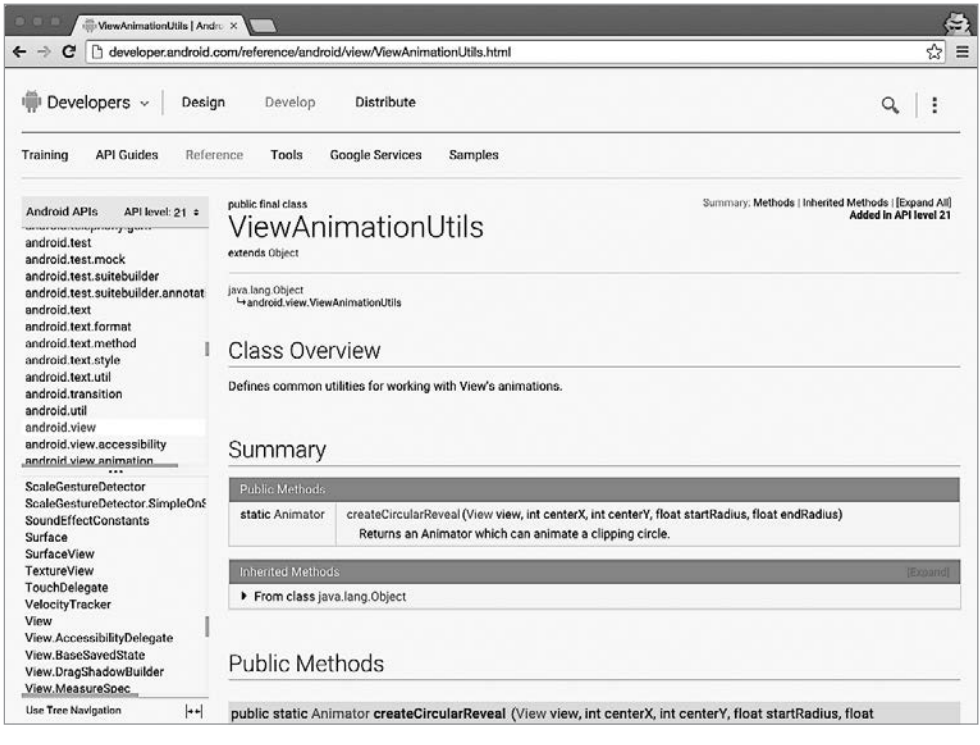


Рис. 6.2. Страница со справочным описанием ViewAnimationUtils

Прокрутите список, найдите метод `createCircularReveal(…)` и щелкните на имени метода, чтобы увидеть описание. Справа от сигнатуры метода видно, что метод `createCircularReveal(…)` появился в API уровня 21.

Если вы хотите узнать, какие методы `ViewAnimationUtils` доступны, скажем, в API уровня 19, отфильтруйте справочник по уровню API. В левой части страницы, где классы индексируются по пакету, найдите категорию API level:21. Щелкните на близлежащем элементе управления и выберите в списке 19. Все, что появилось

в Android после API уровня 19, выделяется в списке серым цветом. В данном случае класс `ViewAnimationUtils` появился в API уровня 21, поэтому вы увидите предупреждение о том, что весь класс полностью недоступен в API уровня 19.

Фильтр уровня API приносит намного больше пользы классам, доступным на используемом вами уровне API. Найдите в документации страницу класса `Activity`. Верните в фильтре уровня API значение `19`; обратите внимание, как много методов появилось с момента выхода API 16: например, метод `onEnterAnimationComplete`, включенный в SDK в Lollipop, позволяет создавать интересные переходы между активностями.

Продолжая чтение книги, старайтесь почаще возвращаться к документации. Она наверняка понадобится вам для выполнения упражнений, однако не пренебрегайте самостоятельными изысканиями каждый раз, когда вам захочется побольше узнать о некотором классе, методе и т. д. Создатели Android постоянно обновляют и совершенствуют свою документацию, и вы всегда узнаете из нее что-то новое.

Упражнение. Вывод версии построения

Добавьте в макет GeoQuiz виджет `TextView` для вывода уровня API устройства, на котором работает программа. На рис. 6.3 показано, как должен выглядеть результат.

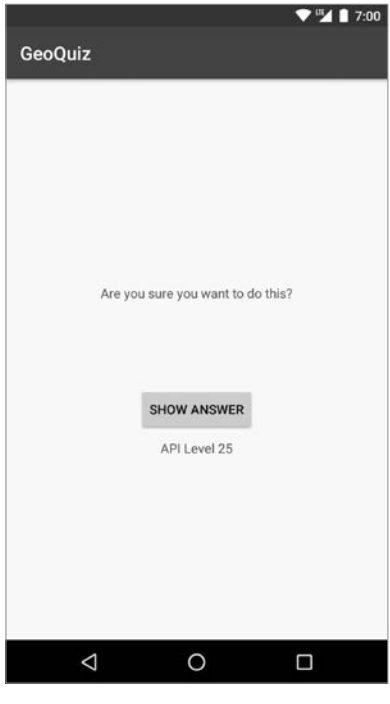


Рис. 6.3. Результат упражнения

Задать текст `TextView` в макете не получится, потому что версия построения устройства не известна до момента выполнения. Найдите метод `TextView` для задания текста в справочной странице `TextView` в документации Android. Вам нужен метод, получающий один аргумент — строку (или `CharSequence`).

Для настройки размера и гарнитуры текста используйте атрибуты XML, перечисленные в описании `TextView`.

Упражнение. Ограничение подсказок

Ограничьте пользователя тремя подсказками. Храните информацию о том, сколько раз пользователь подсматривал ответ, и выводите количество оставшихся подсказок под кнопкой. Если ни одной подсказки не осталось, то кнопка получения подсказки блокируется.

7 UI-фрагменты и FragmentManager

В этой главе мы начнем строить приложение `CriminalIntent`. Оно предназначено для хранения информации об «офисных преступлениях»: грязной посуде, оставленной в раковине, или пустом лотке общего принтера после печати документов.

В приложении `CriminalIntent` пользователь создает запись о преступлении с заголовком, датой и фотографией. Также можно выбрать подозреваемого в списке контактов и отправить жалобу по электронной почте, опубликовать ее в Twitter, Facebook или другом приложении. Сообщив о преступлении, пользователь освобождается от негатива и может сосредоточиться на текущей задаче.

`CriminalIntent` — сложное приложение, для построения которого нам понадобится целых 13 глав. В нем используется интерфейс типа «список/детализация»: на главном экране выводится список зарегистрированных преступлений. Пользователь может добавить новое или выбрать существующее преступление для просмотра и редактирования информации (рис. 7.1).

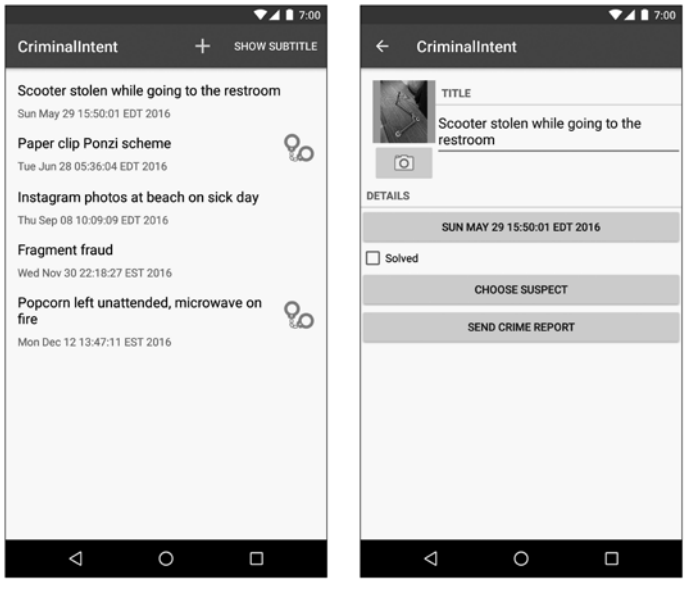


Рис. 7.1. Приложение CriminalIntent

Гибкость пользовательского интерфейса

Разумно предположить, что приложение типа «список/детализация» состоит из двух активностей: для управления списком и для управления детализированным представлением. Щелчок на преступлении в списке запускает экземпляр детализированной активности. Нажатие кнопки BACK уничтожает активность детализации и возвращает на экран список, в котором можно выбрать другое преступление.

Такая архитектура работает, но что делать, если вам потребуется более сложная схема представления информации и навигации между экранами?

- Допустим, пользователь запустил приложение `CriminalIntent` на планшете. Экраны планшетов и некоторых больших телефонов достаточно велики для одновременного отображения списка и детализации, по крайней мере в альбомной ориентации (рис. 7.2).

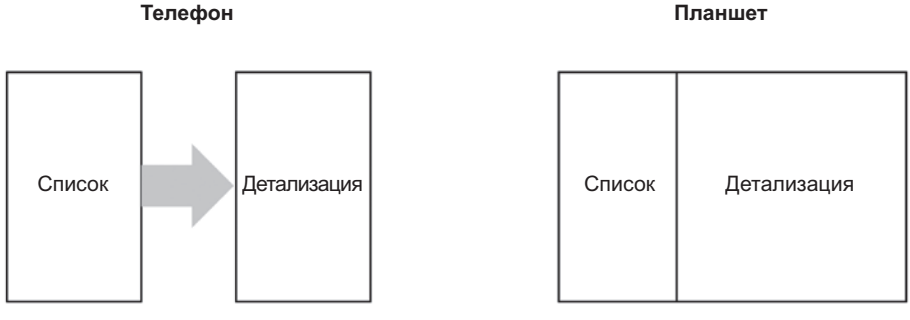


Рис. 7.2. Идеальный интерфейс «список/детализация» для телефонов и планшетов

- Пользователь просматривает описание преступления на телефоне и хочет увидеть следующее преступление в списке. Было бы удобно, если бы пользователь мог провести пальцем по экрану, чтобы перейти к следующему преступлению без возвращения к списку. Каждый жест прокрутки должен обновлять детализированное представление информацией о следующем преступлении.

Эти сценарии объединяет гибкость пользовательского интерфейса: возможность формирования и изменения представления активности во время выполнения в зависимости от того, что нужно пользователю или устройству.

Подобная гибкость в активности не предусмотрена. Представления активности могут изменяться во время выполнения, но код, управляющий этими изменениями, должен находиться в представлении. В результате активность тесно связывается с конкретным экраном, с которым работает пользователь.

Знакомство с фрагментами

Закон Android нельзя нарушить, но можно обойти, передав управление пользовательским интерфейсом приложения от активности одному или нескольким *фрагментам* (fragments).

Фрагмент представляет собой объект контроллера, которому активность может доверить выполнение операций. Чаще всего такой операцией является управление пользовательским интерфейсом — целым экраном или его частью.

Фрагмент, управляющий пользовательским интерфейсом, называется *UI-фрагментом*. UI-фрагмент имеет собственное представление, которое заполняется на основании файла макета. Представление фрагмента содержит элементы пользовательского интерфейса, с которыми будет взаимодействовать пользователь.

Представление активности содержит место, в которое вставляется представление фрагмента. Более того, хотя в нашем примере активность управляет одним фрагментом, она может содержать несколько мест для представлений нескольких фрагментов.

Фрагменты, связанные с активностью, могут использоваться для формирования и изменения экрана в соответствии с потребностями приложения и пользователей. Представление активности формально остается неизменным на протяжении жизненного цикла, и законы Android не нарушаются.

Давайте посмотрим, как эта схема работает в приложении «список/детализация». Представление активности строится из фрагмента списка и фрагмента детализации. Представление детализации содержит подробную информацию о выбранном элементе списка.

При выборе другого элемента на экране появляется новое детализированное представление. С фрагментами эта задача решается легко; активность заменяет фрагмент детализации другим фрагментом детализации (рис. 7.3). Это существенное изменение представления происходит без уничтожения активности.

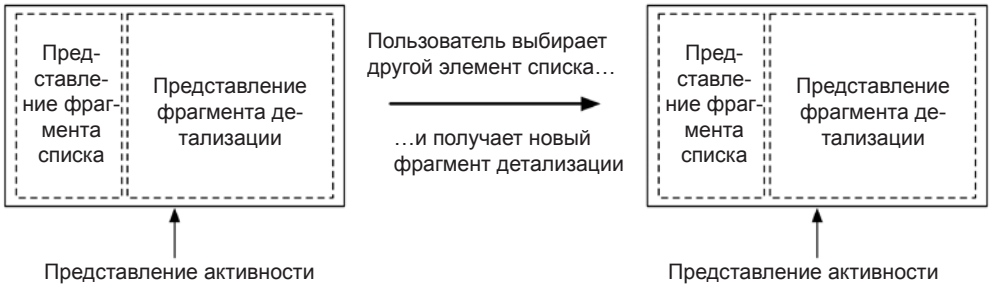


Рис. 7.3. Переключение фрагмента детализации

Применение UI-фрагментов позволяет разделить пользовательский интерфейс вашего приложения на структурные блоки, а это полезно не только для приложений «список/детализация». Работа с отдельными блоками упрощает построение интерфейсов со вкладками, анимированных боковых панелей и многого другого.

Впрочем, за такую гибкость пользовательского интерфейса приходится платить повышением сложности, увеличением количества «подвижных частей», увеличением объема кода. Выгода от использования фрагментов станет очевидной в главах 11 и 17, зато разбираться со сложностью придется прямо сейчас.

Начало работы над CriminalIntent

В этой главе мы возьмемся за представление детализации приложения CriminalIntent. На рис. 7.4 показано, как будет выглядеть приложение CriminalIntent к концу главы.

Рис. 7.4. Приложение CriminalIntent к концу главы

Экраном, показанным на рис. 7.4, будет управлять UI-фрагмент с именем `CrimeFragment`. *Хостом* (host) экземпляра `CrimeFragment` является активность с именем `CrimeActivity`.

Пока считайте, что хост предоставляет позицию в иерархии представлений, в которой фрагмент может разместить свое представление (рис. 7.5). Фрагмент не может вывести представление на экран сам по себе. Его представление отображается только при размещении в иерархии активности.

Проект CriminalIntent будет большим; диаграмма объектов поможет понять логику его работы. На рис. 7.6 изображена общая структура CriminalIntent. Запоминать все объекты и связи между ними не обязательно, но, прежде чем трогаться с места, полезно хотя бы в общих чертах понимать, куда вы направляетесь.

Мы видим, что класс `CrimeFragment` делает примерно то же, что в GeoQuiz делали активности: он создает пользовательский интерфейс и управляет им, а также обеспечивает взаимодействие с объектами модели.

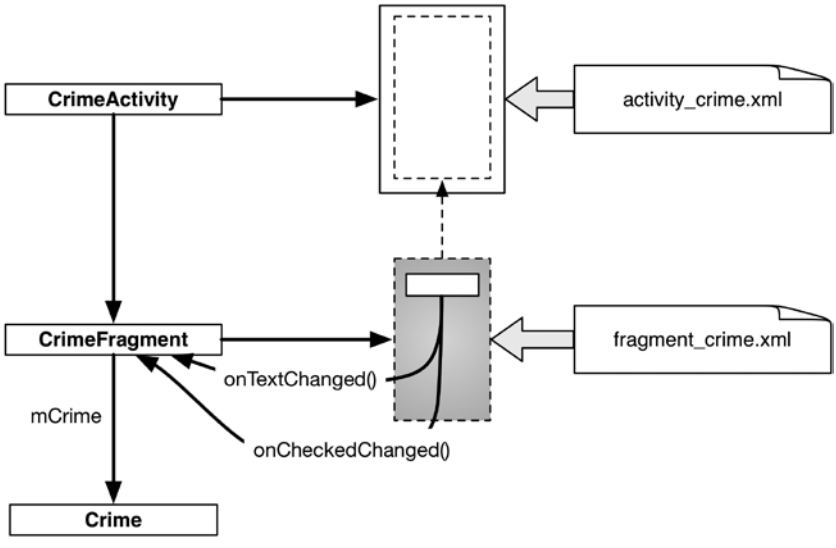


Рис. 7.5. CrimeActivity как хост CrimeFragment

Мы напишем три класса, изображенных на рис. 7.6: `Crime`, `CrimeFragment` и `CrimeActivity`.

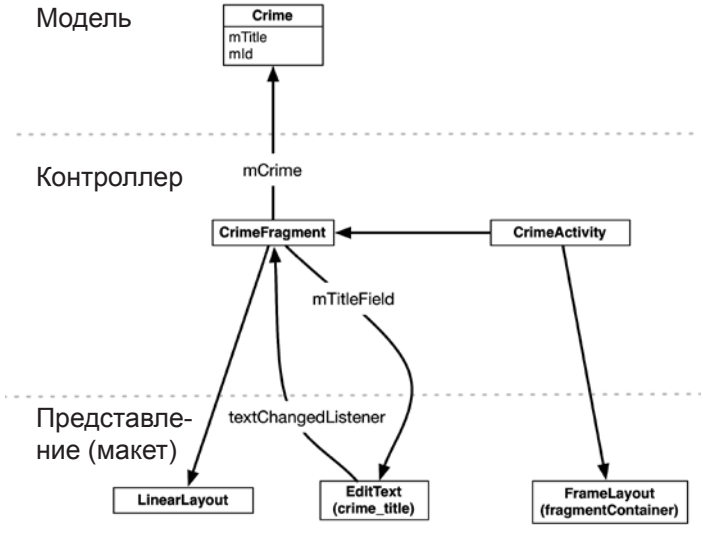


Рис. 7.6. Диаграмма объектов CriminalIntent (для этой главы)

Экземпляр `Crime` представляет одно офисное преступление. В этой главе описание преступления будет состоять только из заголовка и идентификатора. Заголовок содержит содержательный текст (например, «Свалка токсичных отходов

в раковине» или «Кто-то украл мой йогурт!»), а идентификатор однозначно идентифицирует экземпляр `Crime`.

В этой главе мы для простоты будем использовать один экземпляр `Crime`. В класс `CrimeFragment` включается поле (`mCrime`) для хранения этого отдельного инцидента.

Представление `CrimeActivity` состоит из элемента `FrameLayout`, определяющего место, в котором будет отображаться представление `CrimeFragment`.

Представление `CrimeFragment` состоит из элемента `LinearLayout` с несколькими дочерними представлениями, включая `EditText`, `Button` и `CheckBox`. `CrimeFragment` определяет поле для каждого из этих представлений и назначает для него слушателя, обновляющего уровень модели при изменении текста.

Создание нового проекта

Но довольно разговоров; пора построить новое приложение. Создайте новое приложение Android (File ▸ New Project...). Введите имя приложения CriminalIntent и имя пакета *com.bignerdranch.android.criminalintent*, как показано на рис. 7.7.

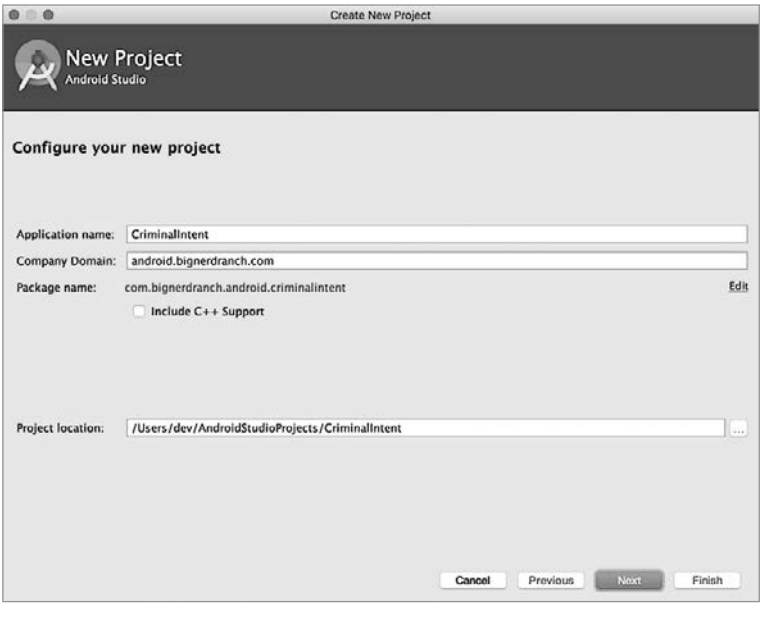


Рис. 7.7. Создание приложения CriminalIntent

Щелкните на кнопке Next и задайте минимальный уровень SDK: API 19: Android 4.4. Также проследите за тем, чтобы для типов приложений был выбран только тип приложения Phone and Tablet.

Снова щелкните на кнопке Next, чтобы выбрать разновидность добавляемой активности. Выберите пустую активность (Empty Activity) и продолжите работу с мастером.

На последнем шаге мастера введите имя активности CrimeActivity и щелкните на кнопке Finish (рис. 7.8).

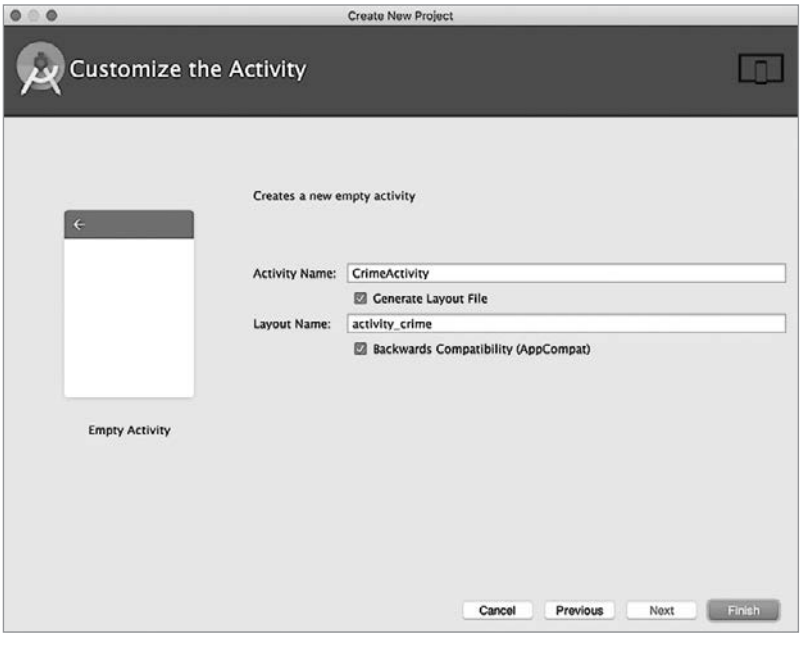


Рис. 7.8. Создание CrimeActivity

Два типа фрагментов

Фрагменты появились в API уровня 11 вместе с первыми планшетами на базе Android и неожиданной потребностью в гибкости пользовательского интерфейса. Вы должны выбрать используемую реализацию фрагментов: *платформенные фрагменты* (native fragments) или *фрагменты из библиотеки поддержки* (support fragments).

Платформенные реализации фрагментов встраиваются в устройство, на котором запускается приложение. Если вы поддерживаете много разных версий Android, эти версии могут использовать разные реализации фрагментов (например, в какой-то версии может быть исправлена ошибка, присутствовавшая в предыдущих версиях). Другая реализация фрагментов встраивается в библиотеку, которая включается в ваше приложение. Это означает, что все устройства, на которых будет запускаться ваше приложение, будут зависеть от единой реализации фрагментов независимо от версии Android.

В `CriminalIntent` будет использоваться реализация фрагментов из библиотеки поддержки. Подробное обоснование этого решения приводится в конце главы, в разделе «Для любознательных: почему фрагменты из библиотеки поддержки лучше».

Добавление зависимостей в Android Studio

Мы будем использовать реализацию фрагментов, встроенную в библиотеку AppCompat — одну из многочисленных библиотек совместимости Google, которые будут встречаться в этой книге. Библиотека AppCompat более подробно рассматривается в главе 13.

Чтобы использовать библиотеку AppCompat, необходимо включить ее в состав зависимостей проекта. Ваш проект содержит два файла build.gradle: один для проекта в целом, другой для модуля app. Откройте файл build.gradle из модуля app.

Листинг 7.1. Зависимости Gradle (app/build.gradle)

```
apply plugin: 'com.android.application'
android {
    ...
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    ...
    compile 'com.android.support:appcompat-v7:25.0.1'
    ...
}
```

Текущее содержимое секции `dependencies` файла build.gradle должно примерно соответствовать листингу 7.1: оно означает, что проект зависит от всех файлов .jar из каталога libs проекта. Также здесь содержатся зависимости других библиотек, автоматически включаемые при создании проекта в Android Studio.

Gradle также позволяет задавать зависимости, не скопированные в ваш проект. Во время компиляции проекта Gradle находит, загружает и включает такие зависимости за вас. От вас потребуется лишь правильно задать волшебную строку зависимости, а Gradle сделает все остальное.

Впрочем, запоминать эти строки никому не хочется, поэтому Android Studio поддерживает список часто используемых библиотек. Перейдите к структуре проекта (File ▸ Project Structure...).

Выберите в левой части модуль app, а в нем перейдите на вкладку Dependencies. На этой вкладке перечислены зависимости модуля app (рис. 7.9).

(Если в вашем списке присутствуют другие зависимости, не удаляйте их.)

В списке должна быть указана зависимость `AppCompat`. Если ее нет, нажмите кнопку + и выберите вариант Library dependency. Выберите в списке библиотеку appcompat-v7 и щелкните на кнопке OK (рис. 7.10).

Вернитесь в окно редактора с содержимым app/build.gradle; добавленная библиотека появляется в списке (см. листинг 7.1).

(Если файл редактируется вручную, за пределами окна инструментов Project, проект нужно будет синхронизировать с файлом Gradle. В процессе синхронизации

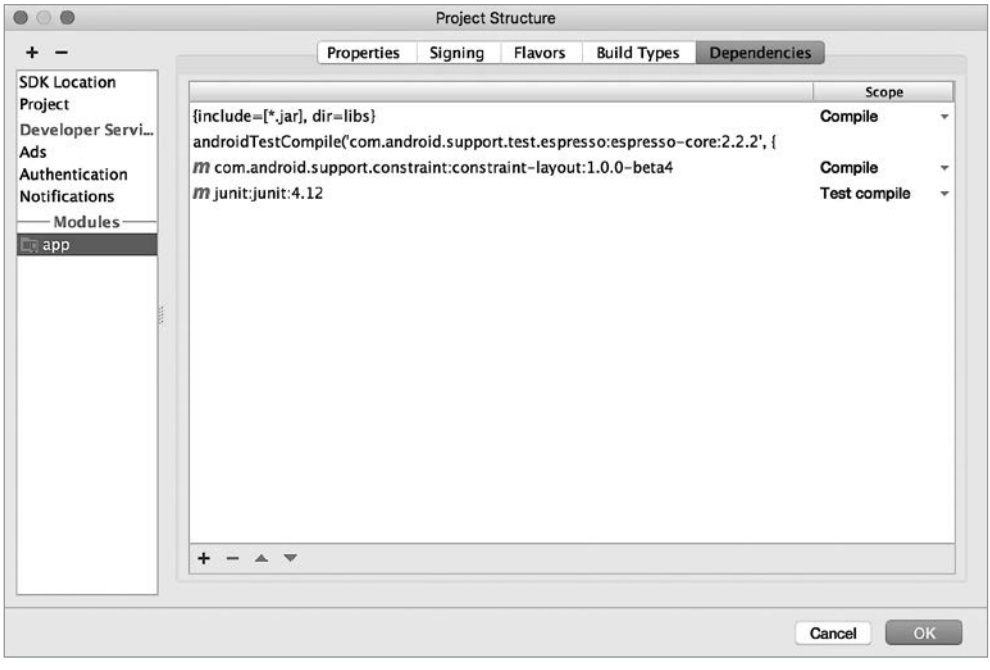


Рис. 7.9. Зависимости app

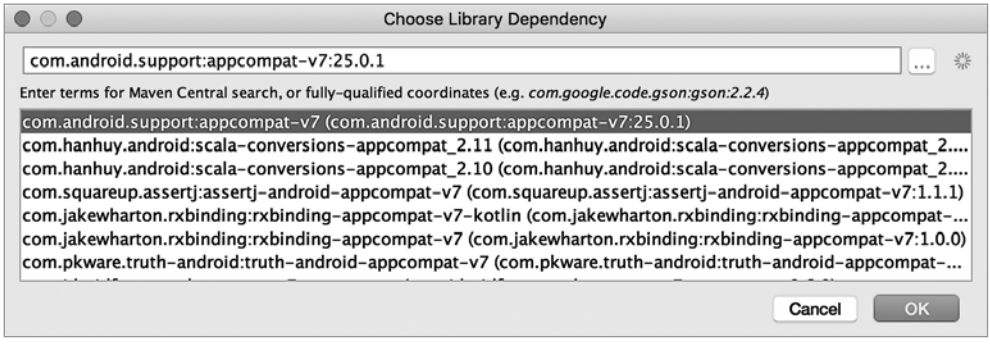


Рис. 7.10. Набор зависимостей

Gradle обновляет построение с учетом изменений, загружая или удаляя необходимые зависимости. При внесении изменений в окне инструментов Project синхронизация запускается автоматически. Чтобы выполнить синхронизацию вручную, выполните команду Tools ▸ Android ▸ Sync Project with Gradle Files.)

В строке зависимости `compile 'com.android.support:appcompat-v7:25.0.0'` используется формат координат Maven: *группа:артефакт:версия*. (Maven — программа управления зависимостями; дополнительную информацию о ней можно найти по адресу *maven.apache.org*.)

Поле *группа* содержит уникальный идентификатор набора библиотек, доступных в репозитории Maven. Часто в качестве идентификатора группы используется базовое имя пакета библиотеки — `com.android.support` для библиотеки AppCompat.

Поле *артефакт* содержит имя конкретной библиотеки в пакете. В нашем примере используется имя `appcompat-v7`.

Наконец, поле *версия* представляет номер версии библиотеки. Приложение CriminalIntent зависит от версии 25.0.0 библиотеки `appcompat-v7`. На момент написания книги последней была версия 25.0.0, но любая более новая версия должна работать в этом проекте. Обычно стоит использовать последнюю версию библиотеки поддержки, чтобы вам были доступны более новые API и вы могли получать свежие исправления. Если Android Studio добавит обновленную версию библиотеки, не возвращайтесь к версии, приведенной в листинге.

Итак, библиотека AppCompat вошла в число зависимостей проекта; теперь ее нужно использовать. В окне инструментов Project найдите и откройте файл CrimeActivity.java. Убедитесь в том, что суперклассом `CrimeActivity` является `AppCompatActivity`.

Листинг 7.2. Настройка шаблонного кода (CrimeActivity.java)

```
public class CrimeActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_crime);
    }

}
```

Прежде чем продолжать работу над CrimeActivity, мы создадим уровень модели `CriminalIntent`. Для этого мы напишем класс `Crime`.

Создание класса Crime

В окне инструментов Project щелкните правой кнопкой мыши на пакете com.bignerdranch.android.criminalintent и выберите команду New ▸ Java Class. Введите имя класса Crime и щелкните на кнопке OK.

Добавьте в Crime.java поля для представления идентификатора преступления, названия, даты и статуса, и конструктор, инициализирующий поля идентификатора и даты (листинг 7.3).

Листинг 7.3. Добавление в классе Crime (Crime.java)

```
public class Crime {

    private UUID mId;
    private String mTitle;
    private Date mDate;
    private boolean mSolved;
```

```
    public Crime() {
        mId = UUID.randomUUID();
        mDate = new Date();
    }
}
```

`UUID` — вспомогательный класс Java, входящий в инфраструктуру Android, — предоставляет простой способ генерирования универсально-уникальных идентификаторов. В конструкторе такой идентификатор генерируется вызовом `UUID.randomUUID()`.

Android Studio может найти два класса с именем `Date`. Импортируйте класс вручную клавишами Option+Return (или Alt+Enter). Когда вам будет предложено выбрать версию класса `Date`, выберите версию `java.util.Date`.

Инициализация переменной `Date` конструктором `Date` по умолчанию присваивает `mDate` текущую дату. Эта дата станет датой преступления по умолчанию.

Затем для свойства `mId`, доступного только для чтения, необходимо сгенерировать только `get`-метод, а для свойств `mTitle`, `mDate` и `mSolved` — `get`- и `set`-методы. Щелкните правой кнопкой мыши после конструктора, выберите команду Generate... ▸ Getter и затем переменную `mId`. Затем сгенерируйте `get`- и `set`-метод для `mTitle`, `mDate` и `mSolved`, повторите этот процесс, но выберите в меню Generate... команду Getter and Setter.

Листинг 7.4. Сгенерированные get- и set-методы (Crime.java)

```
public class Crime {

    private UUID mId;
    private String mTitle;
    private Date mDate;
    private boolean mSolved;

    public Crime() {
        mId = UUID.randomUUID();
        mDate = new Date();
    }

    public UUID getId() {
        return mId;
    }

    public String getTitle() {
        return mTitle;
    }

    public void setTitle(String title) {
        mTitle = title;
    }

    public Date getDate() {
        return mDate;
```

```
    }

    public void setDate(Date date) {
        mDate = date;
    }

    public boolean isSolved() {
        return mSolved;
    }

    public void setSolved(boolean solved) {
        mSolved = solved;
    }
}
```

Вот и все, что нам понадобится для класса `Crime` и уровня модели CriminalIntent в этой главе.

Итак, мы создали уровень модели и активность, которая обеспечивает хостинг фрагмента. А теперь более подробно рассмотрим, как активность выполняет свои функции хоста.

Хостинг UI-фрагментов

Чтобы стать хостом для UI-фрагмента, активность должна:

- определить место представления фрагмента в своем макете;
- управлять жизненным циклом экземпляра фрагмента.

Жизненный цикл фрагмента

На рис. 7.11 показан жизненный цикл фрагмента. Он имеет много общего с жизненным циклом активности: он тоже может находиться в состоянии остановки, приостановки и выполнения, он тоже содержит методы, переопределяемые для выполнения операций в критических точках, многие из которых соответствуют методам жизненного цикла активности.

Эти соответствия играют важную роль. Так как фрагмент работает по поручению активности, его состояние должно отражать текущее состояние активности. Следовательно, ему необходимы соответствующие методы жизненного цикла для выполнения работы активности.

Принципиальное различие между жизненными циклами фрагмента и активности заключается в том, что методы жизненного цикла фрагмента вызываются активностью-хостом, а не ОС. ОС ничего не знает о фрагментах, используемых активностью; фрагменты — внутреннее дело самой активности.

Методы жизненного цикла фрагментов будут более подробно рассмотрены, когда мы продолжим строить приложение CriminalIntent.

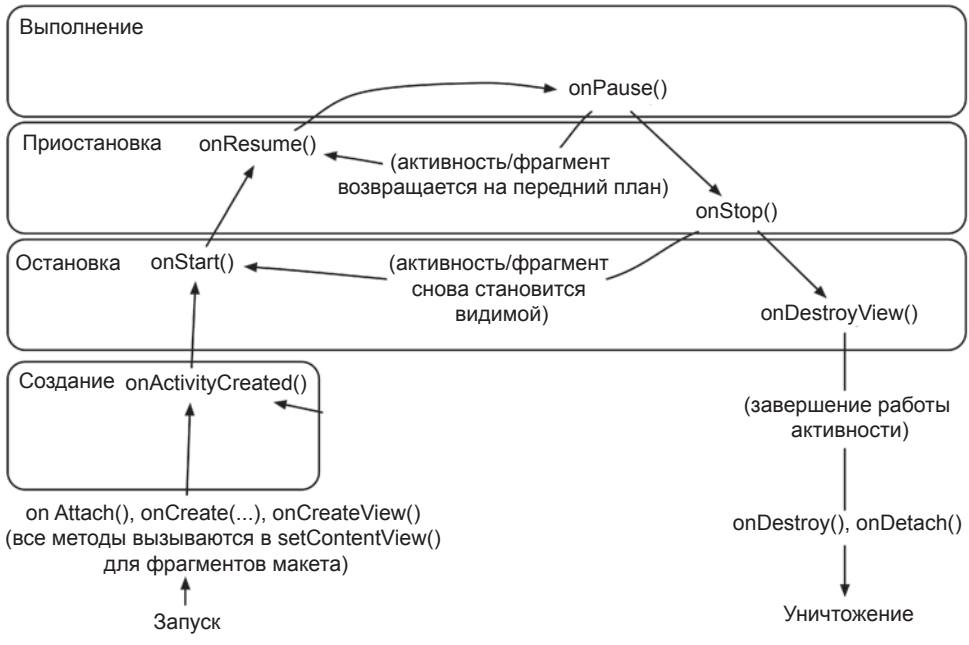


Рис. 7.11. Жизненный цикл фрагмента

Два способа организации хостинга

Существует два основных способа организации хостинга UI-фрагментов в активности:

- добавление фрагмента в *макет* активности;
- добавление фрагмента в *код* активности.

Первый способ — с использованием так называемых *макетных фрагментов* — прост, но недостаточно гибок. Включение фрагмента в макет активности означает жесткую привязку фрагмента и его представления к представлению активности, и вам уже не удастся переключить этот фрагмент на протяжении жизненного цикла активности.

Второй способ сложнее, но только он позволяет управлять фрагментами во время выполнения. Разработчик сам определяет, когда фрагмент добавляется в активность и что с ним происходит после этого. Он может удалить фрагмент, заменить его другим фрагментом, а потом снова вернуть первый.

Итак, для достижения настоящей гибкости пользовательского интерфейса необходимо добавить фрагмент в коде. Именно этот способ мы используем для того, чтобы сделать `CrimeActivity` хостом `CrimeFragment`. Подробности реализации кода будут описаны позднее в этой главе, но сначала мы определим макет `CrimeActivity`.

Определение контейнерного представления

Мы добавим UI-фрагмент в код активности-хоста, но мы все равно должны найти место для представления фрагмента в иерархии представлений активности. В макете `CrimeActivity` этим местом будет элемент `FrameLayout`, изображенный на рис. 7.12.

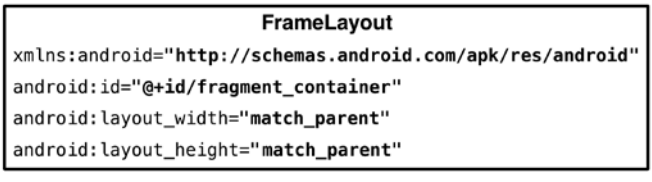


Рис. 7.12. Хостинг фрагмента для CrimeActivity

Элемент `FrameLayout` станет *контейнерным представлением* для `CrimeFragment`. Обратите внимание: контейнерное представление абсолютно универсально; оно не привязывается к классу `CrimeFragment`. Мы можем (и будем) использовать один макет для хостинга разных фрагментов.

Найдите макет `CrimeActivity` в файле res/layout/activity_crime.xml. Откройте файл и замените макет по умолчанию элементом `FrameLayout`, изображенным на рис. 7.12. Разметка XML должна совпадать с приведенной в листинге 7.5.

Листинг 7.5. Создание контейнера фрагмента (activity_crime.xml)

```
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Хотя activity_crime.xml состоит исключительно из контейнерного представления одного фрагмента, макет активности может быть более сложным; он может определять несколько контейнерных представлений, а также собственные виджеты.

Просмотрите файл макета или запустите `CriminalIntent`, чтобы проверить свой код. Вы увидите только пустой элемент `FrameLayout` под панелью инструментов с текстом CriminalIntent, потому что `CrimeActivity` еще не выполняет функции хоста фрагмента (рис. 7.13). (Если в окне предварительного просмотра экран отображается некорректно или вы получите сообщения об ошибках, постройте проект командой Build ▸ Rebuild Project. Если и после этого просмотр не заработает, запустите приложение на эмуляторе или устройстве. На момент написания книги окно предварительного просмотра работало ненадежно.)

Элемент `FrameLayout` пуст, потому что в `CrimeActivity` еще не отображается фрагмент. Позднее мы напишем код, который помещает представление фрагмента в `FrameLayout`. Но сначала фрагмент необходимо создать.

Рис. 7.13. Пустой элемент FrameLayout

(Панель инструментов в верхней части приложения создается автоматически из-за настроек активности. О панелях инструментов будет более подробно рассказано в главе 13.)

Создание UI-фрагмента

Последовательность действий при создании UI-фрагмента не отличается от последовательности действий при создании активности:

- построение интерфейса посредством определения виджетов в файле макета;
- создание класса и назначение макета, который был определен ранее, его представлением;
- подключение виджетов, заполненных на основании макета в коде.

Определение макета CrimeFragment

Представление `CrimeFragment` будет отображать информацию, содержащуюся в экземпляре `Crime`.

Начнем с определения строк, которые увидит пользователь, в файле res/values/strings.xml.

Листинг 7.6. Добавление строк (res/values/strings.xml)

```
<resources>
    <string name="app_name">CriminalIntent</string>
    <string name="crime_title_hint">Enter a title for the crime.</string>
    <string name="crime_title_label">Title</string>
    <string name="crime_details_label">Details</string>
    <string name="crime_solved_label">Solved</string>
</resources>
```

Затем определяется пользовательский интерфейс. Макет представления `CrimeFragment` состоит из вертикального элемента `LinearLayout`, содержащего два виджета `TextView`, виджеты `EditText`, `Button` и `Checkbox`.

Чтобы создать файл макета, щелкните правой кнопкой мыши на папке res/layout в окне инструментов Project и выберите команду New ▸ Layout resource file. Присвойте файлу фрагмента имя fragment_crime.xml. Выберите корневым элементом `LinearLayout` и щелкните на кнопке OK; Android Studio сгенерирует файл за вас.

Когда файл откроется, перейдите к разметке XML. Мастер уже добавил элемент `LinearLayout` за вас. Добавьте виджеты, образующие макет фрагмента, в fragment_crime.xml (рис. 7.7).

Листинг 7.7. Файл макета для представления фрагмента (fragment_crime.xml)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="16dp"
    android:orientation="vertical">

    <TextView
        style="?android:listSeparatorTextViewStyle"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/crime_title_label"/>

    <EditText
        android:id="@+id/crime_title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/crime_title_hint"/>

    <TextView
        style="?android:listSeparatorTextViewStyle"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/crime_details_label"/>

    <Button
        android:id="@+id/crime_date"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>
```

```
    <CheckBox
        android:id="@+id/crime_solved"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/crime_solved_label"/>

</LinearLayout>
```

На вкладке Design отображается представление вашего фрагмента (рис. 7.14).

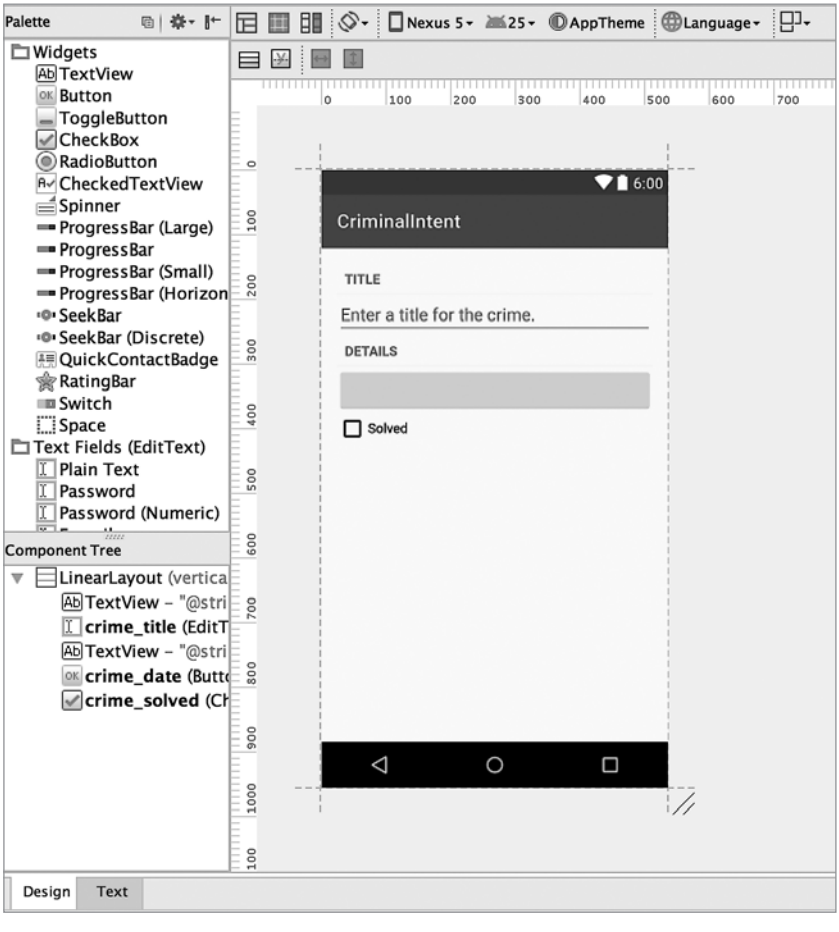


Рис. 7.14. Предварительный просмотр макета фрагмента

(Обновленный код fragment_crime.xml включает новый синтаксис, связанный со стилем представления: `<style="?android:listSeparatorTextViewStyle"`. Не бойтесь. Смысл этого синтаксиса будет объяснен в разделе «Стили, темы и атрибуты тем» в главе 9.)

Создание класса CrimeFragment

Щелкните правой кнопкой мыши на пакете *com.bignerdranch.android.criminalintent* и выберите команду New ▸ Java Class. Введите имя класса `CrimeFragment` и щелкните на кнопке OK, чтобы сгенерировать класс.

Теперь этот класс нужно преобразовать во фрагмент. Измените класс `CrimeFragment` так, чтобы он субклассировал `Fragment`.

Листинг 7.8. Субклассирование класса Fragment (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {

}
```

При субклассировании `Fragment` вы заметите, что Android Studio находит два класса с именем `Fragment`: `Fragment (android.app)` и `Fragment (android.support.v4.app)`. Версия `Fragment` из `android.app` реализует версию фрагментов, встроенную в ОС Android. Мы используем версию библиотеки поддержки, поэтому при появлении диалогового окна выберите версию класса `Fragment` из `android.support.v4.app` (рис. 7.15).

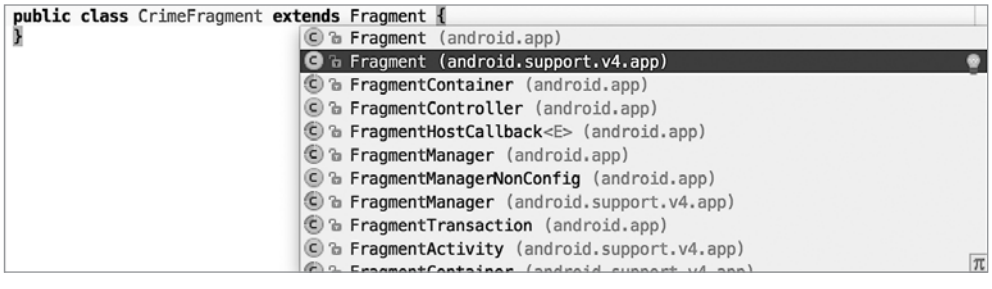


Рис. 7.15. Выбор класса Fragment из библиотеки поддержки

Ваш код должен выглядеть так, как показано в листинге 7.9.

Листинг 7.9. Импортирование Fragment (CrimeFragment.java)

```
package com.bignerdranch.android.criminalintent;

import android.support.v4.app.Fragment;

public class CrimeFragment extends Fragment {

}
```

Если диалоговое окно не открывается или импортировался неверный класс фрагмента, правильный класс можно импортировать вручную. Если файл содержит директиву `import` для `android.app.Fragment`, удалите эту строку кода. Импортируйте правильный класс `Fragment` комбинацией клавиш Option+Return (или

Alt+Enter). Проследите за тем, чтобы была выбрана версия класса `Fragment` из библиотеки поддержки.

Реализация методов жизненного цикла фрагмента

`CrimeFragment` — контроллер, взаимодействующий с объектами модели и представления. Его задача — выдача подробной информации о конкретном преступлении и ее обновление при модификации пользователем.

В приложении GeoQuiz активности выполняли большую часть работы контроллера в методах жизненного цикла. В приложении CriminalIntent эта работа будет выполняться фрагментами в методах жизненного цикла фрагментов. Многие из этих методов соответствуют уже известным вам методам `Activity`, таким как `onCreate(Bundle)`.

В файле CrimeFragment.java добавьте переменную для экземпляра `Crime` и реализацию `Fragment.onCreate(Bundle)`.

Android Studio может немного помочь с переопределением методов. Когда вы определяете метод `onCreate(Bundle)`, введите несколько начальных символов имени метода там, где этот метод должен размещаться. Android Studio выдает список рекомендаций (рис. 7.16).

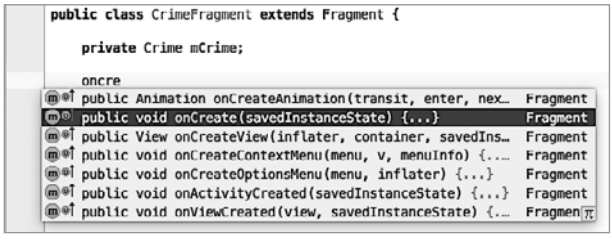


Рис. 7.16. Переопределение метода onCreate(Bundle)

Нажмите Return, чтобы выбрать метод `onCreate(Bundle)`; Android Studio сгенерирует объявление метода за вас. Внесите изменения в свой код, который должен создавать новый объект `Crime` (листинг 7.10).

Листинг 7.10. Переопределение Fragment.onCreate(Bundle) (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
    private Crime mCrime;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mCrime = new Crime();
    }
}
```

В этой реализации стоит обратить внимание на пару моментов. Во-первых, метод `Fragment.onCreate(Bundle)` объявлен открытым, тогда как метод `Activity.`

onCreate(Bundle) объявлен защищенным. Fragment.onCreate(Bundle) и другие методы жизненного цикла Fragment должны быть открытыми, потому что они будут вызываться произвольной активностью, которая станет хостом фрагмента.

Во-вторых, как и в случае с активностью, фрагмент использует объект Bundle для сохранения и загрузки состояния. Вы можете переопределить Fragment.onSaveInstanceState(Bundle) для ваших целей, как и метод Activity.onSaveInstanceState(Bundle).

Обратите внимание на то, что *не происходит* в Fragment.onCreate(Bundle): мы не заполняем представление фрагмента. Экземпляр фрагмента настраивается в Fragment.onCreate(Bundle), но создание и настройка представления фрагмента осуществляются в другом методе жизненного цикла фрагмента:

```
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
    Bundle savedInstanceState)
```

Именно в этом методе заполняется макет представления фрагмента, а заполненный объект View возвращается активности-хосту. Параметры LayoutInflater и ViewGroup необходимы для заполнения макета. Объект Bundle содержит данные, которые используются методом для воссоздания представления по сохраненному состоянию.

В файле CrimeFragment.java добавьте реализацию onCreateView(…), которая заполняет разметку fragment_crime.xml. Для заполнения объявления метода можно воспользоваться приемом, показанным на рис. 7.16.

Листинг 7.11. Переопределение onCreateView(…) (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
    private Crime mCrime;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mCrime = new Crime();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime, container, false);
        return v;
    }
}
```

В методе onCreateView(…) мы явно заполняем представление фрагмента, вызывая LayoutInflater.inflate(…) с передачей идентификатора ресурса макета. Второй параметр определяет родителя представления, что обычно необходимо для правильной настройки виджета. Третий параметр указывает, нужно ли включать заполненное представление в родителя. Мы передаем false, потому что представление будет добавлено в коде активности.

Подключение виджетов во фрагменте

Теперь мы займемся подключением `EditText`, `Checkbox` и `Button` во фрагменте. Это следует делать в методе `onCreateView(…)`.

Начните с `EditText`. После того как представление будет заполнено, метод получит ссылку на `EditText` и добавит слушателя.

Листинг 7.12. Настройка виджета EditText (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
    private Crime mCrime;
    private EditText mTitleField;
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime, container, false);

        mTitleField = (EditText) v.findViewById(R.id.crime_title);
        mTitleField.addTextChangedListener(new TextWatcher() {
            @Override
            public void beforeTextChanged(
                CharSequence s, int start, int count, int after) {
                // Здесь намеренно оставлено пустое место
            }

            @Override
            public void onTextChanged(
                CharSequence s, int start, int before, int count) {
                mCrime.setTitle(s.toString());
            }

            @Override
            public void afterTextChanged(Editable s) {
                // И здесь тоже
            }
        });
        return v;
    }
}
```

Получение ссылок в `Fragment.onCreateView(…)` происходит практически так же, как в `Activity.onCreate(…)`. Единственное различие заключается в том, что для представления фрагмента вызывается метод `View.findViewById(int)`. Метод `Activity.findViewById(int)`, который мы использовали ранее, является вспомогательным методом, вызывающим `View.findViewById(int)` в своей внутренней реализации. У класса `Fragment` аналогичного вспомогательного метода нет, поэтому приходится вызывать основной метод.

Назначение слушателей во фрагменте работает точно так же, как в активности. В листинге 7.12 мы создаем анонимный класс, который реализует интерфейс

слушателя TextWatcher. Этот интерфейс содержит три метода, но нас интересует только один: onTextChanged(…).

В методе onTextChanged(…) мы вызываем toString() для объекта CharSequence, представляющего ввод пользователя. Этот метод возвращает строку, которая затем используется для задания заголовка Crime.

Затем виджет Button настраивается для отображения даты (листинг 7.13).

Листинг 7.13. Настройка текста Button (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
    private Crime mCrime;
    private EditText mTitleField;
    private Button mDateButton;
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime, container, false);
        ...
        mDateButton = (Button) v.findViewById(R.id.crime_date);
        mDateButton.setText(mCrime.getDate().toString());
        mDateButton.setEnabled(false);

        return v;
    }
}
```

Блокировка кнопки вызовом setEnabled(false) гарантирует, что кнопка никак не среагирует на нажатие. Кроме того, внешний вид кнопки изменяется в соответствии с ее заблокированным состоянием. В главе 12 мы снова сделаем кнопку доступной, а пользователь получит возможность выбрать дату преступления.

Переходим к CheckBox: получите ссылку и назначьте слушателя, который обновит поле mSolved объекта Crime, как показано в листинге 7.14.

Листинг 7.14. Прослушивание изменений Checkbox (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
    private Crime mCrime;
    private EditText mTitleField;
    private Button mDateButton;
    private CheckBox mSolvedCheckBox;
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime, container, false);
        ...
        mSolvedCheckBox = (CheckBox)v.findViewById(R.id.crime_solved);
        mSolvedCheckBox.setOnCheckedChangeListener(new OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(CompoundButton buttonView,
```

```
                    boolean isChecked) {
                mCrime.setSolved(isChecked);
            }
        });

        return v;
    }
}
```

После ввода этого кода щелкните на OnCheckedChangeListener:

```
mSolvedCheckBox.setOnCheckedChangeListener(new OnCheckedChangeListener()
```

и воспользуйтесь комбинацией Option+Return (Alt+Enter) для включения необходимой директивы `import`. Вам будут предложены два варианта; выберите версию `android.widget.CompoundButton`.

В зависимости от того, какую версию Android Studio вы используете, функция автозаполнения может вставить `CompoundButton.OnCheckedChangeListener()` вместо того, чтобы оставить код в виде `OnCheckedChangeListener()`. Подойдет любая реализация. Но чтобы она не отличалась от решения, приведенного в книге, щелкните на CompoundButton и нажмите клавиши Option+Return (Alt+Enter).

Выберите вариант добавления статической директивы импортирования для `'android.widget.CompoundButton'` по требованию (рис. 7.17). Код обновляется и приводится в соответствие с листингом 7.14.

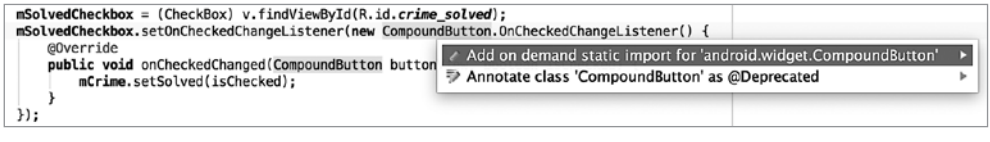


Рис. 7.17. Добавление статической директивы импортирования по требованию

Код `CrimeFragment` готов. Было бы замечательно, если бы вы могли немедленно запустить CriminalIntent и поэкспериментировать с написанным кодом. К сожалению, это невозможно — фрагменты не могут самостоятельно выводить свои представления на экран. Сначала необходимо добавить `CrimeFragment` в `CrimeActivity`.

Добавление UI-фрагмента в FragmentManager

Когда в Honeycomb появился класс `Fragment`, в класс `Activity` были внесены изменения: в него был добавлен компонент, называемый `FragmentManager`. Он отвечает за управление фрагментами и добавление их представлений в иерархию представлений активности (рис. 7.18).

`FragmentManager` управляет двумя структурами: списком фрагментов и стеком транзакций фрагментов (о котором я вскоре расскажу).

В приложении CriminalIntent нас интересует только список фрагментов `FragmentManager`.

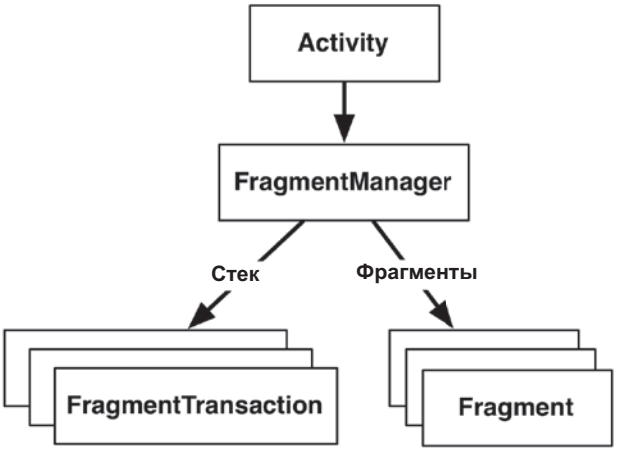


Рис. 7.18. FragmentManager

Чтобы добавить фрагмент в активность в коде, следует явно обратиться с вызовом к объекту `FragmentManager` активности. Прежде всего необходимо получить сам объект `FragmentManager`. В CrimeActivity.java включите следующий код в `onCreate(Bundle)`.

Листинг 7.15. Получение объекта FragmentManager (CrimeActivity.java)

```
public class CrimeActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_crime);

        FragmentManager fm = getSupportFragmentManager();
    }
}
```

Если после добавления этой строки возникнет ошибка, проверьте директивы импортирования и убедитесь в том, что импортируется версия класса `FragmentManager` из библиотеки поддержки.

Мы вызываем `getSupportFragmentManager()`, потому что в приложении используется библиотека поддержки и класс `AppCompatActivity`. Если бы мы не использовали библиотеку поддержки, то вместо этого можно было бы субклассировать `Activity` и вызвать `getFragmentManager()`.

Транзакции фрагментов

После получения объекта `FragmentManager` добавьте следующий код, который передает ему фрагмент для управления. (Позднее мы рассмотрим этот код более подробно, а пока просто включите его в приложение.)

Листинг 7.16. Добавление CrimeFragment (CrimeActivity.java)

```
public class CrimeActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_crime);

        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment = fm.findFragmentById(R.id.fragment_container);

        if (fragment == null) {
            fragment = new CrimeFragment();
            fm.beginTransaction()
                .add(R.id.fragment_container, fragment)
                .commit();
        }
    }
}
```

Разбираться в коде, добавленном в листинге 7.15, лучше всего не с начала. Найдите операцию `add(…)` и окружающий ее код. Этот код создает и закрепляет *транзакцию фрагмента*:

```
if (fragment == null) {
    fragment = new CrimeFragment();
    fm.beginTransaction()
        .add(R.id.fragment_container, fragment)
        .commit();
```

Транзакции фрагментов используются для добавления, удаления, присоединения, отсоединения и замены фрагментов в списке фрагментов. Они лежат в основе механизма использования фрагментов для формирования и модификации экранов во время выполнения. `FragmentManager` ведет стек транзакций, по которому вы можете перемещаться.

Метод `FragmentManager.beginTransaction()` создает и возвращает экземпляр `FragmentTransaction`. Класс `FragmentTransaction` использует *динамичный интерфейс*: методы, настраивающие `FragmentTransaction`, возвращают `FragmentTransaction` вместо `void`, что позволяет объединять их вызовы в цепочку. Таким образом, выделенный код в приведенном выше листинге, означает: «Создать новую транзакцию фрагмента, включить в нее одну операцию `add`, а затем закрепить».

Метод `add(…)` является основным содержанием транзакции. Он получает два параметра: идентификатор контейнерного представления и недавно созданный объект `CrimeFragment`. Идентификатор контейнерного представления должен быть вам знаком: это идентификатор ресурса элемента `FrameLayout`, определенного в файле activity_crime.xml.

Идентификатор контейнерного представления выполняет две функции:

- сообщает `FragmentManager`, где в представлении активности должно находиться представление фрагмента;

- обеспечивает однозначную идентификацию фрагмента в списке `FragmentManager`.

Когда вам потребуется получить экземпляр `CrimeFragment` от `FragmentManager`, запросите его по идентификатору контейнерного представления.

```
FragmentManager fm = getSupportFragmentManager();
Fragment fragment = fm.findFragmentById(R.id.fragment_container);

if (fragment == null) {
    fragment = new CrimeFragment();
    fm.beginTransaction()
        .add(R.id.fragment_container, fragment)
        .commit();
}
```

Может показаться странным, что `FragmentManager` идентифицирует `CrimeFragment` по идентификатору ресурса `FrameLayout`. Однако идентификация UI-фрагмента по идентификатору ресурса его контейнерного представления встроена в механизм работы `FragmentManager`. Если вы добавляете в активность несколько фрагментов, то обычно для каждого фрагмента создается отдельный контейнер со своим идентификатором.

Теперь мы можем кратко описать код, добавленный в листинг 7.15, от начала до конца.

Рис. 7.19. CrimeActivity является хостом представления CrimeFragment

Сначала у `FragmentManager` запрашивается фрагмент с идентификатором контейнерного представления `R.id.fragmentContainer`. Если этот фрагмент уже находится в списке, `FragmentManager` возвращает его.

Почему фрагмент может уже находиться в списке? Вызов `CrimeActivity.onCreate(Bundle)` может быть выполнен в ответ на *воссоздание* объекта `CrimeActivity` после его уничтожения из-за поворота устройства или освобождения памяти. При уничтожении активности ее экземпляр `FragmentManager` сохраняет список фрагментов. При воссоздании активности новый экземпляр `FragmentManager` загружает список и воссоздает хранящиеся в нем фрагменты, чтобы все работало как прежде.

С другой стороны, если фрагменты с заданным идентификатором контейнерного представления отсутствуют, значение `fragment` равно `null`. В этом случае мы создаем новый экземпляр `CrimeFragment` и новую транзакцию, которая добавляет фрагмент в список.

Теперь `CrimeActivity` является хостом для `CrimeFragment`. Чтобы убедиться в этом, запустите приложение `CriminalIntent`. На экране отображается представление, определенное в файле fragment_crime.xml (рис. 7.19).

FragmentManager и жизненный цикл фрагмента

После знакомства с `FragmentManager` стоит еще раз вернуться к жизненному циклу фрагмента (рис. 7.20).

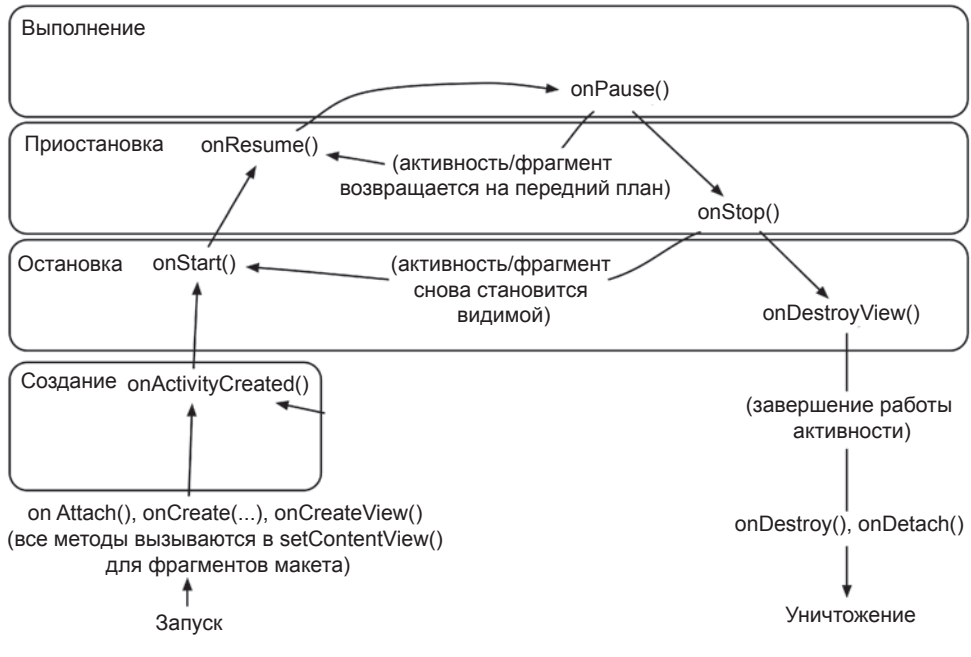


Рис. 7.20. Жизненный цикл фрагмента (повторно)

Объект `FragmentManager` активности отвечает за вызов методов жизненного цикла фрагментов в списке. Методы `onAttach(Context)`, `onCreate(Bundle)` и `onCreateView(…)` вызываются при добавлении фрагмента в `FragmentManager`.

Метод `onActivityCreated(Bundle)` вызывается после выполнения метода `onCreate(Bundle)` активности-хоста. Мы добавляем `CrimeFragment` в `CrimeActivity.onCreate(Bundle)`, так что этот метод будет вызываться после добавления фрагмента.

Что произойдет, если добавить фрагмент в то время, когда активность уже находится в состоянии остановки, приостановки или выполнения? В этом случае `FragmentManager` немедленно проводит фрагмент через все действия, необходимые для его согласования с состоянием активности. Например, при добавлении фрагмента в активность, уже находящуюся в состоянии выполнения, фрагмент получит вызовы `onAttach(Context)`, `onCreate(Bundle)`, `onCreateView(…)`, `onActivityCreated(Bundle)`, `onStart()` и затем `onResume()`.

После того как состояние фрагмента будет согласовано с состоянием активности, объект `FragmentManager` активности-хоста будет вызывать дальнейшие методы жизненного цикла приблизительно одновременно с получением соответствующих вызовов от ОС для синхронизации состояния фрагмента с состоянием активности.

Архитектура приложений с фрагментами

При разработке приложений с фрагментами очень важно правильно подойти к проектированию. Многие разработчики после первого знакомства с фрагментами пытаются применять их ко всем компонентам приложения, подходящим для повторного использования. Такой способ использования фрагментов ошибочен.

Фрагменты предназначены для инкапсуляции основных компонентов для повторного использования. В данном случае основные компоненты находятся на уровне всего экрана приложения. Если на экран одновременно выводится большое количество фрагментов, ваш код замусоривается транзакциями фрагментов и неочевидными обязанностями. С точки зрения архитектуры существует другой, более правильный способ организации повторного использования вторичных компонентов — выделение их в специализированное представление (класс, субклассирующий `View`, или один из его субклассов).

Пользуйтесь фрагментами осмотрительно. На практике не рекомендуется отображать более двух или трех фрагментов одновременно (рис. 7.21).

Почему все наши активности используют фрагменты

С этого момента фрагменты будут использоваться во всех приложениях этой книги — даже в самых простых. На первый взгляд такое решение кажется чрезмерным: многие примеры, которые вам встретятся в следующих главах, могут быть записаны без фрагментов. Для создания пользовательских интерфейсов и управления ими можно обойтись активностями; возможно, это даже уменьшит объем кода.

Тем не менее мы полагаем, что вам стоит поскорее привыкнуть к паттерну, который наверняка пригодится вам в реальной работе.

Рис. 7.21. Не гонитесь за количеством

Кто-то скажет, что лучше сначала написать простое приложение без фрагментов, а затем добавить их, когда потребуется (и если потребуется). Эта идея заложена в основу методологии экстремального программирования YAGNI. Сокращение YAGNI означает «You Aren't Gonna Need It» («Вам это не понадобится»); этот принцип убеждает вас не писать код, который, по вашему мнению, *может* понадобиться позже. Почему? Потому что YAGNI. Возникает соблазн сказать YAGNI фрагментам.

К сожалению, добавление фрагментов в будущем может превратиться в мину замедленного действия. Превратить активность в активность хоста UI-фрагмента несложно, но при этом возникает множество неприятных ловушек. Если одними интерфейсами будут управлять активности, а другими — фрагменты, это только усугубит ситуацию, потому что вам придется отслеживать эти бессмысленные различия. Гораздо проще с самого начала написать код с использованием фрагментов и не возиться с его последующей переработкой и не запоминать, какой стиль контроллера используется в каждой части приложения.

Итак, в том, что касается фрагментов, мы применяем другой принцип: AUF, или «Always Use Fragments» («Всегда используй фрагменты»). Выбирая между использованием фрагмента или активности, вы потратите немало нервных клеток, а это того не стоит. AUF!

Для любознательных: фрагменты и библиотека поддержки

В этой главе для использования фрагментов из библиотеки поддержки мы включали библиотеку AppCompat. Сама по себе библиотека AppCompat не содержит

реализации фрагментов — она зависит от библиотеки `support-v4`, в которой и находятся эти реализации.

Google предоставляет много разных библиотек поддержки, включая `support-v4`, `appcompat-v7`, `recyclerview-v7` и многие другие. Говоря о библиотеке поддержки, обычно имеют в виду `support-v4`. Это была первая библиотека поддержки, которую компания Google предоставила разработчикам. Со временем в нее добавлялись все новые инструменты, и в итоге она превратилась в свалку без сколько-нибудь ясных целей. В этот момент компания Google решила разработать семейство библиотек поддержки вместо одной библиотеки.

Библиотека поддержки (`support-v4`) содержит реализацию фрагментов, которая использовалась в этой главе. Например, именно здесь находится исходный код `android.support.v4.app.Fragment`. Библиотека поддержки также включает субкласс `Activity`: `FragmentActivity`. Чтобы использовать фрагменты из библиотеки поддержки, ваши активности должны субклассировать `FragmentActivity`.

Как видно из рис. 7.22, `AppCompatActivity` является субклассом этого класса `FragmentActivity`, что и позволило нам использовать фрагменты из библиотеки поддержки в этой главе. Если бы вы решили использовать фрагменты из библиотеки поддержки без AppCompat, то вам пришлось бы включить зависимость `support-v4` в проект, и вы бы субклассировали в каждом из классов активностей `FragmentActivity` вместо `AppCompatActivity`.

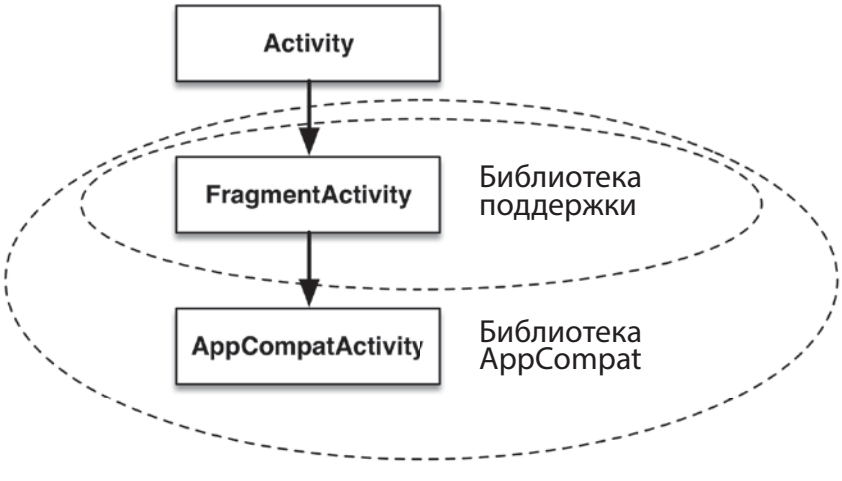


Рис. 7.22. Иерархия классов AppCompatActivity

Если объяснения показались вам излишне запутанными — вы не ошиблись. Но не беспокойтесь, большинство разработчиков Android использует эти библиотеки точно так же, как в этой главе: они используют библиотеку AppCompat, а не работают с библиотекой поддержки напрямую. Возможности библиотеки AppCompat более подробно рассматриваются в главе 13.

Для любознательных: почему фрагменты из библиотеки поддержки лучше

В этой книге мы отдаем предпочтение реализации фрагментов из библиотеки поддержки перед реализацией, встроенной в ОС Android, — на первый взгляд такое решение кажется неожиданным. В конце концов, реализация фрагментов из библиотеки поддержки изначально создавалась для того, чтобы разработчики могли использовать фрагменты в старых версиях Android, не поддерживающих API. Сегодня большинство разработчиков могут работать с версиями Android, включающими поддержку фрагментов.

И все же мы предпочитаем фрагменты из библиотеки поддержки. Почему? Потому что вы можете обновить версию библиотеки поддержки в приложении и в любой момент опубликовать новую версию своего приложения. Новые версии библиотеки поддержки выходят несколько раз в год. Когда в API фрагментов добавляются новые возможности, они также включаются в API фрагментов в библиотеке поддержки вместе со всеми доступными исправлениями. Чтобы использовать новые возможности, достаточно обновить версию библиотеки поддержки в новом приложении.

Например, официальная поддержка вложенных фрагментов (хостинг фрагментов во фрагментах) добавилась в Android 4.2. Если вы используете фрагменты из ОС Android и поддерживаете Android 4.0 и выше, вам не удастся использовать этот API на всех устройствах, поддерживаемых приложением. При использовании библиотеки поддержки вы можете обновить версию библиотеки в своем приложении и создавать вложенные фрагменты, пока не кончится свободная память на устройстве.

У фрагментов из библиотеки поддержки нет сколько-нибудь существенных недостатков. Реализации фрагментов в ОС и библиотеке поддержки почти идентичны. Единственный заметный недостаток — необходимость включения в проект библиотеки поддержки. Однако в текущей версии ее размер меньше мегабайта, причем вполне возможно, что вы все равно будете использовать библиотеку поддержки ради других возможностей.

В этой книге и в разработке своих приложений мы руководствуемся практическими соображениями. Библиотека поддержки лучше.

Если вы не боитесь трудностей и не верите приведенному выше совету, вы можете использовать реализацию фрагментов, встроенную в ОС Android.

Чтобы использовать классы фрагментов стандартной библиотеки, необходимо внести в проект три изменения:

- Субклассируйте класс `Activity` стандартной библиотеки (`android.app.Activity`) вместо `FragmentActivity` или `AppCompatActivity`. В API уровня 11 и выше активности содержат готовую поддержку фрагментов.
- Субклассируйте `android.app.Fragment` вместо `android.support.v4.app.Fragment`.
- Чтобы получить объект `FragmentManager`, используйте вызов `getFragmentManager()` вместо `getSupportFragmentManager()`.

8 Вывод списков и RecyclerView

Уровень модели CriminalIntent в настоящее время состоит из единственного экземпляра `Crime`. В этой главе мы обновим приложение CriminalIntent, чтобы оно поддерживало списки. В списке для каждого преступления будут отображаться краткое описание и дата, а также признак его раскрытия (рис. 8.1).

Рис. 8.1. Список преступлений

На рис. 8.2 показана общая структура приложения CriminalIntent для этой главы.

На уровне модели появляется новый объект `CrimeLab`, который представляет собой централизованное хранилище для объектов `Crime`.

Для отображения списка на уровне контроллера CriminalIntent появляется новая активность и новый фрагмент: `CrimeListActivity` и `CrimeListFragment`.

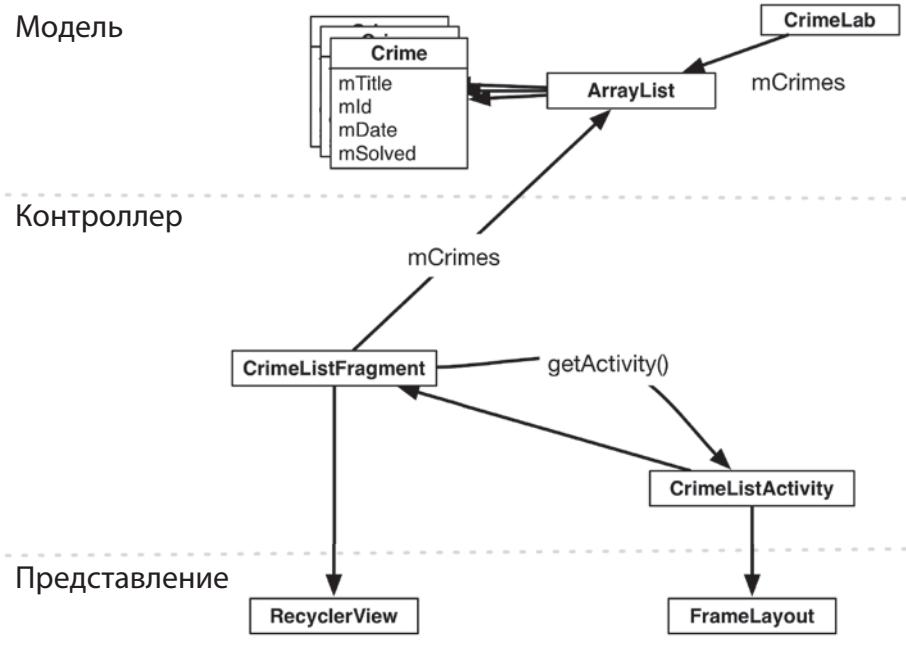


Рис. 8.2. Приложение CriminalIntent со списком

(Где находятся классы `CrimeActivity` и `CrimeFragment` на рис. 8.2? Они являются частью представления детализации, поэтому на рисунке их нет. В главе 10 мы свяжем части списка и детализации CriminalIntent.)

На рис. 8.2 также видны объекты представлений, связанные с `CrimeListActivity` и `CrimeListFragment`. Представление активности состоит из объекта `FrameLayout`, содержащего фрагмент. Представление фрагмента состоит из `RecyclerView`. Класс `RecyclerView` более подробно рассматривается позднее в этой главе.

Обновление уровня модели CriminalIntent

Прежде всего необходимо преобразовать уровень модели `CriminalIntent` из одного объекта `Crime` в список `List` объектов `Crime`.

Синглеты и централизованное хранение данных

Для хранения массива-списка преступлений будет использоваться *синглетный* (singleton) класс. Такие классы допускают создание только одного экземпляра.

Экземпляр синглетного класса существует до тех пор, пока приложение остается в памяти, так что при хранении списка в синглетном объекте данные остаются доступными, что бы ни происходило с активностями, фрагментами и их жизненны-

ми циклами. Будьте внимательны, поскольку синглетные классы уничтожаются, когда Android удаляет ваше приложение из памяти. Синглет `CrimeLab` не подходит для долгосрочного хранения данных, но позволяет приложению назначить одного владельца данных преступления и предоставляет возможность простой передачи этой информации между классами-контроллерами.

(За дополнительной информацией о синглетных классах обращайтесь к разделу «Для любознательных» в конце главы 14.)

Чтобы создать синглетный класс, следует создать класс с закрытым конструктором и методом `get()`. Если экземпляр уже существует, то `get()` просто возвращает его. Если экземпляр еще не существует, то `get()` вызывает конструктор для его создания.

Щелкните правой кнопкой мыши на пакете *com.bignerdranch.android.criminalintent* и выберите команду New ▸ Java Class. Введите имя класса `CrimeLab` и щелкните на кнопке Finish.

В файле CrimeLab.java (листинг 8.1) реализуйте `CrimeLab` как синглетный класс с закрытым конструктором и методом `get()`.

Листинг 8.1. Синглетный класс (CrimeLab.java)

```
public class CrimeLab {
    private static CrimeLab sCrimeLab;

    public static CrimeLab get(Context context) {
        if (sCrimeLab == null) {
            sCrimeLab = new CrimeLab(context);
        }
        return sCrimeLab;
    }
    private CrimeLab(Context context) {

    }
}
```

В реализации `CrimeLab` есть несколько моментов, заслуживающих внимания. Во-первых, обратите внимание на префикс `s` у переменной `sCrimeLab`. Мы используем это условное обозначение Android, чтобы показать, что переменная `sCrimeLab` является статической.

Также обратите внимание на закрытый конструктор `CrimeLab`. Другие классы не смогут создать экземпляр `CrimeLab` в обход метода `get()`.

Наконец, в методе `get()` конструктору `CrimeLab` передается параметр `Context`. Пока этот объект не используется, но мы вернемся к нему в главе 14.

Для начала предоставим `CrimeLab` несколько объектов `Crime` для хранения. В конструкторе `CrimeLab` создайте пустой список `List` объектов `Crime`. Также добавьте два метода: `getCrimes()` возвращает `List`, а `getCrime(UUID)` возвращает объект `Crime` с заданным идентификатором (листинг 8.2).

Листинг 8.2. Создание списка List объектов Crime (CrimeLab.java)

```
public class CrimeLab {
    private static CrimeLab sCrimeLab;

    private List<Crime> mCrimes;

    public static CrimeLab get(Context context) {
        ...
    }

    private CrimeLab(Context context) {
        mCrimes = new ArrayList<>();
    }

    public List<Crime> getCrimes() {
        return mCrimes;
    }

    public Crime getCrime(UUID id) {
        for (Crime crime : mCrimes) {
            if (crime.getId().equals(id)) {
                return crime;
            }
        }
        return null;
    }
}
```

`List<E>` — интерфейс поддержки упорядоченного списка объектов заданного типа. Он определяет методы получения, добавления и удаления элементов. Одна из распространенных реализаций `List` — `ArrayList` — использует для хранения элементов списка обычный массив Java.

Так как `mCrimes` содержит `ArrayList`, а `ArrayList` также является частным случаем `List`, и `ArrayList` и `List` являются действительными типами для `mCrimes`. В подобных ситуациях мы рекомендуем использовать в объявлении переменной тип интерфейса: `List`. В этом случае, если вам когда-либо понадобится перейти на другую реализацию `List`, например `LinkedList`, вы легко сможете это сделать.

В строке создания экземпляра `mCrimes` используется «ромбовидный» синтаксис `<>`, появившийся в Java 7. Эта сокращенная запись приказывает компилятору определить тип элементов, которые будут храниться в `List`, на основании обобщенного аргумента, переданного при объявлении переменной. В данном случае компилятор заключает, что `ArrayList` содержит объекты `Crime`, потому что в объявлении переменной `private List<Crime> mCrimes;` указан обобщенный аргумент `Crime`. (До Java 7 разработчикам приходилось использовать более длинную эквивалентную конструкцию `mCrimes = new ArrayList<Crime>();`.)

Со временем `List` будет содержать объекты `Crime`, созданные пользователем, которые будут сохраняться и загружаться повторно. А пока заполним массив 100 однообразных объектов `Crime` (листинг 8.3).

Листинг 8.3. Генерирование тестовых объектов (CrimeLab.java)

```
private CrimeLab(Context context) {
    mCrimes = new ArrayList<>();
    for (int i = 0; i < 100; i++) {
        Crime crime = new Crime();
        crime.setTitle("Crime #" + i);
        crime.setSolved(i % 2 == 0); // Для каждого второго объекта
        mCrimes.add(crime);
    }
}
```

Теперь у нас имеется полностью загруженный уровень модели и 100 преступлений для вывода на экран.

Абстрактная активность для хостинга фрагмента

Вскоре мы создадим класс `CrimeListActivity`, предназначенный для выполнения функций хоста для `CrimeListFragment`. Но сначала будет создано представление для `CrimeListActivity`.

Обобщенный макет для хостинга фрагмента

Для `CrimeListActivity` можно просто воспользоваться макетом, определенным в файле activity_crime.xml (листинг 8.4). Этот макет определяет виджет `FrameLayout` как контейнерное представление для фрагмента, который затем указывается в коде активности.

Листинг 8.4. Файл activity_crime.xml уже содержит универсальную разметку

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/fragmentContainer"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
/>
```

Поскольку в файле activity_crime.xml не указан конкретный фрагмент, он может использоваться для любой активности, выполняющей функции хоста для одного фрагмента. Переименуем его в activity_fragment.xml, чтобы отразить этот факт.

В окне инструментов Project щелкните правой кнопкой мыши на файле res/layout/activity_crime.xml. (Будьте внимательны — щелкнуть нужно на activity_crime.xml, а не на fragment_crime.xml.)

Выберите в контекстном меню команду Refactor ▸ Rename.... Введите имя activity_fragment.xml и щелкните на кнопке Refactor.

При переименовании ресурса ссылки на него обновляются автоматически. Если вы получите сообщение об ошибке в CrimeActivity.java, вам придется вручную обновить ссылку `CrimeActivity`, как показано в листинге 8.5.

Листинг 8.5. Обновление файла макета для CrimeActivity (CrimeActivity.java)

```
public class CrimeActivity extends AppCompatActivity {
    /** Вызывается при исходном создании активности. */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ~~setContentView(R.layout.activity_crime);~~
        setContentView(R.layout.activity_fragment);

        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment = fm.findFragmentById(R.id.fragment_container);
        if (fragment == null) {
            fragment = new CrimeFragment();
            fm.beginTransaction()
                .add(R.id.fragment_container, fragment)
                .commit();
        }
    }
}
```

Абстрактный класс активности

Для создания класса `CrimeListActivity` можно повторно использовать код `CrimeActivity`. Взгляните на код, написанный для `CrimeActivity` (скопирован в листинг 8.6): он прост и практически универсален. Собственно, в нем есть всего одно не универсальное место: создание экземпляра `CrimeFragment` перед его добавлением в `FragmentManager`.

Листинг 8.6. Класс CrimeActivity почти универсален (CrimeActivity.java)

```
public class CrimeActivity extends AppCompatActivity {
    /** Вызывается при исходном создании активности. */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fragment);

        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment = fm.findFragmentById(R.id.fragment_container);

        if (fragment == null) {
            fragment = new CrimeFragment();
            fm.beginTransaction()
                .add(R.id.fragment_container, fragment)
                .commit();
        }
    }
}
```

Почти в каждой активности, которая будет создаваться в этой книге, будет присутствовать такой же код. Чтобы нам не приходилось вводить его снова и снова, мы выделим его в абстрактный класс.

Щелкните правой кнопкой мыши на пакете com.bignerdranch.android.criminalintent, выберите команду New ▸ Java Class и введите имя нового класса SingleFragmentActivity. Сделайте этот класс субклассом `AppCompatActivity` и объявите абстрактным классом. Сгенерированный файл должен выглядеть так:

Листинг 8.7. Создание абстрактной активности (SingleFragmentActivity.java)

```
public abstract class SingleFragmentActivity extends AppCompatActivity {

}
```

Теперь включите следующий фрагмент в SingleFragmentActivity.java. Не считая выделенных частей, он идентичен старому коду `CrimeActivity`.

Листинг 8.8. Добавление обобщенного суперкласса (SingleFragmentActivity.java)

```
public abstract class SingleFragmentActivity extends AppCompatActivity {

    protected abstract Fragment createFragment();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fragment);

        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment = fm.findFragmentById(R.id.fragment_container);

        if (fragment == null) {
            fragment = createFragment();
            fm.beginTransaction()
                .add(R.id.fragment_container, fragment)
                .commit();
        }
    }
}
```

В этом коде представление активности заполняется по данным activity_fragment.xml. Затем мы ищем фрагмент в `FragmentManager` этого контейнера, создавая и добавляя его, если он не существует.

Код в листинге 8.8 отличается от кода `CrimeActivity` только абстрактным методом `createFragment()`, который используется для создания экземпляра фрагмента. Субклассы `SingleFragmentActivity` реализуют этот метод так, чтобы он возвращал экземпляр фрагмента, хостом которого является активность.

Использование абстрактного класса

Попробуем использовать класс с `CrimeListActivity`. Замените суперкласс `CrimeListActivity` на `SingleFragmentActivity`, удалите реализацию `onCreate(Bundle)` и реализуйте метод `createFragment()` так, как показано в листинге 8.9.

Листинг 8.9. Переработка CrimeListActivity (CrimeListActivity.java)

```
public class CrimeActivity extends AppCompatActivity SingleFragmentActivity {
    /** Вызывается при исходном создании активности. */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fragment);
        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment = fm.findFragmentById(R.id.fragment_container);
        if (fragment == null) {
           fragment = new CrimeFragment();
            fm.beginTransaction()
                .add(R.id.fragment_container, fragment)
               .commit();
        }
    }

    @Override
    protected Fragment createFragment() {
        return new CrimeFragment();
    }
}
```

Создание новых контроллеров

А сейчас мы создадим два новых класса-контроллера: `CrimeListActivity` и `CrimeListFragment`.

Щелкните правой кнопкой мыши на пакете com.bignerdranch.android.criminalintent, выберите команду New ▸ Java Class и присвойте классу имя `CrimeListActivity`.

Измените новый класс `CrimeListActivity` так, чтобы он тоже субклассировал `SingleFragmentActivity` и реализовал метод `createFragment()`.

Листинг 8.10. Реализация CrimeListActivity (CrimeListActivity.java)

```
public class CrimeListActivity extends SingleFragmentActivity {

    @Override
    protected Fragment createFragment() {
        return new CrimeListFragment();
    }
}
```

Если ваш класс `CrimeListActivity` содержит другие методы, такие как `onCreate`, — удалите их. Пусть класс `SingleFragmentActivity` выполняет свою работу, а реализация `CrimeListActivity` будет по возможности простой.

Класс `CrimeListFragment` еще не был создан. Исправим этот недочет.

Снова щелкните правой кнопкой мыши на пакете com.bignerdranch.android.criminalintent, выберите команду New ▸ Java Class и присвойте классу имя `CrimeListFragment`.

Листинг 8.11. Реализация CrimeListFragment (CrimeListFragment.java)

```
public class CrimeListFragment extends Fragment {
      // Пока пусто
}
```

Пока `CrimeListFragment` остается пустой оболочкой фрагмента. Мы будем работать с этим классом позднее в этой главе.

Класс `SingleFragmentActivity` существенно сократит объем кода и сэкономит немало времени в процессе чтения. А пока ваш код активности стал аккуратным и компактным.

Объявление CrimeListActivity

Теперь, когда класс `CrimeListActivity` создан, его следует объявить в манифесте. Кроме того, список преступлений должен выводиться на первом экране, который виден пользователю после запуска CriminalIntent; следовательно, активность `CrimeListActivity` должна быть активностью лаунчера.

Включите в манифест объявление `CrimeListActivity` и переместите фильтр интентов из объявления `CrimeActivity` в объявление `CrimeListActivity`, как показано в листинге 8.12.

Листинг 8.12. Объявление CrimeListActivity активностью лаунчера (AndroidManifest.xml)

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity android:name=".CrimeListActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity android:name=".CrimeActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

</application>
```

`CrimeListActivity` теперь является активностью лаунчера. Запустите CriminalIntent; на экране появляется виджет `FrameLayout` из `CrimeListActivity`, содержащий пустой фрагмент `CrimeListFragment` (рис. 8.3).

Рис. 8.3. Пустой экран CrimeListActivity

RecyclerView, Adapter и ViewHolder

Итак, в `CrimeListFragment` должен отображаться список. Для этого мы воспользуемся классом `RecyclerView`.

Класс `RecyclerView` является субклассом `ViewGroup`. Он выводит список дочерних объектов `View`, по одному для каждого элемента. В зависимости от сложности отображаемых данных дочерние объекты `View` могут быть сложными или очень простыми.

Наша первая реализация передачи данных для отображения будет очень простой: в элементе списка будет отображаться только описание и дата объекта `Crime`, а объект `View` каждой строки содержит два виджета `TextView` (рис. 8.4).

На рис. 8.4 изображены 12 строк с виджетами `TextView`. Позднее вы сможете запустить CriminalIntent и провести по экрану, чтобы прокрутить 100 виджетов `TextView` для просмотра всех объектов `Crime`. Значит ли это, что вам придется управлять 100 объектами `TextView`? Благодаря `RecyclerView` — нет, не придется.

Решение с созданием `TextView` для каждого элемента списка быстро становится нереальным. Как несложно представить, список может содержать намного более 100 элементов, а виджеты `TextView` могут быть намного сложнее использованной нами простой реализации. Кроме того, представление `View` необходимо объекту

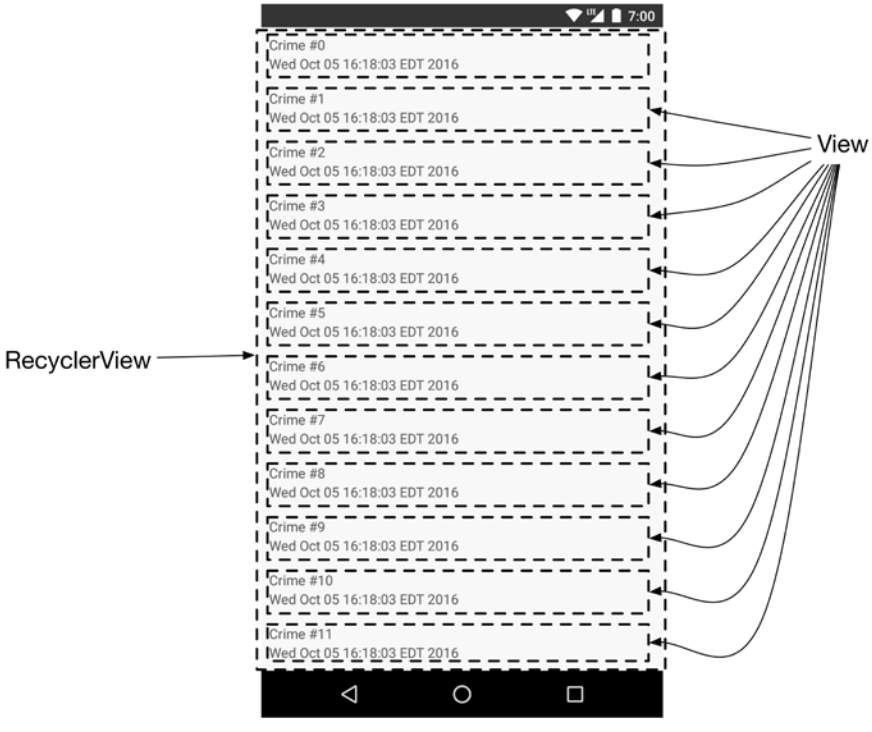


Рис. 8.4. RecyclerView с дочерними виджетами TextView

`Crime` только во время его нахождения на экране, поэтому держать наготове 100 представлений нет необходимости. Будет намного эффективнее создавать объекты представлений только тогда, когда они действительно необходимы.

Виджет `RecyclerView` именно так и поступает. Вместо того чтобы создавать 100 представлений `View`, он создает 12 — достаточно для заполнения экрана. А когда виджет `View` выходит за пределы экрана, `RecyclerView` использует его заново вместо того, чтобы просто выбрасывать. Короче говоря, `RecyclerView` снова и снова перерабатывает использованные виджеты представлений.

ViewHolder и Adapter

Единственная обязанность `RecyclerView` — повторное использование виджетов `TextView` и их позиционирование на экране. Чтобы обеспечить их исходное размещение, он работает с двумя классами, которые мы вскоре построим: субкласс `Adapter` и субкласс `ViewHolder`.

Обязанности класса `ViewHolder` невелики, поэтому начнем с него. `ViewHolder` делает только одно: он удерживает объект `View` (рис. 8.5).

Не много, но для этого `ViewHolder` и существует. Типичный субкласс `ViewHolder` выглядит так (листинг 8.13).

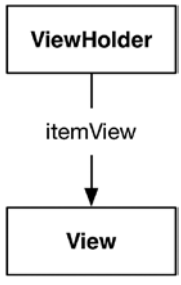


Рис. 8.5. Примитивная работа ViewHolder

Листинг 8.13. Типичный субкласс ViewHolder

```
public class ListRow extends RecyclerView.ViewHolder {
    public ImageView mThumbnail;

    public ListRow(View view) {
        super(view);

        mThumbnail = (ImageView) view.findViewById(R.id.thumbnail);
    }
}
```

После этого вы можете создать объект `ListRow` и работать как с переменной `mThumbnail`, которую вы создали сами, так и с переменной `itemView` — полем, значение которого суперкласс `RecyclerView.ViewHolder` назначает за вас. Поле `itemView` — главная причина для существования `ViewHolder`: в нем хранится ссылка на все представление `View`, переданное `super(view)`.

Листинг 8.14. Типичное использование ViewHolder

```
ListRow row = new ListRow(inflater.inflate(R.layout.list_row, parent, false));
View view = row.itemView;
ImageView thumbnailView = row.mThumbnail;
```

Виджет `RecyclerView` никогда не создает объекты `View`. Он всегда создает объекты `ViewHolder`, которые приносят `itemView` с собой (рис. 8.6).

Для простых представлений `View` класс `ViewHolder` имеет минимум обязанностей. Для более сложных `View` упрощает связывание различных частей `itemView` с `Crime` и повышает его эффективность.

Вы увидите, как работает этот механизм, далее в этой главе, когда мы займемся построением сложного представления `View`.

Адаптеры

Схема на рис. 8.6 несколько упрощена. Виджет `RecyclerView` не создает `ViewHolder` самостоятельно. Вместо этого он обращается с запросом к *адаптеру* (adapter) — объекту контроллера, который находится между `RecyclerView` и набором данных с информацией, которую должен вывести `RecyclerView`.

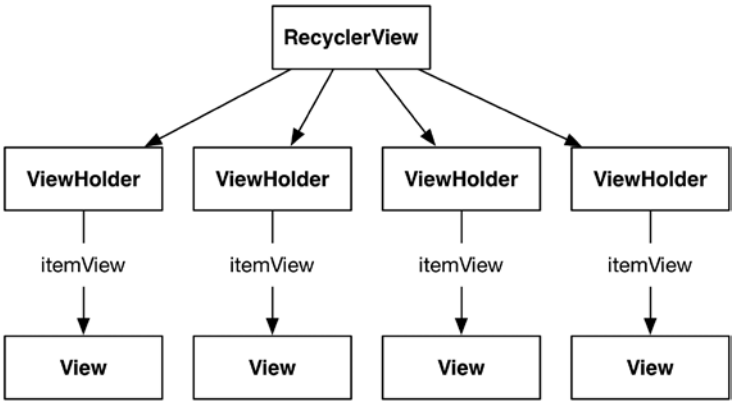


Рис. 8.6. RecyclerView с объектами ViewHolder

Адаптер отвечает за:

- создание необходимых объектов `ViewHolder`;
- связывание `ViewHolder` с данными из уровня модели.

Построение адаптера начинается с определения субкласса `RecyclerView.Adapter`. Ваш субкласс адаптера инкапсулирует список преступлений, полученных от `CrimeLab`.

Когда виджету `RecyclerView` требуется объект представления для отображения, он вступает в диалог со своим адаптером. На рис. 8.7 приведен пример возможного диалога, который может быть инициирован `RecyclerView`.

Сначала `RecyclerView` запрашивает общее количество объектов в списке, для чего вызывает метод `getItemCount()` адаптера.

Затем `RecyclerView` вызывает метод `onCreateViewHolder(ViewGroup, int)` адаптера для создания нового объекта `ViewHolder` вместе с его «полезной нагрузкой»: отображаемым представлением.

Наконец, `RecyclerView` вызывает `onBindViewHolder(ViewHolder, int)`. `RecyclerView` передает этому методу объект `ViewHolder` и позицию. Адаптер получает данные модели для указанной позиции и *связывает* их с представлением `View` объекта `ViewHolder`. Чтобы выполнить связывание, адаптер заполняет `View` в соответствии с данными из объекта модели.

После завершения этого процесса `RecyclerView` помещает элемент списка на экран. Обратите внимание: метод `onCreateViewHolder(ViewGroup, int)` вызывается намного реже, чем `onBindViewHolder(ViewHolder, int)`. После того как будет создано достаточное количество объектов `ViewHolder`, `RecyclerView` перестает вызывать `onCreateViewHolder(...)`. Вместо этого он экономит время и память за счет переработки старых объектов `ViewHolder`.

Использование RecyclerView

Но довольно разговоров, пора взяться за реализацию. Класс `RecyclerView` находится в одной из многочисленных библиотек поддержки Google. Первым шагом

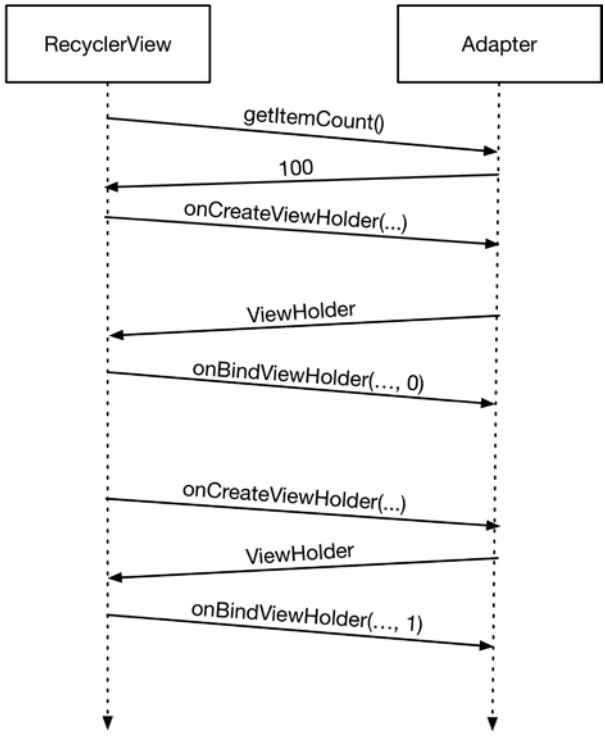


Рис. 8.7. Диалог RecyclerView-Adapter

в использовании `RecyclerView` станет добавление библиотеки `RecyclerView` к зависимостям приложения.

Откройте окно структуры проекта командой File ▸ Project Structure.... Выберите модуль `app` слева, перейдите на вкладку Dependencies. Щелкните на кнопке + и выберите Library dependency, чтобы добавить зависимость.

Найдите и выделите библиотеку `recyclerview-v7`; щелкните на кнопке OK, чтобы добавить библиотеку как зависимость (рис. 8.8).

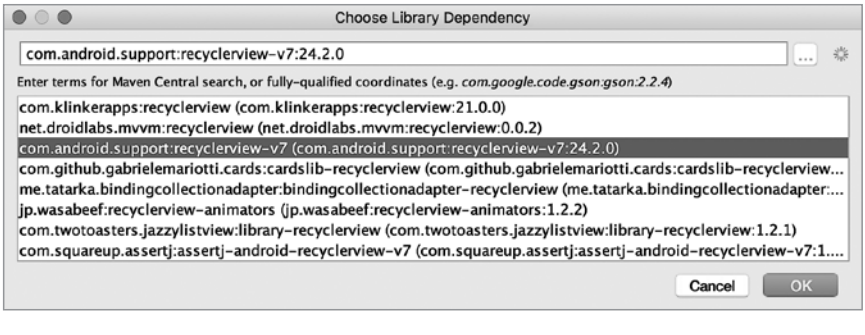


Рис. 8.8. Добавление зависимости для RecyclerView

Виджет `RecyclerView` будет находиться в файле макета `CrimeListFragment`. Сначала необходимо создать файл макета: щелкните правой кнопкой мыши на каталоге res/layout и выберите команду New ▸ Layout resource file. Введите имя `fragment_crime_list` и щелкните на кнопке OK, чтобы создать файл.

Откройте только что созданный файл fragment_crime_list, замените его корневое представление на `RecyclerView` и присвойте ему идентификатор.

Листинг 8.15. Включение RecyclerView в файл макета (fragment_crime_list.xml)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:orientation="vertical"
              android:layout_width="match_parent"
              android:layout_height="match_parent">

</LinearLayout>
<android.support.v7.widget.RecyclerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/crime_recycler_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

Представление `CrimeListFragment` готово; теперь его нужно связать с фрагментом. Измените класс `CrimeListFragment` так, чтобы он использовал этот файл макета и находил `RecyclerView` в файле макета, как показано в листинге 8.16.

Листинг 8.16. Подготовка представления для CrimeListFragment (CrimeListFragment.java)

```
public class CrimeListFragment extends Fragment {

    // Пока пусто
    private RecyclerView mCrimeRecyclerView;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_crime_list, container,
                    false);
        mCrimeRecyclerView = (RecyclerView) view
                .findViewById(R.id.crime_recycler_view);
        mCrimeRecyclerView.setLayoutManager(new LinearLayoutManager
                                            (getActivity()));

        return view;
    }
}
```

Обратите внимание: сразу же после создания виджета `RecyclerView` ему назначается другой объект `LayoutManager`. Это необходимо для работы виджета `RecyclerView`. Если вы забудете предоставить ему объект `LayoutManager`, возникнет ошибка.

Виджет `RecyclerView` не занимается размещением элементов на экране самостоятельно — он поручает эту задачу `LayoutManager`. Объект `LayoutManager` управляет позиционированием элементов, а также определяет поведение прокрутки. Таким образом, при отсутствии `LayoutManager` виджет `RecyclerView` просто погибнет в тщетной попытке что-нибудь сделать.

Вам на выбор предоставляются несколько встроенных вариантов `LayoutManager`; другие варианты можно найти в сторонних библиотеках. Мы будем использовать класс `LinearLayoutManager`, который размещает элементы в вертикальном списке. Позднее вы научитесь использовать `GridLayoutManager` для размещения элементов в виде таблицы.

Запустите приложение. Вы снова видите пустой экран, но сейчас перед вами пустой виджет `RecyclerView`. Объекты `Crime` остаются невидимыми до тех пор, пока не будут определены реализации `Adapter` и `ViewHolder`.

Отображаемое представление

Каждый элемент, отображаемый в `RecyclerView`, имеет собственную иерархию представлений – по аналогии с тем, как `CrimeFragment` имеет иерархию представлений для всего экрана. Новый макет для представления элемента списка создается точно так же, как для представления активности или фрагмента. В окне инструментов Project щелкните правой кнопкой мыши на каталоге res/layout и выберите команду New ▸ Layout resource file. В открывшемся диалоговом окне введите имя list_item_crime и щелкните на кнопке OK.

Добавьте в файл макета два виджета `TextView` (рис. 8.9).

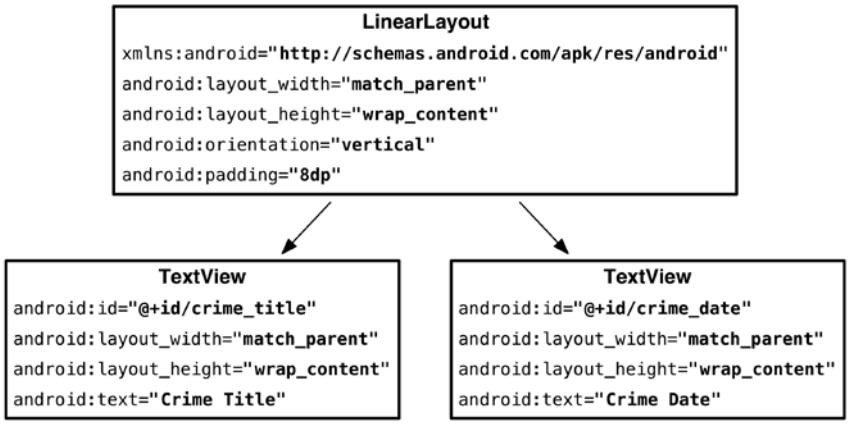


Рис. 8.9. Обновление файла макета элемента списка (list_item_crime.xml)

В области предварительного просмотра видно, что в результате была создана ровно одна строка. Вскоре вы увидите, как `RecyclerView` создает строки за вас.

Реализация адаптера и ViewHolder

Теперь нужно определить класс `ViewHolder`, который будет заполнять ваш макет. Он будет создан как внутренний класс `CrimeListFragment`.

Листинг 8.17. Простая реализация ViewHolder (CrimeListFragment.java)

```
public class CrimeListFragment extends Fragment {
    ...
     private class CrimeHolder extends RecyclerView.ViewHolder {
        public CrimeHolder(LayoutInflater inflater, ViewGroup parent) {
            super(inflater.inflate(R.layout.list_item_crime, parent, false));
        }
    }
}
```

В конструкторе `CrimeHolder` происходит заполнение list_item_crime.xml. Вызов напрямую передается `super(…)`, конструктору `ViewHolder`. Базовый класс `ViewHolder` хранит иерархию представлений fragment_crime_list.xml. Если вам понадобится эта иерархия представлений, вы можете взять ее из поля `itemView` класса `ViewHolder`.

Мы создали минимальную заготовку `CrimeHolder`. Далее в этой главе обязанности `CrimeHolder` будут расширены.

После определения `ViewHolder` создайте адаптер.

Листинг 8.18. Начало работы над адаптером (CrimeListFragment.java)

```
public class CrimeListFragment extends Fragment {
    ...

    private class CrimeAdapter extends RecyclerView.Adapter<CrimeHolder> {

        private List<Crime> mCrimes;

        public CrimeAdapter(List<Crime> crimes) {
            mCrimes = crimes;
        }
    }
}
```

(Код в листинге 8.18 не компилируется. Вскоре мы исправим этот недостаток.)

Класс `RecyclerView` взаимодействует с адаптером, когда требуется создать новый объект `ViewHolder` или связать существующий объект `ViewHolder` с объектом `Crime`. Он обращается за помощью к адаптеру, вызывая его метод. Сам виджет `RecyclerView` ничего не знает об объекте `Crime`, но адаптер располагает полной информацией о `Crime`.

Затем реализуйте три переопределения метода `CrimeAdapter`. (Эти переопределения можно сгенерировать автоматически: наведите курсор на `extends`, нажмите Option+Return (Alt+Enter), выберите Implement methods и щелкните на кнопке OK. Далее остается лишь заполнить тела методов.)

Листинг 8.19. Реализация методов CrimeAdapter (CrimeListFragment.java)

```
    private class CrimeAdapter extends RecyclerView.Adapter<CrimeHolder> {
        ...
        @Override
        public CrimeHolder onCreateViewHolder(ViewGroup parent, int viewType) {
            LayoutInflater layoutInflater = LayoutInflater.from(getActivity());
            return new CrimeHolder(layoutInflater, parent);
        }

        @Override
        public void onBindViewHolder(CrimeHolder holder, int position) {
        }

        @Override
        public int getItemCount() {
            return mCrimes.size();
        }
    }
```

Метод `onCreateViewHolder` вызывается виджетом `RecyclerView`, когда ему требуется новое представление для отображения элемента. В этом методе мы создаем объект `LayoutInflater` и используем его для создания нового объекта `CrimeHolder`.

Адаптер должен содержать переопределение `onBindViewHolder(…)`, но пока его можно оставить пустым. Вскоре мы к нему вернемся.

Итак, адаптер готов; остается связать его с `RecyclerView`. Реализуйте метод `updateUI`, который настраивает пользовательский интерфейс `CrimeListFragment`. Пока он создает объект `CrimeAdapter` и назначает его `RecyclerView`.

Листинг 8.20. Подготовка адаптера (CrimeListFragment.java)

```
public class CrimeListFragment extends Fragment {

    private RecyclerView mCrimeRecyclerView;
    private CrimeAdapter mAdapter;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_crime_list, container,
                    false);
        mCrimeRecyclerView = (RecyclerView) view
                .findViewById(R.id.crime_recycler_view);
        mCrimeRecyclerView.setLayoutManager(new LinearLayoutManager
                                            (getActivity()));

        updateUI();

        return view;
    }

    private void updateUI() {
        CrimeLab crimeLab = CrimeLab.get(getActivity());
        List<Crime> crimes = crimeLab.getCrimes();
```

```
        mAdapter = new CrimeAdapter(crimes);
        mCrimeRecyclerView.setAdapter(mAdapter);
    }
    ...
}
```

В следующих главах метод `updateUI()` будет расширяться по мере усложнения пользовательского интерфейса.

Запустите приложение CriminalIntent и прокрутите новый список `RecyclerView`, который должен выглядеть примерно так, как показано на рис. 8.10.

Рис. 8.10. Список объектов Crime

Хмм... Выглядит немного однообразно. Прокрутив список, вы увидите, как на экране появляются новые одинаковые представления.

На приведенном снимке экрана выводятся 11 строк; это означает, что метод `onCreateViewHolder(…)` был вызван 11 раз. При прокрутке списка могут быть созданы еще несколько объектов `CrimeHolders`, но в какой-то момент виджет `RecyclerView` перестанет создавать новые объекты `CrimeHolder`. Вместо этого он будет заново использовать старые объекты `CrimeHolder`, уходящие за верхний край экрана.

Пока все строки не отличаются друг от друга. На следующем шаге мы заполним повторно используемые объекты `CrimeHolder` свежей информацией, для чего применим связывание.

Связывание с элементами списка

Связыванием (binding) называется подключение кода Java (например, данных модели в `Crime` или слушателей щелчков) к виджету. До сих пор во всех упражнениях, приводившихся в книге, связывание выполнялось при каждом заполнении представления. В таком случае выделять эту работу в отдельный метод бессмысленно. Но теперь, когда представления используются повторно, лучше выполнять создание в одном месте, а связывание в другом.

Весь код, выполняющий реальную работу по связыванию, располагается в `CrimeHolder`. Все начинается с извлечения всех виджетов, которые представляют для вас интерес. Это должно происходить только один раз, поэтому этот код уместно разместить в конструкторе.

Листинг 8.21. Извлечение представлений в конструкторе

```
private class CrimeHolder extends RecyclerView.ViewHolder {

    private TextView mTitleTextView;
    private TextView mDateTextView;

    public CrimeHolder(LayoutInflater inflater, ViewGroup parent) {
        super(inflater.inflate(R.layout.list_item_crime, parent, false));

        mTitleTextView = (TextView) itemView.findViewById(R.id.crime_title);
        mDateTextView = (TextView) itemView.findViewById(R.id.crime_date);
    }
}
```

Объекту `CrimeHolder` также потребуется метод `bind(Crime)`. Он будет вызываться каждый раз, когда в `CrimeHolder` должен отображаться новый объект `Crime`. Сначала добавляется метод `bind(Crime)`.

Листинг 8.22. Метод bind(Crime) (CrimeListFragment.java)

```
private class CrimeHolder extends RecyclerView.ViewHolder {

    private Crime mCrime;
    ...
    public void bind(Crime crime) {
        mCrime = crime;
        mTitleTextView.setText(mCrime.getTitle());
        mDateTextView.setText(mCrime.getDate().toString());
    }
}
```

Получив объект `Crime`, `CrimeHolder` обновит виджеты `TextView` с описанием и датой в соответствии с состоянием `Crime`.

Затем только что созданный метод `bind(Crime)` будет вызываться каждый раз, когда `RecyclerView` потребует связать заданный объект `CrimeHolder` с объектом конкретного преступления.

Листинг 8.23. Вызов метода bind(Crime) (CrimeListFragment.java)

```
private class CrimeAdapter extends RecyclerView.Adapter<CrimeHolder> {
    ...
    @Override
    public void onBindViewHolder(CrimeHolder holder, int position) {
        Crime crime = mCrimes.get(position);
        holder.bind(crime);
    }
    ...
}
```

Запустите приложение CriminalIntent. Во всех видимых элементах `CrimeHolder` должны отображаться разные объекты `Crime` (рис. 8.11).

Рис. 8.11. Новый список

Если прокрутить список вверх, анимация прокрутки идет плавно, как по маслу. Этот эффект является прямым следствием компактности и эффективности `onBindViewHolder(…)`; этот метод выполняет лишь минимально необходимый объем работы.

Стремитесь к тому, чтобы ваша реализация `onBindViewHolder(…)` была как можно более эффективной. В противном случае анимация прокрутки будет запинаться и идти рывками, словно плохо смазанный механизм.

Щелчки на элементах списка

Вместе с `RecyclerView` вы получаете приятное бесплатное дополнение: CriminalIntent теперь может реагировать на касания элементов списка. В главе 10 при касании объекта `Crime` в списке будет запускаться представление детализации. А пока ограничимся простым отображением уведомления `Toast`.

Как вы, возможно, заметили, виджет `RecyclerView` при всей своей мощи и широте возможностей берет на себя минимум реальных обязанностей (нам остается только завидовать). То же происходит и в этом случае: обработкой событий касания в основном придется заниматься вам. `RecyclerView` может передавать вам низкоуровневые события касания, если они вам нужны. Впрочем, в большинстве случаев это излишне.

Вместо этого можно обрабатывать касания так, как это обычно делается: назначением обработчика `OnClickListener`. Так как каждое представление `View` связывается с некоторым `ViewHolder`, вы можете сделать объект `ViewHolder` реализацией `OnClickListener` для своего `View`.

Внесите изменения в класс `CrimeHolder` для обработки касаний в строках.

Листинг 8.24. Обработка касаний в CrimeHolder (CrimeListFragment.java)

```
private class CrimeHolder extends RecyclerView.ViewHolder
        implements View.OnClickListener {
    ...
    public CrimeHolder(LayoutInflater inflater, ViewGroup parent) {
        super(inflater.inflate(R.layout.list_item_crime, parent, false));
        itemView.setOnClickListener(this);
        ...
    }
    ...
    @Override
    public void onClick(View view) {
        Toast.makeText(getActivity(),
                mCrime.getTitle() + " clicked!", Toast.LENGTH_SHORT)
                .show();
    }
}
```

В листинге 8.24 сам объект `CrimeHolder` реализует интерфейс `OnClickListener`. Для `itemView` — представления `View` всей строки — `CrimeHolder` назначается получателем событий щелчка.

Запустите приложение CriminalIntent и коснитесь строки в списке. На экране появится уведомление `Toast` с сообщением о касании.

Для любознательных: ListView и GridView

Базовый вариант ОС Android включает классы `ListView`, `GridView` и `Adapter`. До выхода Android 5.0 эти классы считались наиболее правильным способом создания разнообразных списков или таблиц.

API этих компонентов очень близки к API `RecyclerView`. Класс `ListView` или `GridView` отвечает за прокрутку наборов вариантов, но почти ничего не знает об отдельных элементах списка. `Adapter` отвечает за создание всех представлений в списке. С другой стороны, классы `ListView` и `GridView` не заставляют применять паттерн `ViewHolder` (хотя вы можете — и даже должны — применять его).

Сейчас старые реализации практически вытеснены `RecyclerView` из-за сложностей, связанных с изменением поведения `ListView` или `GridView`.

Например, возможность создания `ListView` с горизонтальной прокруткой не включена в API `ListView`, а ее реализация требует значительной работы. Создание нестандартного макета и реализация прокрутки в `RecyclerView` все равно требует значительной работы, но класс `RecyclerView` строился в расчете на расширение, поэтому все не так плохо.

Другая важная особенность `RecyclerView` — анимация элементов списка. Анимация добавления и удаления элементов в `ListView` или `GridView` — сложная, ненадежная задача. `RecyclerView` существенно упрощает ее, включает несколько встроенных анимаций и позволяет легко настроить эти анимации.

Например, если обнаруживается, что элемент в позиции 0 переместился в позицию 5, перемещение можно анимировать следующим образом:

```
mRecyclerView.getAdapter().notifyItemMoved(0, 5);
```

Для любознательных: синглеты

Паттерн «Синглет», используемый в CrimeLab, очень часто применяется в Android. Синглеты пользуются дурной славой, потому что они открывают возможности для многочисленных злоупотреблений, усложняющих сопровождение кода.

Синглеты часто применяются в Android, так как они живут дольше отдельных фрагментов или активностей. Синглет переживает повороты устройства и существует при перемещении между активностями и фрагментами в приложении.

Синглеты могут стать удобными владельцами объектов модели. Представьте более сложное приложение CriminalIntent с множеством активностей и фрагментов, изменяющих информацию о преступлениях. Когда один контроллер изменяет преступление, как организовать передачу информации об обновленном преступлении другим контроллерам? Если `CrimeLab` является владельцем объектов преступлений и все изменения в них проходят через `CrimeLab`, распространение информации об изменениях значительно упрощается. При передаче управления между контроллерами можно передавать идентификатор преступления, а каждый контроллер извлекает полный объект преступления из `CrimeLab` по идентификатору.

Тем не менее у синглетов есть свои недостатки. Например, хотя они хорошо подходят для хранения данных с бóльшим сроком жизни, чем у контроллера, у синглетов все же имеется срок жизни. Android в какой-то момент после переключения от приложения может освободить память, и тогда синглеты будут уничтожены

вместе со всеми переменными экземпляров. Синглеты не подходят для долгосрочного хранения данных (в отличие, скажем, от записи файлов на диск или их отправки на веб-сервер).

Синглеты также усложняют модульное тестирование кода. Экземпляры `CrimeLab` трудно заменить их фиктивными версиями, потому что код напрямую вызывает статический метод объекта `CrimeLab`. На практике Android-разработчики обычно решают эту проблему при помощи инструмента, называемого *внедрителем зависимостей*. Он позволяет организовать использование синглетных объектов, сохранив при этом возможность замены их при необходимости.

Как говорилось ранее, синглеты часто используются некорректно. У программиста возникает искушение применять синглеты везде и повсюду, потому что они удобны: к ним можно обращаться откуда угодно и в них можно хранить любую информацию, которая может понадобиться позднее. Но при этом вы не пытаетесь ответить на важные вопросы: где используются эти данные? чем важен этот метод?

Синглет обходит эти вопросы. Любой разработчик, который придет после вас, откроет синглет и обнаружит некое подобие сундука, в который сваливают всякий хлам: батарейки, стяжки для кабелей, старые фотографии. Зачем здесь все это? Убедитесь в том, что все содержимое синглета действительно глобально и для хранения данных в синглете существуют веские причины.

Впрочем, с учетом всего сказанного синглеты являются ключевым компонентом хорошо спланированного приложения Android — при условии правильного использования.

Упражнение. ViewType и RecyclerView

В этом упражнении повышенной сложности вы создадите для `RecyclerView` два типа строк: обычные строки и строки для более серьезных преступлений. Для этого нужно воспользоваться новой функциональностью `ViewType`, поддерживаемой `RecyclerView.Adapter`. Добавьте в объект `Crime` новое свойство `mRequiresPolice` и используйте его для определения того, какое представление следует загружать в `CrimeAdapter`; для этого реализуйте метод `getItemViewType(int)` (*developer.android.com/reference/android/support/v7/widget/RecyclerView.Adapter.html#getItemViewType(int)*).

В методе `onCreateViewHolder(ViewGroup, int)` также необходимо добавить логику, которая будет возвращать разные объекты `ViewHolder` в зависимости от нового значения `viewType`, возвращаемого методом `getItemViewType(int)`. Исходный макет используется для преступлений, не требующих вмешательства полиции, а новый макет с измененным интерфейсом и кнопкой вызова полиции — для самых серьезных случаев.

9 Создание пользовательских интерфейсов с использованием макетов и виджетов

В этой главе мы поближе познакомимся с макетами и виджетами, а также немного украсим список элементов в `RecyclerView`. Вы также узнаете о `ConstraintLayout` — новом механизме управления макетами. На рис. 9.1 показано, как выглядит представление `CrimeListFragment` после того, как текущая версия приложения сделает следующий шаг на пути к совершенству.

Прежде чем серьезно браться за `ConstraintLayout`, необходимо провести небольшую подготовку. В проект нужно добавить изображение наручников с рис. 9.1. Откройте файл решений и найдите каталог 09_LayoutsAndWidgets/CriminalIntent/app/src/main/res. Скопируйте версии ic_solved.png для разных плотностей пикселов в соответствующие папки drawable в вашем проекте. О том, как обратиться к файлу решений, рассказано в разделе «Добавление значка» главы 2.

Рис. 9.1. CriminalIntent — теперь с красивыми картинками

Использование графического конструктора

До настоящего момента мы создавали макеты, вводя разметку XML. В этом разделе мы воспользуемся графическим конструктором.

Откройте файл list_item_crime.xml и выберите вкладку Design в нижней части окна файла.

В центре графического конструктора располагается уже знакомая панель предварительного просмотра. Справа от нее находится область *раскладки* (blueprint) — она выглядит почти так же, как область предварительного просмотра, но в ней отображаются контуры всех представлений. Это может быть полезно тогда, когда вы хотите увидеть размеры всех представлений, а не только отображаемую в них информацию.

Слева располагается *палитра*. На ней размещаются практически все виджеты, которые только можно себе представить, упорядоченные по категориям (рис. 9.2).

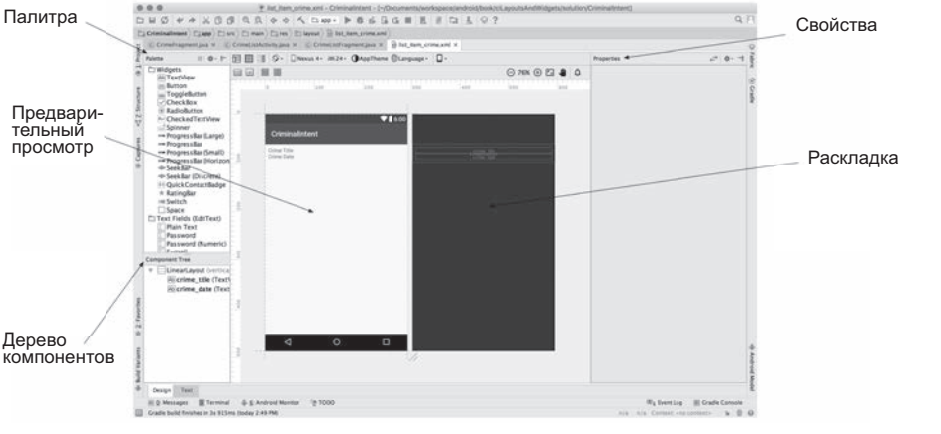


Рис. 9.2. Представления в графическом конструкторе

Справа находится дерево компонентов, которое показывает, как организованы виджеты в макете.

Под деревом компонентов располагается *панель свойств*. На ней можно просматривать и редактировать атрибуты виджета, выделенного в дереве компонентов.

Знакомство с ConstraintLayout

При использовании механизма ограничений макета `ConstraintLayout` вместо применения вложенных макетов в макет включается набор *ограничений*. Ограничение, словно резиновая лента, «притягивает» два визуальных объекта друг к другу. Например, ограничение может связать правый край виджета `ImageView` с правым краем родителя (`ConstraintLayout`), как показано на рис. 9.3. При таком ограничении `ImageView` будет прилегать к правому краю.

Рис. 9.3. Ограничение связывает ImageView с правым краем

Ограничения можно задать для всех четырех сторон `ImageView` (левой, верхней, правой и нижней). Противодействующие ограничения компенсируются, и `ImageView` будет располагаться прямо в центре (рис. 9.4).

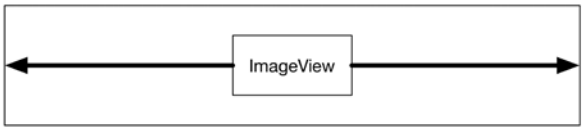


Рис. 9.4. ImageView с противоположными ограничениями

Общая картина выглядит так: чтобы разместить представления в нужном месте `ConstraintLayout`, вы не перетаскиваете их мышью по экрану, а назначаете соответствующие ограничения.

Как насчет определения размеров виджетов? Есть три варианта: предоставить выбор самому виджету (вашему старому знакомому `wrap_content`), решить самостоятельно или разрешить виджету изменять свои размеры в соответствии с ограничениями.

Со всеми этими средствами один контейнер `ConstraintLayout` может вмещать много разных макетов без применения вложения. В этой главе вы узнаете, как использовать ограничения с `list_item_crime`.

Использование ConstraintLayout

Теперь преобразуем list_item_crime.xml для использования `ConstraintLayout`. Щелкните правой кнопкой мыши на корневом узле `LinearLayout` в дереве компонентов и выберите команду Convert LinearLayout to ConstraintLayout (рис. 9.5).

Android Studio выводит диалоговое окно с предложением задать параметры оптимизации преобразования (рис. 9.6). Файл макета list_item_crime очень прост, так что у Android Studio немного возможностей для оптимизации. Оставьте значения по умолчанию и щелкните на кнопке OK.

Наконец, вам будет предложено добавить в проект зависимость от `ConstraintLayout` (рис. 9.7). Поддержка `ConstraintLayout` находится в библиотеке, как и `RecyclerView`. Чтобы использовать ограничения, следует добавить зависимость в файл Gradle. Или же вы можете выбрать кнопку OK в этом окне, и Android Studio сделает все за вас.

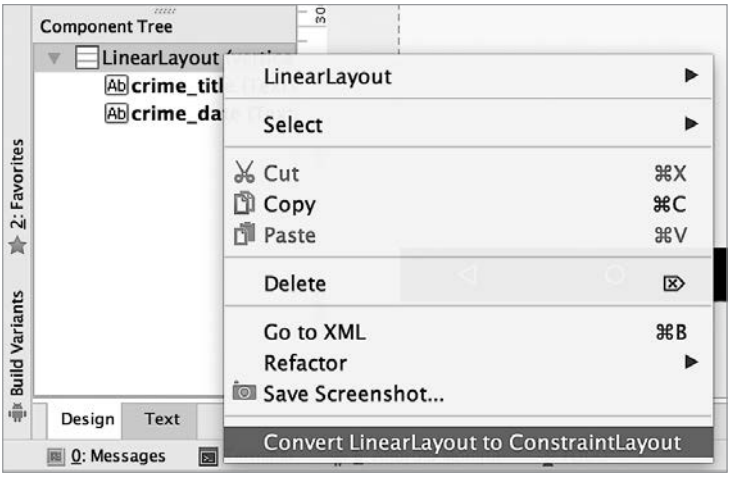


Рис. 9.5. Преобразование корневого представления в ConstraintLayout

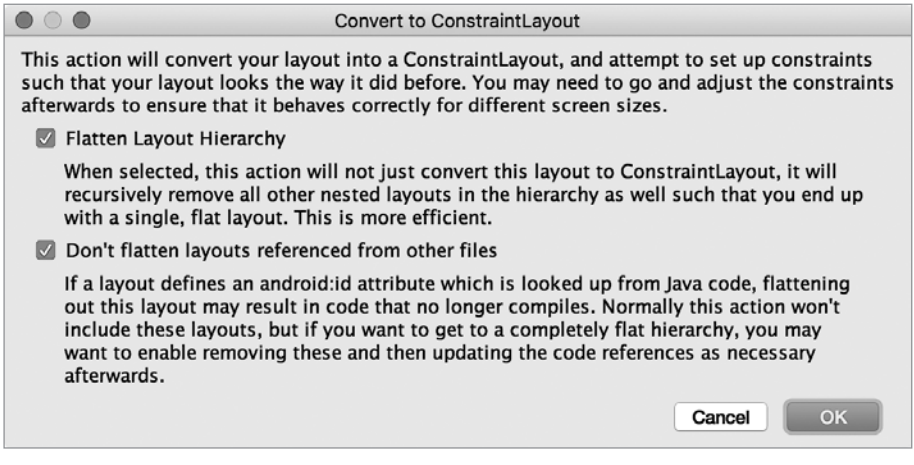


Рис. 9.6. Преобразование с конфигурацией по умолчанию

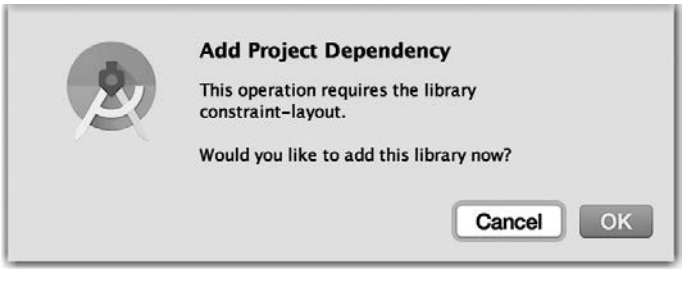


Рис. 9.7. Добавление зависимости ConstraintLayout

Заглянув в файл app/build.gradle, вы увидите, что зависимость действительно была добавлена:

Листинг 9.1. Зависимость ConstraintLayout в проекте (app/build.gradle)

```
dependencies {
    ...
    compile 'com.android.support.constraint:constraint-layout:1.0.0-beta4'
}
```

Элемент `LinearLayout` преобразуется в `ConstraintLayout`.

Графический редактор

На панели в верхней части области предварительного просмотра находится панель инструментов с несколькими кнопками (рис. 9.8).

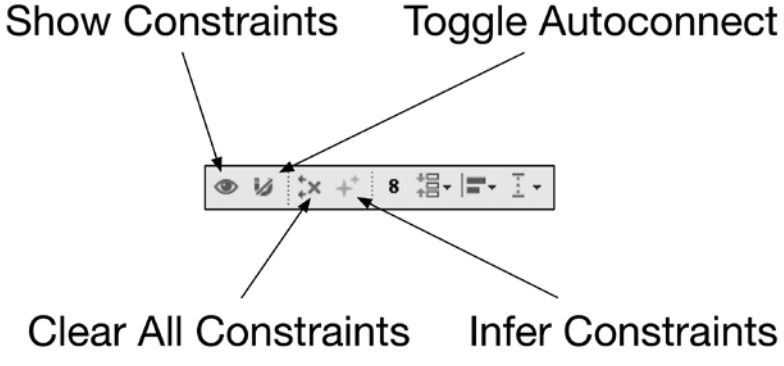


Рис. 9.8. Элементы управления ограничениями

Show Constraints	Кнопка Show Constraints выводит заданные ограничения в режимах предварительного просмотра и раскладки. В одних случаях этот режим просмотра полезен, в других нет. При большом количестве ограничений эта кнопка выводит слишком много информации
Toggle Autoconnect	При включении режима Autoconnect ограничения будут автоматически настраиваться при перетаскивании представлений в область предварительного просмотра. Android Studio старается угадать, какие ограничения должно иметь представление, и создавать их по мере необходимости
Clear All Constraints	Кнопка Clear All Constraints удаляет все ограничения, существующие в файле макета. Вскоре мы воспользуемся этой кнопкой
Infer Constraints	Как и кнопка Toggle Autoconnect, эта кнопка автоматически создает ограничения, но эта функциональность срабатывает только при нажатии кнопки. Функциональность Autoconnect срабатывает каждый раз, когда в файл макета добавляется новое представление

Когда вы преобразуете файл list_item_crime для использования `ConstraintLayout`, среда Android Studio автоматически добавляет ограничения, которые на ее взгляд повторяют поведение старого макета. Но чтобы вы лучше поняли, как работают ограничения, мы сделаем все вручную.

Выберите представление `ConstraintLayout` в дереве компонентов, после чего нажмите кнопку Clear All Constraints на рис. 9.8. В правой верхней части экрана немедленно появляется красный флажок с цифрой 4. Щелкните на нем, чтобы понять, о чем вас предупреждают (рис. 9.9).

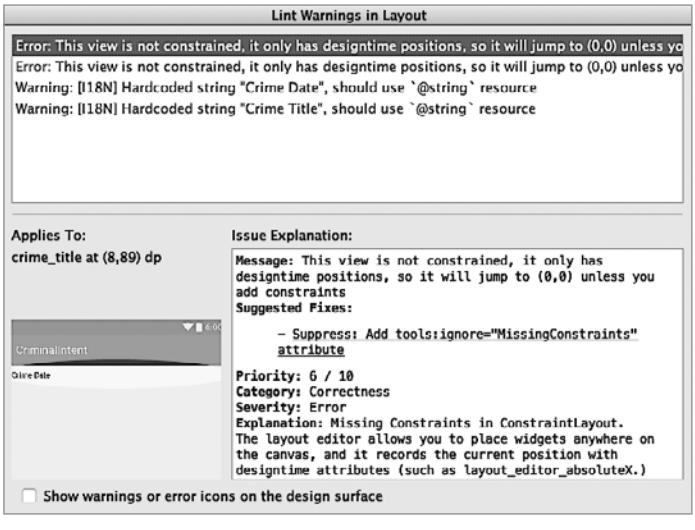


Рис. 9.9. Предупреждения ConstraintLayout

Если заданных ограничений недостаточно, `ConstraintLayout` не сможет точно определить, где разместить представления. Ваши представления `TextView` не имеют ограничений, поэтому для каждого из них выводится предупреждение о том, что оно не будет отображаться в правильном месте во время выполнения.

В этой главе мы добавим необходимые ограничения и избавимся от предупреждений. А во время своей работы следите за предупреждающим индикатором, чтобы избежать непредвиденного поведения во время выполнения.

Освобождение пространства

Сначала необходимо освободить область для размещения — сейчас два представления `TextView` занимают все доступное место, не позволяя разместить что-либо еще. Эти два виджета нужно уменьшить.

Выберите виджет crime_title в дереве компонентов и просмотрите его свойства на панели справа (рис. 9.10).

Вертикальный и горизонтальный размеры `TextView` определяются значениями layout_height и layout_width соответственно. Им можно присвоить одно из трех зна-

чений (рис. 9.11), каждое из которых соответствует значению `layout_width` или `layout_height`.

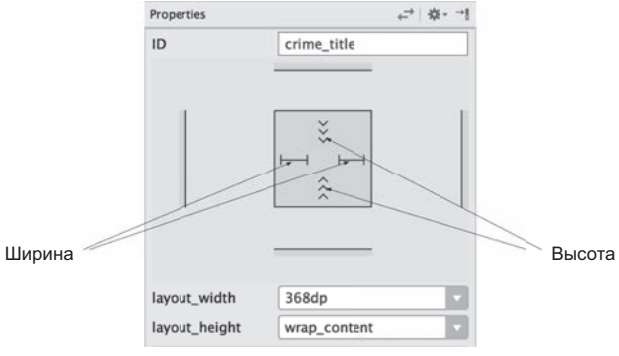


Рис. 9.10. Свойства TextView

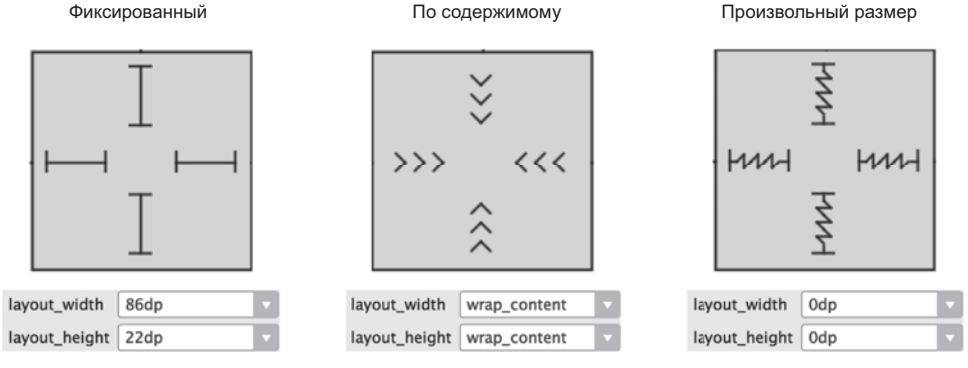


Рис. 9.11. Три варианта определения размера представлений

Таблица 9.1. Типы значений, определяющих размеры представлений

Тип	Значение	Применение
Фиксированный	Xdp	Явно задает размер представления (который не может изменяться) в единицах dp. (Подробнее об этих единицах далее в этой главе.)
По содержимому	wrap_content	Представление получает «желаемый» размер. Для TextView это означает, что представлению будет назначен минимальный размер, необходимый для вывода содержимого
Произвольный размер	0dp	Разрешает изменение размеров представления для соблюдения заданных ограничений

Представлениям `crime_title` и `crime_date` назначается большая фиксированная ширина, поэтому они занимают весь экран. Отрегулируйте высоту и ширину этих виджетов. Пока `crime_title` остается выбранным в дереве компонентов, щелкайте на значении ширины, пока в поле не появится значение wrap_content. При необходимости изменяйте высоту, пока в поле высоты также не появится значение wrap_content (рис. 9.12).

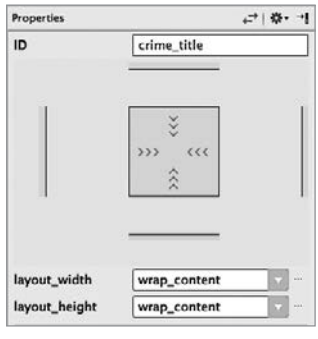


Рис. 9.12. Настройка ширины и высоты

Повторите этот процесс с виджетом `crime_date`, чтобы задать его ширину и высоту. Теперь два виджета перекрываются, но заметно уменьшились в размерах (рис. 9.13).

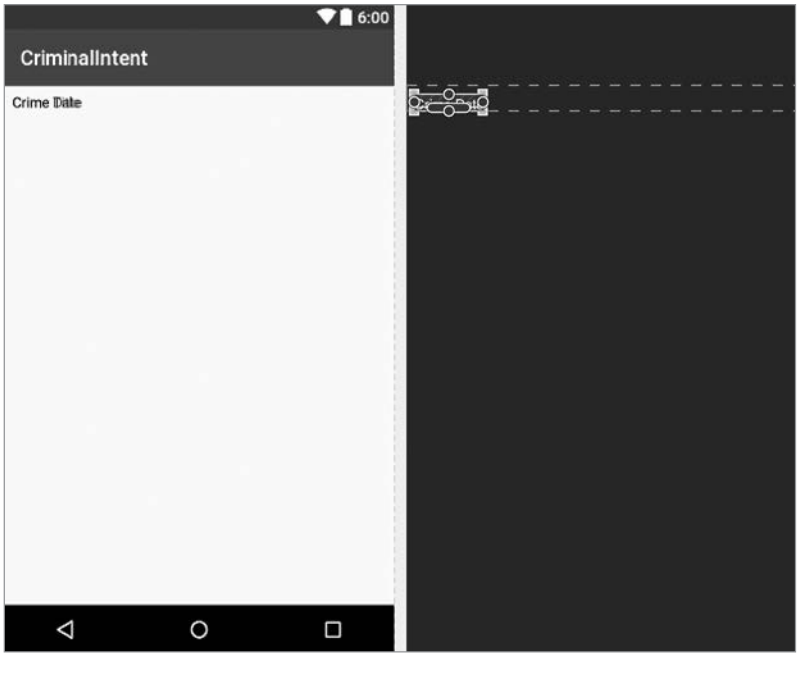


Рис. 9.13. Перекрывающиеся виджеты TextView

Добавление виджетов

В макете освободилось место, и в него можно добавить изображение наручников. Добавьте в файл макета виджет `ImageView`. Найдите его в палитре (рис. 9.14) и перетащите в дерево компонентов как потомка `ConstraintLayout`, прямо под `crime_date`.

Рис. 9.14. Поиск ImageView

Выберите в диалоговом окне ic_solved в качестве ресурса для `ImageView` (рис. 9.15). Изображение будет использоваться для пометки раскрытых преступлений.

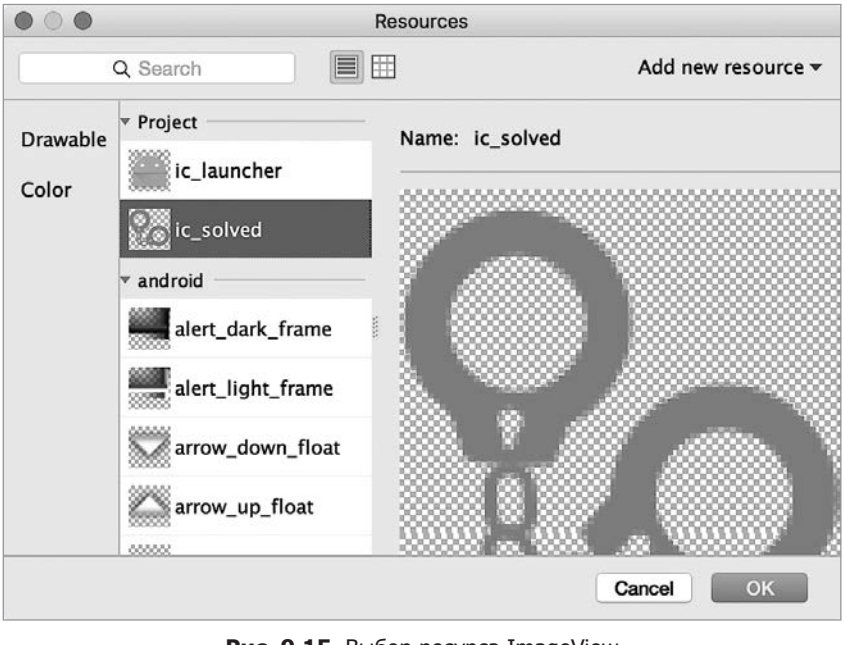


Рис. 9.15. Выбор ресурса ImageView

Виджет `ImageView` теперь является частью макета, но не имеет ограничений. Таким образом, хотя в графическом редакторе ему назначена позиция, в действительности это ничего не значит.

Пришло время добавить ограничения. Щелкните на `ImageView` в области предварительного просмотра или в дереве компонентов. Рядом с `ImageView` появятся маркеры ограничений (рис. 9.16).

Рис. 9.16. Маркеры ограничений для ImageView

Виджет `ImageView` должен прикрепляться к правому краю представления. Для этого нужно определить ограничения для верхней, правой и нижней стороны `ImageView`.

Сначала мы создадим ограничение между верхней стороной `ImageView` и верхней стороной `ConstraintLayout`. Верхнюю сторону `ConstraintLayout` разглядеть непросто, но она находится сразу же под синей панелью инструментов `CriminalIntent`. В режиме предварительного просмотра перетащите верхний маркер ограничения с `ImageView` на верхнюю сторону `ConstraintLayou`. При перетаскивании придется немного сместиться вправо, потому что изображение находится у верхнего края `ConstraintLayout`. Когда маркер ограничения окрасится в зеленый цвет, а на экране появляется подсказка (рис. 9.17), отпустите кнопку мыши.

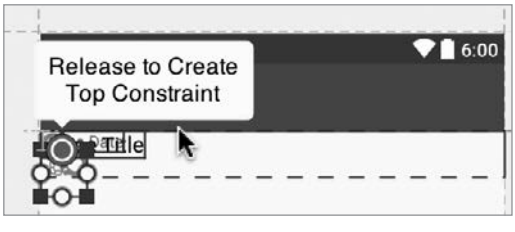


Рис. 9.17. Создание ограничения для верхней стороны ImageView

Будьте внимательны и не щелкайте, когда указатель мыши имеет форму угла — это приведет к изменению размеров `ImageView`. Также следите за тем, чтобы ограничение не соединилось случайно с одним из виджетов `TextView`. Если это произойдет, щелкните на маркере ограничения, чтобы удалить ошибочное ограничение, и повторите попытку.

Когда вы отпускаете кнопку мыши и задаете ограничение, представление фиксируется в позиции, соответствующей новому ограничению. Так происходит расстановка представлений в `ConstraintLayout` — вы создаете и уничтожаете ограничения.

Чтобы убедиться в том, что у `ImageView` существует нужное ограничение (верхняя сторона `ImageView` связана с верхней стороной `ConstraintLayout`), наведите на `ImageView` указатель мыши. Результат должен выглядеть так, как показано на рис. 9.18.

Проделайте то же самое с маркером ограничения для нижней стороны: перетащите его с `ImageView` на нижнюю сторону корневого представления (следите за тем,

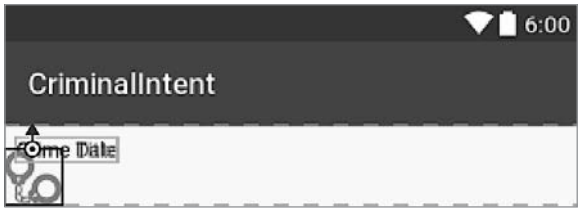


Рис. 9.18. ImageView с ограничением для верхней стороны

чтобы случайно не присоединить его к `TextView`). И снова перетаскивать следует к центру корневого представления, а потом немного вниз (рис. 9.19).

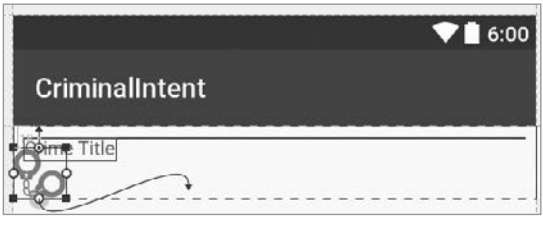


Рис. 9.19. ImageView в процессе

Наконец, перетащите правый маркер ограничения с `ImageView` на правую сторону корневого представления. После этого все необходимые ограничения должны быть успешно созданы. Если навести указатель мыши на `ImageView`, на экране должна появиться информация о них. Результат выглядит так, как показано на рис. 9.20.

Рис. 9.20. ImageView с тремя ограничениями

Внутренние механизмы ConstraintLayout

Все изменения, вносимые в графическом редакторе, отражаются в разметке XML незаметно для разработчика. При этом ничто не мешает напрямую редактировать разметку XML, относящуюся к `ConstraintLayout`, но работать с графическим редактором обычно удобнее, потому что разметка `ConstraintLayout` занимает гораздо больше места, чем разметка других типов `ViewGroup`.

Чтобы увидеть, что произошло с разметкой XML при создании трех ограничений для `ImageView`, переключитесь в текстовый режим.

Листинг 9.2. Новые ограничения ImageView в XML (layout/list_item_crime.xml)

```
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    ...
    <ImageView
        android:id="@+id/imageView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:srcCompat="@drawable/ic_solved"
        android:layout_marginTop="16dp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        android:layout_marginBottom="16dp"
        android:layout_marginEnd="16dp"
        app:layout_constraintRight_toRightOf="parent"/>

</android.support.constraint.ConstraintLayout>
```

Присмотритесь к верхнему ограничению:

```
app:layout_constraintTop_toTopOf="parent"
```

Атрибут начинается с префикса `layout_`. Все атрибуты, начинающиеся с этого префикса, называются *параметрами макета* (layout parameters). В отличие от других атрибутов параметры макета представляют собой инструкции для родителя виджета, а не для самого виджета. Они сообщают родительскому макету, как он должен расположить дочерний элемент внутри себя. Ранее вам уже встречались примеры параметров макетов `layout_width` и `layout_height`.

Имя ограничения `constraintTop` указывает на то, что это ограничение относится к верхней стороне `ImageView`.

Наконец, атрибут завершается суффиксом `toTopOf="parent"`. Из него следует, что ограничение связывается с верхней стороной родителя. Родителем в данном случае является `ConstraintLayout`.

А теперь оставим позади низкоуровневую разметку XML и вернемся к графическому редактору.

Редактирование свойств

Элемент `ImageView` теперь расположен правильно. Пора сделать следующий шаг: разместить и задать размеры заголовочного виджета `TextView`.

Сначала выделите `crime_date` в дереве компонентов и перетащите его в сторону (рис. 9.21). Не забудьте, что любые изменения, внесенные в позицию в области предварительного просмотра, не будут представлены во время работы приложения. Во время выполнения останутся только ограничения.

Рис. 9.21. Дата, в сторону!

Теперь выделите crime_title в дереве компонентов. Это также приведет к выделению `crime_title` в области предварительного просмотра.

Виджет `crime_title` должен находиться в левой верхней части макета, в левой части нового представления `ImageView`.

Для этого необходимо задать три ограничения:

- связать левую сторону представления с левой стороной родителя, с полем 16 dp;
- связать верхнюю сторону представления с верхней стороной родителя, с полем 16 dp;
- связать правую сторону представления с правой стороной родителя, с полем 8 dp.

Измените ваш макет и создайте все перечисленные ограничения. (На момент написания книги найти правильное место для щелчка было непросто. Попробуйте щелкнуть внутри `TextView` и помните, что вы всегда можете нажать Command+Z (Ctrl+Z), чтобы отменить выполненное действие и попробовать снова.)

Убедитесь в том, что ваши ограничения не отличаются от рис. 9.22. (У выбранного виджета ограничения помечаются волнистыми линиями.)

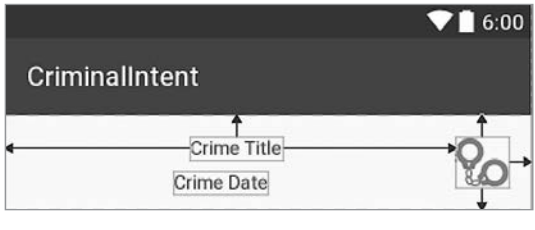


Рис. 9.22. Ограничения заголовочного виджета TextView

Когда вы щелкнете на виджете `TextView`, вы увидите овальную область, которой не было у `ImageView`. У `TextView` имеется дополнительный якорь ограничения, используемый для выравнивания текста. В этой главе он использоваться не будет, но теперь вы знаете, что это такое.

После того как ограничения будут созданы, заголовочный виджет `TextView` можно восстановить во всей красе. Задайте его горизонтальному размеру значение `0dp`,

чтобы заголовочный виджет `TextView` заполнил все место, свободное с учетом имеющихся ограничений. Выберите для вертикального размера значение wrap_content, если это не было сделано ранее; в этом случае для `TextView` будет выбрана минимальная высота, необходимая для вывода списка. Убедитесь в том, что ваши изменения соответствуют приведенным на рис. 9.23.

Теперь добавьте ограничения для виджета `TextView` с датой. Выберите crime_date в дереве компонентов. Нужно задать три ограничения:

- связать левую сторону представления с левой стороной родителя, с полем 16 dp;
- связать верхнюю сторону представления с нижней стороной заголовка, с полем 8 dp;
- связать правую сторону представления с левой стороной нового виджета `ImageView`, с полем 8 dp.

После добавления ограничений настройте свойства `TextView`. В поле ширины виджета `TextView` с датой выбирается значение 0dp, а в поле высоты — wrap_content, как и у заголовочного виджета `TextView`. Убедитесь в том, что ваши настройки соответствуют приведенным на рис. 9.24.

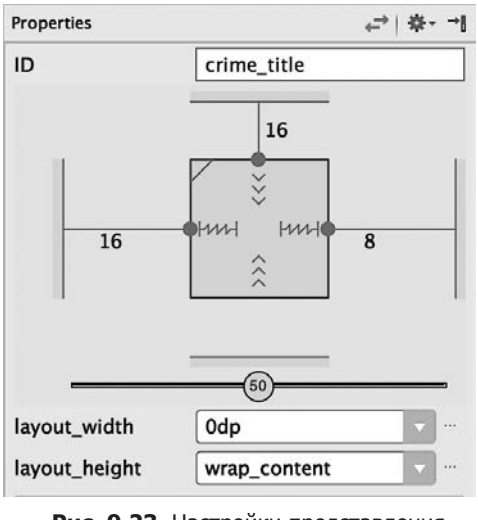


Рис. 9.23. Настройки представления crime_title

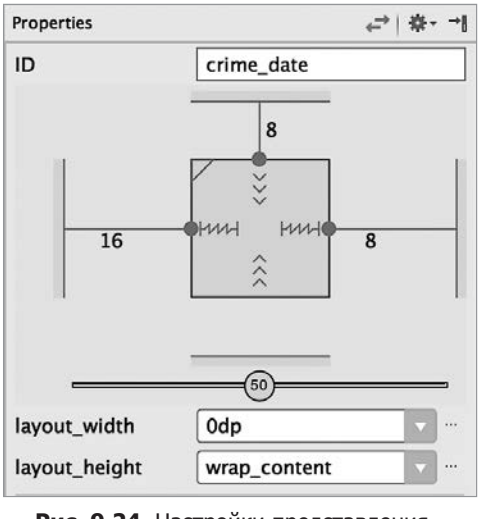


Рис. 9.24. Настройки представления crime_date

В результате макет в области предварительного просмотра должен выглядеть так, как показано на рис. 9.1 в начале главы.

Запустите приложение CriminalIntent и убедитесь в том, что все три компонента аккуратно выстраиваются в каждой строке `RecyclerView` (рис. 9.25).

Рис. 9.25. Каждая строка содержит три представления

Динамическое поведение элемента списка

После того как для макета будут определены необходимые ограничения, следует обновить виджет `ImageView`, чтобы изображение наручников выводилось только для раскрытых преступлений.

Сначала обновите идентификатор `ImageView`. Когда вы добавили виджет `ImageView` в `ConstraintLayout`, ему было присвоено имя по умолчанию — не слишком содержательное. Выберите виджет `ImageView` в файле list_item_crime.xml, затем в окне свойств замените текущее значение атрибута ID на crime_solved (рис. 9.26). Android Studio спросит, нужно ли обновить все вхождения измененного идентификатора; выберите ответ Yes.

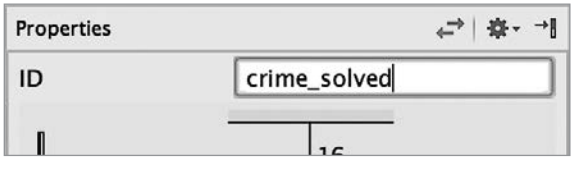


Рис. 9.26. Изменение идентификатора виджета

Листинг 9.3. Изменение видимости изображения (CrimeListFragment.java)

```
private class CrimeHolder extends RecyclerView.ViewHolder
        implements View.OnClickListener {
    ...
    private TextView mDateTextView;
    private ImageView mSolvedImageView;

    public CrimeHolder(LayoutInflater inflater, ViewGroup parent) {
        super(inflater.inflate(R.layout.list_item_crime, parent, false));
        itemView.setOnClickListener(this);
        mTitleTextView = (TextView) itemView.findViewById(R.id.crime_title);
        mDateTextView = (TextView) itemView.findViewById(R.id.crime_date);
        mSolvedImageView = (ImageView) itemView.findViewById(R.id.crime_solved);
    }

    public void bind(Crime crime) {
        mCrime = crime;
        mTitleTextView.setText(mCrime.getTitle());
        mDateTextView.setText(mCrime.getDate().toString());
        mSolvedImageView.setVisibility(crime.isSolved() ? View.VISIBLE :
                        View.GONE);
    }
    ...
}
```

Запустите приложение CriminalIntent и убедитесь в том, что в каждой строке теперь выводится изображение наручников.

Подробнее об атрибутах макетов

Давайте добавим еще несколько настроек в list_item_crime.xml, а заодно ответим на некоторые запоздалые вопросы о виджетах и атрибутах.

Вернитесь к представлению Design файла list_item_crime.xml. Выберите виджет crime_title и измените некоторые атрибуты на панели свойств.

Щелкните на стрелке рядом с textAppearance, чтобы открыть атрибуты текста и шрифта. Измените значение атрибута textColor на @android:color/black (рис. 9.27).

Рис. 9.27. Изменение цвета текста в заголовке

Присвойте атрибуту textSize значение 18sp. Запустите приложение CriminalIntent; вы удивитесь, насколько лучше все смотрится после небольшой доработки.

Плотности пикселов, dp и sp

В файле fragment_crime.xml значение атрибута margin задается в единицах dp. Вы уже видели эти единицы в макетах; пришло время узнать, что они собой представляют.

Иногда значения атрибутов представления задаются в конкретных размерах (как правило, в пикселах, но иногда в пунктах, миллиметрах или дюймах). Чаще всего этот способ используется для атрибутов размера текста, полей и отступов. Размер текста равен высоте текста в пикселах на экране устройства. Поля задают расстояния между представлениями, а отступы задают расстояние между внешней границей представления и его содержимым.

Как было показано в разделе «Добавление значка» в главе 2, Android автоматически масштабирует изображения для разных плотностей пикселов экрана, используя имена каталогов, уточненные спецификаторами плотности (например, drawable-xhdpi). Но что произойдет, если ваши изображения масштабируются, а поля — нет? Или если пользователь выберет размер текста больше стандартного?

Для решения таких проблем в Android поддерживаются единицы, не зависящие от плотности пикселов; используя их, можно получить одинаковые размеры на экранах с разными плотностями. Android преобразует эти единицы в пикселы во время выполнения, так что вам не придется самостоятельно заниматься сложными вычислениями (рис. 9.28).

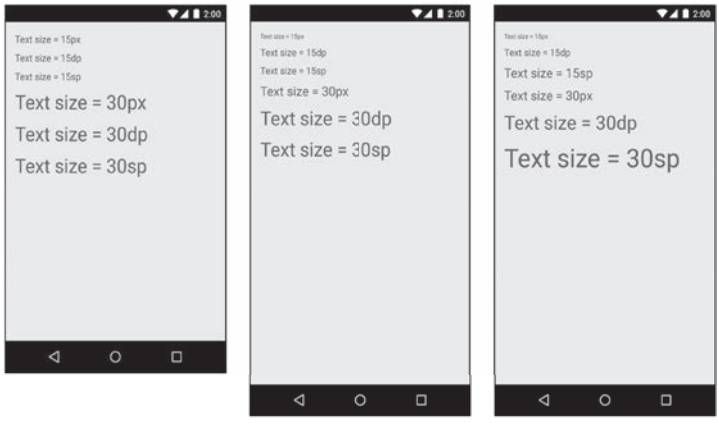


Рис. 9.28. Использование единиц устройства на TextView

- px — сокращение от «пиксел». Один пиксел всегда соответствует одному пикселу на экране независимо от плотности пикселов. Так как пикселы некорректно масштабируются при изменении плотности экрана, пользоваться этими единицами не рекомендуется.
- dp (или dip) — сокращение от «density-independent pixel» (пикселы, не зависящие от плотности); произносится «дип». Обычно эти единицы используются для полей, отступов и всего остального, для чего обычно задаются размеры в пикселах. На экранах с более высокой плотностью единицы dp разворачиваются в большее количество экранных пикселов. Одна единица dp всегда равна 1/160 дюйма на экране устройства. Размер будет одинаковым независимо от плотности пикселов.

- sp — сокращение от «scale-independent pixel» (пикселы, не зависящие от масштаба). Эти единицы, не зависящие от плотности пикселов устройства, также учитывают выбранный пользователем размер шрифта. Единицы sp почти всегда используются для назначения размера текста.
- pt, mm, in — масштабируемые единицы (как и dp), позволяющие задавать размеры интерфейсных элементов в пунктах (1/72 дюйма), миллиметрах или дюймах. Тем не менее мы не рекомендуем их использовать: не все устройства правильно настроены для правильного масштабирования этих единиц.

На практике и в этой книге почти исключительно используются только единицы dp и sp. Android преобразует эти значения в пикселы во время выполнения.

Поля и отступы

В приложениях GeoQuiz и CriminalIntent виджетам назначаются атрибуты `margin` и `padding`. Начинающие разработчики иногда путают эти атрибуты, определяющие *поля* и *отступы* соответственно. Теперь, когда вы понимаете, что такое параметр макета, будет проще объяснить, чем они различаются.

Атрибуты `margin` являются параметрами макета; они определяют расстояние между виджетами. Так как виджет располагает информацией только о самом себе, за соблюдение полей отвечает родитель виджета.

Напротив, отступ не является параметром макета. Атрибут `android:padding` сообщает виджету, с каким превышением размера содержимого он должен прорисовывать себя. Допустим, вы хотите заметно увеличить размеры кнопки даты без изменения размера текста (рис. 9.29).

Добавьте в `Button` следующий атрибут:

Листинг 9.4. Назначение отступов (fragment_crime.xml)

```
<Button android:id="@+id/crime_date"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:layout_marginLeft="16dp"
  android:layout_marginRight="16dp"
  android:padding="80dp" />
```

Не забудьте удалить этот атрибут, прежде чем продолжать работу.

Стили, темы и атрибуты тем

Стиль (style) представляет собой ресурс XML, который содержит атрибуты, описывающие внешний вид и поведение виджета. Например, ниже приведен ресурс стиля, который настраивает виджет на использование увеличенного размера текста:

```
<style name="BigTextStyle">
  <item name="android:textSize">20sp</item>
  <item name="android:padding">3dp</item>
</style>
```

Рис. 9.29. Для любителей больших кнопок

Вы можете создавать собственные стили (этим мы займемся в главе 22). Они добавляются в файл стилей из каталога res/values/, а ссылки на них в макетах выглядят так: `@style/my_own_style`.

Еще раз взгляните на виджеты `TextView` из файла fragment_crime.xml; каждый виджет имеет атрибут `style`, который ссылается на стиль, созданный Android. С этим конкретным стилем виджеты `TextView` выглядят как разделители списка, а берется он из темы приложения. *Тема* (theme) представляет собой набор стилей. Со структурной точки зрения тема сама является ресурсом стиля, атрибуты которого ссылаются на другие ресурсы стилей.

Android предоставляет платформенные темы, которые могут использоваться вашими приложениями. При создании CriminalIntent мастер назначает тему приложения, которая определяется тегом `application` в манифесте.

Стиль из темы приложения можно применить к виджету при помощи ссылки на *атрибут темы* (theme attribute reference). Именно это мы делаем в файле fragment_crime.xml, используя значение `?android:listSeparatorTextViewStyle`.

Ссылка на атрибут темы приказывает менеджеру ресурсов Android: «Перейди к теме приложения и найди в ней атрибут с именем `listSeparatorTextViewStyle`. Этот атрибут указывает на другой ресурс стиля. Помести значение этого ресурса сюда».

Каждая тема Android включает атрибут с именем `listSeparatorTextViewStyle`, но его определение зависит от оформления конкретной темы. Использование ссыл-

ки на атрибут темы гарантирует, что оформление виджетов `TextView` будет соответствовать оформлению вашего приложения.

О том, как работают стили и темы, более подробно рассказано в главе 22.

Рекомендации по проектированию интерфейсов Android

Для полей в наших приложениях Android Studio по умолчанию использует значение `16dp` или `8dp`. Оно следует рекомендации по проектированию материальных интерфейсов Android. Полный список рекомендаций находится по адресу *developer.android.com/design/index.html*.

Современные приложения Android должны соответствовать этим рекомендациям, насколько возможно. Рекомендации в значительной мере зависят от новой функциональности Android SDK, которая не всегда доступна или легко реализуема на старых устройствах. Некоторые рекомендации могут соблюдаться с использованием библиотеки AppCompat, о которой можно прочитать в главе 13.

Графический конструктор макетов

Графический конструктор макетов удобен, особенно при использовании `ConstraintLayout`. Тем не менее не все разработчики в восторге от него. Многие предпочитают простоту и ясность прямого ввода разметки XML.

Никто не заставляет вас выбирать что-то одно. Вы можете переключаться между графическим конструктором и прямым редактированием XML. Выбирайте для создания макетов тот способ, который вам кажется более удобным. В дальнейшем, когда потребуется создать макет, мы будем приводить диаграмму вместо разметки. Вы сами сможете решить, как создать ее: в разметке XML, в графическом конструкторе или сочетанием этих двух способов.

Упражнение. Форматирование даты

Объект `Date` больше напоминает временную метку (timestamp), чем традиционную дату. При вызове `toString()` для `Date` вы получаете именно временную метку, которая используется в строке `RecyclerView`. Временные метки хорошо подходят для отчетов, но на кнопке было бы лучше выводить дату в формате, более привычном для людей (например, «Jul 22, 2016»). Для этого можно воспользоваться экземпляром класса `android.text.format.DateFormat`. Хорошей отправной точкой в работе станет описание этого класса в документации Android.

Используйте методы класса `DateFormat` для формирования строки в стандартном формате или же подготовьте собственную форматную строку. Чтобы задача стала более творческой, попробуйте создать форматную строку для вывода дня недели («Friday, Jul 22, 2016»).

10 Аргументы фрагментов

В этой главе мы наладим совместную работу списка и детализации в приложении CriminalIntent. Когда пользователь щелкает на элементе списка преступлений, на экране возникает новый экземпляр `CrimeActivity`, который является хостом для экземпляра `CrimeFragment` с подробной информацией о конкретном экземпляре `Crime` (рис. 10.1).

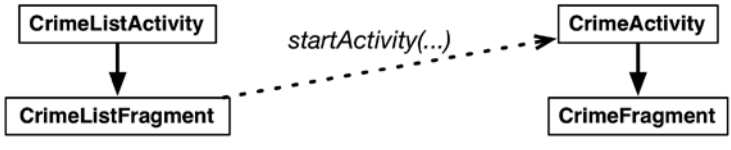


Рис. 10.1. Запуск CrimeActivity из CrimeListActivity

В приложении GeoQuiz одна активность (`QuizActivity`) запускала другую (`CheatActivity`). В приложении CriminalIntent активность `CrimeActivity` будет запускаться из фрагмента, а именно из `CrimeListFragment`.

Запуск активности из фрагмента

Запуск активности из фрагмента осуществляется практически так же, как запуск активности из другой активности. Вы вызываете метод `Fragment.startActivity(Intent)`, который вызывает соответствующий метод `Activity` во внутренней реализации.

В реализации `CrimeHolder` из `CrimeListFragment` замените уведомление кодом, запускающим экземпляр `CrimeActivity`.

Листинг 10.1. Запуск CrimeActivity (CrimeListFragment.java)

```
private class CrimeHolder extends RecyclerView.ViewHolder
        implements View.OnClickListener {
    ...
    @Override
    public void onClick(View view) {
        Toast.makeText(getActivity(),
```

```
                mCrime.getTitle() + " clicked!", Toast.LENGTH_SHORT)
                .show();
        Intent intent = new Intent(getActivity(), CrimeActivity.class);
        startActivity(intent);
    }
}
```

Здесь класс `CrimeListFragment` создает явный интент с указанием класса `CrimeActivity`. `CrimeListFragment` использует метод `getActivity()` для передачи активности-хоста как объекта `Context`, необходимого конструктору `Intent`.

Запустите приложение CriminalIntent. Щелкните на любой строке списка; открывается новый экземпляр `CrimeActivity`, управляющий фрагментом `CrimeFragment` (рис. 10.2).

Экземпляр `CrimeFragment` еще не содержит данных конкретного объекта `Crime`, потому что мы не сообщили ему, какой именно объект `Crime` следует отображать.

Включение дополнения

Чтобы сообщить `CrimeFragment`, какой объект `Crime` следует отображать, можно передать идентификатор в *дополнении* (extra) объекта `Intent` при запуске `CrimeActivity`.

Рис. 10.2. Пустой экземпляр CrimeFragment

Начните с создания нового метода `newIntent` в `CrimeActivity`.

Листинг 10.2. Создание нового метода newIntent (CrimeActivity.java)

```
public class CrimeActivity extends SingleFragmentActivity {

    public static final String EXTRA_CRIME_ID =
            "com.bignerdranch.android.criminalintent.crime_id";

    public static Intent newIntent(Context packageContext, UUID crimeId) {
        Intent intent = new Intent(packageContext, CrimeActivity.class);
        intent.putExtra(EXTRA_CRIME_ID, crimeId);
        return intent;
    }
    ...
}
```

После создания явного интента мы вызываем `putExtra(…)`, передавая строковый ключ и связанное с ним значение (`crimeId`). В данном случае вызывается версия `putExtra(String, Serializable)`, потому что `UUID` является объектом `Serializable`.

Затем обновите класс `CrimeHolder`, чтобы он использовал метод `newIntent` с передачей идентификатора преступления.

Листинг 10.3. Сохранение и передача Crime (CrimeListFragment.java)

```
private class CrimeHolder extends RecyclerView.ViewHolder
        implements View.OnClickListener {
    ...
    @Override
    public void onClick(View view) {
        ~~Intent intent = new Intent(getActivity(), CrimeActivity.class);~~
        Intent intent = CrimeActivity.newIntent(getActivity(), mCrime.getId());
        startActivity(intent);
    }
}
```

Чтение дополнения

Идентификатор преступления сохранен в интенте, принадлежащем `CrimeActivity`, однако прочитать и использовать эти данные должен класс `CrimeFragment`.

Существует два способа, которыми фрагмент может обратиться к данным из интента активности: простой «шоткат» и сложная, гибкая реализация. Сначала мы опробуем первый способ, а потом реализуем сложное гибкое решение с *аргументами фрагментов*.

В простом решении `CrimeFragment` использует метод `getActivity()` для прямого обращения к интенту `CrimeActivity`. В файле CrimeFragment.java прочитайте дополнение из интента `CrimeActivity` и используйте его для получения данных `Crime`.

Листинг 10.4. Чтение дополнения и получение Crime (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
    ...
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mCrime = new Crime();
        UUID crimeId = (UUID) getActivity().getIntent()
                .getSerializableExtra(CrimeActivity.EXTRA_CRIME_ID);
        mCrime = CrimeLab.get(getActivity()).getCrime(crimeId);
    }
    ...
}
```

Если не считать вызова `getActivity()`, листинг 10.4 практически не отличается от кода выборки дополнения из кода активности. Метод `getIntent()` возвращает объект `Intent`, используемый для запуска `CrimeActivity`. Мы вызываем `getSerializableExtra(String)` для `Intent`, чтобы извлечь `UUID` в переменную.

После получения идентификатора мы используем его для получения объекта `Crime` от `CrimeLab`.

Обновление представления CrimeFragment данными Crime

Теперь, когда фрагмент `CrimeFragment` получает объект `Crime`, его представление может отобразить данные `Crime`. Обновите метод `onCreateView(…)`, чтобы он выводил краткое описание преступления и признак раскрытия (код вывода даты уже имеется).

Листинг 10.5. Обновление объектов представления (CrimeFragment.java)

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    ...
    mTitleField = (EditText)v.findViewById(R.id.crime_title);
    mTitleField.setText(mCrime.getTitle());
    mTitleField.addTextChangedListener(new TextWatcher() {
        ...
    });
    ...
    mSolvedCheckBox = (CheckBox)v.findViewById(R.id.crime_solved);
    mSolvedCheckBox.setChecked(mCrime.isSolved());
    mSolvedCheckBox.setOnCheckedChangeListener(new OnCheckedChangeListener() {
        ...
    });
    ...
    return v;
}
```

Запустите приложение CriminalIntent. Выберите строку Crime #4 и убедитесь в том, что на экране появился экземпляр `CrimeFragment` с правильными данными преступления (рис. 10.3).

Рис. 10.3. Преступление, выбранное в списке

Недостаток прямой выборки

Обращение из фрагмента к интенту, принадлежащему активности-хосту, упрощает код. С другой стороны, оно нарушает инкапсуляцию фрагмента. Класс `CrimeFragment` уже не является структурным элементом, пригодным для повторного использования, потому что он предполагает, что его хостом всегда будет активность с объектом `Intent`, определяющим дополнение с именем com.bignerdranch.android.criminalintent.crime_id.

Возможно, для `CrimeFragment` такое предположение разумно, но оно означает, что класс `CrimeFragment` в своей текущей реализации не может использоваться с произвольной активностью.

Другое, более правильное решение — сохранение идентификатора в месте, принадлежащем `CrimeFragment` (вместо хранения его в личном пространстве `CrimeActivity`). В этом случае объект `CrimeFragment` может прочитать данные, не полагаясь на присутствие конкретного дополнения в интенте активности. Такое «место», принадлежащее фрагменту, называется *пакетом аргументов* (arguments bundle).

Аргументы фрагментов

К каждому экземпляру фрагмента может быть прикреплен объект `Bundle`. Этот объект содержит пары «ключ-значение», которые работают так же, как дополнения интентов `Activity`. Каждая такая пара называется *аргументом* (argument).

Чтобы создать аргументы фрагментов, вы сначала создаете объект `Bundle`, а затем используете `put`-методы `Bundle` соответствующего типа (по аналогии с методами `Intent`) для добавления аргументов в пакет:

```
Bundle args = new Bundle();
args.putSerializable(ARG_MY_OBJECT, myObject);
args.putInt(ARG_MY_INT, myInt);
args.putCharSequence(ARG_MY_STRING, myString);
```

Присоединение аргументов к фрагменту

Чтобы присоединить пакет аргументов к фрагменту, вызовите метод `Fragment.setArguments(Bundle)`. Присоединение должно быть выполнено после создания фрагмента, но до его добавления в активность.

Для этого программисты Android используют схему с добавлением в класс `Fragment` статического метода с именем `newInstance()`. Этот метод создает экземпляр фрагмента, упаковывает и задает его аргументы.

Когда активности-хосту потребуется экземпляр этого фрагмента, она вместо прямого вызова конструктора вызывает метод `newInstance()`. Активность может передать `newInstance(…)` любые параметры, необходимые фрагменту для создания аргументов.

Включите в `CrimeFragment` метод `newInstance(UUID)`, который получает `UUID`, создает пакет аргументов, создает экземпляр фрагмента, а затем присоединяет аргументы к фрагменту.

Листинг 10.6. Метод newInstance(UUID) (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {

    private static final String ARG_CRIME_ID = "crime_id";

    private Crime mCrime;
    private EditText mTitleField;
    private Button mDateButton;
    private CheckBox mSolvedCheckbox;

    public static CrimeFragment newInstance(UUID crimeId) {
        Bundle args = new Bundle();
        args.putSerializable(ARG_CRIME_ID, crimeId);

        CrimeFragment fragment = new CrimeFragment();
        fragment.setArguments(args);
        return fragment;
    }
    ...
}
```

Теперь класс `CrimeActivity` должен вызывать `CrimeFragment.newInstance(UUID)` каждый раз, когда ему потребуется создать `CrimeFragment`. При вызове передается значение `UUID`, полученное из дополнения. Вернитесь к классу `CrimeActivity`, в методе `createFragment()` получите дополнение из интента `CrimeActivity` и передайте его `CrimeFragment.newInstance(UUID)`.

Константу `EXTRA_CRIME_ID` также можно сделать закрытой, потому что ни одному другому классу не потребуется работать с этим дополнением.

Листинг 10.7. Использование newInstance(UUID) (CrimeActivity.java)

```
public class CrimeActivity extends SingleFragmentActivity {

    public private static final String EXTRA_CRIME_ID =
            "com.bignerdranch.android.criminalintent.crime_id";
    ...
    @Override
    protected Fragment createFragment() {
        return new CrimeFragment();
        UUID crimeId = (UUID) getIntent()
                .getSerializableExtra(EXTRA_CRIME_ID);
        return CrimeFragment.newInstance(crimeId);
    }
}
```

Учтите, что потребность в независимости не является двусторонней. Класс `CrimeActivity` должен многое знать о классе `CrimeFragment`, например то, что он содержит метод `newInstance(UUID)`. Это нормально; активность-хост должна располагать конкретной информацией о том, как управлять фрагментами, но фрагментам такая информация об их активности не нужна (по крайней мере, если вы хотите сохранить гибкость независимых фрагментов).

Получение аргументов

Когда фрагменту требуется получить доступ к его аргументам, он вызывает метод `getArguments()` класса `Fragment`, а затем один из `get`-методов `Bundle` для конкретного типа.

В методе `CrimeFragment.onCreate(…)` замените код упрощенного решения выборкой `UUID` из аргументов фрагмента.

Листинг 10.8. Получение идентификатора преступления из аргументов (CrimeFragment.java)

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    UUID crimeId = (UUID) getActivity().getIntent()
            .getSerializableExtra(CrimeActivity.EXTRA_CRIME_ID);
    UUID crimeId = (UUID) getArguments().getSerializable(ARG_CRIME_ID);
    mCrime = CrimeLab.get(getActivity()).getCrime(crimeId);
}
```

Запустите приложение CriminalIntent. Оно будет работать точно так же, но архитектура с независимостью `CrimeFragment` должна вызвать у вас приятные чувства. Кроме того, она хорошо подготовит нас к следующей главе, в которой мы реализуем более сложную систему навигации в CriminalIntent.

Перезагрузка списка

Осталась еще одна деталь, которой нужно уделить внимание. Запустите приложение CriminalIntent, щелкните на элементе списка и внесите изменения в подробную информацию о преступлении. Эти изменения сохраняются в модели, но при возвращении к списку содержимое `RecyclerView` остается неизменным.

Мы должны сообщить адаптеру `RecyclerView` , что набор данных изменился (или мог измениться), чтобы тот мог заново получить данные и повторно загрузить список. Работая со стеком возврата `ActivityManager`, можно перезагрузить список в нужный момент.

Когда `CrimeListFragment` запускает экземпляр `CrimeActivity`, последний помещается на вершину стека. При этом экземпляр `CrimeActivity`, который до этого находился на вершине, приостанавливается и останавливается.

Когда пользователь нажимает кнопку Back для возвращения к списку, экземпляр `CrimeActivity` извлекается из стека и уничтожается. В этот момент `CrimeListActivity` запускается и продолжает выполнение (рис. 10.4).

Рис. 10.4. Стек возврата CriminalIntent

Когда экземпляр `CrimeListActivity` продолжает выполнение, он получает вызов `onResume()` от ОС. При получении этого вызова `CrimeListActivity` его экземпляр

FragmentManager вызывает onResume() для фрагментов, хостом которых в настоящее время является активность. В нашем случае это единственный фрагмент CrimeListFragment.

В классе CrimeListFragment переопределите onResume() и инициируйте вызов updateUI() для перезагрузки списка. Измените метод updateUI() для вызова notifyDataSetChanged(), если объект CrimeAdapter уже создан.

Листинг 10.9. Перезагрузка списка в onResume() (CrimeFragment.java)

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    ...
}

@Override
public void onResume() {
    super.onResume();
    updateUI();
}

private void updateUI() {
    CrimeLab crimeLab = CrimeLab.get(getActivity());
    List<Crime> crimes = crimeLab.getCrimes();

  if (mAdapter == null) {
        mAdapter = new CrimeAdapter(crimes);
        mCrimeRecyclerView.setAdapter(mAdapter);
    } else {
        mAdapter.notifyDataSetChanged();
    }
}
```

Почему для обновления RecyclerView переопределяется метод onResume(), а не onStart()? Мы не можем предполагать, что активность останавливается при нахождении перед ней другой активности. Если другая активность прозрачна, ваша активность может быть только приостановлена. Если же ваша активность приостановлена, а код обновления находится в onStart(), список не будет перезагружаться. В общем случае самым безопасным местом для выполнения действий, связанных с обновлением представления фрагмента, является метод onResume().

Запустите приложение CriminalIntent. Выберите преступление в списке и измените его подробную информацию. Вернувшись к списку, вы немедленно увидите свои изменения.

За последние две главы приложение CriminalIntent серьезно продвинулось вперед. Давайте взглянем на обновленную диаграмму объектов (рис. 10.5).

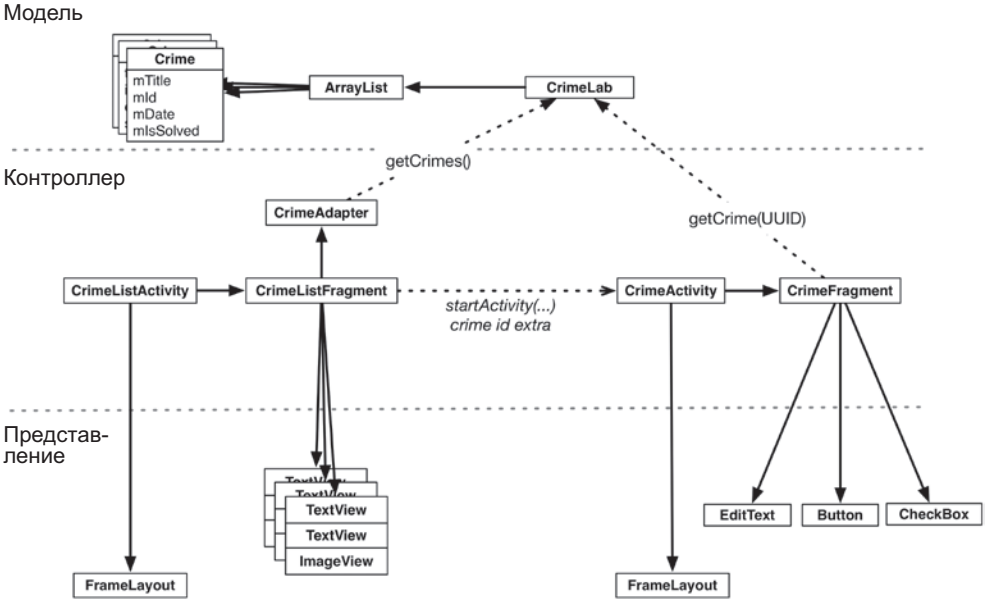


Рис. 10.5. Обновленная диаграмма объектов CriminalIntent

Получение результатов с использованием фрагментов

В этой главе нам не требовалось, чтобы запущенная активность возвращала результат. А если бы это было нужно? Код выглядел бы почти так же, как в приложении GeoQuiz. Вместо метода `startActivityForResult(…)` класса `Activity` использовался бы метод `Fragment.startActivityForResult(…)`. Вместо `Activity.onActivityResult(…)` мы переопределим `Fragment.onActivityResult(…)`:

```
public class CrimeListFragment extends Fragment {

    private static final int REQUEST_CRIME = 1;
    ...
    private class CrimeHolder extends RecyclerView.ViewHolder
            implements View.OnClickListener {
        ...
        @Override
        public void onClick(View view) {
            Intent intent = CrimeActivity.newIntent(getActivity(), mCrime.getId());
            startActivityForResult(intent, REQUEST_CRIME);
        }
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
```

```
        if (requestCode == REQUEST_CRIME) {
            // Обработка результата
        }
    }
    ...
}
```

Метод `Fragment.startActivityForResult(Intent,int)` похож на одноименный метод класса `Activity`. Он включает дополнительный код передачи результата вашему фрагменту от активности-хоста.

Возвращение результата от фрагмента выглядит немного иначе. Фрагмент может получить результат от активности, но не может вернуть собственный результат — на это способны только активности. Таким образом, хотя `Fragment` содержит методы `startActivityForResult(…)` и `onActivityResult(…)`, у него нет методов `setResult(…)`.

Вместо этого вы приказываете *активности-хосту* вернуть значение; вот как это делается:

```
public class CrimeFragment extends Fragment {
    ...
    public void returnResult() {
        getActivity().setResult(Activity.RESULT_OK, null);
    }
}
```

Для любознательных: зачем использовать аргументы фрагментов?

Все это выглядит как-то сложно. Почему бы просто не задать переменную экземпляра в `CrimeFragment` при создании?

Потому что такое решение будет работать не всегда. Когда ОС создает заново ваш фрагмент (либо вследствие изменения конфигурации, либо при переходе пользователя к другому приложению с последующим освобождением памяти ОС), все переменные экземпляров теряются. Также помните о возможной нехватке памяти.

Если вы хотите, чтобы решение работало во всех случаях, придется сохранять аргументы.

Один из возможных вариантов — использование механизма сохранения состояния экземпляров. Идентификатор преступления заносится в обычную переменную экземпляра, сохраняется вызовом `onSaveInstanceState(Bundle)`, а затем извлекается из `Bundle` в `onCreate(Bundle)`. Такое решение будет работать всегда.

С другой стороны, оно усложняет сопровождение. Если вы вернетесь к фрагменту через несколько лет и добавите еще один аргумент, нельзя исключать, что вы забудете сохранить его в `onSaveInstanceState(Bundle)`. Смысл этого решения не столь очевиден.

Разработчики Android предпочитают решение с аргументами фрагментов, потому что в этом случае они предельно явно и четко обозначают свои намерения. Через несколько лет вы вернетесь к коду и будете знать, что идентификатор преступления — это аргумент, который надежно передается новым экземплярам этого фрагмента. При добавлении нового аргумента вы знаете, что его нужно сохранить в пакете аргументов.

Упражнение. Эффективная перезагрузка RecyclerView

Метод `notifyDataSetChanged` адаптера хорошо подходит для того, чтобы приказать `RecyclerView` перезагрузить все элементы, видимые в настоящее время.

В CriminalIntent этот метод ужасающе неэффективен, потому что при возвращении к `CrimeListFragment` заведомо изменилось не более одного объекта `Crime`.

Используйте метод `notifyItemChanged(int)` объекта `RecyclerView.Adapter`, чтобы перезагрузить один элемент в списке. Изменить код для вызова этого метода несложно; труднее обнаружить, в какой позиции произошло изменение, и перезагрузить правильный элемент.

Упражнение. Улучшение быстродействия CrimeLab

Метод `get(UUID)` класса `CrimeLab` работает, но процедуру последовательного сравнения идентификатора каждого преступления с искомым идентификатором можно улучшить. Повысьте эффективность процедуры поиска так, чтобы существующее поведение `CriminalIntent` не изменилось в результате рефакторинга.

11 ViewPager

В этой главе мы создадим новую активность, которая станет хостом для `CrimeFragment`. Макет активности будет состоять из экземпляра `ViewPager`. Включение виджета `ViewPager` в пользовательский интерфейс позволяет «листать» элементы списка, проводя пальцем по экрану (рис. 11.1).

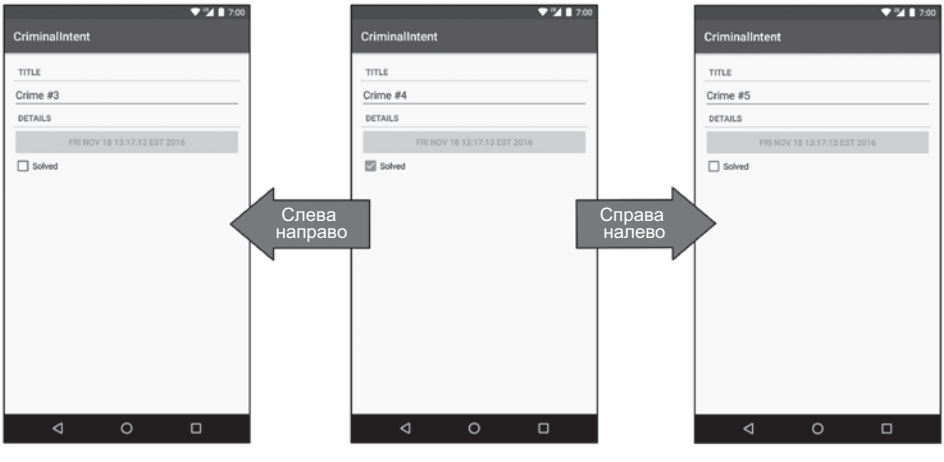


Рис. 11.1. Листание страниц

На рис. 11.2 представлена обновленная диаграмма CriminalIntent. Новая активность с именем `CrimePagerActivity` займет место `CrimeActivity`. Ее макет состоит из экземпляра `ViewPager`.

Все новые объекты, которые необходимо создать, находятся в пунктирном прямоугольнике на приведенной диаграмме. Для реализации листания страничных представлений в CriminalIntent ничего другого менять не придется. В частности, класс `CrimeFragment` останется неизменным благодаря той работе по обеспечению независимости `CrimeFragment`, которую мы проделали в главе 10.

В этой главе нам предстоит:

- создать класс `CrimePagerActivity`;
- определить иерархию представлений, состоящую из `ViewPager`;

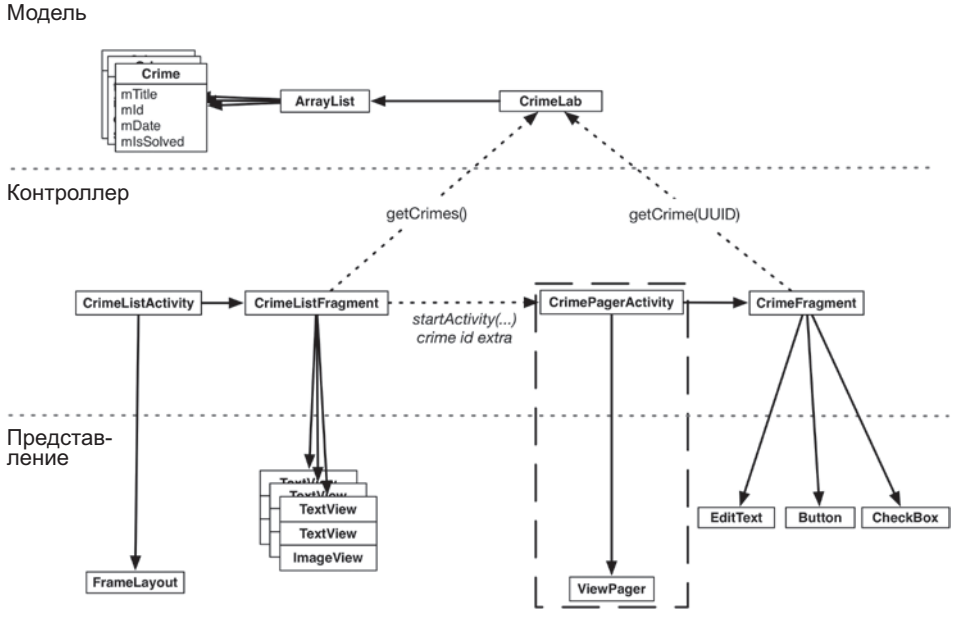


Рис. 11.2. Диаграмма объектов CrimePagerActivity

- связать экземпляр `ViewPager` с его адаптером `CrimePagerActivity`;
- изменить метод `CrimeHolder.onClick(…)` так, чтобы он запускал `CrimePagerActivity` вместо `CrimeActivity`.

Создание CrimePagerActivity

Класс `CrimePagerActivity` будет субклассом `AppCompatActivity`. Он создает экземпляр и управляет `ViewPager`.

Создайте новый класс с именем `CrimePagerActivity`. Назначьте его суперклассом `AppCompatActivity` и создайте представление для активности.

Листинг 11.1. Создание ViewPager (CrimePagerActivity.java)

```
public class CrimePagerActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_crime_pager);
    }
}
```

Файл макета еще не существует. Создайте новый файл макета в res/layout/ и присвойте ему имя activity_crime_pager. Назначьте его корневым представлением

ViewPager и присвойте ему атрибуты, показанные на рис. 11.3. Обратите внимание на необходимость использования полного имени пакета ViewPager (android.support.v4.view.ViewPager).

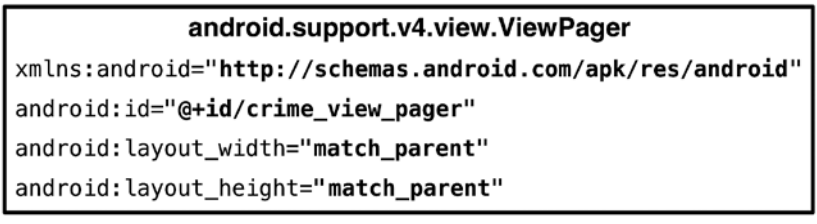


Рис. 11.3. Определение ViewPage в CrimePagerActivity (activity_crime_pager.xml)

Полное имя пакета используется при добавлении в файл макета, потому что класс ViewPager определен в библиотеке поддержки. В отличие от Fragment класс ViewPager доступен только в библиотеке поддержки; в более поздних версиях SDK так и не появилось «стандартного» класса ViewPager.

ViewPager и PagerAdapter

Класс ViewPager в чем-то похож на RecyclerView. Чтобы класс RecyclerView мог выдавать представления, ему необходим экземпляр Adapter. Классу ViewPager необходим адаптер PagerAdapter.

Однако взаимодействие между ViewPager и PagerAdapter намного сложнее взаимодействия между RecyclerView и Adapter. К счастью, мы можем использовать FragmentStatePagerAdapter — субкласс PagerAdapter, который берет на себя многие технические подробности.

FragmentStatePagerAdapter сводит взаимодействие к двум простым методам: getCount() и getItem(int). При вызове метода getItem(int) для позиции в массиве преступлений следует вернуть объект CrimeFragment, настроенный для вывода информации объекта в заданной позиции.

В классе CrimePagerActivity добавьте следующий код для назначения PagerAdapter класса ViewPager и реализации его методов getCount() и getItem(int).

Листинг 11.2. Назначение PagerAdapter (CrimePagerActivity.java)

```
public class CrimePagerActivity extends AppCompatActivity {

    private ViewPager mViewPager;
    private List<Crime> mCrimes;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_crime_pager);
```

```
        mViewPager = (ViewPager) findViewById(R.id.crime_view_pager);

        mCrimes = CrimeLab.get(this).getCrimes();
        FragmentManager fragmentManager = getSupportFragmentManager();
        mViewPager.setAdapter(new FragmentStatePagerAdapter(fragmentManager) {

            @Override
            public Fragment getItem(int position) {
                Crime crime = mCrimes.get(position);
                return CrimeFragment.newInstance(crime.getId());
            }

            @Override
            public int getCount() {
                return mCrimes.size();
            }
        });
    }
}
```

Пройдемся по этому коду. После поиска `ViewPager` в представлении активности мы получаем от `CrimeLab` набор данных — контейнер `List` объектов `Crime`. Затем мы получаем экземпляр `FragmentManager` для активности.

На следующем шаге адаптером назначается безымянный экземпляр `FragmentStatePagerAdapter`. Для создания `FragmentStatePagerAdapter` необходим объект `FragmentManager`. Не забывайте, что `FragmentStatePagerAdapter` — ваш агент, управляющий взаимодействием с `ViewPager`. Чтобы агент мог выполнить свою работу с фрагментами, возвращаемыми в `getItem(int)`, он должен быть способен добавить их в активность. Вот почему ему необходим экземпляр `FragmentManager`.

(Что именно делает агент? Вкратце, он добавляет возвращаемые фрагменты в активность и помогает `ViewPager` идентифицировать представления фрагментов для их правильного размещения. Более подробная информация приведена в разделе «Для любознательных» в конце главы.)

Два метода `PagerAdapter` весьма просты. Метод `getCount()` возвращает текущее количество элементов в списке. Все существенное происходит в методе `getItem(int)`. Он получает экземпляр `Crime` для заданной позиции в наборе данных, после чего использует его идентификатор для создания и возвращения правильно настроенного экземпляра `CrimeFragment`.

Интеграция CrimePagerActivity

Теперь можно переходить к устранению класса `CrimeActivity` и замене его классом `CrimePagerActivity`.

Начнем с добавления метода `newIntent` в `CrimePagerActivity` вместе с дополнением для идентификатора преступления.

Листинг 11.3. Создание newIntent (CrimePagerActivity.java)

```
public class CrimePagerActivity extends AppCompatActivity {
    private static final String EXTRA_CRIME_ID =
            "com.bignerdranch.android.criminalintent.crime_id";

    private ViewPager mViewPager;
    private List<Crime> mCrimes;

    public static Intent newIntent(Context packageContext, UUID crimeId) {
        Intent intent = new Intent(packageContext, CrimePagerActivity.class);
        intent.putExtra(EXTRA_CRIME_ID, crimeId);
        return intent;
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_crime_pager);

        UUID crimeId = (UUID) getIntent()
                .getSerializableExtra(EXTRA_CRIME_ID);
        ...
    }
}
```

Теперь нужно сделать так, чтобы при выборе элемента списка в `CrimeListFragment` запускался экземпляр `CrimePagerActivity` вместо `CrimeActivity`.

Вернитесь к файлу CrimeListFragment.java и измените метод `CrimeHolder.onClick(…)`, чтобы он запускал `CrimePagerActivity`.

Листинг 11.4. Запуск активности (CrimeListFragment.java)

```
private class CrimeHolder extends RecyclerView.ViewHolder
        implements View.OnClickListener {
    ...
    @Override
    public void onClick(View view) {
        Intent intent = CrimeActivity.newIntent(getActivity(), mCrime.getId());
        Intent intent = CrimePagerActivity.newIntent(getActivity(), mCrime.
                        getId());
        startActivity(intent);
    }
}
```

Также необходимо добавить `CrimePagerActivity` в манифест, чтобы ОС могла запустить эту активность. Пока манифест будет открыт, заодно удалите объявление `CrimeActivity`. Для этого достаточно заменить в манифесте `CrimeActivity` на `CrimePagerActivity`.

Листинг 11.5. Добавление CrimePagerActivity в манифест (AndroidManifest.xml)

```
<manifest ...>
    ...
    <application ...>
    ...
        <activity
            android:name=".CrimeActivity"
            android:name=".CrimePagerActivity">
        </activity>
```

Наконец, чтобы не загромождать проект, удалите CrimeActivity.java в окне инструментов Project.

Запустите приложение CriminalIntent. Нажмите на строке Crime #0, чтобы просмотреть подробную информацию. Проведите по экрану влево или вправо, чтобы просмотреть другие элементы списка. Обратите внимание: переключение страниц происходит плавно и без задержек. По умолчанию `ViewPager` загружает элемент, находящийся на экране, а также по одному соседнему элементу в каждом направлении, чтобы отклик на жест прокрутки был немедленным. Количество загружаемых соседних страниц можно настроить вызовом `setOffscreenPageLimit(int)`.

Однако с `ViewPager` еще не все идеально. Вернитесь к списку при помощи кнопки Back и щелкните на другом элементе. Вы снова увидите информацию первого элемента вместо того, который был запрошен.

По умолчанию `ViewPager` отображает в своем экземпляре `PagerAdapter` первый элемент. Чтобы вместо него отображался элемент, выбранный пользователем, назначьте текущим элементом `ViewPager` элемент с указанным индексом.

В конце `CrimePagerActivity.onCreate(…)` найдите индекс отображаемого преступления; для этого переберите и проверьте идентификаторы всех преступлений. Когда вы найдете экземпляр `Crime`, у которого поле `mId` совпадает с `crimeId` в дополнении интента, измените текущий элемент по индексу найденного объекта `Crime`.

Листинг 11.6. Назначение исходного элемента (CrimePagerActivity.java)

```
public class CrimePagerActivity extends AppCompatActivity {
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        FragmentManager fragmentManager = getSupportFragmentManager();
        mViewPager.setAdapter(new FragmentStatePagerAdapter(fragmentManager) {
            ...
        });

        for (int i = 0; i < mCrimes.size(); i++) {
            if (mCrimes.get(i).getId().equals(crimeId)) {
                mViewPager.setCurrentItem(i);
                break;
            }
        }
    }
}
```

Запустите приложение CriminalIntent. При выборе любого элемента списка должна отображаться подробная информация правильного объекта `Crime`. Вот и все! Теперь наш экземпляр `ViewPager` полностью готов к работе.

FragmentStatePagerAdapter и FragmentPagerAdapter

Существует еще один тип `PagerAdapter`, который можно использовать в приложениях; он называется `FragmentPagerAdapter`.

`FragmentPagerAdapter` используется точно так же, как `FragmentStatePagerAdapter` и отличается от него только способом выгрузки неиспользуемых фрагментов.

При использовании класса `FragmentStatePagerAdapter` неиспользуемый фрагмент уничтожается (рис. 11.4). Происходит закрепление транзакции для полного удаления фрагмента из объекта `FragmentManager` активности. Наличие «состояния» у `FragmentStatePagerAdapter` определяется тем фактом, что экземпляр при уничтожении сохраняет объект `Bundle` вашего фрагмента в методе `onSaveInstanceState(Bundle)`. Когда пользователь возвращается обратно, новый фрагмент восстанавливается по состоянию этого экземпляра.

`FragmentPagerAdapter` ничего подобного не делает. Когда фрагмент становится ненужным, `FragmentPagerAdapter` вызывает для транзакции `detach(Fragment)` вместо `remove(Fragment)`. Представление фрагмента при этом уничтожается, но экземпляр фрагмента продолжает существовать в `FragmentManager`. Таким образом, фрагменты, созданные `FragmentPagerAdapter`, никогда не уничтожаются (рис. 11.5).

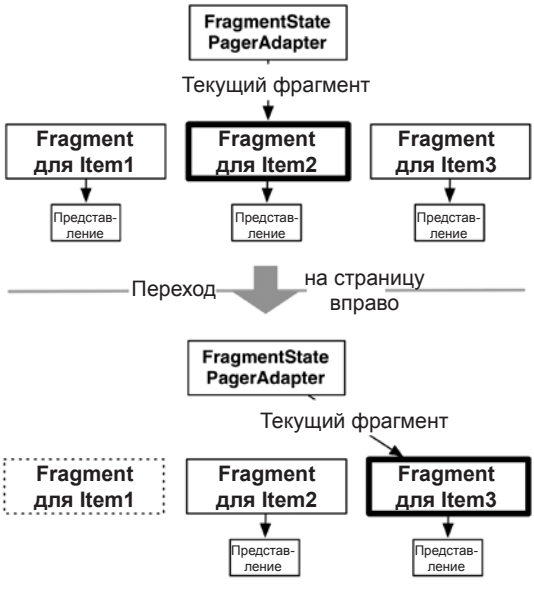


Рис. 11.4. Управление фрагментами FragmentStatePagerAdapter

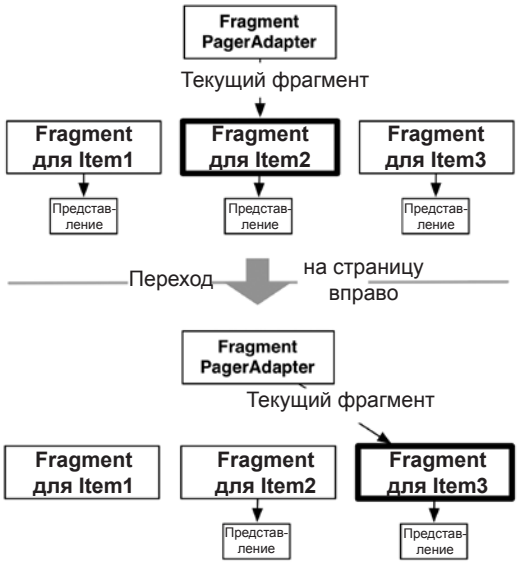


Рис. 11.5. Управление фрагментами FragmentPagerAdapter

Выбор используемого адаптера зависит от приложения. Как правило, `FragmentStatePagerAdapter` более экономно расходует память. Приложение CriminalIntent выводит список, который со временем может стать достаточно длинным, причем к каждому элементу списка может прилагаться фотография. Хранить всю эту информацию в памяти нежелательно, поэтому мы используем `FragmentStatePagerAdapter`.

С другой стороны, если интерфейс содержит небольшое фиксированное количество фрагментов, использование `FragmentPagerAdapter` безопасно и уместно. Самый характерный пример такого рода — интерфейс со вкладками. Некоторые детализированные представления не помещаются на одном экране, поэтому отображаемая информация распределяется между несколькими вкладками. Добавление `ViewPager` с перебором вкладок делает этот интерфейс интуитивным. Хранение фрагментов в памяти упрощает управление кодом контроллера, а поскольку этот стиль интерфейса обычно использует всего два или три фрагмента на активность, проблемы с нехваткой памяти крайне маловероятны.

Для любознательных: как работает ViewPager

Классы `ViewPager` и `PagerAdapter` незаметно выполняют большую часть рутинной работы. В этом разделе приведена более подробная информация о том, что при этом происходит.

Прежде чем мы перейдем к обсуждению, предупреждаем: в большинстве случаев понимать все технические подробности не обязательно.

Но если вы захотите реализовать интерфейс PagerAdapter самостоятельно, вы должны знать, чем отношения ViewPager-PagerAdapter отличаются от обычных отношений RecyclerView-Adapter.

Когда может возникнуть необходимость в самостоятельной реализации интерфейса PagerAdapter? Когда в ViewPager должны размещаться не фрагменты, а нечто иное. Например, если вы захотите разместить в ViewPager обычные представления View, скажем графические поля, вы реализуете интерфейс PagerAdapter.

Почему ViewPager, а не RecyclerView?

Использование RecyclerView в данном случае потребует большого объема работы, потому что вы не сможете использовать существующие экземпляры Fragment. Adapter ожидает, что вы сможете предоставить View мгновенно. Но когда будет создано представление вашего фрагмента, решает FragmentManager, а не вы. Таким образом, когда RecyclerView обратится к Adapter за представлением вашего фрагмента, вы не сможете создать фрагмент *и* немедленно выдать его представление.

Именно по этой причине и существует класс ViewPager. Вместо Adapter он использует класс с именем PagerAdapter. Этот класс сложнее Adapter, потому что он выполняет больший объем работы по управлению представлениями. Ниже кратко перечислены основные различия.

Вместо метода onBindViewHolder(…), возвращающего ViewHolder с соответствующим представлением, PagerAdapter содержит следующие методы:

```
public Object instantiateItem(ViewGroup container, int position)
public void destroyItem(ViewGroup container, int position, Object object)
public abstract boolean isViewFromObject(View view, Object object)
```

Метод pagerAdapter.instantiateItem(ViewGroup, int) приказывает адаптеру создать представление элемента списка для заданной позиции и добавить его в контейнер ViewGroup; метод destroyItem(ViewGroup, int, Object) приказывает уничтожить этот элемент. Обратите внимание: метод instantiateItem(ViewGroup, int) не приказывает создать представление *немедленно*. PagerAdapter может создать представление в любой момент в будущем.

После того как представление было создано, ViewPager в какой-то момент замечает его. Чтобы понять, к какому элементу списка оно относится, ViewPager вызывает метод isViewFromObject(View, Object). Параметр Object содержит объект, полученный при вызове instantiateItem(ViewGroup, int). Таким образом, если ViewPager вызывает instantiateItem(ViewGroup, 5) и получает объект A, вызов isViewFromObject(View, A) должен вернуть true, если переданный экземпляр View относится к элементу 5, и false в противном случае.

Этот процесс достаточно сложен для ViewPager, но не для класса PagerAdapter, который должен уметь только создавать представления, уничтожать представления и определять, к какому объекту относится представление. Менее жесткие требования позволяют реализации PagerAdapter создавать и добавлять новый фрагмент в instantiateItem(ViewGroup, int) и возвращать фрагмент как отслеживаемый экземпляр Object. При этом isViewFromObject(View, Object) выглядит примерно так:

```
@Override
public boolean isViewFromObject(View view, Object object) {
    return ((Fragment)object).getView() == view;
}
```

Реализовывать переопределения `PagerAdapter` каждый раз, когда потребуется использовать `ViewPager`, было бы слишком утомительно. Хорошо, что у нас есть `FragmentPagerAdapter` и `FragmentStatePagerAdapter`.

Для любознательных: формирование макетов представлений в коде

В этой книге макеты представлений определяются исключительно в XML-файлах макетов. Также возможно создавать макеты представлений в коде.

Собственно, `ViewPager` можно было бы определить полностью в коде вообще без файла макета:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ViewPager viewPager = new ViewPager(this);
    setContentView(viewPager);
    ...
}
```

Создание представления не требует никакого волшебства; просто вызовите его конструктор и передайте параметр `Context`. Всю иерархию представлений можно создать на программном уровне, вместо того чтобы использовать файлы макетов.

Тем не менее строить представления в коде не рекомендуется, потому что файлы макетов предоставляют ряд преимуществ.

Первое преимущество файлов макетов заключается в том, что они обеспечивают четкое разделение между контроллером и объектами представлений в приложении. Представление существует в XML, контроллер существует в коде Java. Это разделение упрощает сопровождение кода за счет ограничения объема изменений в контроллере при изменении представления, и наоборот.

Другое преимущество определения представлений в XML заключается в том, что система уточнения ресурсов Android позволяет автоматически выбрать версию файла XML в зависимости от свойств устройства.

Как было показано в главе 3, эта система позволяет легко сменить файл макета в зависимости от ориентации устройства (а также других параметров конфигурации).

Есть ли у файлов макетов какие-либо недостатки? Пожалуй, хлопоты с созданием файла XML и его заполнением. Если вы создаете всего одно представление, иногда эти хлопоты могут показаться излишними.

В остальном сколько-нибудь серьезных недостатков нет — группа разработки Android никогда не рекомендовала строить иерархии представлений на программном уровне, даже в прежние времена, когда разработчикам приходилось беспокоиться о быстродействии больше, чем сейчас. Даже если вам требуется нечто настолько мелкое, как представление с идентификатором (что часто требуется для представлений, созданных на программном уровне), проще создать файл макета.

Упражнение. Восстановление полей CrimeFragment

Возможно, вы заметили, что в `CrimeFragment` загадочным образом пропали поля. В файле fragment_crime.xml заданы поля с размером `16dp`.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="16dp"
    android:orientation="vertical">
```

Эти поля не отображаются. В чем дело? Параметры макета `ViewPager` не поддерживают поля.

Внесите изменения в файл fragment_crime.xml и восстановите поля.

Упражнение. Добавление кнопок для перехода в начало и конец списка

Добавьте в `CrimePagerActivity` две кнопки, позволяющие мгновенно перевести `ViewPager` к первому или последнему преступлению в списке. Заодно обеспечьте блокировку кнопки перехода в начало на первой странице и кнопки перехода в конец на последней странице.

12 Диалоговые окна

Диалоговые окна требуют от пользователя внимания и ввода данных. Обычно они используются для принятия решений или отображения важной информации. В этой главе мы добавим диалоговое окно, в котором пользователь может изменить дату преступления.

При нажатии кнопки даты в `CrimeFragment` открывается диалоговое окно, показанное на рис. 12.1.

Рис. 12.1. Диалоговое окно для выбора даты

Диалоговое окно на рис. 12.1 является экземпляром `AlertDialog` — субкласса `Dialog`. Именно этот многоцелевой субкласс `Dialog` вы будете чаще всего использовать в своих программах.

В версии Lollipop диалоговые окна прошли визуальную переработку. Окна `AlertDialog` в Lollipop автоматически используют новый стиль. В более ранних версиях Android окна `AlertDialog` возвращаются к старому стилю, изображенному слева на рис. 12.2.

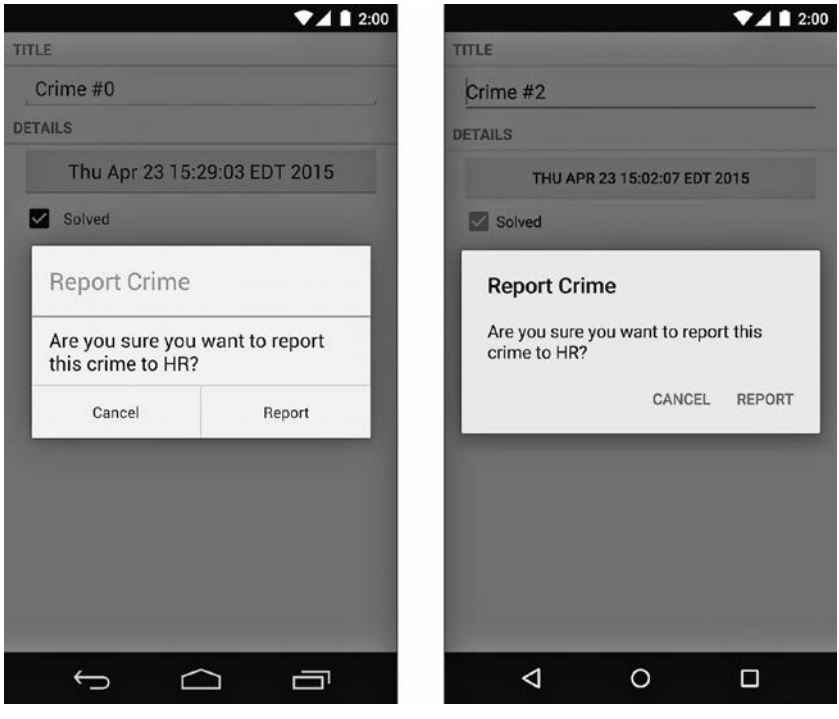


Рис. 12.2. Старый и новый визуальный стиль

Конечно, вместо доисторических диалоговых окон было бы лучше всегда отображать окна в новом стиле независимо от версии Android на устройстве пользователя. Этого можно добиться при помощи версии `AlertDialog` из библиотеки AppCompat. Эта версия `AlertDialog` очень похожа на встроенную в ОС Android, но, как и другие классы AppCompat, совместима с более ранними версиями. Чтобы пользоваться преимуществами версии AppCompat, импортируйте библиотеку `android.support.v7.app.AlertDialog`, когда вам будет предложено.

Создание DialogFragment

При использовании объекта `AlertDialog` обычно удобно упаковать его в экземпляр `DialogFragment` — субкласса `Fragment`. Вообще говоря, экземпляр `AlertDialog` может отображаться и без `DialogFragment`, но Android так поступать не рекомендует. Управление диалоговым окном из `FragmentManager` открывает больше возможностей для его отображения.

Кроме того, «минимальный» экземпляр `AlertDialog` исчезнет при повороте устройства. С другой стороны, если экземпляр `AlertDialog` упакован во фрагмент, после поворота диалоговое окно будет создано заново и появится на экране.

Для приложения CriminalIntent мы создадим субкласс `DialogFragment` с именем `DatePickerFragment`. В коде `DatePickerFragment` создается и настраивается экземпляр `AlertDialog`, отображающий виджет `DatePicker`. В качестве хоста `DatePickerFragment` используется экземпляр `CrimePagerActivity`.

На рис. 12.3 изображена схема этих отношений.

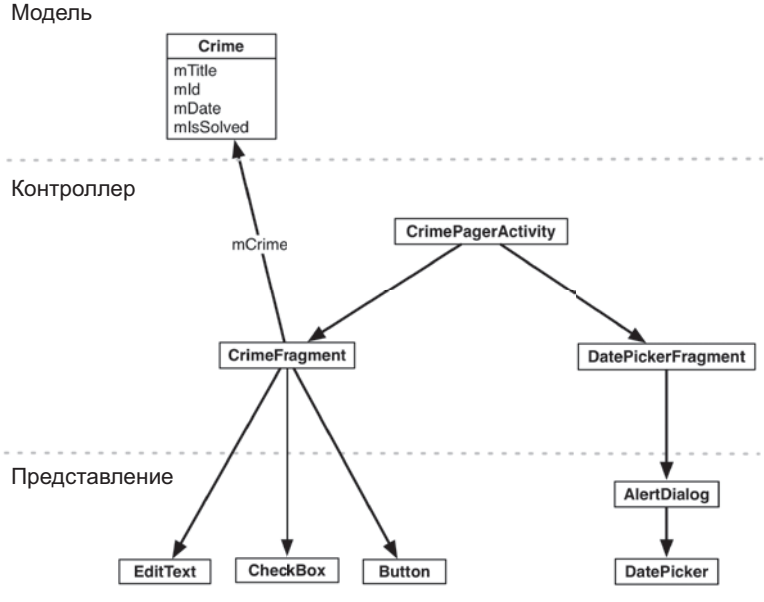


Рис. 12.3. Диаграмма объектов для двух фрагментов с хостом CrimePagerActivity

Наши первоочередные задачи:

- создание класса `DatePickerFragment`;
- построение `AlertDialog`;
- вывод диалогового окна на экран с использованием `FragmentManager`.

Позднее в этой главе мы подключим виджет `DatePicker` и организуем передачу необходимых данных между `CrimeFragment` и `DatePickerFragment`.

Прежде чем браться за работу, добавьте строковый ресурс (листинг 12.1).

Листинг 12.1. Добавление строки заголовка диалогового окна (values/strings.xml)

```
<resources>
    ...
    <string name="crime_solved_label">Solved</string>
    <string name="date_picker_title">Date of crime:</string>
</resources>
```

Создайте новый класс с именем `DatePickerFragment` и назначьте его суперклассом `DialogFragment`. Обязательно выберите версию `DialogFragment` из библиотеки поддержки: `android.support.v4.app.DialogFragment`.

Класс `DialogFragment` содержит следующий метод:

```
public Dialog onCreateDialog(Bundle savedInstanceState)
```

Экземпляр `FragmentManager` активности-хоста вызывает этот метод в процессе вывода `DialogFragment` на экран.

Добавьте в файл DatePickerFragment.java реализацию `onCreateDialog(Bundle)`, которая создает `AlertDialog` с заголовком и одной кнопкой OK. (Виджет `DatePicker` мы добавим позднее.)

Проследите за тем, чтобы в программе импортировалась версия `AlertDialog` из AppCompat: `android.support.v7.app.AlertDialog`.

Листинг 12.2. Создание DialogFragment (DatePickerFragment.java)

```
public class DatePickerFragment extends DialogFragment {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        return new AlertDialog.Builder(getActivity())
            .setTitle(R.string.date_picker_title)
            .setPositiveButton(android.R.string.ok, null)
            .create();
    }
}
```

В этой реализации используется класс `AlertDialog.Builder`, предоставляющий динамичный интерфейс для конструирования экземпляров `AlertDialog`.

Сначала мы передаем объект `Context` конструктору `AlertDialog.Builder`, который возвращает экземпляр `AlertDialog.Builder`.

Затем вызываются два метода `AlertDialog.Builder` для настройки диалогового окна:

```
public AlertDialog.Builder setTitle(int titleId)
public AlertDialog.Builder setPositiveButton(int textId,
    DialogInterface.OnClickListener listener)
```

Метод `setPositiveButton(…)` получает строковый ресурс и объект, реализующий `DialogInterface.OnClickListener`. В листинге 12.2 передается константа Android для кнопки OK и `null` вместо слушателя. Слушатель будет реализован позднее в этой главе.

Положительная кнопка (`Positive`) нажимается пользователем для подтверждения информации в диалоговом окне. В `AlertDialog` также можно добавить еще две кнопки: *отрицательную* (`Negative`) и *нейтральную* (`Neutral`). Эти обозначения определяют позицию кнопок в диалоговом окне (если их несколько).

Построение диалогового окна завершается вызовом `AlertDialog.Builder.create()`, который возвращает настроенный экземпляр `AlertDialog`.

Этим возможности AlertDialog и AlertDialog.Builder не исчерпываются; подробности достаточно хорошо изложены в документации разработчика. А пока давайте перейдем к механике вывода диалогового окна на экран.

Отображение DialogFragment

Как и все фрагменты, экземпляры DialogFragment находятся под управлением экземпляра FragmentManager активности-хоста.

Для добавления экземпляра DialogFragment в FragmentManager и вывода его на экран используются следующие методы экземпляра фрагмента:

```
public void show(FragmentManager manager, String tag)
public void show(FragmentTransaction transaction, String tag)
```

Строковый параметр однозначно идентифицирует DialogFragment в списке FragmentManager. Выбор версии (с FragmentManager или FragmentTransaction) зависит только от вас: если передать FragmentTransaction, за создание и закрепление транзакции отвечаете вы. При передаче FragmentManager транзакция автоматически создается и закрепляется для вас.

В нашем примере передается FragmentManager.

Добавьте в CrimeFragment константу для метки DatePickerFragment. Затем в методе onCreateView(…) удалите код, блокирующий кнопку даты, и назначьте слушателя View.OnClickListener, который отображает DatePickerFragment при нажатии кнопки даты.

Листинг 12.3. Отображение DialogFragment (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {

    private static final String ARG_CRIME_ID = "crime_id";
    private static final String DIALOG_DATE = "DialogDate";
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        ...
        mDateButton = (Button) v.findViewById(R.id.crime_date);
        mDateButton.setText(mCrime.getDate().toString());
        mDateButton.setEnabled(false);
        mDateButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                FragmentManager manager = getFragmentManager();
                DatePickerFragment dialog = new DatePickerFragment();
                dialog.show(manager, DIALOG_DATE);
            }
        });
```

```
        mSolvedCheckBox = (CheckBox) v.findViewById(R.id.crime_solved);
        ...
        return v;
    }
}
```

Запустите приложение CriminalIntent и нажмите кнопку даты, чтобы диалоговое окно появилось на экране (рис. 12.4).

Рис. 12.4. AlertDialog с заголовком и кнопкой

Назначение содержимого диалогового окна

Далее мы включим в `AlertDialog` виджет `DatePicker` при помощи метода `AlertDialog.Builder`:

```
public AlertDialog.Builder setView(View view)
```

Метод настраивает диалоговое окно для отображения переданного объекта `View` между заголовком и кнопкой(-ами).

В окне инструментов Project создайте новый файл макета с именем dialog_date.xml и назначьте его корневым элементом `DatePicker`. Макет будет состоять из одного объекта `View (DatePicker)`, который мы заполним и передадим `setView(…)`.

Настройте макет `DatePicker` так, как показано на рис. 12.5.

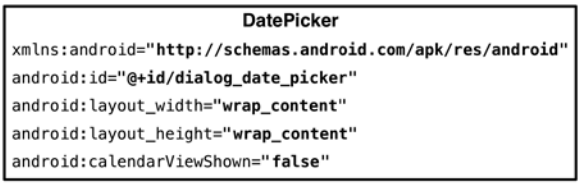


Рис. 12.5. Макет DatePicker (layout/dialog_date.xml)

В методе `DatePickerFragment.onCreateDialog(Bundle)` заполните представление и назначьте его диалоговому окну.

Листинг 12.4. Включение DatePicker в AlertDialog (DatePickerFragment.java)

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    View v = LayoutInflater.from(getActivity())
        .inflate(R.layout.dialog_date, null);

    return new AlertDialog.Builder(getActivity())
        .setView(v)
        .setTitle(R.string.date_picker_title)
        .setPositiveButton(android.R.string.ok, null)
        .create();
}
```

Запустите приложение CriminalIntent. Нажмите кнопку даты и убедитесь в том, что в диалоговом окне теперь отображается `DatePicker`. На устройствах с версией Lollipop отображается календарный виджет (рис. 12.6).

Календарная версия на рис. 12.6 появилась вместе с концепцией материального дизайна (material design). Эта версия виджета `DatePicker` игнорирует атрибут `calendarViewShown`, заданный в макете. Но на устройствах с более старыми версиями Android вы увидите старую «дисковую» версию `DatePicker`, которая учитывает значение этого атрибута (рис. 12.7).

Обе версии работают. Впрочем, новая выглядит лучше.

Почему мы возимся с определением и заполнением макета, когда объект `DatePicker` можно было бы создать в коде так, как показано ниже?

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    DatePicker datePicker = new DatePicker(getActivity());

    return new AlertDialog.Builder(getActivity())
        .setView(datePicker)
        ...
        .create();
}
```

Рис. 12.6. DatePicker с календарем

Рис. 12.7. AlertDialog с DatePicker

Использование макета упрощает изменения, если вы меняете содержимое диалогового окна. Предположим, вы захотели, чтобы рядом с `DatePicker` в диалоговом окне отображался виджет `TimePicker`. При использовании заполнения можно просто обновить файл макета и новое представление появится на экране.

Также обратите внимание на то, что дата, выбранная в `DatePicker`, автоматически сохраняется при поворотах (проверьте сами). Как это происходит? Вспомните, что представления могут сохранять состояние между изменениями конфигурации, но только в том случае, если у них есть атрибут `id`. При создании `DatePicker` в dialog_date.xml вы также приказали инструментарию построения программы сгенерировать уникальный идентификатор для этого виджета `DatePicker`.

Если бы виджет `DatePicker` создавался в коде, то для сохранения состояния вам пришлось бы назначить идентификатор `DatePicker` на программном уровне.

Итак, наше диалоговое окно успешно отображается. В следующем разделе мы свяжем его с полем даты `Crime` и позаботимся о том, чтобы пользователь мог вводить данные.

Передача данных между фрагментами

Мы передавали данные между двумя активностями; мы передавали данные между двумя фрагментными активностями. Теперь нужно передать данные между дву-

мя фрагментами, хостом которых является одна активность, — `CrimeFragment` и `DatePickerFragment` (рис. 12.8).

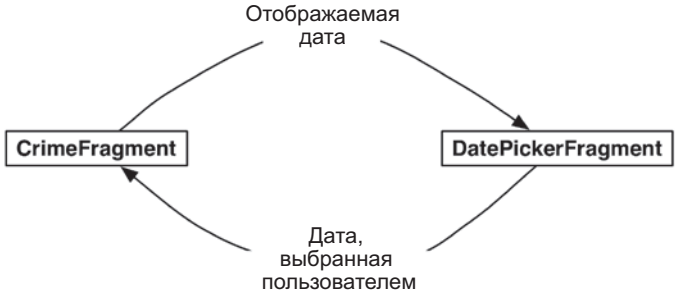


Рис. 12.8. Взаимодействие между CrimeFragment и DatePickerFragment

Чтобы передать дату преступления `DatePickerFragment`, мы напишем метод `newInstance(Date)` и сделаем объект `Date` аргументом фрагмента.

Чтобы вернуть новую дату фрагменту `CrimeFragment` для обновления уровня модели и его собственного представления, мы упакуем ее как дополнение объекта `Intent` и передадим этот объект `Intent` в вызове `CrimeFragment.onActivityResult(…)` (рис. 12.9).

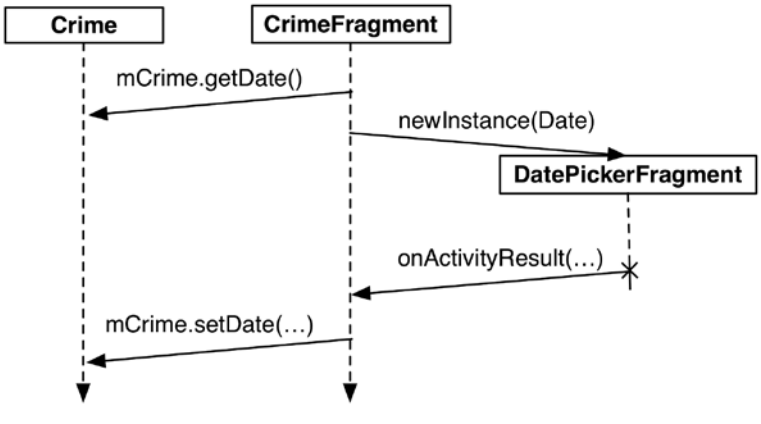


Рис. 12.9. Последовательность событий взаимодействия между CrimeFragment и DatePickerFragment

Вызов `Fragment.onActivityResult(…)` может показаться странным с учетом того, что активность-хост не получает вызова `Activity.onActivityResult(…)` в этом взаимодействии. Но, как будет показано далее в этой главе, использование `onActivityResult(…)` для передачи данных от одного фрагмента к другому не только работает, но и улучшает гибкость отображения фрагмента диалогового окна.

Передача данных DatePickerFragment

Чтобы получить данные в `DatePickerFragment`, мы сохраним дату в пакете аргументов `DatePickerFragment`, где `DatePickerFragment` сможет обратиться к ней.

Создание аргументов фрагмента и присваивание им значений обычно выполняется в методе `newInstance()`, заменяющем конструктор фрагмента. Добавьте в файл DatePickerFragment.java метод `newInstance(Date)`.

Листинг 12.5. Добавление метода newInstance(Date) (DatePickerFragment.java)

```
public class DatePickerFragment extends DialogFragment {

    private static final String ARG_DATE = "date";

    private DatePicker mDatePicker;

    public static DatePickerFragment newInstance(Date date) {
        Bundle args = new Bundle();
        args.putSerializable(ARG_DATE, date);

        DatePickerFragment fragment = new DatePickerFragment();
        fragment.setArguments(args);
        return fragment;
    }
    ...
}
```

В классе `CrimeFragment` удалите вызов конструктора `DatePickerFragment` и замените его вызовом `DatePickerFragment.newInstance(Date)`.

Листинг 12.6. Добавление вызова newInstance() (CrimeFragment.java)

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    ...
    mDateButton = (Button)v.findViewById(R.id.crime_date);
    mDateButton.setText(mCrime.getDate().toString());
    mDateButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            FragmentManager manager = getFragmentManager();
            DatePickerFragment dialog = new DatePickerFragment();
            DatePickerFragment dialog = DatePickerFragment
                .newInstance(mCrime.getDate());
            dialog.show(manager, DIALOG_DATE);
        }
    });
    ...
    return v;
}
```

Экземпляр `DatePickerFragment` должен инициализировать `DatePicker` по информации, хранящейся в `Date`. Однако для инициализации `DatePicker` необходимо иметь целочисленные значения месяца, дня и года. Объект `Date` больше напоминает временную метку и не может предоставить нужные целые значения напрямую.

Чтобы получить нужные значения, следует создать объект `Calendar` и использовать `Date` для определения его конфигурации. После этого вы сможете получить нужную информацию из `Calendar`.

В методе `onCreateDialog(…)` получите объект `Date` из аргументов и используйте его с `Calendar` для инициализации `DatePicker`.

Листинг 12.7. Извлечение даты и инициализация DatePicker (DatePickerFragment.java)

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    Date date = (Date) getArguments().getSerializable(ARG_DATE);

    Calendar calendar = Calendar.getInstance();
    calendar.setTime(date);
    int year = calendar.get(Calendar.YEAR);
    int month = calendar.get(Calendar.MONTH);
    int day = calendar.get(Calendar.DAY_OF_MONTH);

    View v = LayoutInflater.from(getActivity())
            .inflate(R.layout.dialog_date, null);

    mDatePicker = (DatePicker) v.findViewById(R.id.dialog_date_picker);
    mDatePicker.init(year, month, day, null);

    return new AlertDialog.Builder(getActivity())
            .setView(v)
            .setTitle(R.string.date_picker_title)
            .setPositiveButton(android.R.string.ok, null)
            .create();
}
```

Теперь `CrimeFragment` успешно сообщает `DatePickerFragment`, какую дату следует отобразить. Вы можете запустить приложение CriminalIntent и убедиться в том, что все работает так же, как прежде.

Возвращение данных CrimeFragment

Чтобы экземпляр `CrimeFragment` получал данные от `DatePickerFragment`, нам необходимо каким-то образом отслеживать отношения между двумя фрагментами.

С активностями вы вызываете `startActivityForResult(…)`, а `ActivityManager` отслеживает отношения между родительской и дочерней активностью. Когда дочерняя активность прекращает существование, `ActivityManager` знает, какая активность должна получить результат.

Назначение целевого фрагмента

Для создания аналогичной связи можно назначить `CrimeFragment` *целевым фрагментом* (target fragment) для `DatePickerFragment`. Эта связь будет автоматически восстановлена после того, как и `CrimeFragment`, и `DatePickerFragment` будут уничтожены и заново созданы ОС. Для этого вызывается следующий метод `Fragment`:

```
public void setTargetFragment(Fragment fragment, int requestCode)
```

Метод получает фрагмент, который станет целевым, и код запроса, аналогичный передаваемому `startActivityForResult(…)`. По коду запроса целевой фрагмент позднее может определить, какой фрагмент возвращает информацию.

`FragmentManager` сохраняет целевой фрагмент и код запроса. Чтобы получить их, вызовите `getTargetFragment()` и `getTargetRequestCode()` для фрагмента, назначившего целевой фрагмент.

В файле CrimeFragment.java создайте константу для кода запроса, а затем назначьте `CrimeFragment` целевым фрагментом экземпляра `DatePickerFragment`.

Листинг 12.8. Назначение целевого фрагмента (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
    private static final String ARG_CRIME_ID = "crime_id";
    private static final String DIALOG_DATE = "DialogDate";

    private static final int REQUEST_DATE = 0;
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        ...
        mDateButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                FragmentManager manager = getFragmentManager();
                DatePickerFragment dialog = DatePickerFragment
                        .newInstance(mCrime.getDate());
                dialog.setTargetFragment(CrimeFragment.this, REQUEST_DATE);
                dialog.show(manager, DIALOG_DATE);
            }
        });
        ...
        return v;
    }
}
```

Передача данных целевому фрагменту

Итак, связь между `CrimeFragment` и `DatePickerFragment` создана, и теперь нужно вернуть дату `CrimeFragment`. Дата будет включена в объект `Intent` как дополнение.

Какой метод будет использоваться для передачи интента целевому фрагменту? Как ни странно, `DatePickerFragment` передаст его при вызове `CrimeFragment.onActivityResult(int, int, Intent)`.

Метод `Activity.onActivityResult(…)` вызывается `ActivityManager` для родительской активности при уничтожении дочерней активности. При работе с активностями вы не вызываете `Activity.onActivityResult(…)` самостоятельно; это делает `ActivityManager`. После того как активность получит вызов, экземпляр `FragmentManager` активности вызывает `Fragment.onActivityResult(…)` для соответствующего фрагмента.

Если хостом двух фрагментов является одна активность, то для возвращения данных можно воспользоваться методом `Fragment.onActivityResult(…)` и вызывать его непосредственно для целевого фрагмента. Он содержит все необходимое:

- код запроса, соответствующий коду, переданному `setTargetFragment(…)`, по которому целевой фрагмент узнает, кто возвращает результат;
- код результата для определения выполняемого действия;
- экземпляр `Intent`, который может содержать дополнительные данные.

В классе `DatePickerFragment` создайте закрытый метод, который создает интент, помещает в него дату как дополнение, а затем вызывает `CrimeFragment.onActivityResult(…)`.

Листинг 12.9. Обратный вызов целевого фрагмента (DatePickerFragment.java)

```
public class DatePickerFragment extends DialogFragment {

    public static final String EXTRA_DATE =
            "com.bignerdranch.android.criminalintent.date";

    private static final String ARG_DATE = "date";
    ...

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        ...
    }

    private void sendResult(int resultCode, Date date) {
        if (getTargetFragment() == null) {
            return;
        }

        Intent intent = new Intent();
        intent.putExtra(EXTRA_DATE, date);

        getTargetFragment()
                .onActivityResult(getTargetRequestCode(), resultCode, intent);
    }
}
```

Пришло время воспользоваться новым методом `sendResult(...)`. Когда пользователь нажимает кнопку положительного ответа в диалоговом окне, приложение должно получить дату из `DatePicker` и отправить результат `CrimeFragment`. В коде `onCreateDialog(…)` замените параметр `null` вызова `setPositiveButton(…)` реали-

зацией DialogInterface.OnClickListener, которая возвращает выбранную дату и вызывает sendResult(…).

Листинг 12.10. Передача информации (DatePickerFragment.java)

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    ...
    return new AlertDialog.Builder(getActivity())
        .setView(v)
        .setTitle(R.string.date_picker_title)
        .setPositiveButton(android.R.string.ok, null);
        .setPositiveButton(android.R.string.ok,
                new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        int year = mDatePicker.getYear();
                        int month = mDatePicker.getMonth();
                        int day = mDatePicker.getDayOfMonth();
                        Date date = new GregorianCalendar(year, month, day).
                                    getTime();
                        sendResult(Activity.RESULT_OK, date);
                    }
        })
        .create();
}
```

В классе CrimeFragment переопределите метод onActivityResult(…), чтобы он возвращал дополнение, задавал дату в Crime и обновлял текст кнопки даты.

Листинг 12.11. Реакция на получение данных от диалогового окна (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
    ...

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        ...
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (resultCode != Activity.RESULT_OK) {
            return;
        }
        if (requestCode == REQUEST_DATE) {
            Date date = (Date) data
                    .getSerializableExtra(DatePickerFragment.EXTRA_DATE);
            mCrime.setDate(date);
            mDateButton.setText(mCrime.getDate().toString());
        }
    }
}
```

Код, задающий текст кнопки, идентичен коду из `onCreateView(…)`. Чтобы избежать задания текста в двух местах, мы инкапсулируем этот код в закрытом методе `updateDate()`, а затем вызовем его в `onCreateView(…)` и `onActivityResult(…)`.

Вы можете сделать это вручную или поручить работу Android Studio. Выделите всю строку кода, которая задает текст `mDateButton`.

Листинг 12.12. Выделение строки с обновлением кнопки даты (CrimeFragment.java)

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode != Activity.RESULT_OK) {
        return;
    }

    if (requestCode == REQUEST_DATE) {
        Date date = (Date) data
                .getSerializableExtra(DatePickerFragment.EXTRA_DATE);
        mCrime.setDate(date);
        mDateButton.setText(mCrime.getDate().toString());
    }
}
```

Щелкните на ней правой кнопкой мыши и выберите команду Refactor ▸ Extract ▸ Method... (рис. 12.10).

Выберите закрытый уровень видимости метода и введите имя `updateDate`. Щелкните на кнопке OK; среда Android Studio сообщит, что ей удалось найти еще одно место, в котором использовалась эта строка кода. Щелкните на кнопке Yes, чтобы разрешить Android Studio обновить второе вхождение. Убедитесь в том, что код был выделен в метод `updateDate` (листинг 12.13).

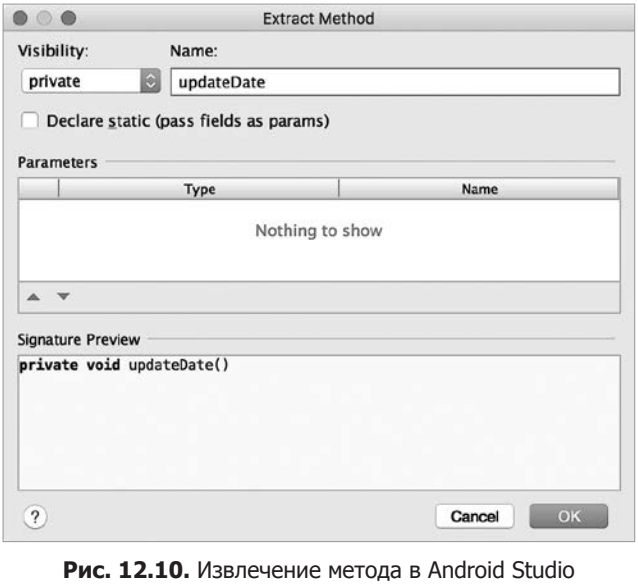


Рис. 12.10. Извлечение метода в Android Studio

Листинг 12.13. Выделение кода в метод updateDate() (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime, container, false);
        ...
        mDateButton = (Button) v.findViewById(R.id.crime_date);
        updateDate();
        ...
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (resultCode != Activity.RESULT_OK) {
            return;
        }

        if (requestCode == REQUEST_DATE) {
            Date date = (Date) data
                    .getSerializableExtra(DatePickerFragment.EXTRA_DATE);
            mCrime.setDate(date);
            updateDate();
        }
    }

    private void updateDate() {
        mDateButton.setText(mCrime.getDate().toString());
    }
}
```

Круг замкнулся — данные передаются туда и обратно.

Запустите приложение CriminalIntent и убедитесь в том, что вы действительно можете управлять датой. Измените дату `Crime`; новая дата должна появиться в представлении `CrimeFragment`. Вернитесь к списку преступлений, проверьте дату `Crime` и убедитесь в том, что уровень модели действительно обновлен.

Больше гибкости в представлении DialogFragment

Использование `onActivityResult(…)` для возвращения данных целевому фрагменту особенно удобно, когда ваше приложение получает много данных от пользователя и нуждается в большем пространстве для их ввода. При этом приложение должно хорошо работать на телефонах и планшетах.

На экране телефона свободного места не так много, поэтому вы, скорее всего, используете для ввода данных активность с полноэкранным фрагментом. Дочерняя активность будет запускаться вызовом `startActivityForResult()` из фрагмента родительской активности. При уничтожении дочерней активности родительская активность будет получать вызов `onActivityResult(…)`, который будет перенаправляться фрагменту, запустившему дочернюю активность (рис. 12.11).

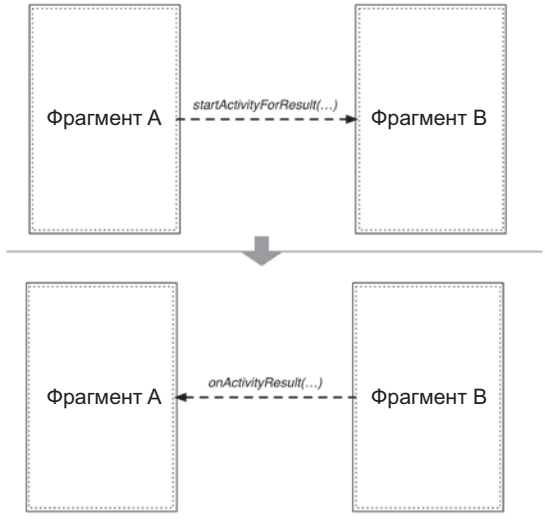


Рис. 12.11. Взаимодействие между активностями на телефонах

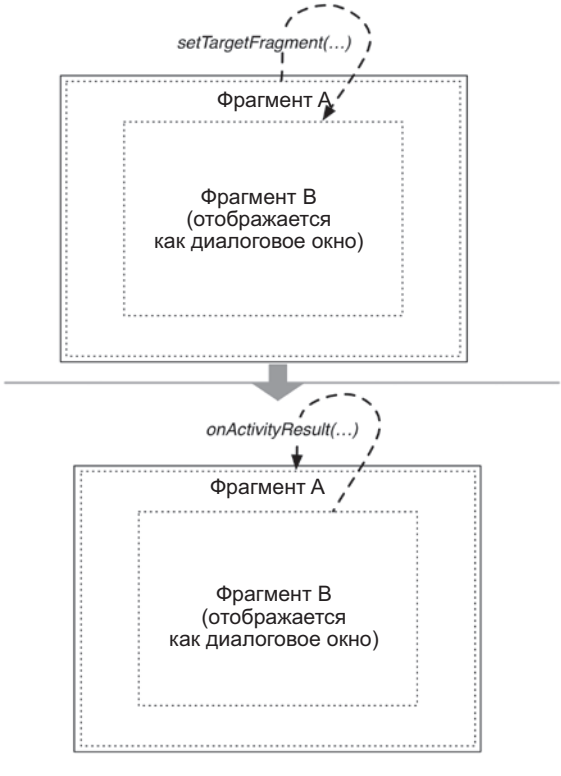


Рис. 12.12. Взаимодействие между фрагментами на планшетах

На планшетах, где экранного пространства больше, часто бывает лучше отобразить DialogFragment для ввода тех же данных. В таком случае вы задаете целевой фрагмент и вызываете show(…) для фрагмента диалогового окна. При закрытии фрагмент диалогового окна вызывает для своей цели onActivityResult(…) (рис. 12.12).

Метод onActivityResult(…) фрагмента будет вызываться всегда, независимо от того, запустил ли фрагмент активность или отобразил диалоговое окно. Следовательно, мы можем использовать один код для разных вариантов представления информации.

Когда один код используется и для полноэкранного, и для диалогового фрагмента, для подготовки вывода в обоих случаях вместо onCreateDialog(…) можно переопределить DialogFragment.onCreateView(…).

Упражнение. Новые диалоговые окна

Напишите еще один диалоговый фрагмент TimePickerFragment для выбора времени преступления. Используйте виджет TimePicker, добавьте в CrimeFragment еще одну кнопку для отображения TimePickerFragment.

Упражнение. DialogFragment

Попробуйте справиться с более сложной задачей: изменением представления DatePickerFragment.

Первая часть упражнения — предоставить представление DatePickerFragment с переопределением onCreateView(…) вместо onCreateDialog(Bundle). При таком способе создания DialogFragment не будет отображаться со встроенными областями заголовка и кнопок в верхней и нижней частях диалогового окна. Вам придется самостоятельно создать кнопку OK в dialog_date.xml.

После того как представление DatePickerFragment будет создано в onCreateView(…), вы можете отобразить фрагмент DatePickerFragment как диалоговое окно или встроить его в активность. Во второй части упражнения создайте новый субкласс SingleFragmentActivity и сделайте эту активность хостом для DatePickerFragment.

При таком представлении DatePickerFragment вы будете использовать механизм startActivityForResult для возвращения даты CrimeFragment. В DatePickerFragment, если целевой фрагмент не существует, используйте метод setResult(int, intent) активности-хоста для возвращения даты фрагменту.

В последней части этого упражнения измените приложение CriminalIntent так, чтобы фрагмент DatePickerFragment отображался как полноэкранная активность при запуске на телефоне. На планшете DatePickerFragment должен отображаться как диалоговое окно. Возможно, вам стоит забежать вперед и прочитать в главе 17 о том, как оптимизировать приложение для разных размеров экрана.

13 Панель инструментов

Панель инструментов (toolbar) является ключевым компонентом любого хорошо спроектированного приложения Android. Панель инструментов содержит действия, которые могут выполняться пользователем, и новые средства навигации, а также обеспечивает единство дизайна и фирменного стиля.

В этой главе мы создадим для приложения CriminalIntent меню, которое будет отображаться на панели инструментов. В этом меню будет присутствовать *элемент действия* (action item) для добавления нового преступления. Также мы обеспечим работу кнопки Up на панели инструментов (рис. 13.1).

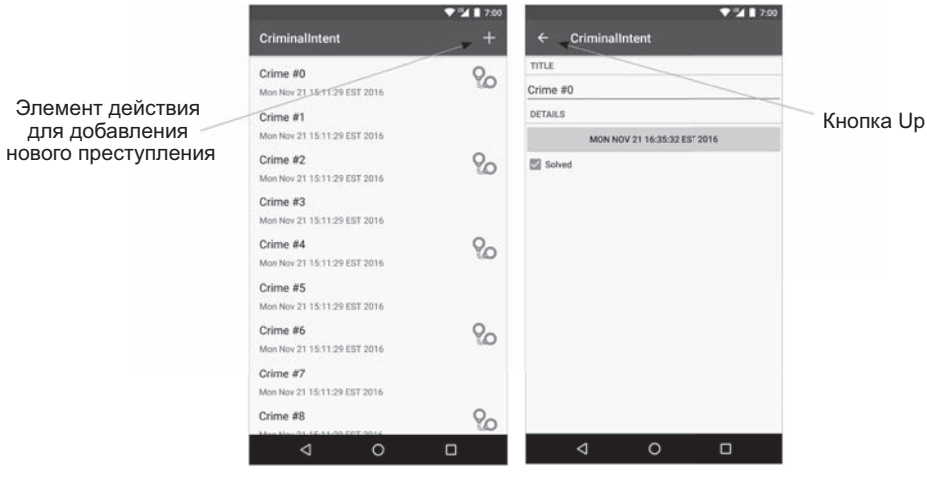


Рис. 13.1. Панель инструментов CriminalIntent

AppCompat

Компонент панели инструментов появился в Android с выходом Android 5.0 (Lollipop). До Lollipop для навигации и размещения действий в приложении рекомендовалось использовать *панель действий* (action bar).

Панель действий и панель инструментов очень похожи. Панель инструментов расширяет возможности панели действий: она обладает улучшенным пользовательским интерфейсом и отличается большей гибкостью в использовании.

Приложение CriminalIntent поддерживает API уровня 19; это означает, что вы не сможете использовать встроенную реализацию панели инструментов во всех поддерживаемых версиях Android. К счастью, панель инструментов была адаптирована для библиотеки AppCompat. Библиотека AppCompat позволяет реализовать функциональность панели инструментов Lollipop в любой версии Android вплоть до API 9 (Android 2.3).

Использование библиотеки AppCompat

Библиотека AppCompat вам уже знакома. На момент написания книги она включается в новые проекты автоматически. А если вам потребуется включить поддержку AppCompat в старый проект? Полная интеграция с AppCompat требует ряда дополнительных шагов.

Для использования библиотеки AppCompat необходимо:

- добавить зависимость AppCompat;
- использовать одну из тем AppCompat;
- проследить за тем, чтобы все активности были субклассами `AppCompatActivity`.

Обновление темы

Так как зависимость для AppCompat уже добавлена, пора сделать следующий шаг и убедиться в том, что вы используете одну из тем (themes) AppCompat. Библиотека AppCompat включает три темы:

- `Theme.AppCompat` — темная;
- `Theme.AppCompat.Light` — светлая;
- `Theme.AppCompat.Light.DarkActionBar` — светлая с темной панелью инструментов.

Тема приложения задается на уровне приложения; также существует необязательная возможность назначения темы на уровне активностей в файле AndroidManifest.xml. Откройте файл AndroidManifest.xml и найдите тег `application`. Обратите внимание на атрибут `android:theme`. Он выглядит примерно так, как показано в листинге 13.1.

Листинг 13.1. Стандартный манифест (AndroidManifest.xml)

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme" >
```

`AppTheme` определяется в файле res/values/styles.xml. Откройте этот файл и убедитесь в том, что атрибут `parent` элемента `AppTheme` соответствует выделенной части в листинге 13.2. Пока не обращайте внимания на атрибуты внутри темы, вскоре мы их обновим.

Листинг 13.2. Использование темы AppCompat (res/values/styles.xml)

```
<resources>

    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
      <!-- Здесь настраивается ваша тема. -->
      <item name="colorPrimary">@color/colorPrimary</item>
      <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
      <item name="colorAccent">@color/colorAccent</item>
    </style>

</resources>
```

Стили и темы более подробно рассматриваются в главе 22.

Использование AppCompatActivity

Остается сделать последний шаг на пути использования AppCompat: проверьте, что все активности субклассируют `AppCompatActivity`. Мы используем `AppCompatActivity` начиная с главы 7, так что никакие изменения не потребуются.

Если вы вносили какие-либо изменения в этой главе, запустите CriminalIntent и убедитесь в том, что запуск происходит без сбоев. Приложение должно выглядеть примерно так, как показано на рис. 13.2.

Теперь можно добавлять действия на панель инструментов.

Меню

Правая верхняя часть панели инструментов зарезервирована для меню. Меню состоит из элементов действий (иногда также называемых *элементами меню*), выполняющих действия на текущем экране или в приложении в целом. Мы добавим элемент действия, при помощи которого пользователь сможет создать описание нового преступления.

Для работы меню потребуется несколько строковых ресурсов. Добавьте их в файл strings.xml (листинг 13.3). Пока эти строки выглядят довольно загадочно, но лучше решить эту проблему сразу. Когда они понадобятся нам позднее, они уже будут на своем месте, и нам не придется отвлекаться от текущих дел.

Листинг 13.3. Добавление строк для меню (res/values/strings.xml)

```
<resources>
    ...
    <string name="date_picker_title">Date of crime:</string>
    <string name="new_crime">New Crime</string>
    <string name="show_subtitle">Show Subtitle</string>
    <string name="hide_subtitle">Hide Subtitle</string>
    <string name="subtitle_format">%1$d crimes</string>

</resources>
```

Рис. 13.2. Панель инструментов

Определение меню в XML

Меню определяются такими же ресурсами, как и макеты. Вы создаете описание меню в XML и помещаете файл в каталог res/menu своего проекта. Android генерирует идентификатор ресурса для файла меню, который затем используется для заполнения меню в коде.

В окне инструментов Project щелкните правой кнопкой мыши на каталоге res и выберите команду New ▸ Android resource file. Выберите тип ресурса Menu, присвойте ресурсу меню имя fragment_crime_list и щелкните на кнопке OK (рис. 13.3).

Здесь для файлов меню используется та же схема формирования имен, что и для файлов макетов. Android Studio генерирует файл res/menu/fragment_crime_list.xml: его имя совпадает с именем файла макета `CrimeListFragment`, но файл находится в папке menu. В новом файле переключитесь в режим представления Text и добавьте элемент `item` (листинг 13.4).

Листинг 13.4. Создание ресурса меню CrimeListFragment (res/menu/fragment_crime_list.xml)

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/new_crime"
        android:icon="@android:drawable/ic_menu_add"
```

```
        android:title="@string/new_crime"
        app:showAsAction="ifRoom|withText"/>
</menu>
```

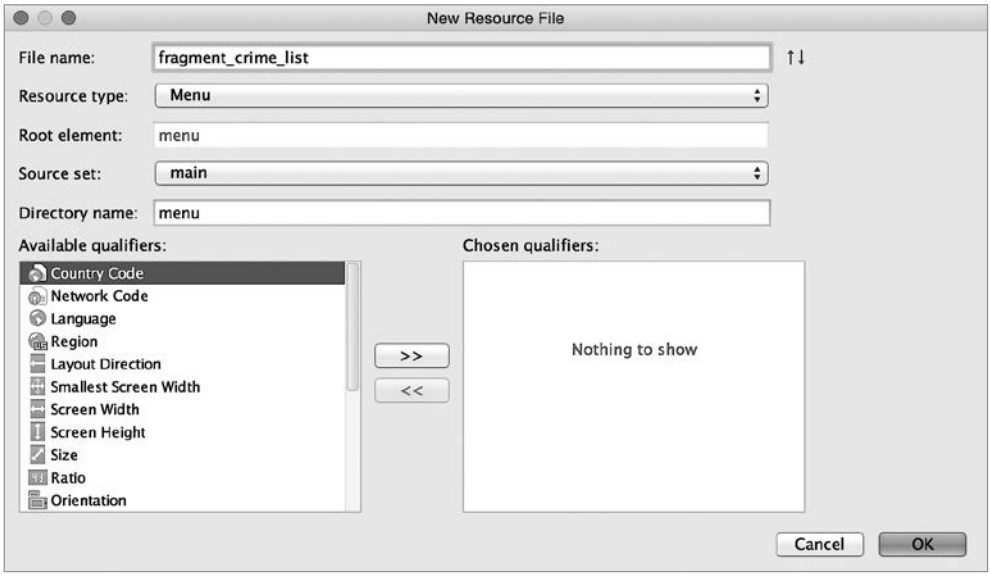


Рис. 13.3. Создание файла меню

Атрибут `showAsAction` показывает, должна команда меню отображаться на самой панели инструментов или в *дополнительном меню* (overflow menu). Мы объединили два значения, `ifRoom` и `withText`, чтобы при наличии свободного места на панели инструментов отображался значок и текст команды. Если на панели не хватает места для текста, то отображается только значок. Если места нет ни для того ни для другого, команда перемещается в дополнительное меню.

Дополнительное меню вызывается значком в виде трех точек в правой части панели инструментов (рис. 13.4). Вскоре мы изменим код для добавления новых команд меню.

Также атрибут `showAsAction` может принимать значения `always` и `never`. Выбирать `always` не рекомендуется; лучше использовать `ifRoom` и предоставить решение ОС. Вариант `never` хорошо подходит для редко выполняемых действий. Как правило, на панели инструментов следует размещать только часто используемые команды меню, чтобы не загромождать экран.

Пространство имен app

Заметьте, что в файле fragment_crime_list.xml тег `xmlns` используется для определения нового пространства имен `app` отдельно от обычного объявления пространства имен `android`. Затем пространство имен `app` используется для назначения атрибута `showAsAction`.

Рис. 13.4. Дополнительное меню на панели инструментов

Это необычное объявление пространства имен существует для обеспечения совместимости с библиотекой AppCompat. API панели действий впервые появился в Android 3.0. Изначально библиотека AppCompat была создана для включения совместимой версии панели действий, поддерживающей более ранние версии Android, чтобы панель действий могла использоваться на любых устройствах, даже на не поддерживающих встроенную панель действий. На устройствах с Android 2.3 и более ранними версиями меню и соответствующая разметка XML не существовали, но атрибут `android:showAsAction` добавился только с выпуском панели действий.

Библиотека AppCompat определяет собственный атрибут `showAsAction`, игнорируя системную реализацию `showAsAction`.

Android Asset Studio

В атрибуте `android:icon` значение `@android:drawable/ic_menu_add` ссылается на *системный значок* (system icon). Системные значки находятся на устройстве, а не в ресурсах проекта.

В прототипе приложения ссылки на системные значки работают нормально. Однако в приложении, готовом к выпуску, лучше быть уверенным в том, что именно пользователь увидит на экране. Системные значки могут сильно различаться между устройствами и версиями ОС, а на некоторых устройствах системные значки могут не соответствовать дизайну приложения.

Одно из возможных решений — создание собственных значков. Вам придется подготовить версии для каждого разрешения экрана, а возможно, и для других конфигураций устройств. За дополнительной информацией обращайтесь к руководству «Android's Icon Design Guidelines» по адресу *developer.android.com/design/style/iconography.html*.

Также можно действовать иначе: найти системные значки, соответствующие потребностям вашего приложения, и скопировать их прямо в графические ресурсы проекта.

Системные значки находятся в каталоге Android SDK. На Mac это обычно каталог вида /Users/*пользователь*/Library/Android/sdk. В Windows по умолчанию используется путь \Users*пользователь*\sdk. Также для получения местонахождения SDK можно открыть окно Project Structure и выбрать категорию SDK Location.

В каталоге SDK вы найдете разные ресурсы Android, включая `ic_menu_add`. Эти ресурсы находятся в каталоге /platforms/android-25/data/res, где 25 — уровень API Android-версии.

Третий, и самый простой, вариант — использовать программу Android Asset Studio, включенную в Android Studio. Asset Studio позволяет создать и настроить изображение для использования на панели инструментов.

Щелкните правой кнопкой мыши на каталоге drawable в окне инструментов Project и выберите команду New ▸ Image Asset. На экране появится Asset Studio (рис. 13.5).

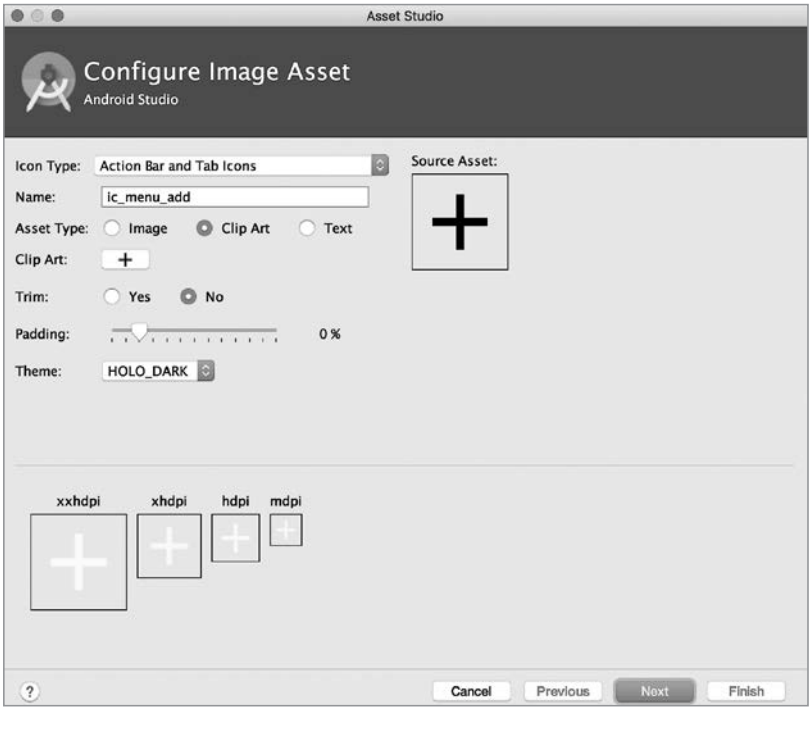


Рис. 13.5. Asset Studio

В Asset Studio можно генерировать значки нескольких разных типов. Выберите в поле Icon Type: вариант Action Bar and Tab Icons. Присвойте ресурсу имя ic_menu_add, затем в группе Asset Type выберите переключатель Clip Art. В списке Theme выберите вариант HOLO_DARK. Так как панель инструментов использует темную тему оформления, изображение должно казаться светлым. Эти изменения отражены на рис. 13.5; обратите внимание: хотя на иллюстрации представлена графическая заготовка, которую вы сейчас выберете, на вашем экране будет изображен логотип Android.

Нажмите кнопку Clip Art, чтобы выбрать графическую заготовку. В открывшемся окне выберите изображение, напоминающее знак «+» (рис. 13.6).

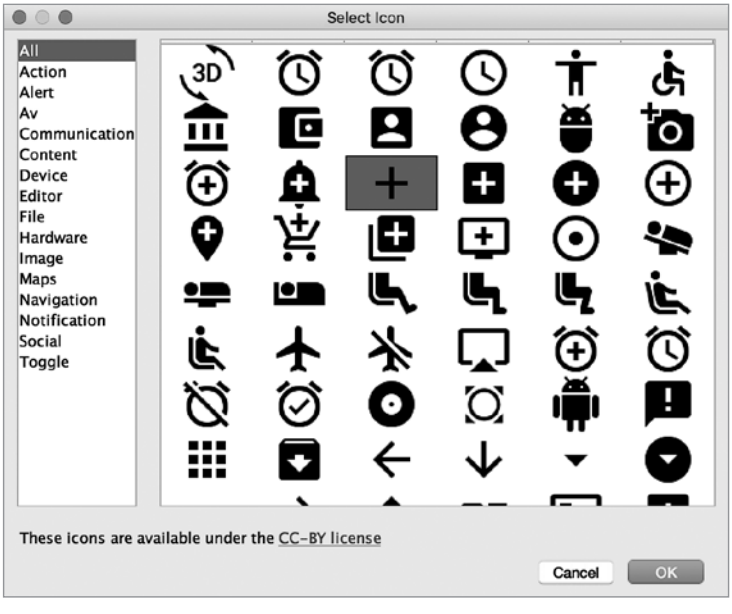


Рис. 13.6. Галерея графических заготовок: где знак «+»?

На основном экране нажмите кнопку Next, чтобы перейти к последнему шагу мастера. Asset Studio также выводит план работы, которую выполнит Asset Studio. Обратите внимание: значки `mdpi`, `hdpi`, `xhdpi` и `xxhdpi` будут созданы автоматически. Класс!

Щелкните на кнопке Finish, чтобы сгенерировать изображения. Затем в файле макета измените атрибут `icon` и включите в него ссылку на новый ресурс из вашего проекта.

Листинг 13.5. Ссылка на локальный ресурс (res/menu/fragment_crime_list.xml)

```
<item
    android:id="@+id/new_crime"
    ~~android:icon="@android:drawable/ic_menu_add"~~
    android:icon="@drawable/ic_menu_add"
    android:title="@string/new_crime"
    app:showAsAction="ifRoom|withText"/>
```

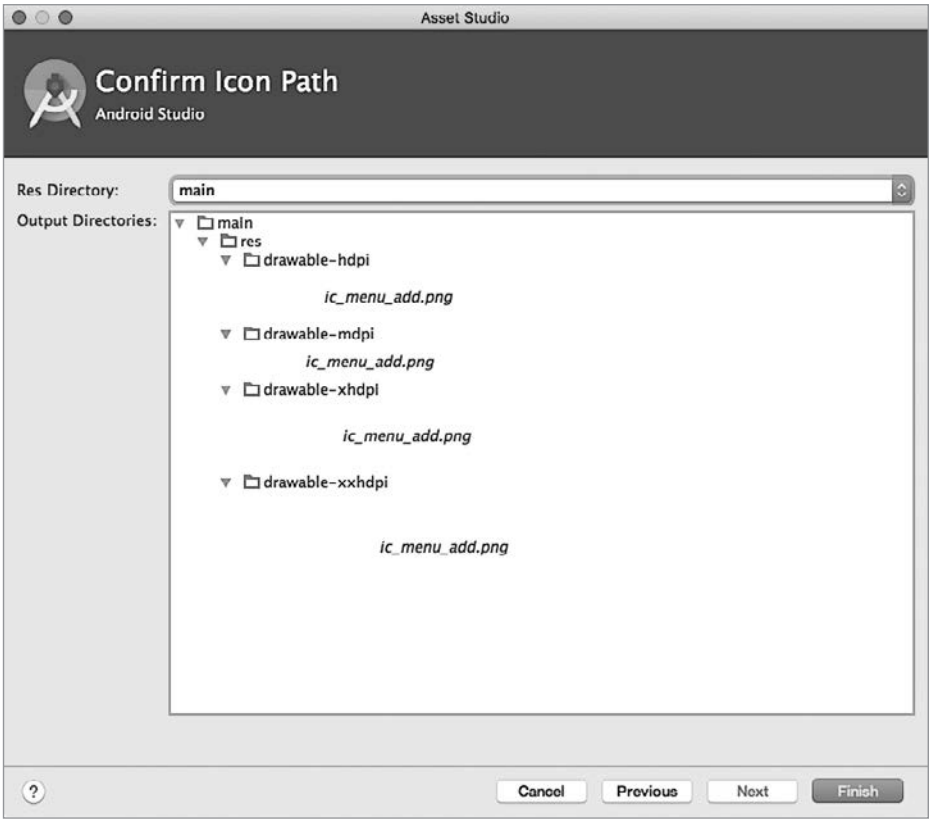


Рис. 13.7. Файлы, сгенерированные Android Studio

Создание меню

Для управления меню в коде используются методы обратного вызова класса `Activity`. Когда возникает необходимость в меню, Android вызывает метод `Activity` с именем `onCreateOptionsMenu(Menu)`.

Однако архитектура нашего приложения требует, чтобы реализация находилась во фрагменте, а не в активности. Класс `Fragment` содержит собственный набор методов обратного вызова для командных меню, которые мы реализуем в `CrimeListFragment`. Для создания меню и обработки выбранных команд используются следующие методы:

```
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater)
public boolean onOptionsItemSelected(MenuItem item)
```

В файле CrimeListFragment.java переопределите метод `onCreateOptionsMenu(Menu, MenuInflater)` так, чтобы он заполнял меню, определенное в файле fragment_crime_list.xml.

Листинг 13.6. Заполнение ресурса меню (CrimeListFragment.java)

```
@Override
public void onResume() {
    super.onResume();
    updateUI();
}

@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_crime_list, menu);
}
...
```

В этом методе мы вызываем метод `MenuInflater.inflate(int, Menu)` и передаем идентификатор ресурса своего файла меню. Вызов заполняет экземпляр `Menu` командами, определенными в файле.

Обратите внимание на вызов реализации `onCreateOptionsMenu(…)` суперкласса. Он не обязателен, но мы рекомендуем вызывать версию суперкласса просто для соблюдения общепринятой схемы, чтобы работала вся функциональность меню, определяемая в суперклассе. Впрочем, в данном случае это лишь формальность — базовая реализация этого метода из `Fragment` не делает ничего.

`FragmentManager` отвечает за вызов `Fragment.onCreateOptionsMenu(Menu, MenuInflater)` при получении активностью обратного вызова `onCreateOptionsMenu(…)` от ОС. Вы должны явно указать `FragmentManager`, что фрагмент должен получить вызов `onCreateOptionsMenu(…)`. Для этого вызывается следующий метод:

```
public void setHasOptionsMenu(boolean hasMenu)
```

В методе `CrimeListFragment.onCreate(Bundle)` сообщите `FragmentManager`, что экземпляр `CrimeListFragment` должен получать обратные вызовы меню.

Листинг 13.7. Получение обратных вызовов (CrimeListFragment.java)

```
public class CrimeListFragment extends Fragment {

    private RecyclerView mCrimeRecyclerView;
    private CrimeAdapter mAdapter;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setHasOptionsMenu(true);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
```

В приложении CriminalIntent появляется меню (рис. 13.8).

Где текст элемента меню? У большинства телефонов в книжной ориентации хватает места только для значка. Текст команды открывается долгим нажатием на значке на панели инструментов (рис. 13.9).

Рис. 13.8. Значок команды меню на панели инструментов

Рис. 13.9. Долгое нажатие на значке на панели инструментов выводит текст команды

В альбомной ориентации на панели инструментов хватает места как для значка, так и для текста (рис. 13.10).

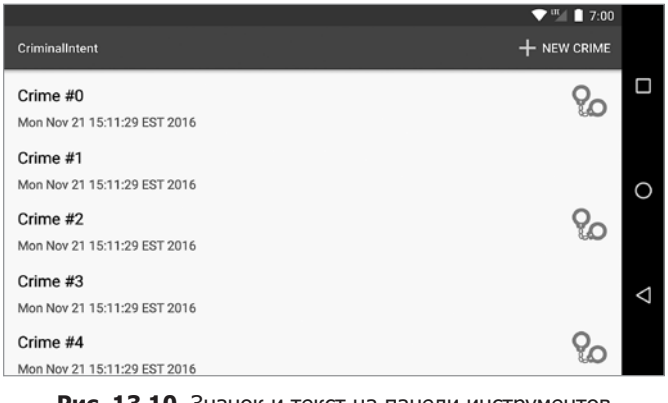


Рис. 13.10. Значок и текст на панели инструментов

Реакция на выбор команд

Чтобы отреагировать на выбор пользователем команды New Crime, нам понадобится механизм добавления нового объекта `Crime` в список. Включите в файл CrimeLab.java следующий метод:

Листинг 13.8. Добавление нового объекта Crime (CrimeLab.java)

```
...
public void addCrime(Crime c) {
    mCrimes.add(c);
}

public List<Crime> getCrimes() {
    return mCrimes;
}
...
```

Теперь, когда вы сможете вводить описания преступлений самостоятельно, программное генерирование 100 объектов становится лишним. В файле CrimeLab.java удалите код, генерирующий эти преступления.

Листинг 13.9. Долой случайные преступления! (CrimeLab.java)

```
private CrimeLab(Context context) {
    mCrimes = new ArrayList<>();
    for (int i = 0; i < 100; i++) {
        Crime crime = new Crime();
        crime.setTitle("Crime #" + i);
        crime.setSolved(i % 2 == 0); // Для каждого второго объекта
        mCrimes.add(crime);
    }
}
```

Когда пользователь выбирает команду в командном меню, фрагмент получает обратный вызов метода `onOptionsItemSelected(MenuItem)`. Этот метод получает экземпляр `MenuItem`, описывающий выбор пользователя.

И хотя наше меню состоит всего из одной команды, в реальных меню их обычно больше. Чтобы определить, какая команда меню была выбрана, проверьте идентификатор команды меню и отреагируйте соответствующим образом. Этот идентификатор соответствует идентификатору, назначенному команде в файле меню.

В файле CrimeListFragment.java реализуйте метод `onOptionsItemSelected(MenuItem)`, реагирующий на выбор команды меню. Реализация создает новый объект `Crime`, добавляет его в `CrimeLab` и запускает экземпляр `CrimePagerActivity` для редактирования нового объекта `Crime`.

Листинг 13.10. Реакция на выбор команды меню (CrimeListFragment.java)

```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_crime_list, menu);
```

```
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.new_crime:
            Crime crime = new Crime();
            CrimeLab.get(getActivity()).addCrime(crime);
            Intent intent = CrimePagerActivity
                    .newIntent(getActivity(), crime.getId());
            startActivity(intent);
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Метод возвращает логическое значение. После того как команда меню будет обработана, верните `true`; тем самым вы сообщаете, что дальнейшая обработка не нужна. Секция `default` вызывает реализацию суперкласса, если идентификатор команды не известен в вашей реализации.

Запустите приложение CriminalIntent и опробуйте новую команду. Добавьте несколько преступлений и отредактируйте их. (Пустой список до начала ввода может сбить с толку пользователя. Упражнение в конце этой главы будет выдавать подсказку для пользователя при пустом списке.)

Включение иерархической навигации

Пока приложение CriminalIntent использует кнопку Back для навигации по приложению. Кнопка Back возвращает приложение к предыдущему состоянию; этот механизм называется *временной навигацией*. С другой стороны, *иерархическая навигация* осуществляет перемещение по иерархии приложения.

В иерархической навигации пользователь переходит на один уровень «наверх» к родителю текущей активности при помощи кнопки Up в левой части панели инструментов.

Чтобы включить иерархическую навигацию в приложение CriminalIntent, добавьте атрибут `parentActivityName` в файл AndroidManifest.xml.

Листинг 13.11. Включение кнопки Up (AndroidManifest.xml)

```
<activity
    android:name=".CrimePagerActivity"
    android:parentActivityName=".CrimeListActivity">
</activity>
```

Запустите приложение CriminalIntent и создайте новое преступление. Обратите внимание на появление кнопки Up (рис. 13.11). Нажатие кнопки Up переводит приложение на один уровень вверх в иерархии CriminalIntent к активности `CrimeListActivity`.

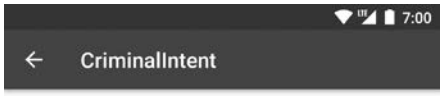


Рис. 13.11. Кнопка Up в CrimePagerActivity

Как работает иерархическая навигация

В приложении CriminalIntent навигация кнопкой Back и кнопкой Up выполняет одну и ту же задачу. Нажатие любой из этих кнопок в `CrimePagerActivity` возвращает пользователя обратно к `CrimeListActivity`. И хотя результат один и тот же, «за кулисами» происходят совершенно разные события. Эти различия важны, потому что в зависимости от приложения кнопка Up может вернуть пользователя на несколько активностей назад в стеке.

Когда пользователь переходит вверх по иерархии из `CrimeActivity`, создается интент следующего вида:

```
Intent intent = new Intent(this, CrimeListActivity.class);
intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
startActivity(intent);
finish();
```

Флаг `FLAG_ACTIVITY_CLEAR_TOP` приказывает Android провести поиск существующего экземпляра активности в стеке и, если он будет найден, вывести из стека все остальные активности, чтобы запускаемая активность была верхней (рис. 13.12).

Рис. 13.12. FLAG_ACTIVITY_CLEAR_TOP в действии

Альтернативная команда меню

В этом разделе все, что мы узнали о меню, совместимости и альтернативных ресурсах, будет использовано для добавления команды меню, скрывающей и отображающей подзаголовок в панели инструментов `CrimeListActivity`.

В файле res/menu/fragment_crime_list.xml добавьте элемент действия SHOW SUBTITLE, который будет отображаться на панели инструментов при наличии свободного

места. (Запись символами верхнего регистра обусловлена встроенными стилями панели инструментов; ранее вы видели аналогичное форматирование на кнопках.)

Листинг 13.12. Добавление элемента действия Show Subtitle (res/menu/fragment_crime_list.xml)

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/new_crime"
        android:icon="@android:drawable/ic_menu_add"
        android:title="@string/new_crime"
        app:showAsAction="ifRoom|withText"/>

    <item
        android:id="@+id/show_subtitle"
        android:title="@string/show_subtitle"
        app:showAsAction="ifRoom"/>
</menu>
```

Создайте новый метод `updateSubtitle()`, который будет задавать подзаголовок с количеством преступлений на панели инструментов.

Листинг 13.13. Назначение подзаголовка панели инструментов

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    ...
}

private void updateSubtitle() {
    CrimeLab crimeLab = CrimeLab.get(getActivity());
    int crimeCount = crimeLab.getCrimes().size();
    String subtitle = getString(R.string.subtitle_format, crimeCount);
    AppCompatActivity activity = (AppCompatActivity) getActivity();
    activity.getSupportActionBar().setSubtitle(subtitle);
}
    ...
```

Метод `updateSubtitle()` генерирует строку подзаголовка при помощи метода `getString(int resId, Object… formatArgs)`, который получает значения, подставляемые на место заполнителей в строковом ресурсе.

Затем активность, являющаяся хостом для `CrimeListFragment`, преобразуется в `AppCompatActivity`. CriminalIntent использует библиотеку AppCompat, поэтому все активности должны быть субклассами `AppCompatActivity`, чтобы панель инструментов была доступной для приложения. (По историческим причинам панель инструментов во многих местах библиотеки AppCompat называется «панелью действий».)

Итак, теперь метод `updateSubtitle()` определен, и мы можем вызвать его при нажатии нового элемента действий.

Листинг 13.14. Обработка элемента действия Show Subtitle (CrimeListFragment.java)

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.new_crime:
            ...
            return true;
        case R.id.show_subtitle:
            updateSubtitle();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Запустите CriminalIntent, нажмите элемент SHOW SUBTITLE и убедитесь в том, что в подзаголовке отображается количество преступлений.

Переключение текста команды

Теперь подзаголовок отображается, но текст команды меню остался неизменным: SHOW SUBTITLE. Было бы лучше, если бы текст команды и функциональность команды меню изменялись в зависимости от текущего состояния подзаголовка.

При вызове `onOptionsItemSelected(MenuItem)` в параметре передается объект `MenuItem` для элемента действия, нажатого пользователем. Текст элемента SHOW SUBTITLE можно было бы обновить в этом методе, но изменения будут потеряны при повороте устройства и повторном создании панели инструментов.

Другое, более правильное решение — обновить объект `MenuItem` для SHOW SUBTITLE в `onCreateOptionsMenu(…)` и инициировать повторное создание панели инструментов при нажатии на элементе действия. Это позволит вам заново использовать код обновления элемента действия как при выборе элемента действия пользователем, так и при повторном создании панели инструментов.

Сначала добавьте переменную для хранения признака видимости подзаголовка.

Листинг 13.15. Хранение признака видимости подзаголовка (CrimeListFragment.java)

```
public class CrimeListFragment extends Fragment {

    private RecyclerView mCrimeRecyclerView;
    private CrimeAdapter mAdapter;
    private boolean mSubtitleVisible;
    ...
```

Затем измените подзаголовок в `onCreateOptionsMenu(…)` и инициируйте повторное создание элементов действий при нажатии элемента действия SHOW SUBTITLE.

Листинг 13.16. Обновление MenuItem (CrimeListFragment.java)

```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_crime_list, menu);

    MenuItem subtitleItem = menu.findItem(R.id.show_subtitle);
    if (mSubtitleVisible) {
        subtitleItem.setTitle(R.string.hide_subtitle);
    } else {
        subtitleItem.setTitle(R.string.show_subtitle);
    }
}
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.new_crime:
            ...
        case R.id.show_subtitle:
            mSubtitleVisible = !mSubtitleVisible;
            getActivity().invalidateOptionsMenu();
            updateSubtitle();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Наконец, проверьте переменную `mSubtitleVisible` при отображении или сокрытии подзаголовка панели инструментов.

Листинг 13.17. Отображение или сокрытие подзаголовка (CrimeListFragment.java)

```
private void updateSubtitle() {
    CrimeLab crimeLab = CrimeLab.get(getActivity());
    int crimeCount = crimeLab.getCrimes().size();
    String subtitle = getString(R.string.subtitle_format, crimeCount);

    if (!mSubtitleVisible) {
        subtitle = null;
    }

    AppCompatActivity activity = (AppCompatActivity) getActivity();
    activity.getSupportActionBar().setSubtitle(subtitle);
}
```

Запустите приложение CriminalIntent и убедитесь в том, что подзаголовок успешно скрывается и отображается. Обратите внимание на изменение текста команды в зависимости от состояния подзаголовка.

«Да, и еще кое-что...»

Программирование Android часто напоминает беседы с лейтенантом Коломбо из сериала. Вы уже думаете, что все прошло как по маслу и у следствия нет претензий. Но Android всегда поворачивается у двери и говорит: «Ах да, извините, еще кое-что...»

Впрочем, у нас целых два «кое-что». Во-первых, при создании нового преступления и последующем возвращении к `CrimeListActivity` кнопкой Back содержимое подзаголовка не соответствует новому количеству преступлений. Во-вторых, при повороте устройства подзаголовок исчезнет.

Начнем с первой проблемы. Она решается обновлением текста подзаголовка при возвращении к `CrimeListActivity`. Инициируйте вызов `updateSubtitle()` в `onResume()`. Ваш метод `updateUI` уже вызывается в `onResume` и `onCreate`; добавьте вызов `updateSubtitle()` в метод `updateUI()`.

Листинг 13.18. Вывод обновленного состояния (CrimeListFragment.java)

```
private void updateUI() {
    CrimeLab crimeLab = CrimeLab.get(getActivity());
    List<Crime> crimes = crimeLab.getCrimes();

    if (mAdapter == null) {
        mAdapter = new CrimeAdapter(crimes);
        mCrimeRecyclerView.setAdapter(mAdapter);
    } else {
        mAdapter.notifyDataSetChanged();
    }

    updateSubtitle();
}
```

Запустите CriminalIntent, отобразите подзаголовок, создайте новое преступление и нажмите на устройстве кнопку Back, чтобы вернуться к `CrimeListActivity`. На этот раз на панели инструментов отображается правильное количество.

Повторите эти действия, но вместо кнопки Back воспользуйтесь кнопкой Up. Подзаголовок снова становится невидимым. Почему это происходит?

У реализации иерархической навигации в Android имеется неприятный побочный эффект: активность, к которой вы переходите кнопкой Up, полностью создается заново, с нуля. Это означает, что теряются все переменные экземпляров, следовательно, и все сохраненное состояние экземпляров. Родительская активность воспринимается как совершенно новая активность.

Не существует простого способа обеспечить вывод подзаголовка при переходе кнопкой Up. Одно из возможных решений — переопределение механизма перехода. В CriminalIntent можно вызвать метод `finish()` для `CrimePagerActivity`, чтобы вернуться к предыдущей активности. Такое решение прекрасно сработает в CriminalIntent, но не подойдет для решений с более реалистичной иерархией, так как возврат произойдет только на одну активность.

Другое возможное решение — передача `CrimePagerActivity` информации о видимости подзаголовка в дополнительных данных интента при запуске. Переопределите метод `getParentActivityIntent()` в `CrimePagerActivity`, чтобы добавить дополнение в интент, используемый для воссоздания `CrimeListActivity`. Это решение требует, чтобы класс `CrimePagerActivity` обладал подробной информацией о том, как работает его родитель.

Оба решения не идеальны, причем хорошей альтернативы не существует. По этой причине мы просто оставим проблему как есть; пользователю придется лишний раз выбрать команду SHOW SUBTITLE, но это не такой уж значительный труд.

Теперь в подзаголовке всегда отображается правильное количество преступлений, и мы можем заняться решением проблемы с поворотом. Для этого следует сохранить переменную экземпляра `mSubtitleVisible` между поворотами при помощи механизма сохранения состояния экземпляров.

Листинг 13.19. Сохранение признака видимости подзаголовка (CrimeListFragment.java)

```
public class CrimeListFragment extends Fragment {

    private static final String SAVED_SUBTITLE_VISIBLE = "subtitle";
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        ...
        if (savedInstanceState != null) {
            mSubtitleVisible = savedInstanceState.getBoolean
                               (SAVED_SUBTITLE_VISIBLE);
        }

        updateUI();

        return view;
    }

    @Override
    public void onResume() {
        ...
    }

    @Override
    public void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        outState.putBoolean(SAVED_SUBTITLE_VISIBLE, mSubtitleVisible);
    }
}
```

Запустите приложение CriminalIntent. Отобразите подзаголовок, поверните устройство. Подзаголовок должен появиться в воссозданном представлении, как и ожидалось (рис. 13.13).

Рис. 13.13. Панель действий и панель инструментов

Для любознательных: панели инструментов и панели действий

Чем панель инструментов отличается от панели действий?

Наиболее очевидное различие между ними — измененный визуальный стиль панели инструментов. В левой части панели инструментов нет знака, а интервалы между элементами действий в правой части сокращаются. Другое существенное визуальное изменение — кнопка Up. На панели действий эта кнопка была менее заметной и играла второстепенную роль.

Помимо этих визуальных различий, панели инструментов проектировались прежде всего с расчетом на бóльшую гибкость, чем у панели действий. Панель действий должна подчиняться множеству ограничений. Она всегда отображается у верхнего края экрана. У приложения может быть только одна панель действий. Размер панели действий фиксирован, и изменить его невозможно. У панелей инструментов этих ограничений нет.

В этой главе мы использовали панель инструментов, позаимствованную из тем AppCompat. Также можно вручную включить панель инструментов как обычное представление в файл макета активности или фрагмента. Панель инструментов можно разместить где угодно, и на экране могут одновременно находиться несколько панелей инструментов. Эта гибкость открывает интересные возможности: например, представьте, что каждый фрагмент, используемый в вашем приложении, поддерживает собственную панель инструментов. При одновременном размещении на экране нескольких фрагментов каждый из них может отображать собственную панель инструментов вместо того, чтобы совместно использовать одну панель инструментов у верхнего края экрана.

Другое интересное дополнение к панели инструментов — возможность размещения `View` на панели инструментов и регулировки ее высоты. Все это значительно расширяет гибкость работы приложений.

Упражнение. Удаление преступлений

Приложение CriminalIntent дает возможность создать новое преступление, но не предоставляет средств для удаления его из «протокола». В этом упражнении добавьте в `CrimeFragment` новый элемент действия для удаления текущего преступления. После того как пользователь нажмет элемент удаления, не забудьте вернуть его к предыдущей активности вызовом метода `finish()` для активности-хоста `CrimeFragment`.

Упражнение. Множественное число в строках

Если список содержит одно преступление, подзаголовок «1 crimes» становится грамматически неправильным. В этом упражнении вам предлагается исправить текст подзаголовка.

Вы можете создать две разные строки и выбрать нужную в коде, но такое решение быстро разваливается при попытке локализовать приложение для разных языков. Лучше использовать строковые ресурсы (иногда называемые количественными строками) во множественном числе.

Сначала определите в файле strings.xml элемент `plurals`:

```
<plurals name="subtitle_plural">
    <item quantity="one">%1$d crime</item>
    <item quantity="other">%1$d crimes</item>
</plurals>
```

Затем используйте метод `getQuantityString` для правильного образования множественного числа:

```
int crimeSize = crimeLab.getCrimes().size();
String subtitle = getResources()
    .getQuantityString(R.plurals.subtitle_plural, crimeSize, crimeSize);
```

Упражнение. Пустое представление для списка

В настоящее время при запуске CriminalIntent отображает пустой виджет `RecyclerView` — большую белую пустоту. Мы должны предоставить пользователям что-то для взаимодействия при отсутствии элементов в списке.

Пусть в пустом представлении выводится сообщение (например, «Список пуст»). Добавьте в представление кнопку, которая будет инициировать создание нового преступления.

Для отображения и сокрытия нового представления-заполнителя используйте метод `setVisibility`, существующий в каждом классе `View`.

14 Базы данных SQLite

Почти каждому приложению необходимо место для долгосрочного хранения данных — более длительного, чем позволяет `savedInstanceState`. Android предоставляет вам такое место: локальную файловую систему во флеш-памяти телефона или планшета.

Каждое приложение на устройстве Android имеет каталог в своей *песочнице* (sandbox). Хранение файлов в песочнице защищает их от других приложений и даже от любопытных глаз пользователей (если только устройство не было «взломано» — в этом случае пользователь сможет делать все, что ему заблагорассудится).

Песочница каждого приложения представляет собой подкаталог каталога /data/data, имя которого соответствует имени пакета приложения. Для CriminalIntent полный путь к каталогу песочницы имеет вид /data/data/com.bignerdranch.android.criminalintent.

Тем не менее большинство данных приложений не хранится в простых файлах. И на то есть веская причина: допустим, у вас имеется файл, в котором записаны данные всех объектов `Crime`. Чтобы изменить краткое описание преступления в начале файла, вам придется прочитать весь файл и записать его заново. При большом количестве записей эта процедура займет много времени.

На помощь приходит SQLite — реляционная база данных с открытым кодом, как и MySQL или Postgresql. В отличие от других баз данных, SQLite хранит свои данные в простых файлах, для чтения и записи которых может использоваться библиотека SQLite. Библиотека SQLite входит в стандартную библиотеку Android вместе с другими вспомогательными классами Java.

В этой главе описаны не все возможности SQLite. Более подробную информацию можно найти в полной документации SQLite по адресу *www.sqlite.org*. Здесь будет показано, как работают основные классы поддержки SQLite в Android. Вы сможете выполнять операции открытия, чтения и записи с базами данных SQLite в песочнице приложения, даже не зная, где именно они хранятся.

Определение схемы

Прежде чем создавать базу данных, необходимо решить, какая информация будет в ней храниться. В CriminalIntent хранится только список преступлений, поэтому мы определим одну таблицу с именем `crimes` (рис. 14.1).

_id	uuid	title	date	solved
1	13090636733242	Stolen yogurt	13090636733242	0
2	13090732131909	Dirty sink	13090732131909	1

Рис. 14.1. Таблица crimes

В мире программирования у задач такого рода существует много разных решений, но разработчик должен неизменно руководствоваться принципом DRY. Это сокращение (от «Don't Repeat Yourself», то есть «Не повторяйтесь») обозначает одно из практических правил, которое нужно соблюдать при написании программы: когда вы пишете какой-то код, запишите его в одном месте. В этом случае вместо того, чтобы плодить дубликаты, вы всегда будете обращаться к одному централизованному источнику информации.

Соблюдение этого принципа при работе с базами данных может быть весьма непростой задачей. Существуют сложные программы, называемые средствами объектно-реляционного отображения (Object-Relational Mappers, сокращенно ORM), которые позволяют использовать объекты модели (такие, как `Crime`) как Единственно Верное Определение. В этой главе мы пойдем по более простому пути и определим в коде Java упрощенную *схему базы данных*, в которой будут храниться имя таблицы и описания ее столбцов.

Начните с создания класса для хранения схемы. Этот класс будет называться `CrimeDbSchema`, но вы должны ввести в диалоговом окне New Class имя `database.CrimeDbSchema`. Файл CrimeDbSchema.java будет помещен в отдельный пакет `database`, который будет использоваться для организации всего кода, относящегося к базам данных.

В классе `CrimeDbSchema` определите внутренний класс `CrimeTable` для описания таблицы.

Листинг 14.1. Определение CrimeTable (CrimeDbSchema.java)

```
public class CrimeDbSchema {
    public static final class CrimeTable {
        public static final String NAME = "crimes";
    }
}
```

Класс `CrimeTable` существует только для определения строковых констант, необходимых для описания основных частей определения таблицы. Определение начинается с имени таблицы в базе данных `CrimeTable.NAME`, за которым следуют описания столбцов.

Листинг 14.2. Определение столбцов таблицы (CrimeDbSchema.java)

```
public class CrimeDbSchema {
    public static final class CrimeTable {
        public static final String NAME = "crimes";

        public static final class Cols {
```

```
            public static final String UUID = "uuid";
            public static final String TITLE = "title";
            public static final String DATE = "date";
            public static final String SOLVED = "solved";
        }
    }
}
```

При наличии такого определения вы сможете обращаться к столбцу с именем title в синтаксисе, безопасном для кода Java: `CrimeTable.Cols.TITLE`. Такой синтаксис существенно снижает риск изменения программы, если вам когда-нибудь понадобится изменить имя столбца или добавить новые данные в таблицу.

Построение исходной базы данных

После определения схемы можно переходить к созданию базы данных. Android предоставляет в классе `Context` низкоуровневые методы для открытия файла базы данных в экземпляр `SQLiteDatabase`: `openOrCreateDatabase(…)` и `databaseList()`.

Тем не менее на практике при открытии базы данных всегда следует выполнить ряд простых действий:

1. Проверить, существует ли база данных.
2. Если база данных не существует, создать ее, создать таблицы и заполнить их необходимыми исходными данными.
3. Если база данных существует, открыть ее и проверить версию `CrimeDbSchema` (возможно, в будущих версиях CriminalIntent вы захотите добавить или удалить какие-то аспекты).
4. Если это старая версия, выполнить код преобразования ее в новую версию.

Android предоставляет класс `SQLiteOpenHelper`, который сделает это все за вас. Создайте в пакете `database` класс с именем `CrimeBaseHelper`.

Листинг 14.3. Создание CrimeBaseHelper (CrimeBaseHelper.java)

```
public class CrimeBaseHelper extends SQLiteOpenHelper {
    private static final int VERSION = 1;
    private static final String DATABASE_NAME = "crimeBase.db";

    public CrimeBaseHelper(Context context) {
        super(context, DATABASE_NAME, null, VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }
}
```

Класс `SQLiteOpenHelper` избавляет разработчика от рутинной работы при открытии `SQLiteDatabase`. Используйте его в `CrimeLab` для создания базы данных.

Листинг 14.4. Открытие SQLiteDatabase (CrimeLab.java)

```
public class CrimeLab {
    private static CrimeLab sCrimeLab;

    private List<Crime> mCrimes;
    private Context mContext;
    private SQLiteDatabase mDatabase;
    ...
    private CrimeLab(Context context) {
      mContext = context.getApplicationContext();
      mDatabase = new CrimeBaseHelper(mContext)
              .getWritableDatabase();
      mCrimes = new ArrayList<>();
    }
```

(Интересуетесь, зачем контекст сохраняется в переменной экземпляра? Он будет использоваться `CrimeLab` в главе 16.)

При вызове `getWritableDatabase()` класс `CrimeBaseHelper` выполняет следующее:

- открывает /data/data/com.bignerdranch.android.criminalintent/databases/crimeBase.db. Если файл базы данных не существует, то он создается;
- если база данных открывается впервые, вызывает метод `onCreate(SQLiteDatabase)` с последующим сохранением последнего номера версии;
- если база данных открывается не впервые, проверяет номер ее версии. Если версия базы данных в `CrimeOpenHelper` выше, то вызывается метод `onUpgrade(SQLiteDatabase, int, int)`.

Мораль: код создания исходной базы данных размещается в `onCreate(SQLiteDatabase)`, код обновления — в `onUpgrade(SQLiteDatabase, int, int)`, а дальше все работает само собой.

Пока приложение CriminalIntent существует только в одной версии, так что на `onUpgrade(…)` можно не обращать внимания. Нужно только создать таблицы базы данных в `onCreate(…)`. Для этого будет использоваться класс `CrimeTable`, являющийся внутренним классом `CrimeDbSchema`.

Процедура импортирования состоит из двух шагов. Сначала запишите начальную часть кода создания SQL.

Листинг 14.5. Первая часть onCreate(SQLiteDatabase) (CrimeBaseHelper.java)

```
@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL("create table " + CrimeDbSchema.CrimeTable.NAME);
}
```

Наведите курсор на слово `CrimeTable` и нажмите Option+Return (Alt+Enter). Затем выберите первый вариант Add import for 'com.bignerdranch.android.criminalintent.database.CrimeDbSchema.CrimeTable', как показано на рис. 14.2.

Рис. 14.2. Добавление директивы import для CrimeTable

Android Studio генерирует директиву `import` следующего вида:

```
import com.bignerdranch.android.criminalintent.database.CrimeDbSchema.CrimeTable;
public class CrimeBaseHelper extends SQLiteOpenHelper {
```

Директива позволяет ссылаться на строковые константы из `CrimeDbSchema.CrimeTable` в форме `CrimeTable.Cols.UUID` (вместо того, чтобы вводить полное имя `CrimeDbSchema.CrimeTable.Cols.UUID`). Используйте это обстоятельство, чтобы завершить ввод кода определения таблицы.

Листинг 14.6. Создание таблицы (CrimeBaseHelper.java)

```
@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL("create table " + CrimeTable.NAME + "(" +
            " _id integer primary key autoincrement, " +
            CrimeTable.Cols.UUID + ", " +
            CrimeTable.Cols.TITLE + ", " +
            CrimeTable.Cols.DATE + ", " +
            CrimeTable.Cols.SOLVED +
            ")"
    );
}
```

Создание таблицы в SQLite происходит проще, чем в других базах данных: вам не нужно задавать тип столбца в момент создания. Сделать это желательно, но мы предпочтем сэкономить немного времени.

Запустите CriminalIntent; приложение создаст базу данных. Если приложение выполняется в эмуляторе или на «рутованном» устройстве, вы сможете увидеть созданную базу данных. (На физическом устройстве это невозможно — база данных хранится в закрытой области.)

На момент написания книги образы эмулятора для Nougat (API 24 и 25) не работали с менеджером файлов Android Device Monitor. Чтобы увидеть эти файлы, вы должны установить приложение в эмуляторе, работающем на более старых версиях Android. Если вы забыли, как настраивать эмулятор, обратитесь к разделу «Выполнение в эмуляторе» главы 1.

Работа с файлами в Android Device Monitor

Если в вашем эмуляторе используется API 23 и выше, откройте Android Device Monitor. Выполните команду Tools ▸ Android ▸ Android Device Monitor из главного

меню. Если откроется диалоговое окно, в котором вам предлагается отключить интеграцию ADB, щелкните на кнопке Yes (рис. 14.3).

Рис. 14.3. Отключение интеграции ADB

Когда откроется экран Android Device Monitor, щелкните на вкладке File Explorer. Чтобы убедиться в том, что файлы базы данных CriminalIntent были успешно созданы, загляните в папку /data/data/com.bignerdranch.android.criminalintent/databases/ (рис. 14.4).

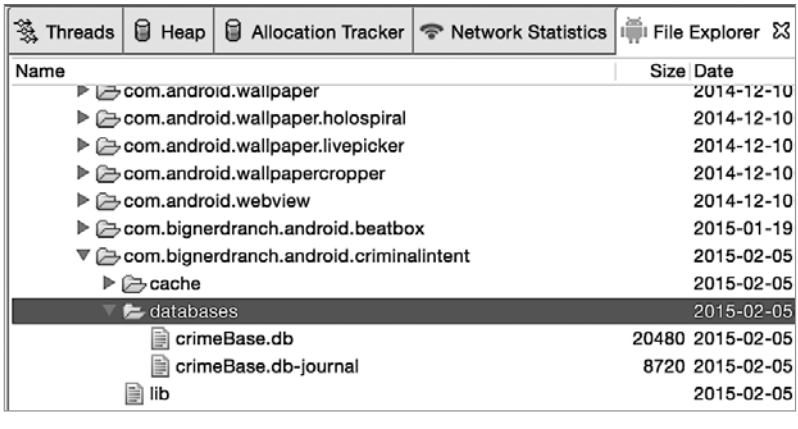


Рис. 14.4. Ваша база данных

Если вы попробуете запустить приложение с отключенной интеграцией ADB, вы увидите следующую ошибку: Instant Run requires 'Tools | Android | Enable ADB integration' to be enabled. Чтобы исправить ошибку, выберите команду Android Studio ▸ Preferences в главном меню. На открывшемся экране введите в поле поиска наверху слева «Instant Run» (рис. 14.5). Снимите верхний флажок Enable Instant Run, чтобы отключить отслеживание изменений в коде/ресурсах (по умолчанию оно включено). Щелкните на кнопке Apply, после чего закройте диалоговое окно кнопкой OK.

Решение проблем при работе с базами данных

При написании кода для работы с базами данных SQLite иногда требуется слегка изменить структуру базы данных. Например, в следующей главе для каждого преступления будет добавлено новое поле подозреваемого. Для этого в таблицу

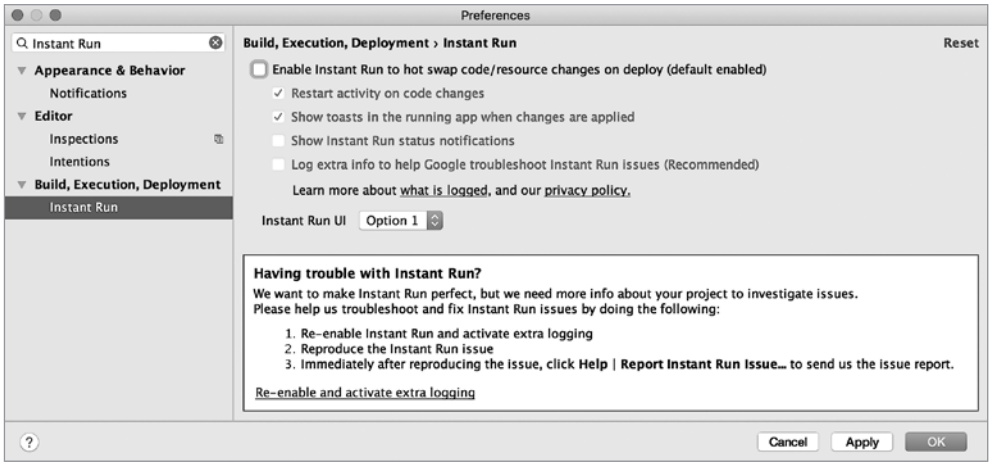


Рис. 14.5. Отключение режима мгновенного запуска

Рис. 14.6. Удаление приложения

преступлений придется добавить новый столбец. «Правильный» способ решения этой задачи заключается во включении в `SQLiteOpenHelper` кода повышения номера версии с последующим обновлением таблиц в `onUpgrade(…)`.

И этот «правильный» способ потребует изрядного объема кода — совершенно смехотворного, когда вы просто пытаетесь довести до ума первую или вторую версию базы данных. На практике лучше всего уничтожить базу данных и начать все заново, чтобы метод `SQLiteOpenHelper.onCreate(…)` был вызван снова.

Самый простой способ уничтожения базы — удаление приложения с устройства. А самый простой способ удаления приложений в стандартном варианте Android — открыть менеджер приложений и перетащить значок CriminalIntent к области Uninstall у верхнего края экрана. (Если на вашем устройстве используется нестандартная версия Android, процесс может выглядеть иначе.) Откроется экран вроде изображенного на рис. 14.6.

Помните об этом приеме, если у вас возникнут проблемы при работе с базами данных в этой главе.

Изменение кода CrimeLab

Итак, теперь у вас есть база данных, и нам предстоит изменить довольно большой объем кода в `CrimeLab`, чтобы для хранения данных вместо `mCrimes` использовалась база данных `mDatabase`.

Для начала расчистим место для работы. Удалите из `CrimeLab` весь код, относящийся к `mCrimes`.

Листинг 14.7. Удаляем лишнее (CrimeLab.java)

```
public class CrimeLab {
    private static CrimeLab sCrimeLab;

    ~~private List<Crime> mCrimes;~~
    private Context mContext;
    private SQLiteDatabase mDatabase;
    public static CrimeLab get(Context context) {
        ...
    }

    private CrimeLab(Context context) {
        mContext = context.getApplicationContext();
        mDatabase = new CrimeBaseHelper(mContext)
                .getWritableDatabase();
       ~~mCrimes = new ArrayList<>();~~
    }

    public void addCrime(Crime c) {
        ~~mCrimes.add(c);~~
    }

    public List<Crime> getCrimes() {
        ~~return mCrimes;~~
        return new ArrayList<>();
    }
```

```
    public Crime getCrime(UUID id) {
        for (Crime crime : mCrimes) {
            if (crime.getId().equals(id)) {
                return crime;
            }
        }
        return null;
    }
}
```

Приложение CriminalIntent остается в состоянии, которое вряд ли можно назвать работоспособным; вы видите пустой список преступлений, но при добавлении преступления отображается пустая активность `CrimePagerActivity`. Неприятно, но пока сойдет.

Запись в базу данных

Работа с SQLiteDatabase начинается с записи данных. Ваше приложение должно вставлять новые записи в существующую таблицу, а также обновлять уже имеющиеся данные при изменении информации.

Использование ContentValues

Запись и обновление баз данных осуществляются с помощью класса `ContentValues`. Класс `ContentValues` обеспечивает хранение пар «ключ-значение», как и контейнер Java `HashMap` или объекты `Bundle`, уже встречавшиеся вам ранее. Однако в отличие от `HashMap` или `Bundle`, он предназначен для хранения типов данных, которые могут содержаться в базах данных SQLite.

Экземпляры `ContentValues` будут несколько раз создаваться из `Crime` в коде `CrimeLab`. Добавьте закрытый метод, который будет преобразовывать объект `Crime` в `ContentValues`. (Не забудьте описанный ранее прием для добавления директивы `import` для `CrimeTable`: добравшись до `CrimeTable.Cols.UUID`, нажмите Option+Return (Alt+Enter) и выберите команду Add import for 'com.bignerdranch.android.criminalintent.database.CrimeDbSchema.CrimeTable'.)

Листинг 14.8. Создание ContentValues (CrimeLab.java)

```
    public Crime getCrime(UUID id) {
        return null;
    }

    private static ContentValues getContentValues(Crime crime) {
        ContentValues values = new ContentValues();
        values.put(CrimeTable.Cols.UUID, crime.getId().toString());
        values.put(CrimeTable.Cols.TITLE, crime.getTitle());
        values.put(CrimeTable.Cols.DATE, crime.getDate().getTime());
        values.put(CrimeTable.Cols.SOLVED, crime.isSolved() ? 1 : 0);
```

```
        return values;
    }
}
```

В качестве ключей используйте имена столбцов. Эти имена не выбираются произвольно; они определяют столбцы, которые должны вставляться или обновляться в базе данных. Если имя отличается от содержащегося в базе данных (например, из-за опечатки), операция вставки или обновления завершится неудачей. В приведенном фрагменте указаны все столбцы, кроме столбца `_id`, который генерируется автоматически для однозначной идентификации записи.

Вставка и обновление записей

Объект `ContentValues` создан, можно переходить к добавлению записи в базу данных. Заполните метод `addCrime(Crime)` новой реализацией.

Листинг 14.9. Вставка записи (CrimeLab.java)

```
public void addCrime(Crime c) {
    ContentValues values = getContentValues(c);

    mDatabase.insert(CrimeTable.NAME, null, values);
}
```

Метод `insert(String, String, ContentValues)` получает два важных аргумента; еще один аргумент используется относительно редко. Первый аргумент определяет таблицу, в которую выполняется вставка, — в нашем случае `CrimeTable.NAME`. Последний аргумент содержит вставляемые данные.

А второй аргумент, который называется `nullColumnHack`? Что же он делает?

Представьте, что вы решили вызвать `insert(…)` с пустым объектом `ContentValues`. SQLite такую вставку не поддерживает, поэтому попытка вызова `insert(…)` окончится неудачей.

Но если передать в `nullColumnHack` значение `uuid`, пустой объект `ContentValues` будет проигнорирован. Вместо него будет передан объект `ContentValues` с полем `uuid`, содержащим `null`. Это позволит методу `insert(…)` успешно выполниться и создать новую запись.

Пригодится? Когда-нибудь — возможно... Хотя не сегодня. Но по крайней мере, теперь вы знаете о такой возможности.

Продолжим использование `ContentValues` и напишем метод обновления строк в базе данных.

Листинг 14.10. Обновление записи (CrimeLab.java)

```
public Crime getCrime(UUID id) {
    return null;
}

public void updateCrime(Crime crime) {
```

```
    String uuidString = crime.getId().toString();
    ContentValues values = getContentValues(crime);

    mDatabase.update(CrimeTable.NAME, values,
            CrimeTable.Cols.UUID + " = ?",
            new String[] { uuidString });
}

private static ContentValues getContentValues(Crime crime) {
```

Метод `update(String, ContentValues, String, String[])` начинается так же, как `insert(…)`, — при вызове передается имя таблицы и объект `ContentValues`, который должен быть присвоен каждой обновляемой записи. Однако последняя часть отличается, потому что в этом случае необходимо указать, *какие именно* записи должны обновляться. Для этого строится условие `WHERE` (третий аргумент), за которым следуют значения аргументов в условии `WHERE` (завершающий массив `String[]`).

Почему мы не вставили `uuidString` прямо в условие `WHERE`? Ведь это проще, чем использовать `?` и передавать значение в `String[]`.

Дело в том, что в некоторых случаях сама строка может содержать код SQL. Если вставить ее содержимое прямо в запрос, этот код может изменить смысл запроса или даже модифицировать базу данных. Подобные ситуации, называемые *атаками внедрения* SQL, безусловно нежелательны.

Но при использовании `?` ваш код будет делать именно то, что положено: значение интерпретируется как строковые данные, а не как код. Таким образом, лучше перестраховаться и привыкнуть к использованию заполнителя `?`, который всегда приведет к желаемому результату независимо от содержимого строки.

Экземпляры `Crime`, изменяемые в `CrimeFragment`, должны быть записаны в базу данных при завершении `CrimeFragment`. Добавьте переопределение `CrimeFragment.onPause()`, которое обновляет копию `Crime` из `CrimeLab`.

Листинг 14.11. Запись обновлений (CrimeFragment.java)

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    UUID crimeId = (UUID) getArguments().getSerializable(ARG_CRIME_ID);
    mCrime = CrimeLab.get(getActivity()).getCrime(crimeId);
}

@Override
public void onPause() {
    super.onPause();

    CrimeLab.get(getActivity())
            .updateCrime(mCrime);
}
```

К сожалению, проверить работоспособность этого кода пока не удастся. Придется подождать, пока приложение не научится читать обновленные данные. Чтобы убедиться в том, что программа нормально компилируется, запустите CriminalIntent еще один раз, прежде чем переходить к следующему разделу. В приложении должен отображаться пустой список.

Чтение из базы данных

Чтение данных из SQLite осуществляется методом `query(…)`. При вызове `SQLiteDatabase.query(…)` происходит много всего; метод существует в нескольких перегруженных версиях. Версия, которую вы будете использовать, выглядит так:

```
public Cursor query(
    String table,
    String[] columns,
    String where,
    String[] whereArgs,
    String groupBy,
    String having,
    String orderBy,
    String limit)
```

Если у вас уже есть опыт работы с SQL, многие имена покажутся знакомыми по аргументам команды `SELECT`. Если же вы не имели дела с SQL, ограничьтесь только теми аргументами, которые вы будете использовать:

```
public Cursor query(
    String table,
    String[] columns,
    String where,
    String[] whereArgs,
    String groupBy,
    String having,
    String orderBy,
    String limit)
```

Аргумент `table` содержит таблицу, к которой обращен запрос. Аргумент `columns` определяет столбцы, значения которых вам нужны, и порядок их извлечения. Наконец, `where` и `whereArgs` работают так же, как и в команде `update(…)`.

Создайте вспомогательный метод, в котором будет вызываться метод `query(…)` для таблицы `CrimeTable`.

Листинг 14.12. Запрос для получения данных Crime (CrimeLab.java)

```
public void updateCrime(Crime crime) {
    ...
}

private Cursor queryCrimes(String whereClause, String[] whereArgs) {
```

```
    Cursor cursor = mDatabase.query(
            CrimeTable.NAME,
            null, // columns - с null выбираются все столбцы
            whereClause,
            whereArgs,
            null, // groupBy
            null, // having
            null  // orderBy
    );

    return cursor;
}
```

Использование CursorWrapper

Класс `Cursor` как средство работы с данными таблиц оставляет желать лучшего. По сути, он просто возвращает низкоуровневые значения столбцов. Процедура получения данных из `Cursor` выглядит так:

```
String uuidString = cursor.getString(
    cursor.getColumnIndex(CrimeTable.Cols.UUID));
String title = cursor.getString(
    cursor.getColumnIndex(CrimeTable.Cols.TITLE));
long date = cursor.getLong(
    cursor.getColumnIndex(CrimeTable.Cols.DATE));
int isSolved = cursor.getInt(
    cursor.getColumnIndex(CrimeTable.Cols.SOLVED));
```

Каждый раз, когда вы извлекаете `Crime` из курсора, этот код придется записывать снова. (Не говоря уже о коде создания экземпляра `Crime` с этими значениями!)

Вспомните правило DRY: «Не повторяйтесь». Вместо того чтобы записывать этот код при каждом чтении данных из курсора, вы можете создать собственный субкласс `Cursor`, который выполняет эту операцию в одном месте. Для написания субкласса курсора проще всего воспользоваться `CursorWrapper` — этот класс позволяет дополнить класс `Cursor`, полученный извне, новыми методами.

Создайте новый класс в пакете базы данных с именем `CrimeCursorWrapper`.

Листинг 14.13. Создание CrimeCursorWrapper (CrimeCursorWrapper.java)

```
public class CrimeCursorWrapper extends CursorWrapper {
    public CrimeCursorWrapper(Cursor cursor) {
        super(cursor);
    }
}
```

Класс создает тонкую «обертку» для `Cursor`. Он содержит те же методы, что и инкапсулированный класс `Cursor`, и вызов этих методов приводит ровно к тем же последствиям. Все это не имело бы смысла, если бы не возможность добавления новых методов для работы с инкапсулированным классом `Cursor`.

Добавьте метод `getCrime()` для извлечения данных столбцов. (Не забудьте использовать для `CrimeTable` прием импортирования, состоящий из двух шагов, как это было сделано ранее.)

Листинг 14.14. Добавление метода getCrime() (CrimeCursorWrapper.java)

```
public class CrimeCursorWrapper extends CursorWrapper {
    public CrimeCursorWrapper(Cursor cursor) {
        super(cursor);
    }

    public Crime getCrime() {
        String uuidString = getString(getColumnIndex(CrimeTable.Cols.UUID));
        String title = getString(getColumnIndex(CrimeTable.Cols.TITLE));
        long date = getLong(getColumnIndex(CrimeTable.Cols.DATE));
        int isSolved = getInt(getColumnIndex(CrimeTable.Cols.SOLVED));

        return null;
    }
}
}
```

Метод должен возвращать объект `Crime` с соответствующим значением `UUID`. Добавьте в `Crime` конструктор для выполнения этой операции.

Листинг 14.15. Добавление конструктора Crime (Crime.java)

```
public Crime() {
    this(UUID.randomUUID());
    mId = UUID.randomUUID();
    mDate = new Date();
}

public Crime(UUID id) {
    mId = id;
    mDate = new Date();
}
```

А затем доработайте метод `getCrime()`.

Листинг 14.16. Окончательная версия getCrime() (CrimeCursorWrapper.java)

```
public Crime getCrime() {
    String uuidString = getString(getColumnIndex(CrimeTable.Cols.UUID));
    String title = getString(getColumnIndex(CrimeTable.Cols.TITLE));
    long date = getLong(getColumnIndex(CrimeTable.Cols.DATE));
    int isSolved = getInt(getColumnIndex(CrimeTable.Cols.SOLVED));

    Crime crime = new Crime(UUID.fromString(uuidString));
    crime.setTitle(title);
    crime.setDate(new Date(date));
    crime.setSolved(isSolved != 0);

    return crime;
    return null;
}
```

(Android Studio предлагает выбрать между `java.util.Date` и `java.sql.Date`. И хотя вы работаете с базой данных, здесь следует выбрать `java.util.Date`.)

Преобразование в объекты модели

С `CrimeCursorWrapper` процесс получения `List<Crime>` от `CrimeLab` достаточно прямолинеен. Курсор, полученный при запросе, упаковывается в `CrimeCursorWrapper`, после чего его содержимое перебирается методом `getCrime()` для получения объектов `Crime`.

Для первой части переведите `queryCrimes(…)` на использование `CrimeCursorWrapper`.

Листинг 14.17. Использование CursorWrapper (CrimeLab.java)

```
~~private Cursor queryCrimes(String whereClause, String[] whereArgs) {~~
private CrimeCursorWrapper queryCrimes(String whereClause, String[] whereArgs)
{
    Cursor cursor = mDatabase.query(
            CrimeTable.NAME,
            null, // columns - с null выбираются все столбцы
            whereClause,
            whereArgs,
            null, // groupBy
            null, // having
            null  // orderBy
    );

    ~~return cursor;~~
    return new CrimeCursorWrapper(cursor);
}
```

Метод `getCrimes()` принял нужную форму. Добавьте код запроса всех преступлений, организуйте перебор курсора и заполнение списка `Crime`.

Листинг 14.18. Возвращение списка преступлений (CrimeLab.java)

```
public List<Crime> getCrimes() {
   ~~return new ArrayList<>();~~
    List<Crime> crimes = new ArrayList<>();

    CrimeCursorWrapper cursor = queryCrimes(null, null);

    try {
        cursor.moveToFirst();
        while (!cursor.isAfterLast()) {
            crimes.add(cursor.getCrime());
            cursor.moveToNext();
        }
    } finally {
        cursor.close();
    }

    return crimes;
}
```

Курсоры базы данных всегда устанавливаются в определенную позицию в результатах запроса. Таким образом, чтобы извлечь данные из курсора, его следует перевести к первому элементу вызовом `moveToFirst()`, а затем прочитать данные строки. Каждый раз, когда потребуется перейти к следующей записи, мы вызываем `moveToNext()`, пока `isAfterLast()` наконец не сообщит, что указатель вышел за пределы набора данных.

Последнее, что осталось сделать, — вызвать `close()` для объекта `Cursor`. Не забывайте об этой служебной операции, это важно. Если вы забудете закрыть курсор, устройство Android начнет выдавать в журнал сообщения об ошибках. Что еще хуже, если ваша забывчивость будет проявляться хронически, со временем это приведет к исчерпанию файловых дескрипторов и сбою приложения. Итак, помните: курсоры нужно закрывать.

Метод `CrimeLab.getCrime(UUID)` похож на `getCrimes()`, не считая того, что он должен извлечь только первый элемент данных (если тот присутствует).

Листинг 14.19. Переработка getCrime(UUID) (CrimeLab.java)

```
public Crime getCrime(UUID id) {
    return null;
    CrimeCursorWrapper cursor = queryCrimes(
            CrimeTable.Cols.UUID + " = ?",
            new String[] { id.toString() }
    );

    try {
        if (cursor.getCount() == 0) {
            return null;
        }

        cursor.moveToFirst();
        return cursor.getCrime();
    } finally {
        cursor.close();
    }
}
```

Нам удалось сделать следующее:

- Вы можете вставлять новые преступления, так что код, добавляющий `Crime` в `CrimeLab` при нажатии элемента действия New Crime, теперь работает.
- Приложение обращается с запросами к базе данных, так что `CrimePagerActivity` видит все объекты `Crime` в `CrimeLab`.
- Метод `CrimeLab.getCrime(UUID)` тоже работает, так что каждый экземпляр `CrimeFragment`, отображаемый в `CrimePagerActivity`, отображает существующий объект `Crime`.

Теперь при нажатии New Crime в CrimePagerActivity появляется новый объект `Crime`. Запустите CriminalIntent и убедитесь в том, что эта функциональность работает. Если что-то пошло не так, перепроверьте свои реализации из этой главы.

Обновление данных модели

Впрочем, работа еще не закончена. Преступления сохраняются в базе данных, но данные не читаются из нее. Таким образом, если вы нажмете кнопку Back в процессе редактирования нового преступления, оно не появится в `CrimeListActivity`.

Дело в том, что `CrimeLab` теперь работает немного иначе. Прежде существовал только один список `List<Crime>` и один объект для каждого преступления: тот, что содержится в `List<Crime>`. Это объяснялось тем, что переменная `mCrimes` была единственным авторитетным источником данных о преступлениях, известных вашему приложению.

Сейчас ситуация изменилась. Переменная `mCrimes` исчезла, так что список `List<Crime>`, возвращаемый `getCrimes()`, представляет собой «моментальный снимок» данных `Crime` на некоторый момент времени. Чтобы обновить `CrimeListActivity`, необходимо обновить этот снимок.

Большинство необходимых составляющих уже готово. `CrimeListActivity` уже вызывает `updateUI()` для обновления других частей интерфейса. Остается лишь заставить этот метод обновить его представление `CrimeLab`.

Сначала добавьте в `CrimeAdapter` метод `setCrimes(List<Crime>)`, чтобы закрепить отображаемые в нем данные.

Листинг 14.20. Добавление setCrimes(List<Crime>) (CrimeListFragment.java)

```
private class CrimeAdapter extends RecyclerView.Adapter<CrimeHolder> {
    ...
    @Override
    public int getItemCount() {
        return mCrimes.size();
    }

    public void setCrimes(List<Crime> crimes) {
        mCrimes = crimes;
    }
}
```

Вызовите `setCrimes(List<Crime>)` в `updateUI()`.

Листинг 14.21. Вызов setCrime(List<>) (CrimeListFragment.java)

```
private void updateUI() {
    CrimeLab crimeLab = CrimeLab.get(getActivity());
    List<Crime> crimes = crimeLab.getCrimes();

    if (mAdapter == null) {
        mAdapter = new CrimeAdapter(crimes);
        mCrimeRecyclerView.setAdapter(mAdapter);
    } else {
        mAdapter.setCrimes(crimes);
        mAdapter.notifyDataSetChanged();
    }

    updateSubtitle();
}
```

Теперь все должно работать правильно. Запустите CriminalIntent и убедитесь в том, что вы можете добавить преступление, нажать кнопку Back, и это преступление появится в `CrimeListActivity`.

Заодно можно проверить, что вызовы `updateCrime(Crime)` в `CrimeFragment` работают. Нажмите на преступлении и отредактируйте его краткое описание в `CrimePagerActivity`. Нажмите кнопку Back и убедитесь в том, что новый текст появился в списке.

Для любознательных: другие базы данных

Ради простоты и краткости мы не стали углубляться во все подробности, которые могут встретиться при работе с базами данных в профессиональном приложении. Разработчики не зря пользуются такими инструментами, как ORM: все это может оказаться весьма непростым делом.

В более серьезном приложении в базах данных обычно реализуются следующие аспекты:

- *Типы данных столбцов*. Строго говоря, в SQLite типизация столбцов отсутствует, так что вы можете обойтись без этой информации. Впрочем, дать SQLite полезную подсказку не помешает.
- *Индексы*. Запросы к столбцам, для которых построены индексы, выполняются намного быстрее, чем запросы к неиндексированным столбцам.
- *Внешние ключи*. В нашем примере база данных содержит всего одну таблицу, но в реальных приложениях могут понадобиться ограничения внешнего ключа.

Также стоит учитывать и более глубокие факторы эффективности. Ваше приложение при каждом запросе к базе данных создает новый список объектов `Crime`. Действительно эффективное приложение оптимизирует такие обращения: оно использует заново старые экземпляры `Crime` или организует из них хранилище объектов в памяти (как это делалось в предшествующих главах). Это приводит к увеличению объема кода — еще одной проблеме, которую часто пытаются решить инструменты ORM.

Для любознательных: контекст приложения

Ранее в этой главе при вызове конструктора `CrimeLab` использовался *контекст приложения*:

```
private CrimeLab(Context context) {
  mContext = context.getApplicationContext();
  ...
}
```

Почему именно контекст приложения? Когда следует использовать его вместо активности в качестве контекста?

Важно учитывать жизненный цикл каждого из этих объектов. Если в приложении существуют какие-либо активности, Android также создает объект *приложения*. Активности появляются и исчезают в процессе работы пользователя с приложением, но объект приложения продолжает существовать. Его срок жизни значительно превышает срок жизни любой отдельной активности.

Объект `CrimeLab` является синглетом, то есть после того, как он будет создан, он продолжит существовать до тех пор, пока не будет уничтожен весь процесс приложения. В `CrimeLab` хранится ссылка на объект `mContext`. Если сохранить в `mContext` активность, эта активность никогда не будет уничтожена уборщиком мусора, потому что ссылка на нее хранится в `CrimeLab`. Даже если пользователь покинет активность, она не уничтожается.

Чтобы избежать столь неэффективного поведения, мы используем контекст приложения: активности приходят и уходят, а `CrimeLab` поддерживает ссылку на объект `Context`. Всегда учитывайте срок жизни своих активностей, если вы сохраняете ссылки на них.

Упражнение. Удаление преступлений

Если ранее вы добавили элемент действия Delete Crime, в этом упражнении он дополняется возможностью удаления информации из базы данных. Для этого вызывается метод `deleteCrime(Crime)` для `CrimeLab`, который вызывает метод `mDatabase.delete(…)` для завершения работы.

А если элемент действия отсутствует? Так добавьте его! Добавьте на панель инструментов `CrimeFragment` элемент действия, который вызывает метод `CrimeLab.deleteCrime(Crime)` с последующим вызовом `finish()` для его активности.

15 Неявные интенты

В Android можно запустить активность из другого приложения на устройстве при помощи *неявного интента* (implicit intent). В явном интенте задается класс запускаемой активности, а ОС запускает его. В неявном интенте вы описываете операцию, которую необходимо выполнить, а ОС запускает активность соответствующего приложения.

В приложении CriminalIntent мы будем использовать неявные интенты для выбора подозреваемых из списка контактов пользователя и отправки текстовых отчетов о преступлении. Пользователь выбирает подозреваемого в контактном приложении, установленном на устройстве, и получает список приложений для отправки отчета (рис. 15.1).

Использовать функциональность других приложений при помощи неявных интентов намного проще, чем писать собственные реализации стандартных задач.

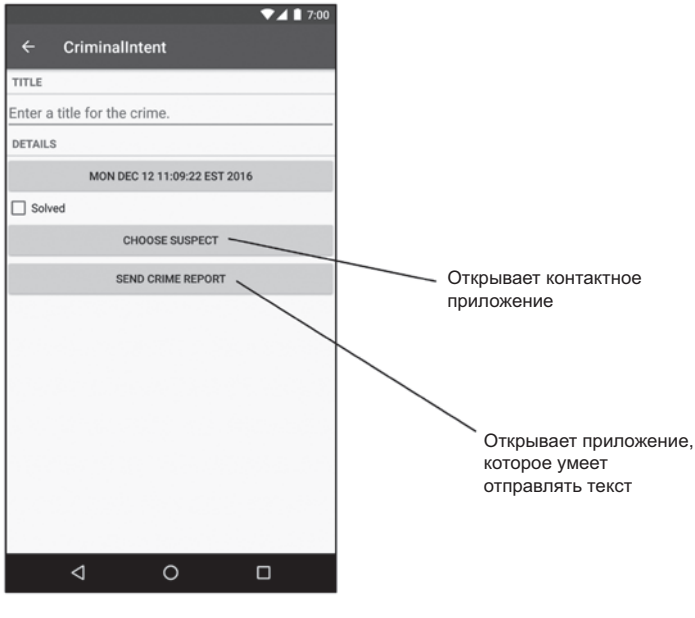


Рис. 15.1. Открытие приложений для выбора контактов и отправки отчетов

Пользователям также нравится работать с приложениями, которые им уже хорошо знакомы, в сочетании с вашим приложением.

Прежде чем создавать неявные интенты, необходимо выполнить с CriminalIntent ряд подготовительных действий:

- добавить в макеты `CrimeFragment` кнопки выбора подозреваемого и отправки отчета;
- добавить в класс `Crime` поле `mSuspect`, в котором будет храниться имя подозреваемого;
- создать отчет о преступлении с использованием *форматных строк ресурсов*.

Добавление кнопок

Начнем с включения в макеты `CrimeFragment` новых кнопок. Прежде всего добавьте строки, которые будут отображаться на кнопках.

Листинг 15.1. Добавление строк для надписей на кнопках (strings.xml)

```
    <string name="subtitle_format">%1$d crimes</string>
    <string name="crime_suspect_text">Choose Suspect</string>
    <string name="crime_report_text">Send Crime Report</string>
</resources>
```

Добавьте в файл layout/fragment_crime.xml два виджета `Button`, представленных на рис. 15.2. Обратите внимание: на диаграмме не показан первый виджет `LinearLayout` и все его потомки, чтобы вы могли сосредоточиться на новых и интересных частях диаграммы.

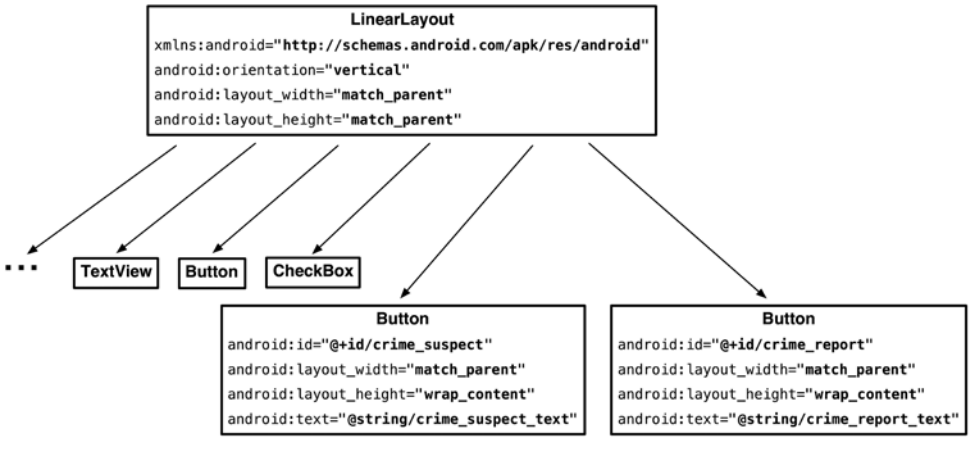


Рис. 15.2. Добавление кнопок для выбора контактов и отправки отчетов (layout/fragment_crime.xml)

На этой стадии вы можете проверить макеты в области предварительного просмотра или запустить приложение CriminalIntent, чтобы убедиться в правильности расположения новых кнопок.

Добавление подозреваемого в уровень модели

Откройте файл Crime.java и добавьте новую переменную для хранения имени подозреваемого.

Листинг 15.2. Добавление поля для имени подозреваемого (Crime.java)

```
public class Crime {
    ...
    private boolean mSolved;
    private String mSuspect;

    public Crime() {
        this(UUID.randomUUID());
    }
    ...
    public void setSolved(boolean solved) {
        mSolved = solved;
    }

    public String getSuspect() {
        return mSuspect;
    }
    public void setSuspect(String suspect) {
        mSuspect = suspect;
    }
}
```

Затем дополнительное поле добавляется в базу данных. Сначала добавьте в `CrimeDbSchema` столбец `suspect`.

Листинг 15.3. Добавление столбца suspect (CrimeDbSchema.java)

```
public class CrimeDbSchema {
    public static final class CrimeTable {
        public static final String NAME = "crimes";

        public static final class Cols {
            public static final String UUID = "uuid";
            public static final String TITLE = "title";
            public static final String DATE = "date";
            public static final String SOLVED = "solved";
            public static final String SUSPECT = "suspect";
        }
    }
}
```

Также добавьте столбец в `CrimeBaseHelper`. (Обратите внимание: новый код начинается с запятой после `CrimeTable.Cols.SOLVED`.)

Листинг 15.4. Добавление столбца suspect в другом месте (CrimeBaseHelper.java)

```
@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL("create table " + CrimeTable.NAME + "(" +
            " _id integer primary key autoincrement, " +
            CrimeTable.Cols.UUID + ", " +
            CrimeTable.Cols.TITLE + ", " +
            CrimeTable.Cols.DATE + ", " +
            CrimeTable.Cols.SOLVED + ", " +
            CrimeTable.Cols.SUSPECT +
            ")"
    );
}
```

Запишите новый столбец в `CrimeLab.getContentValues(Crime)`.

Листинг 15.5. Запись в столбец suspect (CrimeLab.java)

```
    private static ContentValues getContentValues(Crime crime) {
        ContentValues values = new ContentValues();
        values.put(CrimeTable.Cols.UUID, crime.getId().toString());
        values.put(CrimeTable.Cols.TITLE, crime.getTitle());
        values.put(CrimeTable.Cols.DATE, crime.getDate().getTime());
        values.put(CrimeTable.Cols.SOLVED, crime.isSolved() ? 1 : 0);
        values.put(CrimeTable.Cols.SUSPECT, crime.getSuspect());

        return values;
    }
}
```

Теперь прочитайте из него данные в `CrimeCursorWrapper`.

Листинг 15.6. Чтение из столбца suspect (CrimeCursorWrapper.java)

```
    public Crime getCrime() {
        String uuidString = getString(getColumnIndex(CrimeTable.Cols.UUID));
        String title = getString(getColumnIndex(CrimeTable.Cols.TITLE));
        long date = getLong(getColumnIndex(CrimeTable.Cols.DATE));
        int isSolved = getInt(getColumnIndex(CrimeTable.Cols.SOLVED));
        String suspect = getString(getColumnIndex(CrimeTable.Cols.SUSPECT));

        Crime crime = new Crime(UUID.fromString(uuidString));
        crime.setTitle(title);
        crime.setDate(new Date(date));
        crime.setSolved(isSolved != 0);
        crime.setSuspect(suspect);

        return crime;
    }
}
```

Если приложение CriminalIntent уже установлено на вашем устройстве, существующая база данных не содержит столбец `suspect`, и новый метод `onCreate(SQLiteDatabase)` не будет выполнен для добавления нового столбца. В такой ситуации проще всего стереть старую базу данных для создания новой. (Это довольно часто происходит при разработке приложений.)

Сначала удалите приложение CriminalIntent: откройте экран лаунчера и перетащите значок CriminalIntent в верхнюю часть экрана. В процессе удаления будут уничтожены все данные приложения в песочнице вместе с устаревшей схемой базы данных. Затем запустите CriminalIntent из Android Studio. Новая база данных создается вместе с новым столбцом в процессе установки приложения.

Форматные строки

Последним подготовительным шагом станет создание шаблона отчета о преступлении, который заполняется информацией о конкретном преступлении. Так как подробная информация недоступна до стадии выполнения, необходимо использовать форматную строку с заполнителями, которые будут заменяться во время выполнения. Форматная строка будет выглядеть так:

```
<string name="crime_report">%1$s! The crime was discovered on %2$s. %3$s, and %4$s
```

Поля `%1$s`, `%2$s` и т.д. — заполнители для строковых аргументов. В коде вы вызываете `getString(…)` и передаете форматную строку и еще четыре строки в том порядке, в каком они должны заменять заполнители.

Сначала добавьте в strings.xml строки из листинга 15.7.

Листинг 15.7. Добавление строковых ресурсов (strings.xml)

```
    <string name="crime_suspect_text">Choose Suspect</string>
    <string name="crime_report_text">Send Crime Report</string>
    <string name="crime_report">%1$s!
      The crime was discovered on %2$s. %3$s, and %4$s
    </string>
    <string name="crime_report_solved">The case is solved</string>
    <string name="crime_report_unsolved">The case is not solved</string>
    <string name="crime_report_no_suspect">there is no suspect.</string>
    <string name="crime_report_suspect">the suspect is %s.</string>
    <string name="crime_report_subject">CriminalIntent Crime Report</string>
    <string name="send_report">Send crime report via</string>
</resources>
```

В файле CrimeFragment.java добавьте метод, который создает четыре строки, соединяет их и возвращает полный отчет.

Листинг 15.8. Добавление метода getCrimeReport() (CrimeFragment.java)

```
    private void updateDate() {
        mDateButton.setText(mCrime.getDate().toString());
```

```
    }

    private String getCrimeReport() {
        String solvedString = null;
        if (mCrime.isSolved()) {
            solvedString = getString(R.string.crime_report_solved);
        } else {
            solvedString = getString(R.string.crime_report_unsolved);
        }

        String dateFormat = "EEE, MMM dd";
        String dateString = DateFormat.format(dateFormat,
                                              mCrime.getDate()).toString();

        String suspect = mCrime.getSuspect();
        if (suspect == null) {
            suspect = getString(R.string.crime_report_no_suspect);
        } else {
            suspect = getString(R.string.crime_report_suspect, suspect);
        }

        String report = getString(R.string.crime_report,
            mCrime.getTitle(), dateString, solvedString, suspect);
        return report;
    }
}
```

(Обратите внимание: класс `DateFormat` существует в двух версиях, `android.text.format.DateFormat` и `java.text.DateFormat`. Используйте `android.text.format.DateFormat`.)

Приготовления завершены, теперь можно непосредственно заняться неявными интентами.

Использование неявных интентов

Объект `Intent` описывает для ОС некую операцию, которую вы хотите выполнить. Для *явных* интентов, использовавшихся до настоящего момента, разработчик явно указывает активность, которую должна запустить ОС:

```
Intent intent = new Intent(getActivity(), CrimePagerActivity.class);
intent.putExtra(EXTRA_CRIME_ID, crimeId);
startActivity(intent);
```

Для *неявных* интентов разработчик описывает выполняемую операцию, а ОС запускает активность, которая ранее сообщила о том, что она способна выполнять эту операцию. Если ОС находит несколько таких активностей, пользователю предлагается выбрать нужную.

Строение неявного интента

Ниже перечислены важнейшие составляющие интента, используемые для определения выполняемой операции.

- Выполняемое *действие* (action) — обычно определяется константами из класса `Intent`. Так, для просмотра URL-адреса используется константа `Intent.ACTION_VIEW`, а для отправки данных — константа `Intent.ACTION_SEND`.
- Местонахождение *данных* — это может быть как ссылка на данные, находящиеся за пределами устройства (скажем, URL веб-страницы), так и URI файла или URI контента, ссылающийся на запись `ContentProvider`.
- *Тип* данных, с которыми работает действие, — тип MIME (например, `text/html` или `audio/mpeg3`). Если в интент включено местонахождение данных, то тип обычно удается определить по этим данным.
- Необязательные *категории* — если действие указывает, что нужно сделать, категория обычно описывает, где, когда или как вы пытаетесь использовать операцию. Android использует категорию `android.intent.category.LAUNCHER` для обозначения активностей, которые должны отображаться в лаунчере приложений верхнего уровня. С другой стороны, категория `android.intent.category.INFO` обозначает активность, которая выдает пользователю информацию о пакете, но не отображается в лаунчере.

Простой неявный интент для просмотра веб-сайта включает действие `Intent.ACTION_VIEW` и объект данных `Uri` с URL-адресом сайта.

На основании этой информации ОС запускает соответствующую активность соответствующего приложения. (Если ОС обнаруживает более одного кандидата, пользователю предлагается принять решение.)

Активность сообщает о себе как об исполнителе для `ACTION_VIEW` при помощи фильтра интентов в манифесте. Например, если вы пишете приложение-браузер, вы включаете следующий фильтр интентов в объявление активности, реагирующей на `ACTION_VIEW`:

```
<activity
    android:name=".BrowserActivity"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:scheme="http" android:host="www.bignerdranch.com" />
    </intent-filter>
</activity>
```

Категория `DEFAULT` должна явно задаваться в фильтрах интентов. Элемент `action` в фильтре интентов сообщает ОС, что активность способна выполнять операцию, а категория `DEFAULT` — что она желает рассматриваться среди кандидатов на выполнение операции. Категория `DEFAULT` неявно добавляется к почти любому неявному интенту. (Единственное исключение составляет категория `LAUNCHER`, с которой мы будем работать в главе 24.)

Неявные интенты, как и явные, также могут включать дополнения. Однако дополнения неявного интента не используются ОС для поиска соответствующей активности.

Также следует отметить, что компоненты действия и данных интента могут использоваться в сочетании с явными интентами. Результат равнозначен тому, как если бы вы приказали конкретной активности выполнить конкретную операцию.

Отправка отчета

Чтобы увидеть на практике, как работает эта схема, мы создадим неявный интент для отправки отчета о преступлении в приложении CriminalIntent. Операция, которую нужно выполнить, — отправка простого текста; отчет представляет собой строку. Таким образом, действие неявного интента будет представлено константой `ACTION_SEND`. Интент не содержит ссылок на данные и не имеет категорий, но определяет тип `text/plain`.

В методе `CrimeFragment.onCreateView(…)` получите ссылку на кнопку SEND CRIME REPORT и назначьте для нее слушателя. В реализации слушателя создайте неявный интент и передайте его `startActivity(Intent)`.

Листинг 15.9. Отправка отчета о преступлении (CrimeFragment.java)

```
private Crime mCrime;
private EditText mTitleField;
private Button mDateButton;
private CheckBox mSolvedCheckbox;
private Button mReportButton;
...
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    ...

    mReportButton = (Button) v.findViewById(R.id.crime_report);
    mReportButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            Intent i = new Intent(Intent.ACTION_SEND);
            i.setType("text/plain");
            i.putExtra(Intent.EXTRA_TEXT, getCrimeReport());
            i.putExtra(Intent.EXTRA_SUBJECT,
                    getString(R.string.crime_report_subject));
            startActivity(i);
        }
    });

    return v;
}
```

Здесь мы используем конструктор `Intent`, который получает строку с константой, описывающей действие. Также существуют другие конструкторы, которые могут использоваться в зависимости от вида создаваемого неявного интента. Информацию о них можно найти в справочной документации `Intent`. Конструктора, получающего тип, не существует, поэтому мы задаем его явно.

Текст отчета и строка темы включаются в дополнения. Обратите внимание на использование в них констант, определенных в классе `Intent`. Любая активность, реагирующая на интент, знает эти константы и то, что следует делать с ассоциированными значениями.

Запустите приложение CriminalIntent и нажмите кнопку SEND CRIME REPORT. Так как этот интент с большой вероятностью совпадет со многими активностями на устройстве, скорее всего, на экране появится список активностей (рис. 15.3).

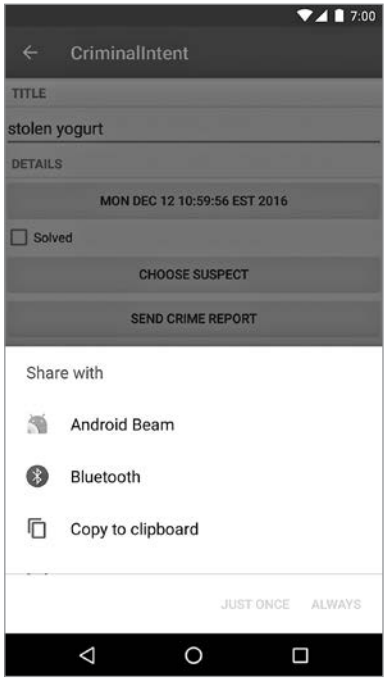


Рис. 15.3. Приложения, готовые отправить ваш отчет

Если на экране появился список, выберите нужный вариант. Вы увидите, что отчет о преступлении загружается в выбранном вами приложении. Вам остается лишь ввести адрес и отправить его.

Если список не появился, это может означать одно из двух: либо вы уже назначили приложение по умолчанию для идентичного неявного интента, либо на вашем устройстве имеется всего одна активность, способная реагировать на этот интент.

Часто лучшим вариантом оказывается использование приложения по умолчанию, выбранного пользователем для действия `ACTION_SEND`. Впрочем, в приложении CriminalIntent лучше всегда предоставлять пользователю выбор: сегодня пользователь предпочтет не поднимать шум и отправит отчет по электронной почте, а завтра выставит нарушителя на общественное осуждение в Твиттере.

Вы можете создать список, который будет отображаться каждый раз при использовании неявного интента для запуска активности. После создания неявного ин-

тента способом, показанным ранее, вы вызываете следующий метод `Intent` и передаете ему неявный интент и строку с заголовком:

```
public static Intent createChooser(Intent target, String title)
```

Затем интент, возвращенный `createChooser(…)`, передается `startActivity(…)`.

В файле CrimeFragment.java создайте список выбора для отображения активностей, реагирующих на неявный интент.

Листинг 15.10. Использование списка выбора (CrimeFragment.java)

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    ...
    mReportButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            Intent i = new Intent(Intent.ACTION_SEND);
            i.setType("text/plain");
            i.putExtra(Intent.EXTRA_TEXT, getCrimeReport());
            i.putExtra(Intent.EXTRA_SUBJECT,
                    getString(R.string.crime_report_subject));
        i = Intent.createChooser(i, getString(R.string.send_report));
        startActivity(i);
        }
```

Запустите приложение CriminalIntent и нажмите кнопку SEND CRIME REPORT. Если в системе имеется несколько активностей, способных обработать ваш интент, на экране появляется список для выбора (рис. 15.4).

Запрос контакта у Android

Теперь мы создадим другой неявный интент, который предлагает пользователю выбрать подозреваемого из списка контактов. Для этого неявного интента будет определено действие и местонахождение соответствующих данных. Действие задается константой `Intent.ACTION_PICK`, а местонахождение данных — `ContactsContract.Contacts.CONTENT_URI`. Короче говоря, вы просите Android помочь с выбором записи из базы данных контактов.

Запущенная активность должна вернуть результат, поэтому мы передаем интент через `startActivityForResult(…)` вместе с кодом запроса. Добавьте в файл CrimeFragment.java константу для кода запроса и поле для кнопки.

Листинг 15.11. Добавление поля для кнопки подозреваемого (CrimeFragment.java)

```
private static final int REQUEST_DATE = 0;
private static final int REQUEST_CONTACT = 1;
...
private Button mSuspectButton;
private Button mReportButton;
```

В конце `onCreateView(…)` получите ссылку на кнопку и назначьте ей слушателя. В реализации слушателя создайте неявный интент и передайте его

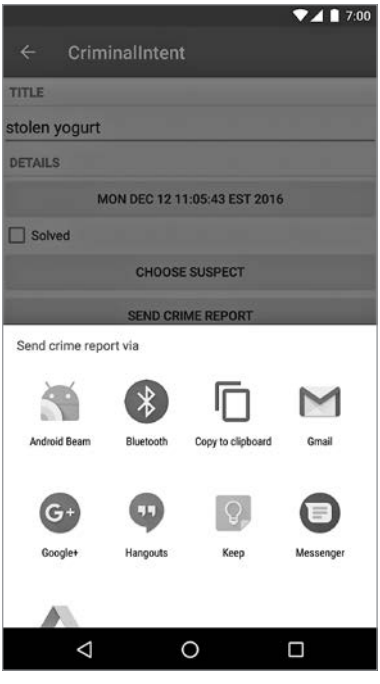


Рис. 15.4. Отправка текста с выбором активности

`startActivityForResult(…)`. Также выведите на кнопке имя подозреваемого (если оно содержится в `Crime`).

Листинг 15.12. Отправка неявного интента (CrimeFragment.java)

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    ...
    final Intent pickContact = new Intent(Intent.ACTION_PICK,
            ContactsContract.Contacts.CONTENT_URI);
    mSuspectButton = (Button) v.findViewById(R.id.crime_suspect);
    mSuspectButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            startActivityForResult(pickContact, REQUEST_CONTACT);
        }
    });

    if (mCrime.getSuspect() != null) {
        mSuspectButton.setText(mCrime.getSuspect());
    }

    return v;
}
```

Мы еще воспользуемся интентом `pickContact`, поэтому он размещается за пределами слушателя `OnClickListener` кнопки `mSuspectButton`.

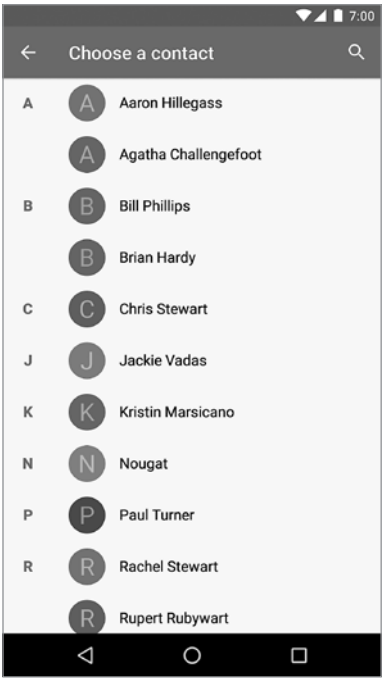


Рис. 15.5. Список подозреваемых

Запустите приложение CriminalIntent и нажмите кнопку CHOOSE SUSPECT. На экране появляется список контактов (рис. 15.5).

Если у вас установлено другое контактное приложение, экран будет выглядеть иначе. Это еще одно преимущество неявных интентов: вам не нужно знать название контактного приложения, чтобы использовать его из своего приложения. Соответственно, пользователь может установить то приложение, которое считает нужным, а ОС найдет и запустит его.

Получение данных из списка контактов

Теперь необходимо получить результат от контактного приложения. Контактная информация совместно используется многими приложениями, поэтому Android предоставляет расширенный API для работы с контактными данными через `ContentProvider`. Экземпляры этого класса инкапсулируют базы данных и предоставляют доступ к ним другим приложениям. Обращение к `ContentProvider` осуществляется через `ContentResolver`.

Так как активность запускалась с возвращением результата с использованием `ACTION_PICK`, вы можете получить интент вызовом `onActivityResult(…)`. Интент включает URI данных — ссылку на конкретный контакт, выбранный пользователем.

В файле CrimeFragment.java добавьте следующий код в реализацию `onActivityResult(…)` из `CrimeFragment`.

Листинг 15.13. Получение имени контакта (CrimeFragment.java)

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode != Activity.RESULT_OK) {
        return;
    }

    if (requestCode == REQUEST_DATE) {
        ...
        updateDate();

    } else if (requestCode == REQUEST_CONTACT && data != null) {
        Uri contactUri = data.getData();
        // Определение полей, значения которых должны быть
        // возвращены запросом.
        String[] queryFields = new String[] {
                ContactsContract.Contacts.DISPLAY_NAME
        };
        // Выполнение запроса - contactUri здесь выполняет функции
        // условия "where"
        Cursor c = getActivity().getContentResolver()
                .query(contactUri, queryFields, null, null, null);
        try {
            // Проверка получения результатов
            if (c.getCount() == 0) {
                return;
            }
            // Извлечение первого столбца данных - имени подозреваемого.
            c.moveToFirst();
            String suspect = c.getString(0);
            mCrime.setSuspect(suspect);
            mSuspectButton.setText(suspect);
        } finally {
            c.close();
        }
    }
}
```

В листинге 15.13 создается запрос на все отображаемые имена контактов из возвращаемых данных. Затем мы выдаем запрос к базе данных контактов и получаем объект `Cursor` для работы с ней. Так как мы знаем, что курсор содержит всего один элемент, мы переходим к первому элементу и используем его как строку. Эта строка содержит имя подозреваемого, которое мы используем для задания подозреваемого в `Crime` и текста кнопки CHOOSE SUSPECT.

(База данных контактов сама по себе является достаточно обширной темой. Здесь она не рассматривается. Если вам захочется узнать больше, обратитесь к руководству по Contacts Provider API: *developer.android.com/guide/topics/providers/contacts-provider.html*.)

Запустите приложение. На некоторых устройствах нет контактного приложения, которое могло бы использоваться для тестирования. В этом случае воспользуйтесь эмулятором.

Разрешения контактов

Как получить разрешение на чтение из базы данных контактов? Контактное приложение распространяет свои разрешения на вас. Оно обладает полными разрешениями на обращение к базе данных. Когда контактное приложение возвращает родительской активности URI данных в интенте, оно также добавляет флаг `Intent.FLAG_GRANT_READ_URI_PERMISSION`. Этот флаг сообщает Android, что родительской активности в CriminalIntent следует разрешить однократное использование этих данных. Такой подход работает хорошо, потому что фактически нам нужен доступ не ко всей базе данных контактов, а к одному контакту в этой базе.

Проверка реагирующих активностей

На первый неявный интент, созданный в этой главе, кто-то гарантированно отреагирует: даже если способа отправки отчета не существует, окно выбора все равно будет отображено. Со вторым интентом дело обстоит иначе: на некоторых устройствах (или у некоторых пользователей) может не оказаться контактного приложения. Если ОС не найдет подходящую активность, в приложении происходит сбой.

Проблема решается предварительной проверкой того, от какой части ОС поступил вызов `PackageManager`. Это удобно сделать в `onCreateView(…)`.

Листинг 15.14. Защита от отсутствия контактных приложений (CrimeFragment.java)

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    ...
    if (mCrime.getSuspect() != null) {
        mSuspectButton.setText(mCrime.getSuspect());
    }

    PackageManager packageManager = getActivity().getPackageManager();
    if (packageManager.resolveActivity(pickContact,
            PackageManager.MATCH_DEFAULT_ONLY) == null) {
        mSuspectButton.setEnabled(false);
    }

    return v;
}
```

`PackageManager` известно все о компонентах, установленных на устройстве Android, включая все его активности. (Другие компоненты встретятся вам позднее в этой книге.) Вызывая `resolveActivity(Intent, int)`, вы приказываете найти активность, соответствующую переданному интенту. Флаг `MATCH_DEFAULT_ONLY` ограничивает поиск активностями с флагом `CATEGORY_DEFAULT` (по аналогии с `startActivity(Intent)`).

Если поиск прошел успешно, возвращается экземпляр `ResolveInfo`, который сообщает полную информацию о найденной активности. С другой стороны, если поиск вернул `null`, все кончено — контактного приложения нет, поэтому бесполезная кнопка просто блокируется.

Если вы хотите убедиться в том, что фильтр работает, но не располагаете устройством без контактного приложения, временно добавьте в интент дополнительную

категорию. Эта категория ничего не делает, а только предотвращает возможные совпадения контактных приложений с вашим интентом.

Листинг 15.15. Фиктивный код для проверки фильтра (CrimeFragment.java)

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    ...
    final Intent pickContact = new Intent(Intent.ACTION_PICK,
            ContactsContract.Contacts.CONTENT_URI);
    pickContact.addCategory(Intent.CATEGORY_HOME);
    mSuspectButton = (Button) v.findViewById(R.id.crime_suspect);
    mSuspectButton.setOnClickListener(new View.OnClickListener() {
        ...
    });
```

На этот раз кнопка выбора подозреваемого недоступна (рис. 15.6).

Рис. 15.6. Заблокированная кнопка выбора подозреваемого

После завершения проверки удалите фиктивный код.

Листинг 15.16. Удаление фиктивного кода (CrimeFragment.java)

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    ...
    final Intent pickContact = new Intent(Intent.ACTION_PICK,
            ContactsContract.Contacts.CONTENT_URI);
```

```
    pickContact.addCategory(Intent.CATEGORY_HOME);
    mSuspectButton = (Button) v.findViewById(R.id.crime_suspect);
    mSuspectButton.setOnClickListener(new View.OnClickListener() {
        ...
    });
```

Упражнение. ShareCompat

Первое упражнение очень простое. В библиотеку поддержки Android входит класс `ShareCompat` с внутренним классом `IntentBuilder`. `ShareCompat.IntentBuilder` упрощает построение интентов точно такого вида, какой мы использовали для кнопки отчета.

Итак, первое упражнение: в слушателе `OnClickListener` кнопки `mReportButton` используйте для построения интента класс `ShareCompat.IntentBuilder` (вместо того, чтобы строить его вручную).

Упражнение. Другой неявный интент

Возможно, вместо отправки отчета разгневанный пользователь предпочтет разобраться с подозреваемым по телефону. Добавьте новую кнопку для звонка подозреваемому.

Вам понадобится извлечь номер телефона из базы данных контактов. Для этого необходимо обратиться с запросом к другой таблице базы данных `ContactsContract`, которая называется `CommonDataKinds.Phone`. За дополнительными сведениями о том, как получить эту информацию, обращайтесь к документации `ContactsContract` и `ContactsContract.CommonDataKinds.Phone`.

Пара подсказок: для запроса дополнительных данных можно воспользоваться разрешением `android.permission.READ_CONTACTS`. С этим разрешением вы сможете прочитать `ContactsContract.Contacts._ID` для получения идентификатора контакта из исходного запроса. Затем полученный идентификатор используется для получения данных из таблицы `CommonDataKinds.Phone`.

После получения телефонного номера можно создать неявный интент с URI телефона:

```
Uri number = Uri.parse("tel:5551234");
```

При этом может использоваться действие `Intent.ACTION_DIAL` или `Intent.ACTION_CALL`. `ACTION_CALL` запускает телефонное приложение и немедленно осуществляет звонок по номеру, отправленному в интенте; `ACTION_DIAL` только вводит номер и ждет, пока пользователь инициирует звонок.

Мы рекомендуем использовать `ACTION_DIAL`. Режим `ACTION_CALL` может быть ограничен, и для него определенно потребуются разрешения. Кроме того, у пользователя будет возможность немного остыть перед нажатием кнопки вызова.

16 Интенты при работе с камерой

Итак, вы научились работать с неявными интентами, и теперь преступления будут документироваться еще точнее. Располагая снимком места преступления, вы сможете поделиться жуткими подробностями со всеми желающими.

Для создания снимков вам понадобится пара новых инструментов, которые используются в сочетании с уже знакомыми вам неявными интентами. Неявный интент используется при запуске любимого приложения для работы с камерой и получения от него нового снимка.

Неявный интент может создать снимок, но где его разместить? И как вывести на экран созданную фотографию? В этой главе даны ответы на оба вопроса.

Место для хранения фотографий

Прежде всего следует обустроить место, в котором будет «жить» ваша фотография. Для этого понадобятся два новых объекта `View`: `ImageView` для отображения фотографии на экране и `Button` для создания снимка (рис. 16.1).

Если выделить отдельную строку под миниатюру и кнопку, приложение будет выглядеть убого и непрофессионально. Чтобы этого не произошло, необходимо аккуратно разместить все компоненты на экране.

Добавьте новые представления создания снимка в fragment_crime.xml, чтобы сформировать новую область. Начните с построения левой стороны: добавьте `ImageView` и `ImageButton` для изображения (рис. 16.2).

Затем создайте правую сторону: переместите заголовочный виджет `TextView` и `EditText` в нового потомка `LinearLayout` виджета `LinearLayout`, созданного на рис. 16.2 (рис. 16.3).

Запустите приложение CriminalIntent; новый пользовательский интерфейс должен выглядеть так, как показано на рис. 16.1.

Выглядит замечательно, но для реагирования на нажатия `ImageButton` и управления содержимым `ImageView` нам понадобятся переменные экземпляров со ссылками на оба виджета. Как обычно, вызовите `findViewById(int)` для заполняемого макета fragment_crime.xml, чтобы найти представления и подключить их.

Рис. 16.1. Новый интерфейс

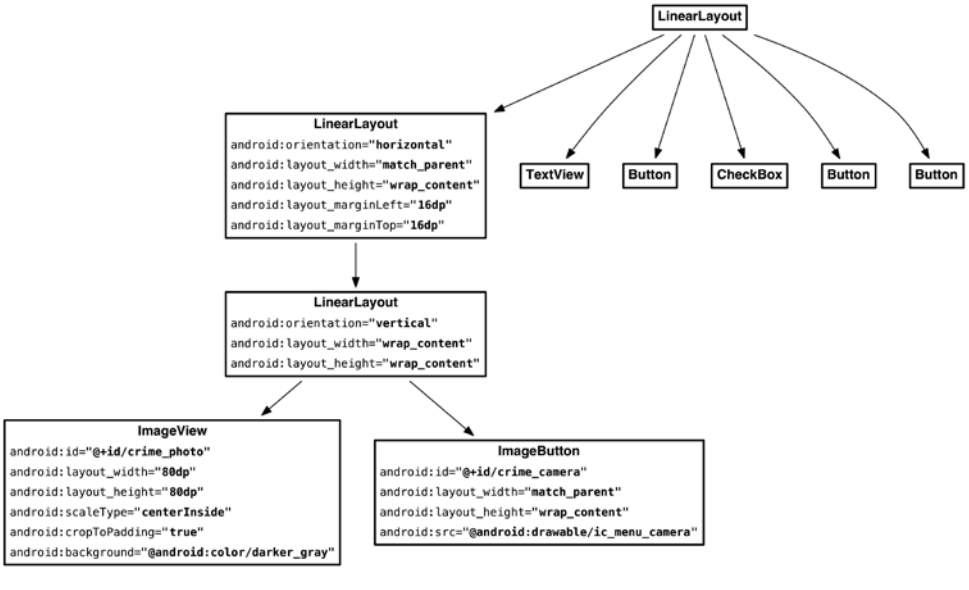


Рис. 16.2. Макет представления камеры (res/layout/fragment_crime.xml)

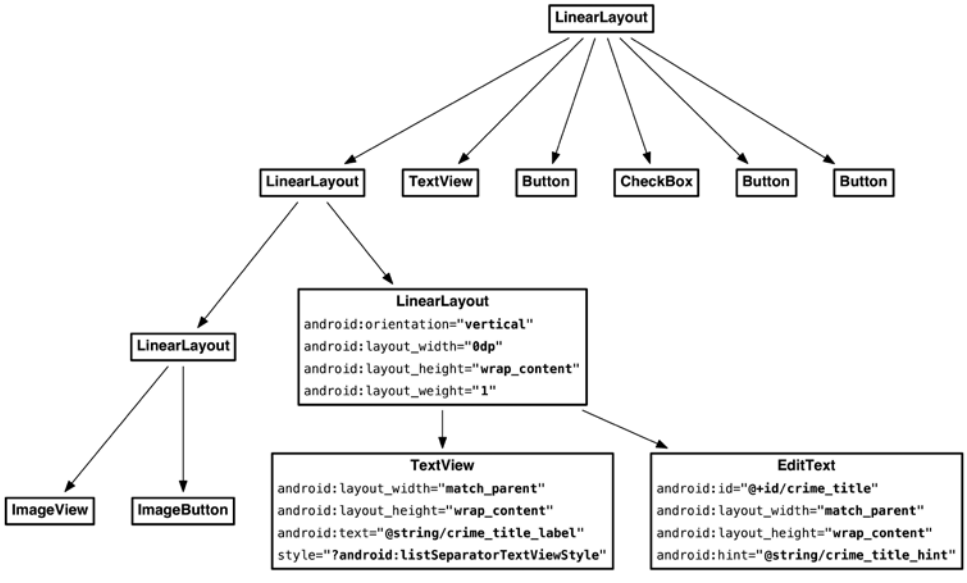


Рис. 16.3. Макет заголовка (res/layout/fragment_crime.xml)

Листинг 16.1. Добавление переменных экземпляров (CrimeFragment.java)

```
private Button mSuspectButton;
private Button mReportButton;
private ImageButton mPhotoButton;
private ImageView mPhotoView;
...

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    ...
    PackageManager packageManager = getActivity().getPackageManager();
    if (packageManager.resolveActivity(pickContact,
            PackageManager.MATCH_DEFAULT_ONLY) == null) {
        mSuspectButton.setEnabled(false);
    }

    mPhotoButton = (ImageButton) v.findViewById(R.id.crime_camera);
    mPhotoView = (ImageView) v.findViewById(R.id.crime_photo);

    return v;
}
```

На этом ненадолго оставим пользовательский интерфейс (через несколько страниц эти кнопки будут подключены к логике приложения).

Внешнее хранилище

Одного лишь места на экране вашим фотографиям недостаточно. Полноразмерная фотография слишком велика для хранения в базе данных SQLite, не говоря уже об интенте. Ей необходимо место для хранения в файловой системе устройства.

Обычно такие данные размещаются в закрытом (приватном) хранилище. Вспомните, что именно в закрытой области хранится наша база данных SQLite. Такие методы, как `Context.getFileStreamPath(String)` и `Context.getFilesDir()`, позволяют хранить в этой же области и обычные файлы (в папке по соседству с папкой `databases`, в которой размещается база данных SQLite).

Основные методы для работы с внешними файлами и каталогами в `Context`:

`File getFilesDir()`

Возвращает дескриптор каталога для закрытых файлов приложения.

`FileInputStream openFileInput(String name)`

Открывает существующий файл для ввода (относительно каталога файлов).

`FileOutputStream openFileOutput(String name, int mode)`

Открывает существующий файл для вывода, возможно, с созданием (относительно каталога файлов).

`File getDir(String name, int mode)`

Получает (и, возможно, создает) подкаталог в каталоге файлов.

`String[] fileList()`

Получает список имен файлов в главном каталоге файлов (например, для использования с `openFileInput(String)`).

`File getCacheDir()`

Возвращает дескриптор каталога, используемого для хранения кэш-файлов. Будьте внимательны, поддерживайте порядок в этом каталоге и старайтесь использовать как можно меньше пространства.

Если вы сохраняете файлы, которые должны использоваться только текущим приложением, — это именно то, что вам нужно.

С другой стороны, если запись в файлы должна осуществляться другим приложением, вам не повезло: хотя существует флаг `Context.MODE_WORLD_READABLE`, который можно передать при вызове `openFileOutput(String, int)`, он официально считается устаревшим, а на новых устройствах его надежность не гарантирована. Когда-то существовала возможность передачи файлов в общедоступное внешнее хранилище, но в последних версиях она была заблокирована по соображениям безопасности.

Если вы сохраняете файлы, которые должны использоваться другими приложениями, или получаете файлы от других приложений (как, например, сохраненные фотографии), к файлам необходимо организовать доступ через `ContentProvider`. `ContentProvider` позволяет открыть доступ к URI контента другим приложениям. Тогда эти приложения смогут загружать или записывать данные по этим URI.

В любом случае ситуация находится под вашим контролем, и вы всегда можете запретить чтение или запись по своему желанию.

Использование FileProvider

Если ваши потребности ограничиваются получением файла из другого приложения, полноценная реализация `ContentProvider` — это перебор. К счастью, Google предоставляет вспомогательный класс с именем `FileProvider`, который берет на себя выполнение всех задач, кроме настройки конфигурации.

Прежде всего следует объявить `FileProvider` как экземпляр `ContentProvider`, связанный с конкретным *хранилищем* (authority). Для этого объявление `ContentProvider` включается в AndroidManifest.xml.

Листинг 16.2. Добавление объявления FileProvider (AndroidManifest.xml)

```
<activity
    android:name=".CrimePagerActivity"
    android:parentActivityName=".CrimeListActivity">
</activity>
<provider
    android:name="android.support.v4.content.FileProvider"
    android:authorities="com.bignerdranch.android.criminalintent.fileprovider"
    android:exported="false"
    android:grantUriPermissions="true">
</provider>
```

Хранилищем называется место, в котором будут сохраняться файлы. Связывая `FileProvider` с хранилищем, вы предоставляете другим приложениям цель для запросов. Добавляя атрибут `exported="false"`, вы запрещаете использование провайдера всеми сторонами, которым вы не предоставите разрешение. Добавляя атрибут `grantUriPermissions`, вы добавляете возможность предоставлять другим приложениям право записи по URI для этого хранилища, передаваемым в интентах. (Об этом чуть позднее.)

Теперь, когда вы сообщили Android, где находится `FileProvider`, также необходимо сообщить `FileProvider`, какие файлы предоставляются. Эта конфигурация определяется в дополнительном ресурсном файле в формате XML. Щелкните правой кнопкой мыши на папке app/res в окне инструментов Project и выберите команду New ▸ Android resource file. Выберите тип ресурса XML и введите имена файлов.

Откройте файл xml/files.xml, переключитесь на вкладку Text и замените содержимое файла следующим:

Листинг 16.3. Заполнение описания путей (res/xml/files.xml)

```
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">

</PreferenceScreen>
<paths>
    <files-path name="crime_photos" path="."/>
</paths>
```

Этот файл XML по сути говорит: «Назначить корневым путем моего закрытого хранилища `crime_photos`». Вы не будете использовать имя `crime_photos` — `FileProvider` делает это в своей внутренней реализации.

Теперь свяжите files.xml с `FileProvider`, добавив тег `meta-data` в AndroidManifest.xml.

Листинг 16.4. Связывание с описанием путей (AndroidManifest.xml)

```
<provider
    android:name="android.support.v4.content.FileProvider"
    android:authorities="com.bignerdranch.android.criminalintent.fileprovider"
    android:exported="false"
    android:grantUriPermissions="true">
    <meta-data
        android:name="android.support.FILE_PROVIDER_PATHS"
        android:resource="@xml/files"/>
</provider>
```

Выбор места для хранения фотографии

Пора выделить фотографиям место, где они будут существовать. Сначала добавьте в `Crime` метод для получения имени файла.

Листинг 16.5. Добавление свойства для получения имени файла (Crime.java)

```
    public void setSuspect(String suspect) {
        mSuspect = suspect;
    }

    public String getPhotoFilename() {
        return "IMG_" + getId().toString() + ".jpg";
    }
}
```

Метод `Crime.getPhotoFilename()` не знает, в какой папке будет храниться фотография. Однако имя файла будет уникальным, поскольку оно строится на основании идентификатора `Crime`.

Затем следует найти место, в котором будут располагаться фотографии. Класс `CrimeLab` отвечает за все, что относится к долгосрочному хранению данных в CriminalIntent, поэтому он становится наиболее естественным кандидатом. Добавьте в `CrimeLab` метод `getPhotoFile(Crime)`, который будет возвращать эту информацию.

Листинг 16.6. Определение местонахождения файла фотографии (CrimeLab.java)

```
public class CrimeLab {
    ...

    public Crime getCrime(UUID id) {
        ...
    }
```

```
    public File getPhotoFile(Crime crime) {
        File filesDir = mContext.getFilesDir();
        return new File(filesDir, crime.getPhotoFilename());
        if (externalFilesDir == null) {
    }

    public void updateCrime(Crime crime) {
        ...
    }
```

Этот код не создает никакие файлы в файловой системе. Он только возвращает объекты `File`, представляющие нужные места. Позднее мы используем класс `FileProvider` для предоставления доступа к этим местам в виде URI.

Использование интента камеры

Следующий шаг — непосредственное создание снимка. Здесь все просто: необходимо снова воспользоваться неявным интентом.

Начнем с сохранения местонахождения файла фотографии. (Эта информация будет использоваться еще в нескольких местах, поэтому сохранение избавит от лишней работы.)

Листинг 16.7. Сохранение местонахождения файла фотографии (CrimeLab.java)

```
private Crime mCrime;
private File mPhotoFile;
private EditText mTitleField;
...
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    UUID crimeId = (UUID) getArguments().getSerializable(ARG_CRIME_ID);
    mCrime = CrimeLab.get(getActivity()).getCrime(crimeId);
    mPhotoFile = CrimeLab.get(getActivity()).getPhotoFile(mCrime);
}
```

На следующем шаге мы подключим кнопку, которая будет непосредственно создавать снимок. Интент камеры определяется в `MediaStore` — верховном повелителе всего, что относится к аудиовизуальной информации в Android. Интент отправляется действием `MediaStore.ACTION_IMAGE_CAPTURE`, по которому Android запускает активность камеры и делает снимок за вас.

Но не стоит торопить события.

Отправка интента

Теперь все готово к отправке интента камеры. Нужное действие `ACTION_CAPTURE_IMAGE` определяется в классе `MediaStore`. Этот класс определяет открытые интерфейсы, используемые в Android при работе с основными аудиовизуальными

материалами — изображениями, видео и музыкой. К этой категории относится и интент, запускающий камеру.

По умолчанию `ACTION_CAPTURE_IMAGE` послушно запускает приложение камеры и делает снимок, но результат не является фотографией в полном разрешении. Вместо нее создается миниатюра с малым разрешением, которая упаковывается в объект `Intent`, возвращаемый в `onActivityResult(…)`.

Чтобы получить выходное изображение в высоком разрешении, необходимо сообщить, где должно храниться изображение в файловой системе. Эта задача решается передачей URI для места, в котором должен сохраняться файл, в `MediaStore.EXTRA_OUTPUT`. URI будет указывать на место, предоставленное `FileProvider`.

Напишите неявный интент для сохранения фотографии в месте, определяемом `mPhotoFile`. Добавьте код, блокирующий кнопку при отсутствии приложения камеры или недоступности места, в котором должна сохраняться фотография. (Чтобы проверить доступность приложения камеры, запросите у `PackageManager` активности, реагирующие на неявный интент камеры. О том, как обращаться за информацией в `PackageManager`, более подробно рассказано в разделе «Проверка реагирующих активностей» главы 15.)

Листинг 16.8. Отправка интента камеры (CrimeFragment.java)

```
private static final int REQUEST_DATE = 0;
private static final int REQUEST_CONTACT = 1;
private static final int REQUEST_PHOTO= 2;
...
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    ...
    mPhotoButton = (ImageButton) v.findViewById(R.id.crime_camera);
    final Intent captureImage = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);

    boolean canTakePhoto = mPhotoFile != null &&
            captureImage.resolveActivity(packageManager) != null;
    mPhotoButton.setEnabled(canTakePhoto);

    mPhotoButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Uri uri = FileProvider.getUriForFile(getActivity(),
                    "com.bignerdranch.android.criminalintent.fileprovider",
                    mPhotoFile);
            captureImage.putExtra(MediaStore.EXTRA_OUTPUT, uri);
            List<ResolveInfo> cameraActivities = getActivity()
                    .getPackageManager().queryIntentActivities(captureImage,
                            PackageManager.MATCH_DEFAULT_ONLY);

            for (ResolveInfo activity : cameraActivities) {
                getActivity().grantUriPermission(activity.activityInfo.packageName,
```

```
                    uri, Intent.FLAG_GRANT_WRITE_URI_PERMISSION);
            }

            startActivityForResult(captureImage, REQUEST_PHOTO);

        }
    });
    mPhotoView = (ImageView) v.findViewById(R.id.crime_photo);

    return v;
}
```

Вызов `FileProvider.getUriForFile(…)` преобразует локальный путь к файлу в объект `Uri`, «понятный» приложению камеры. Но чтобы выполнить запись, необходимо установить флаг `Intent.FLAG_GRANT_WRITE_URI_PERMISSION` для каждой активности, с которой может быть связан интент `cameraImage`. Флаг предоставляет разрешение на запись для этого конкретного `Uri`. Включение атрибута `android:grantUriPermissions` в объявление провайдера было необходимо для того, чтобы разблокировать эту функциональность. Позднее вы отзовете это разрешение, чтобы снова закрыть эту брешь в защите вашего приложения.

Запустите CriminalIntent и нажмите кнопку запуска приложения камеры (рис. 16.4).

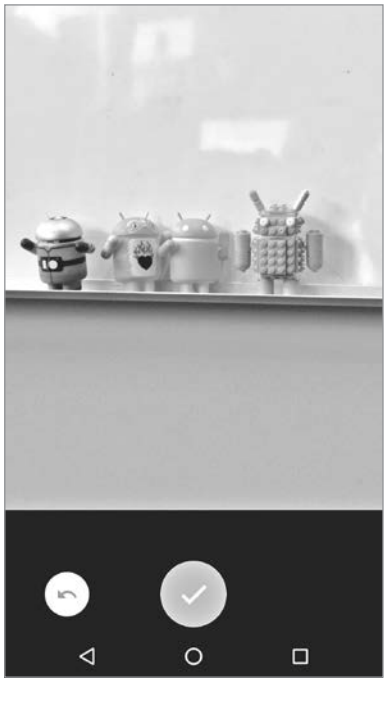


Рис. 16.4. [Подставьте ваше приложение для работы с камерой]

Масштабирование и отображение растровых изображений

После всего, что было сделано, приложение успешно делает снимки, которые сохраняются в файловой системе для дальнейшего использования.

Следующий шаг — поиск файла с изображением, его загрузка и отображение для пользователя. Для этого необходимо загрузить данные изображения в объект `Bitmap` достаточного размера. Чтобы построить объект `Bitmap` на базе файла, достаточно воспользоваться классом `BitmapFactory`:

```
Bitmap bitmap = BitmapFactory.decodeFile(mPhotoFile.getPath())
```

Но ведь должна же быть какая-то загвоздка, верно? Иначе мы бы просто напечатали эту строку жирным шрифтом, вы бы ввели ее — и на этом все закончилось.

Да, загвоздка существует: «достаточный размер» следует понимать буквально. `Bitmap` — простой объект для хранения необработанных данных пикселов. Таким образом, даже если исходный файл был сжат, в объекте `Bitmap` никакого сжатия не будет. Итак, 24-битовое изображение с камеры на 16 мегапикселов, которое может занимать всего 5 Мбайт в формате JPG, в объекте `Bitmap` разрастается до 48(!) Мбайт.

Найти обходное решение возможно, но оно означает, что изображение придется масштабировать вручную. Для этого можно сначала просканировать файл и определить его размер, затем вычислить, насколько его нужно масштабировать для того, чтобы он поместился в заданную область, и, наконец, заново прочитать файл для создания уменьшенного объекта `Bitmap`.

Создайте для этого метода новый класс с именем `PictureUtils.java` и добавьте в него статический метод с именем `getScaledBitmap(String, int, int)`.

Листинг 16.9. Создание метода getScaledBitmap(…) (PictureUtils.java)

```
public class PictureUtils {
    public static Bitmap getScaledBitmap(String path, int destWidth, int
destHeight) {
        // Чтение размеров изображения на диске
        BitmapFactory.Options options = new BitmapFactory.Options();
        options.inJustDecodeBounds = true;
        BitmapFactory.decodeFile(path, options);

        float srcWidth = options.outWidth;
        float srcHeight = options.outHeight;

        // Вычисление степени масштабирования
        int inSampleSize = 1;
        if (srcHeight > destHeight || srcWidth > destWidth) {
            float heightScale = srcHeight / destHeight;
            float widthScale = srcWidth / destWidth;
```

```
            inSampleSize = Math.round(heightScale > widthScale ? heightScale :
                    widthScale);
        }

        options = new BitmapFactory.Options();
        options.inSampleSize = inSampleSize;

        // Чтение данных и создание итогового изображения
        return BitmapFactory.decodeFile(path, options);
    }
}
```

Ключевой параметр `inSampleSize` определяет величину «образца» для каждого пиксела исходного изображения: образец с размером 1 содержит один горизонтальный пиксел для каждого горизонтального пиксела исходного файла, а образец с размером 2 содержит один горизонтальный пиксел для каждых двух горизонтальных пикселов исходного файла. Таким образом, если значение `inSampleSize` равно 2, количество пикселов в изображении составляет четверть от количества пикселов оригинала.

И последняя неприятная новость: при запуске фрагмента вы еще не знаете величину `PhotoView`. До обработки макета никаких экранных размеров не существует. Первый проход этой обработки происходит после выполнения `onCreate(…)`, `onStart()` и `onResume()`, поэтому `PhotoView` и не знает своих размеров.

У проблемы есть два решения: либо подождать, пока будет сделан первый проход, либо воспользоваться консервативной оценкой. Второй способ менее эффективен, но более прямолинеен. Напишите еще один статический метод с именем `getScaledBitmap(String, Activity)` для масштабирования `Bitmap` под размер конкретной активности.

Листинг 16.10. Метод масштабирования с консервативной оценкой (PictureUtils.java)

```
public class PictureUtils {
    public static Bitmap getScaledBitmap(String path, Activity activity) {
        Point size = new Point();
        activity.getWindowManager().getDefaultDisplay()
                .getSize(size);

        return getScaledBitmap(path, size.x, size.y);
    }
```

Метод проверяет размер экрана и уменьшает изображение до этого размера. Виджет `ImageView`, в который загружается изображение, всегда меньше размера экрана, так что эта оценка весьма консервативна.

Чтобы загрузить объект `Bitmap` в `ImageView`, добавьте в `CrimeFragment` метод для обновления `mPhotoView`.

Листинг 16.11. Обновление mPhotoView (CrimeFragment.java)

```
    private String getCrimeReport() {
        ...
    }

    private void updatePhotoView() {
        if (mPhotoFile == null || !mPhotoFile.exists()) {
            mPhotoView.setImageDrawable(null);
        } else {
            Bitmap bitmap = PictureUtils.getScaledBitmap(
                    mPhotoFile.getPath(), getActivity());
            mPhotoView.setImageBitmap(bitmap);
        }
    }
}
```

Затем вызовите этот метод из `onCreateView(…)` и `onActivityResult(…)`.

Листинг 16.12. Вызов updatePhotoView() (CrimeFragment.java)

```
    mPhotoButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            ...
            startActivityForResult(captureImage, REQUEST_PHOTO);
        }
    });

    mPhotoView = (ImageView) v.findViewById(R.id.crime_photo);
    updatePhotoView();

    return v;
}

@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode != Activity.RESULT_OK) {
        return;
    }

    if (requestCode == REQUEST_DATE) {
        ...
    } else if (requestCode == REQUEST_CONTACT && data != null) {
        ...
    } else if (requestCode == REQUEST_PHOTO) {
        Uri uri = FileProvider.getUriForFile(getActivity(),
                "com.bignerdranch.android.criminalintent.fileprovider",
                mPhotoFile);

        getActivity().revokeUriPermission(uri,
                Intent.FLAG_GRANT_WRITE_URI_PERMISSION);

        updatePhotoView();
    }
}
```

Теперь, когда камера завершила запись в файл, можно отозвать разрешение и снова перекрыть доступ к файлу. Запустите приложение снова. Изображение выведется в уменьшенном виде.

Объявление функциональности

Наша реализация камеры сейчас отлично работает. Остается решить еще одну задачу: сообщить о ней потенциальным пользователям. Когда приложение использует некоторое оборудование (например, камеру или NFC) или любой другой аспект, который может отличаться от устройства к устройству, настоятельно рекомендуется сообщить о нем Android. Это позволит другим приложениям (например, магазину Google Play) заблокировать установку приложения, если в нем используются возможности, не поддерживаемые вашим устройством.

Чтобы объявить, что в приложении используется камера, включите тег `<uses-feature>` в AndroidManifest.xml:

Листинг 16.13. Добавление тега uses-feature (AndroidManifest.xml)

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bignerdranch.android.criminalintent" >

    <uses-feature android:name="android.hardware.camera"
                  android:required="false"
        />
```

В этом примере в тег добавляется необязательный атрибут `android:required`. Почему? По умолчанию объявление об использовании некоторой возможности означает, что без нее приложение может работать некорректно. К CriminalIntent это не относится. Мы вызываем `resolveActivity(…)`, чтобы проверить наличие приложения камеры, после чего корректно блокируем кнопку, если приложение не найдено.

Атрибут `android:required="false"` корректно обрабатывает эту ситуацию. Мы сообщаем Android, что приложение может нормально работать без камеры, но некоторые части приложения окажутся недоступными.

Упражнение. Вывод увеличенного изображения

Конечно, вы видите уменьшенное изображение, но вряд ли вам удастся рассмотреть его во всех подробностях. Создайте новый фрагмент `DialogFragment`, в котором отображается увеличенная версия фотографии места преступления. Когда пользователь нажимает в какой-то точке миниатюры, на экране должен появиться `DialogFragment` с увеличенным изображением.

Упражнение. Эффективная загрузка миниатюры

В этой главе нам пришлось использовать довольно грубую оценку размера, до которого должно быть уменьшено изображение. Такое решение не идеально, но оно работает и быстро реализуется.

С существующими API можно использовать `ViewTreeObserver` — объект, который можно получить от любого представления в иерархии `Activity`:

```
ViewTreeObserver observer = mImageView.getViewTreeObserver();
```

Вы можете зарегистрировать для `ViewTreeObserver` разнообразных слушателей, включая `OnGlobalLayoutListener`. Этот слушатель инициирует событие каждый раз, когда происходит проход обработки макета.

Измените свой код так, чтобы он использовал размеры `mPhotoView`, когда они действительны, и ожидал прохода обработки макета перед первым вызовом `updatePhotoView()`.

17 Двухпанельные интерфейсы

В этой главе мы создадим для CriminalIntent планшетный интерфейс, в котором пользователь может одновременно видеть и взаимодействовать со списком преступлений и подробным описанием конкретного преступления. На рис. 17.1 изображен такой интерфейс, также часто называемый интерфейсом типа «*список-детализация*».

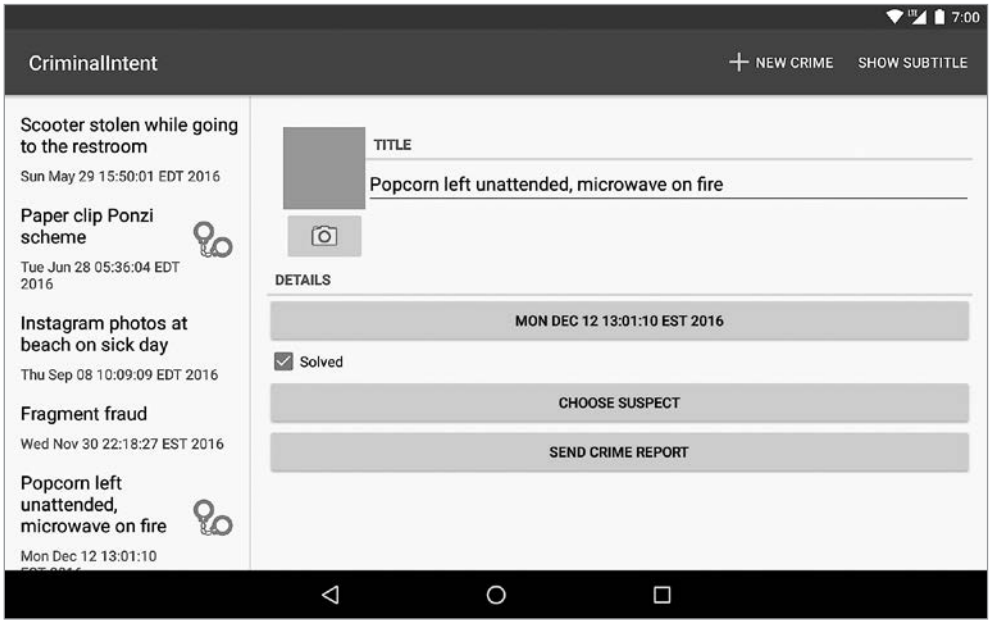


Рис. 17.1. Главное и детализированное представления одновременно находятся на экране

Для тестирования программ этой главы вам понадобится планшетное устройство или AVD. Чтобы создать виртуальный планшет AVD, выполните команду Tools ▸ Android ▸ Android Virtual Device Manager. Щелкните на кнопке Create Virtual

Device... и выберите слева категорию Tablet. Выберите аппаратный профиль, щелкните на кнопке Next и задайте целевой API не менее уровня 21 (рис. 17.2).

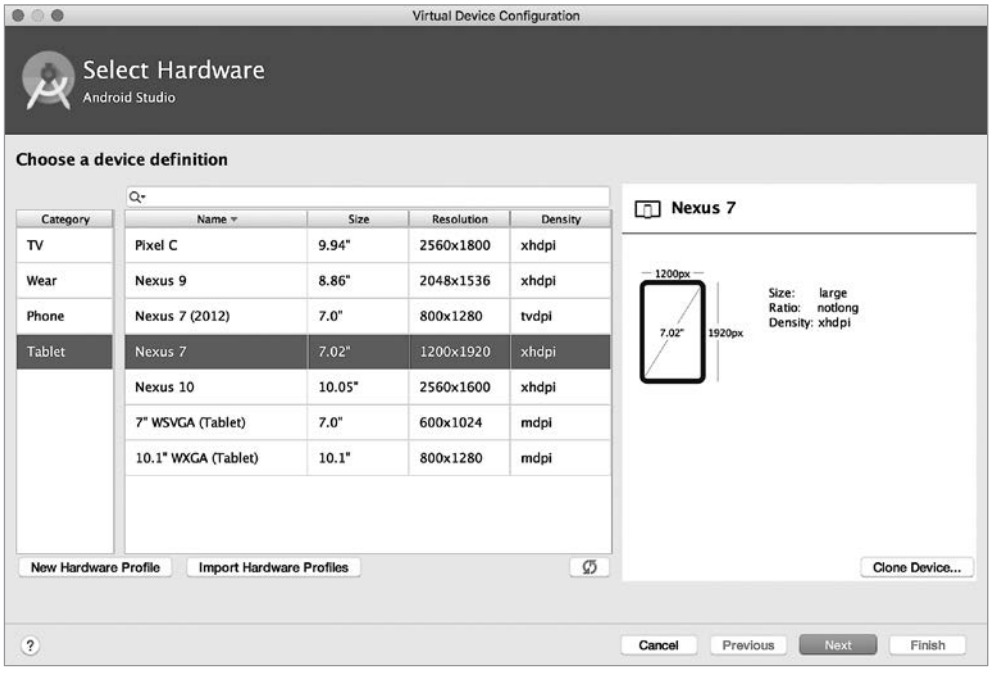


Рис. 17.2. Выбор устройства для планшетного AVD

Гибкость макета

На телефоне активность `CrimeListActivity` должна заполнять однопанельный макет, как она делает в настоящее время. На планшете она должна заполнять двухпанельный макет, способный одновременно отображать главное и детализированное представления.

В двухпанельном макете `CrimeListActivity` будет отображать как `CrimeListFragment`, так и `CrimeFragment`, как показано на рис. 17.3.

Для этого необходимо:

- изменить `SingleFragmentActivity`, чтобы выбор заполняемого макета не был жестко фиксирован в программе;
- создать новый макет, состоящий из двух контейнеров фрагментов;
- изменить `CrimeListActivity`, чтобы на телефонах заполнялся однопанельный макет, а на планшетах — двухпанельный.

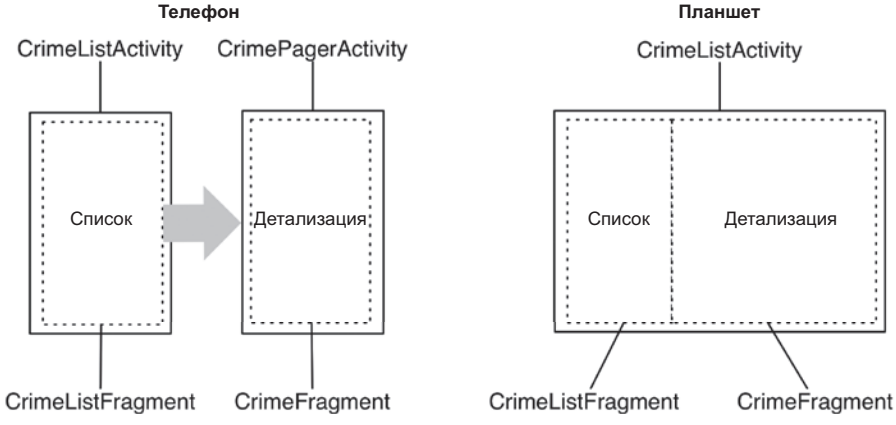


Рис. 17.3. Разновидности макетов

Модификация SingleFragmentActivity

`CrimeListActivity` является субклассом `SingleFragmentActivity`. В настоящее время класс `SingleFragmentActivity` настроен таким образом, чтобы он всегда заполнял activity_fragment.xml. Чтобы класс `SingleFragmentActivity` стал более гибким, мы сделаем так, чтобы субкласс мог предоставлять свой идентификатор ресурса макета.

В файле SingleFragmentActivity.java добавьте защищенный метод, который возвращает идентификатор макета, заполняемого активностью.

Листинг 17.1. Обеспечение гибкости SingleFragmentActivity (SingleFragmentActivity.java)

```
public abstract class SingleFragmentActivity extends AppCompatActivity {
    protected abstract Fragment createFragment();

    @LayoutRes
    protected int getLayoutResId() {
        return R.layout.activity_fragment;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fragment);
        setContentView(getLayoutResId());

        FragmentManager fm = getSupportFragmentManager();
        ...
    }
}
```

Реализация класса `SingleFragmentActivity` по умолчанию будет работать так же, как и прежде, но теперь его субклассы могут переопределить `getLayoutResId()` для воз-

вращения макета, отличного от activity_fragment.xml. Метод `getLayoutResId()` помечается аннотацией `@LayoutRes`, чтобы сообщить Android Studio, что любая реализация этого метода должна возвращать действительный идентификатор ресурса макета.

Создание макета с двумя контейнерами фрагментов

В окне инструментов Project щелкните правой кнопкой мыши на каталоге res/layout/ и создайте новый файл Android в формате XML. Убедитесь в том, что для файла выбран тип ресурса Layout, присвойте файлу имя activity_twopane.xml и назначьте его корневым элементом `LinearLayout`.

Используйте рис. 17.4 для построения разметки XML макета с двумя панелями.

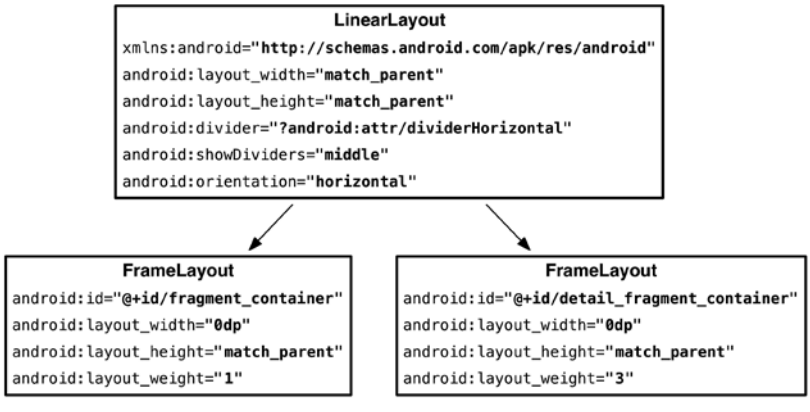


Рис. 17.4. Макет с двумя контейнерами фрагментов (layout/activity_twopane.xml)

Обратите внимание: у первого виджета `FrameLayout` задан идентификатор макета `fragmentContainer`, поэтому код `SingleFragmentActivity.onCreate(…)` может работать так же, как прежде. При создании активности фрагмент, возвращаемый `createFragment()`, появится на левой панели.

Протестируйте макет в `CrimeListActivity`; для этого переопределите метод `getLayoutResId()` так, чтобы он возвращал `R.layout.activity_twopane`.

Листинг 17.2. Переход к файлу двухпанельного макета (CrimeListActivity.java)

```
public class CrimeListActivity extends SingleFragmentActivity {
    @Override
    protected Fragment createFragment() {
        return new CrimeListFragment();
    }

    @Override
    protected int getLayoutResId() {
        return R.layout.activity_twopane;
    }
}
```

Запустите приложение CriminalIntent на планшетном устройстве и убедитесь в том, что на экране отображаются две панели (рис. 17.5). (Чтобы увидеть панели, необходимо добавить преступление.) Большая панель детализации пуста, а нажатие на элементе списка не отображает подробную информацию о преступлении. Контейнер детализированного представления будет подключен далее в этой главе.

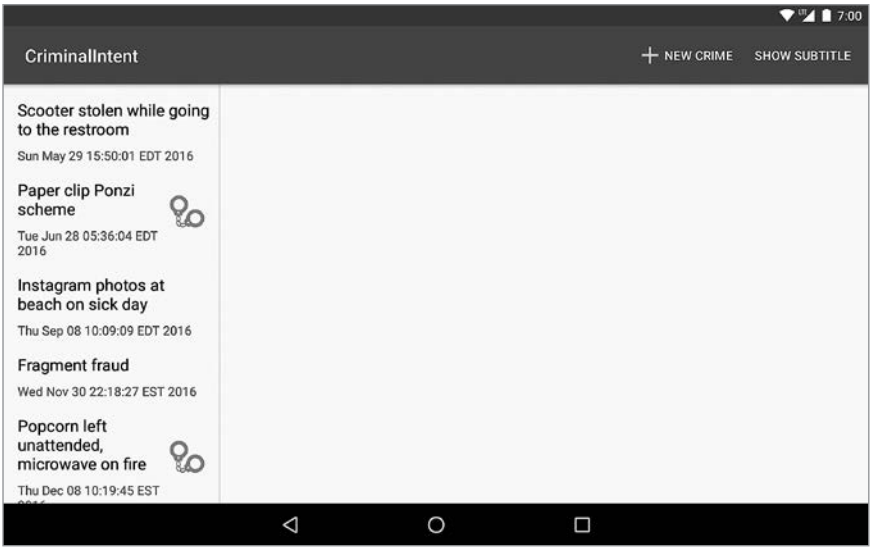


Рис. 17.5. Двухпанельный макет на планшете

В своей текущей версии `CrimeListActivity` также заполняет двухпанельный интерфейс при запуске на телефоне. В следующем разделе мы исправим этот недостаток при помощи *ресурса-псевдонима*.

Использование ресурса-псевдонима

Ресурс-псевдоним (alias resource) представляет собой ресурс, указывающий на другой ресурс. Ресурсы-псевдонимы находятся в каталоге res/values/ и по умолчанию определяются в файле refs.xml.

На следующем шаге мы должны добиться того, чтобы в `CrimeListActivity` отображались разные файлы макетов в зависимости от того, на каком устройстве работает приложение. Это делается точно так же, как мы отображаем разные макеты для книжной и альбомной ориентации: при помощи квалификатора ресурса.

Решение на уровне файлов в res/layout работает, но у него есть свои недостатки. Каждый файл макета должен содержать полную копию отображаемого макета, а это может привести к заметной избыточности. Чтобы создать файл макета activity_masterdetail.xml, пришлось бы скопировать все содержимое activity_fragment.xml в res/layout/activity_masterdetail.xml, а все содержимое activity_twopane.

xml — в res/layout-sw600dp/activity_masterdetail.xml. (Вскоре вы увидите, что означает sw600dp.)

Вместо этого мы воспользуемся ресурсом-псевдонимом. В этом разделе мы создадим ресурс-псевдоним, который ссылается на activity_fragment.xml на телефонах и макет activity_twopane.xml на планшетах.

В окне инструментов Project щелкните правой кнопкой мыши на каталоге res/layout/ и создайте новый ресурсный файл значений. Присвойте файлу имя refs.xml, а каталогу — имя values. Квалификаторов в именах быть не должно. Щелкните на кнопке Finish. Затем добавьте элемент, приведенный в листинге 17.3.

Листинг 17.3. Создание значения по умолчанию для ресурса-псевдонима (res/values/refs.xml)

```
<resources>
  <item name="activity_masterdetail" type="layout">@layout/activity_fragment</item>
</resources>
```

Значение ресурса представляет собой ссылку на однопанельный макет. Ресурс также обладает идентификатором: `R.layout.activity_masterdetail`. Обратите внимание: внутренний класс идентификатора определяется атрибутом `type` псевдонима. И хотя сам псевдоним находится в res/values/, его идентификатор хранится в `R.layout`.

Теперь этот идентификатор ресурса может использоваться вместо `R.layout.activity_fragment`. Внесите следующее изменение в `CrimeListActivity`.

Листинг 17.4. Повторная замена макета (CrimeListActivity.java)

```
@Override
protected int getLayoutResId() {
    return R.layout.activity_~~twopane~~;masterdetail;
}
```

Запустите приложение CriminalIntent и убедитесь в том, что псевдоним работает правильно. Активность `CrimeListActivity` снова должна заполнять однопанельный макет.

Создание альтернативы для планшета

Так как псевдоним находится в res/values/, он используется по умолчанию. Следовательно, по умолчанию `CrimeListActivity` заполняет однопанельный макет.

Теперь мы создадим альтернативный ресурс, чтобы псевдоним `activity_masterdetail` на планшетных устройствах ссылался на activity_twopane.xml.

В окне инструментов Project щелкните правой кнопкой мыши на каталоге res/values и создайте новый файл ресурсов значений. Присвойте ему имя refs.xml и снова присвойте каталогу имя values. Но на этот раз выберите в категории Available qualifiers вариант Smallest Screen Width и щелкните на кнопке >>, чтобы переместить квалификатор вправо (рис. 17.6).

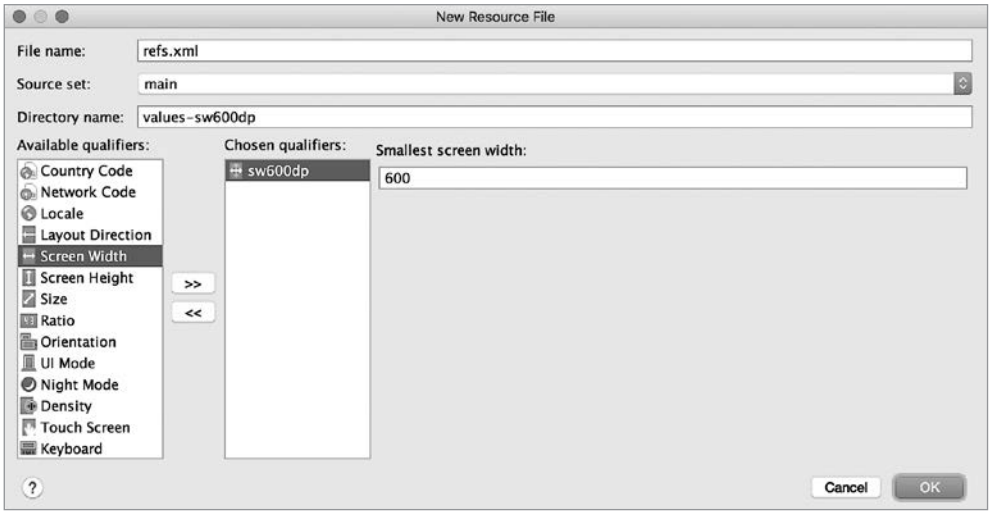


Рис. 17.6. Добавление квалификатора

Этот квалификатор работает несколько иначе: вам предлагается задать значение минимальной длины экрана. Введите 600 и щелкните на кнопке OK. После открытия нового файла ресурсов добавьте псевдоним `activity_masterdetail` и в этот файл, но свяжите его с другим файлом макета.

Листинг 17.5. Альтернативный псевдоним для устройств с экраном большего размера (res/values-sw600dp/refs.xml)

```
<resources>
  <item name="activity_masterdetail" type="layout">@layout/activity_twopane
            </item>
</resources>
```

Разберемся, что же здесь происходит. Наша цель — сформировать логику, которая работает следующим образом:

- для устройств с экраном меньше заданного размера использовать activity_fragment.xml;
- для устройств с экраном больше заданного размера использовать activity_twopane.xml.

Android не предоставляет возможности использовать ресурс только в том случае, если экран устройства меньше определенного размера, но предлагает неплохую альтернативу. Конфигурационный квалификатор `-sw600dp` позволяет предоставить ресурсы только в том случае, если экран устройства больше определенного размера. Сокращение «sw» происходит от «Smallest Width» (наименьшая ширина), но относится к меньшему размеру экрана, а следовательно, не зависит от текущей ориентации устройства.

Используя квалификатор `-sw600dp`, вы указываете: «Этот ресурс должен использоваться на любых устройствах, у которых меньший размер составляет 600dp и выше». Это хороший критерий для определения экранов планшетных устройств.

А что делать с другой частью, где нужно использовать activity_fragment.xml на меньших устройствах? Меньшие устройства не соответствуют квалификатору `-sw600dp`, поэтому для них будет использован вариант по умолчанию: activity_fragment.xml.

Запустите приложение CriminalIntent на телефоне и на планшете. Убедитесь в том, что одно- и двухпанельные макеты отображаются там, где предполагалось.

Активность: управление фрагментами

Итак, макеты ведут себя так, как положено, и мы можем перейти к добавлению `CrimeFragment` в контейнер фрагмента детализации при использовании двухпанельного макета `CrimeListActivity`.

На первый взгляд кажется, что для этого достаточно написать альтернативную реализацию `CrimeHolder.onClick(View)` для планшетов. Вместо запуска нового экземпляра `CrimePagerActivity` метод `onClick(View)` получает экземпляр `FragmentManager`, принадлежащий `CrimeListActivity`, и закрепляет транзакцию, которая добавляет `CrimeFragment` в контейнер фрагмента детализации.

Код из `CrimeListFragment.CrimeHolder` выглядит примерно так:

```
public void onClick(View view) {
    // Включение нового экземпляра CrimeFragment в макете активности
    Fragment fragment = CrimeFragment.newInstance(mCrime.getId());
    FragmentManager fm = getActivity().getSupportFragmentManager();
    fm.beginTransaction()
        .add(R.id.detail_fragment_container, fragment)
        .commit();
}
```

Такое решение работает, но оно противоречит хорошему стилю программирования Android. Предполагается, что фрагменты представляют собой автономные компонуемые блоки. Если написанный вами фрагмент добавляет фрагменты в `FragmentManager` активности, значит, он делает допущения относительно того, как работает активность-хост, и перестает быть автономным компонуемым блоком.

Например, в приведенном выше коде `CrimeListFragment` добавляет `CrimeFragment` в `CrimeListActivity` и предполагает, что в макете `CrimeListActivity` присутствует контейнер `detail_fragment_container`. Такими вопросами должна заниматься активность-хост `CrimeListFragment`, а не `CrimeListFragment`.

Для сохранения независимости фрагментов мы делегируем выполнение работы активности-хосту, определяя интерфейсы обратного вызова в ваших фрагментах. Активности-хосты реализуют эти интерфейсы для выполнения операций по управлению фрагментами и обеспечения макетно-зависимого поведения.

Интерфейсы обратного вызова фрагментов

Чтобы делегировать функциональность активности-хосту, фрагмент обычно определяет интерфейс обратного вызова с именем `Callbacks`. Этот интерфейс определяет работу, которая должна быть выполнена для фрагмента его «начальником» — активностью-хостом. Любая активность, выполняющая функции хоста фрагментов, должна реализовать этот интерфейс.

С интерфейсом обратного вызова фрагмент может вызывать методы активности-хоста, не располагая никакой информацией о ней.

Реализация CrimeListFragment.Callbacks

Чтобы реализовать интерфейс `Callbacks`, следует сначала определить переменную для хранения объекта, реализующего `Callbacks`. Затем активность-хост преобразуется к `Callbacks`, а результат присваивается этой переменной.

Активность назначается в методе жизненного цикла `Fragment`:

```
public void onAttach(Context context)
```

Этот метод вызывается при присоединении фрагмента к активности (независимо от того, был он сохранен или нет). Вспомните: `Activity` является субклассом `Context`, поэтому `onAttach` передает в параметре `Context`; такое решение получается более гибким. Убедитесь в том, что для `onAttach` используется сигнатура `onAttach(Context)`, а не устаревший метод `onAttach(Activity)`, который может быть исключен из будущих версий API.

Аналогичным образом переменной присваивается `null` в соответствующем завершающем методе жизненного цикла:

```
public void onDetach()
```

Переменной присваивается `null`, потому что в дальнейшем вы не сможете обратиться к активности или рассчитывать на то, что активность продолжит существовать.

В файле CrimeListFragment.java включите в класс `CrimeListFragment` интерфейс `Callbacks`. Также добавьте переменную `mCallbacks` и переопределите методы `onAttach(Activity)` и `onDetach()`, в которых задается и сбрасывается ее значение.

Листинг 17.6. Добавление интерфейса обратного вызова (CrimeListFragment.java)

```
public class CrimeListFragment extends Fragment {
    ...
    private boolean mSubtitleVisible;
    private Callbacks mCallbacks;

    /**
     * Обязательный интерфейс для активности-хоста.
     */
    public interface Callbacks {
        void onCrimeSelected(Crime crime);
```

```
    }

    @Override
    public void onAttach(Context context) {
        super.onAttach(context);
        mCallbacks = (Callbacks) context;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setHasOptionsMenu(true);
    }
    ...
    @Override
    public void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        outState.putBoolean(SAVED_SUBTITLE_VISIBLE, mSubtitleVisible);
    }

    @Override
    public void onDetach() {
        super.onDetach();
        mCallbacks = null;
    }
}
```

Теперь у `CrimeListFragment` имеется механизм вызова методов активности-хоста. Неважно, какая активность является хостом, — если она реализует `CrimeListFragment.Callbacks`, внутренняя реализация `CrimeListFragment` будет работать одинаково.

Обратите внимание на то, как `CrimeListFragment` выполняет непроверяемое преобразование своей активности к `CrimeListFragment.Callbacks`. Это означает, что активность-хост *должна* реализовать `CrimeListFragment.Callbacks`. В самой зависимости нет ничего плохого, но ее важно документировать.

Затем в классе `CrimeListActivity` реализуйте `CrimeListFragment.Callbacks`. Метод `onCrimeSelected(Crime)` пока оставьте пустым.

Листинг 17.7. Реализация обратных вызовов (CrimeListActivity.java)

```
public class CrimeListActivity extends SingleFragmentActivity
    implements CrimeListFragment.Callbacks {

    @Override
    protected Fragment createFragment() {
        return new CrimeListFragment();
    }

    @Override
```

```
    protected int getLayoutResId() {
        return R.layout.activity_masterdetail;
    }

    @Override
    public void onCrimeSelected(Crime crime) {
    }
}
```

`CrimeListFragment` будет вызывать этот метод в `CrimeHolder.onClick(…)`, а также тогда, когда пользователь выбирает команду создания новой записи преступления. Для начала нужно понять, как должна быть устроена реализация `CrimeListActivity.onCrimeSelected(Crime)`.

При вызове `onCrimeSelected(Crime)` класс `CrimeListActivity` должен выполнить одну из двух операций:

- если используется телефонный интерфейс — запустить новый экземпляр `CrimePagerActivity`;
- если используется планшетный интерфейс — поместить `CrimeFragment` в `detail_fragment_container`.

Чтобы определить, интерфейс какого типа был заполнен, можно проверить конкретный идентификатор интерфейса. Впрочем, лучше проверить наличие в макете `detail_fragment_container`. Такая проверка будет более точной и надежной. Имена файлов могут изменяться, и вас на самом деле не интересует, по какому файлу заполнялся макет; необходимо знать лишь то, имеется ли у него контейнер `detail_fragment_container` для размещения `CrimeFragment`.

Если макет содержит `detail_fragment_container`, мы создадим транзакцию фрагмента, которая удаляет существующий экземпляр `CrimeFragment` из `detail_fragment_container` (если он имеется) и добавляет экземпляр `CrimeFragment`, который мы хотим там видеть.

В файле CrimeListActivity.java реализуйте метод `onCrimeSelected(Crime)`, который будет обрабатывать выбор преступления в любом варианте интерфейса.

Листинг 17.8. Условный запуск CrimeFragment (CrimeListActivity.java)

```
@Override
public void onCrimeSelected(Crime crime) {
    if (findViewById(R.id.detail_fragment_container) == null) {
        Intent intent = CrimePagerActivity.newIntent(this, crime.getId());
        startActivity(intent);
    } else {
        Fragment newDetail = CrimeFragment.newInstance(crime.getId());

        getSupportFragmentManager().beginTransaction()
                .replace(R.id.detail_fragment_container, newDetail)
                .commit();
    }
}
```

Наконец, в классе `CrimeListFragment` мы будем вызывать `onCrimeSelected(Crime)` в тех местах, где сейчас запускается новый экземпляр `CrimePagerActivity`.

В файле CrimeListFragment.java измените методы `onOptionsItemSelected(MenuItem)` и `CrimeHolder.onClick(View)` так, чтобы в них вызывался метод `Callbacks.onCrimeSelected(Crime)`.

Листинг 17.9. Активизация обратных вызовов (CrimeListFragment.java)

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.new_crime:
            Crime crime = new Crime();
            CrimeLab.get(getActivity()).addCrime(crime);
            Intent intent = CrimePagerActivity
                    .newIntent(getActivity(), crime.getId());
            startActivity(intent);
            updateUI();
            mCallbacks.onCrimeSelected(crime);
            return true;
        ...
    }
}
...
private class CrimeHolder extends RecyclerView.ViewHolder
        implements View.OnClickListener {
    ...
    @Override
    public void onClick(View view) {
        Intent intent = CrimePagerActivity.newIntent(getActivity(),
                        mCrime.getId());
        startActivity(intent);
        mCallbacks.onCrimeSelected(mCrime);
    }
}
```

При обратном вызове в `onOptionsItemSelected(…)` содержимое списка также немедленно перезагружается после добавления нового преступления. Это необходимо, потому что на планшетах при добавлении нового преступления список остается видимым на экране (прежде его закрывал экран детализации).

Запустите приложение CriminalIntent на планшете. Создайте новое преступление; экземпляр `CrimeFragment` добавляется и отображается в `detail_fragment_container`. Затем просмотрите какое-либо старое преступление и убедитесь в том, что `CrimeFragment` заменяется новым экземпляром (рис. 17.7).

Выглядит замечательно! Но есть одна маленькая проблема: внесение изменений в преступление не приводит к обновлению списка. В настоящее время список перезагружается только после добавления преступления и в `CrimeListFragment.onResume()`. При этом на планшетах экземпляр `CrimeListFragment` остается видимым рядом с `CrimeFragment`. `CrimeListFragment` не приостанавливается при появлении `CrimeFragment`, поэтому и возобновления не происходит, а список не перезагружается.

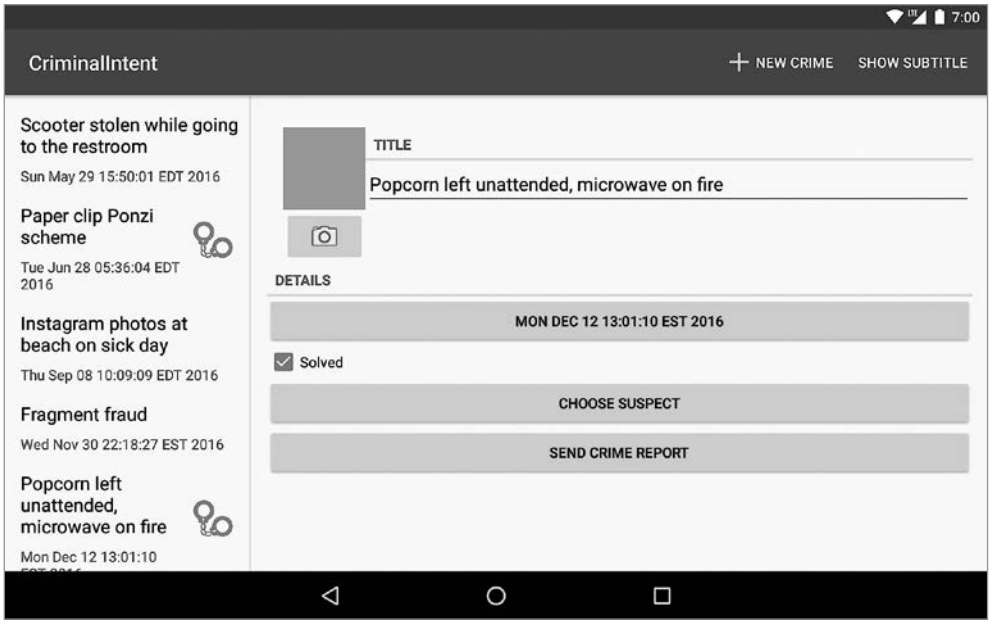


Рис. 17.7. Главное и детализированное представления связаны между собой

Для решения этой проблемы мы используем другой интерфейс обратного вызова из `CrimeFragment`.

Реализация CrimeFragment.Callbacks

`CrimeFragment` определяет следующий интерфейс:

```
public interface Callbacks {
    void onCrimeUpdated(Crime crime);
}
```

Чтобы класс `CrimeFragment` «проталкивал» обновления в другой `Fragment`, должны выполняться два условия. Во-первых, так как единственным источником достоверной информации для CriminalIntent является база данных SQLite, он должен сохранить объект `Crime` в `CrimeLab`. Затем `CrimeFragment` будет вызывать метод `onCrimeUpdated(Crime)` активности-хоста при сохранении любых изменений в `Crime`. `CrimeListActivity` реализует `onCrimeUpdated(Crime)` для перезагрузки списка `CrimeListFragment`, что приводит к извлечению обновленной информации из базы данных и ее отображению.

Прежде чем браться за интерфейс в `CrimeFragment`, измените видимость метода `CrimeListFragment.updateUI()`, чтобы он мог вызываться из `CrimeListActivity`.

Листинг 17.10. Изменение видимости updateUI() (CrimeListFragment.java)

```
~~private~~ public void updateUI() {
    ...
}
```

В файле CrimeListFragment.java добавьте интерфейс обратного вызова вместе с переменной `mCallbacks` и реализациями `onAttach(…)` и `onDetach()`.

Листинг 17.11. Добавление обратных вызовов CrimeFragment (CrimeFragment.java)

```
private ImageButton mPhotoButton;
private ImageView mPhotoView;
private Callbacks mCallbacks;

/**
 * Необходимый интерфейс для активности-хоста.
 */
public interface Callbacks {
    void onCrimeUpdated(Crime crime);
}

public static CrimeFragment newInstance(UUID crimeId) {
    ...
}

@Override
public void onAttach(Context context) {
    super.onAttach(context);
    mCallbacks = (Callbacks) context;
}

@Override
public void onCreate(Bundle savedInstanceState) {
    ...
}

@Override
public void onPause() {
    ...
}

@Override
public void onDetach() {
    super.onDetach();
    mCallbacks = null;
}
```

Затем в классе `CrimeListActivity` реализуйте `CrimeListFragment.Callbacks` для перезагрузки списка в `onCrimeUpdated(Crime)`.

Листинг 17.12. Обновление списка преступлений (CrimeListActivity.java)

```
public class CrimeListActivity extends SingleFragmentActivity
    implements CrimeListFragment.Callbacks, CrimeFragment.Callbacks {
    ...
    public void onCrimeUpdated(Crime crime) {
        CrimeListFragment listFragment = (CrimeListFragment)
                getSupportFragmentManager()
```

```
                        .findFragmentById(R.id.fragment_container);
        listFragment.updateUI();
    }
}
```

Интерфейс `CrimeFragment.Callbacks` должен быть реализован во всех активностях, выполняющих функции хоста для `CrimeFragment`. Следовательно, пустую реализацию также следует включить и в `CrimePagerActivity`.

Листинг 17.13. Реализация пустых обратных вызовов (CrimePagerActivity.java)

```
public class CrimePagerActivity extends AppCompatActivity
        implements CrimeFragment.Callbacks {
    ...
    @Override
    public void onCrimeUpdated(Crime crime) {

    }
}
```

`CrimeFragment` в своей внутренней работе часто будет выполнять этот хитрый маневр из двух шагов: шаг влево, сохранить `mCrime` в `CrimeLab`. Шаг вправо, вызвать `mCallbacks.onCrimeUpdated(Crime)`. Добавим метод, чтобы упростить его выполнение.

Листинг 17.14. Добавление метода updateCrime() (CrimeFragment.java)

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    ...
}

private void updateCrime() {
    CrimeLab.get(getActivity()).updateCrime(mCrime);
    mCallbacks.onCrimeUpdated(mCrime);
}

private void updateDate() {
    mDateButton.setText(mCrime.getDate().toString());
}
```

Добавьте в CrimeFragment.java вызовы `updateCrime()` при изменении краткого описания или состояния раскрытия преступления.

Листинг 17.15. Вызов onCrimeUpdated(Crime) (CrimeFragment.java)

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    ...
    mTitleField.addTextChangedListener(new TextWatcher() {
        ...
        @Override
        public void onTextChanged(CharSequence s, int start, int before, int count) {
```

```
            mCrime.setTitle(s.toString());
            updateCrime();
        }
        ...
    });
    ...
    mSolvedCheckbox.setOnCheckedChangeListener(new OnCheckedChangeListener() {
        @Override
        public void onCheckedChanged(CompoundButton buttonView,
                boolean isChecked) {
            mCrime.setSolved(isChecked);
            updateCrime();
        }
    });
    ...
}
```

Также необходимо вызвать `updateCrime()` в `onActivityResult(…)` при возможном изменении даты, фотографии и подозреваемого. В настоящее время фотография и подозреваемый не отображаются в представлении элемента списка, но `CrimeFragment` может сообщить об этих обновлениях.

Листинг 17.16. Повторный вызов updateCrime() (CrimeFragment.java)

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    ...
    if (requestCode == REQUEST_DATE) {
        Date date = (Date) data
                .getSerializableExtra(DatePickerFragment.EXTRA_DATE);
        mCrime.setDate(date);
        updateCrime();
        updateDate();
    } else if (requestCode == REQUEST_CONTACT && data != null) {
        ...
        try {
            ...
            String suspect = c.getString(0);
            mCrime.setSuspect(suspect);
            updateCrime();
            mSuspectButton.setText(suspect);
        } finally {
            c.close();
        }
    } else if (requestCode == REQUEST_PHOTO) {
        ...
        getActivity().revokeUriPermission(uri,
                Intent.FLAG_GRANT_WRITE_URI_PERMISSION);
        updateCrime();
        updatePhotoView();
    }
}
```

Запустите приложение CriminalIntent на планшете и убедитесь в том, что `RecyclerView` обновляется при внесении изменений в `CrimeFragment`. Затем запустите его на телефоне и убедитесь, что приложение работает как прежде.

Таким образом, ваше приложение работает как на планшетах, так и на телефонах.

Для любознательных: подробнее об определении размера экрана

До выхода Android 3.2 для предоставления альтернативных ресурсов в зависимости от размера устройства использовался квалификатор размера экрана. Этот квалификатор группирует разные устройства на четыре категории — `small`, `normal`, `large` и `xlarge`.

В табл. 17.1 приведены минимальные размеры для каждого квалификатора.

Таблица 17.1. Квалификаторы размера экрана

Имя	Минимальный размер экрана
small	320 × 426dp
normal	320 × 470dp
large	480 × 640dp
xlarge	720 × 960dp

Квалификаторы размера экрана были объявлены устаревшими в Android 3.2. Они были заменены дискретными квалификаторами, позволяющими определять размеры устройства. В табл. 17.2 перечислены эти новые квалификаторы.

Таблица 17.2. Дискретные квалификаторы размера экрана

Формат квалификатора	Описание
wXXXdp	Доступная ширина: ширина больше либо равна XXX dp
hXXXdp	Доступная высота: высота больше либо равна XXX dp
swXXXdp	Минимальная ширина: ширина или высота (меньшая из двух) больше либо равна XXX dp

Предположим, вы хотите задать макет, который должен использоваться только в том случае, если ширина экрана не менее 300dp. В этом случае можно поместить файл макета в каталог res/layout-w300dp («w» — сокращение от «width», то есть «ширина»). То же самое можно сделать для высоты при помощи префикса «h» («height», то есть «высота»).

Впрочем, ширина и высота могут меняться местами в зависимости от ориентации устройства. Для обнаружения конкретного размера экрана используется префикс `sw` («smallest width», то есть «минимальная ширина»). Он задает наименьший размер экрана, которым в зависимости от ориентации устройства может быть как ширина, так и высота. Если размеры экрана равны 1024×800, то метрика `sw` равна 800. Если размеры экрана равны 800×1024, то метрика `sw` все равно равна 800.

Упражнение. Удаление смахиванием

В этом упражнении вы добавите функциональность удаления смахиванием, которая сделает работу с `RecyclerView` в CriminalIntent более удобной. Реализация закрытия смахиванием позволит пользователю удалить преступление, проведя пальцем по экрану слева направо.

Чтобы настроить эту функциональность для виджета `RecyclerView` в `CrimeFragment`, подключите виджет `ItemTouchHelper` (*developer.android.com/reference/android/support/v7/widget/helper/ItemTouchHelper.html*). `ItemTouchHelper` предоставляет реализацию удаления смахиванием и включается в `RecyclerView` в библиотеке поддержки.

18 Локализация

Предвидя бешеную популярность приложения CriminalIntent, вы решаете сделать его доступным для большей аудитории. Первым шагом должна стать локализация всего текста, видимого пользователю, чтобы пользователь мог общаться с приложением как на английском, так и на испанском языке.

Локализацией называется процесс формирования ресурсов приложения в зависимости от языка, выбранного пользователем. В этой главе мы создадим испаноязычную версию файла strings.xml. Когда на устройстве выбирается испанский язык, Android на стадии выполнения автоматически находит и использует испанские строки (рис. 18.1).

Рис. 18.1. IntentoCriminal

Локализация ресурсов

Языковые настройки являются частью конфигурации устройства (см. раздел «Конфигурации устройств и альтернативные ресурсы» в главе 3). Android предоставляет квалификаторы для разных языков, по аналогии с ориентацией, размером экрана и другими аспектами конфигурации. Процесс локализации становится достаточно тривиальным: вы создаете подкаталоги ресурсов, снабжаете их квалификаторами нужных языков, и размещаете в них альтернативные ресурсы. Система ресурсов Android сделает все остальное.

В проекте CriminalIntent создайте файл ресурсов со значениями (как это делалось ранее в главе 17): в окне инструментов Project щелкните правой кнопкой мыши на res/values/ и выберите команду New ▸ Values resource file. В поле File name введите имя strings. Оставьте в поле Source set значение main. Убедитесь в том, что в поле Directory name выбрано значение values. Выберите в списке Available qualifiers вариант Locale и щелкните на кнопке >>, чтобы переместить Locale в группу Chosen qualiifiers. Выберите в списке Language значение es: Spanish. В списке Specific Region Only автоматически выбирается вариант Any region — именно это вам и нужно, оставьте этот вариант.

Окно New Resource File показано на рис. 18.2.

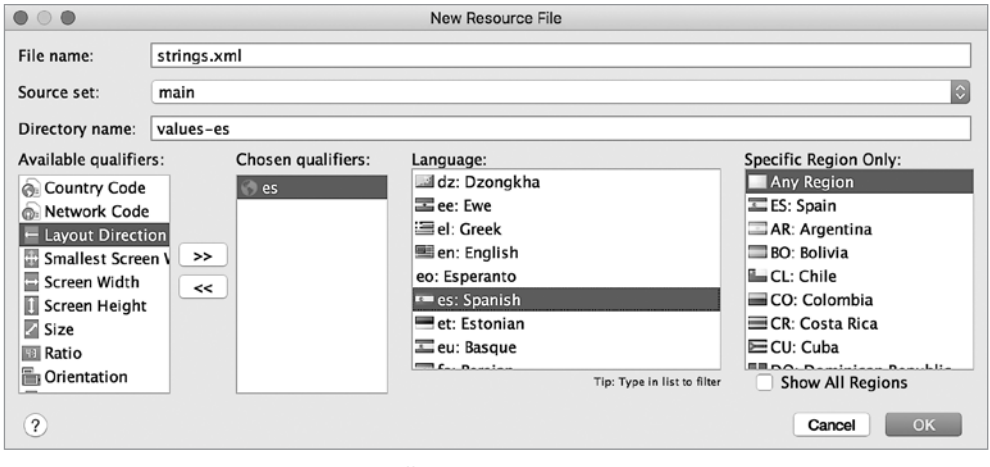


Рис. 18.2. Добавление файла строковых ресурсов с квалификаторами

Android Studio автоматически заносит в поле Directory name значение values-es. Квалификаторы языков берутся из стандарта ISO 639-1 и состоят из двух символов. Для испанского языка используется квалификатор `-es`.

Щелкните на кнопке OK. Новый файл strings.xml появился в папке res/values, за его именем следует суффикс (es). Строковые файлы группируются в режиме представления Android окна инструментов Project (рис. 18.3).

Но если вы просмотрите структуру каталогов, вы увидите, что проект теперь содержит дополнительный каталог res/values-es. Сгенерированный файл strings.xml находится в этом новом каталоге (рис. 18.4).

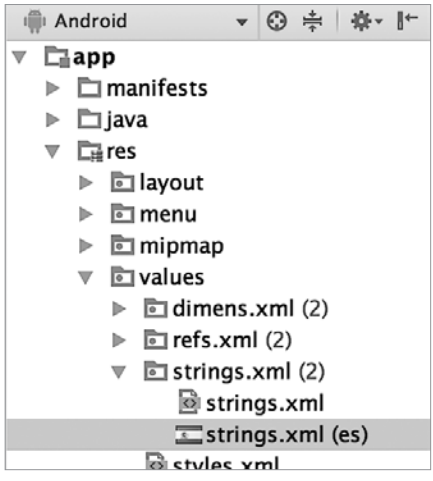


Рис. 18.3. Новый файл strings.xml в режиме представления Android

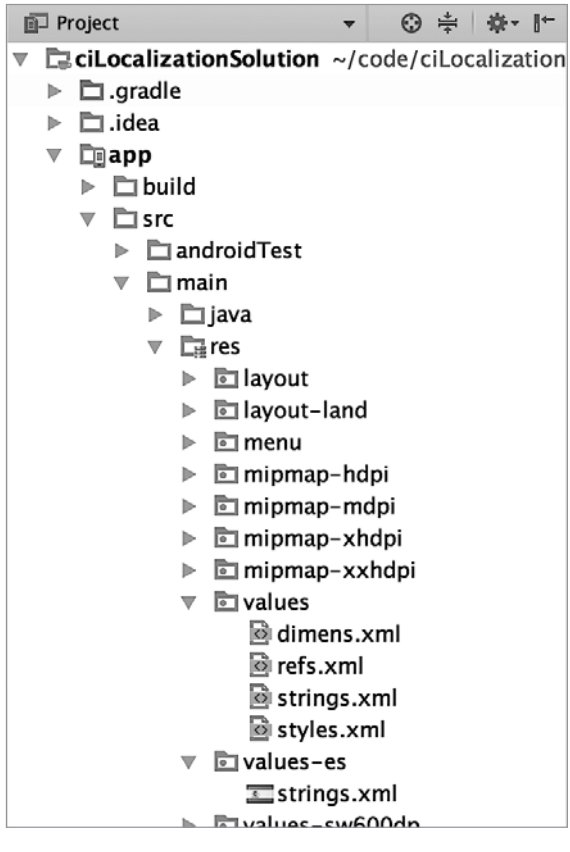


Рис. 18.4. Новый файл strings.xml в режиме представления Project

А теперь пора заняться тем, чтобы английский текст, как по волшебству, заменялся испанским. Добавьте испанские версии всех ваших строк в файл res/values-es/strings.xml. (Если вы не хотите вводить строки вручную, скопируйте содержимое из файла решения. URL-адрес приведен в разделе «Добавление значка» главы 2.)

Листинг 18.1. Добавление строковых ресурсов для испанского языка (res/values-es/strings.xml)

```
<resources>
    <string name="app_name">IntentoCriminal</string>
    <string name="crime_title_hint">Introduzca un título para el crimen.</string>
    <string name="crime_title_label">Título</string>
    <string name="crime_details_label">Detalles</string>
    <string name="crime_solved_label">Solucionado</string>
    <string name="date_picker_title">Fecha del crimen:</string>
    <string name="new_crime">Crimen Nuevo</string>
    <string name="show_subtitle">Mostrar Subtítulos</string>
    <string name="hide_subtitle">Esconder Subtítulos</string>
    <string name="subtitle_format">%1$s crímenes</string>
    <string name="crime_suspect_text">Elegir Sospechoso</string>
    <string name="crime_report_text">Enviar el Informe del Crimen</string>
    <string name="crime_report">%1$s!
        El crimen fue descubierto el %2$s. %3$s, y %4$s
    </string>
    <string name="crime_report_solved">El caso está resuelto</string>
    <string name="crime_report_unsolved">El caso no está resuelto</string>
    <string name="crime_report_no_suspect">no hay sospechoso.</string>
    <string name="crime_report_suspect">el/la sospechoso/a es %s.</string>
    <string name="crime_report_subject">IntentoCriminal Informe del Crimen</string>
    <string name="send_report">Enviar el informe del crimen a través de</string>
</resources>
```

Вот и все, что необходимо сделать для того, чтобы предоставить приложению локализованные строковые ресурсы. Чтобы убедиться в этом, выберите на своем устройстве испанский язык. (Откройте приложение Settings и найдите раздел выбора языка. В зависимости от версии Android этот раздел может называться Language and input, Language and Keyboard или как-нибудь в этом роде.)

Когда на экране появится список языковых настроек, выберите вариант Español. Выбор региона (España или Estados Unidos) роли не играет, потому что квалификатор -es подходит для обоих вариантов. (Обратите внимание: в новых версиях Android пользователь может выбрать несколько языков и назначить приоритеты. Если вы используете новое устройство, убедитесь в том, что вариант Español стоит на первом месте в списке языковых настроек.)

Запустите CriminalIntent и насладитесь видом локализованного приложения. Потом снова включите на устройстве английский язык. Найдите в лаунчере приложение Ajustes, или Configuración, и найдите раздел, в имени которого присутствует слово Idioma (язык).

Ресурсы по умолчанию

Для английского языка используется квалификатор `-en`. В пылу локализационного рвения вам, возможно, захочется переименовать существующий каталог значений в values-en/. Делать так не стоит, но представьте на минуту, что это было сделано: это гипотетическое обновление означает, что ресурсы вашего приложения для английского языка хранятся в файле strings.xml в каталоге values-en, а ресурсы для испанского языка — в файле strings.xml в каталоге values-es.

Только что обновленное приложение нормально строится. Оно также нормально работает на устройствах, на которых выбран испанский или английский язык. Но что произойдет, если на устройстве будет выбран итальянский язык? Ничего хорошего. Совсем ничего. Во время выполнения Android не найдет строковые ресурсы для текущей конфигурации. Последствия зависят от того, где именно используется ссылка на идентификатор строки.

Если ссылка на отсутствующий строковый ресурс встречается в файле макета XML, приложение выводит идентификатор ресурса (вместо осмысленной строки, определенной вами). Чтобы увидеть пример этого поведения, закомментируйте запись `crime_title_label` в файле values/strings.xml. (Чтобы легко закомментировать отдельную строку, щелкните на ней и нажмите Command+/ [Ctrl+/].)

Листинг 18.2. Исключение строкового ресурса crime_title_label для английского языка (res/values/strings.xml)

```
<resources>
    <string name="app_name">IntentoCriminal</string>
    <string name="crime_title_hint">Introduzca un título para el crimen.</string>
    <!--<string name="crime_title_label">Title</string>-->
    ...
```

(Напомним, что ссылка на `crime_title_label` встречается в fragment_crime.xml:)

```
<TextView
    style="?android:listSeparatorTextViewStyle"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/crime_title_label" />
```

Запустите CriminalIntent на устройстве, на котором выбран английский язык. Вместо текста TITLE приложение отображает идентификатор ресурса (рис. 18.5).

Еще хуже, если ссылка на отсутствующий строковый ресурс встречается в коде Java — в этом случае приложение аварийно завершится. Чтобы увидеть, как это происходит, закомментируйте `crime_report_subject`.

Листинг 18.3. Исключение строкового ресурса crime_report_subject (res/values/strings.xml)

```
<string name="crime_report_no_suspect">there is no suspect.</string>
<string name="crime_report_suspect">the suspect is %s.</string>
<!--<string name="crime_report_subject">CriminalIntent Crime Report</string>-->
```

Рис. 18.5. Использование отсутствующей английской версии строки в XML

(Напомним, что ссылка на `crime_report_subject` встречается в слушателе щелчка на кнопке отчета в файле CrimeFragment.java:)

```
mReportButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                Intent i = new Intent(Intent.ACTION_SEND);
                i.setType("text/plain");
                i.putExtra(Intent.EXTRA_TEXT, getCrimeReport());
                i.putExtra(Intent.EXTRA_SUBJECT,
                        getString(R.string.crime_report_subject));
                i = Intent.createChooser(i, getString(R.string.send_report));
                startActivity(i);
            }
        });
```

Запустите приложение и нажмите кнопку SEND CRIME REPORT: приложение аварийно завершится (рис. 18.6).

Мораль: предоставляйте версию по умолчанию для каждого из своих ресурсов. Ресурсы, находящиеся в каталогах ресурсов без квалификаторов, используются по умолчанию — если ресурс, соответствующий текущей конфигурации устройства, не найден. Если Android ищет ресурс, но не может найти версию, соответствующую текущей конфигурации устройства, работа приложения будет нарушена.

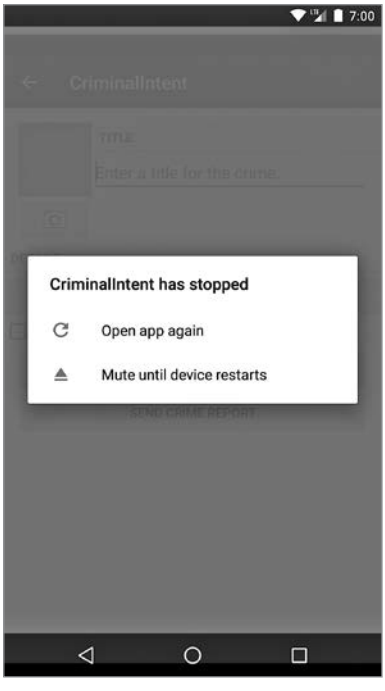


Рис. 18.6. Ошибка из-за отсутствия английской версии строки темы

(Пока оставьте `crime_title_label` и `crime_report_subject` закомментированными. Вскоре мы напомним о том, что их нужно раскомментировать.)

Отличия в плотности пикселов

В правилах назначения ресурсов по умолчанию существует исключение: экранная плотность пикселов. Как вы уже видели, каталоги drawable проекта обычно уточняются квалификаторами плотности экрана -mdpi, -xxhdpi и т. д. Однако при выборе используемого ресурса графического объекта решение Android не сводится к простому подбору экранной плотности устройства или использованию каталога без квалификатора при отсутствии совпадения.

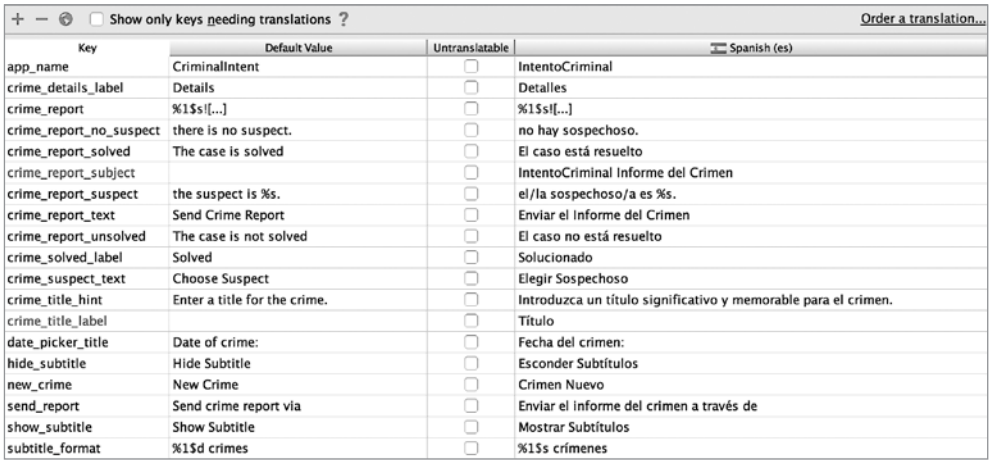
Выбор определяется сочетанием размера экрана и плотности пикселов, и Android может выбрать графический ресурс из каталога с квалификатором более низкой или высокой плотности с последующим масштабированием. Более подробная информация приведена в документации по адресу *developer.android.com/guide/practices/screens_support.html*, а пока просто учтите, что размещать ресурсы графических объектов по умолчанию в res/drawable/ не обязательно.

Проверка покрытия локализации в Translations Editor

С ростом числа поддерживаемых языков становится все труднее следить за тем, чтобы в локализации были представлены переводы всех строк для всех языков. К счастью, в Android Studio существует удобная программа Translations Editor для

централизованной работы со всеми локализациями. Чтобы запустить Translations Editor, щелкните правой кнопкой мыши на одном из файлов strings.xml в окне инструментов Project и выберите команду Open Translations Editor.

Translations Editor выводит все строки приложения и статус перевода для каждого языка, для которого ваше приложение в настоящее время предоставляет строковые значения с квалификаторами. Так как поля `crime_title_label` и `crime_report_subject` закомментированы, их имена выделяются красным цветом (рис. 18.7).



Show only keys needing translations ? Order a translation...

Key	Default Value	Untranslatable	Spanish (es)
app_name	CriminalIntent		IntentoCriminal
crime_details_label	Details		Detalles
crime_report	%1$s![...]		%1$s![...]
crime_report_no_suspect	there is no suspect.		no hay sospechoso.
crime_report_solved	The case is solved		El caso está resuelto
crime_report_subject			IntentoCriminal Informe del Crimen
crime_report_suspect	the suspect is %s.		el/la sospechoso/a es %s.
crime_report_text	Send Crime Report		Enviar el Informe del Crimen
crime_report_unsolved	The case is not solved		El caso no está resuelto
crime_solved_label	Solved		Solucionado
crime_suspect_text	Choose Suspect		Elegir Sospechoso
crime_title_hint	Enter a title for the crime.		Introduzca un título significativo y memorable para el crimen.
crime_title_label			Título
date_picker_title	Date of crime:		Fecha del crimen:
hide_subtitle	Hide Subtitle		Esconder Subtítulos
new_crime	New Crime		Crimen Nuevo
send_report	Send crime report via		Enviar el informe del crimen a través de
show_subtitle	Show Subtitle		Mostrar Subtítulos
subtitle_format	%1$d crimes		%1$s crímenes

Рис. 18.7. Проверка покрытия локализации в Translation Editor

(Прежде чем двигаться далее, раскомментируйте `crime_title_label` и `crime_report_subject`. Чтобы раскомментировать отдельную строку, щелкните на этой строке и снова нажмите Command+/ [Ctrl+/].)

Региональная локализация

Каталогу ресурсов также можно назначить квалификатор «язык-регион», который позволяет еще точнее определять ресурсы. Например, квалификатор для испанского языка, на котором говорят в Испании, имеет вид `-es-rES`, где `r` — признак квалификатора региона, а ES — код Испании согласно стандарту ISO 3166-1-alpha-2. Конфигурационные квалификаторы не учитывают регистр символов, но в них рекомендуется соблюдать схему, принятую в Android: код языка записывается в нижнем регистре, а код региона записывается символами верхнего регистра с префиксом `r` (в нижнем регистре).

Учтите, что квалификатор «язык-регион» (такой, как `-es-rES`) может выглядеть как объединение двух разных конфигурационных квалификаторов, но в действительности квалификатор только один. Регион сам по себе не является действительным квалификатором.

На рис. 18.8 представлена блок-схема разрешения локализованных ресурсов с учетом версии Android.

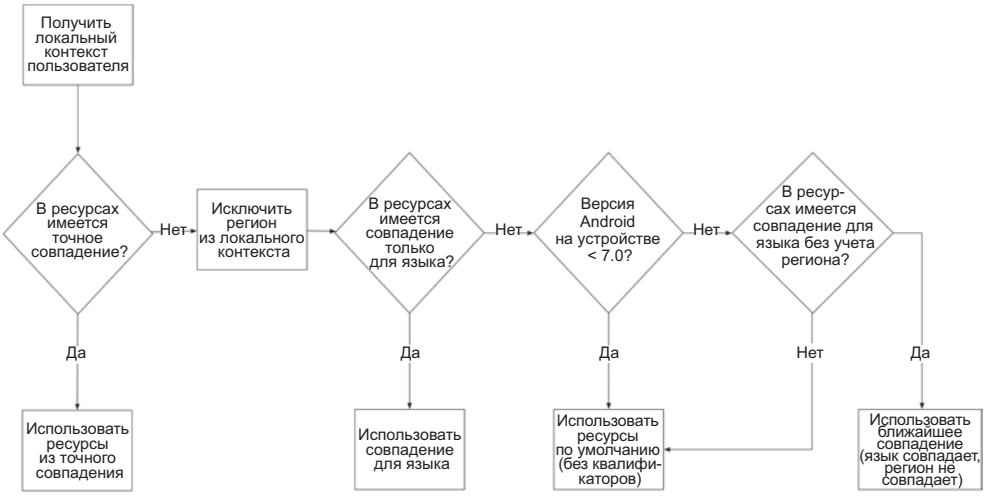


Рис. 18.8. Правила разрешения локализованных ресурсов

Ресурс, уточненный как локальным контекстом, так и регионом, может совпасть с локальным контекстом пользователя в двух вариантах. Точное совпадение происходит в том случае, если квалификаторы и языка, и региона соответствуют локальному контексту пользователя. Если точное совпадение не найдено, система отделяет квалификатор региона и ищет точное совпадение только для языка.

На устройствах с версиями Android, предшествующими Nougat, при отсутствии совпадения для языка используются ресурсы по умолчанию (без квалификаторов). В Nougat поддержка локальных контекстов была усовершенствована: их количество увеличилось, а пользователь получил возможность выбрать несколько локальных контекстов в конфигурации устройства. Система также использует более разумную стратегию разрешения ресурсов, чтобы правильный язык использовался как можно чаще, даже если на устройстве не совпадает регион или язык без квалификатора. Если на устройстве с Nougat точное совпадение не найдено, и не найдено совпадение только для языка, система ищет ресурс с тем же языком, но другим регионом, и использует лучшее совпадение для ресурсов, удовлетворяющих этим критериям.

Рассмотрим пример. Допустим, на устройстве выбран испанский язык и регион Чили (рис. 18.9). Приложение на устройстве содержит файлы strings.xml с ресурсами на испанском языке для Испании и Мексики (в каталогах values-es-rES и values-es-rMX). Каталог values для ресурсов по умолчанию содержит английский файл strings.xml. Если на вашем устройстве работает Android версии до Nougat, вы увидите англоязычный текст из каталога values. Но если на устройстве установлена версия Nougat, результат выглядит более логично: вы увидите ресурсы из файла values-es-rMX/strings.xml — то есть испанский текст, хотя и не адаптированный для Чили.

Пример немного искусственный, но он подчеркивает один важный момент: строки следует предоставлять по возможности в общем контексте, используя каталоги с квалификаторами языка и региона только по необходимости. Вместо того, чтобы сопровождать все испаноязычные строки в трех каталогах, уточненных квалифи-

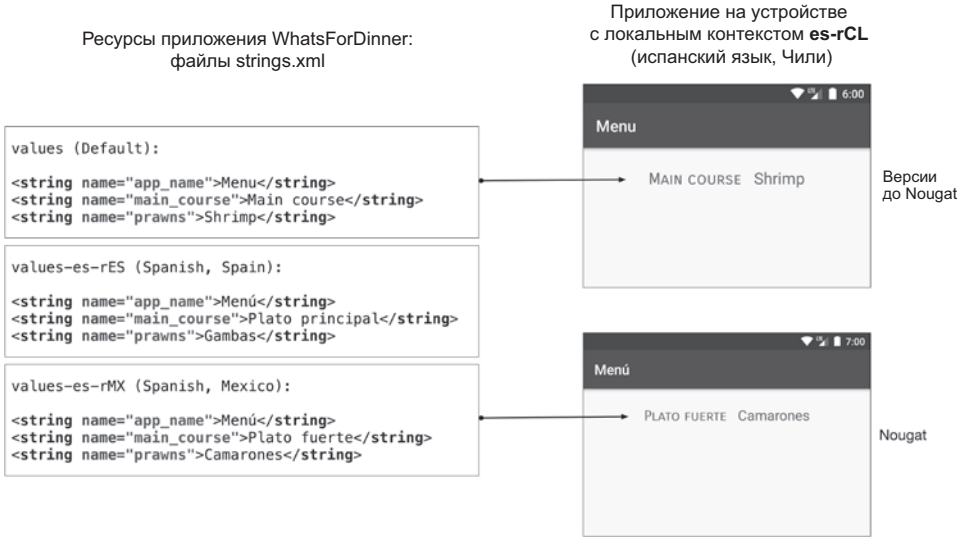


Рис. 18.9. Пример разрешения локализованных ресурсов (до и после Nougat)

каторами регионов, в этом приложении лучше было бы хранить строки в каталоге с квалификатором языка values-es и определять строки с квалификаторами регионов только для слов и выражений, различающихся в региональных диалектах. Такое решение не только упрощает сопровождение строк для программиста, но и упрощает разрешение ресурсов на устройствах с разными версиями Android — как с версией Nougat, так и до нее.

Тестирование нестандартных локальных контекстов

Разные устройства и разные версии Android поддерживают разные локальные контексты. Возможно, в какой-то ситуации вам потребуется предоставить строки и другие ресурсы для локального контекста, недоступного на тестовом устройстве. Не огорчайтесь, вы можете воспользоваться поддержкой нестандартных локальных контекстов в эмуляторе для создания и применения локального контекста, не поддерживаемого образом системы. Эмулятор моделирует конфигурацию времени выполнения, которая включает комбинацию «язык/регион» и позволяет протестировать поведение вашего приложения в этой конфигурации.

Поддержка нестандартных локальных контекстов включена в эмулятор. Откройте в эмуляторе экран App Launcher и щелкните на значке Custom Locale. Запущенное приложение позволяет просматривать существующие локальные контексты, добавлять новые и применять нестандартные локальные контексты при тестировании (рис. 18.10).

Если выбрать нестандартный локальный контекст, не поддерживаемый образом системы, в интерфейсе системы будет отображаться язык по умолчанию. Тем не менее в вашем приложении разрешение ресурсов будет осуществляться с учетом выбранного вами нестандартного локального контекста.

Рис. 18.10. Приложение Custom Locale

Конфигурационные квалификаторы

Вы уже видели и использовали конфигурационные квалификаторы для предоставления альтернативных ресурсов: язык (например, values-es/), ориентацию экрана (например, layout-land/), плотность пикселов (например, drawable-mdpi/) и размер экрана (например, layout-sw600dp).

Android поддерживает конфигурационные квалификаторы для выбора ресурсов в зависимости от следующих аспектов конфигурации устройства:

1. Код страны для мобильной связи (MCC), за которым может следовать код сети мобильной связи (MNC).
2. Код языка, за которым может следовать код региона.
3. Направление макета.
4. Минимальная ширина.
5. Доступная ширина.
6. Доступная высота.
7. Размер экрана.
8. Пропорции экрана.

9. Круглый экран (API уровня 23 и выше).
10. Ориентация экрана.
11. Режим пользовательского интерфейса.
12. Ночной режим.
13. Плотность экрана (dpi).
14. Тип сенсорного экрана.
15. Доступность клавиатуры.
16. Основной метод ввода текста.
17. Доступность клавиш перемещения.
18. Основной несенсорный метод перемещения.
19. Уровень API.

Описания этих характеристик и примеры конкретных конфигурационных квалификаторов можно найти по адресу *developer.android.com/guide/topics/resources/providing-resources.html#AlternativeResources*.

Не все квалификаторы поддерживаются ранними версиями Android. К счастью, система неявно добавляет квалификатор версии платформы к квалификаторам, добавленным после Android 1.0. Таким образом, если вы, например, используете квалификатор `round`, Android автоматически добавляет квалификатор `v23`, потому что квалификатор круглого экрана появился в API уровня 23. Это означает, что при добавлении ресурсов с квалификаторами новых устройств вам не придется беспокоиться о возможных проблемах со старыми устройствами.

Приоритеты альтернативных ресурсов

При таком количестве конфигурационных квалификаторов для определения ресурсов могут возникнуть ситуации, при которых к конфигурации устройства могут подходить сразу несколько альтернативных ресурсов. В таких ситуациях квалификаторам назначаются приоритеты в соответствии с их порядком в приведенном выше списке.

Чтобы понять, как работает система приоритетов, добавим в CriminalIntent альтернативный ресурс — более длинную англоязычную версию строкового ресурса `crime_title_hint`, которая должна отображаться в том случае, если ширина экрана в текущей конфигурации не менее `600dp`. Ресурс `crime_title_hint` отображается в текстовом поле описания до того, как пользователь введет какой-либо текст. Если приложение CriminalIntent выполняется на экране с шириной не менее 600dp (например, на планшете или в альбомном режиме на устройстве с меньшим экраном), с таким изменением в поле описания будет выводиться более содержательная и осмысленная подсказка.

Создайте новый файл строковых ресурсов и поместите его в каталог `values-w600dp` (квалификатор `-w600dp` подойдет для любого устройства, у которого текущая доступная ширина экрана равна 600dp и более; учтите, что квалификатор может

подходить для устройства в альбомной, но не в книжной ориентации). Создайте файл со строковыми ресурсами (см. раздел «Локализация ресурсов»), но выберите в списке Available qualifiers значение Screen Width и щелкните на кнопке >>, чтобы переместить значение Screen Width в список Chosen qualifiers. В остальных полях введите значения, показанные на рис. 18.11.

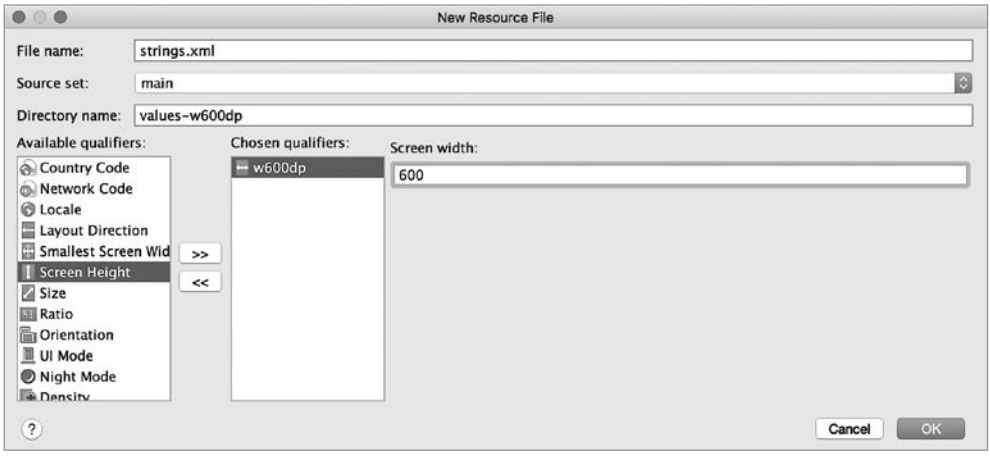


Рис. 18.11. Добавление строк для экрана большей ширины

Добавьте длинную строку `crime_title_hint` в файл values-w600dp/strings.xml.

Листинг 18.4. Создание альтернативных строковых ресурсов для широкого экрана (values-w600dp/strings.xml)

```
<resources>
    <string name="crime_title_hint">
        Enter a meaningful, memorable title for the crime.
    </string>
</resources>
```

Единственный строковый ресурс, который используется только на широких экранах, — `crime_title_hint`. Это единственная строка, хранящаяся в каталоге values-w600dp. Альтернативы для строковых ресурсов (и других ресурсов-значений) определяются на уровне отдельных строк, поэтому дублировать одинаковые строки не нужно. Дублирование только создаст проблемы с будущим сопровождением проекта.

Итак, теперь у вас имеются три версии `crime_title_hint`: версия по умолчанию в файле values/strings.xml, испанская альтернативная версия в файле values-es/strings.xml и альтернативная версия для широкого экрана в файле values-w600dp/strings.xml.

Пока на устройстве остается выбранным испанский язык, запустите приложение CriminalIntent и поверните устройство в альбомную ориентацию (рис. 18.12). Альтернативная версия для испанского языка обладает более высоким приоритетом,

Рис. 18.12. В Android язык обладает более высоким приоритетом, чем доступная ширина экрана

поэтому вместо версии из values-w600dp/strings.xml вы увидите строку из values-es/strings.xml.

Верните на устройстве английский язык. Снова запустите приложение и убедитесь в том, что на экране, как и ожидалось, выводится альтернативная строка для широкого экрана.

Множественные квалификаторы

Вероятно, вы заметили, что диалоговое окно New Resource File содержит много возможных квалификаторов. Каталогу ресурсов можно назначить более одного квалификатора. В таком случае квалификаторы перечисляются в порядке приоритетов. Таким образом, values-es-w600dp является действительным именем каталога, а values-w600dp-es — нет. (Если вы используете диалоговое окно New Resource File, оно автоматически настраивает имя каталога.)

Создайте каталог для строки на испанском языке, предназначенной для широкого экрана. Каталогом должен называться values-es-w600dp, и в нем должен присутствовать файл strings.xml. Добавьте строковый ресурс `crime_title_hint` в файл values-es-w600dp/strings.xml (листинг 18.5).

Листинг 18.5. Создание строкового ресурса для испанского языка на широком экране (values-es-w600dp/strings.xml)

```
<resources>
    <string name="crime_title_hint">
        Introduzca un título significativo y memorable para el crimen.
    </string>
</resources>
```

Выберите испанский язык, запустите приложение CriminalIntent и убедитесь в том, что новый альтернативный ресурс отображается в положенном месте (рис. 18.13).

Рис. 18.13. Испаноязычный строковый ресурс для широкого экрана

Поиск наиболее подходящих ресурсов

Каким образом Android определяет, какая версия `crime_title_hint` должна отображаться при каждом запуске? Возьмем четыре альтернативных варианта строкового ресурса с именем `crime_title_hint` на устройстве Nexus 5x с испанским языком и доступной шириной экрана более 600dp:

Конфигурация устройства	Каталог для crime_title_hint
Язык: es (испанский)	values
Доступная высота: 411dp	values-es
Доступная ширина: 731dp	values-es-w600dp
(и т.д.)	values-w600dp

Исключение несовместимых каталогов

Поиск наиболее подходящего ресурса Android начинает с исключения всех каталогов ресурсов, несовместимых с текущей конфигурацией.

Все четыре варианта совместимы с текущей конфигурацией. (Если повернуть устройство в книжную ориентацию, то доступная ширина станет равной 411dp; в этом случае каталоги values-w600dp/ и values-es-w600dp/ станут несовместимыми и будут исключены.)

Перебор по таблице приоритетов

После исключения несовместимых каталогов ресурсов Android начинает перебирать таблицу приоритетов, приведенную в разделе «Конфигурационные квалификаторы», начиная с самого приоритетного квалификатора — MCC. Если существует каталог ресурсов с квалификатором MCC, то все каталоги ресурсов,

не имеющие квалификатора MCC, будут исключены. Если после этого осталось более одного подходящего каталога, то Android рассматривает следующий по приоритету квалификатор. Перебор продолжается до тех пор, пока не останется только один каталог.

В нашем примере каталогов с квалификатором MCC нет, поэтому ни один каталог не исключается, и Android переходит по списку к квалификатору языка. Квалификаторы языка присутствуют в двух каталогах (values-es/ и values-es-w600dp/). В одном каталоге (values-w600dp/) этого квалификатора нет, поэтому он исключается:

Конфигурация устройства	Каталог для crime_title_hint
Язык: es (испанский)	values
Доступная высота: 411dp	values-es
Доступная ширина: 731dp	values-es-w600dp
(и т.д.)	~~values-w600dp~~ (без привязки к языку)

Кандидатов все еще слишком много, поэтому Android продолжает двигаться по списку квалификаторов. При достижении квалификатора доступной ширины Android находит один каталог с квалификатором доступной ширины и два каталога без него. Каталоги values и values-es/ исключаются, остается только values-es-w600dp/:

Конфигурация устройства	Каталог для crime_title_hint
Язык: es (испанский)	~~values~~ (без привязки к ширине)
Доступная высота: 411dp	~~values-es~~ (без привязки к ширине)
Доступная ширина: 731dp	**values-es-w600dp** (лучшее совпадение)
(и т. д.)	~~values-w600dp~~ (без привязки к языку)

Итак, Android использует ресурс из каталога values-es-w600dp/.

Тестирование альтернативных ресурсов

Приложение следует протестировать на разных конфигурациях устройств, чтобы увидеть, как ваши макеты и другие ресурсы смотрятся в этих конфигурациях. Для тестирования могут использоваться как физические, так и виртуальные устройства. Также можно воспользоваться графическим конструктором.

В графическом конструкторе предусмотрено несколько способов предварительного просмотра макета в разных конфигурациях: для разных размеров экрана, типов устройств, уровней API, языков и т. д.

Чтобы ознакомиться с вариантами предварительного просмотра, откройте файл fragment_crime.xml в графическом конструкторе и опробуйте элементы управления, изображенные на рис. 18.14.

Рис. 18.14. Предварительный просмотр в графическом конструкторе

Чтобы убедиться в том, что вы включили все необходимые ресурсы по умолчанию, включите в конфигурации устройства язык, для которого не определены локализованные ресурсы. Запустите приложение и поработайте с ним. Откройте все представления, попробуйте повернуть устройство. Если в приложении произойдет сбой, проверьте панель Logcat и поищите на ней сообщение «Ресурс не найден», которое поможет найти отсутствующий ресурс по умолчанию. Также следите за ошибками, не приводящими к аварийному завершению, как в случае с выводом идентификаторов ресурсов вместо строк (см. ранее в этой главе).

Прежде чем переходить к следующей главе, не забудьте вернуть на устройстве английский язык.

Поздравляем! Теперь приложение CriminalIntent стало доступным как для англоязычных, так и для испаноязычных пользователей. Преступления регистрируются. Дела раскрываются. И все это с максимумом удобств — пользователь общается с приложением на родном языке. Для поддержки новых языков достаточно добавить новые файлы со строками в каталог с соответствующими квалификаторами.

Упражнение. Локализация дат

Возможно, вы заметили, что независимо от выбранного на устройстве локального контекста даты в CriminalIntent всегда выводятся в формате, принятом в США: месяц перед днем. Продолжите процесс локализации и обеспечьте форматирование дат в соответствии с конфигурацией локального контекста. Это проще, чем может показаться на первый взгляд.

Обратитесь к документации разработчика — а именно к описанию класса `DateFormat` в фреймворке Android. Класс `DateFormat` предоставляет функции форматирования даты/времени по правилам текущего локального контекста. Для управления выводом используются конфигурационные константы, встроенные в `DateFormat`.

19 Доступность

В этой главе мы займемся повышением *доступности* приложения CriminalIntent. Доступное приложение может использоваться кем угодно независимо от ограничений по зрению, слуху или двигательным функциям. Ограничения могут быть как постоянными, так и временными или ситуационными: например, расширенные зрачки после обследования у окулиста могут затруднить чтение. Жирные руки во время приготовления пищи отбивают желание прикасаться к экрану. А если вы находитесь на концерте, громкая музыка заглушает все звуки, производимые вашим устройством. Чем доступнее приложение, тем приятнее с ним работать.

Обеспечение полной доступности приложения — слишком грандиозная задача, но это не повод даже не пытаться. В этой главе мы постараемся сделать приложение CriminalIntent более доступным для пользователей с дефектами зрения. Эта область станет хорошей отправной точкой для изучения проблем доступности и проектирования приложений с учетом доступности.

Изменения этой главы не затронут внешнего вида приложения. Мы поступим иначе: упростим работу с контентом с использованием TalkBack.

TalkBack

TalkBack — экранный диктор для Android, разработанный компанией Google. Он произносит вслух описание содержимого экрана, которое зависит от того, что делает пользователь.

TalkBack представляет собой *сервис доступности* — специальный компонент, который может читать информацию с экрана независимо от того, в каком приложении вы работаете. Любой желающий может написать собственный сервис доступности, но TalkBack является самым популярным решением.

Для использования TalkBack понадобится физическое устройство на базе Android. (К сожалению, в эмуляторе TalkBack не поддерживается.) Убедитесь в том, что на устройстве не отключен звук — но возможно, вам стоит надеть наушники, потому что с TalkBack устройство становится чрезвычайно «разговорчивым».

Чтобы включить TalkBack, запустите приложение Settings и откройте категорию Accessibility. Выберите раздел TalkBack под заголовком Services. Переведите переключатель в верхней части экрана в правое положение, чтобы включить TalkBack (рис. 19.1).

Android отображает диалоговое окно, в котором система запрашивает разрешение на доступ к определенной информации, например к отслеживанию действий пользователя и изменению некоторых настроек (скажем, включению функции Explore by Touch (рис. 19.2)). Нажмите кнопку OK.

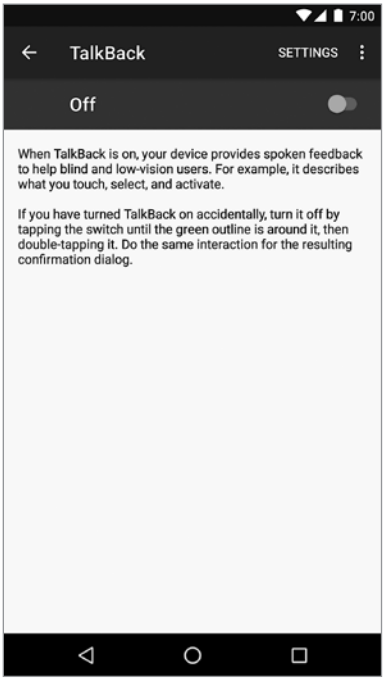


Рис. 19.1. Экран настроек TalkBack

Рис. 19.2. Предоставление разрешений TalkBack

Слева от переключателя в верхней части экрана появляется надпись On. (Если вы впервые используете TalkBack на устройстве, откроется обучающее руководство.) Выйдите из меню при помощи кнопки Up на панели инструментов.

Изменения в работе устройства видны практически сразу. У кнопки Up появится зеленый контур (рис. 19.3), а устройство подскажет: «Кнопка перемещения Up. Выполните двойное касание, чтобы активизировать».

Хотя на устройствах Android обычно применяется термин «нажатие» (press), TalkBack использует термин «касание» (tap). Кроме того, TalkBack использует двойные касания, нечасто встречающиеся в Android.

Зеленый контур показывает, какой элемент пользовательского интерфейса имеет фокус доступности. В любой момент времени фокус доступности может быть только у одного элемента. При получении элементом фокуса доступности TalkBack предоставляет информацию об этом элементе.

При включенной функции TalkBack одно нажатие (или «касание») передает элементу фокус доступности. Двойное касание в любом месте экрана активизирует элемент. Таким образом, когда фокус находится у кнопки Up, двойное касание

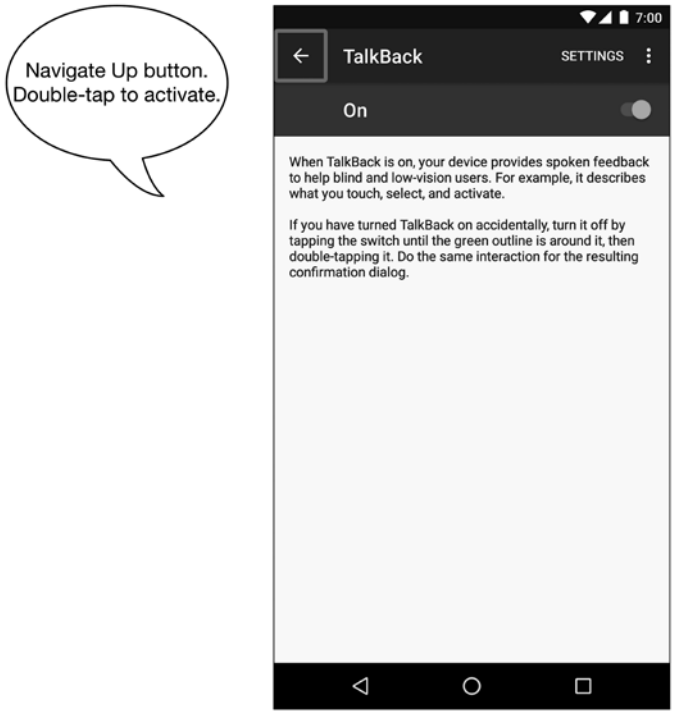


Рис. 19.3. Устройство с включенной функцией TalkBack

активизирует кнопку Up, когда фокус находится у флажка — изменяет его состояние и т. д. (Кроме того, если ваше устройство заблокировано, вы сможете снять блокировку, коснувшись изображения замка и выполнив двойное касание в любой точке экрана.)

Explore by Touch

Включив TalkBack, вы также включили функцию Explore by Touch в TalkBack. Это означает, что устройство будет проговаривать информацию об элементе сразу же после нажатия. (Предполагается, что для нажатого элемента задана информация, которую TalkBack может зачитать, — вскоре мы расскажем об этом подробнее.)

Оставьте у кнопки Up выделение и фокус доступности. Выполните двойное касание в любой точке экрана. Устройство возвращается к меню доступности, а TalkBack произносит его название.

Для виджетов инфраструктуры Android — `Toolbar`, `RecyclerView`, `ListView` и `Button` — реализована встроенная поддержка TalkBack. Старайтесь по возможности использовать виджеты инфраструктуры, чтобы пользоваться результатами уже выполненной работы в области доступности. Также возможно обеспечить правильную реакцию на события доступности для нестандартных виджетов, но эта тема выходит за рамки книги.

Чтобы прокрутить список, приложите два пальца к экрану и проведите вверх или вниз. В зависимости от длины списка вы будете слышать звуковые сигналы, передающие метаинформацию о текущем взаимодействии.

Линейная навигация смахиванием

Представьте, что это такое: впервые исследовать приложение «на ощупь». Вы еще не знаете, что где находится. Что, если единственный способ ознакомиться со структурой экрана — нажимать все подряд, пока вы не попадете на элемент, который TalkBack сможет прочитать вслух? Возможно, какие-то элементы будут нажаты повторно, или, что еще хуже, какие-то элементы будут пропущены.

К счастью, существует способ линейного изучения пользовательского интерфейса. Собственно, это более распространенный вариант использования TalkBack: смахивание направо переводит фокус доступности к следующему элементу на экране, а смахивание налево — к предыдущему элементу. Пользователь последовательно перебирает элементы вместо того, чтобы наугад тыкать в экран в надежде наткнуться на что-то осмысленное.

Попробуйте сами. Запустите приложение CriminalIntent и перейдите к экрану со списком преступлений. Нажмите на заголовке панели инструментов, чтобы передать ей фокус доступности. Устройство зачитывает название приложения «CriminalIntent» (рис. 19.4).

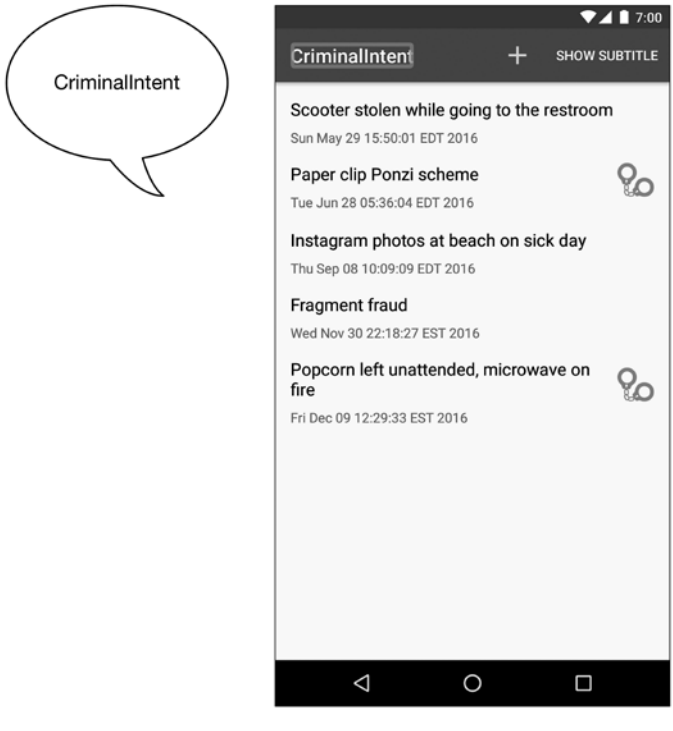


Рис. 19.4. Выбор заголовка

Теперь проведите направо. Доступность переходит к кнопке создания преступления на панели инструментов. TalkBack объявляет: «Новое преступление. Выполните двойное касание, чтобы активизировать. Выполните двойное касание и удерживайте, чтобы выполнить долгое нажатие». Для инфраструктурных виджетов, таких как команды меню и кнопки, TalkBack зачитывает видимый текст, отображаемый в виджете по умолчанию. Однако команда меню — всего лишь значок, у которого нет никакого видимого текста. В таких случаях TalkBack ищет другую доступную информацию. В разметке XML меню был задан заголовок, он и будет прочитан TalkBack. Кроме того, TalkBack предоставляет подробную информацию о действиях, которые пользователь может выполнить с виджетом, а иногда и информацию о типе виджета.

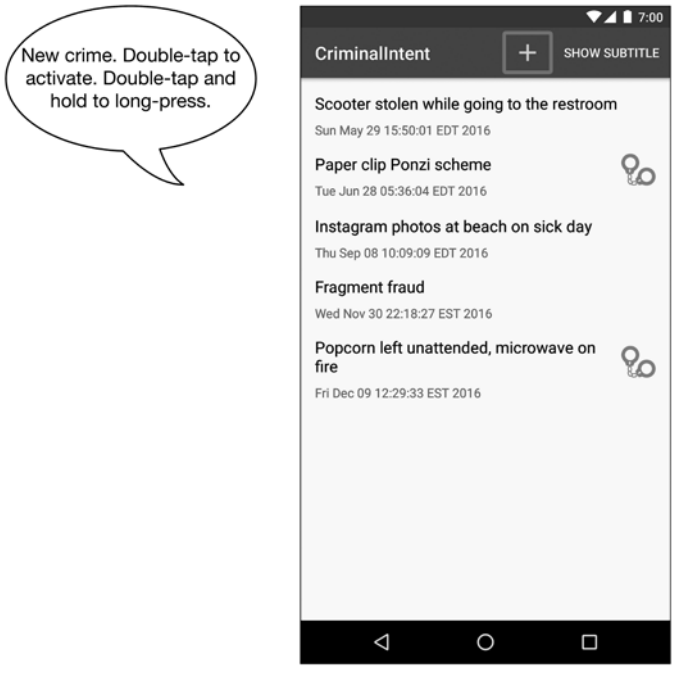


Рис. 19.5. Выбор кнопки создания преступления

Снова проведите направо; TalkBack зачитывает информацию о кнопке меню SHOW SUBTITLE. Проведите направо в третий раз; фокус доступности перейдет к первому преступлению в списке. Проведите налево — фокус опять вернется к кнопке меню SHOW SUBTITLE. Android старается передавать фокус доступности в осмысленном порядке.

Чтение не-текстовых элементов

Теперь нажмите кнопку создания преступления на панели инструментов. Фокус доступности передастся новому преступлению, а TalkBack снова объявит информацию о кнопке. Пока выбрана кнопка создания преступления, выполните двой-

ное касание в любой точке экрана, чтобы открыть экран с подробной информацией о преступлении.

Добавление описаний контента

На экране с подробной информацией о преступлении нажмите кнопку создания фотографии, чтобы передать ей фокус доступности (рис. 19.6). TalkBack объявляет: «Кнопка без подписи. Выполните двойное касание, чтобы активизировать». (Возможно, на вашем устройстве текст будет немного другим — в зависимости от используемой версии Android.)

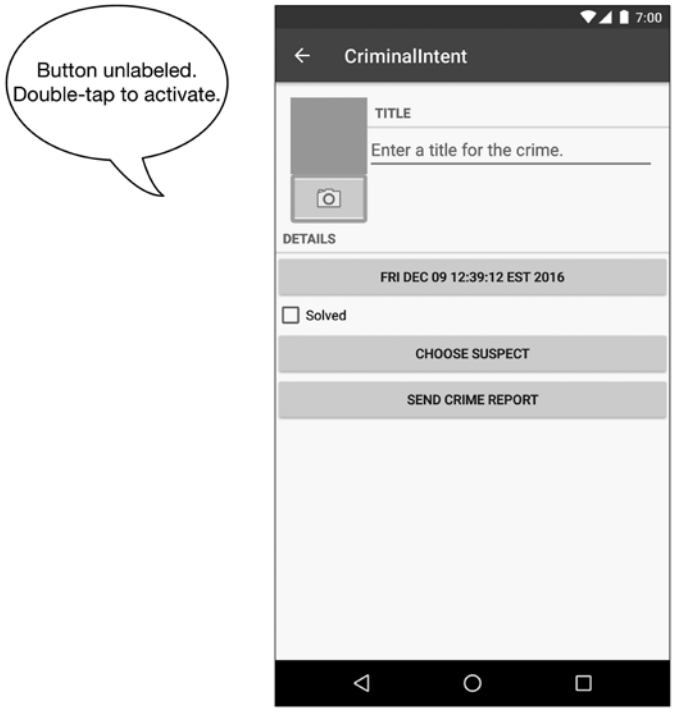


Рис. 19.6. Кнопка создания фотографии

На кнопке управления камерой текста нет, поэтому TalkBack описывает ее, как может. Но несмотря на все старания, эта информация не поможет пользователю с ослабленным зрением.

К счастью, проблема легко решается. Вы сами указываете информацию, которую должен зачитывать диктор TalkBack; для этого следует добавить описание контента к виджету `ImageButton`. *Описание контента* — текст, который описывает виджет, и зачитывается TalkBack. (Заодно мы добавим описание содержимого для виджета `ImageView`, в котором выводится изображение.)

Описание контента можно задать в файле макета XML — необходимое значение присваивается атрибуту `android:contentDescription`. Собственно, именно это мы сейчас и сделаем. Также описание контента может быть задано в коде инициализации вызовами *someView*`.setContentDescription(someString)`; этот способ будет применен позднее в этой главе.

Текст описания должен быть осмысленным, но не слишком длинным. Помните: пользователи TalkBack слушают аудио, а время прослушивания пропорционально длине текста. Скорость воспроизведения в TalkBack можно повысить, но и в этом случае лучше не включать избыточную информацию и не тратить время пользователя. Например, если вы задаете описание для встроенного виджета, не стоит включать информацию о типе виджета (например, «кнопка»), потому что TalkBack уже знает и включает эту информацию.

Сначала небольшая подготовительная работа. Добавьте следующие строки в файл strings.xml без квалификаторов.

Листинг 19.1. Добавление строк с описаниями контента (res/values/strings.xml)

```
<resources>
    ...
    <string name="crime_details_label">Details</string>
    <string name="crime_solved_label">Solved</string>
    <string name="crime_photo_button_description">Take photo of crime scene</string>
    <string name="crime_photo_no_image_description">
        Crime scene photo (not set)
    </string>
    <string name="crime_photo_image_description">Crime scene photo (set)</string>
    ...
</resources>
```

Теперь откройте файл res/layout/fragment_crime.xml и задайте описания контента для `ImageButton` и `ImageView`.

Листинг 19.2. Назначение описаний контента для ImageView и ImageButton (res/layout/fragment_crime.xml)

```
<ImageView
    android:id="@+id/crime_photo"
    android:layout_width="80dp"
    android:layout_height="80dp"
    android:background="@android:color/darker_gray"
    android:cropToPadding="true"
    android:scaleType="centerInside"
    android:contentDescription="@string/crime_photo_no_image_description" />

<ImageButton
    android:id="@+id/crime_camera"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:src="@android:drawable/ic_menu_camera"
    android:contentDescription="@string/crime_photo_button_description" />
```

Запустите приложение CriminalIntent и нажмите кнопку управления камерой. TalkBack подсказывает: «Сделать фотографию места преступления. Выполните двойное касание, чтобы активизировать». Эта информация намного полезнее, чем «Кнопка без подписи».

Теперь нажмите на поле для хранения фотографии (в нашем случае оно содержит серый прямоугольник). Возможно, вы ожидаете, что фокус доступности переместится к `ImageView`, но зеленый конур появится вокруг всего фрагмента, а TalkBack выдаст общую информацию о фрагменте вместо `ImageView`. В чем дело?

Включение фокусировки виджета

Проблема в том, что виджет `ImageView` не зарегистрирован для получения фокуса. Некоторые виджеты (такие как `Button` и `CheckBox`) могут получать фокус по умолчанию. Другие виджеты, такие как `TextView` и `ImageView`, фокус не получают.

Чтобы разрешить получение фокуса представлением, следует задать атрибуту `android:focusable` значение `true`, или добавить слушателя щелчка.

Разрешите получение фокуса виджетом `ImageView` с фотографией преступления, явно присвоив `focusable` значение `true` в разметке XML макета.

Рис. 19.7. Получение фокуса виджетом ImageView

Листинг 19.3. Разрешение получения фокуса виджетом ImageView (res/layout/fragment_crime.xml)

```
<ImageView
  android:id="@+id/crime_photo"
  ...
  android:contentDescription="@string/crime_photo_no_image_description"
  android:focusable="true" />
```

Снова запустите CriminalIntent и нажмите на фотографии. Виджет `ImageView` получает фокус, и TalkBack объявляет: «Фотография места преступления (не задан)» (рис. 19.7).

Создание сопоставимого опыта взаимодействия

Описание контента должно задаваться для любого UI-виджета, который предоставляет информацию пользователю, но не в текстовом виде (например, в форме графики). Если виджет не дает ничего, кроме украшения, прикажите TalkBack игнорировать его, задав в качестве описания контента `null`.

Возможно, вы думаете: «Если пользователь все равно не видит, зачем ему знать, есть изображение или нет?» Однако вы не должны ничего предполагать относительно пользователей. Что еще важнее, вы должны проследить за тем, чтобы пользователь с ослабленным зрением получал столько же информации и функциональности, что и пользователь с нормальным зрением. Общий опыт взаимодействия и последовательность операций могут быть другими, но все пользователи должны иметь доступ к единой функциональности приложения.

Качественно спроектированное доступное приложение не обязано зачитывать описание каждого объекта на экране. Вместо этого оно должно стараться обеспечить сопоставимый опыт взаимодействия. Какая информация и контекст действительно важны?

Непосредственно сейчас опыт взаимодействия, связанный с фотографией преступления, ограничен. TalkBack всегда сообщает, что изображение не задано, даже если оно будет задано. Чтобы убедиться в этом, нажмите на кнопке камеры, а потом выполните двойное касание на экране, чтобы активизировать ее. Запускается приложение камеры, и TalkBack объявляет: «Камера». Сохраните фотографию: нажмите кнопку и выполните двойное касание в любой точке экрана.

Подтвердите получение фотографии. (Конкретная последовательность действий зависит от того, какое приложение камеры вы используете, но помните: чтобы активизировать ее, следует выделить кнопку и выполнить двойное касание в любой точке.) Открывается экран с подробной информацией о преступлении и обновленной фотографией. Нажмите на снимке, чтобы передать ему фокус доступности. TalkBack объявляет: «Фотография места преступления (не задана)».

Чтобы предоставить более актуальную информацию для пользователей TalkBack, динамически назначьте описание контента `ImageView` в `updatePhotoView()`.

Листинг 19.4. Динамическое назначение описания контента (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
    ...
    private void updatePhotoView() {
        if (mPhotoFile == null || !mPhotoFile.exists()) {
            mPhotoView.setImageDrawable(null);
            mPhotoView.setContentDescription(
                    getString(R.string.crime_photo_no_image_description));
        } else {
            ...
            mPhotoView.setImageBitmap(bitmap);
            mPhotoView.setContentDescription(
                    getString(R.string.crime_photo_image_description));
        }
    }
}
```

Теперь при обновлении представления с фотографией `updatePhotoView()` будет обновляться и описание контента. Если поле `mPhotoFile` пусто, то назначается описание контента, сообщающее об отсутствии фотографии. В противном случае назначается описание, сообщающее о том, что фотография есть.

Рис. 19.8. Виджет ImageView с возможностью получения фокуса и динамическим описанием

Запустите приложение CriminalIntent. Просмотрите экран с подробным описанием преступления, в котором была добавлена фотография. Нажмите на фотографии места преступления (рис. 19.8). TalkBack объявляет: «Фотография места преступления (задана)».

Использование надписей для передачи контекста

Прежде чем двигаться дальше, введите описание нового преступления. Нажмите на поле `EditText`. TalkBack объявляет: «Текстовое поле. Введите описание преступления» (рис. 19.9).

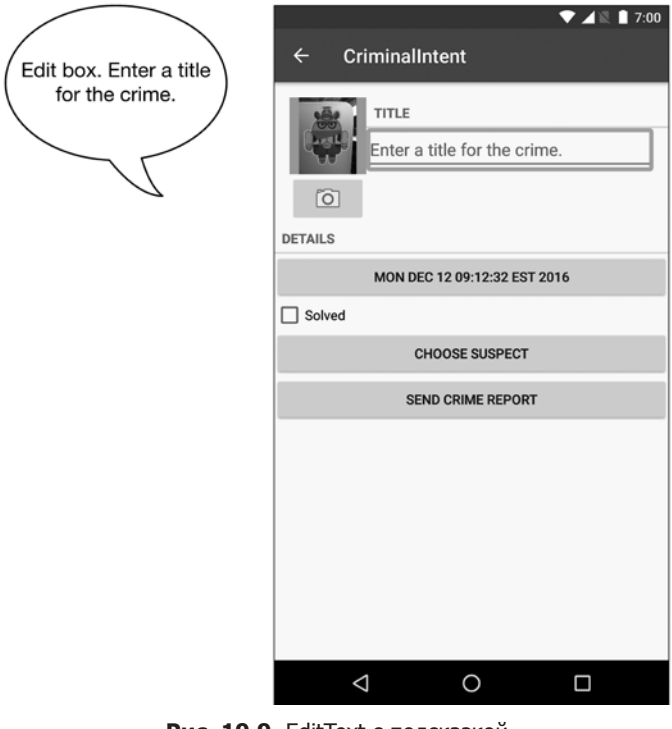


Рис. 19.9. EditText с подсказкой

По умолчанию TalkBack зачитывает текст, содержащийся в `EditText`. Поскольку описание еще не введено, это означает, что TalkBack зачитает значение, заданное свойству `android:hint`. А значит, задавать описание контента для `EditText` необязательно (да и нежелательно).

Однако в приложении есть проблема: чтобы увидеть ее, введите текст. Например, «Вандализм с наклейками». Затем нажмите на поле `EditText`. TalkBack объявляет: «Текстовое поле. Вандализм с наклейками» (рис. 19.10).

Проблема в том, что после ввода текста пользователи TalkBack теряют контекст информации, содержащейся в `EditText`. Пользователь с нормальным зрением

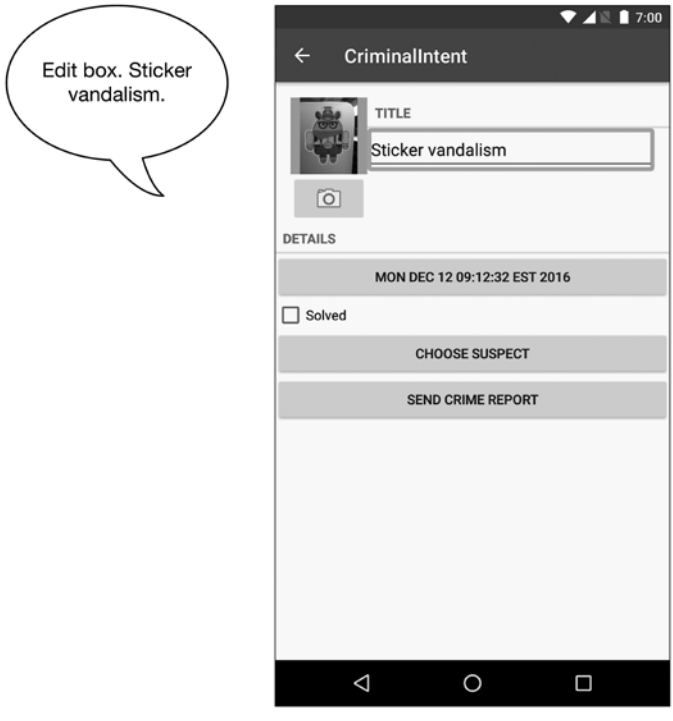


Рис. 19.10. EditText с описанием преступления

поймет, что виджет `EditText` предназначен для описания преступления, благодаря расположенному выше виджету `TextView`. Так как данные преступления довольно просты, пользователи TalkBack, вероятно, поймут, для чего нужен виджет `EditText`, по его содержимому. Но это означает, что пользователям TalkBack придется выполнить больше работы, чем пользователям с нормальным зрением.

Вы сможете легко предоставить тот же контекст пользователям TalkBack, для этого следует явно обозначить связь между `TextView` и `EditText`. Для этого добавьте атрибут `android:labelFor` к виджету надписи в файл макета.

Листинг 19.5. Назначение надписи для виджета EditText (res/layout/fragment_crime.xml)

```
<TextView
    style="?android:listSeparatorTextViewStyle"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/crime_title_label"
    android:labelFor="@+id/crime_title"/>
```

Атрибут `android:labelFor` сообщает TalkBack, что `TextView` служит пояснительной надписью для представления, заданного значением идентификатора. Атри-

бут `labelFor` определяется для класса `View`, поэтому любое представление может быть назначено в качестве надписи для любого другого представления. Обратите внимание на необходимость использования синтаксиса `@+id`, потому что вы ссылаетесь на идентификатор, который еще не был определен в этой точке файла. Теперь можно удалить `+` из строки `android:id="@+id/crime_title"` в определении `EditText`, но делать это необязательно.

Запустите приложение и нажмите на виджете `EditText`. На этот раз TalkBack объявляет: «Текстовое поле. Вандализм с наклейками, для описания».

Поздравляем: приложение стало более доступным! Объясняя отсутствие поддержки доступности в своих приложениях, разработчики обычно отговариваются тем, что не знали, что это нужно. Теперь вы это знаете и видите, как легко сделать приложение доступным для пользователей TalkBack. Кроме того, улучшение поддержки TalkBack в вашем приложении означает, что оно с большей вероятностью будет поддерживать другие средства доступности, такие, как BrailleBack.

Проектирование и реализация доступного приложения выглядит достаточно серьезным делом. Разработчики делают карьеру в качестве специалистов по доступности. Но не стоит полностью отказываться от поддержки доступности только потому, что вы опасаетесь сделать что-то неправильно; лучше начать с азов — проследите за тем, чтобы каждый осмысленный фрагмент контента мог воспроизводиться TalkBack. Убедитесь в том, что пользователи TalkBack получают достаточно контекста, чтобы понять, что происходит в вашем приложении, — и при этом они не тратят время на прослушивание лишней информации. А самое важное — прислушивайтесь к мнению пользователей!

На этом наше знакомство с CriminalIntent подходит к концу. За 13 глав мы создали сложное приложение, которое использует фрагменты, взаимодействует с другими приложениями, делает снимки, сохраняет данные и даже говорит на испанском языке. Почему бы не отпраздновать окончание работы куском торта?

Только не забудьте убрать за собой крошки, чтобы ваш проступок не попал в CriminalIntent.

Для любознательных: Accessibility Scanner

В этой главе мы работали над тем, чтобы приложение стало более доступным для людей, использующих TalkBack. Но это еще не все — ослабленное зрение составляет всего лишь одну из подкатегорий доступности.

В тестировании приложения для обеспечения доступности следует задействовать пользователей, которые регулярно пользуются средствами доступности. Но даже если это невозможно, вы все равно должны постараться по возможности улучшить доступность своего приложения.

Компания Google разработала программу Accessibility Scanner, который анализирует приложения на предмет обеспечения доступности. Программа предоставляет рекомендации на основании своих результатов. Опробуйте ее на приложении CriminalIntent.

Для начала выполните простые инструкции по адресу *play.google.com/store/apps/details?id=com.google.android.apps.accessibility.auditor* и установите Accessibility Scanner на своем устройстве.

Когда программа будет установлена и на экране появится плавающая синяя «галочка», начнется самое интересное. Запустите CriminalIntent из лаунчера или диспетчера задач. Когда приложение запустится, откройте в нем экран с подробной информацией о преступлении (рис. 19.11).

Нажмите на «галочке», и Accessibility Scanner приступит к работе. Во время анализа на экране появится индикатор ожидания. Когда анализ будет завершен, появится окно с рекомендациями (рис. 19.12).

Рис. 19.11. Запуск CriminalIntent для анализа

Рис. 19.12. Сводка результатов Accessibility Scanner

Виджеты `ImageView`, `EditText` и `CheckBox` окружены рамками. Это означает, что программа обнаружила потенциальные проблемы с доступностью этих виджетов. Нажмите на `EditText`, чтобы просмотреть рекомендации по доступности этого виджета (рис. 19.13).

У Accessibility Scanner есть три рекомендации. Первая связана с размером `EditText`. Рекомендованный минимальный размер всех элементов, которых дол-

жен касаться пользователь, равен 48dp. Высота `EditText` меньше; эта проблема легко решается назначением атрибута `android:minHeight` для виджета.

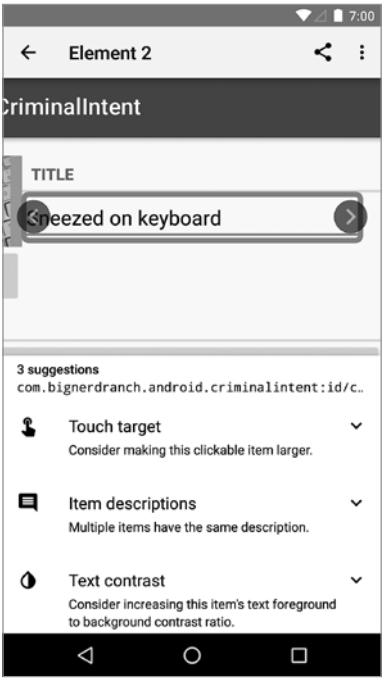


Рис. 19.13. Рекомендации по доступности EditText

Вторая проблема связана с надписью. В рекомендации указано, что информация, предоставляемая `TextView` и `EditText`, может быть избыточной, так как `EditText` ссылается на `TextView`. Тем не менее в нашем случае избыточность не важна, поэтому эту рекомендацию можно игнорировать.

Последняя рекомендация относится к контрасту между цветом текста и цветом фона. В Интернете можно найти инструменты для оценки контрастности двух цветов на основании их яркости. Также существуют специальные сайты (например, *randoma11y.com*) с комбинациями цветов, удовлетворяющим стандартам контрастности. Что это за стандарты? Комитет World Wide Web Consortium — международная организация, разрабатывающая открытые стандарты для веб-дизайна, — рекомендует использовать для текста высотой 18 пунктов соотношение контрастности не менее 4,5:1. Для цветов `EditText` в приложении CriminalIntent это соотношение равно 3,52. Проблема решается назначением атрибутов `android:textColor` и `android:background`.

Чтобы получить более подробную информацию о каждой из рекомендаций Accessibility Scanner, вызовите подробную информацию (стрелка, направленная вниз) и нажмите Learn More.

Упражнение. Улучшение списка

На экране со списком преступлений TalkBack читает описание и дату каждого элемента. Однако диктор не сообщает, было преступление раскрыто или нет. Чтобы устранить этот недостаток, назначьте описание контента значку с изображением наручников.

Сводка получается довольно длинной (с учетом формата даты), а статус раскрытия преступления зачитывается в самом конце — или вообще не зачитывается, если преступление не раскрыто. Чтобы задание стало еще более интересным, вместо добавления описания контента к значку в XML добавьте динамическое описание для каждого видимого элемента `RecyclerView`. Описание должно содержать сводку данных, выводимых в одной строке.

Упражнение. Предоставление контекста для ввода данных

Кнопке даты и кнопке выбора подозреваемого присущ тот же недостаток, что и исходному текстовому полю с описанием: в приложении нет никаких явных признаков того, для чего нужна кнопка с датой (независимо от того, используется функция TalkBack или нет). Аналогичным образом после выбора контакта в качестве подозреваемого пользователь не получает никакой информации о том, что представляет кнопка. Возможно, пользователь и так догадается о предназначении кнопок и текста на этих кнопках, но стоит ли держать его в неведении?

В этом проявляется одна из тонкостей проектирования пользовательских интерфейсов. Вы (или ваша дизайн-группа) определяете, какие решения будут наиболее логичными для вашего приложения — чтобы выдержать баланс между простотой интерфейса и простотой взаимодействий.

Измените реализацию экрана с подробной информацией, чтобы пользователь не терял смысловой контекст выбранных им данных. Проблема может решаться простым добавлением надписей ко всем полям, как это было сделано для `EditText` с заголовком. Возможны и более сложные решения, например полная переработка экрана детализации. Выбор за вами. Сделайте свое приложение более доступным, чтобы все желающие могли обсудить поведение своих невоспитанных коллег.

Упражнение. Оповещения о событиях

Динамические описания контента для виджетов `ImageView` с фотографиями места преступления упрощают работу с фотографиями. С другой стороны, пользователь TalkBack должен прикоснуться к виджету `ImageView`, чтобы проверить его статус. Пользователю с нормальным зрением проще: он видит, что изображение изменилось (или не изменилось) при возвращении из приложения камеры.

Аналогичную возможность можно реализовать и средствами TalkBack: вы сообщаете пользователю о произошедшем в результате закрытия приложения камеры. Прочитайте описание метода `View.announceForAccessibility(…)` в документации и используйте его в CriminalIntent в подходящий момент.

Возможно, вы решите выдать оповещение в `onActivityResult(…)`. В таком случае могут возникнуть проблемы с временем выдачи оповещения, связанные с жизненным циклом активности. Проблему можно решить небольшой задержкой оповещения за счет отправки `Runnable` (эта тема более подробно рассматривается в главе 26). Реализация выглядит примерно так:

```
mSomeView.postDelayed(new Runnable() {
    @Override
    public void run() {
        // Код оповещения
    }
}, SOME_DURATION_IN_MILLIS);
```

Также можно обойтись без `Runnable` и воспользоваться другим механизмом для определения момента оповещения об изменениях. Например, оповещение можно выдать в `onResume()` — хотя в этом случае нужно будет проверять, вернулся ли пользователь из приложения камеры или нет.

20 Привязка данных и MVVM

В GeoQuiz и CriminalIntent для построения приложений Android использовались стандартные средства: модель, которая строилась из объектов Java, иерархия представлений и объекты-контроллеры — активности и фрагменты. Для этих проектов архитектура MVC хорошо подходила.

В этом проекте вы научитесь использовать механизм *привязки данных*. Привязка данных — всего лишь инструмент, который не дает никаких рекомендаций относительно того, как им пользоваться. Впрочем, у нас есть свое мнение по этому поводу, и мы покажем, как мы используем привязку данных для реализации архитектуры *MVVM* (Model-View-ViewModel). Кроме того, в этой главе представлено использование системы активов для хранения звуковых файлов.

В этой главе также начнется работа над новым приложением BeatBox (рис. 20.1), предназначенным для воспроизведения всевозможных угрожающих звуков.

Рис. 20.1. Приложение BeatBox к концу этой главы

Другие архитектуры: для чего?

Во всех приложениях, написанных вами до сих пор, используется простая версия MVC. И до сих пор — если мы хорошо справились со своим делом — все приложения выглядели вполне логично. Зачем что-то менять? В чем проблема?

Архитектура MVC в том виде, в каком она применяется в книге, хорошо подходит для небольших, простых приложений. В нее легко добавляются новые функции, и она позволяет легко представить взаимодействие между взаимодействующими частями приложения. MVC закладывает надежную основу для разработки, позволяет быстро построить работоспособное приложение и эффективно работает на ранних стадиях работы над проектом.

Проблемы начинаются тогда, когда размеры вашей программы выходят за рамки примеров, приведенных в книге, — как это происходит практически во всех программах. Большие фрагменты и активности трудно расширять, и в логике их работы трудно разобраться. На реализацию новых возможностей и исправление ошибок уходит больше времени. На какой-то стадии контроллеры необходимо разделить на меньшие, более удобные части.

Как это сделать? Определите различные операции, выполняемые большим классом контроллера, и выделите каждую операцию в отдельный класс. Вместо одного большого класса появятся экземпляры нескольких классов, которые будут совместно выполнять общую работу.

Как определить эти разные задания? Ответ на этот вопрос лежит в определении архитектуры. Такие описания, как «Модель-Представление-Контроллер» и «Модель-Представление-Презентатор», используются для описания таких ответов на высоком уровне. Однако в конечном итоге на этот вопрос всегда отвечаете только вы, и итоговая архитектура определяется только вами.

Приложение BeatBox построено на базе архитектуры MVVM. Нам архитектура MVVM очень нравится, потому что она отлично справляется с вынесением большого объема рутинного кода контроллера в файл макета, где вы сразу видите, какие части интерфейса являются динамическими. В то же время содержательный динамический код контроллера выносится в класс модели представления `ViewModel`, что упрощает его тестирование и проверку работоспособности.

Величина модели представления всегда должна определяться здравым смыслом. Если ваша модель представления будет слишком большой, ее можно разбить на несколько частей. Ваша архитектура принадлежит только вам, и никому другому.

Создание приложения BeatBox

Пора браться за дело. Начнем с создания приложения BeatBox. Выполните в Android Studio команду File ▸ New Project... для создания нового проекта. Присвойте ему имя BeatBox и введите домен компании android.bignerdranch.com. Выберите минимальный уровень SDK API 19 и начните с пустой активности с именем BeatBoxActivity. Оставьте остальным параметрам значения по умолчанию.

В приложении снова будет использоваться виджет `RecyclerView`, поэтому откройте настройки проекта и добавьте зависимость com.android.support:recyclerview-v7.

Перейдем к построению основы приложения. На главном экране будет отображаться сетка из кнопок, каждая из которых воспроизводит определенный звук. Нам понадобятся два файла макета: один для сетки, другой для кнопок.

Сначала создайте файл макета для `RecyclerView`. Файл res/layout/activity_beat_box.xml нам не понадобится, поэтому переименуйте его в fragment_beat_box.xml.

Откройте переименованный файл fragment_beat_box.xml. Удалите существующее содержимое файла и заполните его следующим образом:

Листинг 20.1. Изменение основного файла макета (res/layout/fragment_beat_box.xml)

```
<android.support.v7.widget.RecyclerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/recycler_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Теперь создайте в com.bignerdranch.android.beatbox новый фрагмент с именем `BeatBoxFragment`.

Листинг 20.2. Создание BeatBoxFragment (BeatBoxFragment.java)

```
public class BeatBoxFragment extends Fragment {
    public static BeatBoxFragment newInstance() {
        return new BeatBoxFragment();
    }
}
```

Пока оставьте его пустым.

Затем создайте активность `BeatBoxActivity`, в которой должен отображаться новый фрагмент. Мы воспользуемся той же архитектурой `SingleFragmentActivity`, которая использовалась в CriminalIntent.

В своем любимом файловом менеджере или терминальном приложении скопируйте файл SingleFragmentActivity.java из CriminalIntent в BeatBox/app/src/main/java/com/bignerdranch/android/beatbox/, а затем скопируйте файл activity_fragment.xml в BeatBox/app/src/main/res/layout/. (Возьмите их из папки CriminalIntent или из решений. За информацией о том, как получить доступ к файлам решений, обращайтесь к разделу «Добавление значка» главы 2.)

Теперь удалите весь код из тела класса `BeatBoxActivity`, назначьте его субклассом `SingleFragmentActivity` и переопределите метод `createFragment()`:

Листинг 20.3. Код BeatBoxActivity (BeatBoxActivity.java)

```
public class BeatBoxActivity extends SingleFragmentActivity {
    @Override
    protected Fragment createFragment() {
        return BeatBoxFragment.newInstance();
    }
}
```

Простая привязка данных

Следующая задача — заполнение файла fragment_beat_box.xml и подключение `RecyclerView`. Вы уже делали это прежде, но на этот раз мы используем привязку данных для ускорения работы. Начните с включения привязки данных в файле build.gradle приложения.

Листинг 20.4. Включение привязки данных (app/build.gradle)

```
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
              'proguard-rules.pro'
        }
    }
    dataBinding {
        enabled = true
    }
}

dependencies {
```

При этом активизируется механизм интеграции IDE, который позволит вам обращаться к классам, генерируемым привязкой данных, и интегрировать генерирование этих классов в процесс построения.

Чтобы использовать привязку данных с конкретным файлом макета, следует преобразовать его в файл макета с привязкой данных. Для этого весь файл XML заключается в тег `<layout>`.

Листинг 20.5. Заключение разметки в тег <layout> (res/layout/fragment_beat_box.xml)

```
<layout
    xmlns:android="http://schemas.android.com/apk/res/android">
    <android.support.v7.widget.RecyclerView
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/recycler_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
</layout>
```

Тег `<layout>` сигнализирует средствам привязки данных, что они должны обработать файл макета. После завершения обработки будет сгенерирован *класс привязки*. По умолчанию имя класса выбирается в соответствии с именем файла макета, но вместо стандартной схемы регистра (`snake_case`) используется «верблюжья» схема `CamelCase`.

Ваш файл fragment_beat_box.xml уже сгенерировал файл привязки с именем `FragmentBeatBoxBinding`. Этот класс будет использоваться для привязки данных: вместо заполнения иерархии представлений с использованием `LayoutInflater` мы заполним экземпляр `FragmentBeatBoxBinding`. `FragmentBeatBoxBinding` сохраняет иерархию представлений в `get`-методе с именем `getRoot()`. Кроме того, сохраняются именованные ссылки для всех представлений, помеченных в файле макета атрибутом `android:id`.

Таким образом, класс `FragmentBeatBoxBinding` сохраняет две ссылки: `getRoot()` для всего макета и `recyclerView` для `RecyclerView` (рис. 20.2).

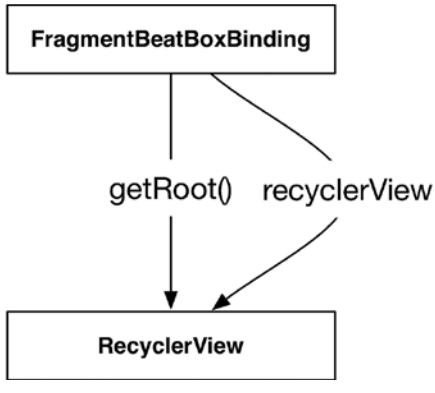


Рис. 20.2. Класс привязки

Конечно, в нашем макете только одно представление, поэтому обе ссылки указывают на одно и то же представление: `RecyclerView`.

Перейдем к использованию класса привязки. Переопределите `onCreateView(…)` в `BeatBoxFragment` и используйте `DataBindingUtil` для заполнения экземпляра `FragmentBeatBoxBinding` (листинг 20.6). (Класс `FragmentBeatBoxBinding` должен импортироваться, как и любой другой класс. Если Android Studio не может найти `FragmentBeatBoxBinding`, это означает, что класс не был автоматически сгенерирован по какой-либо причине. Прикажите Android Studio сгенерировать класс командой Build ▸ Rebuild Project. Если класс не был сгенерирован после повторного построения проекта, перезапустите Android Studio.)

Листинг 20.6. Заполнение класса привязки (BeatBoxFragment.java)

```
public class BeatBoxFragment extends Fragment {
    public static BeatBoxFragment newInstance() {
        return new BeatBoxFragment();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {

        FragmentBeatBoxBinding binding = DataBindingUtil
```

```
                .inflate(inflater, R.layout.fragment_beat_box, container, false);
        return binding.getRoot();
    }
}
```

После создания привязки можно заняться виджетом `RecyclerView` и его настройкой.

Листинг 20.7. Настройка RecyclerView (BeatBoxFragment.java)

```
public class BeatBoxFragment extends Fragment {
    public static BeatBoxFragment newInstance() {
        return new BeatBoxFragment();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        FragmentBeatBoxBinding binding = DataBindingUtil
                .inflate(inflater, R.layout.fragment_beat_box, container, false);

        binding.recyclerView.setLayoutManager(new GridLayoutManager
                             (getActivity(), 3));

        return binding.getRoot();
    }
}
```

Это пример так называемой *простой привязки данных* — использования привязки данных для автоматического извлечения представлений вместо `findViewById(…)`. Далее будут рассмотрены более сложные применения привязки данных, а пока продолжим связывание данных с `RecyclerView`.

Теперь создайте макет для кнопок, res/layout/list_item_sound.xml. Здесь также будет использоваться привязка данных, поэтому заключите файл макета в тег `<layout>`.

Листинг 20.8. Создание макета (res/layout/list_item_sound.xml)

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools">
    <Button
        android:layout_width="match_parent"
        android:layout_height="120dp"
        tools:text="Sound name"/>
</layout>
```

Затем создайте объект `SoundHolder`, связанный с list_item_sound.xml.

Листинг 20.9. Создание объекта SoundHolder (BeatBoxFragment.java)

```
public class BeatBoxFragment extends Fragment {
    public static BeatBoxFragment newInstance() {
        return new BeatBoxFragment();
    }

    @Override
```

```
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        ...
    }

    private class SoundHolder extends RecyclerView.ViewHolder {
        private ListItemSoundBinding mBinding;

        private SoundHolder(ListItemSoundBinding binding) {
            super(binding.getRoot());
            mBinding = binding;
        }
    }
}
```

Затем создайте адаптер, связанный с `SoundHolder`. (Если вы поместите курсор в `RecyclerView.Adapter`, прежде чем вводить любые из приведенных ниже методов, и нажмете Option+Return (Alt+Enter), Android Studio сгенерирует большую часть кода за вас.)

Листинг 20.10. Создание класса SoundAdapter (BeatBoxFragment.java)

```
public class BeatBoxFragment extends Fragment {
    ...
    private class SoundHolder extends RecyclerView.ViewHolder {
        ...
    }

    private class SoundAdapter extends RecyclerView.Adapter<SoundHolder> {
        @Override
        public SoundHolder onCreateViewHolder(ViewGroup parent, int viewType) {
            LayoutInflater inflater = LayoutInflater.from(getActivity());
            ListItemSoundBinding binding = DataBindingUtil
                    .inflate(inflater, R.layout.list_item_sound, parent, false);
            return new SoundHolder(binding);
        }

        @Override
        public void onBindViewHolder(SoundHolder holder, int position) {
        }

        @Override
        public int getItemCount() {
            return 0;
        }
    }
}
```

Подключите `SoundAdapter` в методе `onCreateView(…)`.

Листинг 20.11. Подключение SoundAdapter (BeatBoxFragment.java)

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    FragmentBeatBoxBinding binding = DataBindingUtil
            .inflate(inflater, R.layout.fragment_beat_box, container, false);

    binding.recyclerView.setLayoutManager(new GridLayoutManager(getActivity(), 3));
    binding.recyclerView.setAdapter(new SoundAdapter());

    return binding.getRoot();
}
```

Импортирование активов

Пришло время добавить в проект звуковые файлы и загрузить их на стадии выполнения. Вместо системы ресурсов будут использоваться *активы* (assets). Активы можно рассматривать как усеченные аналоги ресурсов: они упаковываются в APK, как и ресурсы, но без конфигурационного инструментария, дополняющего ресурсы.

В некоторых отношениях это хорошо. Из-за отсутствия системы конфигурации активам можно присваивать любые имена и упорядочивать их в структуру папок. Впрочем, есть и оборотная сторона: без системы конфигурации ваше приложение не сможет автоматически реагировать на изменение плотности пикселов, языка или ориентации или же автоматически использовать активы в файлах макетов или других ресурсах.

Обычно предпочтение отдается ресурсам. Однако в тех случаях, когда вы обращаетесь к файлам только на программном уровне, активы выходят на первый план. Например, многие игры используют активы для хранения графики и звука — и такое же решение будет использовано в BeatBox.

Используемые активы следует импортировать в проект. Создайте в проекте папку для активов: щелкните правой кнопкой мыши на модуле app и выберите команду New ▸ Folder ▸ Assets Folder (рис. 20.3). Оставьте флажок Change Folder Location снятым, а в списке Target Source Set выберите вариант main.

Щелкните на кнопке Finish, чтобы создать папку для активов.

Щелкните правой кнопкой мыши на папке assets и выберите команду New ▸ Directory. Введите имя каталога sample_sounds (рис. 20.4).

Все содержимое папки assets интегрируется в приложение. Для удобства и порядка мы создали вложенную папку с именем sample_sounds. Впрочем, в отличие от ресурсов, делать это было необязательно.

Где найти звуки? Мы воспользуемся подборкой, распространяемой на условиях лицензии Creative Commons, которую мы впервые увидели у пользователя *plagasul* (*www.freesound.org/people/plagasul/packs/3/*). Мы поместили все звуки в один zip-архив по адресу: *www.bignerdranch.com/solutions/sample_sounds.zip*.

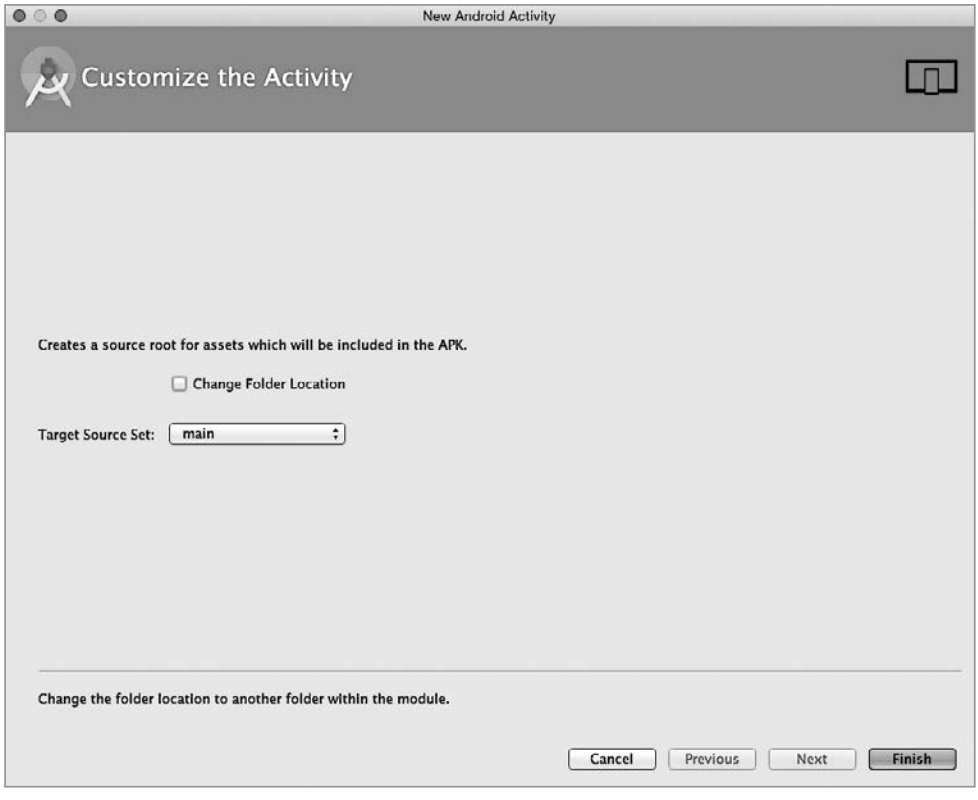


Рис. 20.3. Создание папки assets

Рис. 20.4. Создание папки sample_sounds

Загрузите архив и распакуйте его содержимое в папку assets/sample_sounds (рис. 20.5).

(Кстати, проследите за тем, чтобы там находились файлы .wav, а не только файл .zip, из которого они были извлечены.)

Постройте приложение заново. Следующим шагом станет получение списка активов и вывод его для пользователя.

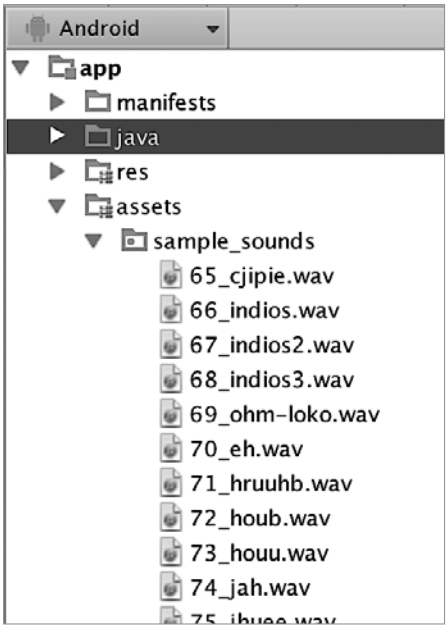


Рис. 20.5. Импортированные активы

Получение информации об активах

Приложение будет выполнять множество операций, относящихся к управлению активами: поиск, отслеживание и в конечном итоге воспроизведение их как звуков. Для выполнения этих операций создайте новый класс с именем `BeatBox` в пакете `com.bignerdranch.android.beatbox`. Добавьте пару констант: для вывода информации в журнал и для имени папки, в которой были сохранены звуки.

Листинг 20.12. Новый класс BeatBox (BeatBox.java)

```
public class BeatBox {
    private static final String TAG = "BeatBox";

    private static final String SOUNDS_FOLDER = "sample_sounds";
}
```

Для обращения к активам используется класс `AssetManager`. Экземпляр этого класса можно получить для любой разновидности `Context`. Так как BeatBox понадобится такой экземпляр, предоставьте ему конструктор, который получает `Context`, извлекает `AssetManager` и сохраняет на будущее.

Листинг 20.13. Сохранение AssetManager (BeatBox.java)

```
public class BeatBox {
    private static final String TAG = "BeatBox";

    private static final String SOUNDS_FOLDER = "sample_sounds";

    private AssetManager mAssets;

    public BeatBox(Context context) {
        mAssets = context.getAssets();
    }
}
```

Как правило, при работе с активами вам не нужно беспокоиться о том, какой именно объект `Context` будет использоваться. Во всех ситуациях, которые вам встретятся на практике, объект `AssetManager` всех разновидностей `Context` будет связан с одним набором активов.

Чтобы получить список доступных активов, используйте метод `list(String)`. Напишите метод `loadSounds()`, который обращается к активам при помощи метода `list(String)`.

Листинг 20.14. Получение списка активов (BeatBox.java)

```
public BeatBox(Context context) {
    mAssets = context.getAssets();
    loadSounds();
}

private void loadSounds() {
    String[] soundNames;
    try {
        soundNames = mAssets.list(SOUNDS_FOLDER);
        Log.i(TAG, "Found " + soundNames.length + " sounds");
    } catch (IOException ioe) {
        Log.e(TAG, "Could not list assets", ioe);
        return;
    }
}
```

Метод `AssetManager.list(String)` возвращает список имен файлов, содержащихся в заданной папке. Передав ему путь к папке со звуками, вы получите информацию обо всех файлах .wav в этой папке.

Чтобы убедиться в том, что система активов работает правильно, создайте экземпляр `BeatBox` в `BeatBoxFragment`.

Листинг 20.15. Создание экземпляра BeatBox (BeatBoxFragment.java)

```
public class BeatBoxFragment extends Fragment {

    private BeatBox mBeatBox;

    public static BeatBoxFragment newInstance() {
        return new BeatBoxFragment();
```

```
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        mBeatBox = new BeatBox(getActivity());
    }
    ...
}
```

Запустите приложение. В журнале должна появиться информация о том, сколько звуковых файлов было обнаружено. Мы использовали 22 файла в формате .wav, и если вы использовали наши файлы, результат должен выглядеть так:

```
…1823-1823/com.bignerdranch.android.beatbox I/BeatBox: Found 22 sounds
```

Подключение активов для использования

Имена файлов активов получены, теперь нужно вывести их для пользователя. В конечном итоге файлы должны воспроизводиться, поэтому стоит создать объект для хранения имени файла, того имени, которое видит пользователь, и всей остальной информации, относящейся к данному звуку.

Создайте класс `Sound` для хранения всей этой информации. (Не забудьте поручить Android Studio сгенерировать `get`-методы.)

Листинг 20.16. Создание объекта Sound (Sound.java)

```
public class Sound {
    private String mAssetPath;
    private String mName;

    public Sound(String assetPath) {
        mAssetPath = assetPath;
        String[] components = assetPath.split("/");
        String filename = components[components.length - 1];
        mName = filename.replace(".wav", "");
    }

    public String getAssetPath() {
        return mAssetPath;
    }

    public String getName() {
        return mName;
    }
}
```

В конструкторе выполняется небольшая подготовительная работа для построения удобочитаемого имени звука. Сначала имя файла отделяется вызовом `String.split(String)`, после чего вызов `String.replace(String, String)` удаляет расширение.

Далее в методе `BeatBox.loadSounds()` строится список объектов `Sound`.

Листинг 20.17. Создание списка объектов Sound (BeatBox.java)

```
public class BeatBox {
    ...
    private AssetManager mAssets;
    private List<Sound> mSounds = new ArrayList<>();

    public BeatBox(Context context) {
        ...
    }

    private void loadSounds() {
        String[] soundNames;
        try {
            ...
        } catch (IOException ioe) {
            ...
        }

        for (String filename : soundNames) {
            String assetPath = SOUNDS_FOLDER + "/" + filename;
            Sound sound = new Sound(assetPath);
            mSounds.add(sound);
        }
    }

    public List<Sound> getSounds() {
        return mSounds;
    }
}
```

Затем свяжите `SoundAdapter` со списком объектов `Sound`.

Листинг 20.18. Связывание со списком объектов Sound (BeatBoxFragment.java)

```
private class SoundAdapter extends RecyclerView.Adapter<SoundHolder> {
    private List<Sound> mSounds;

    public SoundAdapter(List<Sound> sounds) {
        mSounds = sounds;
    }
    ...
    @Override
    public void onBindViewHolder(SoundHolder soundHolder, int position) {
    }

    @Override
    public int getItemCount() {
        return 0;
        return mSounds.size();
    }
}
```

Звуки из `BeatBox` передаются в методе `onCreateView(…)`.

Листинг 20.19. Передача звуков адаптеру (BeatBoxFragment.java)

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
    FragmentBeatBoxBinding binding = DataBindingUtil
            .inflate(inflater, R.layout.fragment_beat_box, container, false);

    binding.recyclerView.setLayoutManager(new GridLayoutManager(getActivity(), 3));
    binding.recyclerView.setAdapter(new SoundAdapter());
    binding.recyclerView.setAdapter(new SoundAdapter(mBeatBox.getSounds()));

    return binding.getRoot();
}
```

После добавления этого кода при запуске BeatBox на экране появляется сетка с кнопками (рис. 20.6).

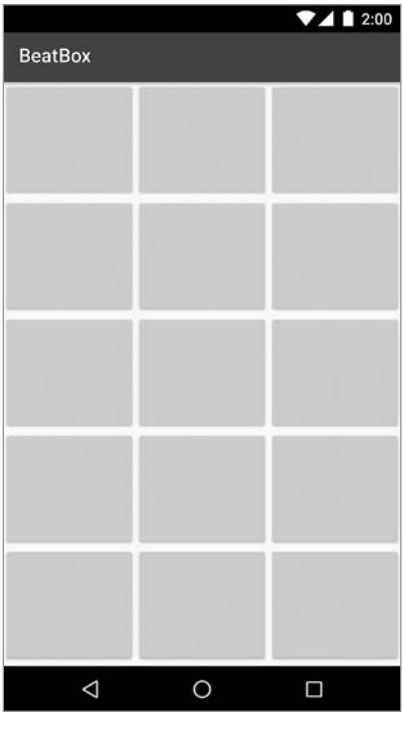


Рис. 20.6. Пустые кнопки

Чтобы заполнить кнопки названиями, необходимо воспользоваться некоторыми дополнительными инструментами привязки данных.

Установление связи с данными

При использовании привязки данных объекты данных могут объявляться в файле макета:

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools">
    <data>
        <variable
            name="crime"
            type="com.bignerdranch.android.criminalintent.Crime"/>
    </data>
    ...
</layout>
```

Далее значения из этих объектов используются прямо в файле макета при помощи оператора привязки `@{}`:

```
<CheckBox
    android:id="@+id/list_item_crime_solved_check_box"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentRight="true"
    android:checked="@{crime.isSolved()}"
    android:padding="4dp"/>
```

Диаграмма объектов выглядит так:

Рис. 20.7. Связи данных

Наша непосредственная цель — разместить на кнопках названия звуков. Самое простое решение на базе привязки данных основано на прямом связывании с объектом Sound в list_item_sound.xml, как показано на рис. 20.8.

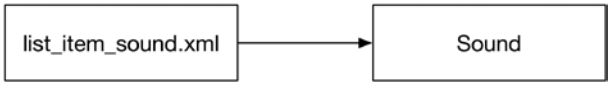


Рис. 20.8. Прямое связывание

Однако такое решение создает ряд архитектурных проблем. Чтобы понять, в чем дело, взгляните на происходящее с точки зрения MVC (рис. 20.9).

Направляющим принципом любой архитектуры должен быть *принцип единственной обязанности*. Этот принцип гласит, что каждый класс должен иметь ровно одну обязанность. MVC дает представление о том, какими могут быть эти обязанности: модель описывает, как работает ваше приложение; контроллер управляет отображением; а представление отображает его на экране так, как вы пожелаете.

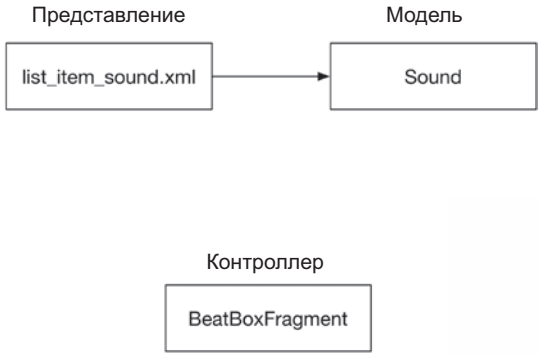


Рис. 20.9. Нарушение архитектуры MVC

Использование привязки данных так, как показано на рис. 20.8, нарушит это разделение обязанностей — ведь скорее всего, решать, как должны отображаться данные, придется объекту модели `Sound`. В приложении быстро воцарится хаос, потому что файл Sound.java будет загромождён кодом двух видов: определяющим работу приложения и определяющим способ отображения данных.

Вместо того, чтобы запутывать обязанности `Sound`, мы добавим новый объект `ViewModel` (VM) для привязки данных. Объект `ViewModel` отвечает за принятие решений относительно того, как должны отображаться данные (рис. 20.10).

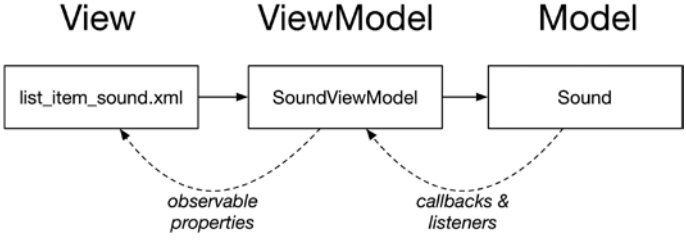


Рис. 20.10. Архитектура MVVM

Такая архитектура называется *MVVM*. Большая часть работы по форматированию данных, которая ранее выполнялась классами контроллеров, переходит в `ViewModel`. Подключение виджетов к данным будет осуществляться непосредственно в файле макета с использованием привязки данных к `ViewModel`. Контроллер (ваша активность или фрагмент) будет отвечать за такие операции, как инициализация привязки и `ViewModel` и создание связи между ними.

Создание ViewModel

Создайте новый класс с именем `SoundViewModel`. Класс должен содержать два свойства: для объекта `Sound` и для объекта `BeatBox` (который в конечном итоге будет использоваться для воспроизведения).

Листинг 20.20. Создание SoundViewModel (SoundViewModel.java)

```
public class SoundViewModel {
    private Sound mSound;
    private BeatBox mBeatBox;

    public SoundViewModel(BeatBox beatBox) {
        mBeatBox = beatBox;
    }

    public Sound getSound() {
        return mSound;
    }

    public void setSound(Sound sound) {
        mSound = sound;
    }
}
```

Эти свойства образуют интерфейс, который будет использоваться вашим адаптером. Файлу макета потребуется дополнительный метод для получения названия, которое должно отображаться на кнопке. Добавьте его в `SoundViewModel`.

Листинг 20.21. Добавление методов привязки (SoundViewModel.java)

```
public class SoundViewModel {
    private Sound mSound;
    private BeatBox mBeatBox;

    public SoundViewModel(BeatBox beatBox) {
        mBeatBox = beatBox;
    }

    public String getTitle() {
        return mSound.getName();
    }

    public Sound getSound() {
        return mSound;
```

Связывание с ViewModel

Теперь класс `ViewModel` следует связать с файлом макета. Начните с объявления свойства в файле макета.

Листинг 20.22. Объявление свойства для ViewModel (list_item_sound.xml)

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools">
    <data>
        <variable
            name="viewModel"
            type="com.bignerdranch.android.beatbox.SoundViewModel"/>
    </data>
    <Button
```

Этот блок разметки определяет свойство `viewModel` для класса привязки, вместе с `get`- и `set`-методом. В классе привязки свойство `viewModel` может использоваться в выражениях привязки.

Листинг 20.23. Привязка названия кнопки (list_item_sound.xml)

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools">
    <data>
        <variable
            name="viewModel"
            type="com.bignerdranch.android.beatbox.SoundViewModel"/>
    </data>
    <Button
        android:layout_width="match_parent"
        android:layout_height="120dp"
        android:text="@{viewModel.title}"
        tools:text="Sound name"/>
</layout>
```

В операторе привязки `{ }` можно использовать простые выражения Java, включая сцепленные вызовы методов, математические выражения... и вообще практически все, что вы сочтете нужным включить. Также поддерживаются некоторые вспомогательные конструкции, избавляющие от необходимости вводить лишние символы. Например, запись `viewModel.title` является сокращением для `viewModel.getTitle()`. Механизм привязки знает, что ссылку на свойство следует преобразовать в вызов соответствующего метода.

Остается подключить `ViewModel`. Создайте объект `SoundViewModel` и присоедините его к классу привязки. Затем добавьте метод привязки в `SoundHolder`.

Листинг 20.24. Подключение ViewModel (BeatBoxFragment.java)

```
private class SoundHolder extends RecyclerView.ViewHolder {
    private ListItemSoundBinding mBinding;

    private SoundHolder(ListItemSoundBinding binding) {
        super(binding.getRoot());
        mBinding = binding;
        mBinding.setViewModel(new SoundViewModel(mBeatBox));
    }

    public void bind(Sound sound) {
        mBinding.getViewModel().setSound(sound);
        mBinding.executePendingBindings();
    }
}
```

В конструкторе создается и присоединяется объект `ViewModel`. Затем в методе `bind` обновляются данные, с которыми работает `ViewModel`.

Вызов `executePendingBindings()` обычно не нужен. Однако в данном случае данные привязки обновляются в виджете `RecyclerView`, который обновляет представ-

ления с очень высокой скоростью. Вызывая этот метод, вы приказываете макету обновить себя немедленно вместо того, чтобы ожидать одну-две миллисекунды. Таким образом обеспечивается быстрота реакции `RecyclerView`.

Наконец, завершите подключение `ViewModel` реализацией `onBindViewHolder(…)`.

Листинг 20.25. Вызов метода bind(Sound) (BeatBoxFragment.java)

```
    return new SoundHolder(binding);
}

@Override
public void onBindViewHolder(SoundHolder holder, int position) {
    Sound sound = mSounds.get(position);
    holder.bind(sound);
}

@Override
public int getItemCount() {
    return mSounds.size();
}
```

Запустите приложение. На всех кнопках на экране выводятся названия (рис. 20.11).

Рис. 20.11. Кнопки с заполненными названиями

Отслеживаемые данные

На первый взгляд все хорошо, но в вашем коде кроется проблема. Чтобы убедиться в этом, достаточно прокрутить экран вниз (рис. 20.12).

Рис. 20.12. Что-то знакомое

Видите элемент 67_INDIOS2 внизу? Он уже встречался выше. Прокручивая список вверх и вниз, вы увидите, что названия других файлов появляются в неожиданных, случайных на первый взгляд местах.

Почему это происходит? Ваш макет не знает о том, что вы обновили объект `Sound` из `SoundViewModel` в методе `SoundHolder.bind(Sound)`. Объект `ViewModel` не «отвечает» файлу макета на рис. 20.10.

Необходимо добавить эту связь. Для этого класс `ViewModel` должен реализовать интерфейс `Observable` привязки данных. Этот интерфейс позволяет классу привязки установить слушателей для `ViewModel`, чтобы он мог автоматически получать обратные вызовы при изменении полей.

Реализация всего интерфейса возможна, но это потребует большого объема работы. Мы в Big Nerd Ranch работы не боимся, но стараемся избегать ее, насколько это возможно. Поэтому мы покажем, как решить эту задачу более эффективно, — при помощи класса `BaseObservable` механизма привязки данных.

Этот способ состоит из трех шагов.

1. Субклассируйте `BaseObservable` в своем классе `ViewModel`.
2. Снабдите свойства, используемые в привязке, аннотацией `@Bindable`.
3. Вызывайте `notifyChange()` или `notifyPropertyChanged(int)` при каждом изменении значения свойства привязки.

В `SoundViewModel` вся процедура состоит из нескольких строк кода. Внесите изменения в `SoundViewModel`.

Листинг 20.26.

```
public class SoundViewModel extends BaseObservable {
    private Sound mSound;
    private BeatBox mBeatBox;

    public SoundViewModel(BeatBox beatBox) {
        mBeatBox = beatBox;
    }

    @Bindable
    public String getTitle() {
        return mSound.getName();
    }

    public Sound getSound() {
        return mSound;
    }

    public void setSound(Sound sound) {
        mSound = sound;
        notifyChange();
    }
}
```

Когда вы вызываете метод `notifyChange()`, он оповещает класс привязки о том, что все `Bindable`-поля ваших объектов были обновлены. Класс привязки выполняет код внутри `{ }` для повторного заполнения представления. Таким образом, при вызове `setSound(Sound)` объект `ListItemSoundBinding` получит уведомление и вызовет `Button.setText(String)`, как указано в файле list_item_sound.xml.

Выше упоминался другой метод: `notifyPropertyChanged(int)`. Он делает то же самое, что и `notifyChange()`, но отличается большей точностью. Используя запись `notifyChange()`, вы говорите: «Все `Bindable`-свойства изменились; все нужно обновить». При использовании `notifyPropertyChanged(BR.title)` можно сообщить: «Изменилось только значение `getTitle()`».

Снова запустите BeatBox. На этот раз при прокрутке все должно работать нормально (рис. 20.13).

Рис. 20.13. Готовый интерфейс BeatBox

Обращение к активам

Вся работа этой главы успешно завершена. В следующей главе приложение BeatBox начнет непосредственную работу с содержимым активов.

Но перед этим давайте чуть подробнее поговорим о том, как работают активы.

С объектом `Sound` связан некоторый путь к файлу. Попытка открыть такой файл с использованием объекта `File` завершается неудачей; для работы с ним следует использовать `AssetManager`:

```
String assetPath = sound.getAssetPath();
InputStream soundData = mAssets.open(assetPath);
```

В результате вы получаете стандартный объект `InputStream` для данных, с которым можно работать в коде точно так же, как с любым другим `InputStream`.

Некоторые API вместо объекта `InputStream` требуют объект `FileDescriptor`. (Именно он будет использоваться при работе с `SoundPool` в следующей главе.) В такой ситуации можно вызвать метод `AssetManager.openFd(String)`:

```
String assetPath = sound.getAssetPath();
// Объекты AssetFileDescriptor отличаются от FileDescriptor,
AssetFileDescriptor assetFd = mAssets.openFd(assetPath);
// но при необходимости вы можете легко получить
// обычный объект FileDescriptor.
FileDescriptor fd = assetFd.getFileDescriptor();
```

Для любознательных: подробнее о привязке данных

Полноценное описание привязки данных выходит за рамки книги. Тем не менее мы отметим хотя бы некоторые интересные моменты.

Лямбда-выражения

Короткие обратные вызовы можно записывать прямо в файле макета в виде *лямбда-выражений*. Они представляют собой упрощенные версии лямбда-выражений Java:

```
<Button
    android:layout_width="match_parent"
    android:layout_height="120dp"
    android:text="@{viewModel.title}"
    android:onClick="@{(view) -> viewModel.onButtonClick()}"
    tools:text="Sound name"/>
```

Как и лямбда-выражения Java 8, они преобразуются в реализации интерфейса, для которого они используются (в данном случае `View.OnClickListener`). Однако, в отличие от лямбда-выражений Java, эти выражения должны использовать строго определенный синтаксис: параметр должен быть заключен в круглые скобки, а в правой части должно стоять ровно одно выражение.

Кроме того, в отличие от лямбда-выражений Java, можно опустить параметры, если вы их не используете. Таким образом, следующая строка тоже нормально работает:

```
    android:onClick="@{() -> viewModel.onButtonClick()}"
```

Другие синтаксические удобства

Также поддерживаются другие удобные синтаксические конструкции привязки данных. Особенно удобна возможность обозначения двойных кавычек обратными апострофами:

```
    android:text="@{`File name: ` + viewModel.title}"
```

Здесь `` `File name` `` означает то же самое, что и `"File name"`.

Выражения привязки также поддерживают оператор выбора с проверкой null:

```
android:text="@{`File name: ` + viewModel.title ?? `No file`}"
```

Если значение title равно null, оператор ?? возвращает значение "No file".

Кроме того, в привязке данных поддерживается автоматическая обработка null. Даже если в приведенном выше коде viewModel содержит null, привязка данных должна предоставить необходимые проверки, которые предотвратят сбой приложения. Вместо ошибки подвыражение viewModel.title вернет "null".

BindingAdapter

По умолчанию привязка данных интерпретирует выражение привязки как вызов метода свойства. Таким образом, конструкция

```
android:text="@{`File name: ` + viewModel.title ?? `No file`}"
```

преобразуется в вызов метода setText(String).

Впрочем, иногда этого оказывается недостаточно, и для какого-то атрибута должно применяться нестандартное поведение. В таких случаях следует написать BindingAdapter:

```
public class BeatBoxBindingAdapter {
    @BindingAdapter("app:soundName")
    public static void bindAssetSound(Button button, String assetFileName) {
        ...
    }
}
```

Просто создайте в любом классе своего проекта статический метод и снабдите его аннотацией @BindingAdapter, передавая имя привязываемого атрибута в параметре (да, это действительно работает). Каждый раз, когда механизму привязки данных потребуется применить этот атрибут, он вызовет ваш статический метод.

Вероятно, вы легко представите одну-две операции, в которых привязка данных использовалась бы со виджетами стандартной библиотеки. Для многих распространенных операций уже определены адаптеры привязки. Например, TextViewBindingAdapter предоставляет дополнительные атрибуты для TextView. Информацию о них можно получить прямо при просмотре исходного кода в Android Studio.

Для любознательных: почему активы, а не ресурсы?

Вообще говоря, в этой главе вместо активов можно было бы использовать ресурсы. В них можно хранить звуки; сохраните файл 79_long_scream.wav в папке res/raw, и вы сможете обратиться к нему по идентификатору R.raw.79_long_scream.

При хранении звуков в ресурсах можно делать все то, что обычно можно делать с ресурсами — например, определять разные звуки для разных ориентаций, языков, версий Android и т. д.

Почему же мы выбрали активы? Дело в том, что в BeatBox используется большое количество звуков: более 20 разных файлов. Работать с ними в системе ресурсов было бы неудобно. Ресурсы не позволяют распространять все звуки в одной папке и не позволяют использовать что-либо, кроме плоской структуры папок.

Как раз для таких ситуаций отлично подходят активы. Они образуют нечто вроде миниатюрной файловой системы, которая упаковывается в состав приложения. С активами вы можете использовать такую структуру папок, какую сочтете более удобной.

Чтобы добавить новый звуковой файл при работе с активами, вам достаточно поместить его в папку sample_sounds. Благодаря этим организационным средствам активы обычно используются в приложениях с большим количеством графики и звуков, например в компьютерных играх.

Для любознательных: «не-активы»?

В классе `AssetManager` определены методы с именем `openNonAssetFd(…)`. Возникает резонный вопрос: зачем классу, предназначенному для работы с активами, методы для работы с «не-активами» (non-assets)? В принципе, мы могли бы ответить: «Не обращайте внимания» и попытаться убедить вас в том, что вы никогда не слышали о существовании `openNonAssetFd(…)`.

Нам не известны никакие причины, по которым вам могли бы понадобиться эти методы, так что для их изучения тоже нет никаких реальных причин.

Однако вы купили нашу книгу, так что мы приведем ответ — просто для полноты картины.

Помните, ранее мы говорили о том, что в Android существуют две разные системы: активы и ресурсы? В системе ресурсов хорошо реализован механизм подбора и сопоставления. Однако некоторые большие ресурсы (как правило, изображения и необработанные звуковые файлы) слишком велики, поэтому фактически они хранятся в виде активов.

Во внутренней реализации Android открывает эти ресурсы для своих целей методами `openNonAsset`, часть из которых недоступна для внешнего пользователя.

Когда вам могут понадобиться эти методы? Насколько нам известно — никогда. По крайней мере, теперь вы знаете почему.

21 Модульное тестирование и воспроизведение звуков

Одно из достоинств архитектуры MVVM заключается в том, что она упрощает важнейшую практическую задачу программирования: *модульное тестирование*. Под этим термином понимается практика написания небольших программ для проверки автономного поведения отдельных частей (модулей) основного приложения. Так как каждый модуль BeatBox является классом, модульные тесты будут тестировать классы.

В этой главе мы наконец-то воспроизведем файлы `.wav`, загруженные в предыдущей главе. В процессе построения и интеграции воспроизведения звука мы напишем модульные тесты для интеграции `SoundViewModel` с BeatBox.

API для работы со звуком в Android в основном является низкоуровневым, но существует инструмент, практически идеально подходящий для нашего приложения: `SoundPool`. Класс `SoundPool` позволяет загрузить в память большой набор звуков и управлять максимальным количеством звуков, воспроизводимых одновременно. Таким образом, если пользователь войдет в азарт и начнет жать на все кнопки одновременно, это не приведет к сбою приложения или чрезмерному расходованию ресурсов на телефоне.

Готовы? Пора начинать.

Создание объекта SoundPool

Начнем с создания объекта `SoundPool` для воспроизведения звуков в `BeatBox`.

Листинг 21.1. Создание объекта SoundPool (Sound.java)

```
public class BeatBox {
    private static final String TAG = "BeatBox";

    private static final String SOUNDS_FOLDER = "sample_sounds";
    private static final int MAX_SOUNDS = 5;

    private AssetManager mAssets;
    private List<Sound> mSounds = new ArrayList<>();
    private SoundPool mSoundPool;
```

```
    public BeatBox(Context context) {
        mAssets = context.getAssets();
        // Этот конструктор считается устаревшим,
        // но он нужен для обеспечения совместимости
        mSoundPool = new SoundPool(MAX_SOUNDS, AudioManager.STREAM_MUSIC, 0);
        loadSounds();
    }
    ...
}
```

В Lollipop появился новый способ создания объектов `SoundPool` с использованием класса `SoundPool.Builder`. Но поскольку `SoundPool.Builder` не доступен в минимальной поддерживаемой версии API 19, мы вместо этого используем более старый конструктор `SoundPool(int, int, int)`.

Первый параметр определяет, сколько звуков может воспроизводиться в любой момент времени. В данном случае передается значение 5. Если пять звуков воспроизводятся одновременно и вы попытаетесь воспроизвести шестой, `SoundPool` прекращает воспроизведение самого старого.

Второй параметр определяет тип аудиопотока, который может воспроизводиться объектом `SoundPool`. В Android поддерживаются разные аудиопотоки, каждый из которых обладает независимыми настройками громкости. Вот почему снижение громкости музыки не приводит к снижению громкости сигналов. За информацией о других вариантах обращайтесь к описанию констант `AUDIO_*` в документации `AudioManager`. `STREAM_MUSIC` устанавливает тот же уровень громкости, что у музыки и игр на устройстве.

А последний параметр? Он задает качество дискретизации. В документации сказано, что этот параметр игнорируется, поэтому в нем можно передать 0.

Загрузка звуков

Следующий шаг — загрузка звуков в `SoundPool`. Главное преимущество класса `SoundPool` перед другими механизмами воспроизведения звука — быстрая реакция: когда вы приказываете ему воспроизвести звук, воспроизведение начинается немедленно, без задержки.

За это приходится расплачиваться необходимостью загрузки звуков в `SoundPool` перед воспроизведением. Каждому загружаемому звуку назначается собственный целочисленный идентификатор. Добавьте в `Sound` поле `mSoundId` и сгенерированные `get-` и `set`-методы для работы с ним.

Листинг 21.2. Добавление поля для идентификатора (Sound.java)

```
public class Sound {
    private String mAssetPath;
    private String mName;
    private Integer mSoundId;
    ...
```

```
    public String getName() {
        return mName;
    }

    public Integer getSoundId() {
        return mSoundId;
    }

    public void setSoundId(Integer soundId) {
        mSoundId = soundId;
    }
}
```

Объявление `mSoundId` с типом `Integer` вместо `int` позволяет представить неопределенное состояние `Sound` — для этого `mSoundId` присваивается `null`.

Теперь можно переходить к загрузке звуков. Добавьте в `BeatBox` метод `load(Sound)` для загрузки `Sound` в `SoundPool`.

Листинг 21.3. Загрузка звуков в SoundPool (BeatBox.java)

```
    private void loadSounds() {
        ...
    }

    private void load(Sound sound) throws IOException {
        AssetFileDescriptor afd = mAssets.openFd(sound.getAssetPath());
        int soundId = mSoundPool.load(afd, 1);
        sound.setSoundId(soundId);
    }

    public List<Sound> getSounds() {
        return mSounds;
    }
}
```

Вызов `mSoundPool.load(AssetFileDescriptor, int)` загружает файл в `SoundPool` для последующего воспроизведения. Для управления звуком, его повторного воспроизведения (или выгрузки) `mSoundPool.load(…)` возвращает идентификатор типа `int`, который сохраняется в только что определенном поле `mSoundId`. А так как вызов `openFd(String)` инициирует `IOException`, `load(Sound)` тоже инициирует `IOException`.

Загрузите все звуки, вызывая метод `load(Sound)` в `BeatBox.loadSounds()`.

Листинг 21.4. Загрузка всех звуков (BeatBox.java)

```
private void loadSounds() {
    ...
    for (String filename : soundNames) {
        try {
            String assetPath = SOUNDS_FOLDER + "/" + filename;
            Sound sound = new Sound(assetPath);
```

```
            load(sound);
            mSounds.add(sound);
        } catch (IOException ioe) {
            Log.e(TAG, "Could not load sound " + filename, ioe);
        }
    }
}
```

Запустите приложение BeatBox и убедитесь в том, что все звуки были загружены правильно. Если при загрузке произошла ошибка, на панели LogCat появятся красные сообщения об исключениях.

Воспроизведение звуков

Остается последний шаг: воспроизвести загруженные звуки. Добавьте в `BeatBox` метод `play(Sound)`.

Листинг 21.5. Воспроизведение звуков (BeatBox.java)

```
public BeatBox(Context context) {
    mAssets = context.getAssets();
    // Этот конструктор считается устаревшим,
    // но он нужен для обеспечения совместимости
    mSoundPool = new SoundPool(MAX_SOUNDS, AudioManager.STREAM_MUSIC, 0);
    loadSounds();
}

public void play(Sound sound) {
    Integer soundId = sound.getSoundId();
    if (soundId == null) {
        return;
    }
    mSoundPool.play(soundId, 1.0f, 1.0f, 1, 0, 1.0f);
}

private void loadSounds() {
```

Прежде чем воспроизводить звук с идентификатором `soundId`, необходимо сначала убедиться в том, что он отличен от `null`. Такое возможно, если объект `Sound` не удалось загрузить.

Если вы уверены, что значение отлично от `null`, воспроизведите звук вызовом `SoundPool.play(int, float, float, int, int, float)`. Параметры содержат соответственно: идентификатор звука, громкость слева, громкость справа, приоритет (игнорируется), признак циклического воспроизведения и скорость воспроизведения. Для полной громкости и нормальной скорости воспроизведения передайте `1.0`. Передача `0` в признаке циклического воспроизведения означает «без зацикливания». (Передайте `-1`, если хотите, чтобы воспроизведение длилось бесконечно долго. Мы считаем, что это только раздражает.)

После написания такого метода вы сможете организовать воспроизведение звука при нажатии одной из кнопок. Мы выполним эту интеграцию по принципу «сначала тесты» — иначе говоря, сначала пишется модульный тест для неудачного случая, и только потом реализуется интеграция, обеспечивающая его прохождение.

Зависимости при тестировании

Прежде чем писать тест, необходимо добавить в среду тестирования пару инструментов: Mockito и Hamcrest. Mockito — фреймворк Java, упрощающий создание простых тестовых объектов. Эти объекты обеспечивают изоляцию тестов `SoundViewModel`, чтобы тестирование разных объектов не проводилось одновременно.

Hamcrest — библиотека для проверки условий в коде и инициирования сбоев в случае нарушения этих условий. Hamcrest позволяет убедиться в том, что ваш код работает так, как вы ожидаете. В тестовых построениях вам понадобятся только эти две библиотеки, поэтому мы добавим их в качестве тестовых зависимостей. Щелкните правой кнопкой мыши на модуле app и выберите команду Open Module Settings.

Выберите вкладку Dependencies в верхней части экрана, щелкните на кнопке + в нижней части диалогового окна и выберите зависимость Library dependency. Введите строку `«mockito»` и нажмите Return (рис. 21.1).

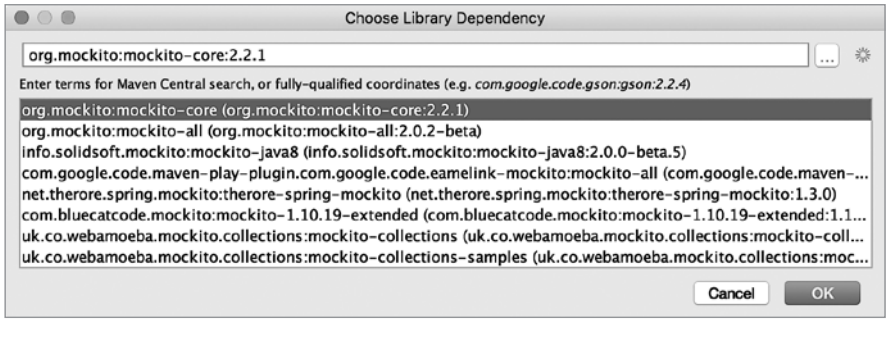


Рис. 21.1. Импортирование Mockito

Выберите зависимость org.mockito:mockito-core и щелкните на кнопке OK. Повторите процесс для Hamcrest: проведите поиск по строке hamcrest-junit и выберите org.hamcrest:hamcrest-junit.

После этого в списке зависимостей появляются две новые зависимости. Справа от зависимостей mockito-core и hamcrest-junit находятся раскрывающийся список для выбора области действия зависимости. Вы сможете выбрать только интеграционную область действия, так что файл build.gradle придется изменять вручную.

Откройте файл build.gradle из модуля app и замените в директиве `dependency` область действия `compile` на `testCompile`.

Листинг 21.6. Изменение области действия зависимости Mockito (app/build.gradle)

```
dependencies {
    compile fileTree(include: ['*.jar'], dir: 'libs')
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
    compile 'com.android.support:appcompat-v7:24.2.0'
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:recyclerview-v7:24.2.0'
    compile testCompile 'org.mockito:mockito-core:2.2.1'
    compile testCompile 'org.hamcrest:hamcrest-junit:2.0.0.0'
}
```

Область действия `testCompile` означает, что эти две зависимости будут включаться только в тестовые построения вашего приложения. Таким образом предотвращается загромождение APK неиспользуемым кодом.

Создание класса теста

Модульные тесты удобнее всего создавать при помощи тестового фреймворка. Фреймворк упрощает написание и запуск пакетов тестов и просмотр их вывода в Android Studio.

Практически все программисты используют при Android-программировании тестовый фреймворк JUnit. В JUnit предусмотрены удобные средства интеграции с Android Studio. Работа начинается с создания класса, в котором должны находиться тесты JUnit. Для этого откройте файл SoundViewModel.java и нажмите Command+Shift+T (Ctrl+Shift+T). Android Studio пытается перейти к классу теста, связанному с тем классом, который вы просматриваете. Если класс теста не найден (как в нашем случае), вам будет предложено создать новый класс теста (рис. 21.2).

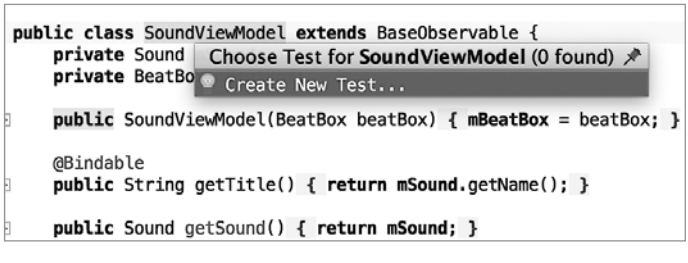


Рис. 21.2. Попытка открытия класса теста

Выберите команду Create New Test..., чтобы создать новый класс теста. Выберите тестовую библиотеку JUnit и установите флажок setUp/@Before. Оставьте остальным полям значения по умолчанию (рис. 21.3).

Щелкните на кнопке OK, чтобы перейти к следующему диалоговому окну.

Остается выбрать тип класса теста. Тесты в папке androidTest относятся к категории *интеграционных тестов*. Интеграционные тесты работают на устройстве Android или в эмуляторе. Преимущество такого решения заключается в том, что вы можете протестировать любые аспекты поведения вашего приложения во время выполнения; недостаток — в том, что тесты должны строить и запускать APK на устройстве, из-за чего их выполнение может занимать много времени. (За дополнительной информацией об интеграционных тестах обращайтесь к разделу «Для любознательных: Espresso и интеграционное тестирование».)

Тесты в папке test относятся к категории *модульных тестов*. Модульные тесты выполняются на локальной машине и не требуют наличия исполнительной среды Android. Отказ от балласта ускоряет их выполнение.

Модульные тесты содержат минимум тестового кода — тест одного компонента. Для их запуска не требуется все приложение или устройство, и они должны выполняться достаточно быстро для многократного проведения тестирования в ходе работы. Выберите папку test для своего класса теста (рис. 21.4) и щелкните на кнопке OK.

Рис. 21.3. Создание нового класса теста

Рис. 21.4. Выбор целевого каталога

Подготовка теста

На следующем шаге строится тест `SoundViewModel`. Шаблон начинается с вызова единственного метода с именем `setUp()`:

Листинг 21.7. Пустой класс теста (SoundViewModelTest.java)

```
public class SoundViewModelTest {
    @Before
    public void setUp() throws Exception {

    }
}
```

(Этот класс находится в группе test в модуле app.)

Для большинства объектов тест должен делать одно и то же: строить экземпляр объекта для тестирования и создавать другие объекты, от которых зависит этот объект. Вместо того, чтобы писать одинаковый код для всех тестов, JUnit предоставляет аннотацию с именем `@Before`. Код, содержащийся в методе с пометкой `@Before`, будет выполнен один раз перед выполнением каждого теста. По действующим соглашениям большинство классов модульных тестов содержит один метод `setUp()` с пометкой `@Before`.

Фиктивные зависимости

Внутри метода `setUp()` строится экземпляр `SoundViewModel` для тестирования. Для этого понадобится экземпляр `BeatBox`, потому что `SoundViewModel` получает `BeatBox` в аргументе конструктора.

В своем приложении вы создаете экземпляр `BeatBox`... просто берете и создаете:

```
SoundViewModel viewModel = new SoundViewModel(new BeatBox());
```

Но если вы попытаетесь действовать так в модульном тесте, возникает проблема: если код `BeatBox` неработоспособен, то тесты, написанные вами в `SoundViewModel` и использующие `BeatBox`, тоже могут «сломаться», а это нежелательно. Модульные тесты `SoundViewModel` не должны проходить только в том случае, если ошибка допущена в классе `SoundViewModel`.

Проблема решается использованием фиктивного объекта `BeatBox`. Он представляет собой субкласс `BeatBox`, который содержит те же методы, что и `BeatBox`, но ни один из этих методов ничего не делает. В этом случае тест `SoundViewModel` может проверить, что `SoundViewModel` использует `BeatBox` правильно, и никак не зависит от особенностей внутреннего строения `BeatBox`.

Чтобы создать фиктивный объект с Mockito, вызовите статический метод `mock(Class)` и передайте ему класс, для которого создается фиктивный объект. Создайте фиктивный объект для `BeatBox` и поле для его хранения.

Листинг 21.8. Создание фиктивного объекта BeatBox (SoundViewModelTest.java)

```
public class SoundViewModelTest {
    private BeatBox mBeatBox;

    @Before
    public void setUp() throws Exception {
        mBeatBox = mock(BeatBox.class);
    }
}
```

Метод `mock(Class)` необходимо будет импортировать, как и ссылку на класс. Метод автоматически создаст фиктивную версию `BeatBox` за вас. Удобно!

Теперь, когда у вас имеется фиктивная зависимость, завершите создание `SoundViewModel`. Создайте объект `SoundViewModel` и используемый им объект `Sound`. (`Sound` — простой объект данных, не обладающий поведением, поэтому для него фиктивный объект можно не создавать — это не несет в себе ни малейшего риска.)

Листинг 21.9. Создание тестового объекта SoundViewModel (SoundViewModelTest.java)

```
public class SoundViewModelTest {
    private BeatBox mBeatBox;
    private Sound mSound;
    private SoundViewModel mSubject;

    @Before
    public void setUp() throws Exception {
        mBeatBox = mock(BeatBox.class);
        mSound = new Sound("assetPath");
        mSubject = new SoundViewModel(mBeatBox);
        mSubject.setSound(mSound);
    }
}
```

В любом другом контексте переменной для `SoundViewModel` мы бы присвоили имя `mSoundViewModel`. Однако в данном случае ей было присвоено имя `mSubject`. Мы в Big Nerd Ranch предпочитаем использовать это соглашение по двум причинам:

- Оно ясно показывает, что тестируется объект `mSubject` (а все остальные — нет).
- Если какие-либо методы `SoundViewModel` будут перемещены в другой класс (допустим, `BeatBoxSoundViewModel`), тестовые методы можно будет скопировать без переименования `mSoundViewModel` в `mBeatBoxSoundViewModel`.

Написание тестов

Итак, метод `setUp()` готов; можно переходить к написанию тестов. Тест представляет собой метод класса теста, помеченный аннотацией `@Test`.

Начните с написания теста, который проверяет существующее поведение в `SoundViewModel`: свойство `getTitle()` связывается со свойством `Sound getName()` объекта `Sound`. Напишите метод для тестирования этого поведения.

Листинг 21.10. Тестирование свойства title (SoundViewModelTest.java)

```
@Before
public void setUp() throws Exception {
    mBeatBox = mock(BeatBox.class);
    mSound = new Sound("assetPath");
    mSubject = new SoundViewModel(mBeatBox);
    mSubject.setSound(mSound);
}

@Test
public void exposesSoundNameAsTitle() {
    assertThat(mSubject.getTitle(), is(mSound.getName()));
}
```

Два метода выделяются красным цветом: `assertThat(…)` и `is(…)`. Нажмите Option+Return (Alt+Enter) в `assertThat(…)` и выберите команду Static import method…,

после чего выберите `MatcherAssert.assertThat(…)` из hamcrest-core-1.3. Сделайте то же самое для метода `is(…)`, но на этот раз выберите `Is.is` из hamcrest-core-1.3.

Тест использует проверку `is(…)` Hamcrest с методом `assertThat(…)` JUnit. Если два метода возвращают разные значения, тест не пройдет.

Чтобы выполнить все модульные тесты, щелкните правой кнопкой мыши на app/java/com.bignerdranch.android.beatbox (test) и выберите команду Run 'Tests in 'beatbox''. На экране появляется результат (рис. 21.5).

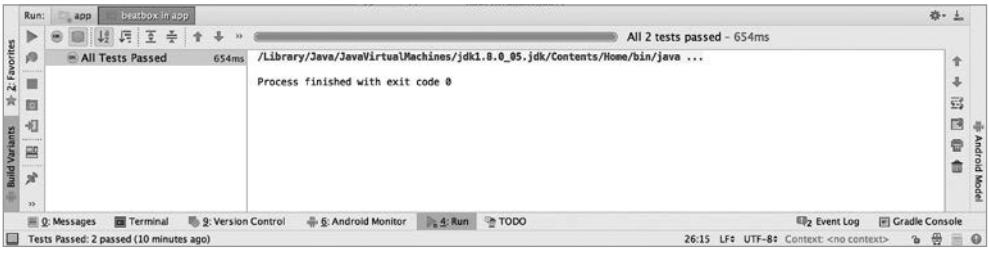


Рис. 21.5. Тесты прошли успешно

По умолчанию в результатах тестов выводится информация только о тестах, которые не прошли, потому что только эти тесты представляют интерес для разработчика. Приведенный результат означает, что все прошло замечательно — тесты выполнены, и выполнены успешно.

Взаимодействия тестовых объектов

А теперь займемся настоящей работой: организации взаимодействия между `SoundViewModel` и новым методом `BeatBox.play(Sound)`. Обычно для этого пишется тест, который показывает, какое поведение ожидается от нового метода, причем делается это до написания самого метода. Мы напишем новый метод класса `SoundViewModel` с именем `onButtonClicked()`, который вызывает `BeatBox.play(Sound)`. Напишите тестовый метод, который вызывает `onButtonClicked()`.

Листинг 21.11. Написание теста для onButtonClicked() (SoundViewModelTest.java)

```
    @Test
    public void exposesSoundNameAsTitle() {
        assertThat(mSubject.getTitle(), is(mSound.getName()));
    }

    @Test
    public void callsBeatBoxPlayOnButtonClicked() {
        mSubject.onButtonClicked();
    }
}
```

Метод еще не существует, поэтому он выделяется красным цветом. Наведите на него курсор и нажмите Option+Return (Alt+Enter). Выберите команду Create method 'onButtonClicked'; метод будет сгенерирован автоматически.

Листинг 21.12. Создание onButtonClicked() (SoundViewModel.java)

```
    public void setSound(Sound sound) {
        mSound = sound;
        notifyChange();
    }

    public void onButtonClicked() {

    }
}
```

Пока оставьте метод пустым и нажмите Command+Shift+T (Ctrl+Shift+T), чтобы вернуться в `SoundViewModelTest`.

Тест вызывает метод, но он также должен проверить, что метод делает то, что положено: вызывает `BeatBox.play(Sound)`. Mockito поможет вам в решении этой несколько странной задачи. Все фиктивные объекты Mockito следят за тем, какие методы вызывались и какие параметры передавались при каждом вызове. Метод Mockito `verify(Object)` может проверить, были ли вызваны эти методы так, как вы ожидали.

Вызовите `verify(Object)`, чтобы убедиться в том, что `onButtonClicked()` вызывает `BeatBox.play(Sound)` с объектом `Sound`, связанным с `SoundViewModel`.

Листинг 21.13. Проверка вызова BeatBox.play(Sound) (SoundViewModelTest.java)

```
        assertThat(mSubject.getTitle(), is(mSound.getName()));
    }

    @Test
    public void callsBeatBoxPlayOnButtonClicked() {
        mSubject.onButtonClicked();

        verify(mBeatBox).play(mSound);
    }
}
```

Метод `verify(Object)` использует динамический интерфейс по аналогии с классом `AlertDialog.Builder`, который использовался ранее в книге. Он является сокращением для следующего кода:

```
verify(mBeatBox);
mBeatBox.play(mSound);
```

Вызов `verify(mBeatBox)` означает: «Я хочу проверить, что для `mBeatBox` был вызван метод». Следующий вызов метода интерпретируется так: «Проверить, что этот метод был вызван именно так». Таким образом, вызов `verify(…)` означает: «Проверить, что метод `play(…)` был вызван для `mBeatBox` с передачей `mSound` в параметре».

Конечно, ничего подобного не происходило. Код `SoundViewModel.onButtonClicked()` пуст, так что метод `mBeatBox.play(Sound)` не вызывался. А следовательно, тест не пройдет. При опережающем написании тестов это хорошо — если тест проходит с первого раза, то он ничего не проверяет.

Запустите тест и убедитесь в том, что он не проходит. Используйте действия, описанные выше, или нажмите клавиши Command+R (Ctrl+R) для повторения последней выполненной команды запуска. Результат показан на рис. 21.6.

Рис. 21.6. Тест не прошел

В результатах указано, что тест ожидает вызов `mBeatBox.play(Sound)`, но не получает его:

```
Wanted but not invoked:
beatBox.play(
    com.bignerdranch.android.beatbox.Sound@64cd705f
);
-> at ….callsBeatBoxPlayOnButtonClicked(SoundViewModelTest.java:28)
```

Во внутренней реализации `verify(Object)` проверяет условие, как и `assertThat(…)`. Условие оказалось не выполненным, тест не прошел, и был выдан результат с описанием проблемы.

Теперь нужно привести тест в порядок. Реализуйте `onButtonClicked()`, чтобы метод делал то, чего от него ожидает тест.

Листинг 21.14. Реализация onButtonClicked() (SoundViewModel.java)

```
    public void setSound(Sound sound) {
        mSound = sound;
        notifyChange();
    }

    public void onButtonClicked() {
        mBeatBox.play(mSound);
    }
}
```

Снова выполните тест. На этот раз зеленый индикатор указывает на то, что все тесты прошли успешно (рис. 21.7).

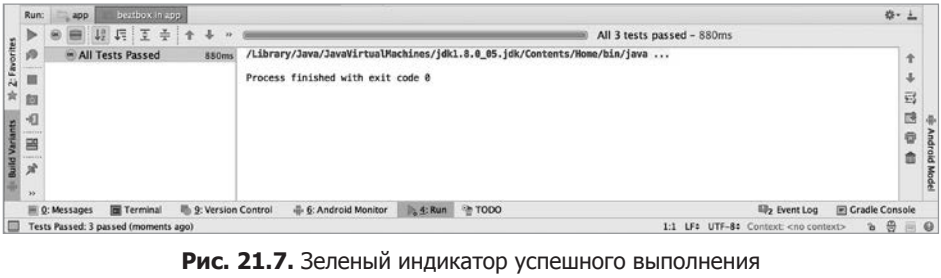


Рис. 21.7. Зеленый индикатор успешного выполнения

Обратные вызовы привязки данных

Чтобы кнопки заработали, осталось сделать последний шаг: связать `onButtonClicked()` с кнопкой.

По аналогии с тем, как вы использовали привязку данных для размещения данных в пользовательском интерфейсе, вы также можете воспользоваться ей для подключения слушателей щелчков и т. д. при помощи лямбда-выражений. (Если вы забыли, что это такое, обратитесь к разделу «Лямбда-выражения» главы 20.)

Добавьте выражение обратного вызова для связывания кнопки с `SoundViewModel.onButtonClicked()`.

Листинг 21.15. Подключение кнопки (list_item_sound.xml)

```
<Button
    android:layout_width="match_parent"
    android:layout_height="120dp"
    android:onClick="@{() -> viewModel.onButtonClicked()}"
    android:text="@{viewModel.title}"
    tools:text="Sound name"/>
```

При следующем запуске BeatBox кнопки будут успешно воспроизводить звуки. При этом попытка запустить BeatBox зеленой кнопкой снова приведет к выполнению тестов. Дело в том, что щелчок правой кнопкой мыши изменил *конфигурацию выполнения* — настройку, которая определяет, что должна делать среда Android Studio при нажатии кнопки запуска.

Чтобы запустить приложение BeatBox, щелкните на селекторе конфигурации выполнения рядом с кнопкой запуска и вернитесь к конфигурации запуска приложения (рис. 21.8).

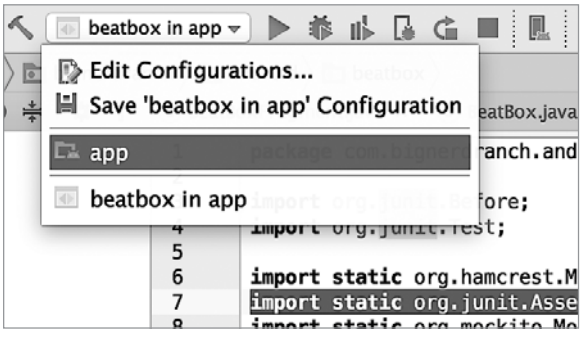


Рис. 21.8. Изменение конфигурации выполнения

Запустите BeatBox и поэкспериментируйте с нажатием кнопок. Приложение будет издавать всевозможные угрожающие звуки. Не пугайтесь — оно для этого и создавалось.

Выгрузка звуков

Приложение работает, но мы еще должны прибрать за собой. Сознательное приложение должно освободить ресурсы `SoundPool` вызовом `SoundPool.release()` после завершения работы. Добавьте соответствующий метод `BeatBox.release()`.

Листинг 21.16. Освобождение SoundPool (BeatBox.java)

```
public class BeatBox {
    ...
    public void play(Sound sound) {
        ...
    }

    public void release() {
        mSoundPool.release();
    }
    ...
}
```

Добавьте соответствующий метод `BeatBox.release()` в `BeatBoxFragment`.

Листинг 21.17. Освобождение BeatBox (BeatBoxFragment.java)

```
public class BeatBoxFragment extends Fragment {
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        ...
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        mBeatBox.release();
    }
    ...
}
```

Запустите приложение и убедитесь в том, что оно правильно работает с новым методом `release()`.

Повороты и преемственность объектов

Ваше приложение ведет себя цивилизованно, и это хорошо. К сожалению, оно перестало правильно обрабатывать повороты. Попробуйте воспроизвести звук 69_ohm-loko и повернуть устройство: воспроизведение неожиданно прерывается.

(Если этого не происходит, убедитесь в том, что приложение было построено и запущено с последней реализацией `onDestroy()`.)

Проблема заключается в следующем: при повороте `BeatBoxActivity` уничтожается. Заодно `FragmentManager` также уничтожает и `BeatBoxFragment`. При этом вызываются методы прекращения жизненного цикла `BeatBoxFragment`: `onPause()`, `onStop()` и `onDestroy()`. В `BeatBoxFragment.onDestroy()` вызывается `BeatBox.release()`, что приводит к освобождению `SoundPool` и остановке воспроизведения.

Вы уже видели, как экземпляры `Activity` и `Fragment` «умирают» при поворотах; тогда проблема решалась при помощи `onSaveInstanceState(Bundle)`. Однако на этот раз такое решение не сработает, потому что оно основано на сохранении и восстановлении данных `Parcelable` в объекте `Bundle`.

`Parcelable`, как и `Serializable`, представляет собой API для сохранения объекта в потоке байтов. В Java объекты сохраняются либо посредством включения в `Bundle`, либо пометкой объекта `Serializable`, либо реализацией интерфейса `Parcelable`. Какой бы способ вы ни выбрали, всегда действует один принцип: ни один из этих инструментов не следует использовать, если объект не является *сохраняемым* (stashable).

Чтобы вы лучше поняли, что имеется в виду, представьте, что вы смотрите телепередачу с другом. Вы можете записать канал, который вы смотрите, уровень громкости, настройку цветов и т. д. Даже если сработает пожарная сигнализация и выключится электричество, вы сможете позднее прочитать записанные данные и продолжить просмотр так, словно ничего не произошло.

Таким образом, конфигурация телевизора является сохраняемой. С другой стороны, время, проведенное за просмотром, для сохранения не пригодно: если выключится питание, сеанс завершается. Позднее вы можете вернуться и создать новый сеанс, но просмотр будет прерван, что бы вы ни делали. Следовательно, сеанс *не является* сохраняемым.

Некоторые части BeatBox являются сохраняемыми: например, все содержимое `Sound` сохраняемо. Однако `SoundPool` больше напоминает сеанс просмотра. Да, вы можете создать новый объект `SoundPool`, в котором загружены те же звуки, что и в старом. Вы даже можете снова начать воспроизведение с того места, в котором оно было прервано. Однако при этом воспроизведение всегда будет ненадолго прерываться, что бы вы ни делали. Это означает, что класс `SoundPool` не является сохраняемым.

Несохраняемость имеет склонность к распространению. Если несохраняемый объект критичен для деятельности другого объекта, то и другой объект, скорее всего, не является сохраняемым. Класс `BeatBox` имеет ту же цель, что и `SoundPool`: он воспроизводит звуки. Из этого следует, что класс `BeatBox` не является сохраняемым. (Нам очень жаль.)

Обычный механизм `savedInstanceState` работает с сохраняемыми данными, но класс `BeatBox` таковым не является. Экземпляр `BeatBox` должен оставаться постоянно доступным при создании и уничтожении вашей активности.

Что же делать?

Удержание фрагмента

К счастью, у фрагментов существует свойство `retainInstance`, которое позволит сохранить экземпляр `BeatBox` между изменениями конфигурации. Переопределите `BeatBoxFragment.onCreate(…)` и задайте свойство фрагмента.

Листинг 21.18. Вызов setRetainInstance(true) (BeatBoxFragment.java)

```
public static BeatBoxFragment newInstance() {
    return new BeatBoxFragment();
}

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setRetainInstance(true);

    mBeatBox = new BeatBox(getActivity());
}
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
```

По умолчанию свойство `retainInstance` у фрагментов равно `false`. Это означает, что фрагмент не удерживается, а уничтожается и создается заново при поворотах вместе с активностью-хостом. Вызов `setRetainInstance(true)` удерживает фрагмент. Такой фрагмент не уничтожается вместе с активностью, а сохраняется и передается новой активности.

При удержании фрагмента можно рассчитывать на то, что все его переменные экземпляров (такие, как `mBeatBox`) сохранят прежние значения. Когда вы обратитесь к ним, они просто будут находиться на своих местах.

Снова запустите приложение BeatBox. Воспроизведите звук `69_ohm-loko`, поверните устройство и убедитесь в том, что воспроизведение продолжается беспрепятственно.

Повороты и удержание фрагментов

Давайте повнимательнее разберемся с тем, как работают удерживаемые фрагменты. В удержании используется тот факт, что представление фрагмента может быть уничтожено и создано заново без уничтожения самого фрагмента.

Во время изменения конфигурации `FragmentManager` сначала уничтожает представления фрагментов из своего списка. Представления фрагментов всегда уничтожаются и создаются заново при изменении конфигурации по тем же причинам, по которым уничтожаются и создаются заново представления активностей: в новой конфигурации могут быть задействованы новые ресурсы. На случай появления ресурсов, лучше соответствующих новой конфигурации, представление строится заново.

Затем `FragmentManager` проверяет свойство `retainInstance` каждого фрагмента. Если оно равно `false` (по умолчанию), то `FragmentManager` уничтожает экземпляр фрагмента. Фрагмент и его представление будут созданы заново новым экземпляром `FragmentManager` новой активности «по ту сторону» (рис. 21.9).

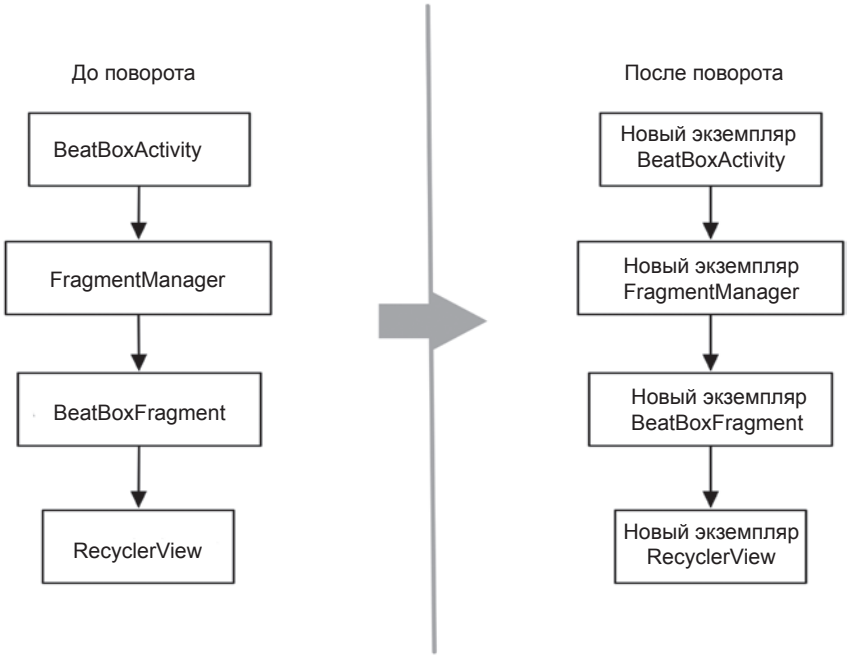


Рис. 21.9. Фрагмент пользовательского интерфейса при выполнении поворота (поведение по умолчанию)

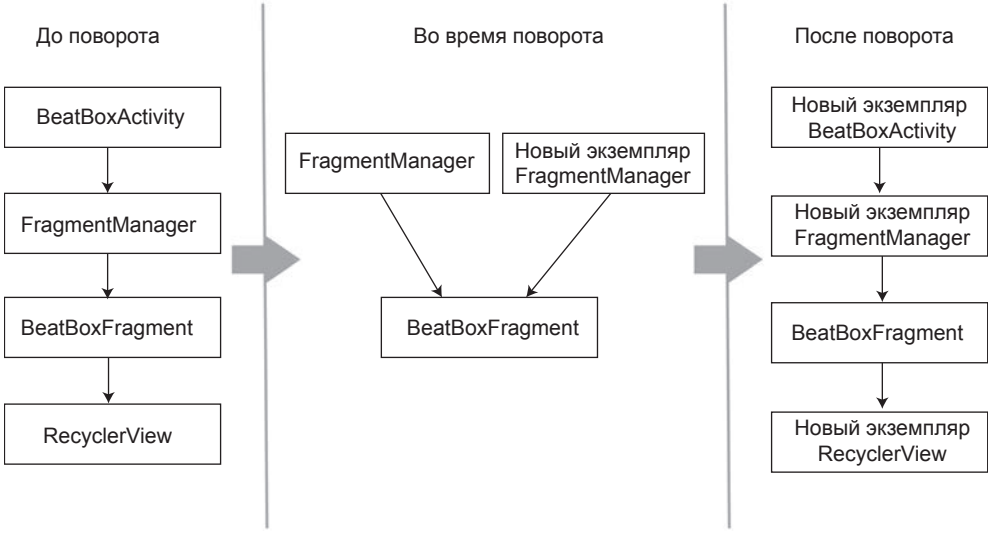


Рис. 21.10. Фрагмент пользовательского интерфейса при выполнении поворота (с удержанием)

С другой стороны, если свойство `retainInstance` равно `true`, то уничтожается только представление фрагмента, но не сам фрагмент. При создании новой активности новый экземпляр `FragmentManager` находит удерживаемый фрагмент и создает заново его представление (рис. 21.10).

Удерживаемый фрагмент не уничтожается, а *отсоединяется* (detached) от гибнущей активности. При этом фрагмент переходит в удерживаемое состояние: фрагмент продолжает существовать, но не имеет активности-хоста (рис. 21.11).

Переход в состояние удержания происходит только при выполнении двух условий:

- Для фрагмента был вызван метод `setRetainInstance(true)`.
- Активность-хост уничтожается при изменении конфигурации (обычно при повороте).

Фрагмент находится в состоянии удержания в течение очень короткого времени — между отсоединением от старой активности и присоединением к новой, немедленно создаваемой активности.

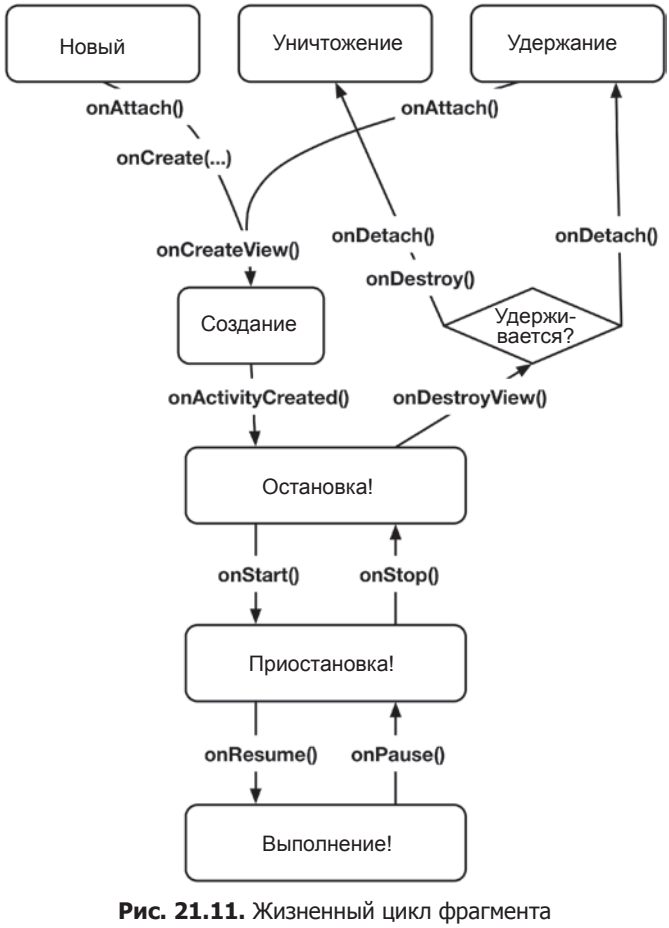


Рис. 21.11. Жизненный цикл фрагмента

Для любознательных: когда удерживать фрагменты

Удерживаемые фрагменты: удобно, верно? Да! Очень удобно. На первый взгляд они решают все проблемы, возникающие при уничтожении активностей и фрагментов при поворотах. При изменении конфигурации устройства создание нового представления обеспечивает подбор наиболее подходящих ресурсов, а у вас появляется простой способ сохранения данных и объектов.

Возникает вопрос: почему бы не удерживать все фрагменты или почему фрагменты не удерживаются по умолчанию? Тем не менее мы рекомендуем использовать этот механизм только при крайней необходимости.

Во-первых, удерживаемые фрагменты попросту сложнее обычных. Когда с ними возникают проблемы, вам будет труднее разобраться в причинах происходящего. Программы всегда оказываются более сложными, чем нам хотелось бы, и когда можно обойтись без лишних сложностей — лучше так и поступить.

Во-вторых, фрагменты, обрабатывающие повороты с использованием сохранения состояния экземпляров, работают в любых ситуациях жизненного цикла, тогда как удерживаемые фрагменты работают только в том случае, когда активность уничтожается при изменении конфигурации. Если ваша активность уничтожается из-за того, что ОС понадобилось освободить память, то все удерживаемые фрагменты тоже будут уничтожены; это может привести к потере данных.

Для любознательных: Espresso и интеграционное тестирование

Как вы помните, упоминавшийся ранее тест `SoundViewModelTest` был модульным тестом. Также вам предоставлялась другая возможность: создать интеграционный тест. Собственно, что это — интеграционный тест?

В модульных тестах тестируемой единицей является отдельный класс. В интеграционных тестах тестируется все приложение. Обычно они пишутся по принципу «экран за экраном». Например, сначала вы проверяете, что при открытии экрана `BeatBoxActivity` на первой кнопке выводится имя первого файла из sample_sounds: 65_cjipie.

Интеграционные тесты должны проходить тогда, когда *поведение* приложения соответствует вашим ожиданиям, а не тогда, когда его *реализация* соответствует вашим ожиданиям. Изменение идентификатора кнопки не повлияет на работу приложения, но если вы напишете интеграционный тест «Вызвать `findViewById(R.id.button)` и убедиться в том, что на найденной кнопке выводится правильный текст», этот тест не пройдет. Итак, вместо стандартных средств Android (таких, как `findViewById(int)`) интеграционные тесты чаще пишутся во фреймворках тестирования пользовательского интерфейса, в которых проще выражаются запросы вида «Убедиться в том, что на экране присутствует кнопка с текстом, который на ней должен выводиться».

Espresso — фреймворк компании Google для тестирования пользовательского интерфейса приложений Android. Чтобы включить его, добавьте `com.android.support.test.espresso:espresso-core` как артефакт `androidTestCompile` в файл app/build.gradle.

После добавления в состав зависимостей вы можете использовать Espresso для проверки условий относительно запущенной активности. Следующий пример показывает, как проверить существование на экране представления с первым именем из sample_sounds:

```
@RunWith(AndroidJUnit4.class)
public class BeatBoxActivityTest {
    @Rule
    public ActivityTestRule<BeatBoxActivity> mActivityRule =
            new ActivityTestRule<>(BeatBoxActivity.class);

    @Test
    public void showsFirstFileName() {
        onView(withText("65_cjipie"))
                .check(matches(anything()));
    }
}
```

Работа этого кода основана на паре аннотаций. Аннотация `@RunWith(AndroidJUnit4.class)` указывает на то, что перед нами тест Android, который должен работать с активностями и другими средствами времени выполнения Android. После этого аннотация `@Rule` у `mActivityRule` сообщает JUnit о необходимости запускать экземпляр `BeatBoxActivity` перед запуском каждого теста.

После предварительной подготовки вы сможете проверять условия, относящиеся к `BeatBoxActivity`, в ваших тестах. В `showsFirstFileName()` строка `onView(withText("65_cjipie"))` находит представление с текстом `"65_cjipie"` для выполнения тестовой операции. Вызов `check(matches(anything()))` проверяет, что такое представление существует, — если представления с таким текстом нет, то проверка завершается неудачей. Метод `check(…)` используется Espresso для проверки условий типа `assertThat(…)` относительно представлений.

Часто требуется щелкнуть на представлении и проверить некоторое условие относительно результата щелчка. Также можно воспользоваться Espresso для моделирования щелчков на представлениях или других взаимодействий с ними:

```
onView(withText("65_cjipie"))
        .perform(click());
```

Когда вы взаимодействуете с представлением, Espresso дождется бездействия приложения перед тем, как продолжать тестирование. В Espresso существуют встроенные средства проверки завершения обновлений пользовательского интерфейса, но если вам потребуется более продолжительное ожидание — используйте

субкласс `IdleResource` для передачи Espresso сигнала о том, что приложение еще не завершило свои операции.

За дополнительной информацией о том, как использовать Espresso для манипуляций с пользовательским интерфейсом и его тестирования, обращайтесь к документации Espresso по адресу *google.github.io/android-testing-support-library/docs/espresso*.

Интеграционные и модульные тесты служат разным целям. Как правило, разработчики предпочитают начинать с модульных тестов, потому что они выполняются достаточно быстро; их можно запускать без долгих раздумий, что упрощает выработку привычки. Интеграционные тесты выполняются достаточно долго, что не позволяет выполнять их с такой частотой. При этом каждый вид тестов дает важную информацию о состоянии вашего приложения под особым углом, поэтому действительно хорошие разработчики выполняют тесты обоих типов.

Для любознательных: фиктивные объекты и тестирование

Фиктивные объекты играют в интеграционном тестировании совершенно иную роль, чем в модульном тестировании. Они выдают себя за другие несвязанные компоненты и тем самым обеспечивают изоляцию тестируемых компонентов. Модульные тесты тестируют отдельные классы; каждый класс обладает собственными специфическими зависимостями, поэтому каждый класс теста использует свой набор фиктивных объектов. Так как фиктивные объекты различаются между классами тестов, а поведение особой роли не играет, для модульных тестов замечательно подходят фреймворки, упрощающие создание простых фиктивных объектов (такие, как Mockito).

С другой стороны, интеграционные тесты предназначены для тестирования всего приложения как единого целого. Вместо того, чтобы обеспечивать изоляцию компонентов приложения, фиктивные объекты используются для изоляции приложения от всех внешних объектов, с которыми может взаимодействовать приложение, — например, моделированием веб-службы с фиктивными данными и ответами. В приложении BeatBox можно было бы предоставить фиктивный объект `SoundPool`, который сообщал бы о воспроизведении конкретного звукового файла. Так как фиктивные объекты имеют большие размеры и совместно используются многими тестами, и поскольку они чаще применяются для реализации фиктивного поведения, при интеграционном тестировании лучше избегать использования автоматизированных фреймворков и вместо этого писать фиктивные объекты вручную.

В любом случае действует один принцип: моделирование сущностей на границе тестируемого компонента. Он определяет область действия теста и гарантирует, что тест не будет проходить только в случае неработоспособности самого компонента.

Упражнение. Управление скоростью воспроизведения

В этом упражнении вы добавите в BeatBox элемент управления скоростью воспроизведения, который значительно расширит репертуар воспроизводимых звуков (рис. 21.12). Включите в `BeatBoxFragment` виджет `SeekBar` (см. документацию по адресу *developer.android.com/reference/android/widget/SeekBar.html*) для управления значением скорости, передаваемым вызову `play(int, float, float, int, int, float)` класса `SoundPool`.

Рис. 21.12. BeatBox с возможностью управления скоростью воспроизведения

22 Стили и темы

Приложение BeatBox эффектно звучит; пора придать ему столь же эффектный внешний вид.

До настоящего момента в BeatBox использовалось стандартное стилевое оформление пользовательского интерфейса. Кнопки — стандартные. Цвета — стандартные. Приложение ничем не выделяется на общем фоне. У него нет своего «лица».

К счастью, ситуацию можно изменить. У нас для этого имеются подходящие технологии.

На рис. 22.1 изображено заметно улучшенное (или по крайней мере более стильное) приложение BeatBox.

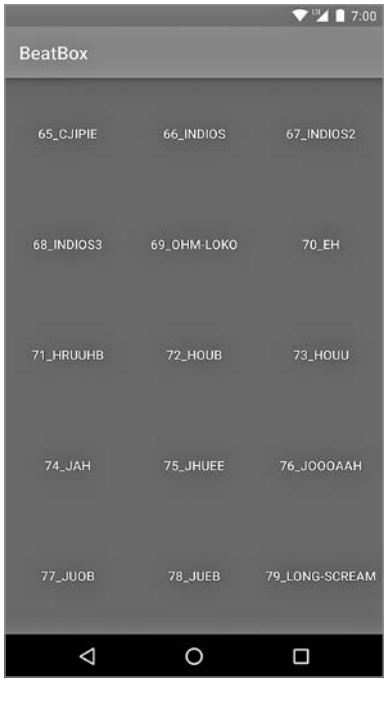


Рис. 22.1. Приложение BeatBox с новым оформлением

Цветовые ресурсы

Начнем с определения нескольких цветов, которые будут использоваться в этой главе. Создайте файл colors.xml в папке res/values.

Листинг 22.1. Определение цветов (res/values/colors.xml)

```
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>

    <color name="red">#F44336</color>
    <color name="dark_red">#C3352B</color>
    <color name="gray">#607D8B</color>
    <color name="soothing_blue">#0083BF</color>
    <color name="dark_blue">#005A8A</color>
</resources>
```

Цветовые ресурсы удобны тем, что вы можете в одном месте определить цветовые значения, которые затем будут использоваться в разных местах приложения.

Стили

А теперь займемся изменением внешнего вида кнопок BeatBox. *Стиль* (style) представляет собой набор атрибутов, которые могут применяться к виджетам.

Откройте файл res/values/styles.xml и добавьте стиль с именем `BeatBoxButton`. (При создании приложения BeatBox в новый проект должен быть включен встроенный файл styles.xml. Если в вашем проекте его нет, создайте этот файл.)

Листинг 22.2. Добавление стиля (res/values/styles.xml)

```
<resources>

    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

    <style name="BeatBoxButton">
        <item name="android:background">@color/dark_blue</item>
    </style>

</resources>
```

Мы создали стиль с именем `BeatBoxButton`. Этот стиль определяет всего один атрибут `android:background`, которому назначается темно-синий цвет. Вы можете применить этот стиль к любому количеству виджетов, а потом обновить атрибуты всех этих виджетов в одном месте.

Примените стиль `BeatBoxButton` к кнопкам приложения BeatBox.

Листинг 22.3. Использование стиля (res/layout/list_item_sound.xml)

```
<Button
    style="@style/BeatBoxButton"
    android:layout_width="match_parent"
    android:layout_height="120dp"
    android:onClick="@{() -> viewModel.onButtonClicked()}"
    android:text="@{viewModel.title}"
    tools:text="Sound name"/>
```

Запустив BeatBox, вы увидите, что все кнопки окрасились в темно-синий цвет фона (рис. 22.2).

Рис. 22.2. Приложение BeatBox со стилями кнопок

Стиль можно создать для любого набора атрибутов, которые должны многократно использоваться в приложении. Что и говорить, удобно.

Наследование стилей

Стили также поддерживают наследование. Стиль может наследовать и переопределять атрибуты из других стилей.

Создайте новый стиль с именем `BeatBoxButton.Strong`, который наследует от `BeatBoxButton`, но дополнительно выделяет текст полужирным шрифтом.

Листинг 22.4. Наследование от BeatBoxButton (res/values/styles.xml)

```
<style name="BeatBoxButton">
    <item name="android:background">@color/dark_blue</item>
</style>

<style name="BeatBoxButton.Strong">
    <item name="android:textStyle">bold</item>
</style>
```

(Хотя атрибут `android:textStyle` также можно было добавить в стиль `BeatBoxButton` напрямую, мы создали `BeatBoxButton.Strong` для демонстрации механизма наследования стилей.)

Схема формирования имен в данном случае выглядит немного странно. Когда вы присваиваете стилю имя `BeatBoxButton.Strong`, вы тем самым указываете, что он наследует атрибуты от `BeatBoxButton`.

Также существует альтернативный механизм формирования имен при наследовании — с явным указанием родителя при объявлении стиля:

```
<style name="BeatBoxButton">
    <item name="android:background">@color/dark_blue</item>
</style>

<style name="StrongBeatBoxButton" parent="@style/BeatBoxButton">
    <item name="android:textStyle">bold</item>
</style>
```

В приложении BeatBox будет использоваться схема `BeatBoxButton.Strong`.

Обновите файл list_item_sound.xml, чтобы использовать новый стиль с жирным шрифтом.

Листинг 22.5. Применение жирного шрифта (res/layout/list_item_sound.xml)

```
<Button
    style="@style/BeatBoxButton.Strong"
    android:layout_width="match_parent"
    android:layout_height="120dp"
    android:onClick="@{() -> viewModel.onButtonClicked()}"
    android:text="@{viewModel.title}"
    tools:text="Sound name"/>
```

Запустите приложение BeatBox и убедитесь в том, что текст на кнопках действительно выводится жирным шрифтом (рис. 22.3).

Темы

Стили — классная штука. Они позволяют определить набор атрибутов в одном месте, а затем применить их к любому количеству виджетов на ваше усмотрение. Впрочем, у них есть и недостаток: вам придется последовательно применять их

Рис. 22.3. Приложение BeatBox с жирным шрифтом

к каждому виджету. А если вы пишете сложное приложение с множеством кнопок в многочисленных макетах? Добавление стиля `BeatBoxButton` ко всем кнопкам станет весьма масштабной задачей.

На помощь приходят *темы* (themes). Темы идут еще дальше стилей: они, как и стили, позволяют определить набор атрибутов в одном месте, но затем эти атрибуты автоматически применяются во всем приложении. В атрибутах темы могут содержаться ссылки на конкретные ресурсы (например, цвета), а также ссылки на стили. Например, в определении темы можно сказать: «Я хочу, чтобы все кнопки использовали *этот* стиль». И вам не придется отыскивать каждый виджет кнопки и приказывать ему использовать указанную тему.

Изменение темы

При создании приложения BeatBox ему была назначена тема по умолчанию. Откройте файл AndroidManifest.xml и найдите атрибут `theme` в теге `application`.

Листинг 22.6. Тема BeatBox (AndroidManifest.xml)

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bignerdranch.android.beatbox" >

    <application
```

```
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">
        ...
    </application>

</manifest>
```

Атрибут `theme` ссылается на тему с именем `AppTheme`. Тема `AppTheme` была объявлена в файле styles.xml, который мы изменили ранее.

Как видите, тема также является стилем. Однако темы определяют другие атрибуты (вскоре вы убедитесь в этом). Кроме того, определение тем в манифесте наделяет их суперспособностями: именно этот факт позволяет автоматически применить тему в границах целого приложения.

Перейдите к определению темы `AppTheme`, щелкнув с нажатой клавишей Command (или Ctrl в Windows) на `@style/AppTheme`. Android Studio открывает файл res/values/styles.xml.

Листинг 22.7. Тема AppTheme в BeatBox (res/values/styles.xml)

```
<resources>

    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        ...
    </style>

    <style name="BeatBoxButton">
        <item name="android:background">@color/dark_blue</item>
    </style>
    ...
</resources>
```

(На момент написания книги новым проектам, создаваемым в Android Studio, назначалась тема `AppCompat`. Если тема `AppCompat` отсутствует в вашем решении, выполните инструкции из главы 13 и переведите BeatBox на использование библиотеки AppCompat.)

`AppTheme` наследует атрибуты от `Theme.AppCompat.Light.DarkActionBar`. В `AppTheme` можно добавлять или переопределять отдельные значения родительской темы.

Библиотека AppCompat включает три основные темы:

- `Theme.AppCompat` — темная тема;
- `Theme.AppCompat.Light` — светлая тема;
- `Theme.AppCompat.Light.DarkActionBar` — светлая тема с темной панелью инструментов.

Замените родительскую тему на `Theme.AppCompat`, чтобы в BeatBox использовалась базовая темная тема.

Листинг 22.8. Переключение на темную тему (res/values/styles.xml)

```
<resources>

    <style name="AppTheme" parent="Theme.AppCompat~~.Light.DarkActionBar~~">

    </style>
    ...
</resources>
```

Запустите BeatBox и посмотрите, как выглядит темная тема (рис. 22.4).

Рис. 22.4. BeatBox с темной темой

Добавление цветов в тему

Разобравшись с выбором базовой темы, мы переходим к настройке атрибутов темы `AppTheme` приложения BeatBox.

В файле styles.xml содержатся три атрибута вашей темы. Приведите их в соответствие с листингом 22.9.

Листинг 22.9. Изменение атрибутов темы (res/values/styles.xml)

```
<style name="AppTheme" parent="Theme.AppCompat">
    <item name="colorPrimary">@color/~~colorPrimary~~ red</item>
    <item name="colorPrimaryDark">@color/~~colorPrimaryDark~~ dark_red</item>
    <item name="colorAccent">@color/~~colorAccent~~ gray</item>
</style>
```

Эти атрибуты похожи на атрибуты стилей, которыми мы занимались ранее, но они задают другие свойства. Атрибуты стиля задают свойства индивидуального виджета: как, например, `textStyle` при выделении текста кнопки жирным шрифтом. Атрибуты темы имеют более широкую область действия: это свойства, которые задаются на уровне темы и становятся доступными для любых виджетов. Например, панель инструментов обращается к атрибуту темы `colorPrimary` для назначения своего цвета фона.

Назначение этих трех атрибутов приводит к масштабным последствиям. Атрибут `colorPrimary` определяет первичный цвет фирменного стиля вашего приложения. Этот цвет будет использоваться для фона панели инструментов, а также в нескольких других местах.

Атрибут `colorPrimaryDark` используется для окраски строки состояния, отображаемой у верхнего края экрана. Обычно цвет `colorPrimaryDark` представляет собой чуть более темную версию цвета `colorPrimary`. Тематическое оформление строки состояния относится к числу возможностей, добавленных в Android в Lollipop. Помните, что строка состояния будет окрашена в черный цвет на старых устройствах (независимо от настроек темы). На рис. 22.5 показан эффект от назначения этих двух атрибутов темы в приложении BeatBox.

Рис. 22.5. BeatBox с цветовыми атрибутами AppCompat

Наконец, `colorAccent` назначается серый цвет. Цвет `colorAccent` должен контрастировать с атрибутом `colorPrimary`; он используется для формирования оттенка в некоторых виджетах (например, `EditText`).

Атрибут `colorAccent` на внешнем виде приложения BeatBox не отражается, потому что кнопки не поддерживают оттенки. Тем не менее мы все равно задаем `colorAccent`, потому что эти три цветовых атрибута удобнее рассматривать вместе. Запустите приложение BeatBox и понаблюдайте за новыми цветами в действии.

Ваше приложение должно выглядеть так, как показано на рис. 22.5.

Переопределение атрибутов темы

Теперь пора поглубже изучить вопрос и поближе познакомиться с атрибутами тем, которые вы можете переопределять. Учтите, что исследование тем — непростое дело. Почти не существует документации, в которой было бы указано, какие существуют атрибуты, какие из них вы можете переопределять и что делает тот или иной атрибут. Вам придется самостоятельно прокладывать путь в этих джунглях. Хорошо, что у вас есть надежный проводник (эта книга).

Начнем с изменения цвета фона BeatBox посредством модификации темы. Хотя теоретически вы можете открыть файл res/layout/fragment_beat_box.xml и вручную задать атрибут `android:background` для виджета `RecyclerView`, а затем повторить процесс со всеми остальными файлами макетов фрагментов и активностей. Конечно, это будет весьма неэффективно — и не только по затратам времени, но и по затратам труда программиста.

Тема всегда назначает цвет фона. Назначая другой цвет поверх этого, вы выполняете лишнюю работу. Кроме того, дублирование атрибута в приложении усложняет сопровождение кода.

Исследование тем

Вместо этого было бы правильнее переопределить атрибут цвета фона в теме. Чтобы узнать имя атрибута, взгляните, как атрибут задается в родительской теме: `Theme.AppCompat`.

Возникает резонный вопрос: как узнать, какой атрибут нужно переопределить, если я не знаю его имени? Никак. Вы будете читать имена атрибутов, и в какой-то момент у вас блеснет догадка: «А вот это похоже». Вы переопределите атрибут, запустите приложение и будете надеяться, что сделали разумный выбор.

В конечном итоге требуется найти самого первого предка вашей темы: ее прапрапра... в общем, предка неизвестно какого уровня. Для этого вы будете переходить к родителю более высокого уровня, пока не найдете подходящий атрибут.

Откройте файл styles.xml и щелкните на `Theme.AppCompat` с нажатой клавишей Command (или Ctrl в Windows). Посмотрим, как далеко уходит этот лабиринт.

(Если вам не удается переходить по атрибутам тем прямо в Android Studio или вы предпочитаете делать это вне Android Studio, исходные файлы тем Android находятся в каталоге *ваш-каталог-SDK*/platforms/android-24/data/res/values.)

На момент написания книги при этом открывался очень большой файл, в котором выделялась следующая строка:

```
<style name="Theme.AppCompat" parent="Base.Theme.AppCompat" />
```

Тема `Theme.AppCompat` наследует атрибуты от `Base.Theme.AppCompat`. Интересно, что `Theme.AppCompat` не переопределяет никакие атрибуты, а только содержит ссылку на своего родителя.

Щелкните на `Base.Theme.AppCompat` с нажатой клавишей Command. Android Studio сообщит, что тема уточняется по ресурсам. Существует несколько разных версий этой темы в зависимости от используемой версии Android.

Выберите версию values/values.xml; открывается определение `Base.Theme.AppCompat` (рис. 22.6).

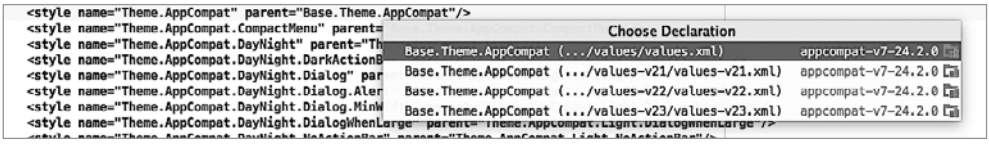


Рис. 22.6. Выбор родителя

(Мы выбрали версию без квалификаторов, потому что BeatBox поддерживает API уровня 19 и выше. Если бы вы выбрали версию v21, то могли бы столкнуться с новыми возможностями, добавленными в API уровня 21.)

```
<style name="Base.Theme.AppCompat" parent="Base.V7.Theme.AppCompat">
</style>
```

`Base.Theme.AppCompat` — еще одна тема, которая существует только ради имени и не переопределяет никакие атрибуты. Вернитесь к родительской теме: `Base.V7.Theme.AppCompat`.

```
<style name="Base.V7.Theme.AppCompat" parent="Platform.AppCompat">
    <item name="windowNoTitle">false</item>
    <item name="windowActionBar">true</item>
    <item name="windowActionBarOverlay">false</item>
    ...
</style>
```

Постепенно приближаемся. Просмотрите список атрибутов `Base.V7.Theme.AppCompat`.

На первый взгляд нет атрибута, который соответствовал бы цели — изменению цвета фона. Перейдите к `Platform.AppCompat`. Вы увидите, что и эта тема уточняется по ресурсам. Выберите версию values/values.xml.

```
<style name="Platform.AppCompat" parent="android:Theme">
    <item name="android:windowNoTitle">true</item>

    <!-- Цвета окна -->
    <item name="android:colorForeground">@color/foreground_material_dark</item>
    <item name="android:colorForegroundInverse">@color/
                                foreground_material_light</item>
    ...
</style>
```

Наконец-то мы видим, что родителем темы `Platform.AppCompat` является `android:Theme`.

Обратите внимание: ссылка на родительскую тему не записывается в виде `Theme`. К ней также добавляется префикс пространства имен `android`.

Считайте, что библиотека AppCompat живет внутри вашего приложения. При построении проекта вы включаете библиотеку AppCompat, которая приносит с собой набор файлов с кодом Java и разметкой XML. Эти файлы практически ничем не отличаются от файлов, которые вы пишете самостоятельно. Если вы захотите сослаться на какой-либо компонент из библиотеки AppCompat, вы делаете это напрямую, используя запись `Theme.AppCompat`, потому что эти файлы существуют в вашем приложении.

Темы, существующие в ОС Android, такие как `Theme`, должны объявляться с пространством имен, указывающим на их местоположение. Библиотека AppCompat использует запись `android:Theme`, потому что тема существует в ОС Android.

Наконец, мы пришли к месту назначения. Здесь представлено гораздо больше атрибутов, чем вам захочется переопределять в своей теме. Конечно, вы можете перейти к родителю `Platform.AppCompat`: `Theme`, но это не обязательно. Тема определяет все атрибуты, которые вам понадобятся.

В самом начале объявляется `windowBackground`. По имени атрибута можно предположить, что он определяет цвет фона темы.

```
<style name="Platform.AppCompat" parent="android:Theme">
    <item name="android:windowNoTitle">true</item>

    <!-- Цвета окна -->
    <item name="android:colorForeground">@color/foreground_material_dark</item>
    <item name="android:colorForegroundInverse">@color/
                            foreground_material_light</item>
    <item name="android:colorBackground">@color/background_material_dark</item>
    <item name="android:colorBackgroundCacheHint">@color/
                            abc_background_cache_hint_selector_material_dark</
item>
    <item name="android:disabledAlpha">@dimen/abc_disabled_alpha_material_dark</
item>
    <item name="android:backgroundDimAmount">0.6</item>
    <item name="android:windowBackground">@color/background_material_dark</item>
    ...
</style>
```

Этот атрибут должен переопределяться в приложении BeatBox. Вернитесь к файлу styles.xml и переопределите атрибут `windowBackground`.

Листинг 22.10. Настройка фона окна (res/values/styles.xml)

```
<style name="AppTheme" parent="Theme.AppCompat">
    <item name="colorPrimary">@color/red</item>
    <item name="colorPrimaryDark">@color/dark_red</item>
    <item name="colorAccent">@color/gray</item>

    <item name="android:windowBackground">@color/soothing_blue</item>
</style>
```

Обратите внимание на необходимость использования пространства имен `android` при переопределении атрибута, потому что `windowBackground` объявляется в ОС Android.

Запустите BeatBox, прокрутите список до конца `RecyclerView` и убедитесь в том, что в том месте, где фон не закрыт кнопками, он окрашен в синий цвет (рис. 22.7).

Рис. 22.7. BeatBox с окраской фона в теме

Действия, которые мы только что совершили для нахождения атрибута `windowBackground`, приходится выполнять каждому разработчику Android при модификации темы приложения. Документация по этим атрибутам практически отсутствует. Чтобы узнать о доступных возможностях, обычно приходится обращаться к источникам.

Итак, мы переходили по следующим темам:

- `Theme.AppCompat`
- `Base.Theme.AppCompat`
- `Base.V7.Theme.AppCompat`
- `Platform.AppCompat`

Мы переходили по иерархии тем до тех пор, пока не добрались до корневой темы AppCompat. Когда вы лучше освоитесь с возможностями работы с темами, вероятно, вы сразу будете переходить к соответствующей теме. Тем не менее полезно пройти по иерархии до ее корня, чтобы понять логику происхождения темы.

Иерархия тем может измениться со временем, но задача перемещения по иерархии останется неизменной. Вы переходите по иерархии тем до тех пор, пока не будет найден атрибут, который требуется переопределить.

Изменение атрибутов кнопки

Ранее мы настраивали кнопки приложения BeatBox вручную, задавая атрибут `style` в файле res/layout/list_item_sound.xml. В более сложных приложениях, в которых кнопки распределены по многим фрагментам, решение с назначением атрибута `style` для каждой кнопки плохо масштабируется. В такой ситуации можно пойти дальше и определить в теме стиль для всех кнопок приложения.

Прежде чем добавлять в тему стиль кнопок, удалите атрибут `style` из файла res/layout/list_item_sound.xml.

Листинг 22.11. Долой! Есть способ получше (res/layout/list_item_sound.xml)

```
<Button
    style="@style/BeatBoxButton.Strong"
    android:layout_width="match_parent"
    android:layout_height="120dp"
    android:onClick="@{() -> viewModel.onButtonClicked()}"
    android:text="@{viewModel.title}"
    tools:text="Sound name"/>
```

Запустите приложение BeatBox и убедитесь в том, что кнопки вернулись к старому невыразительному виду.

Пора снова заняться исследованиями темы. На этот раз нас интересует атрибут `buttonStyle`. Вы найдете его в `Base.V7.Theme.AppCompat`.

```
<style name="Base.V7.Theme.AppCompat" parent="Platform.AppCompat">
    ...
    <!-- Стили кнопок -->
    <item name="buttonStyle">@style/Widget.AppCompat.Button</item>
    <item name="buttonStyleSmall">@style/Widget.AppCompat.Button.Small</item>
    ...
</style>
```

Атрибут `buttonStyle` определяет стиль любой нормальной кнопки в приложении.

Атрибуту `buttonStyle` вместо значения назначается ресурс стиля. При обновлении атрибута `colorBackground` передается значение: цвет. В нашем случае атрибут `buttonStyle` должен ссылаться на стиль. Перейдите к `Widget.AppCompat.Button`, чтобы просмотреть стиль кнопки.

```
<style name="Widget.AppCompat.Button" parent="Base.Widget.AppCompat.Button"/>
```

Стиль `Widget.AppCompat.Button` самостоятельно никакие атрибуты не определяет. Чтобы просмотреть его содержимое, перейдите к его родителю. Вы увидите, что базовый стиль существует в двух версиях. Выберите версию values/values.xml.

```
<style name="Base.Widget.AppCompat.Button" parent="android:Widget">
    <item name="android:background">@drawable/abc_btn_default_mtrl_shape</item>
    <item name="android:textAppearance">?android:attr/textAppearanceButton</item>
    <item name="android:minHeight">48dip</item>
    <item name="android:minWidth">88dip</item>
    <item name="android:focusable">true</item>
    <item name="android:clickable">true</item>
    <item name="android:gravity">center_vertical|center_horizontal</item>
</style>
```

Каждой кнопке `Button`, используемой в BeatBox, назначаются эти атрибуты.

Воспроизведите в BeatBox то, что происходит в теме Android. Измените родителя `BeatBoxButton`, чтобы атрибуты наследовались от существующего стиля кнопки. Также удалите стиль `BeatBoxButton.Strong`, который использовался ранее.

Листинг 22.12. Создание стиля кнопки (res/values/styles.xml)

```
<resources>

    <style name="AppTheme" parent="Theme.AppCompat">
        <item name="colorPrimary">@color/red</item>
        <item name="colorPrimaryDark">@color/dark_red</item>
        <item name="colorAccent">@color/gray</item>

        <item name="android:windowBackground">@color/soothing_blue</item>
    </style>

    <style name="BeatBoxButton" parent="Widget.AppCompat.Button">
        <item name="android:background">@color/dark_blue</item>
    </style>

    <style name="BeatBoxButton.Strong">
        <item name="android:textStyle">bold</item>
    </style>

</resources>
```

Мы указали родителя `Widget.AppCompat.Button`. Наша кнопка должна наследовать все свойства обычной кнопки, а затем избирательно изменять некоторые атрибуты.

Если родительская тема для `BeatBoxButton` не указана, то внешний вид кнопок резко ухудшается, и кнопка вообще перестает напоминать кнопку. Многие привычные свойства, например выравнивание текста по центру кнопки, пропадают.

Итак, стиль `BeatBoxButton` полностью определен, и мы можем использовать его в приложении. Вернитесь к атрибуту `buttonStyle`, который мы обнаружили ранее в ходе исследования тем Android. Продублируйте этот атрибут в своей теме.

Листинг 22.13. Использование стиля BeatBoxButton (res/values/styles.xml)

```
<resources>

    <style name="AppTheme" parent="Theme.AppCompat">
        <item name="colorPrimary">@color/red</item>
```

```
        <item name="colorPrimaryDark">@color/dark_red</item>
        <item name="colorAccent">@color/gray</item>

        <item name="android:windowBackground">@color/soothing_blue</item>
        <item name="buttonStyle">@style/BeatBoxButton</item>
    </style>

    <style name="BeatBoxButton" parent="Widget.AppCompat.Button">
        <item name="android:background">@color/dark_blue</item>
    </style>

</resources>
```

Обратите внимание: при определении `buttonStyle` префикс `android:` не используется. Дело в том, что переопределяемый атрибут `buttonStyle` реализован в библиотеке AppCompat.

Мы переопределяем атрибут `buttonStyle` и подставляем свой собственный стиль: `BeatBoxButton`.

Запустите приложение BeatBox и обратите внимание на то, что все кнопки окрасились в темно-синий цвет (рис. 22.8). Таким образом, мы изменили внешний вид всех обычных кнопок в BeatBox без прямой модификации файлов макетов. Атрибуты темы в Android — сила!

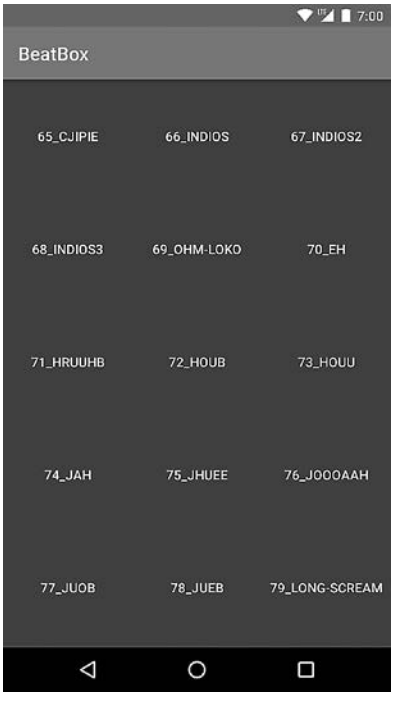


Рис. 22.8. Приложение BeatBox с полностью определенными темами

Конечно, было бы хорошо, если бы кнопки были больше похожи на кнопки. В следующей главе проблема будет решена, и тогда кнопки проявят себя в полной мере.

Для любознательных: подробнее о наследовании стилей

Описание наследований стилей, приведенное ранее, не содержит полной информации. Возможно, вы заметили изменение в стиле наследования во время изучения иерархии тем. В темах `AppCompat` имя темы используется для обозначения наследования — до того момента, когда мы приходим к теме `Platform.AppCompat`.

```
<style name="Platform.AppCompat" parent="android:Theme">
...
</style>
```

Здесь вместо «наследного» стиля формирования имен происходит переключение на прямое определение родителя в атрибуте `parent`. Почему?

Указание родительской темы в имени темы работает только для тем, существующих в одном пакете. Таким образом, в темах ОС Android в большинстве случаев используется «наследный» стиль имен, и библиотека AppCompat действует так же. Но как только наследование пересекает границу AppCompat, используется атрибут `parent`.

Это правило желательно соблюдать и в ваших приложениях. Укажите родителя темы в имени темы, если вы наследуете от одной из своих собственных тем. Если же наследование осуществляется от стиля или темы ОС Android, используйте явное задание атрибута `parent`.

Для любознательных: обращение к атрибутам тем

После того как атрибуты будут объявлены в теме, вы сможете обращаться к ним из XML или из кода.

Для обращения к атрибуту темы из разметки XML используется запись, продемонстрированная для атрибута `listSeparatorTextViewStyle` в главе 7. Для ссылки на конкретное значение в XML (например, цвет) используется синтаксис `@`. Запись `@color/gray` указывает на конкретный ресурс.

Для ссылок на ресурс в теме используется знак `?`:

```
<Button xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/list_item_sound_button"
    android:layout_width="match_parent"
```

```
    android:layout_height="120dp"
    android:background="?attr/colorAccent"
    tools:text="Sound name"/>
```

Знак `?` указывает на использование ресурса, на который указывает атрибут `colorAccent` вашей темы. В данном случае это серый цвет, определенный в файле colors.xml.

Также можно использовать атрибуты темы в коде, хотя на этот раз запись получается не столь компактной.

```
Resources.Theme theme = getActivity().getTheme();
int[] attrsToFetch = { R.attr.colorAccent };
TypedArray a = theme.obtainStyledAttributes(R.style.AppTheme, attrsToFetch);
int accentColor = a.getInt(0, 0);
a.recycle();
```

В объекте `Theme` мы приказываем разрешить атрибут `R.attr.colorAccent`, определенный в `AppTheme`: `R.style.AppTheme`. Вызов возвращает массив `TypedArray`, содержащий данные. Из массива `TypedArray` извлекается значение типа `int`, которое в дальнейшем может использоваться, например, для изменения фона кнопки.

Панель инструментов и кнопки в BeatBox именно так поступают для определения своего стиля на основании атрибутов темы.

23 Графические объекты

После назначения тем BeatBox пришло время сделать что-то с кнопками.

В текущей версии кнопки никак не реагируют на нажатия; это просто синие прямоугольники. В этой главе мы при помощи *графических объектов* (drawable) XML поднимем приложение BeatBox на новый уровень (рис. 23.1).

Рис. 23.1. Новая версия BeatBox

В Android графическим объектом называется все, что предназначено для прорисовки на экране, будь то абстрактная фигура, класс, производный от `Drawable`, или растровое изображение. В этой главе мы рассмотрим несколько видов графических объектов: *списки состояний*, *геометрические фигуры* и *списки слоев*. Все три вида определяются в файлах XML, поэтому мы объединим их в более широкую категорию графических объектов XML.

Унификация кнопок

Прежде чем браться за создание графических объектов XML, внесите изменения в файл list_item_sound.xml.

Листинг 23.1. Определение размеров (res/layout/list_item_sound.xml)

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools">
    <data>
        <variable
            name="viewModel"
            type="com.bignerdranch.android.beatbox.SoundViewModel"/>
    </data>
    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="8dp">
        <Button
            android:layout_width="match_parent"
            android:layout_height="120dp"
            android:layout_width="100dp"
            android:layout_height="100dp"
            android:layout_gravity="center"
            android:onClick="@{() -> viewModel.onButtonClicked()}"
            android:text="@{viewModel.title}"
            tools:text="Sound name"/>
    </FrameLayout>
</layout>
```

Каждой кнопке назначается ширина и высота 100dp, чтобы последующее преобразование кнопок в круги не приводило к искажениям.

В `RecyclerView` независимо от размера экрана всегда отображаются три столбца. При наличии свободного места `RecyclerView` растянет эти столбцы по размерам экрана. В нашем приложении кнопки растягиваться не должны, поэтому они были заключены в виджет `FrameLayout`: он будет растягиваться, а кнопки — нет.

Запустите приложение BeatBox. Вы увидите, что все кнопки имеют одинаковые размеры и разделяются небольшими интервалами (рис. 23.2).

Геометрические фигуры

Придадим кнопкам круглую форму с помощью `ShapeDrawable`. Поскольку графические объекты XML не привязаны к конкретной плотности пикселов, обычно они размещаются в папке drawable по умолчанию (а не в одной из специализированных папок).

В окне инструментов Project создайте в папке res/drawable новый файл с именем button_beat_box_normal.xml. (Почему кнопка названа «normal», то есть «нормальной»? Потому что скоро появится другая, менее нормальная.)

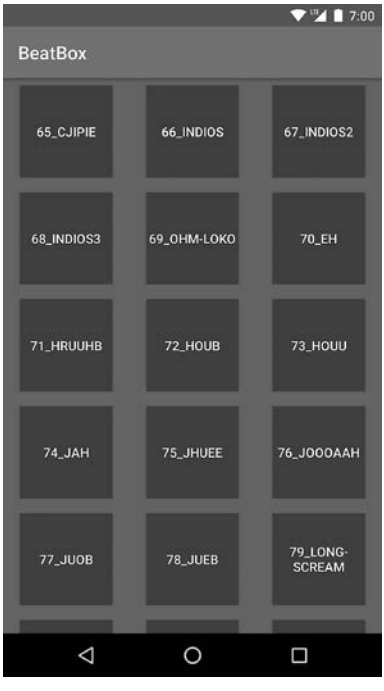


Рис. 23.2. Равномерное распределение кнопок

Листинг 23.2. Создание графического объекта (res/drawable/button_beat_box_normal.xml)

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="oval">

    <solid
        android:color="@color/dark_blue"/>

</shape>
```

Этот файл создает эллиптический графический объект, заполненный темно-синим цветом. Вы можете использовать элемент `shape` для создания других фигур: прямоугольников, линий, градиентов и т. д. За подробностями обращайтесь к документации по адресу *developer.android.com/guide/topics/resources/drawable-resource.html*.

Назначьте `button_beat_box_normal` в качестве фона кнопок.

Листинг 23.3. Изменение фона кнопок (res/drawable/button_beat _box_normal.xml)

```
<resources>

    <style name="AppTheme" parent="Theme.AppCompat">
        ...
```

```
    </style>

    <style name="BeatBoxButton" parent="android:style/Widget.Holo.Button">
        <item name="android:background">@color/dark_blue</item>
        <item name="android:background">@drawable/button_beat_box_normal</item>
    </style>

</resources>
```

Запустите приложение BeatBox. Все кнопки стали круглыми (рис. 23.3).

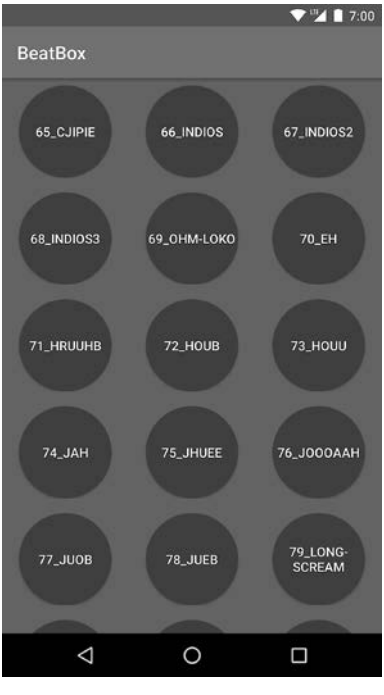


Рис. 23.3. Круглые кнопки

Нажмите кнопку. Вы услышите звук, но внешний вид кнопки не изменится. Было бы лучше, если бы нажатая кнопка отличалась внешним видом от ненажатой.

Списки состояний

Чтобы решить эту проблему, сначала определите новую геометрическую фигуру для кнопки в нажатом состоянии.

Создайте в папке res/drawable файл button_beat_box_pressed.xml. Нажатая кнопка будет выглядеть так же, как обычная, но ей будет назначен красный цвет фона.

Листинг 23.4. Определение графического объекта для нажатой кнопки (res/drawable/button_beat _box_pressed.xml)

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="oval">

    <solid
        android:color="@color/red"/>

</shape>
```

Новая версия должна использоваться тогда, когда пользователь нажимает кнопку. Для решения этой задачи используется *список состояний*.

Список состояний представляет собой графический объект, который указывает на другие графические объекты в зависимости от состояния некоторого селектора. Кнопка может находиться в нажатом и ненажатом состоянии. Список состояний будет определять один графический объект как фон для кнопки в нажатом состоянии и другой графический объект для ненажатых кнопок.

Определите список состояний в папке drawable.

Листинг 23.5. Создание списка состояний (res/drawable/button_beat_box.xml)

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/button_beat_box_pressed"
        android:state_pressed="true"/>
    <item android:drawable="@drawable/button_beat_box_normal" />
</selector>
```

Измените стиль кнопок, чтобы список состояний использовался в качестве фона.

Листинг 23.6. Назначение списка состояний (res/values/styles.xml)

```
<resources>

    <style name="AppTheme" parent="Theme.AppCompat">
        ...
    </style>

    <style name="BeatBoxButton" parent="android:style/Widget.Holo.Button">
        <item name="android:background">@drawable/button_beat_box_normal</item>
        <item name="android:background">@drawable/button_beat_box</item>
    </style>

</resources>
```

Когда кнопка находится в нажатом состоянии, в качестве фона используется графический объект `button_beat_box_pressed`. В противном случае фон кнопки определяется `button_beat_box_normal`.

Запустите приложение BeatBox и нажмите кнопку. Фон кнопки изменится (рис. 23.4). Впечатляет, не правда ли?

Рис. 23.4. BeatBox с кнопкой в нажатом состоянии

Списки состояний — удобный инструмент настройки элементов интерфейса. Также поддерживаются и другие состояния, включая состояние блокировки, наличия фокуса и активации. За подробностями обращайтесь к документации по адресу *developer.android.com/guide/topics/resources/drawable-resource.html#StateList*.

Списки слоев

Приложение BeatBox хорошо смотрится: в нем используются круглые кнопки, которые визуально реагируют на нажатия. Но пришло время сделать что-то более интересное.

Списки слоев (layer list) позволяют объединить два графических объекта XML в один объект. Вооружившись этим инструментом, мы добавим темное кольцо вокруг кнопки в нажатом состоянии.

Листинг 23.7. Использование списка слоев (res/drawable/button_beat_box_pressed.xml)

```
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
    <item>
        <shape xmlns:android="http://schemas.android.com/apk/res/android"
            android:shape="oval">
          <solid
```

```
            android:color="@color/red"/>
        </shape>
    </item>
    <item>
        <shape
            android:shape="oval">

          <stroke
              android:width="4dp"
              android:color="@color/dark_red"/>
        </shape>
    </item>
</layer-list>
```

Список слоев состоит из двух графических объектов. Первый объект — красный круг, как и перед изменением. Второй графический объект будет нарисован поверх первого. Второй графический объект представляет собой другой овал с толщиной линии 4dp; в результате создается кольцо темно-красного цвета.

Объединение этих двух графических объектов образует список слоев. В список слоев можно включить более двух графических объектов, чтобы добиться еще более сложного результата.

Запустите приложение BeatBox и нажмите кнопку. Вы увидите, что в нажатом состоянии она выделяется кольцом (рис. 23.5). Еще лучше, чем прежде.

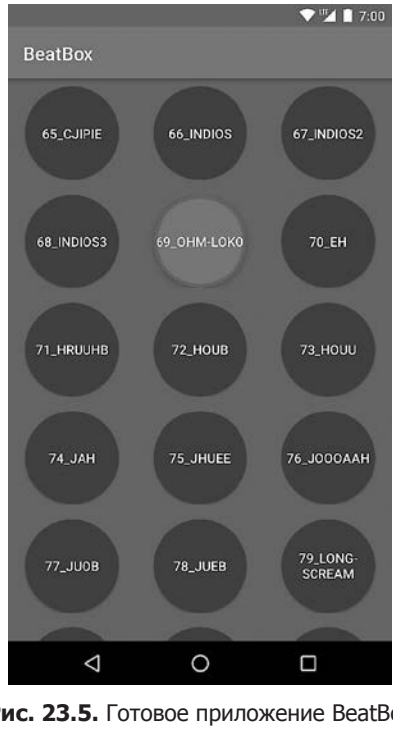


Рис. 23.5. Готовое приложение BeatBox

Добавление списка слоев завершает приложение BeatBox. Помните, как банально выглядел исходный интерфейс? Чтобы создать нечто куда более интересное, нам потребовалось совсем немного времени. Сделайте ваше приложение приятным для глаз — и ваши усилия окупятся его популярностью.

Для любознательных: для чего нужны графические объекты XML?

Нам всегда нужно нажатое состояние для кнопок, поэтому список состояний является критическим компонентом любого приложения Android. Но как насчет геометрических фигур и списков слоев? Стоит ли использовать их?

Графические объекты XML отличаются гибкостью. Их можно использовать для многих целей и легко обновлять в будущем. Комбинации списков слоев и геометрических фигур позволяют создавать сложные фоны без графического редактора. Если вы решите изменить цветовую схему в BeatBox, обновить цвета в графическом объекте XML будет несложно.

В этой главе графические объекты XML определялись в каталоге drawable без квалификаторов, уточняющих плотность пикселов. Это объясняется тем, что графические объекты XML не зависят от плотности. Стандартные растровые изображения обычно приходится создавать для нескольких вариантов плотности, чтобы изображение выглядело четко на большинстве устройств. Графические объекты XML достаточно определить только один раз, и они будут четко выглядеть при любой плотности.

Для любознательных: Mipmap

Квалификаторы ресурсов и графические объекты удобны. Когда вам потребуется добавить графику в приложение, вы генерируете изображение с несколькими разными размерами и раскладываете их по папкам с квалификаторами: drawable-mdpi, drawable-hdpi и т. д. Затем вы обращаетесь к изображению по имени, а Android выбирает версию с нужной плотностью в зависимости от текущего устройства.

Однако у этой системы есть свой недостаток. Файл APK, публикуемый в Google Play Store, содержит все изображения из каталогов drawable для всех вариантов плотности, добавленные в проект, — при том что многие из них использоваться не будут. Конечно, это приводит к неэффективному увеличению размера приложения.

Для решения этой проблемы можно сгенерировать разные APK для всех вариантов плотности пикселов: APK с mdpi-версией приложения, APK с hdpi-версией приложения и т. д. (За дополнительной информацией о построении разных версий APK обращайтесь к документации по адресу: *tools.android.com/tech-docs/new-build-system/user-guide/apk-splits.*)

Впрочем, у этого правила есть одно исключение: вы должны поддерживать значки приложения для лаунчера во всех вариантах плотности.

В Android лаунчер представляет собой домашний экран со значками приложений (эта тема подробнее рассматривается в главе 24); он открывается при нажатии кнопки Home.

Некоторые новые лаунчеры отображают значки приложений в большем размере, чем принято. Чтобы увеличенный значок нормально смотрелся, такие лаунчеры должны взять значок из версии со следующей плотностью. Например, на hdpi-устройствах для представления приложения в лаунчере будет использоваться значок xhdpi. Но если xhdpi-версия исключена из APK, лаунчеру придется вернуться к версии с более низким разрешением.

Масштабированные значки с низким разрешением кажутся размытыми, а мы хотим, чтобы значок приложения выглядел четко и аккуратно.

В Android для решения этой проблемы используется каталог mipmap. При включении разбивки APK ресурсы mipmap не удаляются из APK. В остальном такие ресурсы не отличаются от графических объектов.

На момент написания книги новые проекты в Android Studio настроены на использование mipmap-ресурсов для значка лаунчера (рис. 23.6).

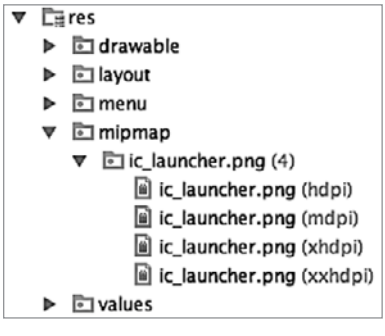


Рис. 23.6. Значки mipmap

Итак, мы рекомендуем просто разместить значок лаунчера в разных каталогах mipmap. Всем остальным изображениям место в каталогах drawable.

Для любознательных: 9-зонные изображения

Иногда (а может быть, очень часто) вы будете использовать обычные графические изображения в качестве фона кнопок. Но что произойдет с этими изображениями, если кнопка может отображаться в разных вариантах размеров? Если ширина кнопки превышает ширину фонового изображения, то изображение просто растянется, верно? Но всегда ли результат будет хорошо выглядеть?

Равномерное растяжение фонового изображения не всегда приводит к хорошему результату. Иногда требуется лучше управлять тем, как будет растягиваться изображение.

В этом разделе приложение BeatBox будет переведено на использование *9-зонного изображения* в качестве фона кнопок (вскоре эта тема будет рассмотрена более подробно). Выбор решения объясняется даже не тем, что оно хорошо подходит для BeatBox, — просто оно демонстрирует, как работают 9-зонные изображения в ситуациях с использованием графического файла.

Для начала изменим файл list_item_sound.xml для того, чтобы размеры кнопки могли изменяться по размерам свободного пространства.

Листинг 23.8. Включение растяжения кнопок (res/layout/list_item_sound.xml)

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools">
    <data>
        <variable
            name="viewModel"
            type="com.bignerdranch.android.beatbox.SoundViewModel"/>
    </data>
    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="8dp">
        <Button
            android:layout_width="~~100dp~~ match_parent"
            android:layout_height="~~100dp~~ match_parent"
            android:layout_gravity="center"
            android:onClick="@{() -> viewModel.onButtonClicked()}"
            android:text="@{viewModel.title}"
            tools:text="Sound name"/>
    </FrameLayout>
</layout>
```

Теперь кнопки занимают все доступное место, оставляя поля шириной 8dp. Изображение на рис. 23.7 будет использоваться в качестве нового фона кнопок BeatBox.

Рис. 23.7. Новое фоновое изображение кнопки (res/drawable-xxhdpi/ic_button_beat_box_default.png)

В решениях этой главы (см. раздел «Добавление значка» главы 2) вы найдете это изображение вместе с версией для нажатого состояния в папке xxhdpi. Скопируйте оба изображения в папку drawable-xxhdpi вашего проекта и внесите изменения в файл button_beat_box.xml, чтобы назначить их фонами кнопок.

Листинг 23.9. Назначение новых фоновых изображений кнопок (res/drawable/button_beat_box.xml)

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/button_beat_box_pressed"
    <item android:drawable="@drawable/ic_button_beat_box_pressed"
        android:state_pressed="true"/>
    <item android:drawable="@drawable/button_beat_box_normal"
    <item android:drawable="@drawable/ic_button_beat_box_normal"
</selector>
```

Запустите BeatBox; вы увидите, что кнопки выводятся с новым фоном (рис. 23.8).

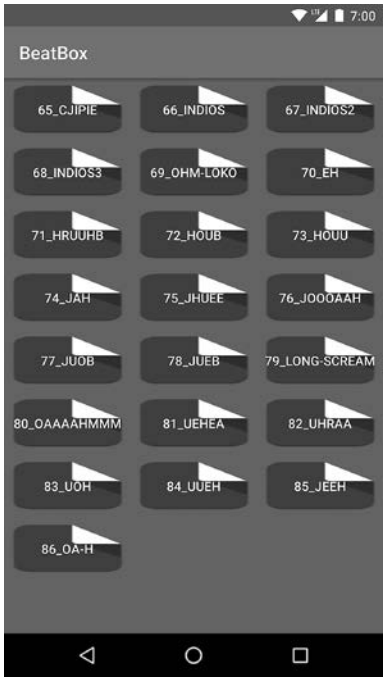


Рис. 23.8. Искажение вида кнопок

И что получилось?.. Да ничего хорошего.

Почему результат плохо выглядит? Android равномерно растягивает изображение ic_button_beat_box_button.png, включая загнутый уголок и закругления. Было бы лучше, если бы вы могли явно указать, какие части изображения можно растягивать, а какие должны остаться в исходном виде. На помощь приходят *9-зонные* изображения.

Файл *9-зонного* изображения специально форматируется так, чтобы система Android знала, какие части изображения можно или нельзя форматировать. При правильной реализации это гарантирует, что края и углы фона будут соответствовать изображению в том виде, в каком оно было создано.

Почему изображения называются «9-зонными»? Такие изображения разбиваются сеткой 3 × 3 — такая сетка состоит из 9 элементов, или *зон* (patches). Углы сетки не масштабируются, стороны масштабируются только в одном направлении, а центральная зона масштабируется в обоих направлениях (рис. 23.9).

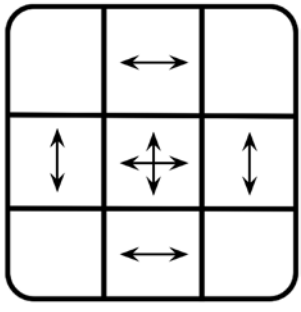


Рис. 23.9. Как работает 9-зонное изображение

9-зонное изображение во всех отношениях напоминает обычный файл png, за исключением двух аспектов: имя файла завершается суффиксом .9.png и изображение имеет дополнительную рамку толщиной в один пиксел. Рамка задает местонахождение центрального квадрата. Черными пикселами рамки обозначается центр, а прозрачными — края.

9-зонное изображение можно создать в любом графическом редакторе, но проще воспользоваться программой draw9patch, включенной в поставку Android SDK, или средствами Android Studio.

Сначала преобразуйте два новых фоновых изображения в 9-зонный формат: щелкните правой кнопкой мыши на файле ic_button_beat_box_default.png в окне инструментов Project и выберите команду Refactor ▸ Rename.... Переименуйте файл в ic_button_beat_box_default.9.png. (Если Android Studio предупредит вас о том, что такой файл уже существует, нажмите кнопку Continue.) Затем повторите процесс и переименуйте файл с «нажатой» версией изображения в ic_button_beat_box_pressed.9.png.

Сделайте двойной щелчок на стандартном изображении в окне инструментов Project, чтобы открыть его во встроенном 9-зонном редакторе Android Studio (рис. 23.10). (Если Android Studio не откроет редактор, попробуйте закрыть файл и свернуть папку drawable в окне инструментов Project, после чего откройте изображение заново.)

В 9-зонном редакторе сначала установите флажок Show patches, чтобы зоны были лучше видны. Заполните черные пикселы на верхней и левой границе, чтобы обозначить растягиваемые области изображения, как показано на рис. 23.10. Также перетащите стороны цветной накладки в соответствии с иллюстрацией.

Черная линия в верхней части изображения определяет регион, который должен растягиваться при растяжении изображения по горизонтали. Линия слева показывает, какие пикселы должны растягиваться при растяжении изображения по вертикали.

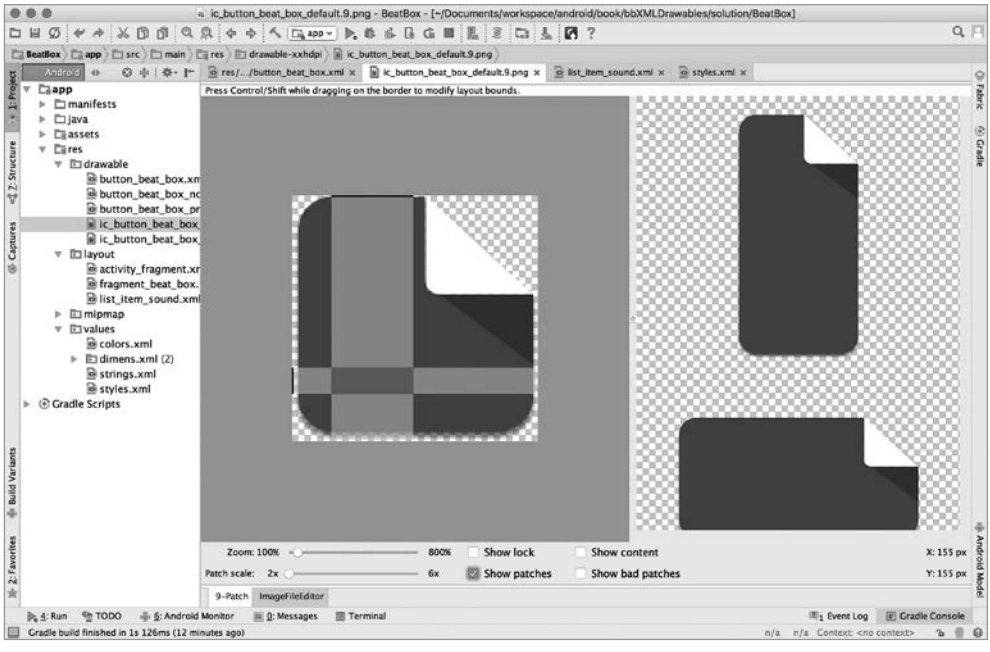


Рис. 23.10. Создание 9-зонного изображения

Рис. 23.11. Новая улучшенная версия

Повторите процедуру с «нажатой» версией изображения. Запустите приложение BeatBox и посмотрите, как работает новое 9-зонное изображение (рис. 23.11).

Итак, левый и верхний участки границы отмечают растягиваемую область изображения. А как насчет правого и нижнего участков? Они отмечают необязательную *область содержимого* — область, в которой должно выводиться некое содержимое (обычно текст). Если область содержимого не задана, по умолчанию считается, что она совпадает с растягиваемой областью.

Воспользуемся областью содержимого для выравнивая по центру текста внутри кнопок. Вернитесь к файлу ic_button_beat_box_default.9.png и добавьте линии внизу и справа, как показано на рис. 23.12. Установите флажок Show content в 9-зонном редакторе. На экране выделяются области изображения, которые будут содержать текст.

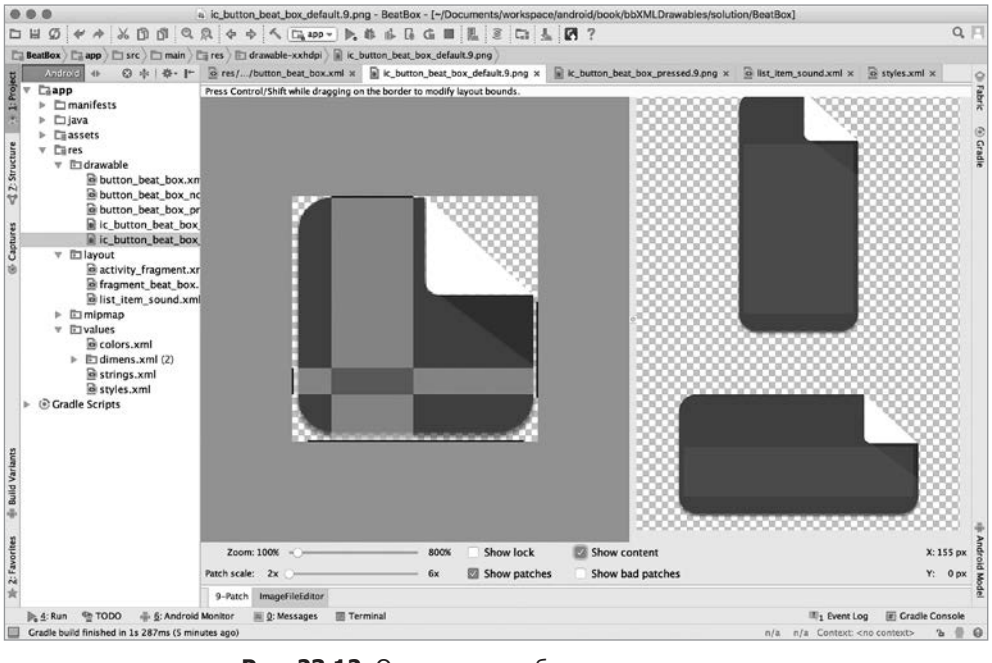


Рис. 23.12. Определение области содержимого

Повторите процесс для «нажатой» версии изображения. Внимательно проследите за тем, чтобы оба изображения обновились с правильными границами области содержимого. Когда 9-зонные изображения задаются при помощи графических объектов списка состояний (как в BeatBox), область содержимого может работать не так, как вы ожидаете. Android задает область содержимого при инициализации фона и не изменяет ее при нажатии кнопки. Таким образом, область содержимого одного из двух изображений будет игнорироваться! Изображение, которое Android будет использовать для заполнения области содержимого, не задано, поэтому будет лучше, если все ваши 9-зонные изображения в списке состояний будут использовать одну область содержимого.

Запустите BeatBox — текст аккуратно выравнивается по центру (рис. 23.13).

Рис. 23.13. Дополнительные усовершенствования

Попробуйте повернуть устройство в альбомную ориентацию. Изображения растягиваются еще сильнее, но фон кнопки все равно выглядит хорошо, а текст остается выровненным по центру.

Упражнение. Темы кнопок

После обновления 9-зонного изображения можно заметить, что с фоном кнопок что-то не так. За загнутым уголком виднеется нечто похожее на тень. Возможно, вы также заметите, что тень появляется только в Android версии 21 и выше.

Тень является частью имитации рельефа, который назначается всем кнопкам по умолчанию в Lollipop и выше (21+). Нажатая кнопка визуально «приближается» к пальцу (эта тема более подробно рассматривается в главе 35).

С загнутым уголком эту тень лучше убрать. Используйте свои навыки исследования тем для определения того, как применяется эта тень. Существует ли другой стиль кнопки, который можно использовать в качестве родителя для стиля `BeatBoxButton`?

24 Подробнее об интентах и задачах

В этой главе мы используем неявные интенты для создания приложения-лаунчера, заменяющего стандартный лаунчер Android. На рис. 24.1 показано, как будет выглядеть приложение NerdLauncher.

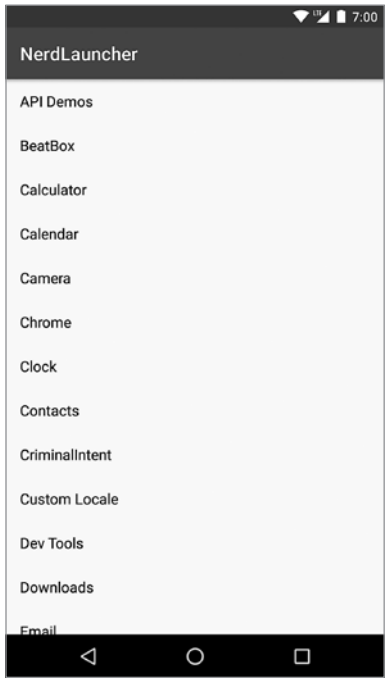


Рис. 24.1. Итоговый вид NerdLauncher

NerdLauncher выводит список приложений на устройстве. Пользователь нажимает элемент списка, чтобы запустить соответствующее приложение.

Чтобы приложение работало правильно, нам придется углубить свое понимание интентов, фильтров интентов и схем взаимодействий между приложениями в среде Android.

Создание приложения NerdLauncher

Создайте новый проект приложения Android с именем NerdLauncher. Выберите форм-фактор Phone and Tablet и минимальный уровень SDK API 19: Android 4.4 (KitKat). Создайте пустую активность с именем `NerdLauncherActivity`.

Класс `NerdLauncherActivity` станет хостом для одного фрагмента, а сам он должен быть субклассом `SingleFragmentActivity`. Скопируйте файлы SingleFragmentActivity.java и activity_fragment.xml в NerdLauncher из проекта CriminalIntent.

Откройте файл NerdLauncherActivity.java и измените суперкласс `NerdLauncherActivity` на `SingleFragmentActivity`. Удалите код шаблона и переопределите метод `createFragment()` так, чтобы он возвращал `NerdLauncherFragment`. (Пока не обращайте внимания на ошибку, вызванную строкой `return` в `createFragment()`; проблема вскоре будет решена при создании класса `NerdLauncherFragment`.)

Листинг 24.1. Субкласс SingleFragmentActivity (NerdLauncherActivity.java)

```
public class NerdLauncherActivity extends SingleFragmentActivityAppCompatActivity {

    @Override
    protected Fragment createFragment() {
        return NerdLauncherFragment.newInstance();
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        /* Автоматически сгенерированный шаблонный код... */
    }
}
```

`NerdLauncherFragment` выводит список названий приложений в `RecyclerView`. Добавьте библиотеку RecyclerView в число зависимостей, как это было сделано в главе 8.

Переименуйте файл layout/activity_nerd_launcher.xml в layout/fragment_nerd_launcher.xml, чтобы создать макет для фрагмента. Замените его содержимое разметкой `RecyclerView`, изображенной на рис. 24.2.

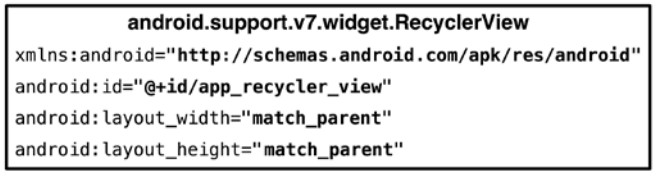


Рис. 24.2. Создание макета NerdLauncherFragment (layout/fragment_nerd_launcher.xml)

Наконец, добавьте новый класс с именем `NerdLauncherFragment`, расширяющий `android.support.v4.app.Fragment`. Добавьте метод `newInstance()` и переопределите метод `onCreateView(...)` для сохранения ссылки на объект `RecyclerView` в переменной класса. (Вскоре мы свяжем данные с `RecyclerView`.)

Листинг 24.2. Базовая реализация NerdLauncherFragment (NerdLauncherFragment.java)

```
public class NerdLauncherFragment extends Fragment {

    private RecyclerView mRecyclerView;

    public static NerdLauncherFragment newInstance() {
        return new NerdLauncherFragment();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_nerd_launcher, container,
                                  false);
        mRecyclerView = (RecyclerView) v.findViewById(R.id.app_recycler_view);
        mRecyclerView.setLayoutManager(new LinearLayoutManager(getActivity()));

        return v;
    }
}
```

Запустите приложение и убедитесь в том, что пока все компоненты взаимодействуют правильно. Если все сделано без ошибок, вы становитесь владельцем приложения NerdLauncher, в котором отображается пустой виджет `RecyclerView` (рис. 24.3).

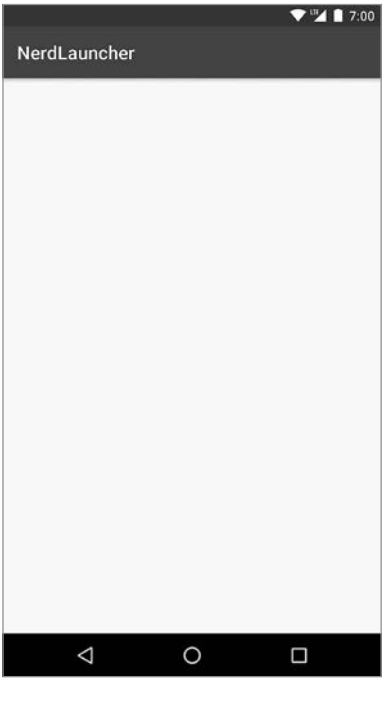


Рис. 24.3. NerdLauncher — начало

Обработка неявного интента

NerdLauncher отображает список запускаемых (launchable) приложений на устройстве. («Запускаемым» называется приложение, которое может быть запущено пользователем, если он щелкнет на значке на экране Home или на экране лаунчера.) Для этого NerdLauncher запрашивает у системы (при помощи `PackageManager`) список запускаемых главных активностей, то есть активностей, фильтры интентов которых включают действие `MAIN` и категорию `LAUNCHER`. Вы уже видели в своих проектах фильтр интентов в файле AndroidManifest.xml:

```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

В файле NerdLauncherFragment.java добавьте метод с именем `setupAdapter()` и вызовите его из `onCreateView(…)`. (Позднее этот метод создаст экземпляр `RecyclerView.Adapter` и назначит его объекту `RecyclerView`, но пока он просто генерирует список данных приложения.) Также создайте неявный интент и получите список активностей, соответствующих интенту, от `PackageManager`. Пока мы ограничимся простой регистрацией количества активностей, возвращенных `PackageManager`.

Листинг 24.3. Получение информации у PackageManager (NerdLauncherFragment.java)

```
public class NerdLauncherFragment extends Fragment {
    private static final String TAG = "NerdLauncherFragment";

    private RecyclerView mRecyclerView;

    public static NerdLauncherFragment newInstance() {
        return new NerdLauncherFragment();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        ...
        setupAdapter();
        return v;
    }

    private void setupAdapter() {
        Intent startupIntent = new Intent(Intent.ACTION_MAIN);
        startupIntent.addCategory(Intent.CATEGORY_LAUNCHER);

        PackageManager pm = getActivity().getPackageManager();
        List<ResolveInfo> activities = pm.queryIntentActivities(startupIntent, 0);

        Log.i(TAG, "Found " + activities.size() + " activities.");
    }
}
```

Запустите приложение NerdLauncher и посмотрите в данных LogCat, сколько приложений вернул экземпляр `PackageManager` (у нас при первом пробном запуске их было 30).

В CriminalIntent для отправки отчетов использовался неявный интент. Чтобы представить на экране список выбора приложений, мы создали неявный интент, упаковали его в интент выбора и отправили ОС вызовом `startActivity(Intent)`:

```
Intent i = new Intent(Intent.ACTION_SEND);
... // Создание и размещение дополнений интентов
i = Intent.createChooser(i, getString(R.string.send_report));
startActivity(i);
```

Почему мы не используем этот подход здесь? Вкратце дело в том, что фильтр интентов `MAIN/LAUNCHER` может соответствовать или не соответствовать неявному интенту `MAIN/LAUNCHER`, отправленному через `startActivity(Intent)`.

Оказывается, вызов `startActivity(Intent)` не означает «Запустить активность, соответствующую этому неявному интенту». Он означает «Запустить активность *по умолчанию*, соответствующую этому неявному интенту». Когда вы отправляете неявный интент с использованием `startActivity(…)` (или `startActivityForResult(Intent)`), ОС незаметно включает в интент категорию `Intent.CATEGORY_DEFAULT`.

Таким образом, если вы хотите, чтобы фильтр интентов соответствовал неявным интентам, отправленным через `startActivity(…)`, вы должны включить в этот фильтр интентов категорию `DEFAULT`.

Активность с фильтром интентов `MAIN/LAUNCHER` является главной точкой входа приложения, которому она принадлежит. Для нее важно лишь то, что она является главной точкой входа приложения, а является ли она главной точкой входа «по умолчанию» — несущественно, поэтому она не обязана включать категорию `CATEGORY_DEFAULT`.

Так как фильтры интентов `MAIN/LAUNCHER` могут не включать `CATEGORY_DEFAULT`, надежность их соответствия неявным интентам, отправленным вызовом `startActivity(Intent)`, не гарантирована. Поэтому мы используем интент для прямого запроса у PackageManager информации об активностях с фильтром интентов `MAIN/LAUNCHER`.

Следующий шаг — отображение меток этих активностей в списке `RecyclerView` экземпляра `NerdLauncherFragment`. *Метка* (label) активности представляет собой отображаемое имя — нечто, понятное пользователю. Если учесть, что эти активности являются активностями лаунчера, такой меткой, скорее всего, должно быть имя приложения.

Метки активностей вместе с другими метаданными содержатся в объектах `ResolveInfo`, возвращаемых `PackageManager`.

Сначала отсортируйте объекты `ResolveInfo`, возвращаемые `PackageManager`, в алфавитном порядке меток, получаемых методом `ResolveInfo.loadLabel(PackageManager)`.

Листинг 24.4. Алфавитная сортировка (NerdLauncherFragment.java)

```
public class NerdLauncherFragment extends Fragment {
    ...
    private void setupAdapter() {
        ...
        List<ResolveInfo> activities = pm.queryIntentActivities(startupIntent, 0);
        Collections.sort(activities, new Comparator<ResolveInfo>() {
            public int compare(ResolveInfo a, ResolveInfo b) {
                PackageManager pm = getActivity().getPackageManager();
                return String.CASE_INSENSITIVE_ORDER.compare(
                        a.loadLabel(pm).toString(),
                        b.loadLabel(pm).toString());
            }
        });
        Log.i(TAG, "Found " + activities.size() + " activities.");
    }
}
```

Теперь определите класс `ViewHolder` для отображения метки активности. Сохраните объект `ResolveInfo` активности в переменной класса (позднее мы еще не раз используем его).

Листинг 24.5. Реализация ViewHolder (NerdLauncherFragment.java)

```
public class NerdLauncherFragment extends Fragment {
    ...
    private void setupAdapter() {
        ...
    }

    private class ActivityHolder extends RecyclerView.ViewHolder {
        private ResolveInfo mResolveInfo;
        private TextView mNameTextView;

        public ActivityHolder(View itemView) {
            super(itemView);
            mNameTextView = (TextView) itemView;
        }

        public void bindActivity(ResolveInfo resolveInfo) {
            mResolveInfo = resolveInfo;
            PackageManager pm = getActivity().getPackageManager();
            String appName = mResolveInfo.loadLabel(pm).toString();
            mNameTextView.setText(appName);
        }
    }
}
```

Затем добавьте реализацию `RecyclerView.Adapter`.

Листинг 24.6. Реализация RecyclerView.Adapter (NerdLauncherFragment.java)

```
public class NerdLauncherFragment extends Fragment {
    ...
    private class ActivityHolder extends RecyclerView.ViewHolder {
        ...
    }

    private class ActivityAdapter extends RecyclerView.Adapter<ActivityHolder> {
        private final List<ResolveInfo> mActivities;
        public ActivityAdapter(List<ResolveInfo> activities) {
            mActivities = activities;
        }

        @Override
        public ActivityHolder onCreateViewHolder(ViewGroup parent, int viewType) {
            LayoutInflater layoutInflater = LayoutInflater.from(getActivity());
            View view = layoutInflater
                    .inflate(android.R.layout.simple_list_item_1, parent, false);
            return new ActivityHolder(view);
        }

        @Override
        public void onBindViewHolder(ActivityHolder holder, int position) {
            ResolveInfo resolveInfo = mActivities.get(position);
            holder.bindActivity(resolveInfo);
        }

        @Override
        public int getItemCount() {
            return mActivities.size();
        }
    }
}
```

Наконец, обновите код `setupAdapter()`, чтобы он создавал экземпляр `ActivityAdapter` и назначал его адаптером `RecyclerView`.

Листинг 24.7. Назначение адаптера RecyclerView (NerdLauncherFragment.java)

```
public class NerdLauncherFragment extends Fragment {
    ...
    private void setupAdapter() {
        ...
        Log.i(TAG, "Found " + activities.size() + " activities.");
        mRecyclerView.setAdapter(new ActivityAdapter(activities));
    }
    ...
}
```

Запустите NerdLauncher; вы увидите список `RecyclerView`, заполненный метками активностей (рис. 24.4).

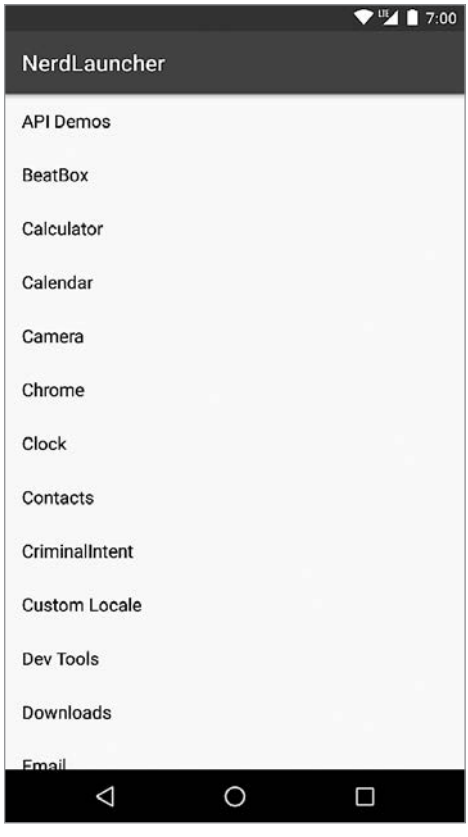


Рис. 24.4. Список активностей

Создание явных интентов на стадии выполнения

Мы использовали неявный интент для сбора информации об активностях и выводе ее в формате списка. Следующим шагом должен стать запуск выбранной активности при нажатии пользователем на элементе списка. Для запуска активности будет использоваться явный интент.

Для создания явного интента необходимо извлечь из `ResolveInfo` имя пакета и имя класса активности. Эти данные можно получить из части `ResolveInfo` с именем `ActivityInfo`. (О том, какие данные доступны в разных частях `ResolveInfo`, можно узнать из документации: *developer.android.com/reference/android/content/pm/ResolveInfo.html*.)

Реализуйте в `ActivityHolder` слушателя нажатий. При нажатии на активности в списке по данным `ActivityInfo` этой активности создайте явный интент. Затем используйте этот явный интент для запуска выбранной активности.

Листинг 24.8. Запуск выбранной активности (NerdLauncherFragment.java)

```
private class ActivityHolder extends RecyclerView.ViewHolder
        implements View.OnClickListener {
    private ResolveInfo mResolveInfo;
    private TextView mNameTextView;

    public ActivityHolder(View itemView) {
        super(itemView);
        mNameTextView = (TextView) itemView;
        mNameTextView.setOnClickListener(this);
    }

    public void bindActivity(ResolveInfo resolveInfo) {
        ...
    }

    @Override
    public void onClick(View v) {
        ActivityInfo activityInfo = mResolveInfo.activityInfo;

        Intent i = new Intent(Intent.ACTION_MAIN)
                .setClassName(activityInfo.applicationInfo.packageName,
                        activityInfo.name);

        startActivity(i);
    }
}
```

Обратите внимание: в этом интенте мы отправляем действие как часть явного интента. Большинство приложений ведет себя одинаково независимо от того, включено действие или нет, однако некоторые приложения могут изменять свое поведение. Одна и та же активность может отображать разные интерфейсы в зависимости от того, как она была запущена. Вам как программисту лучше всего четко объявить свои намерения и позволить активностям запуститься так, как они считают нужным.

В листинге 24.8 мы получаем имя пакета и имя класса из метаданных и используем их для создания явной активности методом `Intent`:

```
public Intent setClassName(String packageName, String className)
```

Этот способ отличается от того, который использовался нами для создания явных интентов в прошлом. Ранее мы использовали конструктор `Intent`, получающий объекты `Context` и `Class`:

```
public Intent(Context packageContext, Class<?> cls)
```

Этот конструктор использует свои параметры для получения `ComponentName` — имени пакета, объединенного с именем класса. Когда вы передаете `Activity` и `Class` для создания `Intent`, конструктор определяет полное имя пакета по `Activity`.

Также можно самостоятельно создать `ComponentName` по именам пакета и класса и использовать следующий метод `Intent` для создания явного интента:

```
public Intent setComponent(ComponentName component)
```

Однако решение с методом `setClassName(…)`, автоматически создающим имя компонента, получается более компактным.

Запустите NerdLauncher и посмотрите, как работает запуск приложений.

Задачи и стек возврата

Android использует задачи для отслеживания текущего состояния пользователя в каждом выполняемом приложении. Каждому приложению, открытому из стандартного лаунчера Android, назначается собственная задача. К сожалению для NerdLauncher, это вполне логичное поведение не используется по умолчанию. Прежде чем разбираться, как заставить приложение запуститься в отдельной задаче, следует понять, что такое задачи и как они работают.

Задача (task) представляет собой стек активностей, с которыми имеет дело пользователь. Активность в нижней позиции стека называется *базовой активностью*, а активность в верхней позиции видна пользователю. При нажатии кнопки Back верхняя активность извлекается из стека. Если нажать кнопку Back при просмотре базовой активности, вы вернетесь к домашнему экрану.

По умолчанию новые активности запускаются в текущей задаче. В приложении CriminalIntent все запускавшиеся активности добавлялись к текущей задаче (рис. 24.5). Это относилось даже к активностям, которые не являлись частью приложения CriminalIntent (например, при запуске активности для отправки отчета).

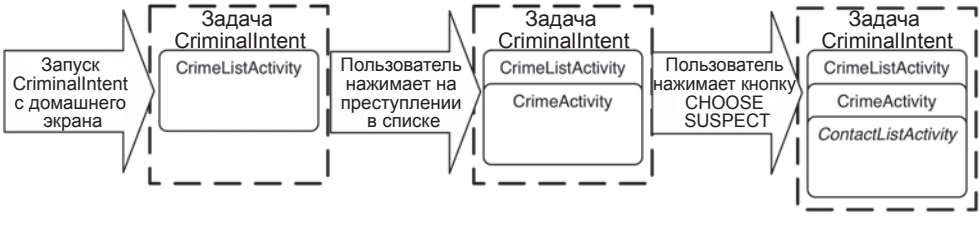


Рис. 24.5. Задача CriminalIntent

Преимущество добавления активности к текущей задаче заключается в том, что пользователь может выполнять обратную навигацию внутри задачи, а не в иерархии приложений (рис. 24.6).

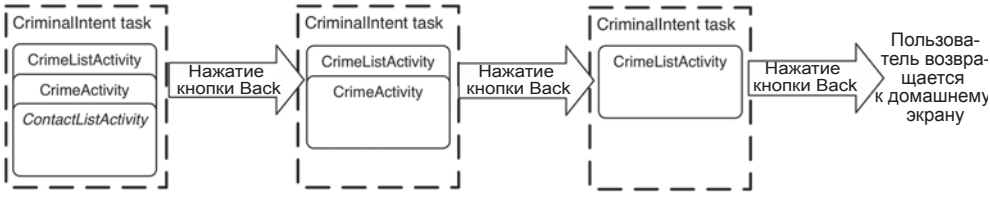


Рис. 24.6. Задача CriminalIntent

Переключение между задачами

Диспетчер задач позволяет переключаться между задачами без изменения состояния каждой задачи (эта возможность упоминалась в главе 3). Например, если вы начинаете вводить новый контакт и переключаетесь на проверку своей публикации в Твиттере, будут запущены две задачи. При переключении на редактирование контактов сохраняется ваша текущая позиция в обеих задачах.

Проверьте, как работает диспетчер задач на вашем устройстве или в эмуляторе. Сначала запустите приложение CriminalIntent с домашнего экрана или из приложения-лаунчера. (Если на вашем устройстве или в эмуляторе приложение CriminalIntent не установлено, откройте и запустите проект CriminalIntent в Android Studio.) Выберите преступление в списке и нажмите кнопку Home, чтобы вернуться к домашнему экрану. Запустите BeatBox с домашнего экрана или из приложения-лаунчера (или при необходимости из Android Studio).

Откройте диспетчер задач. Конкретный способ зависит от устройства; нажмите кнопку Recents, если она присутствует на вашем устройстве. (На кнопке Recents обычно изображен квадрат или два перекрывающихся прямоугольника, и она находится у правого края панели навигации. Два примера кнопки Recents изображены на рис. 24.7.) Если этот способ не работает, попробуйте выполнить долгое нажатие кнопки Home. Если и это не подходит, дважды нажмите кнопку Home.

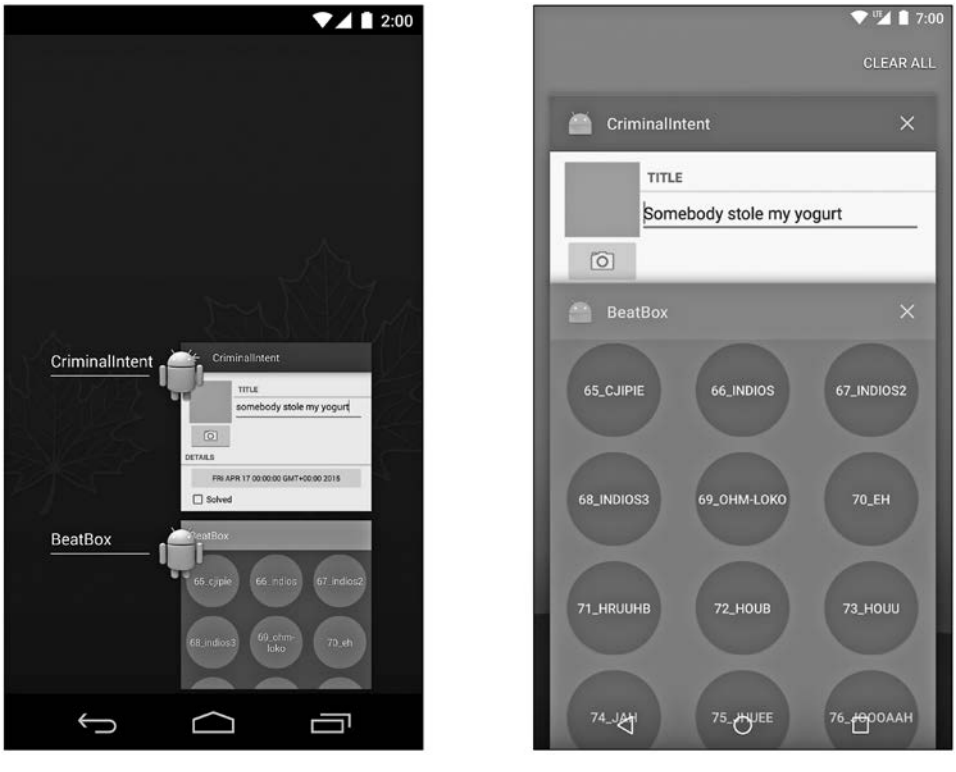


Рис. 24.7. Разновидности диспетчера задач

Слева на рис. 24.7 изображен диспетчер задач, который увидят пользователи, работающие в версии KitKat. Справа изображен диспетчер задач в Lollipop. В обоих случаях элемент, отображаемый для каждого приложения (начиная с Lollipop, эти элементы называются *картами*), представляет задачу этого приложения. В диспетчере отображается снимок экрана активности, находящейся на вершине стека возврата каждой задачи. Пользователь может нажать на элементе BeatBox или CriminalIntent, чтобы вернуться к приложению (и той задаче, с которой он взаимодействовал в этом приложении).

Пользователь может удалить из памяти задачу приложения; для этого следует провести пальцем на элементе задачи. При удалении задачи все ее активности исключаются из стека возврата приложения.

Попробуйте удалить задачу CriminalIntent и перезапустить приложение. Вы увидите список преступлений вместо того преступления, которое редактировалось до удаления задачи.

Запуск новой задачи

Иногда запускаемая активность должна добавляться к текущей задаче. В других случаях она должна запускаться в новой задаче, независимой от запустившей ее активности.

В текущей версии все активности, запускаемые из NerdLauncher, добавляются в задачу NerdLauncher, как показано на рис. 24.8.

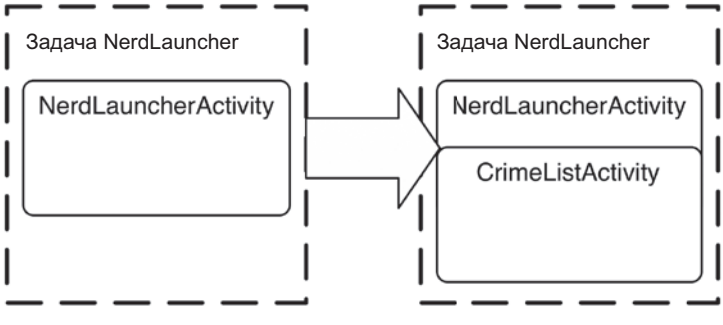


Рис. 24.8. Задача NerdLauncher содержит CriminalIntent

Чтобы убедиться в этом, удалите все задачи, отображаемые в диспетчере задач. Запустите NerdLauncher и щелкните на элементе CriminalIntent, чтобы запустить приложение CriminalIntent. Снова откройте экран диспетчера задач — вы не найдете на нем CriminalIntent. Запущенная активность `CrimeListActivity` была добавлена в задачу NerdLauncher (рис. 24.9). Нажатие на задаче NerdLauncher вернет вас к экрану CriminalIntent, который вы просматривали перед запуском диспетчера задач.

Мы хотим, чтобы приложение NerdLauncher запускало активности в новых задачах (рис. 24.10). Далее пользователь может переключаться между выполняемыми

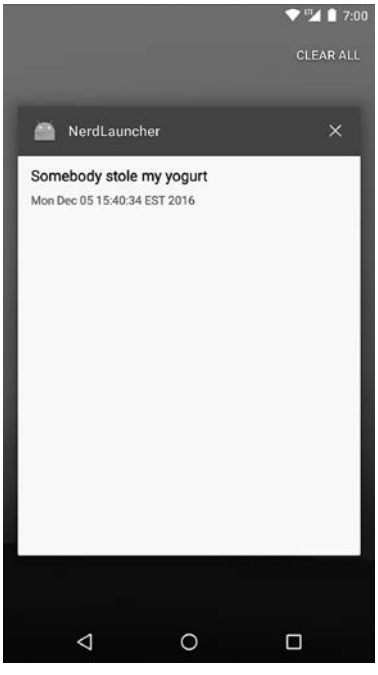


Рис. 24.9. Приложение CriminalIntent не выполняется в отдельной задаче

приложениями так, как считает нужным (при помощи диспетчера задач, NerdLauncher или домашнего экрана).

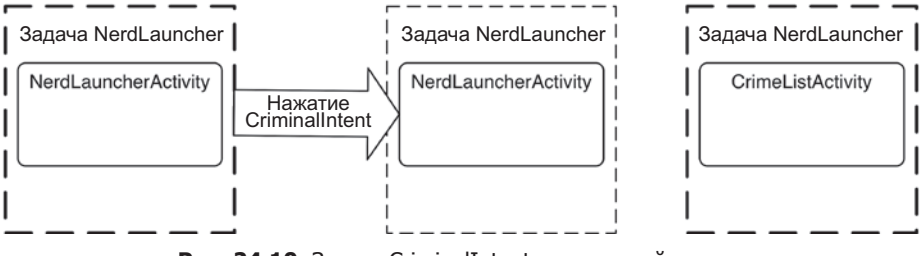


Рис. 24.10. Запуск CriminalIntent в отдельной задаче

Чтобы при запуске новой активности запускалась новая задача, следует добавить в интент соответствующий флаг в файле NerdLauncherFragment.java.

Листинг 24.9. Добавление флага новой задачи в интент (NerdLauncherFragment.java)

```
public class NerdLauncherFragment extends Fragment {
    ...
    private class ActivityHolder extends RecyclerView.ViewHolder
            implements View.OnClickListener {
        ...
        @Override
```

```
        public void onClick(View v) {
            ...
            Intent i = new Intent(Intent.ACTION_MAIN)
                    .setClassName(activityInfo.applicationInfo.packageName,
                            activityInfo.name)
                    .addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);

            startActivity(i);
        }
    }
    ...
}
```

Удалите задачи в диспетчере. Запустите приложение NerdLauncher и выберите CriminalIntent. На этот раз при вызове диспетчера задач становится видно, что CriminalIntent выполняется в отдельной задаче (рис. 24.11).

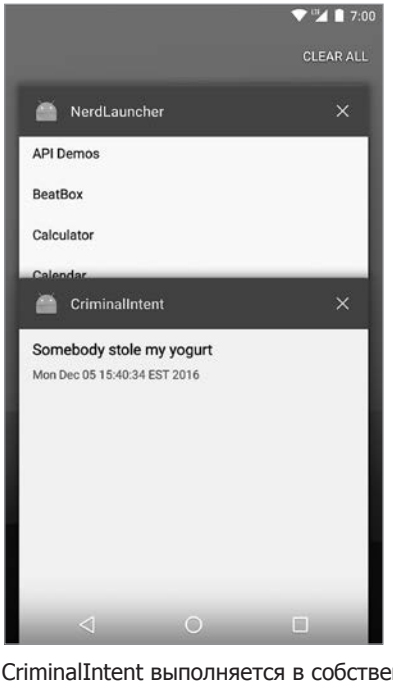


Рис. 24.11. CriminalIntent выполняется в собственной задаче

Повторный запуск CriminalIntent из NerdLauncher не приведет к созданию второй задачи CriminalIntent. Флаг `FLAG_ACTIVITY_NEW_TASK` создает только одну задачу на активность. У CrimeListActivity уже имеется работающая задача, поэтому Android переключится на эту задачу вместо запуска новой.

Убедитесь в этом. Откройте экран с подробной информацией об одном из преступлений в CriminalIntent. Используйте диспетчер задач для переключения на NerdLauncher. Нажмите на элемент CriminalIntent в списке. Вы вернетесь к преж-

нему состоянию в приложении CriminalIntent: просмотру подробной информации об отдельном преступлении.

Использование NerdLauncher в качестве домашнего экрана

Но кому захочется запускать приложение, чтобы запускать другие приложения? Гораздо логичнее использовать NerdLauncher как замену для домашнего экрана устройства. Откройте файл AndroidManifest.xml в NerdLauncher и добавьте следующий фрагмент в главный фильтр интентов.

Листинг 24.10. Изменение категорий NerdLauncher (AndroidManifest.xml)

```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
    <category android:name="android.intent.category.HOME" />
    <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
```

Добавление категорий `HOME` и `DEFAULT` означает, что активность NerdLauncher должна включаться в число вариантов домашнего экрана. Нажмите кнопку Home, и вам будет предложено использовать NerdLauncher (рис. 24.12).

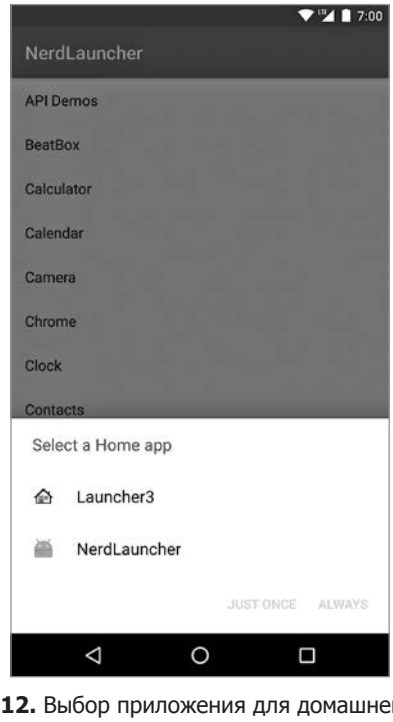


Рис. 24.12. Выбор приложения для домашнего экрана

(Если вы назначите NerdLauncher домашним экраном, а потом захотите отменить свой выбор, запустите приложение Settings из NerdLauncher. Если вы работаете в Lollipop, перейдите в раздел Settings ▸ Apps и выберите в списке приложений NerdLauncher. Если вы работаете в версии Android, предшествующей Lollipop, выполните команду Settings ▸ Applications ▸ Manage Applications. Выберите All, найдите NerdLauncher и сбросьте режим запуска по умолчанию Launch by default при помощи кнопки CLEAR DEFAULTS. При следующем нажатии кнопки Home вы сможете выбрать новый домашний экран по умолчанию.)

Упражнение. Значки

В этой главе мы использовали метод `ResolveInfo.loadLabel(PackageManager)` для отображения содержательных имен в лаунчере. Класс `ResolveInfo` предоставляет аналогичный метод `loadIcon()` для получения значка, отображаемого для каждого приложения. Вам предлагается несложное упражнение: снабдить каждое приложение в NerdLauncher значком.

Для любознательных: процессы и задачи

Для существования любого объекта необходима память и виртуальная машина. *Процесс* представляет собой место, созданное ОС, в котором существуют объекты вашего приложения и в котором выполняется само приложение.

Процессам могут принадлежать ресурсы, находящиеся под управлением ОС, — память, сетевые сокеты, открытые файлы и т. д. Процесс также содержит минимум один (а вероятно, несколько) программный поток (thread). На платформе Android процесс всегда выполняется ровно на одной *виртуальной машине*.

Как правило, каждый компонент приложения в Android связывается ровно с одним процессом (хотя встречаются довольно смутные исключения). Приложение создается с собственным процессом, который становится процессом по умолчанию для всех компонентов приложения.

(Отдельные компоненты можно назначать разным процессам, но мы рекомендуем придерживаться процесса по умолчанию. Если вы думаете, что какой-то код должен выполняться в другом процессе, аналогичного результата обычно удается добиться с использованием многопоточности (multi-threading), которая программируется в Android намного проще, чем многопроцессное выполнение.)

Каждый экземпляр активности существует ровно в одном процессе и ровно в одной задаче. Впрочем, на этом все сходство и завершается. Задачи содержат только активности и часто состоят из активностей разных приложений. С другой стороны, процессы содержат только выполняемый код и объекты приложения.

Процессы и задачи легко спутать, потому что эти концепции отчасти перекрываются, а для ссылок на них часто используются имена приложений. Например, при запуске CriminalIntent из NerdLauncher ОС создает процесс CriminalIntent

и новую задачу, для которой `CrimeListActivity` является базовой активностью. В диспетчере задач эта задача снабжается меткой CriminalIntent.

Задача, в которой существует активность, может быть не связана с процессом, в котором она существует. Для примера возьмем приложение CriminalIntent и контактное приложение и рассмотрим следующий сценарий.

Откройте CriminalIntent, выберите преступление в списке (или добавьте новое) и нажмите кнопку CHOOSE SUSPECT. При этом запускается контактное приложение для выбора подозреваемого. Активность списка контактов добавляется в задачу CriminalIntent. Это означает, что когда пользователь нажимает кнопку Back для перехода между разными активностями, он может незаметно для себя переключаться между процессами.

При этом экземпляр активности списка контактов создается в пространстве памяти процесса контактного приложения и выполняется на виртуальной машине, существующей в процессе контактного приложения. (Состояния экземпляров активностей и задачи в этом сценарии изображены на рис. 24.13.)

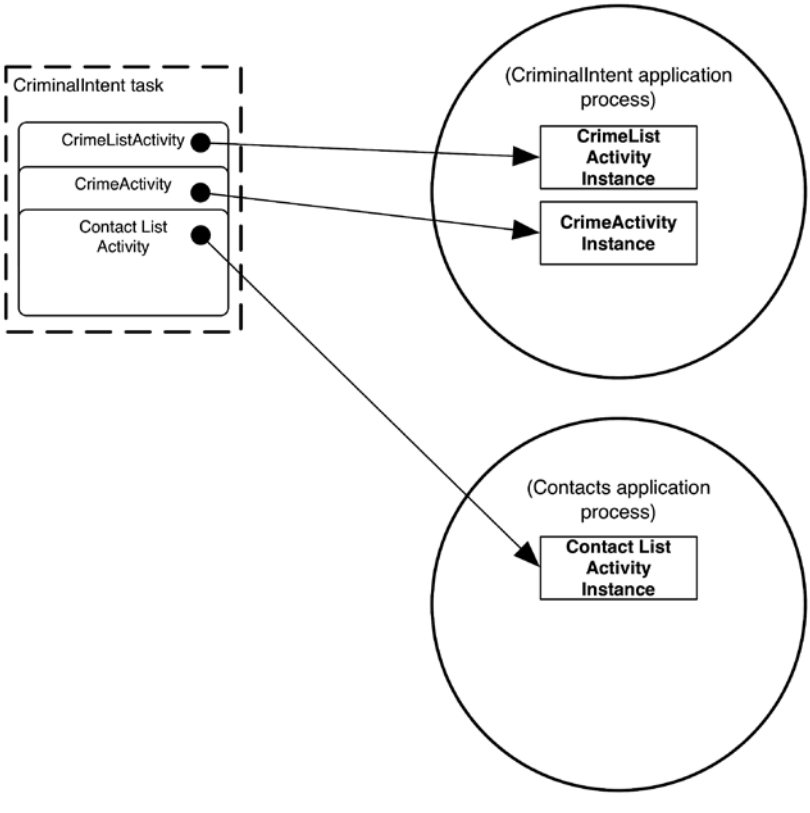


Рис. 24.13. Задачи и процессы

Чтобы продолжить исследование различий между задачами и процессами, оставьте CriminalIntent работать. (Проследите за тем, чтобы само контактное приложение не было представлено в диспетчере задач; если оно там есть, удалите задачу.) Нажмите кнопку Home. Запустите контактное приложение с домашнего экрана. Выберите контакт в списке (или добавьте новый контакт).

При этом экземпляры активности нового списка контактов и подробной информации о контакте будут созданы в процессе контактного приложения. Для контактного приложения будет создана новая задача, которая содержит ссылки на экземпляры активностей списка контактов и подробной информации (рис. 24.14).

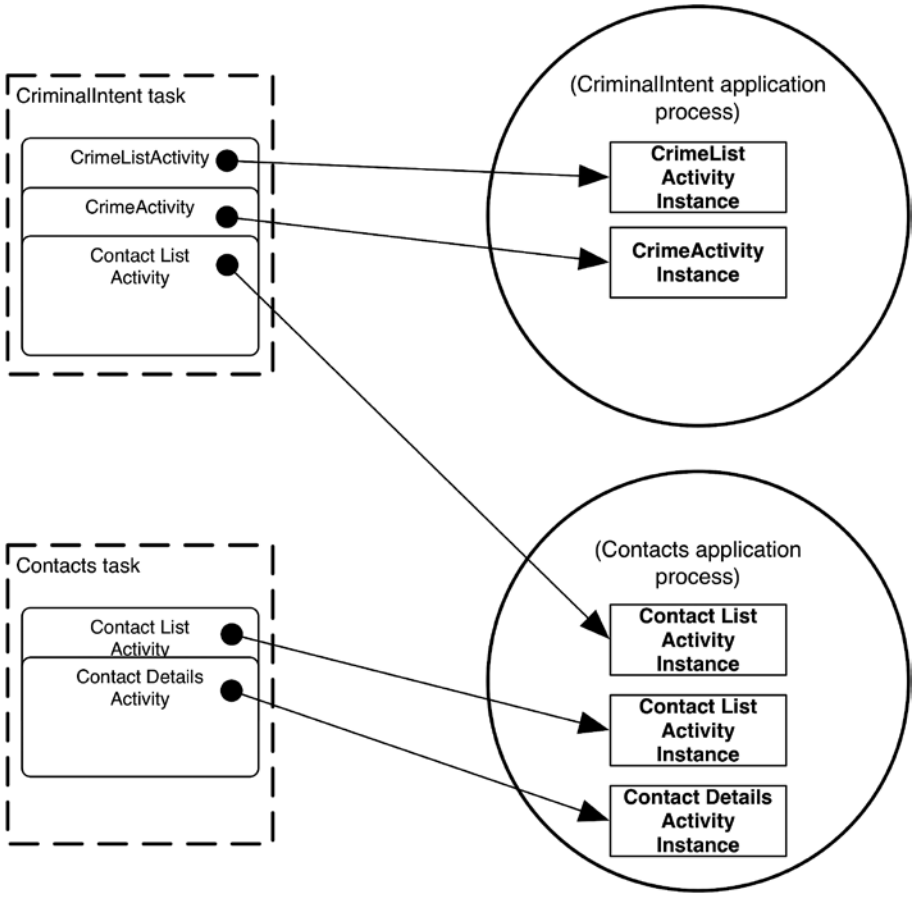


Рис. 24.14. Задачи и процессы

В этой главе мы создавали задачи и переключались между ними. Как насчет замены стандартного диспетчера задач Android? К сожалению, Android не предоставляет средств для решения этой задачи. Также следует знать, что приложения, рекламируемые в магазине Google Play как «уничтожители задач», в действитель-

ности являются уничтожителями процессов. Такие приложения уничтожают конкретный процесс, что может привести к уничтожению активностей, на которые ссылаются задачи других приложений.

Для любознательных: параллельные документы

Запуская приложения на устройстве Lollipop, вы заметите один интересный аспект поведения в отношении CriminalIntent и диспетчера задач. При отправке отчета о преступлении из CriminalIntent активность приложения, выбранного в окне выбора, добавляется в отдельную задачу, а не в задачу CriminalIntent (рис. 24.15).

Начиная с версии Lollipop, для активностей, запускаемых действиями `android.intent.action.SEND` или `action.intent.action.SEND_MULTIPLE`, создаются новые отдельные задачи. (В старых версиях Android этого не происходит, поэтому активность создания сообщений Gmail будет добавлена непосредственно в задачу CriminalIntent.)

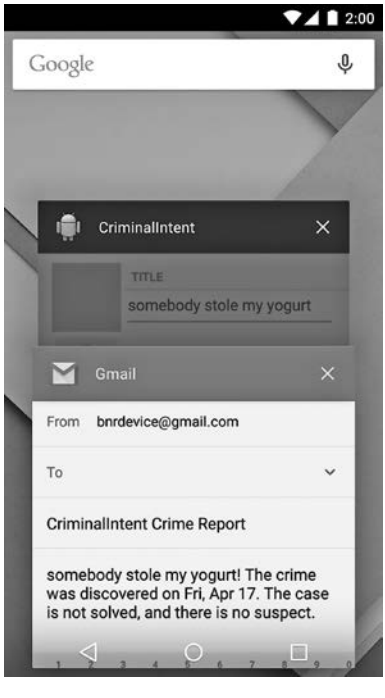


Рис. 24.15. Gmail запускается отдельной задачей

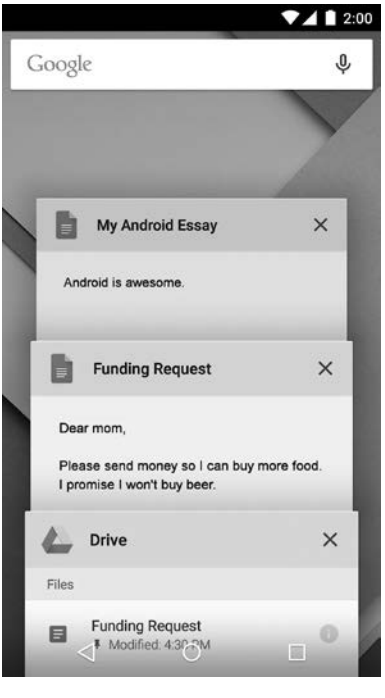


Рис. 24.16. Задачи Google Drive в Lollipop

В этом поведении используется новая концепция Lollipop — параллельные документы (concurrent documents). Механизм параллельных документов позволяет

динамически создать для приложения любое количество задач во время выполнения. До выхода Lollipop приложения могли использовать только заранее определенный набор задач, имена которых должны быть указаны в манифесте.

Типичный пример практического использования параллельных документов — приложение Google Drive. Вы можете открывать и редактировать сразу несколько документов, каждый из которых получает собственную задачу на сводном экране Lollipop (рис. 24.16). Выполнив те же действия с Google Drive на устройстве с версией, предшествующей Lollipop, вы увидите на сводном экране только одну задачу. Это объясняется необходимостью опережающего объявления задач в манифесте в версиях, предшествующих Lollipop. В этих версиях невозможно сгенерировать для приложения динамический набор задач.

Чтобы запустить несколько «документов» (задач) из приложения, работающего на устройстве с Lollipop, либо добавьте флаг `Intent.FLAG_ACTIVITY_NEW_DOCUMENT` в интент перед вызовом `startActivity(…)`, либо присвойте активности в манифесте атрибут `documentLaunchMode`:

```
<activity
    android:name=".CrimePagerActivity"
    android:label="@string/app_name"
    android:parentActivityName=".CrimeListActivity"
    android:documentLaunchMode="intoExisting" />
```

При использовании этого способа для каждого документа будет создаваться только одна задача (и при отправке интента с теми же данными, что у существующей задачи, новая задача создана не будет). Также можно включить режим принудительного создания новой задачи даже в том случае, если она уже существует для данного документа: либо добавьте перед отправкой интента флаг `Intent.FLAG_ACTIVITY_MULTIPLE_TASK` вместе с `Intent.FLAG_ACTIVITY_NEW_DOCUMENT`, либо используйте в манифесте атрибут `documentLaunchMode` со значением `always`.

За дополнительной информацией о сводном экране и изменениях в нем в Lollipop обращайтесь по адресу *developer.android.com/guide/components/recents.html*.

25 HTTP и фоновые задачи

В умах пользователей господствуют сетевые приложения. Чем заняты эти люди, которые за обедом возятся со своими телефонами вместо дружеской беседы? Они маниакально обновляют новостную ленту, отвечают на текстовые сообщения или играют в сетевые игры.

Для экспериментов с сетевыми возможностями Android мы создадим новое приложение PhotoGallery. Это клиент для сайта фотообмена Flickr, который будет загружать и отображать последние общедоступные фото, отправленные на Flickr. Рисунок 25.1 дает примерное представление о внешнем виде приложения.

(Мы добавили в свою реализацию PhotoGallery фильтр, с которым отображаются только фотографии, опубликованные на Flickr «без известных ограничений авторского права». За дополнительной информацией об использовании такого

Рис. 25.1. Приложение PhotoGallery

контента обращайтесь по адресу *www.flickr.com/commons/usage/*. Все остальные фотографии на сайте Flickr являются собственностью человека, отправившего их, и не могут повторно использоваться без разрешения владельца. Дополнительная информация об использовании независимого контента, загруженного с Flickr, доступна по адресу *www.flickr.com/creativecommons/*.)

Приложению PhotoGallery отведено шесть глав. Две главы будут посвящены основам загрузки и разбора JSON, а также выводу изображений. В последующих главах будут реализованы дополнительные функции: поиск, сервисы, оповещения, получатели широковещательных рассылок и веб-представления.

В этой главе вы научитесь использовать высокоуровневые сетевые средства HTTP в Android. Почти все программирование веб-служб в наши дни базируется на сетевом протоколе HTTP. К концу этой главы наше приложение будет загружать с Flickr, разбирать и отображать названия фотографий (рис. 25.2; загрузка и отображение самих фотографий откладывается до главы 26).

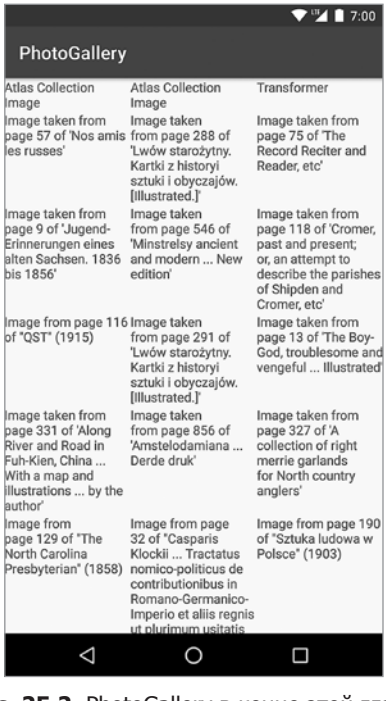


Рис. 25.2. PhotoGallery в конце этой главы

Создание приложения PhotoGallery

Создайте новый проект приложения Android. Задайте его параметры так, как показано на рис. 25.3.

Щелкните на кнопке Next. Задайте форм-фактор Phone and Tablet и выберите в раскрывающемся списке Minimum SDK вариант API 19: Android 4.4 (KitKat).

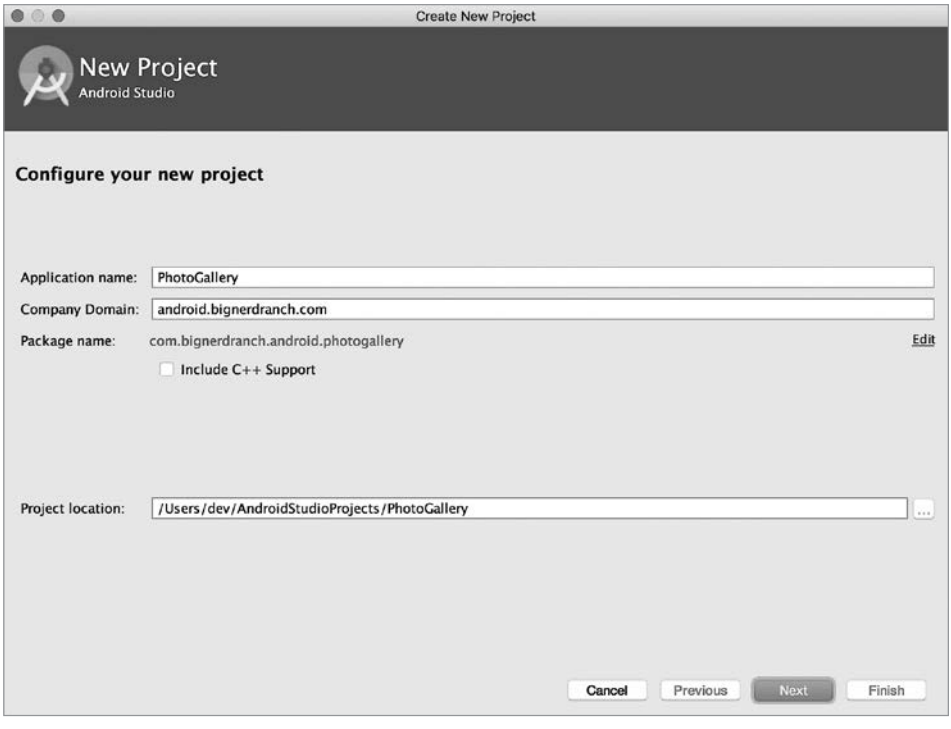


Рис. 25.3. Создание приложения PhotoGallery

Прикажите мастеру создать пустую активность с именем `PhotoGalleryActivity`.

В приложении PhotoGallery используется такая же архитектура, как и во всех предыдущих приложениях. Активность `PhotoGalleryActivity` будет субклассом `SingleFragmentActivity`, а ее представлением будет контейнерное представление, определяемое в файле activity_fragment.xml. Эта активность станет хостом фрагмента, а именно экземпляра `PhotoGalleryFragment`, который мы вскоре создадим.

Скопируйте файлы SingleFragmentActivity.java и activity_fragment.xml в свой проект из предыдущего проекта.

В файле PhotoGalleryActivity.java настройте `PhotoGalleryActivity` как `SingleFragmentActivity`; для этого удалите код, сгенерированный шаблоном, и замените его реализацией `createFragment()`. Метод `createFragment()` должен возвращать экземпляр `PhotoGalleryFragment`. (Пока не обращайте внимания на ошибку, которая будет обнаружена в этом коде. Она исчезнет после того, как вы создадите класс `PhotoGalleryFragment`.)

Листинг 25.1. Настройка активности (PhotoGalleryActivity.java)

```
public class PhotoGalleryActivity extends ~~Activity~~ SingleFragmentActivity {

    @Override
    protected Fragment createFragment() {
```

```
        return PhotoGalleryFragment.newInstance();
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        /* Auto-generated template code... */
    }
}
```

PhotoGallery будет отображать свои результаты в виджете `RecyclerView`, использующем встроенный класс `GridLayoutManager` для размещения элементов в виде таблицы.

Для начала добавьте библиотеку `RecyclerView` в число зависимостей, как это было сделано в главе 8. Откройте окно Project Structure и выберите на левой панели модуль app. Перейдите на вкладку Dependencies и щелкните на кнопке +. Выберите в открывшемся меню вариант Library dependency. Найдите в списке и выделите библиотеку recyclerview-v7, затем щелкните на кнопке OK.

Чтобы создать макет фрагмента, переименуйте файл layout/activity_photo_gallery.xml в layout/fragment_photo_gallery.xml. Затем замените его содержимое определением `RecyclerView`, приведенным на рис. 25.4.

android.support.v7.widget.RecyclerView

```
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/photo_recycler_view"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context="com.bignerdranch.android.photogallery.PhotoGalleryActivity"
```

Рис. 25.4. RecyclerView (layout/fragment_photo_gallery.xml)

Наконец, создайте класс `PhotoGalleryFragment`. Включите удержание фрагмента, заполните созданный макет и инициализируйте переменную класса ссылкой на `RecyclerView` (листинг 25.2).

Листинг 25.2. Заготовка кода (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {

    private RecyclerView mPhotoRecyclerView;

    public static PhotoGalleryFragment newInstance() {
        return new PhotoGalleryFragment();
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setRetainInstance(true);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_photo_gallery, container,
false);

        mPhotoRecyclerView = (RecyclerView) v.findViewById(R.id.photo_
                             recycler_view);
        mPhotoRecyclerView.setLayoutManager(new GridLayoutManager(getActivity(), 3));

        return v;
    }
}
```

(Интересуетесь, зачем мы удерживаем фрагмент вызовом `setRetainInstance(true)`? Не будем забегать вперед — ответ на этот вопрос будет приведен далее, в разделе «Уничтожение AsyncTask».)

Запустите приложение PhotoGallery и убедитесь в том, что все работает правильно (то есть если вы стали гордым владельцем пустого экрана).

Основы сетевой поддержки

В нашем приложении все сетевые взаимодействия PhotoGallery будут обеспечиваться одним классом. Создайте новый класс Java. Поскольку мы будем подключаться к Flickr, назовите класс `FlickrFetchr`.

Исходная версия `FlickrFetchr` будет состоять всего из двух методов: `getUrlBytes(String)` и `getUrlString(String)`. Метод `getUrlBytes(String)` получает низкоуровневые данные по URL и возвращает их в виде массива байтов. Метод `getUrlString(String)` преобразует результат из `getUrlBytes(String)` в `String`.

Добавьте в файл FlickrFetchr.java реализации `getUrlBytes(String)` и `getUrlString(String)` (листинг 25.3).

Листинг 25.3. Основной сетевой код (FlickrFetchr.java)

```
public class FlickrFetchr {
    public byte[] getUrlBytes(String urlSpec) throws IOException {
        URL url = new URL(urlSpec);
        HttpURLConnection connection = (HttpURLConnection)url.openConnection();

        try {
            ByteArrayOutputStream out = new ByteArrayOutputStream();
            InputStream in = connection.getInputStream();
            if (connection.getResponseCode() != HttpURLConnection.HTTP_OK) {
                throw new IOException(connection.getResponseMessage() +
                        ": with " +
                        urlSpec);
```

```
            }

            int bytesRead = 0;
            byte[] buffer = new byte[1024];
            while ((bytesRead = in.read(buffer)) > 0) {
                out.write(buffer, 0, bytesRead);
            }
            out.close();
            return out.toByteArray();
        } finally {
            connection.disconnect();
        }
    }

    public String getUrlString(String urlSpec) throws IOException {
        return new String(getUrlBytes(urlSpec));
    }
}
```

Этот код создает объект URL на базе строки: например, *www.bignerdranch.com*. Затем вызов метода `openConnection()` создает объект подключения к заданному URL-адресу. Вызов `URL.openConnection()` возвращает `URLConnection`, но поскольку подключение осуществляется по протоколу HTTP, мы можем преобразовать его в `HttpURLConnection`. Это открывает доступ к HTTP-интерфейсам для работы с методами запросов, кодами ответов, методами потоковой передачи и т. д.

Объект `HttpURLConnection` представляет подключение, но связь с конечной точкой будет установлена только после вызова `getInputStream()` (или `getOutputStream()` для POST-вызовов). До этого момента вы не сможете получить действительный код ответа.

После создания объекта URL и открытия подключения программа многократно вызывает `read()`, пока в подключении не кончатся данные. Объект `InputStream` предоставляет байты по мере их доступности. Когда чтение будет завершено, программа закрывает его и выдает массив байтов из `ByteArrayOutputStream`.

Хотя всю основную работу выполняет метод `getUrlBytes(String)`, в этой главе мы будем использовать метод `getUrl(String)`. Он преобразует байты, полученные вызовом `getUrlBytes(String)`, в `String`. На данный момент это решение смотрится немного странно — зачем разбивать выполняемую работу на два метода? Однако наличие двух методов будет полезно в следующей главе, когда мы займемся загрузкой данных изображений.

Разрешение на работу с сетью

Для работы сетевой поддержки необходимо сделать еще одно: вы должны попросить разрешения. Никому из пользователей не понравится, если вы будете тайно загружать их фотографии.

Чтобы запросить разрешение на работу с сетью, добавьте следующую строку в файл AndroidManifest.xml.

Листинг 25.4. Включение разрешения на работу с сетью в манифест

```
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bignerdranch.android.photogallery" >

    <uses-permission android:name="android.permission.INTERNET" />

    <application
      ...
    </application>

</manifest>
```

Когда пользователь пытается загрузить ваше приложение, открывается диалоговое окно с этими разрешениями. Пользователь может подтвердить или отказаться от установки.

Android не считает разрешение `INTERNET` опасным, потому что оно необходимо многим приложениям. В результате от вас потребуется лишь одно: объявить это разрешение в манифесте. Более опасные разрешения (например, разрешение на определение текущей позиции устройства) также требуют запроса на стадии выполнения. (Более подробная информация о них приведена в главе 33.)

Использование AsyncTask для выполнения в фоновом потоке

На следующем шаге мы должны вызвать и протестировать только что добавленный сетевой код. Однако мы не можем просто вызвать `FlickrFetchr.getURL(String)` прямо из `PhotoGalleryFragment`. Вместо этого необходимо создать фоновый программный поток и выполнить код в нем.

Для работы с фоновыми потоками проще всего использовать вспомогательный класс с именем `AsyncTask`. `AsyncTask` создает фоновый поток и выполняет в нем код, содержащийся в методе `doInBackground(…)`.

В файле PhotoGalleryFragment.java добавьте в конце `PhotoGalleryFragment` новый внутренний класс с именем `FetchItemsTask`. Переопределите метод `AsyncTask.doInBackground(…)` для получения данных с сайта и их регистрации в журнале.

Листинг 25.5. Реализация AsyncTask, часть I (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {

    private static final String TAG = "PhotoGalleryFragment";

    private RecyclerView mPhotoRecyclerView;
    ...
    private class FetchItemsTask extends AsyncTask<Void,Void,Void> {
        @Override
        protected Void doInBackground(Void... params) {
            try {
```

```
                String result = new FlickrFetchr()
                        .getUrlString("https://www.bignerdranch.com");
                Log.i(TAG, "Fetched contents of URL: " + result);
            } catch (IOException ioe) {
                Log.e(TAG, "Failed to fetch URL: ", ioe);
            }
            return null;
        }
    }
}
```

Затем в методе `PhotoGalleryFragment.onCreate(…)` вызовите `execute()` для нового экземпляра `FetchItemsTask`.

Листинг 25.6. Реализация AsyncTask, часть I (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {

    private static final String TAG = "PhotoGalleryFragment";

    private RecyclerView mPhotoRecyclerView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRetainInstance(true);
        new FetchItemsTask().execute();
    }
    ...
}
```

Вызов `execute()` активизирует класс `AsyncTask`, который запускает свой фоновый поток и вызывает `doInBackground(…)`. Выполните свой код, и вы увидите, что в LogCat появляется разметка HTML домашней страницы Big Nerd Ranch — примерно так, как показано на рис. 25.5.

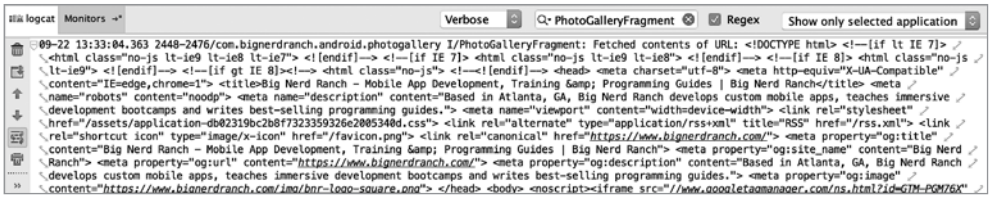


Рис. 25.5. Разметка HTML от Big Nerd Ranch в LogCat

Найти нужные результаты на панели LogCat может быть нелегко. Попробуйте поискать что-нибудь конкретное. В данном случае введите в поле поиска LogCat строку «PhotoGalleryFragment», как показано на рисунке.

Теперь, когда мы создали фоновый поток и выполнили в нем сетевой код, давайте поближе познакомимся с программными потоками в Android.

Главный программный поток

Сетевые взаимодействия не происходят моментально. Веб-серверу может потребоваться одна-две секунды на ответ, а загрузка файла может занять еще больше времени. Из-за продолжительности сетевых операций Android запрещает их выполнение в *главном программном потоке*. Если вы попытаетесь нарушить это ограничение, Android выдает исключение `NetworkOnMainThreadException`.

Почему? Чтобы понять это, необходимо понимать, что такое *программный поток* (thread), что собой представляет главный программный поток приложения и что он делает.

Программным потоком (thread) называется отдельная последовательность выполнения команд программы. Жизненный цикл каждого приложения Android начинается с *главного потока*. Однако главный поток не является заранее определенной последовательностью действий. Он в бесконечном цикле ожидает событий, инициированных пользователем или системой, и выполняет код как реакцию на эти события по мере их возникновения (рис. 25.6).

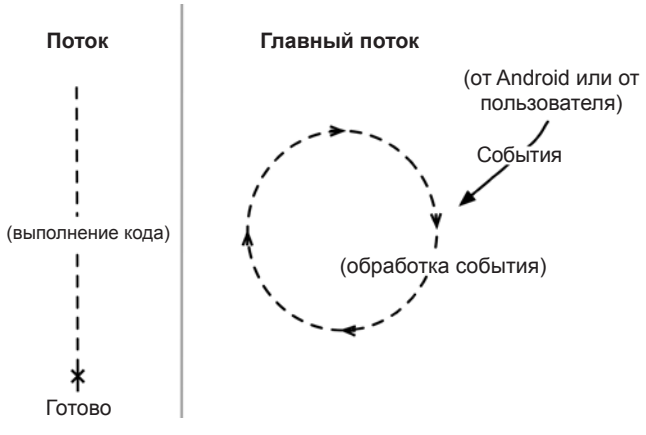


Рис. 25.6. Обычные потоки и главный поток

Представьте, что ваше приложение — огромный обувной магазин и у вас всего один работник — Флэш[1]. В магазине необходимо многое делать для того, чтобы покупатели остались довольны, — расставлять товары на полках, приносить обувь для примерки, определять размер ноги покупателя. С таким продавцом, как Флэш, все делается своевременно, хотя всю работу выполняет всего один человек.

Чтобы магазин нормально работал, Флэш не может тратить слишком много времени на что-то одно. Что, если часть товара будет распродана? Кому-то придется потратить много времени за беседой по телефону, заказывая у поставщика новую партию. Пока Флэш будет занят, покупатели начнут сердиться.

[1] Персонаж комиксов, наделенный способностью к сверхъестественно быстрым перемещениям. — *Примеч. пер.*

Флэш — аналог главного потока вашего приложения. Он выполняет весь код, обновляющий пользовательский интерфейс. В частности, сюда относится код реакции на различные события пользовательского интерфейса — запуск активностей, нажатия кнопок и т. д. (Поскольку все события тем или иным образом связаны с пользовательским интерфейсом, главный поток иногда называется потоком пользовательского интерфейса, или *UI-потоком*.)

Цикл событий обеспечивает последовательное выполнение кода пользовательского интерфейса. Он гарантирует, что операции не будут «перебегать дорогу» друг другу, одновременно обеспечивая своевременное выполнение кода. Таким образом, весь написанный вами код (за исключением кода, написанного для `AsyncTask`) выполнялся в главном потоке.

Кроме главного потока

Сетевые операции можно сравнить с телефонным звонком поставщику обуви: они занимают много времени по сравнению с другими задачами. В это время пользовательский интерфейс будет полностью парализован, что может привести к ошибке *ANR* (Application Not Responding).

Эта ошибка происходит тогда, когда система мониторинга Android обнаруживает, что главный поток не среагировал на важное событие, такое как нажатие кнопки Back. Для пользователя это выглядит так, как показано на рис. 25.7.

Рис. 25.7. Приложение не отвечает

В обувном магазине вы бы решили проблему вполне естественным образом — наняли бы второго работника для звонков поставщику. Похожее решение используется и в Android: вы создаете *фоновый поток* и выполняете сетевые операции из него.

А как проще всего работать с фоновым потоком? При помощи `AsyncTask`.

Позднее в этой главе вы увидите, на что еще способен класс `AsyncTask`. Но прежде чем изучать его возможности, давайте выполним какую-нибудь реальную работу в сетевом коде.

Загрузка XML из Flickr

Формат *JSON* (JavaScript Object Notation) стал популярным в последнее время, особенно в области веб-служб. Android включает стандартный пакет `org.json`, классы которого предоставляют средства для создания и разбора файлов в формате JSON. В документации разработчика Android приведено описание `org.json`, а более подробную информацию о формате JSON можно получить по адресу *json.org*.

Flickr предлагает удобный JSON API. Вся необходимая информация доступна в документации по адресу *www.flickr.com/services/api/*. Загрузите ее в своем браузере и найдите список Request Formats. Мы будем использовать простейший формат — REST, соответственно конечной точкой API становится адрес *https://api.flickr.com/services/rest/*. Вы можете вызывать методы, которые Flickr предоставляет в этой конечной точке.

Вернитесь к главной странице документации API и найдите список API Methods. Прокрутите его до раздела photos и найдите элемент `flickr.photos.getRecent`. Щелкните на нем; в открывшейся документации говорится, что этот метод «Возвращает список последних общедоступных фотографий, отправленных на flickr». Это именно то, что нам нужно для PhotoGallery.

Единственным обязательным параметром метода `getRecent` является ключ API. Чтобы получить ключ API, обратитесь по адресу *www.flickr.com/services/api/* и проследуйте по ссылке API keys. Для входа вам понадобится идентификатор Yahoo. После входа зарегистрируйте новый некоммерческий ключ API; обычно эта процедура занимает несколько секунд. Ваш ключ API будет выглядеть примерно так: *4f721bgafa75bf6d2cb9af54f937bb70*. («Секрет» в данном случае не нужен — он используется только при работе с пользовательской информацией или изображениями.)

После получения ключа вам остается лишь обратиться с запросом к веб-службе Flickr. URL-адрес GET-запроса должен выглядеть примерно так:

https://api.flickr.com/services/rest/?method=flickr.photos.getRecent&api_key=xxx&format=json&nojsoncallback=1.

По умолчанию Flickr возвращает ответ в формате XML. Чтобы получить действительный ответ в формате JSON, необходимо задать значения параметров `format` и `nojsoncallback`. Присваивая `nojsoncallback` значение 1, вы приказываете Flickr исключить имя метода и круглые скобки из возвращаемого ответа. Это необходимо для того, чтобы упростить разбор ответа в коде Java.

Скопируйте URL-адрес из примера в свой браузер, заменив «xxx» в значении `api_key` фактическим ключом API. Примерный вид данных ответа показан на рис. 25.8.

Рис. 25.8. Пример вывода JSON

Переходим к программированию. Начнем с добавления нескольких констант в `FlickrFetchr`.

Листинг 25.7. Добавление констант (FlickrFetchr.java)

```
public class FlickrFetchr {
    private static final String TAG = "FlickrFetchr";

    private static final String API_KEY = "ваш_ключ_Api";
    ...
}
```

Не забудьте заменить *ваш_ключ_Api* ключом API, сгенерированным ранее.

Используйте эти константы для написания метода, который строит соответствующий URL-адрес запроса и загружает его содержимое.

Листинг 25.8. Добавление метода fetchItems() (FlickrFetchr.java)

```
public class FlickrFetchr {
    ...
    public String getUrlString(String urlSpec) throws IOException {
        return new String(getUrlBytes(urlSpec));
    }

    public void fetchItems() {
        try {
            String url = Uri.parse("https://api.flickr.com/services/rest/")
                    .buildUpon()
                    .appendQueryParameter("method", "flickr.photos.getRecent")
                    .appendQueryParameter("api_key", API_KEY)
                    .appendQueryParameter("format", "json")
                    .appendQueryParameter("nojsoncallback", "1")
                    .appendQueryParameter("extras", "url_s")
                    .build().toString();
            String jsonString = getUrlString(url);
            Log.i(TAG, "Received JSON: " + jsonString);
```

```
        } catch (IOException ioe) {
            Log.e(TAG, "Failed to fetch items", ioe);
        }
    }
}
```

Здесь мы используем класс `Uri.Builder` для построения полного URL-адреса для API-запроса к Flickr. `Uri.Builder` — вспомогательный класс для создания параметризованных URL-адресов с правильным кодированием символов. Метод `Uri.Builder.appendQueryParameter(String,String)` автоматически кодирует строки запросов.

Обратите внимание на добавленные значения параметров `method`, `api_key`, `format` и `nojsoncallback`. Мы также задали дополнительный параметр `extras` со значением `url_s`. Значение `url_s` приказывает Flickr включить URL-адрес для уменьшенной версии изображения, если оно доступно.

Наконец, измените код `AsyncTask` в `PhotoGalleryFragment` для вызова нового метода `fetchItems()`.

Листинг 25.9. Вызов fetchItems() (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {
    ...
    private class FetchItemsTask extends AsyncTask<Void,Void,Void> {
        @Override
        protected Void doInBackground(Void... params) {
            try {
                String result = new FlickrFetchr()
                        .getUrlString("https://www.bignerdranch.com");
                Log.i(TAG, "Fetched contents of URL: " + result);
            } catch (IOException ioe) {
                Log.e(TAG, "Failed to fetch URL: ", ioe);
            }
            new FlickrFetchr().fetchItems();
            return null;
        }
    }
}
```

Запустите приложение PhotoGallery. В LogCat отображается полноценная, настоящая разметка JSON (рис. 25.9). Поиск по строке «FlickrFetcher» поможет найти нужную информацию.

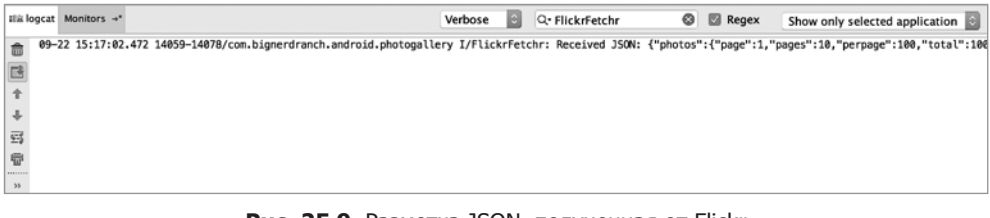


Рис. 25.9. Разметка JSON, полученная от Flickr

К сожалению, на момент написания книги панель LogCat в Android Studio не обеспечивала нормального переноса разметки. Чтобы увидеть продолжение чрезвычайно длинной строки ответа JSON, прокрутите содержимое панели вправо. (LogCat порой ведет себя загадочно. Не паникуйте, если ваши результаты отличаются от наших. Иногда подключение к эмулятору работает не совсем корректно, и регистрируемые сообщения не выводятся. Обычно со временем проблема исчезает, но иногда приходится запускать приложение заново и даже перезапускать эмулятор.)

Итак, разметка JSON с Flickr получена; что теперь с ней делать? То же, что мы делаем со всеми данными, — поместить в один или несколько объектов модели. Класс модели, который мы создадим для PhotoGallery, называется `GalleryItem`. На рис. 25.10 изображена диаграмма объектов PhotoGallery.

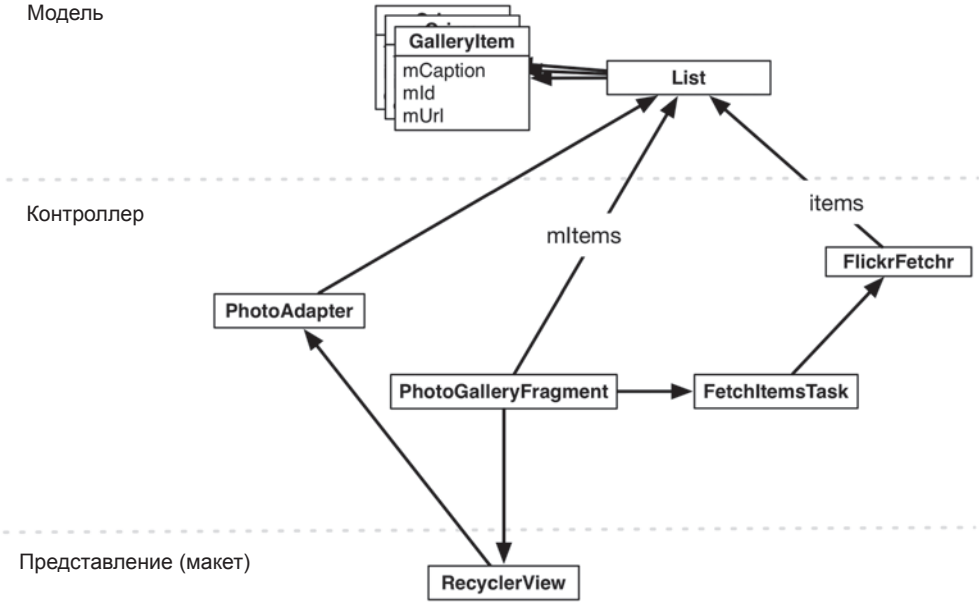


Рис. 25.10. Диаграмма объектов PhotoGallery

Обратите внимание: на рис. 25.10 не показана активность-хост, чтобы диаграмма была сконцентрирована на фрагментах и сетевом коде.

Создайте класс `GalleryItem` и добавьте следующий код.

Листинг 25.10. Создание класса объекта модели (GalleryItem.java)

```
public class GalleryItem {
    private String mCaption;
    private String mId;
    private String mUrl;

    @Override
```

```
    public String toString() {
        return mCaption;
    }
}
```

Прикажите Android Studio сгенерировать `get-` и `set`-методы для `mCaption`, `mId` и `mUrl`.

После того как объекты модели будут созданы, их следует заполнить данными из разметки JSON, полученной от Flickr.

Разбор текста в формате JSON

Ответ JSON, отображаемый в браузере и на панели LogCat, плохо читается. Если отформатировать ответ с отступами, он будет выглядеть примерно так, как показано на рис. 25.11.

```
{
  "photos": {
    "page": 1,
    "pages": 10,
    "perpage": 100,
    "total": 1000,
    "photo": [
      {
        "id": "9452133594",
        "owner": "44494372@N05",
        "secret": "d6d20af93e",
        "server": "7365",
        "farm": 8,
        "title": "Low and Wisoff at Work",
        "ispublic": 1,
        "isfriend": 0,
        "isfamily": 0
      }, ...
      {
        "id": "16317817559",
        "owner": "44494372@N05",
        "secret": "137d97804f",
        "server": "8683",
        "farm": 9,
        "title": "Challenger as seen from SPAS",
        "ispublic": 1,
        "isfriend": 0,
        "isfamily": 0
      }
    ]
  },
  "stat": "ok"
}
```

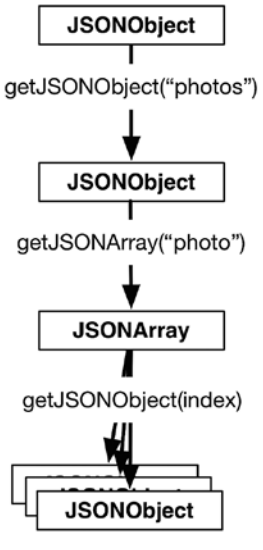


Рис. 25.11. Иерархия JSON

Объект JSON представляет собой набор пар «имя-значение», заключенный в фигурные скобки `{ }`. Массив JSON представляет собой разделенный запятыми список объектов JSON, заключенный в квадратные скобки `[ ]`. Объекты могут вкладываться друг в друга, образуя иерархии.

API `json.org` предоставляет объекты Java, соответствующие тексту JSON, такие как `JSONObject` и `JSONArray`. Тексты JSON легко разбираются в соответствующие объекты Java при помощи конструктора `JSONObject(String)`. Внесите соответствующие изменения в метод `fetchItems()`.

Листинг 25.11. Чтение строки JSON в JSONObject (FlickrFetchr.java)

```
public class FlickrFetchr {

    private static final String TAG = "FlickrFetchr";
    ...
    public void fetchItems() {
        try {
            ...
            Log.i(TAG, "Received JSON: " + jsonString);
            JSONObject jsonBody = new JSONObject(jsonString);
        } catch (IOException ioe) {
            Log.e(TAG, "Failed to fetch items", ioe);
        } catch (JSONException je){
            Log.e(TAG, "Failed to parse JSON", je);
        }
    }
}
```

Конструктор `JSONObject` разбирает переданную строку JSON и строит иерархию объектов, соответствующую исходному тексту JSON. Иерархия объектов JSON, полученная от Flickr, показана на рис. 25.11.

Мы получаем объект `JSONObject` верхнего уровня, соответствующий внешним фигурным скобкам в исходном тексте JSON. Объект верхнего уровня содержит вложенный объект `JSONObject` с именем `photos`. Во вложенном объекте `JSONObject` находится объект `JSONArray` с именем `photo`. Этот массив содержит набор объектов `JSONObject`, каждый из которых представляет метаданные одной фотографии.

Напишите метод для извлечения информации каждой фотографии. Создайте для каждой фотографии объект `GalleryItem` и добавьте его в список.

Листинг 25.12. Разбор фотографий Flickr (FlickrFetchr.java)

```
public class FlickrFetchr {

    private static final String TAG = "FlickrFetchr";
    ...
    public void fetchItems() {
        ...
    }

    private void parseItems(List<GalleryItem> items, JSONObject jsonBody)
```

```
            throws IOException, JSONException {

        JSONObject photosJsonObject = jsonBody.getJSONObject("photos");
        JSONArray photoJsonArray = photosJsonObject.getJSONArray("photo");

        for (int i = 0; i < photoJsonArray.length(); i++) {
            JSONObject photoJsonObject = photoJsonArray.getJSONObject(i);

            GalleryItem item = new GalleryItem();
            item.setId(photoJsonObject.getString("id"));
            item.setCaption(photoJsonObject.getString("title"));

            if (!photoJsonObject.has("url_s")) {
                continue;
            }

            item.setUrl(photoJsonObject.getString("url_s"));
            items.add(item);
        }
    }
}
```

Этот код использует такие вспомогательные методы, как `getJSONObject(String name)` и `getJSONArray(String name)`, для перемещения по иерархии `JSONObject`. (Эти методы также показаны на рис. 25.11.)

Flickr не всегда возвращает компонент `url_s` для каждого изображения. Добавьте проверку для игнорирования изображений, не имеющих URL-адреса изображения.

Методу `parseItems(…)` необходим контейнер `List` и `JSONObject`. Обновите метод `fetchItems()` так, чтобы он вызывал `parseItems(…)` и возвращал `List` с объектами `GalleryItem`.

Листинг 25.13. Вызов parseItems(…) (FlickrFetchr.java)

```
public void List<GalleryItem> fetchItems() {

    List<GalleryItem> items = new ArrayList<>();

    try {
        String url = ...;
        String jsonString = getUrlString(url);
        Log.i(TAG, "Received JSON: " + jsonString);
        JSONObject jsonBody = new JSONObject(jsonString);
        parseItems(items, jsonBody);
    } catch (JSONException je) {
            Log.e(TAG, "Failed to parse JSON", je);
    } catch (IOException ioe) {
        Log.e(TAG, "Failed to fetch items", ioe);
    }

    return items;
}
```

Запустите приложение PhotoGallery и протестируйте код разбора JSON. У PhotoGallery пока нет средств для вывода информации о содержимом `List`, поэтому если вы захотите убедиться в том, что все работает правильно, вам придется установить точку прерывания в отладчике.

От AsyncTask к главному потоку

Напоследок мы вернемся к уровню представления и выведем некоторые названия фотографий в виджете `RecyclerView` экземпляра `PhotoGalleryFragment`.

Начнем с определения `ViewHolder` во внутреннем классе.

Листинг 25.14. Добавление реализации ViewHolder (PhotoGalleryFragment.java)

```
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        ...
    }

    private class PhotoHolder extends RecyclerView.ViewHolder {
        private TextView mTitleTextView;

        public PhotoHolder(View itemView) {
            super(itemView);

            mTitleTextView = (TextView) itemView;
        }

        public void bindGalleryItem(GalleryItem item) {
            mTitleTextView.setText(item.toString());
        }
    }

    private class FetchItemsTask extends AsyncTask<Void,Void,Void> {
        ...
    }
}
```

Затем добавьте класс `RecyclerView.Adapter`, который будет предоставлять необходимые объекты `PhotoHolder` на основании списка `GalleryItem`.

Листинг 25.15. Добавление реализации RecyclerView.Adapter (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {

    private static final String TAG = "PhotoGalleryFragment";
    ...

    private class PhotoHolder extends RecyclerView.ViewHolder {
```

```
        ...
    }

    private class PhotoAdapter extends RecyclerView.Adapter<PhotoHolder> {

        private List<GalleryItem> mGalleryItems;

        public PhotoAdapter(List<GalleryItem> galleryItems) {
            mGalleryItems = galleryItems;
        }

        @Override
        public PhotoHolder onCreateViewHolder(ViewGroup viewGroup, int viewType) {
            TextView textView = new TextView(getActivity());
            return new PhotoHolder(textView);
        }

        @Override
        public void onBindViewHolder(PhotoHolder photoHolder, int position) {
            GalleryItem galleryItem = mGalleryItems.get(position);
            photoHolder.bindGalleryItem(galleryItem);
        }

        @Override
        public int getItemCount() {
            return mGalleryItems.size();
        }
    }
    ...
}
```

Вся необходимая инфраструктура для `RecyclerView` готова. Перейдем к добавлению кода настройки и присоединению адаптера.

Листинг 25.16. Реализация setupAdapter() (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {

    private static final String TAG = "PhotoGalleryFragment";

    private RecyclerView mPhotoRecyclerView;
    private List<GalleryItem> mItems = new ArrayList<>();
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_photo_gallery, container,
                false);
        mPhotoRecyclerView = (RecyclerView) v.findViewById(R.id.photo_
                                recycler_view);
        mPhotoRecyclerView.setLayoutManager(new GridLayoutManager
                           (getActivity(), 3));

        setupAdapter();
```

```
        return v;
    }

    private void setupAdapter() {
        if (isAdded()) {
            mPhotoRecyclerView.setAdapter(new PhotoAdapter(mItems));
        }
    }
    ...
}
```

Только что добавленный метод `setupAdapter()` проверяет текущее состояние модели (а именно список `List` объектов `GalleryItem`) и соответствующим образом настраивает адаптер для `RecyclerView`. Метод `setupAdapter()` вызывается в `onCreateView(…)`, чтобы каждый раз при создании нового объекта `RecyclerView` он связывался с подходящим адаптером. Метод также должен вызываться при каждом изменении набора объектов модели.

Обратите внимание на проверку `isAdded()` перед назначением адаптера. Проверка подтверждает, что фрагмент был присоединен к активности, а следовательно, что результат `getActivity()` будет отличен от `null`.

Помните, что фрагменты могут существовать и автономно, не будучи связанными с какой-либо активностью. До настоящего момента мы не сталкивались с такой возможностью, потому что вызовы методов управлялись обратными вызовами от инфраструктуры. Если фрагмент получает обратные вызовы, он определенно присоединен к активности. Нет активности — нет обратных вызовов.

Теперь, когда мы используем `AsyncTask`, некоторые действия инициируются самостоятельно, и мы уже не можем предполагать, что фрагмент присоединен к активности. Таким образом, сначала необходимо убедиться в том, что фрагмент остается присоединенным; в противном случае попытка выполнения операций, зависящих от активности (например, создания объекта `PhotoAdapter`, при котором создается виджет `TextView` с использованием активности-хоста как контекста), завершится неудачей. Вот почему в приведенном выше коде перед назначением адаптера мы проверяем `isAdded()` на истинность.

После получения данных от Flickr следует вызвать метод `setupAdapter()`. Первое, что приходит в голову, — вызов `setupAdapter()` в конце метода `doInBackground(…)` класса `FetchItemsTask`. Это не лучшая мысль. Вспомните, что сейчас «в магазине работают два Флэша» — один обслуживает многочисленных покупателей, другой общается по телефону с Flickr. Что произойдет, если второй Флэш, повесив трубку, захочет подключиться к обслуживанию покупателей? Скорее всего, два Флэша будут только мешать друг другу.

На компьютере такая путаница может привести к повреждению объектов в памяти. По этой причине фоновым потокам запрещается обновлять пользовательский интерфейс, поскольку такие операции явно небезопасны.

Что делать? У `AsyncTask` имеется метод `onPostExecute(…)`, который можно переопределить. Метод `onPostExecute(…)` выполняется после завершения `doInBack-`

ground(…). Кроме того, onPostExecute(…) выполняется в главном, а не в фоновом потоке, поэтому обновление пользовательского интерфейса в нем безопасно.

Внесите изменения в метод FetchItemsTask, чтобы он обновлял поле mItems и вызывал setupAdapter() после загрузки фотографий для обновления источника данных RecyclerView.

Листинг 25.17. Добавление кода обновления адаптера (PhotoGalleryFragment.java)

```
private class FetchItemsTask extends AsyncTask<Void,Void,~~Void~~ List<GalleryItem>> {
    @Override
    protected ~~Void~~ List<GalleryItem> doInBackground(Void... params) {

        return new FlickrFetchr().fetchItems();
        ~~return null;~~
    }

    @Override
    protected void onPostExecute(List<GalleryItem> items) {
        mItems = items;
        setupAdapter();
    }
}
```

Мы внесли три изменения. Во-первых, мы изменили тип третьего обобщенного параметра FetchItemsTask. Этот параметр определяет тип результата, производимого AsyncTask; он задает тип значения, возвращаемого doInBackground(…), а также тип входного параметра onPostExecute(…).

Во-вторых, мы изменили метод doInBackground(…) так, чтобы он возвращал список элементов GalleryItem. Тем самым мы устранили ошибку в коде и обеспечили его нормальную компиляцию. Также при вызове передается список элементов, чтобы он мог использоваться в коде onPostExecute(…).

В-третьих, была добавлена реализация onPostExecute(…). Этот метод получает список, загруженный в doInBackground(…), помещает его в mItems и обновляет адаптер RecyclerView.

На этом наша работа для этой главы завершается. Запустите приложение, и вы увидите текст, отображаемый для каждого загруженного элемента GalleryItem (см. рис. 25.2).

Уничтожение AsyncTask

В этой главе реализация AsyncTask была тщательно структурирована таким образом, чтобы нам не приходилось хранить информацию о ней. Например, мы удерживали фрагмент (вызовом setRetainInstance(true)), чтобы поворот не приводил к многократному порождению новых объектов AsyncTask для загрузки данных JSON. Однако в других ситуациях может возникнуть необходимость в отслеживании AsyncTask и даже их периодической отмене и повторном запуске.

В подобных более сложных сценариях использования `AsyncTask` присваивается переменной экземпляра, после чего для нее можно вызвать `AsyncTask.cancel(boolean)` для отмены текущей фоновой операции `AsyncTask`.

`AsyncTask.cancel(boolean)` может работать в более жестком или менее жестком режиме. Если вызвать `cancel(false)`, метод действует менее жестко и просто возвращает `true` при вызове `isCancelled()`. Далее `AsyncTask` проверяет `isCancelled()` внутри `doInBackground(…)` и принимает решение о досрочном завершении операции.

Но в случае вызова `cancel(true)` метод действует жестко и прерывает программный поток, в котором выполняется `doInBackground(…)`. Вызов `AsyncTask.cancel(true)` является агрессивным способом остановки `AsyncTask`. Если этого можно избежать, лучше так и сделать.

Где и как следует отменять задачи `AsyncTask`? Зависит от обстоятельств. Сначала спросите себя: должна ли работа, выполняемая `AsyncTask`, останавливаться, если фрагмент или активность уничтожается или становится невидимой? Если да, экземпляр `AsyncTask` следует отменить либо в `onStop(…)` (чтобы отменить задачу, когда представление стало невидимым), либо в `onDestroy(…)` (чтобы отменить задачу при уничтожении экземпляра фрагмента/активности).

А если вы хотите, чтобы работа, выполняемая `AsyncTask`, не ограничивалась сроками жизни фрагмента/активности и их представления? Можно просто приказать `AsyncTask` выполняться до завершения без отмены. Однако при этом появляется опасность утечки памяти (например, экземпляр `Activity` продолжает существовать после того момента, когда он должен быть уничтожен) или возникновения проблем, связанных с обновлением или обращением к пользовательскому интерфейсу, находящемуся в недействительном состоянии. Если некая важная работа должна быть завершена независимо от того, чем занимается пользователь, лучше рассмотреть альтернативы, например запуск службы (см. главу 28).

Для любознательных: подробнее об AsyncTask

В этой главе вы видели пример использования последнего параметра-типа `AsyncTask`, определяющего возвращаемый тип. А как насчет двух других параметров?

Первый параметр-тип позволяет задать тип входных параметров, передаваемых `execute()`, которые в свою очередь определяют тип параметров, получаемых `doInBackground(…)`. Он используется следующим образом:

```
AsyncTask<String,Void,Void> task = new AsyncTask<String,Void,Void>() {
    public Void doInBackground(String... params) {
        for (String parameter : params) {
            Log.i(TAG, "Received parameter: " + parameter);
        }

        return null;
    }
};
```

Входные параметры передаются методу `execute(…)`, который вызывается с переменным числом аргументов:

```
task.execute("First parameter", "Second parameter", "Etc.");
```

Эти переменные аргументы передаются `doInBackground(…)`.

Второй параметр-тип позволяет задать тип для передачи информации о ходе выполнения операции. Вот как это выглядит:

```
final ProgressBar gestationProgressBar = /* Индикатор прогресса */;
gestationProgressBar.setMax(42); /* Максимальный срок развития */

AsyncTask<Void,Integer,Void> haveABaby = new AsyncTask<Void,Integer,Void>() {
    public Void doInBackground(Void... params) {
        while (!babyIsBorn()) {
            Integer weeksPassed = getNumberOfWeeksPassed();
            publishProgress(weeksPassed);
            patientlyWaitForBaby();
        }
    }

    public void onProgressUpdate(Integer... params) {
        int progress = params[0];
        gestationProgressBar.setProgress(progress);
    }
};

/* Вызывается для выполнения AsyncTask */
haveABaby.execute();
```

Обновление обычно выполняется в продолжительном фоновом процессе. Проблема в том, что необходимые обновления пользовательского интерфейса не могут выполняться прямо из фонового процесса, поэтому `AsyncTask` предоставляет методы `publishProgress(…)` и `onProgressUpdate(…)`.

Механизм обновления работает следующим образом: в методе `doInBackground(…)` в фоновом потоке вызывается `publishProgress(…)`. Это приводит к вызову `onProgressUpdate(…)` в потоке пользовательского интерфейса. Таким образом, пользовательский интерфейс обновляется в `onProgressUpdate(…)`, но управление обновлениями осуществляется вызовами `publishProgress(…)` в `doInBackground(…)`.

Для любознательных: альтернативы для AsyncTask

Если объекты `AsyncTask` используются для загрузки данных, вы отвечаете за управление их жизненным циклом во время изменений конфигурации (например, поворотов) и за сохранение данных в месте, в котором они смогут эти изменения пережить. Часто эта задача упрощается вызовом `setRetainInstance(true)` для фрагмента и сохранением данных во фрагменте, но в некоторых ситуациях

вам приходится вмешиваться и писать код, обеспечивающий правильность всех выполняемых действий. К числу таких ситуаций относится нажатие пользователем кнопки Back во время выполнения `AsyncTask` или уничтожение фрагмента, запустившего `AsyncTask`, из-за нехватки памяти.

Класс `Loader` предоставляет альтернативное решение, которое снимает с вас часть (но только часть!) ответственности. Этот класс предназначен для загрузки некоторых данных (объекта) из некоторого источника: диска, базы данных, `ContentProvider`, сети или другого процесса.

`AsyncTaskLoader` — абстрактная специализация `Loader`, использующая `AsyncTask` для вынесения работы по загрузке данных в другой поток. Почти все полезные классы `Loader`, которые вам придется создавать, будут субклассами `AsyncTaskLoader`. Класс `AsyncTaskLoader` обеспечит загрузку данных без блокирования главного потока и доставит результаты стороне, заинтересованной в их получении.

Зачем использовать `Loader`, скажем, вместо прямого использования `AsyncTask`? Самая убедительная причина заключается в том, что `LoaderManager` сохранит жизнь экземпляров `Loader` ваших компонентов вместе со всеми их данными между изменениями конфигурации, такими как повороты. Класс `LoaderManager` отвечает за запуск, остановку и управление жизненным циклом всех экземпляров `Loader`, связанных с компонентом.

Если после изменения конфигурации вы инициализируете экземпляр `Loader`, который уже завершил загрузку данных, он может доставить эти данные немедленно, вместо того чтобы пытаться получить их заново. Такое решение не зависит от того, удерживается фрагмент или нет; это упростит вашу работу, потому что вам не нужно учитывать сложности, связанные с жизненным циклом, которые могут возникнуть из-за удерживаемых фрагментов.

Упражнение. Gson

Десериализация JSON в объекты Java, как было сделано в листинге 25.12, — стандартная задача в разработке приложений на любой платформе. Многие умные люди занимались разработкой библиотек, упрощающих процесс преобразования JSON в объекты Java и наоборот.

К числу таких библиотек относится библиотека Gson (*github.com/google/gson*). Gson автоматически отображает данные JSON на объекты Java; отсюда следует, что вам не придется писать код разбора. По этой причине Gson в настоящее время является нашей любимой библиотекой разбора JSON.

Упростите код разбора JSON в `FlickrFetchr`, включив в свое приложение поддержку Gson.

Упражнение. Страничная навигация

По умолчанию метод `getRecent` возвращает одну страницу со 100 результатами. При помощи дополнительного параметра `page` можно получить вторую, третью и так далее страницу результатов.

Напишите реализацию `RecyclerView.OnScrollListener`, которая обнаруживает достижение конца результатов и заменяет текущую страницу следующей страницей результатов. Чтобы немного усложнить упражнение, организуйте присоединение данных последующих страниц к результатам.

Упражнение. Динамическая настройка количества столбцов

В настоящее время количество столбцов в сетке фиксированно (три). Внесите изменения в свой код, чтобы количество столбцов могло динамически изменяться, а в альбомной ориентации и на больших устройствах отображалось больше столбцов.

В простом решении можно было бы предоставить целочисленный ресурс с квалификаторами для разных ориентаций и/или размеров экранов — по аналогии с тем, как мы предоставляли разные макеты для разных размеров экранов в главе 17. Целочисленные ресурсы должны размещаться в папке(-ах) res/values. За дополнительной информацией обращайтесь к документации разработчика Android.

Предоставление ресурсов с квалификаторами не отличается динамизмом. Чтобы усложнить задачу (и повысить гибкость реализации), вычисляйте и задавайте количество столбцов каждый раз при создании представления фрагмента. Количество столбцов должно вычисляться на основании текущей ширины `RecyclerView` и заранее определенной постоянной ширины столбца.

Возникает только одна проблема: количество столбцов не может вычисляться в `onCreateView()`, потому что размеры `RecyclerView` еще не определены. Вместо этого реализуйте `ViewTreeObserver.OnGlobalLayoutListener` и разместите код вычисления в `onGlobalLayout()`. Добавьте слушателя к `RecyclerView` методом `addOnGlobalLayoutListener()`.

26 Looper, Handler и HandlerThread

После загрузки и разбора JSON от Flickr нашей следующей задачей станет загрузка и вывод изображений. В этой главе вы научитесь использовать классы `Looper`, `Handler` и `HandlerThread` для динамической загрузки и вывода фотографий в PhotoGallery.

Подготовка RecyclerView к выводу изображений

Текущая реализация `PhotoHolder` в `PhotoGalleryFragment` просто предоставляет виджеты `TextView` для вывода объектом `GridLayoutManager` компонента `RecyclerView`. В каждом представлении `TextView` выводится содержимое заголовка `GalleryItem`.

Для вывода фотографий внесите изменения в `PhotoHolder`, чтобы вместо текстовых представлений предоставлялись `ImageView`. В конечном итоге `ImageView` выведет фотографию, загруженную в поле `mUrl` экземпляра `GalleryItem`.

Начнем с создания нового файла макета для элементов фотогалереи в файле gallery_item.xml. Макет будет состоять из единственного виджета `ImageView` (рис. 26.1).

Эти виджеты `ImageView` находятся под управлением экземпляра `GridLayoutManager` компонента `RecyclerView`, что означает, что их ширина будет величиной переменной. При этом высота будет оставаться фиксированной. Чтобы наиболее эффективно использовать пространство виджета `ImageView`, следует задать его свойству

ImageView

```
xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/item_image_view"
android:layout_width="match_parent"
android:layout_height="120dp"
android:layout_gravity="center"
android:scaleType="centerCrop"
```

Рис. 26.1. Макет элемента фотогалереи (res/layout/gallery_item.xml)

scaleType значение centerCrop. С этим значением изображение выравнивается по центру и масштабируется, чтобы меньшая сторона была равна размеру представления, а большая усекалась с обеих сторон.

Также обновите класс PhotoHolder, чтобы вместо TextView он содержал ImageView. Замените bindGalleryItem() методом, назначающим объект Drawable виджету ImageView.

Листинг 26.1. Обновление PhotoHolder (PhotoGalleryFragment.java)

```
private class PhotoHolder extends RecyclerView.ViewHolder {
    private ~~TextView mTitleTextView~~ ImageView mItemImageView;

    public PhotoHolder(View itemView) {
        super(itemView);

        ~~mTitleTextView = (TextView) itemView;~~
        mItemImageView = (ImageView) itemView.findViewById(R.id.item_image_view);
    }

    ~~public void bindGalleryItem(GalleryItem item) {~~
        ~~mTitleTextView.setText(item.toString());~~
    ~~}~~

    public void bindDrawable(Drawable drawable) {
        mItemImageView.setImageDrawable(drawable);
    }
}
```

Ранее конструктор PhotoHolder предполагал, что ему будет передаваться просто объект TextView. Новая версия рассчитывает получить иерархию представлений, которая содержит ImageView с идентификатором ресурса R.id.fragment_photo_gallery_image_view.

Измените метод onCreateViewHolder() в PhotoAdapter, чтобы он заполнял файл gallery_item и передавал его конструктору PhotoHolder.

Листинг 26.2. Обновление метода onCreateViewHolder() класса PhotoAdapter (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {
  ...
  private class PhotoAdapter extends RecyclerView.Adapter<PhotoHolder> {
        ...
        @Override
        public PhotoHolder onCreateViewHolder(ViewGroup viewGroup, int viewType) {
            ~~TextView textView = new TextView(getActivity());~~
            ~~return new PhotoHolder(textView);~~
            LayoutInflater inflater = LayoutInflater.from(getActivity());
            View view = inflater.inflate(R.layout.gallery_item, viewGroup, false);
            return new PhotoHolder(view);
        }
        ...
    }
  ...
}
```

Также нам понадобится временное изображение для каждого виджета `ImageView`, которое будет отображаться до завершения загрузки изображения. Найдите файл bill_up_close.jpg в файле решений и поместите его в каталог res/drawable. (За дополнительной информацией о решениях обращайтесь к разделу «Добавление значка» главы 2.)

Внесите изменения в метод `onBindViewHolder()` класса `PhotoAdapter`, чтобы временное изображение назначалось объектом `Drawable` виджета `ImageView`.

Листинг 26.3. Назначение временного изображения (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {
  ...
  private class PhotoAdapter extends RecyclerView.Adapter<PhotoHolder> {
        ...
        @Override
        public void onBindViewHolder(PhotoHolder photoHolder, int position) {
          GalleryItem galleryItem = mGalleryItems.get(position);
          photoHolder.bindGalleryItem(galleryItem);
          Drawable placeholder = getResources().getDrawable(R.drawable.bill_
                                 up_close);
          photoHolder.bindDrawable(placeholder);
        }
        ...
    }
  ...
}
```

Запустив приложение PhotoGallery, вы увидите набор увеличенных Биллов (рис. 26.2).

Множественные загрузки

В настоящее время сетевая часть PhotoGallery работает следующим образом: `PhotoGalleryFragment` запускает экземпляр `AsyncTask`, который получает JSON от Flickr в фоновом потоке, и разбирает JSON в массив объектов `GalleryItem`. В каждом объекте `GalleryItem` теперь хранится URL, по которому находится миниатюрная версия фотографии.

Следующим шагом должна стать загрузка этих миниатюр. Казалось бы, дополнительный сетевой код можно просто добавить в метод `doInBackground()` класса `FetchItemsTask`. Массив объектов `GalleryItem` содержит 100 URL-адресов для загрузки. Изображения будут загружаться одно за одним, пока у вас не появятся все сто. При выполнении `onPostExecute(…)` они все вместе появятся в `RecyclerView`.

Однако единовременная загрузка всех миниатюр создает две проблемы. Во-первых, она займет довольно много времени, а пользовательский интерфейс не будет обновляться до момента ее завершения. На медленном подключении пользователям придется долго рассматривать стену из Биллов.

Рис. 26.2. Биллы повсюду

Во-вторых, хранение полного набора изображений требует ресурсов. Сотня миниатюр легко уместится в памяти. Но что, если их будет 1000? Что, если вы захотите реализовать бесконечную прокрутку? Со временем свободная память будет исчерпана.

С учетом этих проблем реальные приложения часто загружают изображения только тогда, когда они должны выводиться на экране. Загрузка по мере надобности предъявляет дополнительные требования к `RecyclerView` и его адаптеру. Адаптер инициирует загрузку изображения как часть реализации `onBindViewHolder(…)`.

`AsyncTask` — самый простой способ получения фоновых потоков, но для многократно выполняемых и продолжительных операций этот механизм изначально малопригоден. (О том, почему это так, рассказано в разделе «Для любознательных» в конце этой главы.)

Вместо использования `AsyncTask` мы создадим специализированный фоновый поток. Это самый распространенный способ реализации загрузки по мере надобности.

Взаимодействие с главным потоком

Специализированный поток будет загружать фотографии, но как он будет взаимодействовать с адаптером `RecyclerView` для их отображения, если он не может напрямую обращаться к главному потоку?

Вспомните пример с обувным магазином и двумя продавцами-Флэшами. Фоновый Флэш завершил свой телефонный разговор с поставщиком, и теперь ему нужно сообщить Главному Флэшу о том, что обувь была заказана. Если Главный Флэш занят, Фоновый Флэш не может сделать это немедленно. Ему придется подождать у стойки и перехватить Главного Флэша в свободный момент. Такая схема работает, но не слишком эффективно.

Лучше дать каждому Флэшу по почтовому ящику. Фоновый Флэш пишет сообщение о том, что обувь заказана, и кладет его в ящик Главного Флэша. Главный Флэш делает то же самое, когда он хочет сообщить Фоновому Флэшу о том, что какой-то товар закончился.

Идея почтового ящика чрезвычайно полезна. Возможно, у продавца имеется задача, которая должна быть выполнена скоро, но не прямо сейчас. В таком случае он кладет сообщение в свой почтовый ящик и обрабатывает его в свободное время.

В Android такой «почтовый ящик», используемый потоками, называется *очередью сообщений* (message queue). Поток, работающий с использованием очереди сообщений, называется *циклом сообщений* (message loop); он снова и снова проверяет новые сообщения, которые могли появиться в очереди (рис. 26.3).

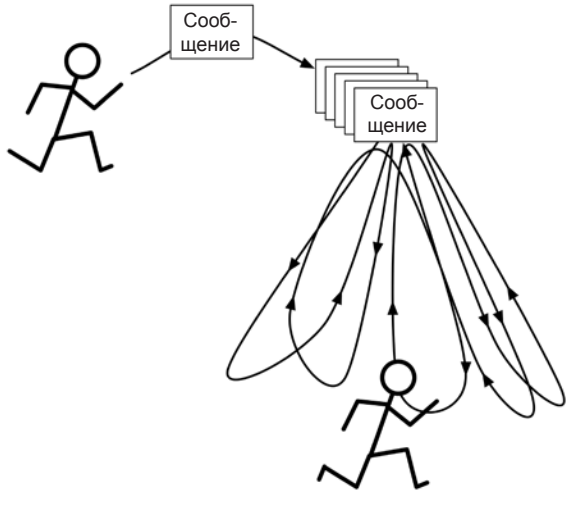


Рис. 26.3. Цикл сообщений

Цикл сообщений состоит из потока и объекта `Looper`, управляющего очередью сообщений потока.

Главный поток представляет собой цикл сообщений, и у него есть управляющий объект, который извлекает сообщения из очереди сообщений и выполняет задачу, описанную в сообщении.

Мы создадим фоновый поток, который тоже использует цикл сообщений. При этом будет использоваться класс `HandlerThread`, который предоставляет готовый объект `Looper`.

Создание фонового потока

Создайте новый класс с именем `ThumbnailDownloader`, расширяющий `HandlerThread`. Определите для него конструктор и заглушку реализации метода с именем `queueThumbnail()` (листинг 26.4).

Листинг 26.4. Исходная версия кода потока (ThumbnailDownloader.java)

```
public class ThumbnailDownloader<T> extends HandlerThread {
    private static final String TAG = "ThumbnailDownloader";

    private boolean mHasQuit = false;

    public ThumbnailDownloader() {
        super(TAG);
    }

    @Override
    public boolean quit() {
        mHasQuit = true;
        return super.quit();
    }

    public void queueThumbnail(T target, String url) {
        Log.i(TAG, "Got a URL: " + url);
    }
}
```

Классу передается один обобщенный аргумент `<T>`. Пользователю `ThumbnailDownloader` понадобится объект для идентификации каждой загрузки и определения элемента пользовательского интерфейса, который должен обновляться после завершения загрузки. Вместо того чтобы ограничивать пользователя одним конкретным типом объекта, мы используем обобщенный параметр и сделаем реализацию более гибкой.

Метод `queueThumbnail()` ожидает получить объект типа `T`, выполняющий функции идентификатора загрузки, и `String` с URL-адресом для загрузки. Этот метод будет вызываться `PhotoAdapter` в его реализации `onBindViewHolder(…)`.

Откройте файл PhotoGalleryFragment.java. Определите в `PhotoGalleryFragment` поле типа `ThumbnailDownloader`. В методе `onCreate(…)` создайте поток и запустите его. Переопределите метод `onDestroy()` для завершения потока.

Листинг 26.5. Создание класса ThumbnailDownloader (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {

    private static final String TAG = "PhotoGalleryFragment";

    private RecyclerView mPhotoRecyclerView;
    private List<GalleryItem> mItems = new ArrayList<>();
```

```
    private ThumbnailDownloader<PhotoHolder> mThumbnailDownloader;
    ...

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRetainInstance(true);
        new FetchItemsTask().execute();

        mThumbnailDownloader = new ThumbnailDownloader<>();
        mThumbnailDownloader.start();
        mThumbnailDownloader.getLooper();
        Log.i(TAG, "Background thread started");
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        ...
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        mThumbnailDownloader.quit();
        Log.i(TAG, "Background thread destroyed");
    }
    ...
}
```

В обобщенном аргументе `ThumbnailDownloader` можно указать любой тип. Однако вспомните, что этот аргумент задает тип объекта, который будет использоваться в качестве идентификатора для загрузки. В данном случае в качестве идентификатора удобно использовать объект `PhotoHolder`, так как он заодно определяет место, куда в конечном итоге поступят загруженные изображения.

Пара примечаний: во-первых, обратите внимание на то, что вызов `getLooper()` следует после вызова `start()` для `ThumbnailDownloader` (вскоре мы рассмотрим объект `Looper` более подробно). Тем самым гарантируется, что внутреннее состояние потока готово для продолжения, чтобы исключить теоретически возможную (хотя и редко встречающуюся) ситуацию гонки (race condition). До вызова `getLooper()` ничто не гарантирует, что метод `onLooperPrepared()` был вызван, поэтому существует вероятность того, что вызов `queueThumbnail(…)` завершится неудачей так как ссылка на `Handler` равна `null`.

Во-вторых, вызов `quit()` завершает поток внутри `onDestroy()`. Это очень важный момент. Если не завершать потоки `HandlerThread`, они никогда не умрут, словно зомби. Или рок-н-ролл.

Наконец, в методе `PhotoAdapter.onBindViewHolder(…)` вызовите метод `queueThumbnail()` потока и передайте ему объект `PhotoHolder`, в котором в конечном итоге будет размещено изображение, и URL-адрес объекта `GalleryItem` для загрузки.

Листинг 26.6. Подключение ThumbnailDownloader (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {
  ...
  private class PhotoAdapter extends RecyclerView.Adapter<PhotoHolder> {
    ...
    @Override
    public void onBindViewHolder(PhotoHolder photoHolder, int position) {
        GalleryItem galleryItem = mGalleryItems.get(position);
        Drawable placeholder = getResources().getDrawable
                               (R.drawable.bill_up_close);
        photoHolder.bindDrawable(placeholder);
        mThumbnailDownloader.queueThumbnail(photoHolder, galleryItem.getUrl());
    }
    ...
  }
    ...
}
```

Запустите приложение PhotoGallery и проверьте данные LogCat. При прокрутке `RecyclerView` в LogCat появляются строки, сообщающие о том, что `ThumbnailDownloader` получает все запросы на загрузку.

Теперь, когда наша реализация `HandlerThread` заработала, следующим шагом становится создание сообщения с информацией, переданной `queueThumbnail()`, и его размещение в очереди сообщений `ThumbnailDownloader`.

Сообщения и обработчики сообщений

Прежде чем создавать *сообщение*, необходимо сначала понять, что оно собой представляет и какие отношения связывают его с *обработчиком сообщения* (message handler).

Строение сообщения

Начнем с сообщений. Сообщения, которые Флэш кладет в почтовый ящик (свой собственный или принадлежащий другому продавцу), содержат не ободряющие записки типа «Ты бегаешь очень быстро», а описания задач, которые необходимо выполнить.

Сообщение является экземпляром класса `Message` и состоит из нескольких полей.

Для нашей реализации важны три поля:

- `what` — определяемое пользователем значение `int`, описывающее сообщение;
- `obj` — заданный пользователем объект, передаваемый с сообщением;
- `target` — приемник, то есть объект `Handler`, который будет обрабатывать сообщение.

Приемником сообщения `Message` является экземпляр `Handler`. Когда вы создаете сообщение, оно автоматически присоединяется к `Handler`. А когда ваше сообщение будет готово к обработке, именно `Handler` становится объектом, отвечающим за эту обработку.

Строение обработчика

Итак, для выполнения реальной работы с сообщениями необходимо иметь экземпляр `Handler`. Объект `Handler` — не просто приемник для обработки сообщений; он также предоставляет интерфейс для создания и отправки сообщений. Взгляните на рис. 26.4.

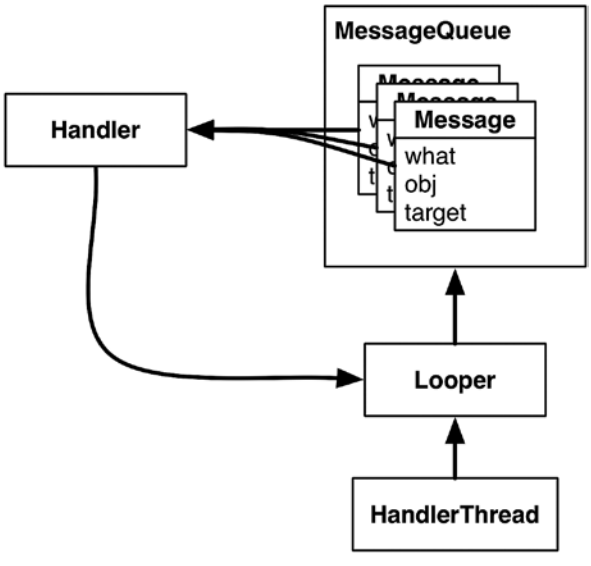


Рис. 26.4. Looper, Handler, HandlerThread и Message

Сообщения `Message` отправляются и потребляются объектом `Looper`, потому что `Looper` является владельцем почтового ящика объектов `Message`. Соответственно, `Handler` всегда содержит ссылку на своего коллегу `Looper`.

`Handler` всегда присоединяется ровно к одному объекту `Looper`, а `Message` присоединяется ровно к одному объекту `Handler`, называемому его *приемником*. Объект `Looper` обслуживает целую очередь сообщений `Message`. Многие сообщения `Message` могут содержать ссылку на один целевой объект `Handler` (рис. 26.4).

К одному объекту `Looper` могут быть присоединены несколько объектов `Handler` (рис. 26.5). Это означает, что сообщения `Message` объекта `Handler` могут сосуществовать с сообщениями другого объекта `Handler`.

Использование обработчиков

Обычно приемные объекты `Handler` сообщений не задаются вручную. Лучше построить сообщение вызовом `Handler.obtainMessage(…)`. Вы передаете методу другие поля сообщения, а он автоматически назначает приемник.

Метод `Handler.obtainMessage(…)` использует общий пул объектов, чтобы избежать создания новых объектов `Message`, поэтому он также работает более эффективно, чем простое создание новых экземпляров.

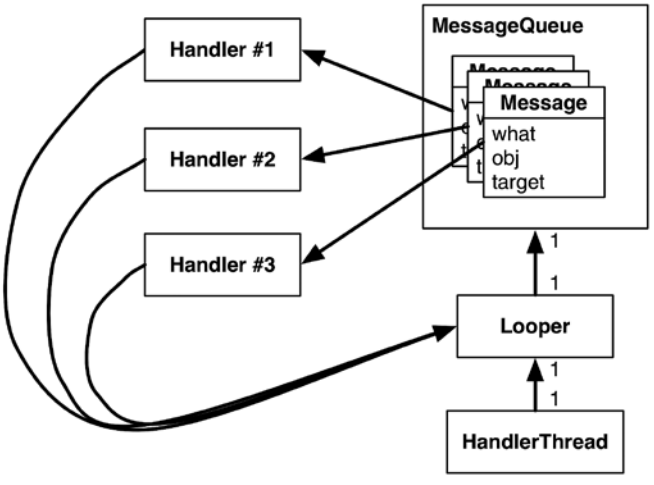


Рис. 26.5. Несколько объектов Handler, один объект Looper

Когда объект `Message` будет получен, вы можете вызвать метод `sendToTarget()`, чтобы отправить сообщение его обработчику. Обработчик помещает сообщение в конец очереди сообщений объекта `Looper`.

Мы собираемся получить сообщение и отправить его приемнику в реализации `queueThumbnail()`. В поле `what` сообщения будет содержаться константа, определяемая под именем `MESSAGE_DOWNLOAD`. В поле `obj` будет содержаться объект типа `T`, предназначенный для идентификации загрузки. В нашем случае это экземпляр `PhotoHolder`, переданный адаптером методу `queueThumbnail()`.

Когда объект `Looper` добирается до конкретного сообщения в очереди, он передает сообщение приемнику сообщения для обработки. Как правило, сообщение обрабатывается в реализации `Handler.handleMessage(…)` приемника.

Схема отношений между объектами изображена на рис. 26.6.

В нашем случае реализация `handleMessage(…)` будет использовать `FlickrFetchr` для загрузки байтов по URL-адресу и их преобразования в растровое изображение.

Добавьте константы и переменные из листинга 26.7.

Листинг 26.7. Добавление констант и переменных (ThumbnailDownloader.java)

```
public class ThumbnailDownloader<T> extends HandlerThread {
    private static final String TAG = "ThumbnailDownloader";
    private static final int MESSAGE_DOWNLOAD = 0;

    private boolean mHasQuit = false;
    private Handler mRequestHandler;
    private ConcurrentMap<T,String> mRequestMap = new ConcurrentHashMap<>();
    ...
}
```

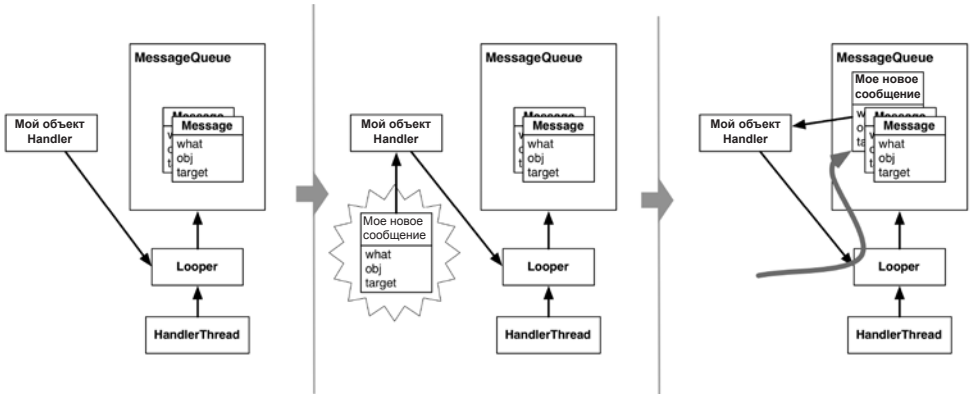


Рис. 26.6. Создание сообщения и его отправка

Значение `MESSAGE_DOWNLOAD` будет использоваться для идентификации сообщений как запросов на загрузку. (`ThumbnailDownloader` присваивает его полю `what` создаваемых сообщений загрузки.)

В переменной `mRequestHandler` будет храниться ссылка на объект `Handler`, отвечающий за постановку в очередь запросов на загрузку в фоновом потоке `ThumbnailDownloader`. Этот объект также будет отвечать за обработку сообщений запросов на загрузку при извлечении их из очереди.

Переменная `mRequestMap` содержит `ConcurrentHashMap` — разновидность `HashMap`, безопасную по отношению к потокам. В данном случае использование объекта-идентификатора типа `T` запроса на загрузку в качестве ключа позволяет хранить и загрузить URL-адрес, связанный с конкретным запросом. (Здесь объектом-идентификатором является `PhotoHolder`, так что по ответу на запрос можно легко вернуться к элементу пользовательского интерфейса, в котором должно находиться загруженное изображение.)

Затем добавьте в `queueThumbnail(…)` код обновления `mRequestMap` и постановки нового сообщения в очередь сообщений фонового потока.

Листинг 26.8. Отправка сообщения (ThumbnailDownloader.java)

```
public class ThumbnailDownloader<T> extends HandlerThread {
    private static final String TAG = "ThumbnailDownloader";
    private static final int MESSAGE_DOWNLOAD = 0;

    private boolean mHasQuit = false;
    private Handler mRequestHandler;
    private ConcurrentMap<T,String> mRequestMap = new ConcurrentHashMap<>();

    public ThumbnailDownloader() {
        super(TAG);
    }

    @Override
    public boolean quit() {
```

```
        mHasQuit = true;
        return super.quit();
    }

    public void queueThumbnail(T target, String url) {
        Log.i(TAG, "Got a URL: " + url);

        if (url == null) {
            mRequestMap.remove(target);
        } else {
            mRequestMap.put(target, url);
            mRequestHandler.obtainMessage(MESSAGE_DOWNLOAD, target)
                    .sendToTarget();
        }
    }
}
```

Сообщение берется непосредственно из `mRequestHandler`, в результате чего поле `target` нового объекта `Message` немедленно заполняется `mRequestHandler`. Это означает, что `mRequestHandler` будет отвечать за обработку сообщения при его извлечении из очереди сообщений. Поле `what` сообщения заполняется значением `MESSAGE_DOWNLOAD`. В поле `obj` заносится значение `T target` (`PhotoHolder` в данном случае), переданное `queueThumbnail(…)`.

Новое сообщение представляет запрос на загрузку заданного `T target` (`PhotoHolder` из `RecyclerView`). Вспомните, что реализация адаптера `RecyclerView` из `PhotoGalleryFragment` вызывает `queueThumbnail(…)` из `onBindViewHolder(…)`, передавая объект `PhotoHolder`, для которого загружается изображение, и URL-адрес загружаемого изображения.

Обратите внимание: в само сообщение URL-адрес не входит. Вместо этого `mRequestMap` обновляется связью между идентификатором запроса (`PhotoHolder`) и URL-адресом запроса. Позднее мы получим URL из `mRequestMap`, чтобы гарантировать, что для заданного экземпляра `PhotoHolder` всегда загружается последний из запрашивавшихся URL-адресов. (Это важно, потому что объекты `ViewHolder` в `RecyclerView` перерабатываются и используются повторно.)

Наконец, инициализируйте `mRequestHandler` и определите, что будет делать объект `Handler`, когда сообщения извлекаются из очереди и передаются ему.

Листинг 26.9. Обработка сообщения (ThumbnailDownloader.java)

```
public class ThumbnailDownloader<T> extends HandlerThread {
    private static final String TAG = "ThumbnailDownloader";
    private static final int MESSAGE_DOWNLOAD = 0;

    private boolean mHasQuit = false;
    private Handler mRequestHandler;
    private ConcurrentMap<T,String> mRequestMap = new ConcurrentHashMap<>();

    public ThumbnailDownloader() {
        super(TAG);
    }
```

```
    @Override
    protected void onLooperPrepared() {
        mRequestHandler = new Handler() {
            @Override
            public void handleMessage(Message msg) {
                if (msg.what == MESSAGE_DOWNLOAD) {
                    T target = (T) msg.obj;
                    Log.i(TAG, "Got a request for URL: " + mRequestMap.
get(target));
                    handleRequest(target);
                }
            }
        };
    }

    @Override
    public boolean quit() {
        mHasQuit = true;
        return super.quit();
    }

    public void queueThumbnail(T target, String url) {
        ...
    }
    private void handleRequest(final T target) {
        try {
            final String url = mRequestMap.get(target);

            if (url == null) {
                return;
            }

            byte[] bitmapBytes = new FlickrFetchr().getUrlBytes(url);
            final Bitmap bitmap = BitmapFactory
                    .decodeByteArray(bitmapBytes, 0, bitmapBytes.length);
            Log.i(TAG, "Bitmap created");

        } catch (IOException ioe) {
            Log.e(TAG, "Error downloading image", ioe);
        }
    }
}
```

Метод `Handler.handleMessage(…)` реализуется в субклассе `Handler` внутри `onLooperPrepared()`. Метод `HandlerThread.onLooperPrepared()` вызывается до того, как `Looper` впервые проверит очередь, поэтому он хорошо подходит для создания реализации `Handler`.

В коде `Handler.handleMessage(…)` мы проверяем тип сообщения, читаем значение `obj` (которое имеет тип `T` и служит идентификатором для запроса) и передаем его `handleRequest(…)`. (Вспомните, что `Handler.handleMessage(…)` будет вызываться, когда сообщение загрузки извлечено из очереди и готово к обработке.)

Вся загрузка осуществляется в методе `handleRequest()`. Мы проверяем существование URL-адреса, после чего передаем его новому экземпляру знакомого класса `FlickrFetchr`. При этом используется метод `FlickrFetchr.getUrlBytes(…)`, который мы так предусмотрительно создали в последней главе.

Наконец, мы используем класс `BitmapFactory` для построения растрового изображения с массивом байтов, возвращенным `getUrlBytes(…)`.

Запустите приложение PhotoGallery и проверьте в данных LogCat ваши подтверждающие команды регистрации.

Разумеется, запрос не будет полностью обработан до момента назначения изображения в объекте `PhotoHolder`, поступившем от `PhotoAdapter`. Однако эта операция относится к пользовательскому интерфейсу, поэтому она должна выполняться в главном потоке.

До настоящего момента мы ограничивались использованием обработчиков и сообщений в одном потоке — помещением сообщений в собственный почтовый ящик `ThumbnailDownloader`. В следующем разделе вы увидите, как `ThumbnailDownloader` использует `Handler` для отправки запросов главному потоку.

Передача Handler

Итак, вы знаете, как спланировать выполнение работы в фоновом потоке из главного потока с использованием значения `mRequestHandler` объекта `ThumbnailDownloader`. Соответствующая схема изображена на рис. 26.7.

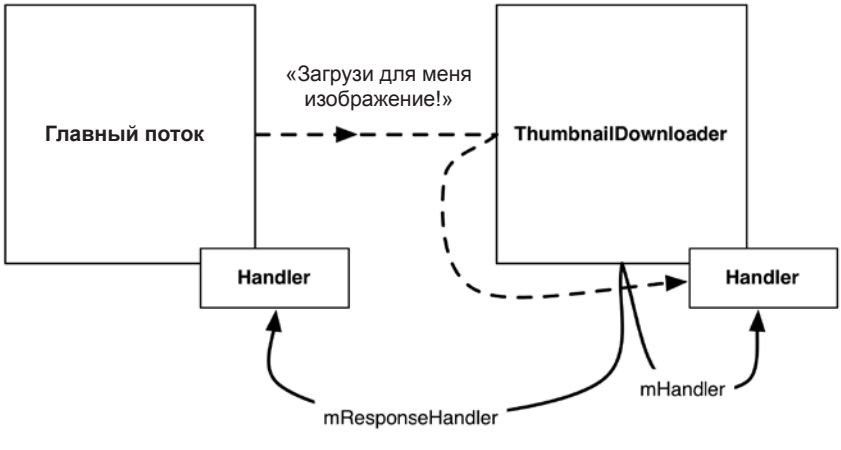


Рис. 26.7. Планирование операций в ThumbnailDownloader из главного потока

Аналогичным образом можно планировать операции в главном потоке из фонового потока с использованием экземпляра `Handler`, присоединенного к главному потоку (рис. 26.8).

Главный поток представляет собой цикл сообщений с обработчиками и `Looper`. При создании экземпляра `Handler` в главном потоке он ассоциируется с экземп-

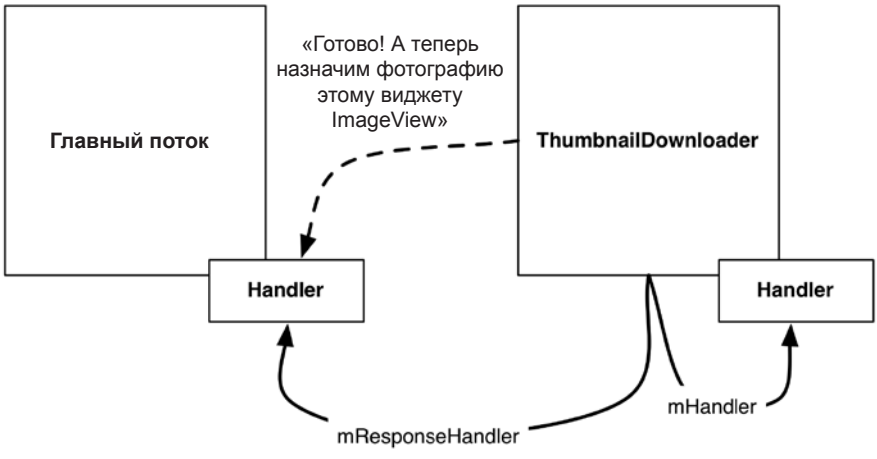


Рис. 26.8. Планирование операций в главном потоке из ThumbnailDownloader

ляром `Looper` главного потока. Затем этот экземпляр `Handler` можно передать другому потоку. Переданный экземпляр `Handler` сохраняет связь с `Looper` потока-создателя. Все сообщения, за которые отвечает `Handler`, будут обрабатываться в очереди главного потока.

В файле ThumbnailDownloader.java добавьте упоминавшуюся выше переменную `mResponseHandler` для хранения экземпляра `Handler`, переданного из главного потока. Затем замените конструктор другим, который получает `Handler` и задает переменную, и добавьте интерфейс слушателя для передачи ответов (загруженных изображений) запрашивающей стороне (главному потоку).

Листинг 26.10. Обработка сообщения (ThumbnailDownloader.java)

```
public class ThumbnailDownloader<T> extends HandlerThread {
    private static final String TAG = "ThumbnailDownloader";
    private static final int MESSAGE_DOWNLOAD = 0;

    private boolean mHasQuit = false;
    private Handler mRequestHandler;
    private ConcurrentMap<T,String> mRequestMap = new ConcurrentHashMap<>();
    private Handler mResponseHandler;
    private ThumbnailDownloadListener<T> mThumbnailDownloadListener;

    public interface ThumbnailDownloadListener<T> {
        void onThumbnailDownloaded(T target, Bitmap thumbnail);
    }

    public void setThumbnailDownloadListener(ThumbnailDownloadListener<T> listener)
{
        mThumbnailDownloadListener = listener;
    }

    public ThumbnailDownloader(Handler responseHandler) {
```

```
        super(TAG);
        mResponseHandler = responseHandler;
    }
    ...
}
```

Метод `onThumbnailDownloaded(…)`, определенный в новом интерфейсе `ThumbnailDownloadListener`, будет вызван через некоторое время, когда изображение было полностью загружено и готово к добавлению в пользовательский интерфейс. Использование слушателя передает ответственность за обработку загруженного изображения другому классу (в данном случае `PhotoGalleryFragment`). Тем самым задача загрузки отделяется от задачи обновления пользовательского интерфейса (связывания изображений с `ImageView`), чтобы класс `ThumbnailDownloader` при необходимости мог использоваться для загрузки данных других разновидностей объектов `View`.

Затем измените класс `PhotoGalleryFragment` так, чтобы он передавал классу `ThumbnailDownloader` объект `Handler`, присоединенный к главному потоку. Также назначьте `ThumbnailDownloadListener` для обработки загруженного изображения после завершения загрузки.

Листинг 26.11. Подключение к обработчику ответа (PhotoGalleryFragment.java)

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setRetainInstance(true);
    new FetchItemsTask().execute();

    Handler responseHandler = new Handler();
    mThumbnailDownloader = new ThumbnailDownloader<>(responseHandler);
    mThumbnailDownloader.setThumbnailDownloadListener(
        new ThumbnailDownloader.ThumbnailDownloadListener<PhotoHolder>() {
            @Override
            public void onThumbnailDownloaded(PhotoHolder photoHolder,
                    Bitmap bitmap) {
                Drawable drawable = new BitmapDrawable(getResources(), bitmap);
                photoHolder.bindDrawable(drawable);
            }
        }
    );
    mThumbnailDownloader.start();
    mThumbnailDownloader.getLooper();
    Log.i(TAG, "Background thread started");
}
```

Вспомните, что по умолчанию `Handler` присоединяется к объекту `Looper` для текущего потока. Поскольку объект `Handler` создан в `onCreate(…)`, он будет присоединен к объекту `Looper` главного потока.

Теперь `ThumbnailDownloader` имеет доступ к экземпляру `Handler`, связанному с экземпляром `Looper` главного потока, через поле `mResponseHandler`. Кроме того,

он приказывает `ThumbnailDownloadListener` выполнять операции пользовательского интерфейса с возвращаемыми объектами `Bitmap`. А конкретно, реализация `onThumbnailDownloaded` назначает объекту `PhotoHolder`, от которого поступил исходный запрос, загруженный объект `Bitmap`.

Аналогичным образом можно отправить главному потоку нестандартный объект `Messsage`, запрашивающий добавление изображения в пользовательский интерфейс, по аналогии с тем, как мы ставили в очередь запрос к фоновому потоку на загрузку изображения. Для этого потребуется другой субкласс `Handler` с переопределением `handleMessage(…)`.

Однако вместо этого мы используем другой удобный метод `Handler` — `post(Runnable)`.

`Handler.post(Runnable)` — вспомогательный метод для отправки сообщений следующего вида:

```
Runnable myRunnable = new Runnable() {
    @Override
    public void run() {
        /* Ваш код */
    }
};
Message m = mHandler.obtainMessage();
m.callback = myRunnable;
```

Если у `Message` задано поле `callback`, то вместо передачи приемнику `Handler` при извлечении из очереди сообщений выполняется объект `Runnable` из поля `callback`.

Включите в `ThumbnailDownloader.handleRequest()` следующий код.

Листинг 26.12. Загрузка и вывод (ThumbnailDownloader.java)

```
public class ThumbnailDownloader<T> extends HandlerThread {
    ...
    private Handler mResponseHandler;
    private ThumbnailDownloadListener<T> mThumbnailDownloadListener;
    ...
    private void handleRequest(final T target) {
        try {
            final String url = mRequestMap.get(target);

           if (url == null) {
                return;
            }

            byte[] bitmapBytes = new FlickrFetchr().getUrlBytes(url);
            final Bitmap bitmap = BitmapFactory
                    .decodeByteArray(bitmapBytes, 0, bitmapBytes.length);
            Log.i(TAG, "Bitmap created");

            mResponseHandler.post(new Runnable() {
                public void run() {
```

```
                    if (mRequestMap.get(target) != url ||
                            mHasQuit) {
                        return;
                    }

                    mRequestMap.remove(target);
                    mThumbnailDownloadListener.onThumbnailDownloaded(target,
                                                bitmap);
                }
            });

        } catch (IOException ioe) {
            Log.e(TAG, "Error downloading image", ioe);
        }
    }
}
```

А поскольку `mResponseHandler` связывается с `Looper` главного потока, весь код `run()` будет выполнен в главном потоке.

Что делает этот код? Сначала он проверяет `requestMap`. Такая проверка необходима, потому что `RecyclerView` заново использует свои представления. К тому времени, когда `ThumbnailDownloader` завершит загрузку `Bitmap`, может оказаться, что виджет `RecyclerView` уже переработал `ImageView` и запросил для него изображение с другого URL-адреса. Эта проверка гарантирует, что каждый объект `PhotoHolder` получит правильное изображение, даже если за прошедшее время был сделан другой запрос.

Затем проверяется `mHasQuit`. Если выполнение `ThumbnailDownloader` уже завершилось, выполнение каких-либо обратных вызовов небезопасно.

Наконец, мы удаляем из `requestMap` связь «PhotoHolder—URL» и назначаем изображение для `PhotoHolder`.

Прежде чем запускать приложение, чтобы увидеть завоеванные тяжелым трудом изображения, необходимо принять во внимание одну последнюю опасность. Если пользователь повернет экран, `ThumbnailDownloader` может оказаться связанным с недействительными экземплярами `PhotoHolder`. Нажатие на них грозит всевозможными неприятностями.

Напишите метод `clearQueue()` для удаления всех запросов из очереди.

Листинг 26.13. Добавление метода очистки очереди (ThumbnailDownloader.java)

```
public class ThumbnailDownloader<T> extends HandlerThread {
    ...
    public void queueThumbnail(T target, String url) {
        ...
    }

    public void clearQueue() {
        mRequestHandler.removeMessages(MESSAGE_DOWNLOAD);
        mRequestMap.clear();
    }
```

```
    private void handleRequest(final T target) {
        ...
    }
}
```

Затем очистите загрузчик в `PhotoGalleryFragment` при уничтожении представления.

Листинг 26.14. Вызов метода очистки очереди (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {
  ...
  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
      ...
  }

  @Override
  public void onDestroyView() {
      super.onDestroyView();
      mThumbnailDownloader.clearQueue();
  }

  @Override
  public void onDestroy() {
      ...
  }
  ...
}
```

На этом наша работа в этой главе подходит к концу. Запустите приложение PhotoGallery. Прокрутите список и понаблюдайте за тем, как происходит динамическая загрузка изображений.

Приложение PhotoGallery выполняет свою основную функцию — вывод изображений из Flickr. В нескольких следующих главах мы дополним его новыми функциями: поиском фотографий и открытием страницы Flickr каждой фотографии в веб-представлении.

Для любознательных: AsyncTask и потоки

Теперь вы понимаете, как работают классы `Handler` и `Looper`, и класс `AsyncTask` уже кажется не таким волшебным. При этом он требует меньшего объема работы по сравнению с тем, что мы сделали сейчас. Так почему бы не использовать `AsyncTask` вместо `HandlerThread`?

По нескольким причинам. Самая принципиальная заключается в том, что класс `AsyncTask` проектировался не для этого. Он предназначен для кратковременной работы, которая не повторяется слишком часто. `AsyncTask` отлично подходит для

таких ситуаций, как в нашем коде из предыдущей главы. Но если вы создаете множество `AsyncTask` или они выполняются в течение долгого времени, вероятно, вы неправильно выбрали класс.

Есть и другая, более убедительная техническая причина: в Android 3.2 реализация `AsyncTask` была серьезно изменена. Начиная с Android 3.2 `AsyncTask` не создает поток для каждого экземпляра `AsyncTask`. Вместо этого он создает объект `Executor` для выполнения фоновой работы всех экземпляров `AsyncTask` в одном фоновом потоке. Это означает, что экземпляры `AsyncTask` будут выполняться друг за другом, а затянувшаяся операция `AsyncTask` не позволит другим экземплярам `AsyncTask` получить процессорное время.

Организовать безопасное параллельное выполнение `AsyncTask` с использованием пула потоков возможно, но мы не рекомендуем так поступать. Если вы рассматриваете такое решение, обычно лучше самостоятельно организовать многопоточное выполнение, используя объекты `Handler` для взаимодействия с главным потоком, когда возникнет такая необходимость.

Для любознательных: решение задачи загрузки изображений

Эта книга призвана научить вас пользоваться инструментами из стандартной библиотеки Android. Но если вы не боитесь пользоваться сторонними библиотеками, то знайте, что они способны сэкономить уйму времени в различных ситуациях, в том числе и при загрузке изображений, реализованной в PhotoGallery.

Откровенно говоря, решение, представленное в этой главе, далеко не идеально. Когда вам приходится решать проблемы кэширования, преобразований и повышения быстродействия, естественно спросить себя: а не занимался ли кто-нибудь решением этой задачи до вас? Ответ: конечно, занимался. Существует несколько готовых библиотек, решающих задачу загрузки изображений. В своих коммерческих приложениях мы предпочитаем использовать для загрузки изображений библиотеку Picasso (*square.github.io/picasso/*).

С Picasso все, что мы делали в этой главе, делается в одной строке:

```
private class PhotoHolder extends RecyclerView.ViewHolder {
       ...
        public void bindGalleryItem(GalleryItem galleryItem) {
            Picasso.with(getActivity())
                    .load(galleryItem.getUrl())
                    .placeholder(R.drawable.bill_up_close)
                    .into(mItemImageView);
        }
        ...
}
```

Динамичный интерфейс требует задания контекста конструкцией `with(Context)`. URL-адрес загружаемого изображения задается конструкцией `load(String)`,

а объект `ImageView` для загрузки результатов — конструкцией `into(ImageView)`. Также поддерживается много других настраиваемых параметров: например, назначение изображения, которое должно выводиться до полной загрузки запрошенного изображения (`placeholder(int)` или `placeholder(drawable)`).

В `PhotoAdapter.onBindViewHolder(…)` существующий код заменяется сквозным вызовом нового метода `bindGalleryItem(…)`.

Picasso выполняет всю работу `ThumbnailDownloader` (вместе с обратным вызовом `ThumbnailDownloader.ThumbnailDownloadListener<T>`) и всю работу `FlickrFetchr`, связанную с графикой. Это означает, что при использовании Picasso класс `ThumbnailDownloader` можно удалить (хотя класс `FlickrFetchr` еще понадобится для загрузки данных JSON). Кроме упрощения кода, Picasso поддерживает такие дополнительные возможности, как преобразования изображений и организация кэширования с минимальными усилиями с вашей стороны.

Библиотека Picasso добавляется в проект как зависимость в окне структуры проекта, как это делалось для других зависимостей (например, `RecyclerView`).

К недостаткам Picasso можно отнести намеренное ограничение функциональности для сокращения ее размера. В результате Picasso не может загружать и отображать анимированные изображения. Если у вас возникнет такая необходимость, проверьте библиотеку Google Glide или библиотеку Facebook Fresco. Из этих двух библиотек Glide занимает меньше памяти, а Fresco обладает лучшим быстродействием.

Для любознательных: StrictMode

Есть некоторые вещи, которые просто не следует делать в приложениях Android, — ошибки, которые напрямую приводят к сбоям и дефектам безопасности. Например, выполнение сетевого запроса в главном потоке при плохом состоянии сети с большой вероятностью приведет к ошибке ANR.

Вместо того, чтобы спокойно разрешить выполнение сетевого запроса в главном потоке приложения, Android выдает исключение `NetworkOnMainThread` и сохраняет сообщение в журнале. Это обусловлено действием режима StrictMode: он замечает вашу ошибку и любезно сообщает вам о ней. Режим StrictMode был создан для того, чтобы помочь разработчику в обнаружении таких и многих других ошибок и дефектов безопасности в коде.

Сетевые операции в главном потоке запрещаются без какой-либо дополнительной конфигурации. Также StrictMode поможет обнаруживать другие ошибки, способные ухудшить быстродействие приложения. Чтобы включить все рекомендованные политики StrictMode, вызовите метод `StrictMode.enableDefaults()` (*developer.android.com/reference/android/os/StrictMode.html#enableDefaults()*).

После вызова `StrictMode.enableDefaults()` в Logcat будет выводиться информация о следующих нарушениях:

- сетевые операции в главном потоке;
- операции чтения и записи на диск в главном потоке;

- продолжающееся существование активностей за пределами их естественного жизненного цикла (т. наз. «утечка активностей»);
- незакрытые курсоры баз данных SQLite;
- передача незашифрованного текста в сетевом трафике без защиты SSL/TLS.

Классы `ThreadPolicy.Builder` и `VmPolicy.Builder` предоставляют средства для расширенного управления тем, что должно происходить при нарушении политик: выдача исключения, появление диалогового окна или просто сохранение в журнале информации, уведомляющей вас о происходящем.

Упражнение. Предварительная загрузка и кэширование

Пользователи понимают, что не все происходит мгновенно (или по крайней мере большинство пользователей). Но, несмотря на это, программисты стремятся к совершенству.

Для достижения моментального отклика в большинстве реальных приложений приведенный код расширяется в двух направлениях: добавление кэширования и предварительная загрузка изображений.

Кэш в нашем приложении представляет собой место для хранения определенного количества объектов `Bitmap`, чтобы они оставались в памяти даже после завершения использования. Объем кэша ограничен, поэтому вам понадобится стратегия выбора сохраняемых объектов при исчерпании свободного места. Многие кэши используют стратегию LRU (Least Recently Used): при нехватке свободного места из кэша удаляется элемент, который дольше всего не использовался.

Библиотека поддержки Android содержит класс с именем `LruCache`, реализующий стратегию LRU. В первом упражнении используйте `LruCache` для добавления простейшего кэширования `ThumbnailDownloader`. Каждый раз, когда для URL-адреса загружается объект `Bitmap`, вы помещаете его в кэш. Затем, когда требуется загрузить новое изображение, вы сначала проверяете содержимое кэша и смотрите, нет ли его в кэше.

После того как в программе будет создан кэш, он может использоваться для предварительной загрузки, то есть загрузки данных в кэш еще до того, как они фактически потребуются программе. Тем самым предотвращается задержка для загрузки объектов `Bitmap` до их вывода.

Качественно реализовать предварительную загрузку непросто, но она существенно меняет восприятие приложения пользователем. Во втором, более сложном упражнении для каждого выводимого элемента `GalleryItem` выполните предварительную загрузку 10 предшествующих и 10 следующих элементов `GalleryItem`.

27 Поиск

Следующим шагом в работе над приложением PhotoGallery станет поиск фотографий на Flickr. В этой главе вы узнаете, как правильно интегрировать поиск в приложение с использованием виджета `SearchView`. `SearchView` является классом *представления действия* (action view) — представления, которое может быть встроено прямо в панель инструментов.

Пользователь нажимает на `SearchView`, вводит запрос и отправляет его. Flickr проводит поиск введенной строки с использованием поискового API и заполняет `RecyclerView` полученными результатами (рис. 27.1). Отправленная строка запроса сохраняется в файловой системе. Это означает, что последний запрос пользователя «переживет» перезапуск приложения и даже устройства.

Рис. 27.1. Поиск в приложении

Поиск в Flickr

Начнем с того, что нужно сделать на стороне Flickr. Для выполнения поиска в Flickr следует вызвать метод `flickr.photos.search`. Вот как выглядит запрос GET для поиска текста «cat»:

```
https://api.flickr.com/services/rest/?method=flickr.photos.search
&api_key=xxx&format=json&nojsoncallback=1&text=cat
```

В запросе выбирается метод `flickr.photos.search`. Новый параметр `text` определяет условия поиска («cat» в данном случае).

Хотя URL-адрес поискового запроса отличается от того, который мы использовали для запроса информации о последних фотографиях, формат возвращаемой разметки JSON остается прежним. И это хорошо, потому что вы сможете использовать уже написанный код разбора JSON независимо от того, проводите вы поиск или получаете последние фотографии.

Начнем с переработки старого кода `FlickrFetchr`, чтобы использовать один код разбора в обоих сценариях. Добавьте константы для частей URL-адреса, как показано в листинге 27.1. Скопируйте код построения URI из `fetchItems` и вставьте его как значение `ENDPOINT`. Будьте внимательны и включите только выделенные части. Константа `ENDPOINT` не должна содержать параметр запроса метода, а команда построения не должна быть преобразована в строку вызовом `toString()`.

Листинг 27.1. Добавление констант (FlickrFetcher.java)

```
public class FlickrFetchr {

    private static final String TAG = "FlickrFetchr";

    private static final String API_KEY = "ВашКлючApi";
    private static final String FETCH_RECENTS_METHOD = "flickr.photos.getRecent";
    private static final String SEARCH_METHOD = "flickr.photos.search";
    private static final Uri ENDPOINT = Uri
            .parse("https://api.flickr.com/services/rest/")
            .buildUpon()
            .appendQueryParameter("api_key", API_KEY)
            .appendQueryParameter("format", "json")
            .appendQueryParameter("nojsoncallback", "1")
            .appendQueryParameter("extras", "url_s")
            .build();
    ...
    public List<GalleryItem> fetchItems() {

        List<GalleryItem> items = new ArrayList<>();

        try {
            String url = Uri.parse("https://api.flickr.com/services/rest/")
                    .buildUpon()
                    .appendQueryParameter("method", "flickr.photos.getRecent")
```

```
                    .appendQueryParameter("api_key", API_KEY)
                    .appendQueryParameter("format", "json")
                    .appendQueryParameter("nojsoncallback", "1")
                    .appendQueryParameter("extras", "url_s")
                    .build().toString();
            String jsonString = getUrlString(url);
            ...
        } catch (IOException ioe) {
            Log.e(TAG, "Failed to fetch items", ioe);
        } catch (JSONException je) {
            Log.e(TAG, "Failed to parse JSON", je);
        }

        return items;
    }
    ...
}
```

(Эти изменения приводят к ошибке в `fetchItems()`. Пока не обращайте внимания на эту ошибку, все равно метод `fetchItems()` скоро будет удален.)

Переименуйте `fetchItems()` в `downloadGalleryItems(String url)`. Переименование отражает новый, более общий характер метода. Кроме того, метод не должен оставаться открытым — замените его уровень доступа на `private`.

Листинг 27.2. Переработка кода Flickr (FlickrFetcher.java)

```
public class FlickrFetchr {
    ...
    public List<GalleryItem> fetchItems() {
    private List<GalleryItem> downloadGalleryItems(String url) {
        List<GalleryItem> items = new ArrayList<>();

        try {
            String jsonString = getUrlString(url);
            Log.i(TAG, "Received JSON: " + jsonString);
            JSONObject jsonBody = new JSONObject(jsonString);
            parseItems(items, jsonBody);
        } catch (IOException ioe) {
            Log.e(TAG, "Failed to fetch items", ioe);
        } catch (JSONException je) {
            Log.e(TAG, "Failed to parse JSON", je);
        }

        return items;
    }
    ...
}
```

Новый метод `downloadGalleryItems(String)` получает URL-адрес, поэтому строить URL внутри уже не нужно. Вместо этого добавьте новый метод для построения URL-адреса по значениям метода и запроса.

Листинг 27.3. Вспомогательный метод для построения URL (FlickrFetcher.java)

```
public class FlickrFetchr {
    ...
    private List<GalleryItem> downloadGalleryItems(String url) {
        ...
    }

   private String buildUrl(String method, String query) {
        Uri.Builder uriBuilder = ENDPOINT.buildUpon()
                .appendQueryParameter("method", method);

        if (method.equals(SEARCH_METHOD)) {
            uriBuilder.appendQueryParameter("text", query);
        }

        return uriBuilder.build().toString();
    }

    private void parseItems(List<GalleryItem> items, JSONObject jsonBody)
            throws IOException, JSONException {
        ...
    }
}
```

Метод `buildUrl(…)` присоединяет необходимые параметры подобно тому, как когда-то делал удаленный метод `fetchItems()`, но значение параметра метода подставляется динамически. Кроме того, метод присоединяет значение параметра `text` только в том случае, если параметру `method` задано значение `search`.

Теперь добавьте методы, инициирующие загрузку: они строят URL и вызывают `downloadGalleryItems(String)`.

Листинг 27.4. Добавление методов для получения последних фотографий и поиска (FlickrFetchr.java)

```
public class FlickrFetchr {
    ...
    public String getUrlString(String urlSpec) throws IOException {
        return new String(getUrlBytes(urlSpec));
    }

    public List<GalleryItem> fetchRecentPhotos() {
        String url = buildUrl(FETCH_RECENTS_METHOD, null);
        return downloadGalleryItems(url);
    }

    public List<GalleryItem> searchPhotos(String query) {
        String url = buildUrl(SEARCH_METHOD, query);
        return downloadGalleryItems(url);
    }

    private List<GalleryItem> downloadGalleryItems(String url) {
        List<GalleryItem> items = new ArrayList<>();
        ...
```

```
        return items;
    }
    ...
}
```

Класс `FlickrFetchr` теперь способен выполнять как поиск, так и получение последних фотографий. Методы `fetchRecentPhotos()` и `searchPhotos(String)` образуют открытый интерфейс для получения списка объектов `GalleryItem` от веб-службы Flickr.

Код фрагмента также следует обновить в соответствии с рефакторингом, проведенным в `FlickrFetchr`.

Откройте код `PhotoGalleryFragment` и внесите изменения в `FetchItemsTask`.

Листинг 27.5. Код с фиксированным запросом поиска (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {
    ...
    private class FetchItemsTask extends AsyncTask<Void,Void,List<GalleryItem>> {
        @Override
        protected List<GalleryItem> doInBackground(Void... params) {

            return new FlickrFetchr().fetchItems();
            String query = "robot"; // Для тестирования

            if (query == null) {
                return new FlickrFetchr().fetchRecentPhotos();
            } else {
                return new FlickrFetchr().searchPhotos(query);
            }
        }

        @Override
        protected void onPostExecute(List<GalleryItem> items) {
            mItems = items;
            setupAdapter();
        }
    }
}
```

Если поисковый запрос отличен от `null` (а теперь это условие выполняется всегда), то `FetchItemsTask` получает результаты поиска. В противном случае `FetchItemsTask` по умолчанию загружает последние фотографии, как это делалось ранее.

Использование фиксированного запроса позволит протестировать новый код поиска, хотя мы еще и не предусмотрели средств для ввода запроса в пользовательском интерфейсе.

Запустите PhotoGallery и проверьте результаты. Если повезет, вы увидите пару классных роботов (рис. 27.2).

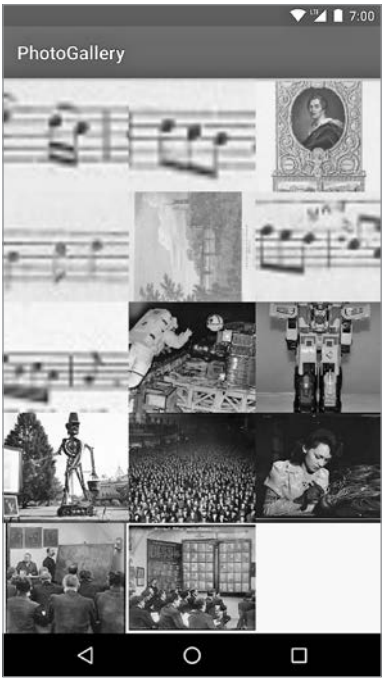


Рис. 27.2. Результаты поиска с фиксированным запросом

Использование SearchView

Итак, мы обеспечили поддержку поиска в `FlickrFetchr`; теперь необходимо дать пользователю средства для ввода запроса и запуска поиска. Для этого мы добавим виджет `SearchView`.

`SearchView` является *представлением действия* (action view) — представлением, которое может быть размещено на панели инструментов. `SearchView` позволяет вынести весь поисковый интерфейс приложения на панель инструментов.

Для начала убедитесь в том, что в верхней части приложения отображается панель инструментов (с названием приложения). Если ее там нет, выполните действия по добавлению панели инструментов, описанные в главе 13.

Создайте новый файл меню в формате XML для `PhotoGalleryFragment` с именем res/menu/fragment_photo_gallery.xml. В этом файле будут определяться элементы, которые должны располагаться на панели инструментов.

Листинг 27.6. Добавление файла XML с меню (res/menu/fragment_photo_gallery.xml)

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">

    <item android:id="@+id/menu_item_search"
          android:title="@string/search"
```

```
            app:actionViewClass="android.support.v7.widget.SearchView"
            app:showAsAction="ifRoom" />

    <item android:id="@+id/menu_item_clear"
            android:title="@string/clear_search"
            app:showAsAction="never" />
</menu>
```

Вы получите пару сообщений об ошибках в XML, в которых будет сказано, что строки, используемые в атрибутах `android:title`, еще не определены. Пока не обращайте на них внимания; вскоре мы решим эту проблему.

Первое определение элемента в листинге 27.6 приказывает вывести на панели инструментов виджет `SearchView`, для чего атрибуту `app:actionViewClass` присваивается значение `android.support.v7.widget.SearchView`. (Обратите внимание на использование пространства имен `app` для атрибутов `showAsAction` и `actionViewClass`. Если вы забыли, для чего оно используется, вернитесь к главе 13.)

Виджет `SearchView` (`android.widget.SearchView`) впервые появился в API 11 (Honeycomb 3.0). Позднее он был включен в библиотеку поддержки (`android.support.v7.widget.SearchView`). Какую версию `SearchView` следует использовать? Вы уже видели наш ответ в только что введенном коде: версию из библиотеки поддержки. На первый взгляд это выглядит странно, так как для приложения установлена минимальная версия SDK 19.

Мы рекомендуем использовать библиотеку поддержки по тем же причинам, которые были описаны в главе 7. Новые возможности, добавляемые в каждой версии Android, часто включаются в библиотеку поддержки (как, например, темы). С выпуском API 21 (Lollipop 5.0) во встроенной реализации `SearchView` появилось много параметров для настройки внешнего вида `SearchView`. Чтобы получить доступ к ним в более ранних версиях Android (до API 7), необходимо использовать версию `SearchView` из библиотеки поддержки.

Второй элемент в листинге 27.6 добавляет действие Clear Search. Это действие всегда будет отображаться в дополнительном меню, потому что атрибуту `app:showAsAction` присвоено значение `never`. Позднее мы настроим это действие так, чтобы при нажатии хранимый запрос пользователя стирался с диска, но пока о нем можно забыть.

Пришло время разобраться с ошибками в разметке меню. Откройте файл strings.xml и добавьте недостающие строки:

Листинг 27.7. Добавление строк для поиска (res/values/strings.xml)

```
<resources>
        ...
        <string name="search">Search</string>
        <string name="clear_search">Clear Search</string>

</resources>
```

Наконец, откройте `PhotoGalleryFragment`. Добавьте в `onCreate(…)` вызов `setHasOptionsMenu(true)`, чтобы зарегистрировать фрагмент для получения об-

ратных вызовов меню. Переопределите `onCreateOptionsMenu(…)` и заполните созданный файл XML с определением меню. Элементы, перечисленные в разметке меню, добавляются на панель инструментов.

Листинг 27.8. Переопределение onCreateOptionsMenu(…) (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {
    ...
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRetainInstance(true);
        setHasOptionsMenu(true);
        new FetchItemsTask().execute();
        ...
    }
    ...
    @Override
    public void onDestroy() {
        ...
    }

    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater menuInflater) {
        super.onCreateOptionsMenu(menu, menuInflater);
        menuInflater.inflate(R.menu.fragment_photo_gallery, menu);
    }

    private void setupAdapter() {
        ...
    }
    ...
}
```

Запустите приложение PhotoGallery и посмотрите, как выглядит `SearchView`. При нажатии на значке Search открывается представление с текстовым полем для ввода запроса (рис. 27.3).

Когда панель `SearchView` открыта, справа появляется значок X. Однократное нажатие стирает введенный текст, а повторное нажатие сворачивает SearchView в значок.

Если вы попытаетесь отправить запрос, ничего не произойдет. Для беспокойства нет причин — сейчас мы научим `SearchView` делать что-то полезное.

Реакция SearchView на взаимодействия с пользователем

Когда пользователь отправляет запрос, приложение должно произвести поиск в веб-службе Flickr и обновить содержимое экрана полученными результатами. К счастью, интерфейс `SearchView.OnQueryTextListener` предоставляет возможность получения обратных вызовов при отправке запроса.

Рис. 27.3. SearchView в свернутом и развернутом состоянии

Обновите метод `onCreateOptionsMenu(…)` и добавьте в `SearchView` реализацию `SearchView.OnQueryTextListener`.

Листинг 27.9. Регистрация событий SearchView.OnQueryTextListener (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {
    ...
    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater menuInflater) {
        super.onCreateOptionsMenu(menu, menuInflater);
        menuInflater.inflate(R.menu.fragment_photo_gallery, menu);

        MenuItem searchItem = menu.findItem(R.id.menu_item_search);
        final SearchView searchView = (SearchView) searchItem.getActionView();

        searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {
            @Override
            public boolean onQueryTextSubmit(String s) {
                Log.d(TAG, "QueryTextSubmit: " + s);
                updateItems();
                return true;
```

```
            }

            @Override
            public boolean onQueryTextChange(String s) {
                Log.d(TAG, "QueryTextChange: " + s);
                return false;
            }
        });
    }

    private void updateItems() {
        new FetchItemsTask().execute();
    }
    ...
}
```

В коде `onCreateOptionsMenu(…)` мы получаем объект `MenuItem`, представляющий поле поиска, и сохраняем его в `searchItem`. Затем из `searchItem` извлекается объект `SearchView` методом `getActionView()`.

(Примечание: метод `MenuItem.getActionView()` был добавлен в API 11. В данном случае это нормально, так как минимальный уровень SDK, выбранный для нашего приложения, равен API 19. Но если вам потребуется создать приложение с максимальной обратной совместимостью, обеспечиваемой библиотекой поддержки, придется поискать другой способ получения доступа к объекту `SearchView`.)

Располагая ссылкой на `SearchView`, вы можете назначить слушателя `SearchView.OnQueryTextListener` при помощи метода `setOnQueryTextListener(…)`. В реализации `SearchView.OnQueryTextListener` необходимо переопределить два метода: `onQueryTextSubmit(String)` и `onQueryTextChange(String)`.

Обратный вызов `onQueryTextChange(String)` выполняется при каждом изменении текста в текстовом поле `SearchView`. Это означает, что метод будет вызываться каждый раз, когда изменяется хотя бы один символ. В нашем приложении этот метод не будет делать ничего, кроме регистрации входной строки в журнале.

Обратный вызов `onQueryTextSubmit(String)` выполняется при отправке запроса пользователем. Отправленный запрос передается во входном параметре. Возвращение `true` сообщает системе, что поисковый запрос был обработан. В этом методе мы будем запускать `FetchItemsTask` для получения новых результатов. (В текущем состоянии `FetchItemsTask` все еще использует фиксированный запрос. Вскоре мы переработаем код `FetchItemsTask` так, чтобы в нем использовался запрос, отправленный пользователем, — если он есть.)

Метод `updateItems()` пока не делает ничего особенно полезного. Как будет показано позднее, в коде приложения есть несколько мест, в которых потребуется выполнять `FetchItemsTask`. Метод `updateItems()` представляет собой обертку для выполнения этой операции.

И наконец, замените строку, которая создает и выполняет `FetchItemsTask`, вызовом `updateItems()` в методе `onCreate(…)`.

Листинг 27.10. Замена в коде onCreate(...) (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {
    ...
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRetainInstance(true);
        setHasOptionsMenu(true);
        new FetchItemsTask().execute();
        updateItems();
        ...
        Log.i(TAG, "Background thread started");
    }
    ...
}
```

Запустите приложение и отправьте запрос. Результаты поиска по-прежнему соответствуют фиксированному запросу из листинга 27.5, но изображения должны перезагрузиться. Также обратите внимание на вывод в журнале, свидетельствующий о том, что методы обратного вызова `SearchView.OnQueryTextListener` были выполнены.

(Примечание: если для отправки поискового запроса в эмуляторе используется физическая клавиатура (например, на ноутбуке), вы увидите, что запрос выполняется два раза. Все выглядит так, словно изображения начинают загружаться, а потом загружаются заново. Такое поведение связано с незначительной ошибкой в `SearchView`. Не обращайте на него внимания — это побочный эффект использования эмулятора, который не проявляется при запуске приложения на реальном устройстве Android.)

Простое сохранение с использованием механизма общих настроек

Последняя функция, которую необходимо добавить в приложение, — использование фактического текста, введенного в `SearchView`, при отправке запроса.

В нашем приложении в любой момент времени существует только один активный запрос. Он должен сохраняться (запоминаться приложением) между перезапусками — даже если пользователь отключит устройство. Для этого строка запроса будет записываться в хранилище *общих настроек*. Каждый раз, когда пользователь отправляет запрос, приложение первым делом записывает запрос в общие настройки (заменяя запрос, который хранился до этого). При выполнении поиска в Flickr строка запроса читается из общих настроек, а ее значение определяет параметр `text`.

Хранилище общих настроек (shared preferences) представляет собой файлы в файловой системе, для чтения и редактирования которых используется класс `SharedPreferences`. Экземпляр `SharedPreferences` работает как хранилище пар

«ключ-значение», имеющее много общего с `Bundle`, но с возможностью долгосрочного хранения. Ключами являются строки, а значениями — атомарные типы данных. При ближайшем рассмотрении выясняется, что файлы содержат простую разметку XML, но благодаря классу `SharedPreferences` на эту подробность реализации можно не обращать внимания. Файлы общих настроек хранятся в песочнице вашего приложения, поэтому в них не следует хранить конфиденциальную информацию (например, пароли).

Для получения конкретного экземпляра `SharedPreferences` можно воспользоваться методом `Context.getSharedPreferences(String,int)`. Однако на практике часто важен не конкретный экземпляр, а его совместное использование в пределах всего приложения. В таких ситуациях лучше использовать метод `PreferenceManager.getDefaultSharedPreferences(Context)`, который возвращает экземпляр с именем по умолчанию и закрытыми (private) разрешениями (чтобы настройки были доступны только в границах вашего приложения).

Добавьте новый класс с именем `QueryPreferences`, который будет предоставлять удобный интерфейс для чтения/записи запроса в хранилище общих настроек.

Листинг 27.11. Добавление класса для работы с хранимым запросом (QueryPreferences.java)

```
public class QueryPreferences {
    private static final String PREF_SEARCH_QUERY = "searchQuery";

    public static String getStoredQuery(Context context) {
        return PreferenceManager.getDefaultSharedPreferences(context)
                .getString(PREF_SEARCH_QUERY, null);
    }

    public static void setStoredQuery(Context context, String query) {
        PreferenceManager.getDefaultSharedPreferences(context)
                .edit()
                .putString(PREF_SEARCH_QUERY, query)
                .apply();
    }
}
```

Значение `PREF_SEARCH_QUERY` используется в качестве ключа для хранения запроса. Этот ключ применяется во всех операциях чтения или записи запроса.

Метод `getStoredQuery(Context)` возвращает значение запроса, хранящееся в общих настройках. Для этого метод сначала получает объект `SharedPreferences` по умолчанию для заданного контекста. (Так как `QueryPreferences` не имеет собственного контекста, вызывающий компонент должен передать свой контекст как входной параметр.)

Получение ранее сохраненного значения сводится к простому вызову `SharedPreferences.getString(…)`, `getInt(…)` или другого метода, соответствующего типу данных. Второй параметр `SharedPreferences.getString(PREF_SEARCH_QUERY, null)` определяет возвращаемое значение по умолчанию, которое должно возвращаться при отсутствии записи с ключом `PREF_SEARCH_QUERY`.

Метод `setStoredQuery(Context)` записывает запрос в хранилище общих настроек для заданного контекста. В приведенном выше коде вызов `SharedPreferences.edit()` используется для получения экземпляра `SharedPreferences.Editor`. Этот класс используется для сохранения значений в `SharedPreferences`. Он позволяет объединять изменения в транзакции, по аналогии с тем, как это делается в `FragmentTransaction`. Множественные изменения могут быть сгруппированы в одну операцию записи в хранилище.

После того как все изменения будут внесены, вызовите `apply()` для объекта `Editor`, чтобы эти изменения стали видимыми для всех пользователей файла `SharedPreferences`. Метод `apply()` вносит изменения в память немедленно, а непосредственная запись в файл осуществляется в фоновом потоке.

`QueryPreferences` предоставляет всю функциональность долгосрочного хранения данных для `PhotoGallery`. Теперь, когда у вас есть механизм простого сохранения и выборки последнего запроса пользователя, можно переходить к обновлению `PhotoGalleryFragment` для чтения и записи запроса.

Начните с обновления сохраненного запроса при отправке нового запроса пользователем.

Листинг 27.12. Сохранение сохраненного запроса в общих настройках (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {
    ...
    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater menuInflater) {
        ...
        searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {
            @Override
            public boolean onQueryTextSubmit(String s) {
                Log.d(TAG, "QueryTextSubmit: " + s);
                QueryPreferences.setStoredQuery(getActivity(), s);
                updateItems();
                return true;
            }

            @Override
            public boolean onQueryTextChange(String s) {
                Log.d(TAG, "QueryTextChange: " + s);
                return false;
            }
        });
    }
    ...
}
```

Каждый раз, когда пользователь выбирает элемент Clear Search в дополнительном меню, стирайте сохраненный запрос (присваиванием ему `null`).

Листинг 27.13. Стирание сохраненного запроса (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {
    ...
    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater menuInflater) {
        ...
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.menu_item_clear:
                QueryPreferences.setStoredQuery(getActivity(), null);
                updateItems();
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }
    ...
}
```

Обратите внимание на вызов `updateItems()` после обновления сохраненного запроса по аналогии с тем, как это делалось в листинге 27.12. Это гарантирует, что изображения, отображаемые в `RecyclerView`, соответствуют самому последнему поисковому запросу.

Наконец, обновите код `FetchItemsTask`, чтобы в нем использовался сохраненный запрос вместо фиксированной строки. Добавьте в `FetchItemsTask` конструктор, который получает строку запроса и сохраняет ее в переменной. Обновите метод `updateItems()`, чтобы он читал сохраненный запрос из общих настроек и использовал его для создания нового экземпляра `FetchItemsTask`. Все эти изменения представлены в листинге 27.14.

Листинг 27.14. Использование сохраненного запроса в FetchItemsTask (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {
    ...
    private void updateItems() {
        String query = QueryPreferences.getStoredQuery(getActivity());
        new FetchItemsTask(query).execute();
    }
    ...
    private class FetchItemsTask extends AsyncTask<Void,Void,List<GalleryItem>> {
        private String mQuery;
        public FetchItemsTask(String query) {
            mQuery = query;
        }
```

```
        @Override
        protected List<GalleryItem> doInBackground(Void... params) {
            String query = "robot"; // Для тестирования

            if (querymQuery == null) {
                return new FlickrFetchr().fetchRecentPhotos();
            } else {
                return new FlickrFetchr().searchPhotos(querymQuery);
            }
        }

        @Override
        protected void onPostExecute(List<GalleryItem> items) {
            mItems = items;
            setupAdapter();
        }
    }
}
```

Запустите приложение PhotoGallery, попробуйте провести поиск и посмотрите, что из этого выйдет.

Последний штрих

Осталось внести последнее усовершенствование: заполнить текстовое поле поиска сохраненным запросом, когда пользователь нажимает кнопку поиска для открытия `SearchView`. Метод `View.OnClickListener.onClick()` `SearchView` вызывается при нажатии этой кнопки. Подключите метод обратного вызова и задайте текст запроса `SearchView` при раскрытии представления.

Листинг 27.15. Предварительное заполнение SearchView (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {
    ...
    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater menuInflater) {
        ...
        searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {
            ...
        });

        searchView.setOnSearchClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String query = QueryPreferences.getStoredQuery(getActivity());
                searchView.setQuery(query, false);
            }
        });
    }
    ...
}
```

Запустите приложение и поэкспериментируйте с отправкой нескольких поисковых запросов. Проверьте, как работает последнее добавленное усовершенствование. Конечно, можно сделать еще один штрих...

Упражнение. Еще одно усовершенствование

Возможно, вы заметили, что при отправке запроса перед обновлением `RecyclerView` происходит небольшая задержка. В этом упражнении вам предлагается субъективно ускорить отклик приложения на отправленный запрос: сразу же после отправки запроса скройте виртуальную клавиатуру и сверните `SearchView`.

Очистите содержимое `RecyclerView` и отобразите индикатор загрузки (с неопределенным состоянием) сразу же после отправки запроса. Закройте индикатор загрузки сразу же после завершения загрузки данных JSON. Иначе говоря, индикатор загрузки не должен отображаться, когда код перейдет к загрузке отдельных изображений.

28 Фоновые службы

Весь код, написанный нами до настоящего момента, был связан с активностью; это подразумевало, что он связывается с некоей информацией на экране, видимой пользователю.

А если приложение не использует экран? Что, если выполняемые им операции не требуют визуального представления, как, скажем, воспроизведение музыки или проверка новых сообщений в блогах из поставки RSS? Для таких целей следует создать *службу* (service).

В этой главе мы добавим в PhotoGallery новую функцию фонового оповещения о появлении новых результатов поиска. Каждый раз, когда становится доступным новый результат, пользователь получает уведомление на панели состояния.

Создание IntentService

Начнем с создания службы. В этой главе мы будем использовать класс `IntentService`. Это не единственная разновидность служб, но, пожалуй, самая распространенная. Создайте субкласс `IntentService` с именем `PollService`. Эта служба будет использоваться нами для опроса результатов поиска.

Листинг 28.1. Создание PollService (PollService.java)

```
public class PollService extends IntentService {
    private static final String TAG = "PollService";

    public static Intent newIntent(Context context) {
        return new Intent(context, PollService.class);
    }

    public PollService() {
        super(TAG);
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        Log.i(TAG, "Received an intent: " + intent);
    }
}
```

Это очень простая реализация `IntentService`. Что она делает? В общем-то она отчасти похожа на активность. Она является контекстом (`Service` — субкласс `Context`) и реагирует на интенты (как видно из `onHandleIntent(Intent)`). Следуя общепринятой схеме, мы добавили метод `newIntent(Context)`. Каждый компонент, который хочет запустить эту службу, должен использовать `newIntent(…)`.

Интенты службы называются *командами* (commands). Каждая команда представляет собой инструкцию по выполнению некоторой операции для службы. Способ обработки команды зависит от вида службы.

Служба `IntentService` извлекает команды из очереди, как показано на рис. 28.1.

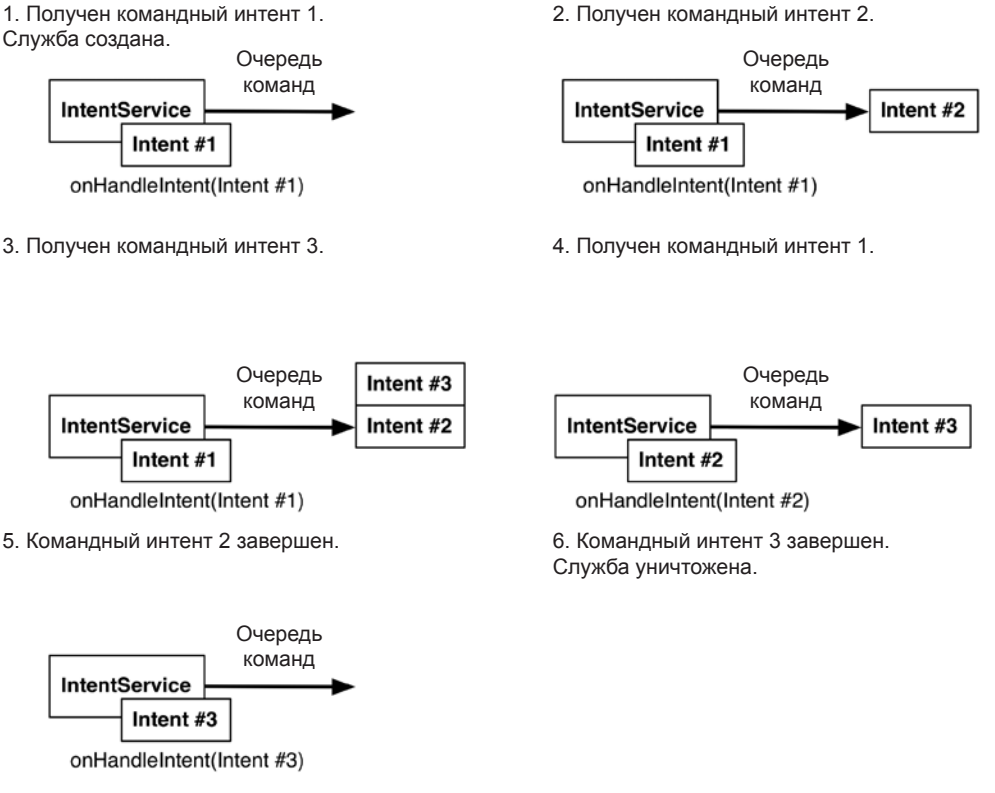


Рис. 28.1. Как IntentService обслуживает команды

При получении первой команды `IntentService` инициализируется, запускает фоновый поток и помещает команду в очередь.

Затем `IntentService` переходит к последовательной обработке команд, с вызовом метода `onHandleIntent(Intent)` своего фонового потока для каждой команды. Новые поступающие команды ставятся в очередь. Когда в очереди не остается ни одной команды, служба останавливается и уничтожается.

Приведенное описание относится только к `IntentService`. Позднее в этой главе службы и принципы обработки команд будут рассмотрены в более широкой перспективе.

Поскольку службы, как и активности, реагируют на интенты, они также должны объявляться в файле AndroidManifest.xml. Добавьте в манифест элемент для службы `PollService`.

Листинг 28.2. Добавление службы в манифест (AndroidManifest.xml)

```
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bignerdranch.android.photogallery" >

    <uses-permission android:name="android.permission.INTERNET" />

    <application
        ... >
        <activity
            android:name=".PhotoGalleryActivity"
            android:label="@string/app_name" >
            ...
        </activity>
        <service android:name=".PollService" />
    </application>

</manifest>
```

Добавьте код запуска службы в `PhotoGalleryFragment`.

Листинг 28.3. Добавление кода запуска службы (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {

    private static final String TAG = "PhotoGalleryFragment";
    ...
    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...
        updateItems();

        Intent i = PollService.newIntent(getActivity());
        getActivity().startService(i);

        Handler responseHandler = new Handler();
        mThumbnailDownloader = new ThumbnailDownloader<>(responseHandler);
        ...
    }
    ...
}
```

Запустите приложение и посмотрите в LogCat, что получилось:

```
02-23 14:25:32.450    2692-2717/com.bignerdranch.android.photogallery
                      I/PollService:
 Received an intent: Intent { cmp=com.bignerdranch.android.photogallery/
                     .PollService }
```

Зачем нужны службы

Ладно, признаем: все эти записи LogCat выглядят скучно. Но сам-то код очень интересный! Почему? Что с ним можно сделать?

Пора вернуться в вымышленный мир, где мы уже не программисты, а хозяева обувного магазина с продавцами-супергероями.

Продавцы-Флэши могут работать в двух местах: в торговом зале, где они общаются с покупателями, и на складе, куда покупатели не заходят. Склад может быть большим или маленьким в зависимости от магазина.

До настоящего момента весь наш код выполнялся в активностях. Активности — «прилавок» приложений Android. Весь этот код направлен на то, чтобы обеспечить приятные визуальные впечатления у пользователя.

Службы — своего рода «склад» приложений Android. Здесь происходит то, о чем пользователю знать не обязательно. Работа здесь может продолжаться после того, как торговый зал будет закрыт, когда активности давно перестали существовать.

Впрочем, довольно о магазинах. Что такого можно сделать со службой, чего нельзя сделать с активностью? Например, службу можно запустить, пока пользователь занимается другими делами.

Безопасные сетевые операции в фоновом режиме

Наша служба будет опрашивать Flickr в фоновом режиме. Чтобы выполнение сетевых операций в фоновом режиме проходило безопасно, потребуется дополнительный код. Android дает пользователю возможность отключить сетевые операции для фоновых приложений. Если вы применяете множество приложений, интенсивно использующих вычислительные ресурсы, это может существенно повысить производительность.

Однако это означает, что при выполнении операций в фоновом режиме необходимо при помощи объекта `ConnectivityManager` убедиться в том, что сеть доступна.

Добавьте код из листинга 28.4 для выполнения этих проверок.

Листинг 28.4. Проверка доступности сети для фоновых операций (PollService.java)

```
public class PollService extends IntentService {
    private static final String TAG = "PollService";
    ...
    @Override
    protected void onHandleIntent(Intent intent) {
        if (!isNetworkAvailableAndConnected()) {
```

```
            return;
        }

        Log.i(TAG, "Received an intent: " + intent);
    }

    private boolean isNetworkAvailableAndConnected() {
        ConnectivityManager cm =
                (ConnectivityManager) getSystemService(CONNECTIVITY_SERVICE);
        boolean isNetworkAvailable = cm.getActiveNetworkInfo() != null;
        boolean isNetworkConnected = isNetworkAvailable &&
                cm.getActiveNetworkInfo().isConnected();

        return isNetworkConnected;
    }
}
```

Логика проверки доступности сети сосредоточена в методе `isNetworkAvailableAndConnected()`. Запрет фоновой загрузки данных полностью блокирует возможность использования сети фоновыми службами. В этом случае `ConnectivityManager.getActiveNetworkInfo()` возвращает `null`, а с точки зрения фоновой службы все выглядит так, словно сеть недоступна (независимо от ее фактического состояния).

Если сеть доступна для вашей фоновой службы, она получает экземпляр `android.net.NetworkInfo`, представляющий текущее сетевое подключение. Затем код проверяет наличие полноценного сетевого подключения, вызывая `NetworkInfo.isConnected()`.

Если приложение не воспринимает сеть как доступную или же устройство не полностью подключено к сети, `onHandleIntent(…)` возвращает управление без выполнения оставшейся части метода (и не будет пытаться загружать данные, когда вы добавите соответствующий код). Привыкните выполнять подобную проверку, потому что приложение не сможет загрузить никакие данные, если оно не подключено к сети.

И еще одно: чтобы использовать метод `getActiveNetworkInfo()`, также необходимо получить разрешение `ACCESS_NETWORK_STATE`. Как вы уже знаете, управление разрешениями происходит в манифесте.

Листинг 28.5. Получение разрешения для проверки состояния сети (AndroidManifest.xml)

```
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bignerdranch.android.photogallery" >

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

    <application
        ... >
        ...
    </application>

</manifest>
```

Поиск новых результатов

Наша служба будет опрашивать Flickr на появление новых результатов, поэтому ей нужно знать результат последней выборки. Для этой работы идеально подойдет механизм `SharedPreferences`.

Добавьте в `QueryPreferences` константу для хранения идентификатора последней загруженной фотографии.

Листинг 28.6. Добавление константы для хранения идентификатора (QueryPreferences.java)

```
public class QueryPreferences {
    private static final String PREF_SEARCH_QUERY = "searchQuery";
    private static final String PREF_LAST_RESULT_ID = "lastResultId";

    public static String getStoredQuery(Context context) {
        ...
    }

    public static void setStoredQuery(Context context, String query) {
        ...
    }

    public static String getLastResultId(Context context) {
        return PreferenceManager.getDefaultSharedPreferences(context)
                .getString(PREF_LAST_RESULT_ID, null);
    }

    public static void setLastResultId(Context context, String lastResultId) {
        PreferenceManager.getDefaultSharedPreferences(context)
                .edit()
                .putString(PREF_LAST_RESULT_ID, lastResultId)
                .apply();
    }
}
```

Следующим шагом станет заполнение кода службы. Необходимо сделать следующее:

1. Прочитать текущий запрос и идентификатор последнего результата из `SharedPreferences` по умолчанию.
2. Загрузить последний набор результатов с использованием `FlickrFetchr`.
3. Если набор не пуст, получить первый результат.
4. Проверить, отличается ли его идентификатор от идентификатора последнего результата.
5. Сохранить первый результат в `SharedPreferences`.

Давайте вернемся к файлу PollService.java и претворим этот план в жизнь. В листинге 28.7 содержится довольно длинный блок кода, но в нем нет ничего такого, чего бы вы не видели ранее.

Листинг 28.7. Проверка новых результатов (PollService.java)

```
public class PollService extends IntentService {
    private static final String TAG = "PollService";
    ...
    @Override
    protected void onHandleIntent(Intent intent) {
        ...
        Log.i(TAG, "Received an intent: " + intent);
        String query = QueryPreferences.getStoredQuery(this);
        String lastResultId = QueryPreferences.getLastResultId(this);
        List<GalleryItem> items;

        if (query == null) {
            items = new FlickrFetchr().fetchRecentPhotos();
        } else {
            items = new FlickrFetchr().searchPhotos(query);
        }

        if (items.size() == 0) {
            return;
        }
        String resultId = items.get(0).getId();
        if (resultId.equals(lastResultId)) {
            Log.i(TAG, "Got an old result: " + resultId);
        } else {
            Log.i(TAG, "Got a new result: " + resultId);
        }

        QueryPreferences.setLastResultId(this, resultId);
    }
    ...
}
```

Видите каждый шаг из приведенного списка? Хорошо.

Запустите приложение PhotoGallery, и вы увидите, как приложение получает исходные результаты. Если поисковый запрос уже выбран, вероятно, при последующих запусках будут отображаться устаревшие результаты.

Отложенное выполнение и AlarmManager

Чтобы служба реально использовалась в фоновом режиме, нам понадобится какой-то механизм организации операций при отсутствии работающих активностей, например таймер, который срабатывает каждые пять минут, или что-нибудь в этом роде.

Это можно сделать при помощи `Handler`, вызовами методов `Handler.sendMessageDelayed(…)` или `Handler.postDelayed(…)`. Впрочем, такое решение с большой вероятностью перестанет работать, если пользователь уйдет со всех ак-

тивностей. Процесс закроется, и вместе с ним прекратят существование сообщения `Handler`.

По этой причине вместо `Handler` мы будем использовать `AlarmManager` — системную службу, которая может отправлять интенты за вас.

Как сообщить `AlarmManager`, какие интенты нужно отправить? При помощи объекта `PendingIntent`. По сути, в объекте `PendingIntent` упаковывается пожелание: «Я хочу запустить `PollService`». Затем это пожелание отправляется другим компонентам системы, таким как `AlarmManager`.

Включите в `PollService` новый метод с именем `setServiceAlarm(Context,boolean)`, который включает и отключает сигнал за вас. Метод будет объявлен статическим; это делается для того, чтобы код сигнала размещался рядом с другим кодом `PollService`, с которым он связан, но мог вызываться и другими компонентами. Обычно включение и отключение должно осуществляться из интерфейсного кода фрагмента или из другого контроллера.

Листинг 28.8. Добавление сигнального метода (PollService.java)

```
public class PollService extends IntentService {
    private static final String TAG = "PollService";

    // 60 секунд
    private static final long POLL_INTERVAL_MS = TimeUnit.MINUTES.toMillis(1);

    public static Intent newIntent(Context context) {
        return new Intent(context, PollService.class);
    }

    public static void setServiceAlarm(Context context, boolean isOn) {
        Intent i = PollService.newIntent(context);
        PendingIntent pi = PendingIntent.getService(context, 0, i, 0);

        AlarmManager alarmManager = (AlarmManager)
                context.getSystemService(Context.ALARM_SERVICE);

        if (isOn) {
            alarmManager.setRepeating(AlarmManager.ELAPSED_REALTIME,
                    SystemClock.elapsedRealtime(), POLL_INTERVAL_MS, pi);
        } else {
            alarmManager.cancel(pi);
            pi.cancel();
        }
    }
    ...
}
```

Метод начинается с создания объекта `PendingIntent`, который запускает `PollService`. Задача решается вызовом метода `PendingIntent.getService(…)`, в котором упаковывается вызов `Context.startService(Intent)`. Метод получает четыре параметра: `Context` для отправки интента; код запроса, по которому этот объект `PendingIntent` отличается от других; отправляемый объект `Intent` и, нако-

нец, набор флагов, управляющий процессом создания PendingIntent (вскоре мы используем один из них).

После этого сигнал либо устанавливается, либо отменяется.

Чтобы установить сигнал, следует вызвать AlarmManager.setRepeating(…). Этот метод тоже получает четыре параметра: константу для описания временной базы сигнала (об этом чуть позже), время запуска сигнала, временной интервал повторения сигнала, и наконец, объект PendingIntent, запускаемый при срабатывании сигнала.

Использование константы AlarmManager.ELAPSED_REALTIME задает начальное время запуска относительно прошедшего реального времени: SystemClock.elapsedRealtime(). В результате сигнал срабатывает по истечении заданного промежутка времени. Если бы вместо этого использовалась константа AlarmManager.RTC, то начальное время определялось бы текущим временем (то есть System.currentTimeMillis()), а сигнал срабатывал в заданный фиксированный момент времени.

Отмена сигнала осуществляется вызовом AlarmManager.cancel(PendingIntent). Обычно при этом также следует отменить и PendingIntent. Вскоре вы увидите, как отмена PendingIntent помогает в отслеживании статуса сигнала.

Добавьте простой тестовый код для запуска сигнала из PhotoGalleryFragment.

Листинг 28.9. Добавление кода запуска сигнала (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {
    private static final String TAG = "PhotoGalleryFragment";
    ...
    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...
        updateItems();

        Intent i = PollService.newIntent(getActivity());
        getActivity().startService(i);
        PollService.setServiceAlarm(getActivity(), true);

        Handler responseHandler = new Handler();
        mThumbnailDownloader = new ThumbnailDownloader<>(responseHandler);
        ...
    }
    ...
}
```

Введите этот код и запустите PhotoGallery. Сразу нажмите кнопку Back и выйдите из приложения.

Замечаете что-нибудь в LogCat? PollService честно продолжает работать, запускаясь каждые 60 секунд. Для этого и нужен класс AlarmManager. Даже если процесс будет завершен, AlarmManager будет выдавать интенты, снова и снова запуская PollService. (Конечно, такое поведение в высшей степени возмутительно. Возможно, вам стоит удалить приложение, пока ситуация не будет исправлена.)

Правильное использование сигналов

Насколько точным должно быть повторение? Многократное повторение работы в фоновой службе может израсходовать запас аккумулятора и создает лишнюю нагрузку для службы управления данными. Кроме того, выход устройства из спящего режима (запуск процессора после отключения экрана для выполнения работы) может быть затратной операцией. К счастью, сигнал можно настроить так, чтобы снизить затраты на интервальное планирование работы и пробуждение.

Неточное и точное повторение

Метод `setRepeating(…)` задает повторение сигнала, но это повторение не является точным. Иначе говоря, Android сохраняет за собой право немного сдвинуть его во времени. В результате наименьшее возможное время, которое можно установить в стандартной версии Android, составляет 60 секунд. (На других устройствах может быть чуть больше.)

Дело в том, что сигналы могут очень сильно влиять на заряд батареи. Каждый раз, когда срабатывает сигнал, устройство должно «пробудиться» и активизировать приложение. Многие приложения (в том числе и PhotoGallery) используют передатчик телефона для доступа к Интернету, что приводит к еще более интенсивному расходованию заряда.

Если бы других приложений не было, точность сигнала была бы не особенно существенной. В конце концов, если ваш сигнал активизируется каждые 15 минут, и других 15-минутных сигналов нет, то телефон будет пробуждаться и включать передатчик 4 раза в час независимо от точности сигнала.

Но если на устройстве работает ваше приложение и 9 других приложений с 15-минутными сигналами, ситуация изменяется. Так как каждый сигнал должен быть точным, устройство должно пробуждаться по каждому сигналу. А это означает, что передатчик будет включаться 40 раз в час, а не 4.

Механизм «неточного повторения» означает, что Android предоставляется право сдвигать сигналы во времени, так что они могут не срабатывать каждые 15 минут. В результате каждые 15 минут ваше устройство может пробуждаться для одновременной отработки всех десяти 15-минутных сигналов. Количество пробуждений устройства сокращается с 40 до 4, что обеспечивает значительную экономию заряда.

Конечно, некоторым приложениям необходимы точные сигналы. В таких случаях приходится использовать метод `AlarmManager.setWindow(…)` или `AlarmManager.setExact(…)` для назначения точного сигнала, срабатывающего ровно один раз. Повторение придется реализовать самостоятельно.

Временная база

Другое важное решение — выбор временной базы. Есть два основных варианта: `AlarmManager.ELAPSED_REALTIME` и `AlarmManager.RTC`.

С AlarmManager.ELAPSED_REALTIME за основу для интервальных вычислений берется промежуток времени, прошедший с момента последней загрузки устройства (включая время сна). Режим ELAPSED_REALTIME лучше всего подходит для сигнала в PhotoGallery, потому что он базируется на относительном течении времени, а значит, не зависит от физического времени. (Кроме того, в документации рекомендуется использовать ELAPSED_REALTIME вместо RTC везде, где это возможно.)

AlarmManager.RTC использует абсолютное время в формате UTC. Учтите, что формат UTC не учитывает локальный контекст (locale), а это, вероятно, противоречит представлениям пользователя. Это означает, что если вы захотите установить сигнал на конкретное абсолютное время, вам придется самостоятельно реализовать обработку локального контекста при использовании режима RTC. В остальных случаях используйте временную базу ELAPSED_REALTIME.

Если вы используете один из вариантов базы, описанных выше, ваш сигнал не сработает при нахождении устройства в спящем режиме (с выключенным экраном) даже по истечении заданного интервала. Чтобы ваш сигнал срабатывал с более точным интервалом или моментом времени, используйте одну из констант временной базы, выводящих устройство из спящего режима: AlarmManager.ELAPSED_REALTIME_WAKEUP и AlarmManager.RTC_WAKEUP. Тем не менее режимов с пробуждением следует по возможности избегать, если только сигнал не должен срабатывать в точно заданное время.

PendingIntent

Давайте поближе познакомимся с PendingIntent. PendingIntent представляет собой объект-маркер. Когда вы получаете такой объект вызовом PendingIntent.getService(…), вы тем самым говорите ОС: «Пожалуйста, запомни, что я хочу отправлять этот интент вызовом startService(Intent)». Позднее вы можете вызвать send() для PendingIntent, и ОС отправит изначально упакованный интент — точно так, как вы приказали.

А лучше всего здесь то, что если передать маркер PendingIntent другой стороне и эта сторона его использует, маркер будет отправлен *от имени вашего приложения*. А поскольку объект PendingIntent существует в ОС, вы сохраняете полный контроль над ним. Например, вы можете (просто из вредности) предоставить кому-нибудь объект PendingIntent и немедленно отменить его, так что вызов send() ничего не сделает.

Если вы запросите PendingIntent дважды с одним интентом, то получите тот же PendingIntent. Например, таким образом можно проверить существование PendingIntent или отменить ранее выданный объект PendingIntent.

Управление сигналами с использованием PendingIntent

Для каждого объекта PendingIntent можно зарегистрировать только один сигнал. Именно так работает вызов setServiceAlarm(boolean) при ложном значении isOn: он вызывает AlarmManager.cancel(PendingIntent) для отмены сигнала, связанного с вашим объектом PendingIntent, а потом отменяет PendingIntent.

Так как PendingIntent также удаляется при отмене сигнала, вы можете проверить, существует ли PendingIntent, чтобы узнать, активен сигнал или нет. Эта операция выполняется передачей флага PendingIntent.FLAG_NO_CREATE вызову PendingIntent.getService(…). Флаг говорит, что если объект PendingIntent не существует, то вместо его создания следует вернуть null.

Напишите новый метод isServiceAlarmOn(Context), использующий флаг PendingIntent.FLAG_NO_CREATE для проверки сигнала.

Листинг 28.10. Добавление метода isServiceAlarmOn() (PollService.java)

```
public class PollService extends IntentService {
    ...
    public static void setServiceAlarm(Context context, boolean isOn) {
        ...
    }

    public static boolean isServiceAlarmOn(Context context) {
        Intent i = PollService.newIntent(context);
        PendingIntent pi = PendingIntent
                .getService(context, 0, i, PendingIntent.FLAG_NO_CREATE);
        return pi != null;
    }
    ...
}
```

Так как этот объект PendingIntent используется только для установки сигнала, null вместо PendingIntent означает, что сигнал не установлен.

Управление сигналом

Теперь, когда мы можем включать и отключать сигнал (а также определять, включен ли он), давайте добавим интерфейс для его включения и выключения. Добавьте в файл menu/fragment_photo_gallery.xml новую команду меню.

Листинг 28.11. Переключение режима опроса (menu/fragment_photo_gallery.xml)

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">
    <item android:id="@+id/menu_item_search"
          ... />

    <item android:id="@+id/menu_item_clear"
          ... />

    <item android:id="@+id/menu_item_toggle_polling"
          android:title="@string/start_polling"
          app:showAsAction="ifRoom" />
</menu>
```

Затем необходимо добавить несколько новых строк — одну для начала опроса, другую для завершения опроса. (Позднее нам понадобится еще пара строк для оповещений на панели состояния; добавим их сейчас.)

Листинг 28.12. Добавление строк для режима опроса (res/values/strings.xml)

```
<resources>
    ...
    <string name="search">Search</string>
    <string name="clear_search">Clear Search</string>
    <string name="start_polling">Start polling</string>
    <string name="stop_polling">Stop polling</string>
    <string name="new_pictures_title">New PhotoGallery Pictures</string>
    <string name="new_pictures_text">You have new pictures in PhotoGallery.</
string>

</resources>
```

Удалите старый отладочный код для запуска сигнала и добавьте реализацию команды меню.

Листинг 28.13. Реализация команды переключения режима опроса (PhotoGalleryFragment.java)

```
private static final String TAG = "PhotoGalleryFragment";
...
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ...
    updateItems();

    PollService.setServiceAlarm(getActivity(), true);

    Handler responseHandler = new Handler();
    ...
}
...
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_clear:
            QueryPreferences.setStoredQuery(getActivity(), null);
            updateItems();
            return true;
        case R.id.menu_item_toggle_polling:
            boolean shouldStartAlarm = !PollService.isServiceAlarmOn
                                       (getActivity());
            PollService.setServiceAlarm(getActivity(), shouldStartAlarm);
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Теперь вы сможете включать и отключать сигнал. Однако следует заметить, что в тексте элемента меню опроса всегда выводится надпись Start polling, даже если опрос в настоящее время активен. Вместо этого следует переключить заголовок команды меню подобно тому, как это делалось в приложении CriminalIntent для SHOW SUBTITLE (глава 13).

В `onCreateOptionsMenu(…)` проверьте, что сигнал активен, и измените текст `menu_item_toggle_polling`, чтобы приложение выводило надпись, соответствующую текущему состоянию.

Листинг 28.14. Переключение текста элемента меню (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {
    private static final String TAG = "PhotoGalleryFragment";
    ...
    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater menuInflater) {
        super.onCreateOptionsMenu(menu, menuInflater);
        menuInflater.inflate(R.menu.fragment_photo_gallery, menu);

        MenuItem searchItem = menu.findItem(R.id.menu_item_search);
        final SearchView searchView = (SearchView) searchItem.getActionView();

        searchView.setOnQueryTextListener(…);

        searchView.setOnSearchClickListener(…);

        MenuItem toggleItem = menu.findItem(R.id.menu_item_toggle_polling);
        if (PollService.isServiceAlarmOn(getActivity())) {
            toggleItem.setTitle(R.string.stop_polling);
        } else {
            toggleItem.setTitle(R.string.start_polling);
        }
    }
    ...
}
```

Затем в методе `onOptionsItemSelected(MenuItem)` прикажите `PhotoGalleryActivity` обновить меню на панели инструментов.

Листинг 28.15. Перерисовка меню (PhotoGalleryFragment.java)

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_clear:
            ...
        case R.id.menu_item_toggle_polling:
            boolean shouldStartAlarm = !PollService.isServiceAlarmOn
                                        (getActivity());
            PollService.setServiceAlarm(getActivity(), shouldStartAlarm);
            getActivity().invalidateOptionsMenu();
            return true;
```

```
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

После этого изменения код переключения содержимого меню должен работать идеально. Однако... чего-то все же не хватает.

Оповещения

Служба успешно работает в фоновом режиме. Однако пользователь об этом понятия не имеет, так что пользы от нее будет не много.

Когда службе требуется передать какую-то информацию пользователю, для этого она почти всегда использует оповещения (notifications) — элементы, появляющиеся на выдвижной панели оповещений, которую пользователь вызывает, проводя пальцем вниз от верха экрана.

Чтобы опубликовать оповещение, необходимо сначала создать объект Notification. Объекты `Notification` создаются с использованием объектов-построителей — подобно тому, как это делалось с `AlertDialog` из главы 12. Как минимум объект `Notification` должен иметь:

- текст *бегущей строки*, отображаемый на панели состояния при первом появлении оповещения (начиная с Android 5.0 (Lollipop) бегущая строка уже не отображается на панели состояния, но может использоваться для улучшения доступности приложения);
- *значок*, отображаемый на панели состояния (на устройствах, предшествующих Lollipop, появляется после исчезновения бегущей строки);
- *представление*, отображаемое на выдвижной панели оповещений для вывода оповещения;
- объект `PendingIntent`, срабатывающий при нажатии на оповещении на выдвижной панели.

После того как объект `Notification` будет создан, его можно отправить вызовом метода `notify(int, Notification)` для системной службы `NotificationManager`.

Сначала необходимо добавить служебный код, приведенный в листинге 28.16. Откройте `PhotoGalleryActivity` и добавьте статический метод `newIntent-(Context)`. Этот метод возвращает экземпляр `Intent`, который может использоваться для запуска `PhotoGalleryActivity`. (Вскоре `PollService` будет вызывать `PhotoGalleryActivity.newIntent(…)`, упаковывать полученный интент в `PendingIntent` и назначать `PendingIntent` оповещению.)

Листинг 28.16. Добавление newIntent(…) в PhotoGalleryActivity (PhotoGalleryActivity.java)

```
public class PhotoGalleryActivity extends SingleFragmentActivity {

    public static Intent newIntent(Context context) {
```

```
        return new Intent(context, PhotoGalleryActivity.class);
    }

    @Override
    protected Fragment createFragment() {
        return PhotoGalleryFragment.newInstance();
    }
}
```

Чтобы служба `PollService` оповещала пользователя о появлении нового результата, добавьте код из листинга 28.17. Этот код создает объект `Notification` и вызывает `NotificationManager.notify(int, Notification)`.

Листинг 28.17. Добавление оповещения (PollService.java)

```
@Override
protected void onHandleIntent(Intent intent) {
    ...
    String resultId = items.get(0).getId();
    if (resultId.equals(lastResultId)) {
        Log.i(TAG, "Got an old result: " + resultId);
    } else {
        Log.i(TAG, "Got a new result: " + resultId);

        Resources resources = getResources();
        Intent i = PhotoGalleryActivity.newIntent(this);
        PendingIntent pi = PendingIntent.getActivity(this, 0, i, 0);

        Notification notification = new NotificationCompat.Builder(this)
                .setTicker(resources.getString(R.string.new_pictures_title))
                .setSmallIcon(android.R.drawable.ic_menu_report_image)
                .setContentTitle(resources.getString(R.string.new_pictures_title))
                .setContentText(resources.getString(R.string.new_pictures_text))
                .setContentIntent(pi)
                .setAutoCancel(true)
                .build();

        NotificationManagerCompat notificationManager =
                NotificationManagerCompat.from(this);
        notificationManager.notify(0, notification);
    }

    QueryPreferences.setLastResultId(this, resultId);
}
```

Пройдемся по этому коду сверху вниз. Сначала мы задаем текст бегущей строки и значок вызовами `setTicker(CharSequence)` и `setSmallIcon(int)`. (Обратите внимание: упоминаемый ресурс значка является частью инфраструктуры Android и идентифицируется по имени пакета `android.R.drawable.some_drawable_resource_name`, поэтому вам не нужно перемещать изображение значка в папку ресурсов.)

После этого задается внешний вид оповещения на самой выдвижной панели. Можно полностью определить внешний вид оповещения, но проще использовать стандартное оформление со значком, заголовком и текстовой областью. Для значка будет использоваться значение из `setSmallIcon(int)`. Заголовок и текст задаются вызовами `setContentTitle(CharSequence)` и `setContentText(CharSequence)` соответственно.

Теперь необходимо указать, что происходит при нажатии на оповещении. Как и в случае с `AlarmManager`, для этого используется `PendingIntent`. Объект `PendingIntent`, передаваемый `setContentIntent(PendingIntent)`, будет запускаться при нажатии пользователем на вашем оповещении на выдвижной панели. Вызов `setAutoCancel(true)` слегка изменяет это поведение: с этим вызовом оповещение при нажатии также будет удаляться с выдвижной панели оповещений.

Код завершается получением экземпляра `NotificationManagerCompat` из текущего контекста (`NotificationManagerCompat.from(this)`) и вызовом `NotificationManager.notify(…)`. Передаваемый целочисленный параметр содержит идентификатор оповещения, уникальный в границах приложения. Если вы отправите второе оповещение с тем же идентификатором, оно заменит последнее оповещение, отправленное с этим идентификатором. Так реализуются индикаторы прогресса и другие динамические визуальные эффекты.

Собственно, это все. Запустите приложение и включите опрос. Через некоторое время на панели состояния появляется значок оповещения, а на панели оповещений — информация о появлении новых результатов.

Когда вы убедитесь в том, что все работает правильно, замените константу интервала опроса более разумным значением. (Одна из предопределенных интервальных констант `AlarmManager` обеспечит неточное повторение на устройствах с версиями, предшествующими KitKat.)

Листинг 28.18. Изменение периодичности опроса (PollService.java)

```
public class PollService extends IntentService {
    private static final String TAG = "PollService";

    // Set interval to 1 minute
    private static final long POLL_INTERVAL_MS = TimeUnit.MINUTES.toMillis(1);
    private static final long POLL_INTERVAL_MS = TimeUnit.MINUTES.toMillis(15);
    ...
}
```

Упражнение. Уведомления в Android Wear

Так как мы используем `NotificationCompat` и `NotificationManagerCompat`, уведомления будут автоматически появляться на устройстве Android Wear, если оно работает в паре с устройством Android, на котором выполняется ваше приложение. Пользователь, получивший оповещение на устройстве Wear, может смахнуть влево, чтобы на экране появилось предложение открыть приложение на связанном устройстве. Нажатие Open на устройстве Wear инициирует отложенный интент оповещения на подключенном устройстве.

Чтобы проверить этот факт, настройте эмулятор Android Wear и свяжите его с портативным устройством, на котором выполняется ваше приложение. За подробной информацией обращайтесь по адресу *developer.android.com*.

Для любознательных: подробнее о службах

Мы рекомендуем использовать `IntentService` для большинства реализаций служб. Если паттерн `IntentService` не подходит для вашей архитектуры, вам придется поближе познакомиться со службами для составления собственной реализации. Приготовьтесь, при изучении служб приходится учитывать множество подробностей и нюансов.

Что делают (и чего не делают) службы

Служба представляет собой компонент приложения, предоставляющий обратные вызовы жизненного цикла (в этом она похожа на активность). Эти обратные вызовы даже выполняются в главном потоке без каких-либо усилий с вашей стороны, как и в случае с активностью.

В своем исходном состоянии служба *не выполняет* никакой код в фоновом потоке. Это главная причина, по которой мы рекомендуем `IntentService`. Для большинства нетривиальных служб потребуется тот или иной фоновый поток, а `IntentService` автоматически управляет шаблонным кодом, необходимым для достижения этой цели.

Давайте посмотрим, какие обратные вызовы жизненного цикла предоставляет служба.

Жизненный цикл службы

Жизненный цикл службы, запущенной `startService(Intent)`, весьма прост. Ниже перечислены три метода обратного вызова жизненного цикла.

- `onCreate(…)` — вызывается при создании службы.
- `onStartCommand(Intent,int,int)` — вызывается каждый раз, когда компонент запускает службу вызовом `startService(Intent)`. Два целочисленных параметра содержат набор флагов и идентификатор запуска. Флаги указывают, является интент повторной отправкой ранее доставленного интента или повторной попыткой после неудачи при доставке. Идентификатор запуска отличается при разных вызовах `onStartCommand(Intent,int,int)`, поэтому по нему можно отличить эту команду от других.
- `onDestroy()` — вызывается, когда дальнейшее существование службы не требуется. Часто происходит после остановки службы.

Остается один вопрос: как происходит остановка службы? Это можно сделать разными способами в зависимости от типа службы. Тип службы определяется значением, возвращаемым методом `onStartCommand(…)`; возможные значения — `Service.START_NOT_STICKY, START_REDELIVER_INTENT` или `START_STICKY`.

Незакрепляемые службы

`IntentService` является *незакрепляемой* (non-sticky) службой, поэтому начнем с нее. Незакрепляемая служба останавливается, когда сама служба сообщает о завершении своей работы. Чтобы сделать свою службу незакрепляемой, верните `START_NOT_STICKY` или `START_REDELIVER_INTENT`.

Чтобы сообщить Android о завершении работы службы, вызовите метод `stopSelf()` или `stopSelf(int)`. Первый метод, `stopSelf()`, является безусловным. Он всегда останавливает службу независимо от того, сколько раз был вызван метод `onStartCommand(…)`.

Второй метод, `stopSelf(int)`, получает идентификатор запуска, полученный в `onStartCommand(…)`. Служба останавливается только в том случае, если этот идентификатор является самым последним из полученных идентификаторов запуска (так работает внутренняя реализация `IntentService`).

Чем же отличаются `START_NOT_STICKY` и `START_REDELIVER_INTENT`? Поведением службы, если системе потребуется завершить ее преждевременно. Служба `START_NOT_STICKY` просто прекращает существование и уходит в никуда. Служба `START_REDELIVER_INTENT`, напротив, попытается запуститься позднее, когда ресурсы не будут столь ограниченны.

Выбор между `START_NOT_STICKY` и `START_REDELIVER_INTENT` определяется важностью этой операции для вашего приложения. Если служба не критична, выберите режим `START_NOT_STICKY`. В PhotoGallery служба запускается по сигналу. Пропажа одного вызова не критична, поэтому мы выбираем `START_NOT_STICKY`. Такое поведение используется по умолчанию для `IntentService`. Чтобы переключиться на режим `START_REDELIVER_INTENT`, вызовите `IntentService.setIntentRedelivery(true)`.

Закрепляемые службы

Закрепляемая (sticky) служба остается запущенной, пока кто-то находящийся вне службы не прикажет ей остановиться, вызвав метод `Context.stopService(Intent)`. Чтобы сделать службу закрепляемой, верните значение `START_STICKY`.

После запуска закрепляемая служба остается «включенной», пока компонент не вызовет `Context.stopService(Intent)`. Если службу по какой-то причине потребуется уничтожить, она будет снова перезапущена с передачей `onStartCommand(…)` `null`-интента.

Закрепляемый режим хорошо подходит для долгоживущих служб (например, проигрывателя музыки), которые должны работать до тех пор, пока пользователь не прикажет им остановиться. Впрочем, даже в этом случае стоит рассмотреть альтернативную архитектуру с использованием незакрепляемых служб. Управлять закрепляемыми службами неудобно, потому что трудно определить, была ли уже запущена служба.

Привязка к службам

Также существует возможность привязки (binding) к службе при помощи метода `bindService(Intent,ServiceConnection,int)`. Привязка к службе — ме-

ханизм подключения к службе и непосредственного вызова ее методов. Привязка осуществляется вызовом `bindService(Intent,ServiceConnection,int)`. `ServiceConnection` — объект, представляющий привязку к службе и получающий все обратные вызовы привязки.

Во фрагменте код привязки выглядит примерно так:

```
private ServiceConnection mServiceConnection = new ServiceConnection() {
    public void onServiceConnected(ComponentName className,
            IBinder service) {
        // Используется для взаимодействия со службой
        MyBinder binder = (MyBinder)service;
    }

    public void onServiceDisconnected(ComponentName className) {
    }
};

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    Intent i = new Intent(getActivity(), MyService.class);
    getActivity().bindService(i, mServiceConnection, 0);
}

@Override
public void onDestroy() {
    super.onDestroy();
    getActivity().unbindService(mServiceConnection);
}
```

На стороне службы привязка определяет два дополнительных обратных вызова жизненного цикла:

- `onBind(Intent)` — вызывается каждый раз, когда создается привязка к службе. Возвращает объект `IBinder`, получаемый при вызове `ServiceConnection.onServiceConnected(ComponentName,IBinder)`.
- `onUnbind(Intent)` — вызывается при завершении привязки к службе.

Локальная привязка к службам

Что же собой представляет `MyBinder`? Если служба является локальной, это может быть простой объект Java, существующий в локальном процессе. Обычно он предоставляет дескриптор (handle), используемый для прямого вызова методов службы:

```
private class MyBinder extends IBinder {
    public MyService getService() {
        return MyService.this;
    }
}
```

```
@Override
public void onBind(Intent intent) {
    return new MyBinder();
}
```

Паттерн выглядит довольно заманчиво — это единственный механизм Android, позволяющий одному компоненту Android напрямую взаимодействовать с другим. Тем не менее мы не рекомендуем его использовать. Службы по сути являются синглетами, и такое их использование не предоставляет никаких заметных преимуществ перед использованием синглета.

Удаленная привязка к службе

Привязка приносит больше пользы для удаленных служб, потому что она дает возможность приложениям из других процессов вызывать методы вашей службы. Создание привязки к удаленной службе — нетривиальная тема, выходящая за рамки нашей книги. За подробностями обращайтесь к документации по AIDL или описанию класса `Messenger`.

Для любознательных: JobScheduler и JobServices

В этой главе было показано, как использовать `AlarmManager`, `IntentService` и `PendingIntents` для организации периодического выполнения фоновых задач. При этом вам приходится вручную:

- планировать периодическое выполнение задачи;
- проверять, выполняется ли периодическая задача в настоящее время;
- проверять, активна ли сеть.

Разработчику приходилось вручную «сшивать» несколько разных API для создания одного работоспособного фонового работника, которого можно запускать и останавливать. Решение работало, но реализация требовала большого количества работы.

В Android Lollipop (API 21) появился новый API `JobScheduler`, который мог реализовывать подобные операции самостоятельно. Он также был способен на большее: например, он может избежать запуска службы в случае недоступности сети. Он может реализовать политику «отступить и попробовать еще раз», если запрос проходит неудачно или доступ к сети ограничен из-за ограничений канала. Вы также можете ограничить обновления, чтобы они могли происходить только во время зарядки устройства. Хотя все это можно было сделать при помощи `AlarmManager` и `IntentService`, решение получалось достаточно сложным.

`JobScheduler` позволяет определять службы для выполнения некоторых заданий, а затем планировать их для выполнения при соблюдении заданных условий. Вот как это происходит: сначала вы создаете службу для выполнения задания. Класс службы должен быть субклассом `JobService`. Класс `JobService` содержит два пе-

реопределяемых метода: `onStartJob(JobParameters)` и `onStopJob(JobParameters)`. (Не вводите приведенный ниже код — он предназначен исключительно для демонстрации в контексте обсуждения.)

```
public class PollService extends JobService {

    @Override
    public boolean onStartJob(JobParameters params) {
        return false;
    }

    @Override
    public boolean onStopJob(JobParameters params) {
        return false;
    }
}
```

Когда система Android готова к запуску задания, ваша служба запустится и вы получите вызов `onStartJob(…)` в главном потоке. Возвращение `false` этим методом означает: «Я проявил инициативу и сделал все необходимое, так что все готово». Возвращение `true` означает: «Вас понял. Сейчас я над этим работаю, но работа еще не завершена».

В отличие от `IntentService`, `JobService` предполагает, что вы самостоятельно реализуете управление потоками; это немного добавляет хлопот. Например, можно воспользоваться классом `AsyncTask`:

```
private PollTask mCurrentTask;
@Override
public boolean onStartJob(JobParameters params) {
    mCurrentTask = new PollTask();
    mCurrentTask.execute(params);
    return true;
}

private class PollTask extends AsyncTask<JobParameters,Void,Void> {
    @Override
    protected Void doInBackground(JobParameters... params) {
        JobParameters jobParams = params[0];

        // Проверка новых изображений на Flickr

        jobFinished(jobParams, false);
        return null;
    }
}
```

После завершения задания вызывается метод `jobFinished(JobParameters, boolean)`. Передача `true` во втором параметре означает, что выполнить задание в этот раз не удалось и задание следует запланировать заново на будущее.

Метод `onStopJob(JobParameters)` предназначен для ситуаций, в которых выполнение задания требуется прервать. Предположим, вы хотите, чтобы задание вы-

полнялось только при наличии доступного подключения WiFi. Если телефон выходит из зоны действия WiFi перед вызовом `jobFinished(…)`, вы получите вызов `onStopJob(…)`, по которому следует немедленно прервать все происходящее.

```
@Override
public boolean onStopJob(JobParameters params) {
    if (mCurrentTask != null) {
        mCurrentTask.cancel(true);
    }
    return true;
}
```

Вызов `onStopJob(…)` — признак того, что служба собирается завершиться. Ожидание недопустимо: все должно остановиться немедленно. Возвращение `true` означает, что задание должно быть заново спланировано на будущее. Возвращение `false` означает: «Ладно, будем считать, что это все. Планировать заново не нужно».

Когда вы регистрируете свою службу в манифесте, ее необходимо экспортировать и добавить разрешение:

```
<service
    android:name=".PollService"
    android:permission="android.permission.BIND_JOB_SERVICE"
    android:exported="true"/>
```

Экспортирование открывает доступ к службе, а добавление разрешения снова ограничивает его, так что служба может запускаться только `JobScheduler`.

После того как служба `JobService` будет создана, запустить ее уже несложно. Чтобы проверить, было ли задание запланировано для выполнения, можно воспользоваться `JobScheduler`.

```
final int JOB_ID = 1;

JobScheduler scheduler = (JobScheduler)
    context.getSystemService(Context.JOB_SCHEDULER_SERVICE);

boolean hasBeenScheduled = false;
for (JobInfo jobInfo : scheduler.getAllPendingJobs()) {
    if (jobInfo.getId() == JOB_ID) {
        hasBeenScheduled = true;
    }
}
```

Если задание не запланировано, вы можете создать новый объект `JobInfo` с информацией о том, когда оно должно выполняться. Когда же должна запускаться наша служба `PollService`? Как насчет чего-нибудь в этом роде:

```
final int JOB_ID = 1;

JobScheduler scheduler = (JobScheduler)
    context.getSystemService(Context.JOB_SCHEDULER_SERVICE);
```

```
JobInfo jobInfo = new JobInfo.Builder(
        JOB_ID, new ComponentName(context, PollService.class))
        .setRequiredNetworkType(JobInfo.NETWORK_TYPE_UNMETERED)
        .setPeriodic(1000 * 60 * 15)
        .setPersisted(true)
        .build();
scheduler.schedule(jobInfo);
```

Этот фрагмент планирует выполнение задания каждые 15 минут, но только при наличии WiFi или другой неограниченной сети. Вызов `setPersisted(true)` также обеспечивает долгосрочное существование задания: оно переживет перезагрузку. За информацией о других возможностях настройки `JobInfo` обращайтесь к справочной документации.

JobScheduler и будущее фоновых операций

В этой главе мы показали, как реализовать фоновые операции без использования `JobScheduler`. `JobScheduler` поддерживается только в Lollipop и выше, а реализация в библиотеке поддержки отсутствует, поэтому если вы хотите, чтобы ваше решение использовало только стандартные библиотеки и работало во всех версиях Android, единственным вариантом оказывается решение на базе `AlarmManager`.

Однако стоит заметить, что время выполнения такой работы в `AlarmManager` подходит к концу. В последние годы одним из главных приоритетов разработчиков платформы Android было повышение эффективности расходования заряда батареи. Для этого они стремятся сильнее влиять на планирование операций при использовании передатчика, WiFi и других подсистем, способных быстро расходовать заряд батареи.

Именно из-за этого смысл команд `AlarmManager` изменялся с годами: Android знает, что разработчики используют `AlarmManager` для планирования фоновой работы. Вызовы API адаптировались для того, чтобы ваше приложение нормально уживалось с другими приложениями.

Однако по своей сути класс `AlarmManager` плохо подходит для этой цели. Он ничего не сообщает Android о том, что делает ваше приложение — использует передатчик GPS, обновляет виджет на домашнем экране пользователя или что-нибудь еще. Система Android об этом ничего не знает, поэтому ей приходится обрабатывать все сигналы одинаково. Это мешает Android принимать обоснованные решения относительно энергопотребления.

А следовательно, как только большинство приложений сможет использовать `JobScheduler`, популярность `AlarmManager` быстро сойдет на нет. Итак, хотя решения на базе `JobScheduler` сейчас не обладают полной совместимостью, мы настоятельно рекомендуем при первой возможности переводить приложения на эту технологию.

Если вы хотите сделать что-то прямо сейчас (вместо того, чтобы планировать переход на другой API в будущем), используйте стороннюю библиотеку совместимости. На момент написания книги лучшим вариантом была библиотека Evernote android-job (*github.com/evernote/android-job*).

Упражнение. Использование JobService в Lollipop

Создайте вторую реализацию `PollService`, которая субклассирует `JobService` и выполняется с использованием `JobScheduler`. В стартовом коде `PollService` проверьте, работает ли приложение в Lollipop. Если проверка дает положительный результат, используйте `JobScheduler`, а если нет — вернитесь к старой реализации с `AlarmManager`.

Для любознательных: синхронизирующие адаптеры

Другой способ создания веб-службы для регулярного опроса основан на использовании *синхронизирующих адаптеров* (sync adapter). Синхронизирующие адаптеры не похожи на те адаптеры, с которыми вы имели дело ранее. Их единственное назначение — синхронизация данных с источником данных (отправка и/или загрузка). В отличие от `JobScheduler`, синхронизирующие адаптеры существуют уже давно, и вам не придется беспокоиться о том, в какой версии Android работает приложение.

Как и `JobScheduler`, синхронизирующие адаптеры могут использоваться как замена для той настройки `AlarmManager`, которую мы проводили в PhotoGallery. Синхронизации от нескольких приложений по умолчанию группируются, не требуя от вас специально устанавливать какие-либо флаги. Более того, вам не нужно беспокоиться о сбросе синхросигналов между перезагрузками, потому что синхронизирующие адаптеры сделают это за вас.

Синхронизирующие адаптеры также хорошо интегрируются с ОС с точки зрения пользователя. Приложение можно представить как учетную запись с поддержкой синхронизации, которой пользователь может управлять из меню Settings ▸ Accounts. В этом меню пользователи управляют учетными записями других приложений, использующих синхронизирующие адаптеры, например приложениями из пакета Google (рис. 28.2).

Хотя синхронизирующие адаптеры упрощают правильную организацию планирования повторяющихся сетевых операций и позволяют избавиться от кода управления сигналами и отложенными интентами, общий объем кода при работе с ними несколько возрастает. Во-первых, синхронизирующий адаптер не выполняет за вас никакие веб-запросы, поэтому вам придется написать соответствующий код (например, `FlickrFetchr`). Во-вторых, ему необходима реализация контент-провайдера для упаковки классов данных, учетной записи и удостоверений для представления учетной записи на удаленном сервере (даже если сервер не требует аутентификации); добавьте к этому реализацию синхронизирующего адаптера и службы синхронизации. Также от вас потребуется хорошее знание связываемых служб.

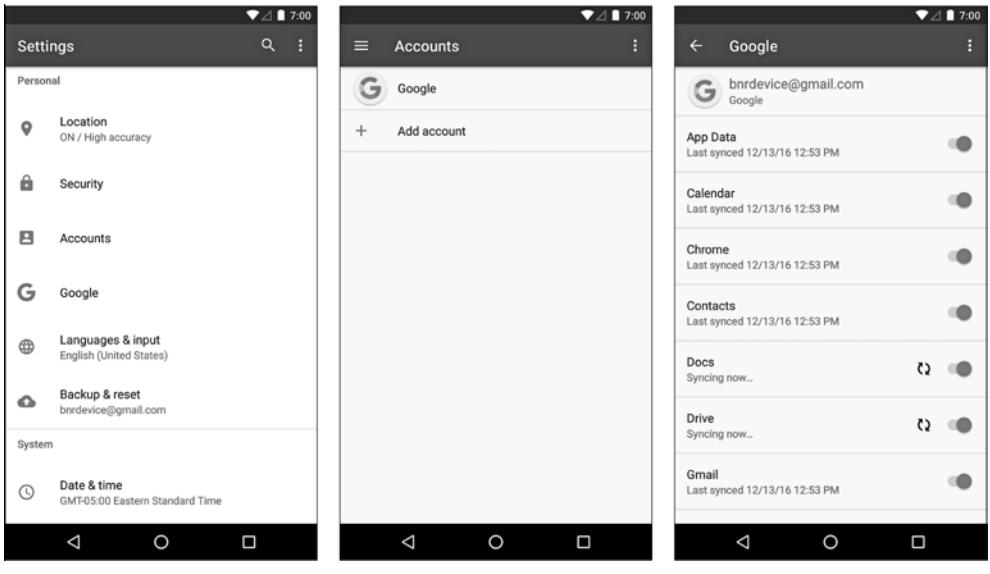


Рис. 28.2. Настройка учетных записей

Итак, если ваше приложение уже использует `ContentProvider` на уровне данных и требует аутентификации учетной записи — рассмотрите возможность использования синхронизирующих адаптеров, это может быть хорошим вариантом. Большим преимуществом такого решения является интеграция синхронизирующих адаптеров с пользовательским интерфейсом, предоставляемым ОС. `JobScheduler` такой возможности не предоставляет. Если ни один из этих факторов не действует, необходимость написания дополнительного кода может не оправдать такое решение.

В электронной документации разработчика имеется учебное руководство по использованию синхронизирующих адаптеров: *developer.android.com/training/sync-adapters/index.html*. В нем вы найдете дополнительную информацию по теме.

29 Широковещательные интенты

В этой главе в приложение PhotoGallery будут внесены два существенных усовершенствования. Во-первых, мы научим приложение проводить опрос наличия новых результатов поиска и оповещать пользователя при их обнаружении (даже если пользователь не открывал приложение с момента загрузки устройства). Во-вторых, оповещения о новых результатах будут отправляться только в том случае, если пользователь не взаимодействует с приложением. (Появление оповещений одновременно с обновлением экрана во время активного просмотра приложения только раздражает.)

При внесении этих обновлений вы узнаете, как прослушивать *широковещательные интенты* от системы и как обрабатывать их при помощи *широковещательных приемников*. Также мы займемся динамической отправкой и получением широковещательных интентов во время выполнения. Наконец, мы используем упорядоченные широковещательные рассылки для определения того, выполняется в настоящий момент приложение на переднем плане или нет.

Обычные и широковещательные интенты

На устройствах Android постоянно что-нибудь происходит. Точки WiFi входят и выходят из зоны приема, устанавливаются пакеты, поступают телефонные звонки и текстовые сообщения.

Если о возникновении некоторого события должны узнать многие компоненты системы, Android использует для распространения информации широковещательные интенты (broadcast intent). Широковещательные интенты работают примерно так же, как уже знакомые вам обычные интенты, не считая того, что их могут получать сразу несколько компонентов, называемых широковещательными приемниками (broadcast receivers) (рис. 29.1).

Активности и службы должны реагировать на неявные интенты, когда они используются как часть открытого API. В других обстоятельствах явных интентов почти всегда хватает. С другой стороны, широковещательные интенты существуют именно для того, чтобы отправлять интенты более чем одному пользователю. Итак, хотя широковещательные приемники *могут* реагировать на явные интенты, они редко используются таким образом, потому что у явных интентов только один получатель.

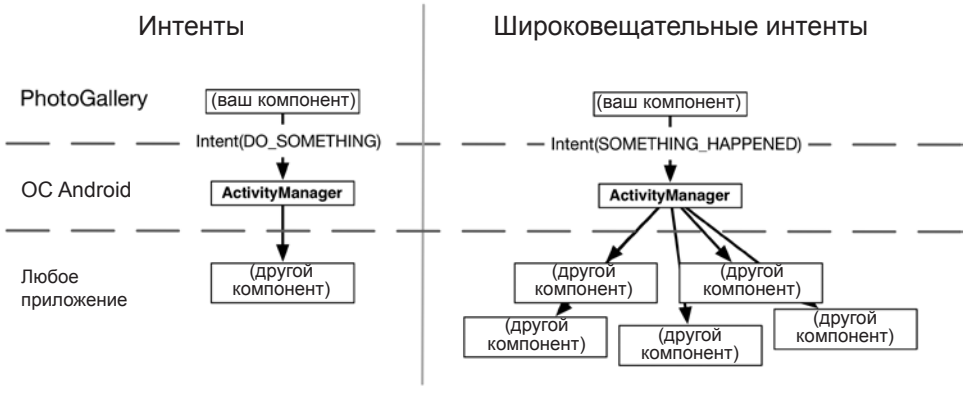


Рис. 29.1. Обычные и широковещательные интенты

Пробуждение при загрузке

Фоновый сигнал PhotoGallery работает, но он не идеален. Если пользователь перезагрузит свой телефон, то сигнал будет потерян.

Приложения, выполняющие продолжительный процесс для пользователя, обычно должны пробуждаться после загрузки устройства. Чтобы узнать о завершении загрузки, следует прослушивать широковещательный интент с действием `BOOT_COMPLETED`. Система отправляет широковещательный интент `BOOT_COMPLETED` при каждом включении устройства. Чтобы прослушивать его, создайте и зарегистрируйте автономный широковещательный приемник, который отфильтровывает подходящее действие.

Создание и регистрация автономного широковещательного приемника

Автономным приемником называется широковещательный приемник, объявленный в манифесте. Такой приемник может активизироваться даже в том случае, если процесс приложения мертв. (Далее вы узнаете о *динамических приемниках*, которые могут быть связаны с жизненным циклом визуального компонента приложения — фрагмента, активности и т. д.)

Подобно службам и активностям, широковещательные приемники должны быть зарегистрированы в системе для выполнения любой полезной работы. Если приемник не зарегистрирован, то система не будет отправлять ему интенты, а метод `onReceive()` приемника не будет выполняться, как ожидалось.

Прежде чем регистрировать широковещательный приемник, его нужно написать. Создайте новый класс Java с именем `StartupReceiver`, являющийся субклассом `android.content.BroadcastReceiver`.

Листинг 29.1. Наш первый широковещательный приемник (StartupReceiver.java)

```
public class StartupReceiver extends BroadcastReceiver {
    private static final String TAG = "StartupReceiver";

    @Override
    public void onReceive(Context context, Intent intent) {
        Log.i(TAG, "Received broadcast intent: " + intent.getAction());
    }
}
```

Широковещательный приемник — компонент, который получает интенты, как и служба или активность. При получении интента экземпляром `StartupReceiver` будет вызван его метод `onReceive(…)`.

Откройте файл AndroidManifest.xml и включите объявление `StartupReceiver` как автономного приемника.

Листинг 29.2. Включение приемника в манифест (AndroidManifest.xml)

```
<manifest ...>

    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />

    <application
        ...>
        <activity
            android:name=".PhotoGalleryActivity"
            android:label="@string/app_name">
            ...
        </activity>
        <service android:name=".PollService"/>

        <receiver android:name=".StartupReceiver">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED"/>
            </intent-filter>
        </receiver>
    </application>

</manifest>
```

Регистрация автономного приемника происходит точно так же, как и регистрация службы или активности: вы используете тег `receiver` с соответствующими фильтрами интентов. `StartupReceiver` будет прослушивать действие `BOOT_COMPLETED`. Для этого действия также требуется разрешение, поэтому в манифест необходимо включить тег `uses-permission`.

После того как широковещательный приемник был объявлен в вашем манифесте, он будет пробуждаться при каждой отправке соответствующего широковещательного интента, даже если ваше приложение в настоящий момент не выполняется. При пробуждении выполняется метод `onReceive(Context,Intent)`

эфемерного широковещательного приемника, после чего он прекращает существование (рис. 29.2).

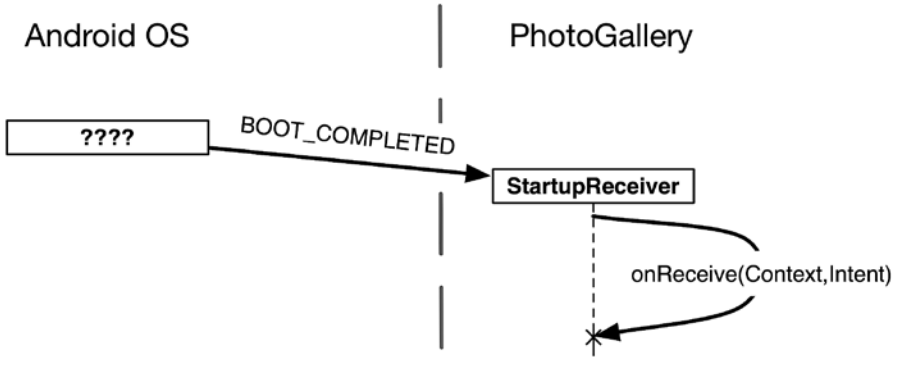


Рис. 29.2. Получение BOOT_COMPLETED

Пора убедиться в том, что метод `onReceive(…)` класса `StartupReceiver` выполняется при загрузке устройства. Для начала запустите приложение PhotoGallery, чтобы установить его новейшую версию на своем устройстве.

Отключите устройство. Если вы используете физическое устройство, полностью выключите питание. При использовании эмулятора проще всего завершить работу эмулятора, закрыв его окно.

Снова включите устройство. На физическом устройстве воспользуйтесь кнопкой питания. В эмуляторе перезапустите приложение или запустите устройство при помощи AVD Manager. Убедитесь в том, что при запуске используется тот же образ эмулятора, который только что использовался при отключении.

Откройте Android Device Monitor командой Tools ▸ Android ▸ Android Device Monitor.

Щелкните на устройстве на вкладке Devices в Android Device Monitor. (Если вы не видите устройство в списке, попробуйте переподключить устройство USB или перезапустить эмулятор.)

Найдите на панели LogCat в окне Android Device Monitor свои данные (рис. 29.3).

Если все сделано правильно, вы увидите строку с сообщением о выполнении приемника. Но если вы проверите состояние своего устройства на вкладке Devices, скорее всего, вы не найдете процесс PhotoGallery. Процесс просуществовал ровно столько, сколько было необходимо для запуска широковещательного приемника, после чего снова «умер».

(Проверка выполнения приемника с LogCat может быть ненадежной, особенно в эмуляторе. Если вы не видите результатов при первом выполнении инструкций, попробуйте повторить попытку несколько раз. Если ничего не получится, продолжайте читать. Когда вы доберетесь до того места, где мы займемся подключением оповещений, у вас появится более надежный способ проверки работоспособности приемника.)

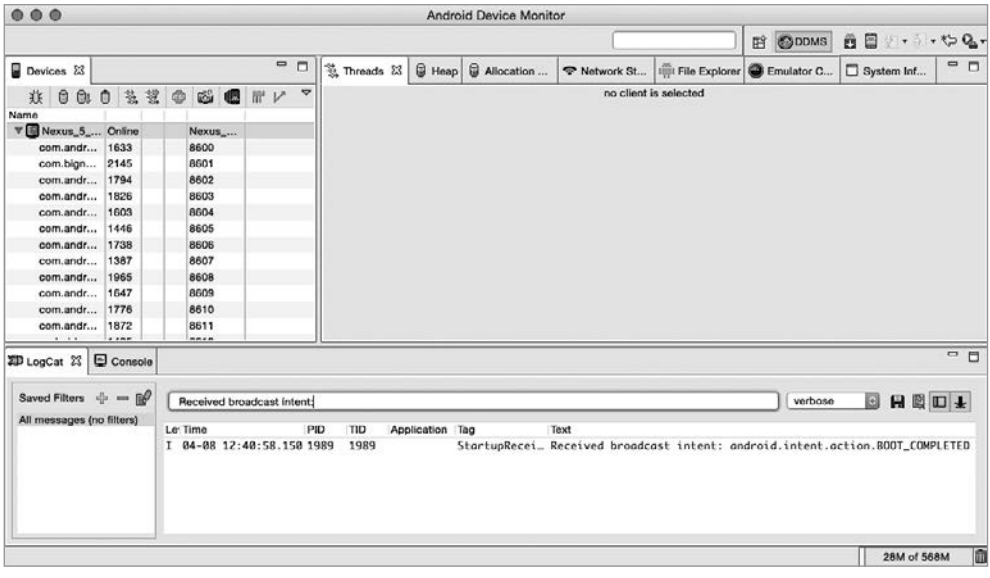


Рис. 29.3. Выходные данные LogCat

Использование приемников

Тот факт, что широковещательные приемники живут такой короткой жизнью, ограничивает возможности их применения. Например, в них нельзя использовать асинхронные API или регистрировать слушателей, потому что срок жизни приемника не превысит продолжительности выполнения `onReceive(Context,Intent)`. Кроме того, `onReceive(Context,Intent)` выполняется в главном потоке, поэтому здесь нельзя выполнять никаких интенсивных вычислений — то есть никаких сетевых операций или серьезной работы с долговременным хранилищем.

Впрочем, это вовсе не значит, что приемники бесполезны. Они чрезвычайно полезны для выполнения всевозможного служебного кода: например, запуска активности или службы (если они не должны возвращать результат) или сброса сигнала при завершении запуска системы (как это будет сделано в нашем упражнении).

Приемник обязан знать, должен сигнал находиться во включенном или в отключенном состоянии. Добавьте в `QueryPreferences` общую настройку, в которой будет храниться эта информация.

Листинг 29.3. Добавление настройки для хранения состояния сигнала (QueryPreferences.java)

```
public class QueryPreferences {
    private static final String PREF_SEARCH_QUERY = "searchQuery";
    private static final String PREF_LAST_RESULT_ID = "lastResultId";
    private static final String PREF_IS_ALARM_ON = "isAlarmOn";
    ...

    public static void setLastResultId(Context context, String lastResultId) {
```

```
        ...
    }

    public static boolean isAlarmOn(Context context) {
        return PreferenceManager.getDefaultSharedPreferences(context)
                .getBoolean(PREF_IS_ALARM_ON, false);
    }

    public static void setAlarmOn(Context context, boolean isOn) {
        PreferenceManager.getDefaultSharedPreferences(context)
                .edit()
                .putBoolean(PREF_IS_ALARM_ON, isOn)
                .apply();
    }
}
```

Затем добавьте в `PollService.setServiceAlarm(…)` запись общей настройки при включенном сигнале.

Листинг 29.4. Запись настройки для хранения состояния сигнала (PollService.java)

```
public class PollService extends IntentService {
    ...
    public static void setServiceAlarm(Context context, boolean isOn) {
        ...
        if (isOn) {
            alarmManager.setRepeating(AlarmManager.ELAPSED_REALTIME,
                    SystemClock.elapsedRealtime(), POLL_INTERVAL_MS, pi);
        } else {
            alarmManager.cancel(pi);
            pi.cancel();
        }

        QueryPreferences.setAlarmOn(context, isOn);
    }
    ...
}
```

После этого `StartupReceiver` может использовать настройку для включения сигнала при загрузке.

Листинг 29.5. Включение сигнала при загрузке (StartupReceiver.java)

```
public class StartupReceiver extends BroadcastReceiver{
    private static final String TAG = "StartupReceiver";

    @Override
    public void onReceive(Context context, Intent intent) {
        Log.i(TAG, "Received broadcast intent: " + intent.getAction());

        boolean isOn = QueryPreferences.isAlarmOn(context);
        PollService.setServiceAlarm(context, isOn);
    }
}
```

Снова запустите приложение PhotoGallery. (Возможно, в ходе тестирования величину интервала `PollService.POLL_INTERVAL` стоит сократить — скажем, до 60 секунд.) Включите опрос кнопкой Start Polling на панели инструментов. Перезагрузите устройство. На этот раз фоновый опрос должен автоматически перезапуститься после перезагрузки телефона, планшета или эмулятора.

Фильтрация оповещений переднего плана

Разобравшись с одним недостатком, мы обращаемся к другому изъяну PhotoGallery. Оповещения прекрасно работают, но они отправляются даже тогда, когда приложение уже открыто.

Эта проблема также решается при помощи широковещательных интентов, но работать они будут совершенно иначе.

Сначала мы будем отправлять (и получать) собственный широковещательный интент (который в конечном итоге будет заблокирован, чтобы его могли получать только компоненты вашего приложения). Затем мы динамически зарегистрируем приемник для широковещательного интента в коде (а не в манифесте). Наконец, мы отправим упорядоченный широковещательный интент, проходящий по цепочке приемников, чтобы некоторый приемник выполнялся последним. (Пока вы еще не знаете, как это делается, но скоро узнаете.)

Отправка широковещательных интентов

Самая простая часть решения: отправка ваших собственных широковещательных интентов. Если говорить конкретнее, вы разошлете широковещательный интент, который уведомляет заинтересованные компоненты о том, что оповещение о новых результатах поиска готово. Чтобы отправить широковещательный интент, просто создайте интент и передайте его `sendBroadcast(Intent)`. В нашем случае широковещательная рассылка будет применяться к определенному нами действию, поэтому также следует определить константу действия.

Добавьте код из следующего листинга в `PollService`.

Листинг 29.6. Отправка широковещательного интента (PollService.java)

```
public class PollService extends IntentService {
    private static final String TAG = "PollService";

    private static final long POLL_INTERVAL_MS = TimeUnit.MINUTES.toMillis(15);

    public static final String ACTION_SHOW_NOTIFICATION =
            "com.bignerdranch.android.photogallery.SHOW_NOTIFICATION";
    ...
    @Override
    protected void onHandleIntent(Intent intent) {
        ...
        String resultId = items.get(0).getId();
        if (resultId.equals(lastResultId)) {
```

```
            Log.i(TAG, "Got an old result: " + resultId);
        } else {
            ...
            NotificationManagerCompat notificationManager =
                    NotificationManagerCompat.from(this);
            notificationManager.notify(0, notification);

            sendBroadcast(new Intent(ACTION_SHOW_NOTIFICATION));
        }

        QueryPreferences.setLastResultId(this, resultId);
    }
    ...
}
```

Теперь приложение будет отправлять широковещательный интент при каждом появлении новых результатов поиска.

Создание и регистрация динамического приемника

Теперь вам понадобится приемник для широковещательного интента `ACTION_SHOW_NOTIFICATION`.

В принципе, для этого можно написать широковещательный приемник, зарегистрированный в манифесте, наподобие `StartupReceiver`, но в нашем случае это не решит проблему. Мы хотим, чтобы класс `PhotoGalleryFragment` получал интент только в то время, пока он живет. Автономный приемник, объявленный в манифесте, не сможет легко справиться с этой задачей. Он всегда будет получать интент, и ему необходимо как-то узнать, что класс `PhotoGalleryFragment` живет (а в Android это сделать не так просто).

Задача решается использованием *динамического широковещательного приемника*. Динамический приемник регистрируется в коде, а не в манифесте. Для регистрации приемника используется вызов `registerReceiver(BroadcastReceiver, IntentFilter)`, а для ее отмены — вызов `unregisterReceiver(BroadcastReceiver)`. Сам приемник обычно определяется как внутренний экземпляр, по аналогии со слушателем щелчка на кнопке. Но поскольку в `registerReceiver(…)` и `unregisterReceiver(…)` должен использоваться один экземпляр, приемник необходимо присвоить переменной экземпляра.

Создайте новый абстрактный класс `VisibleFragment`, суперклассом которого является `Fragment`. Этот класс будет представлять обобщенный фрагмент, скрывающий оповещения переднего плана (другой такой фрагмент мы напишем в главе 30).

Листинг 29.7. Класс VisibleFragment (VisibleFragment.java)

```
public abstract class VisibleFragment extends Fragment {
    private static final String TAG = "VisibleFragment";

    @Override
    public void onStart() {
        super.onStart();
```

```
        IntentFilter filter = new IntentFilter(PollService.ACTION_SHOW_
                              NOTIFICATION);
        getActivity().registerReceiver(mOnShowNotification, filter);
    }

    @Override
    public void onStop() {
        super.onStop();
        getActivity().unregisterReceiver(mOnShowNotification);
    }

    private BroadcastReceiver mOnShowNotification = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            Toast.makeText(getActivity(),
                    "Got a broadcast:" + intent.getAction(),
                    Toast.LENGTH_LONG)
                 .show();
        }
    };
}
```

Обратите внимание: чтобы передать объект `IntentFilter`, необходимо создать его в коде. В данном случае объект `IntentFilter` идентичен фильтру, определяемому следующей разметкой XML:

```
<intent-filter>
    <action android:name="com.bignerdranch.android.photogallery.SHOW_NOTIFICATION" />
</intent-filter>
```

Любой объект `IntentFilter`, который можно выразить в XML, также может быть представлен в коде подобным образом. Просто вызывайте `addCategory(String)`, `addAction(String)`, `addDataPath(String)` и так далее для настройки фильтра.

Динамически регистрируемые широковещательные приемники также должны принять меры для своей деинициализации. Как правило, если вы регистрируете приемник в методе жизненного цикла, вызываемом при запуске, в соответствующем методе завершения вызывается метод `Context.unregisterReceiver(BroadcastReceiver)`. В нашем примере регистрация выполняется в `onResume()` и отменяется в `onStop()`. Аналогичным образом, если бы регистрация выполнялась в `onActivityCreated(…)`, то отменяться она должна была бы в `onActivityDestroyed()`.

(Кстати, будьте осторожны с `onCreate(…)` и `onDestroy()` при удержании фрагментов. Метод `getActivity()` будет возвращать разные значения в `onCreate(…)` и `onDestroy()`, если экран был повернут. Если вы хотите регистрировать/отменять регистрацию в `Fragment.onCreate(Bundle)` и `Fragment.onDestroy()`, используйте `getActivity().getApplicationContext()`.)

Сделайте `PhotoGalleryFragment` субклассом только что созданного класса `VisibleFragment`.

Листинг 29.8. Фрагмент делается видимым (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends ~~Fragment~~ VisibleFragment {
    ...
}
```

Запустите PhotoGallery и пару раз переключите режим фонового опроса. Вы увидите, как наряду с бегущей строкой на экране появится окно сообщения (рис. 29.4).

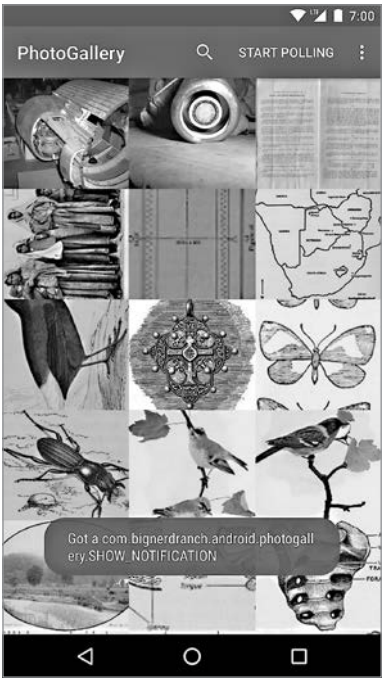


Рис. 29.4. Доказательство существования широковещательной рассылки

Ограничение широковещательной рассылки

Одна из проблем с использованием широковещательной рассылки заключается в том, что любой компонент в системе может прослушивать ее или инициировать ваши приемники. И то и другое обычно нежелательно.

Эти несанкционированные вмешательства в ваши личные дела можно предотвратить парой способов. Если приемник объявлен в манифесте и является внутренним по отношению к вашему приложению, добавьте в тег `receiver` атрибут `android:exported="false"`. С ним приемник становится невидимым для других приложений в системе.

В других ситуациях вы можете создать собственные разрешения, для чего в AndroidManifest.xml включается тег `permission`. Именно этот способ будет использован в PhotoGallery.

Объявите и получите закрытое (private) разрешение в AndroidManifest.xml.

Листинг 29.9. Добавление закрытого разрешения (AndroidManifest.xml)

```
<manifest ...>

    <permission android:name="com.bignerdranch.android.photogallery.PRIVATE"
      android:protectionLevel="signature" />

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    <uses-permission android:name="com.bignerdranch.android.photogallery.PRIVATE" />
      ...
</manifest>
```

В этой разметке мы определяем собственное разрешение с *уровнем защиты* `signature` (вскоре об уровнях защиты будет рассказано более подробно). Само разрешение представляет собой простую строку — как и действия интентов, категории и системные разрешения, использовавшиеся ранее. Разрешение всегда необходимо получить для его использования, даже если вы сами определяете его. Таковы правила.

Обратите внимание на константу, выделенную цветом фона. Эта строка должна присутствовать в трех разных местах, и всюду она должна быть полностью идентичной. Лучше скопируйте ее, вместо того чтобы вводить вручную.

Чтобы использовать разрешение, определите соответствующую константу в коде и передайте ее при вызове `sendBroadcast(…)`.

Листинг 29.10. Отправка с разрешением (PollService.java)

```
public class PollService extends IntentService {
    ...
    public static final String ACTION_SHOW_NOTIFICATION =
            "com.bignerdranch.android.photogallery.SHOW_NOTIFICATION";
    public static final String PERM_PRIVATE =
            "com.bignerdranch.android.photogallery.PRIVATE";

    public static Intent newIntent(Context context) {
        return new Intent(context, PollService.class);
    }
    ...
    @Override
    protected void onHandleIntent(Intent intent) {
        ...
        String resultId = items.get(0).getId();
        if (resultId.equals(lastResultId)) {
            Log.i(TAG, "Got an old result: " + resultId);
        } else {
            ...
```

```
            notificationManager.notify(0, notification);
            sendBroadcast(new Intent(ACTION_SHOW_NOTIFICATION), PERM_PRIVATE);
        }

        QueryPreferences.setLastResultId(this, resultId);
    }
    ...
}
```

Чтобы использовать разрешение, передайте его в параметре `sendBroadcast(…)`. Теперь любое приложение сможет получить ваш интент, только указав то же разрешение, которое было указано при отправке.

Как насчет широковещательного приемника? Другая сторона сможет создать свой широковещательный интент, чтобы заставить ваш приемник сработать. Эта проблема также решается указанием разрешения при вызове `registerReceiver(…)`.

Листинг 29.11. Разрешения для широковещательного приемника (VisibleFragment.java)

```
public abstract class VisibleFragment extends Fragment {
    ...
    @Override
    public void onStart() {
        super.onStart();
        IntentFilter filter = new IntentFilter(PollService.ACTION_SHOW_
NOTIFICATION);
        getActivity().registerReceiver(mOnShowNotification, filter,
                PollService.PERM_PRIVATE, null);
    }
    ...
}
```

Теперь только ваше приложение сможет заставить этот приемник сработать.

Подробнее об уровнях защиты

Каждое пользовательское разрешение должно задавать *уровень защиты* — атрибут `android:protectionLevel`. Уровень защиты сообщает Android, как будет использоваться разрешение. В нашем примере используется уровень защиты `signature`.

Он означает, что если другое приложение захочет использовать ваше разрешение, то оно должно быть снабжено цифровой подписью с таким же ключом, как у вашего приложения. Обычно этот вариант оптимален для разрешений, используемых в вашем приложении. Так как ваш ключ недоступен для других разработчиков, они не могут получить доступ к функциональности, которую защищает ваше разрешение. Вдобавок, поскольку у вас *есть* собственный ключ, вы можете использовать разрешение в других приложениях, которые будут написаны позднее.

Таблица 29.1. Значения атрибута protectionLevel

Значение	Описание
normal	Используется для защиты функциональности приложения, которая не выполняет потенциально опасных операций, например обращений к защищенным личным данным или отправки данных в Интернет. Пользователь видит разрешение перед установкой приложения, но не получает запрос на его явное предоставление. android.permission.RECEIVE_BOOT_COMPLETED использует этот уровень, как и разрешение на вибрацию телефона
dangerous	Используется для всего, для чего не используется normal: для обращений к личным данным, отправки и получения данных из сетевых интерфейсов, обращения к оборудованию, которое может использоваться для шпионских целей, и вообще ко всему, что может создать реальные проблемы для пользователя. В частности, разрешения на доступ к Интернету, камере и контактам относятся к этой категории. Начиная с версии Marshmallow, разрешения dangerous требуют вызова requestPermission(…) во время выполнения для получения от пользователя явного подтверждения на выполнение опасной операции (см. главу 33)
signature	Система предоставляет это разрешение, если приложение подписано тем же сертификатом, что и приложение, в котором объявлено разрешение, или отклоняет его в противном случае. Если разрешение предоставляется, пользователь об этом не оповещается. Значение обычно используется для функциональности, внутренней для вашего приложения: так как у вас имеется сертификат, а приложение может использоваться только приложениями, подписанными тем же сертификатом, вы контролируете состав пользователей разрешения. В нашем случае значение не позволит посторонним видеть вашу широковещательную рассылку, но при желании вы можете написать другое приложение, которое также сможет ее прослушивать
signatureOrSystem	Аналог signature, но разрешение также предоставляется всем пакетам в образе системы Android. Используется для взаимодействия с приложениями, встроенными в образ системы; если разрешение предоставляется, то пользователь не оповещается. Для большинства разработчиков интереса не представляет

Передача и получение данных с упорядоченной широковещательной рассылкой

Пора сделать последний шаг: позаботиться о том, чтобы динамически зарегистрированный получатель всегда получал широковещательный интент `PollService.ACTION_SHOW_NOTIFICATION` до того, как он будет принят другими приемниками, и отменял отправку оповещения.

На данный момент наша собственная широковещательная закрытая рассылка работает, но передача данных осуществляется только в одном направлении (рис. 29.5).

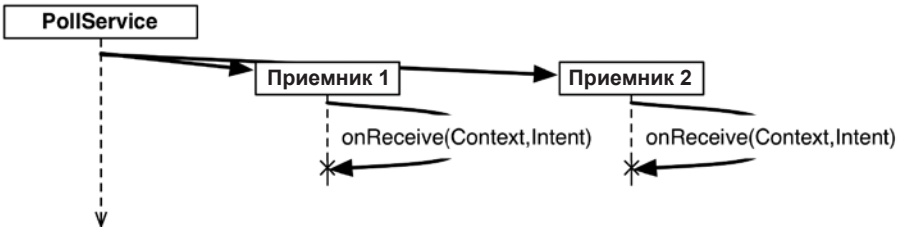


Рис. 29.5. Обычные широковещательные интенты

Это объяснялось тем, что на концептуальном уровне обычный широковещательный интент принимается всеми одновременно. Сейчас `onReceive(…)` вызывается в главном потоке, поэтому на практике приемники не выполняются параллельно. Мы не можем рассчитывать на то, что они будут выполняться в каком-то конкретном порядке, или на то, чтобы узнать, когда все они завершат выполнение. Этот факт значительно затрудняет взаимодействие между широковещательными приемниками или получение информации отправителем интента от приемников.

Двустороннее взаимодействие можно реализовать с использованием *упорядоченных широковещательных интентов* (рис. 27.6). Упорядоченные широковещательные интенты позволяют серии широковещательных приемников обработать широковещательный интент по порядку. Кроме того, отправитель широковещательного интента может получать результаты, передавая при вызове специальный широковещательный приемник, называемый *приемником результата* (result receiver).

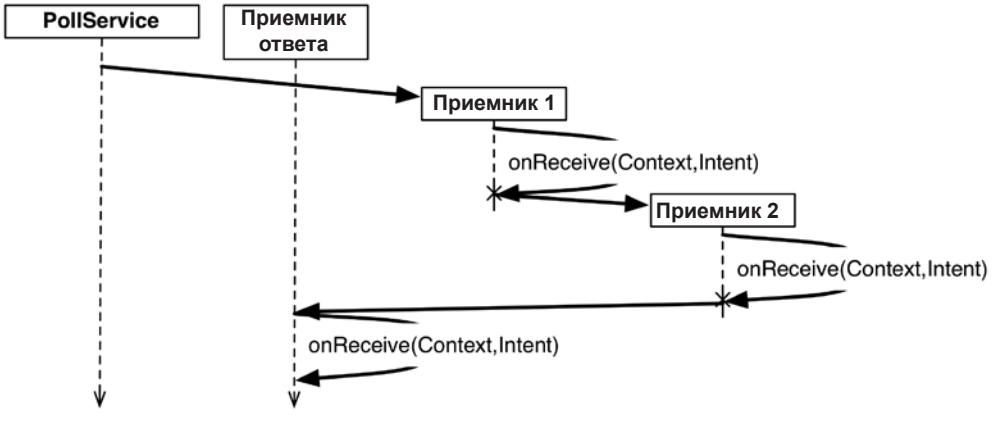


Рис. 29.6. Упорядоченные широковещательные интенты

На получающей стороне все выглядит практически так же, как при обычной широковещательной рассылке. Однако в вашем распоряжении появляется дополнительный инструмент: набор методов, используемых для изменения возвращаемого значения вашего приемника. В нашей ситуации нужно отменить оповещение;

эта информация передается в виде простого целочисленного кода результата. Соответственно, мы используем метод `setResultCode(int)` для назначения кода результата `Activity.RESULT_CANCELED`.

Внесите изменения в `VisibleFragment`, чтобы вернуть информацию отправителю `SHOW_NOTIFICATION`. Информация также будет отправляться другим широковещательным приемникам по цепочке.

Листинг 29.12. Возвращение простого результата (VisibleFragment.java)

```
public abstract class VisibleFragment extends Fragment {
    private static final String TAG = "VisibleFragment";
    ...
    private BroadcastReceiver mOnShowNotification = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            Toast.makeText(getActivity(),
                    "Got a broadcast:" + intent.getAction(),
                    Toast.LENGTH_LONG)
                .show();
            // Получение означает, что пользователь видит приложение,
            // поэтому оповещение отменяется
            Log.i(TAG, "canceling notification");
            setResultCode(Activity.RESULT_CANCELED);
        }
    };
    ...
}
```

Так как в нашем примере необходимо лишь подать сигнал «да/нет», нам достаточно кода результата. Если потребуется вернуть более сложные данные, используйте `setResultData(String)` или `setResultExtras(Bundle)`. А если вы захотите задать все три значения, вызовите `setResult(int,String,Bundle)`. После того как все возвращаемые значения будут заданы, каждый последующий приемник сможет увидеть или изменить их.

Чтобы эти методы делали что-то полезное, широковещательная передача должна быть упорядоченной. Напишите новый метод для отправки упорядоченных широковещательных интентов из `PollService`. Этот метод будет упаковывать обращение к `Notification` и отправлять его в широковещательном режиме. Обновите метод `onHandleIntent(…)`, чтобы он вызывал ваш новый метод и отправлял упорядоченный широковещательный интент, вместо того чтобы отправлять оповещение непосредственно `NotificationManager`.

Листинг 29.13. Отправка упорядоченных широковещательных интентов (PollService.java)

```
public static final String PERM_PRIVATE =
        "com.bignerdranch.android.photogallery.PRIVATE";
public static final String REQUEST_CODE = "REQUEST_CODE";
public static final String NOTIFICATION = "NOTIFICATION";
...
```

```
@Override
protected void onHandleIntent(Intent intent) {
    ...
    String resultId = items.get(0).getId();
    if (resultId.equals(lastResultId)) {
        Log.i(TAG, "Got an old result: " + resultId);
    } else {
        Log.i(TAG, "Got a new result: " + resultId);
        ...

        Notification notification = ...;

        NotificationManagerCompat notificationManager =
                NotificationManagerCompat.from(this);
        notificationManager.notify(0, notification);

        sendBroadcast(new Intent(ACTION_SHOW_NOTIFICATION), PERM_PRIVATE);
        showBackgroundNotification(0, notification);
    }

    QueryPreferences.setLastResultId(this, resultId);
}

private void showBackgroundNotification(int requestCode, Notification notification)
{
    Intent i = new Intent(ACTION_SHOW_NOTIFICATION);
    i.putExtra(REQUEST_CODE, requestCode);
    i.putExtra(NOTIFICATION, notification);
    sendOrderedBroadcast(i, PERM_PRIVATE, null, null,
            Activity.RESULT_OK, null, null);
}
```

Метод `Context.sendOrderedBroadcast(Intent,String,BroadcastReceiver,Handler, int,String,Bundle)` имеет пять дополнительных параметров кроме используемых в `sendBroadcast(Intent,String)`. Вот они, по порядку: *приемник результата*, объект `Handler` для запуска приемника результата, а затем исходные значения кода результата, данных результата и дополнения результата для упорядоченной широковещательной рассылки.

Приемник результата — специальный приемник, который выполняется после всех остальных приемников упорядоченного широковещательного интента. В других обстоятельствах мы смогли бы использовать получателя результата для получения широковещательного интента и отправки объекта оповещения. Однако в данном случае такое решение не сработает. Широковещательный интент часто будет отправляться непосредственно перед прекращением существования `PollService`. Это означает, что и приемник широковещательного интента может быть мертв.

Таким образом, последний приемник должен быть автономным, и вы должны обеспечить его выполнение после динамически зарегистрированного приемника.

Создайте новый субкласс `BroadcastReceiver` с именем `NotificationReceiver`. Реализуйте его следующим образом.

Листинг 29.14. Реализация получателя результата (NotificationReceiver.java)

```
public class NotificationReceiver extends BroadcastReceiver {
    private static final String TAG = "NotificationReceiver";

    @Override
    public void onReceive(Context c, Intent i) {
        Log.i(TAG, "received result: " + getResultCode());
        if (getResultCode() != Activity.RESULT_OK) {
            // Активность переднего плана отменила рассылку
            return;
        }

        int requestCode = i.getIntExtra(PollService.REQUEST_CODE, 0);
        Notification notification = (Notification)
                i.getParcelableExtra(PollService.NOTIFICATION);

        NotificationManagerCompat notificationManager =
                NotificationManagerCompat.from(c);
        notificationManager.notify(requestCode, notification);
    }
}
```

Наконец, зарегистрируйте новый приемник и назначьте ему приоритет. Чтобы `NotificationReceiver` получал рассылку после динамически зарегистрированного приемника (чтобы он мог проверить, нужно ли передавать оповещение `NotificationManager`), ему нужно назначить низкий приоритет –999 (значения –1000 и ниже зарезервированы).

А поскольку приемник используется только во внутренней работе приложения, его не обязательно делать видимым извне. Задайте атрибут `android:exported="false"`, чтобы ограничить доступ к приемнику.

Листинг 29.15. Регистрация приемника оповещений (AndroidManifest.xml)

```
<receiver android:name=".StartupReceiver">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED" />
    </intent-filter>
</receiver>
<receiver android:name=".NotificationReceiver"
          android:exported="false">
    <intent-filter android:priority="-999">
        <action
            android:name="com.bignerdranch.android.photogallery.SHOW_NOTIFICATION"
/>
    </intent-filter>
</receiver>
```

Запустите приложение PhotoGallery и пару раз переключите режим фонового опроса. Вы увидите, что оповещения перестают появляться, когда приложение находится на переднем плане. (Если вы еще не сделали этого ранее, снова задайте значение

PollService.POLL_INTERVAL равным 60 секундам, чтобы вам не пришлось ждать 15 минут для проверки работоспособности оповещений в фоновом режиме.)

Приемники и продолжительные задачи

Что делать, если вы хотите, чтобы широковещательный интент запускал задачу более продолжительную, чем допускают ограничения главного цикла выполнения?

Есть два варианта. Первый — выделить эту работу в службу и запустить ее в широковещательном приемнике. Мы рекомендуем именно этот способ. Служба может обрабатывать запрос столько времени, столько нужно. Она может создать очередь из нескольких запросов и обслуживать их по порядку или обрабатывать запросы так, как считает нужным.

Второй вариант основан на использовании метода BroadcastReceiver.goAsync(). Этот метод возвращает объект BroadcastReceiver.PendingResult, который может использоваться для передачи результата в будущем. Таким образом, вы передаете объект PendingResult экземпляру AsyncTask для выполнения продолжительной работы, а потом отвечаете на широковещательную передачу, вызывая методы PendingResult.

У этого способа есть недостаток: он менее гибок. Вам все равно приходится обрабатывать широковещательную передачу за десять секунд или около того, и в вашем распоряжении меньше архитектурных вариантов, чем при использовании службы.

Конечно, у метода goAsync() есть одно огромное преимущество: в нем можно задавать результаты упорядоченных широковещательных интентов. Если вам нужно именно это, другие решения не подойдут. Только позаботьтесь о том, чтобы выполнение не заняло слишком много времени.

Для любознательных: локальные события

Широковещательные интенты предоставляют возможность глобального распространения информации в системе. А если вы хотите распространить информацию о событии только в границах процесса приложения? Для этого существует хорошая альтернатива — *шина событий* (event bus).

Работа *шины событий* основана на принципе использования общей шины (или потока данных), на которую может подписаться ваше приложение. При отправке события по шине происходит активизация подписавшихся компонентов с выполнением их кода обратного вызова.

Для работы с шиной событий в своих Android-приложениях мы используем стороннюю библиотеку EventBus (разработчик greenrobot). Среди других альтернатив стоит рассмотреть Otto (разработчик Square) — еще одну реализацию шины событий или классы Subject и Observable из библиотеки RxJava для моделирования шины событий.

Android предоставляет локальный механизм отправки широковещательных интентов, который называется `LocalBroadcastManager`. Однако наш опыт показывает, что упоминавшиеся сторонние библиотеки предоставляют более гибкие и удобные API для широковещательных локальных событий.

Использование EventBus

Чтобы использовать EventBus в своих приложениях, необходимо включить в проект зависимость для библиотеки. После создания зависимости вы определяете класс, представляющий событие (чтобы передать дополнительные данные, можно добавить поля в событие):

```
public class NewFriendAddedEvent { }
```

Событие можно отправить по шине практически из любой точки приложения:

```
EventBus eventBus = EventBus.getDefault();
eventBus.post(new NewFriendAddedEvent());
```

Чтобы подписаться на получение событий, приложение сначала регистрируется на прослушивание шины. Часто регистрация и отмена регистрации активностей или фрагментов осуществляется в соответствующих методах жизненного цикла, таких как `onStart(…)` и `onStop(…)`:

```
// В некотором фрагменте или активности...
private EventBus mEventBus;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    mEventBus = EventBus.getDefault();
}

@Override
public void onStart() {
    super.onStart();
    mEventBus.register(this);
}

@Override
public void onStop() {
    super.onStop();
    mEventBus.unregister(this);
}
```

Чтобы указать, что должен сделать подписчик при отправке интересующего его события, реализуйте метод с передачей типа события во входном параметре, и снабдите его аннотацией `@Subscribe`. Использование аннотации `@Subscribe` без параметров означает, что событие будет обрабатываться в том же потоке, из ко-

торого оно было отправлено. (Запись `@Subscribe(threadMode = ThreadMode.MAIN` гарантирует, что событие, отправленное из фонового потока, будет обработано в главном потоке приложения.)

```
// В некотором зарегистрированном компоненте -
// например, фрагменте или активности...
@Subscribe(threadMode = ThreadMode.MAIN)
public void onNewFriendAdded(NewFriendAddedEvent event){
    Friend newFriend = event.getFriend();
    // Обновить пользовательский интерфейс или сделать что-то
    // в ответ на событие...
}
```

Использование RxJava

Для реализации механизма широковещательной рассылки событий может использоваться RxJava — библиотека написания кода Java в «реактивном» стиле. Концепция «реактивного» кода широка и выходит за рамки излагаемого материала. В двух словах: она позволяет публиковать серии событий и подписываться на них, а также предоставляет широкий инструментарий общих инструментов для выполнения операций с сериями событий.

Разработчик создает `Subject` — объект, к которому можно обращаться с запросами на публикацию событий, а также с запросами на подписку:

```
Subject<Object, Object> eventBus =
    new SerializedSubject<>(PublishSubject.create());
```

Этот объект используется для публикации событий:

```
Friend someNewFriend = ...;
NewFriendAddedEvent event = new NewFriendAddedEvent(someNewFriend);
eventBus.onNext(event);
```

Пример подписки на события:

```
eventBus.subscribe(new Action1<Object>() {
    @Override
    public void call(Object event) {
        if (event instanceof NewFriendAddedEvent) {
            Friend newFriend = ((NewFriendAddedEvent)event).getFriend();
            // Обновление пользовательского интерфейса
        }
    }
})
```

Преимущество решений на базе RxJava состоит в том, что объект `eventBus` также является реализацией `Observable`, представлением потока событий в RxJava. Это означает, что в вашем распоряжении оказываются все средства для работы с событиями в RxJava. Если вас заинтересует эта тема, обращайтесь к вики на странице проекта RxJava: *github.com/ReactiveX/RxJava/wiki*.

Для любознательных: проверка видимости фрагмента

Немного поразмыслив над реализацией PhotoGallery, мы видим, что глобальный механизм широковещательной рассылки использовался для рассылки интента `SHOW_NOTIFICATION`. При этом получение рассылки при помощи разрешений ограничивается элементами, локальными для вашего приложения. Возникает естественный вопрос: «Почему я использую глобальный механизм, если я просто передаю данные в своем приложении? Разве не логичнее использовать локальный механизм?»

Дело в том, что мы намеренно пытались решить задачу проверки того, виден фрагмент `PhotoGalleryFragment` или нет. Для достижения цели мы воспользовались комбинацией упорядоченных рассылок, автономных приемников и динамически регистрируемых приемников. В Android существует и более прямолинейное решение этой задачи.

Если говорить конкретно, `LocalBroadcastManager` не подходит для широковещательных оповещений `PhotoGallery` и проверки видимости фрагментов по двум причинам.

Во-первых, `LocalBroadcastManager` не поддерживает упорядоченные широковещательные рассылки (хотя и предоставляет механизм широковещательной рассылки с блокировкой, а именно `sendBroadcastSync(Intent intent)`). Он не подойдет для PhotoGallery, потому что приемник `NotificationReceiver` должен гарантированно выполняться последним в цепочке.

Во-вторых, `sendBroadcastSync(Intent intent)` не поддерживает отправку и получение широковещательных интентов в разных потоках. В PhotoGallery интент должен отправляться из фонового потока (в `PollService.onHandleIntent(…)`), а приниматься в главном потоке (динамическим приемником, зарегистрированным `PhotoGalleryFragment`, в главном потоке в `onResume(…)`).

На момент написания книги семантика потоковой доставки `LocalBroadcastManager` была плохо документирована, и по нашему опыту, ей не хватало логичности. Например, при вызове `sendBroadcastSync(…)` из фонового потока все необработанные рассылки будут выданы в фоновый поток независимо от того, были ли они отправлены из главного потока.

Это не значит, что механизм `LocalBroadcastManager` бесполезен. Просто этот инструмент не подходит для задач, которыми мы занимались в этой главе.

30 Просмотр веб-страниц и WebView

С каждой фотографией, загружаемой с Flickr, связана страница. В этой главе мы сделаем так, чтобы пользователь мог нажать на фотографии в PhotoGallery и просмотреть ее страницу. Вы освоите два разных способа интеграции веб-контента в ваши приложения, представленные на рис. 30.1. Первый способ работает с браузером, установленным на устройстве (слева), а второй использует класс `WebView` для отображения веб-контента (справа).

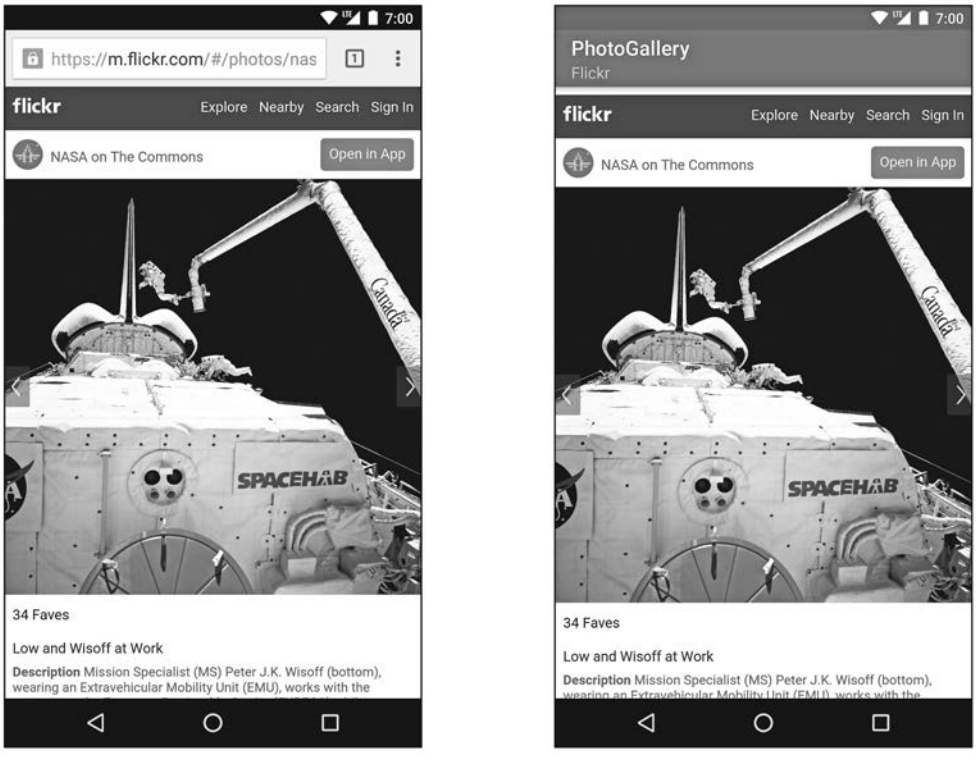


Рис. 30.1. Веб-контент: два разных подхода

И еще один блок данных Flickr

Для обоих способов нам понадобится URL-адрес страницы фотографии на Flickr. Если присмотреться к разметке JSON, которую мы в настоящее время получаем для каждой фотографии, становится ясно, что страница в эти результаты не включена.

```
{
  "photos": {
    ...,
    "photo": [
      {
        "id": "9452133594",
        "owner": "44494372@N05",
        "secret": "d6d20af93e",
        "server": "7365",
        "farm": 8,
        "title": "Low and Wisoff at Work",
        "ispublic": 1,
        "isfriend": 0,
        "isfamily": 0,
        "url_s":"https://farm8.staticflickr.com/7365/9452133594_d6d20af93e_m.jpg"
      }, ...
    ]
  },
  "stat": "ok"
}
```

Неужели придется писать новые запросы JSON? К счастью, это не так. Обратившись к странице документации Flickr по адресу *www.flickr.com/services/api/misc.urls.html*, мы видим, что URL-адреса страниц отдельных фотографий строятся по схеме:

```
http://www.flickr.com/photos/идентификатор-пользователя/идентификатор-фото
```

Идентификатор фотографии совпадает со значением атрибута `photo_id` в разметке JSON. Мы уже сохранили его в поле `mId` объекта `GalleryItem`. Как насчет идентификатора пользователя? Немного покопавшись в документации, мы находим, что атрибут owner в JSON содержит идентификатор пользователя. Таким образом, извлекая атрибут `owner`, мы можем построить URL-адрес по атрибутам из JSON фотографии:

```
http://www.flickr.com/photos/владелец/идентификатор
```

Чтобы реализовать этот план, включите следующий код в `GalleryItem`.

Листинг 30.1. Добавление кода построения URL страницы фотографии (GalleryItem.java)

```
public class GalleryItem {
    private String mCaption;
    private String mId;
```

```
    private String mUrl;
    private String mOwner;
    ...
    public void setUrl(String url) {
        mUrl = url;
    }

    public String getOwner() {
        return mOwner;
    }

    public void setOwner(String owner) {
        mOwner = owner;
    }

    public Uri getPhotoPageUri() {
        return Uri.parse("https://www.flickr.com/photos/")
                .buildUpon()
                .appendPath(mOwner)
                .appendPath(mId)
                .build();
    }

    @Override
    public String toString() {
        return mCaption;
    }
}
```

Здесь мы создаем новое свойство `mOwner` и добавляем короткий метод с именем `getPhotoPageUri()` для построения URL-адреса страницы способом, описанным выше.

Теперь изменим метод `parseItems(…)` для чтения атрибута `owner`.

Листинг 30.2. Чтение атрибута owner (FlickrFetchr.java)

```
public class FlickrFetchr {
    ...
    private void parseItems(List<GalleryItem> items, JSONObject jsonBody)
            throws IOException, JSONException {

        JSONObject photosJsonObject = jsonBody.getJSONObject("photos");
        JSONArray photoJsonArray = photosJsonObject.getJSONArray("photo");

        for (int i = 0; i < photoJsonArray.length(); i++) {
            JSONObject photoJsonObject = photoJsonArray.getJSONObject(i);

            GalleryItem item = new GalleryItem();
            item.setId(photoJsonObject.getString("id"));
            item.setCaption(photoJsonObject.getString("title"));

            if (!photoJsonObject.has("url_s")) {
                continue;
            }
```

```
            item.setUrl(photoJsonObject.getString("url_s"));
            item.setOwner(photoJsonObject.getString("owner"));
            items.add(item);
        }
    }
}
```

Проще простого! Теперь можно поразвлечься с URL-адресом новой страницы.

Простой способ: неявные интенты

Сначала мы откроем страницу по этому URL-адресу при помощи старого знакомого — неявного интента. Этот интент запустит браузер с URL-адресом страницы фотографии.

Для начала нужно организовать прослушивание нажатий на элементах представления `RecyclerView`. Обновите реализацию `PhotoHolder` из `PhotoGalleryFragment` и включите в нее слушателя щелчков, который будет выдавать неявный интент.

Листинг 30.3. Выдача неявных интентов при нажатии (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends VisibleFragment {
    ...
    private class PhotoHolder extends RecyclerView.ViewHolder
             implements View.OnClickListener {
        private ImageView mItemImageView;
        private GalleryItem mGalleryItem;

        public PhotoHolder(View itemView) {
            super(itemView);

            mItemImageView = (ImageView) itemView.findViewById(R.id.item_image_
                              view);
            itemView.setOnClickListener(this);
        }

        public void bindDrawable(Drawable drawable) {
            mItemImageView.setImageDrawable(drawable);
        }

        public void bindGalleryItem(GalleryItem galleryItem) {
            mGalleryItem = galleryItem;
        }

        @Override
        public void onClick(View v) {
            Intent i = new Intent(Intent.ACTION_VIEW, mGalleryItem.
                       getPhotoPageUri());
            startActivity(i);
        }
    }
    ...
}
```

Затем свяжите `PhotoHolder` с `GalleryItem` в `PhotoAdapter.onBindViewHolder(…)`.

Листинг 30.4. Связывание GalleryItem (PhotoGalleryFragment.java)

```
private class PhotoAdapter extends RecyclerView.Adapter<PhotoHolder> {
    ...
    @Override
    public void onBindViewHolder(PhotoHolder photoHolder, int position) {
        GalleryItem galleryItem = mGalleryItems.get(position);
        photoHolder.bindGalleryItem(galleryItem);
        Drawable placeholder = getResources().getDrawable(R.drawable.bill_
                               up_close);
        photoHolder.bindDrawable(placeholder);
        mThumbnailDownloader.queueThumbnail(photoHolder, galleryItem.getUrl());
    }
    ...
}
```

Запустите приложение PhotoGallery и нажмите на фотографии. На экране открывается браузер, в котором загружается страница выбранной вами фотографии (как в левой части рис. 30.1).

Более сложный способ: WebView

Решение с использованием неявного интента для отображения веб-страницы просто и эффективно. Но что, если вы не хотите, чтобы ваше приложение открывало браузер?

На практике веб-контент чаще требуется отобразить прямо в активности вашего приложения. Допустим, вы хотите отобразить самостоятельно сгенерированную разметку HTML или просто обойтись без браузера. Справочная документация в приложениях часто реализуется в виде веб-страницы, чтобы ее было удобнее обновлять. Запуск браузера для просмотра справочных страниц выглядит непрофессионально и препятствует изменению поведения, а также интеграции веб-страницы в ваш пользовательский интерфейс.

Для представления веб-контента в пользовательском интерфейсе приложения используется класс `WebView`. Мы назвали этот способ «более сложным», но на самом деле он очень прост (хотя по сравнению с неявными интентами все можно назвать сложным).

Нашим первым шагом станет создание новой активности и фрагмента для отображения `WebView`. Начнем, как обычно, с определения файла макета; присвойте ему имя fragment_photo_page.xml. Назначьте `ConstraintLayout` макетом верхнего уровня. В визуальном конструкторе перетащите `WebView` на `ConstraintLayout` как дочернее представление. (Виджет `WebView` находится в разделе Containers.)

После добавления `WebView` добавьте к родителю ограничения для всех сторон:

- от верхней стороны `WebView` к верхней стороне родителя;
- от нижней стороны `WebView` к нижней стороне родителя;

- от левой стороны `WebView` к левой стороне родителя;
- от правой стороны `WebView` к правой стороне родителя.

Наконец, задайте высоте и ширине значения Any Size и уменьшите величину всех полей до 0. Также присвойте `WebView` идентификатор `web_view`.

Возможно, у вас возникла мысль: «От `RelativeLayout` нет никакой пользы», — и верно. Однако позднее в этой главе мы наполним его дополнительным «хромом».

Теперь займитесь настройкой фрагмента. Создайте `PhotoPageFragment` как субкласс класса `VisibleFragment`, созданного в предыдущей главе. Необходимо заполнить файл макета, выделить из него `WebView` и передать URL-адрес как аргумент фрагмента.

Листинг 30.5. Создание фрагмента браузера (PhotoPageFragment.java)

```
public class PhotoPageFragment extends VisibleFragment {
    private static final String ARG_URI = "photo_page_url";

    private Uri mUri;
    private WebView mWebView;

    public static PhotoPageFragment newInstance(Uri uri) {
        Bundle args = new Bundle();
        args.putParcelable(ARG_URI, uri);
        PhotoPageFragment fragment = new PhotoPageFragment();
        fragment.setArguments(args);
        return fragment;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mUri = getArguments().getParcelable(ARG_URI);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
                savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_photo_page, container, false);
        mWebView = (WebView) v.findViewById(R.id.web_view);
        return v;
    }
}
```

Пока это всего лишь заготовка — вскоре мы заполним ее кодом. А пока создайте класс-контейнер `PhotoPageActivity` на основе хорошо знакомого класса `SingleFragmentActivity`.

Листинг 30.6. Создание веб-активности (PhotoPageActivity.java)

```
public class PhotoPageActivity extends SingleFragmentActivity {

    public static Intent newIntent(Context context, Uri photoPageUri) {
        Intent i = new Intent(context, PhotoPageActivity.class);
```

```
        i.setData(photoPageUri);
        return i;
    }

    @Override
    protected Fragment createFragment() {
        return PhotoPageFragment.newInstance(getIntent().getData());
    }
}
```

Измените код `PhotoGalleryFragment`, чтобы вместо неявного интента запускалась новая активность.

Листинг 30.7. Переключение на запуск новой активности (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends VisibleFragment {
    ...
    private class PhotoHolder extends RecyclerView.ViewHolder
            implements View.OnClickListener{
        ...
        @Override
        public void onClick(View v) {
            Intent i = new Intent(Intent.ACTION_VIEW, mGalleryItem.
                    getPhotoPageUri());
             Intent i = PhotoPageActivity
                 .newIntent(getActivity(), mGalleryItem.getPhotoPageUri());
            startActivity(i);
        }
    }
    ...
}
```

Наконец, добавьте новую активность в манифест.

Листинг 30.8. Добавление активности в манифест (AndroidManifest.xml)

```
<manifest ... >
    ...
    <application
        ...>
        <activity
            android:name=".PhotoGalleryActivity"
            android:label="@string/app_name" >
            ...
        </activity>

        <activity
            android:name=".PhotoPageActivity" />

        <service android:name=".PollService" />
        ...
    </application>

</manifest>
```

Запустите приложение PhotoGallery и нажмите на фотографии. На экране появится новая пустая активность.

Возьмемся за дело и заставим наш фрагмент делать что-то полезное. Чтобы виджет `WebView` успешно отображал страницу фотографии на сайте Flickr, необходимо выполнить три условия. Первое условие очевидно — нужно сообщить ему, какой URL-адрес необходимо загрузить.

Второе условие — необходимо включить поддержку JavaScript. По умолчанию она отключена. Постоянно держать ее включенной не обязательно, но для Flickr она нужна. (Android Lint выдает предупреждение из-за потенциальной опасности межсайтовых сценарных атак, так что предупреждения Lint тоже нужно отключить — для этого следует пометить `onCreateView(…)` аннотацией `@SuppressLint("SetJavaScriptEnabled")`.)

Наконец, необходимо предоставить реализацию интерфейса `WebViewClient`, используемого при визуализации событий виджетом `WebView`. Мы рассмотрим этот класс после того, как вы введете код.

Листинг 30.9. Загрузка URL в WebView (PhotoPageFragment.java)

```
public class PhotoPageFragment extends VisibleFragment {
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_photo_page, container, false);

        mWebView = (WebView) v.findViewById(R.id.web_view);
        mWebView.getSettings().setJavaScriptEnabled(true);
        mWebView.setWebViewClient(new WebViewClient());
        mWebView.loadUrl(mUri.toString());

        return v;
    }
}
```

Загрузка данных с URL-адреса должна происходить после настройки `WebView`, поэтому она выполняется в последнюю очередь. До этого мы включаем JavaScript, вызывая `getSettings()` для получения экземпляра `WebSettings`, с последующим вызовом `WebSettings.setJavaScriptEnabled(true)`. Объект `WebSettings` — первый из трех механизмов настройки `WebView`. Он содержит различные свойства, которые можно задать в коде, например строку пользовательского агента и размер текста.

После этого реализация `WebViewClient` добавляется к `WebView`. Чтобы понять, для чего это нужно, сначала посмотрим, что произойдет без `WebViewClient`.

Новый URL-адрес может загружаться парой разных способов: страница может приказать вам перейти по другому URL-адресу (перенаправление), или же вы можете щелкнуть на ссылке. Без `WebViewClient` виджет `WebView` предложит менеджеру активностей найти подходящую активность для загрузки нового URL.

Но это совсем не то, что нам нужно. Многие сайты (включая страницы с фотографиями Flickr) при загрузке в браузере на телефоне немедленно перенаправляют пользователя на мобильную версию того же сайта. Нет особого смысла создавать собственное представление страницы, если все равно все кончится инициированием неявного интента.

С другой стороны, если вы предоставите собственную реализацию `WebViewClient` для своего виджета `WebView`, процесс выглядит иначе. Вместо того чтобы обращаться за помощью к менеджеру активностей, виджет обращается к вашей реализации `WebViewClient`. А реализация `WebViewClient` по умолчанию говорит: «Загрузи URL самостоятельно!» И страница появится в вашем виджете `WebView`.

Запустите приложение PhotoGallery, и вы увидите `WebView` на экране (как справа на рис. 30.1).

Класс WebChromeClient

Раз уж мы занялись созданием собственной реализации `WebView`, давайте немного украсим ее, добавив представление заголовка и индикатор прогресса. Снова откройте файл fragment_photo_page.xml.

Перетащите виджет `ProgressBar` на `ConstraintLayout` (как второй дочерний виджет.) Используйте горизонтальную версию индикатора прогресса ProgressBar (Horizontal). Удалите верхнее ограничение `WebView` и назначьте виджету фиксированную высоту (Fixed), чтобы вам было удобнее работать с маркерами ограничений.

После того как это будет сделано, создайте следующие дополнительные ограничения:

- от `ProgressBar` к верхней, правой и левой стороне его родителя;
- от верхней стороны `WebView` к нижней стороне `ProgressBar`.

Затем верните высоте `WebView` значение Any Size, задайте высоте `ProgressBar` значение `wrap_content` и измените ширину `ProgressBar` на Any Size.

Наконец, выделите `ProgressBar` и обратитесь к окну свойств. Задайте свойству visibility значение gone и tools visibility — значение visible. Присвойте виджету идентификатор progress_bar.

Примерный результат показан на рис. 30.2.

Чтобы добавить `ProgressBar`, нам понадобится вторая точка обратного вызова `WebView: WebChromeClient`. Если `WebViewClient` определяет интерфейс обработки событий визуализации, `WebChromeClient` определяет событийный интерфейс обработки событий, которые должны изменять элементы «*хрома*» (chrome) в браузере. К этой категории относятся сигналы (alerts) JavaScript, значки сайтов favicon, обновления прогресса загрузки и т. д.

Подключите его в методе `onCreateView(…)`.

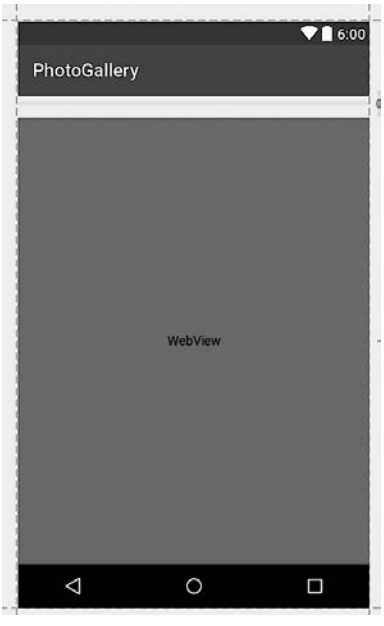


Рис. 30.2. Добавление индикатора прогресса

Листинг 30.10. Использование WebChromeClient (PhotoPageFragment.java)

```
public class PhotoPageFragment extends VisibleFragment {
    ...
    private WebView mWebView;
    private ProgressBar mProgressBar;
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_photo_page, container, false);

        mProgressBar = (ProgressBar)v.findViewById(R.id.progress_bar);
        mProgressBar.setMax(100); // Значения в диапазоне 0-100

        mWebView = (WebView) v.findViewById(R.id.web_view);
        mWebView.getSettings().setJavaScriptEnabled(true);
        mWebView.setWebChromeClient(new WebChromeClient() {
            public void onProgressChanged(WebView webView, int newProgress) {
                if (newProgress == 100) {
                    mProgressBar.setVisibility(View.GONE);
                } else {
                    mProgressBar.setVisibility(View.VISIBLE);
                    mProgressBar.setProgress(newProgress);
                }
            }
```

```
            public void onReceivedTitle(WebView webView, String title) {
                AppCompatActivity activity = (AppCompatActivity) getActivity();
                activity.getSupportActionBar().setSubtitle(title);
            }
        });
        mWebView.setWebViewClient(new WebViewClient());
        mWebView.loadUrl(mUri.toString());

        return v;
    }
}
```

Обновления индикатора прогресса и заголовка имеют собственные методы обратного вызова, `onProgressChanged(WebView,int)` и `onReceivedTitle(WebView, String)`. Информация о прогрессе, получаемая от `onProgressChanged(WebView,int)`, представляет собой целое число от 0 до 100. Если результат равен 100, значит, загрузка страницы завершена, поэтому мы скрываем `ProgressBar`, задавая режим `View.GONE`.

Запустите приложение PhotoGallery и протестируйте внесенные изменения. Результат должен выглядеть так, как показано на рис. 30.3.

При нажатии на фотографии открывается `PhotoPageActivity`. Индикатор отображает информацию о ходе загрузки страниц, а на панели инструментов появляется подзаголовок с текстом, полученным в `onReceivedTitle(…)`. После завершения загрузки индикатор прогресса исчезает.

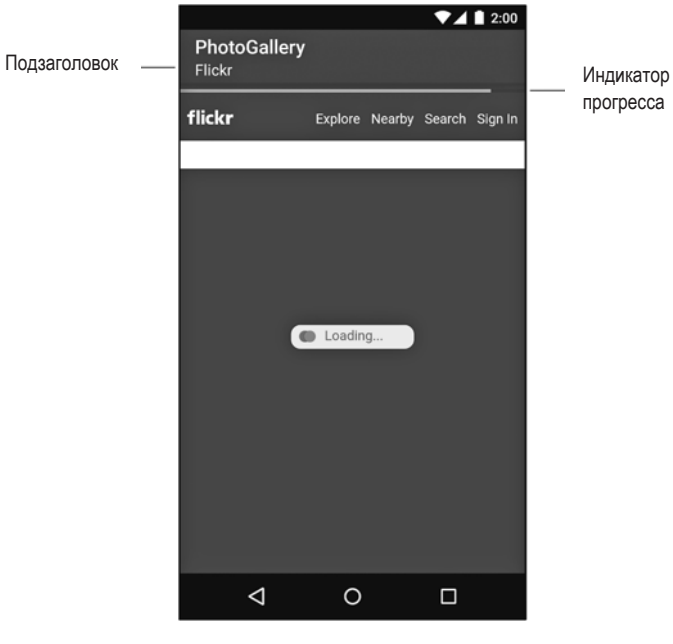


Рис. 30.3. WebView в приложении

Повороты в WebView

Попробуйте повернуть экран. Хотя приложение работает правильно, вы заметите, что `WebView` полностью перезагружает веб-страницу. Дело в том, что у `WebView` слишком много данных, чтобы сохранить их все в `onSaveInstanceState(…)`, и их приходится собирать с нуля каждый раз, когда их приходится создавать заново при повороте.

Может показаться, что проблема проще всего решается удержанием `PhotoPageFragment`. Тем не менее это решение не работает, потому что `WebView` является частью иерархии представлений и поэтому все равно уничтожается и создается заново при повороте.

Для таких классов (другой пример — `VideoView`) документация Android рекомендует позволить активности самой обработать все изменения конфигурации. Это означает, что вместо уничтожения активности она просто перемещает свои представления для размещения по новым размерам экрана. В результате `WebView` не приходится заново загружать свои данные.

Чтобы заставить класс `PhotoPageActivity` обрабатывать свои изменения конфигурации, внесите следующие изменения в AndroidManifest.xml.

Листинг 30.11. Самостоятельный обработчик изменений конфигурации (AndroidManifest.xml)

```
<manifest ... >
    ...
    <activity
          android:name=".PhotoPageActivity"
          android:configChanges="keyboardHidden|orientation|screenSize" />
      ...
</manifest>
```

Атрибут сообщает, что в случае изменения конфигурации из-за открытия или закрытия клавиатуры, изменения ориентации или размеров экрана (которое также происходит при переключении между книжной и альбомной ориентацией в Android после версии 3.2) активность должна обрабатывать изменения самостоятельно.

Вот и все. Попробуйте снова повернуть устройство; на этот раз все должно быть в ажуре.

Опасности при обработке изменений конфигурации

Решение получается настолько простым и эффективным, что у вас может возникнуть вопрос: почему бы не делать так всегда? Ведь жизнь разработчика стала бы намного проще. Тем не менее самостоятельная обработка конфигурации — опасная привычка.

Во-первых, квалификаторы ресурсов перестают работать автоматически, и вам придется перезагружать представление вручную. Это сложнее, чем может показаться.

Во-вторых, при обработке изменений конфигурации в активностях вы, скорее всего, не станете возиться с переопределением `Activity.onSavedInstanceState(…)`

для сохранения временных состояний пользовательского интерфейса. Однако это все равно необходимо, даже если активность обрабатывает изменения конфигурации самостоятельно, потому что возможность уничтожения и повторного создания при нехватке памяти все равно остается. (Помните: активность может быть уничтожена системой в любой момент, когда она не находится в состоянии выполнения (см. рис. 3.14).)

Для любознательных: внедрение объектов JavaScript

Вы уже видели, как следует использовать `WebViewClient` и `WebChromeClient` для обработки некоторых событий, происходящих в `WebView`. Однако еще больше возможностей открывает внедрение произвольных объектов JavaScript в документ, содержащийся в виджете `WebView`. Откройте документацию по адресу *developer.android.com/reference/android/webkit/WebView.html* и прокрутите до описания метода `addJavascriptInterface(Object, String)`. Этот метод позволяет внедрить в документ произвольный объект с заданным именем.

```
mWebView.addJavascriptInterface(new Object() {
    @JavascriptInterface
    public void send(String message) {
        Log.i(TAG, "Received message: " + message);
    }
}, "androidObject");
```

После этого объект используется следующим образом:

```
<input type="button" value="In WebView!"
    onClick="sendToAndroid('In Android land')" />

<script type="text/javascript">
    function sendToAndroid(message) {
        androidObject.send(message);
    }
</script>
```

В этом коде есть пара нетривиальных моментов. Во-первых, при вызове `send(String)` метод Java не вызывается в основном потоке. Он вызывается в потоке, принадлежащем `WebView`. Итак, если вы хотите обновить пользовательский интерфейс Android, вам придется использовать `Handler` для возврата управления в основной поток.

Кроме того, набор поддерживаемых типов данных сильно ограничен. В вашем распоряжении `String`, основные примитивные типы... и все. Любой более сложный тип должен передаваться через `String`, обычно с преобразованием в JSON перед отправкой и последующим разбором при получении.

Начиная с API 17 (Jelly Bean 4.2) в JavaScript экспортируются только открытые методы с пометкой `@JavascriptInterface`. До этого были доступны все открытые методы в иерархии объектов.

В любом случае такое решение весьма рискованно — фактически вы разрешаете потенциально небезопасной веб-странице вмешиваться в работу вашей программы. Следовательно, его стоит применять только для принадлежащей вам разметки HTML или ограничиться предоставлением в высшей степени консервативного интерфейса.

Для любознательных: переработка WebView в KitKat

С выходом версии KitKat (Android 4.4, API 19) компонент `WebView` подвергся значительной переработке. Новая реализация `WebView` основана на проекте с открытым кодом Chromium. В ней используется то же ядро визуализации, что и в приложении Chrome для Android; это означает большее сходство внешнего вида и поведения страниц. (Впрочем, `WebView` не обладает всей функциональностью Chrome для Android. Хорошая сравнительная таблица доступна по адресу *developer.chrome.com/multidevice/webview/overview*.)

Переход на Chromium означал целый ряд интересных усовершенствований `WebView`, включая переход на новые веб-стандарты (такие, как HTML5 и CSS3), обновленное ядро JavaScript и улучшение быстродействия. С точки зрения разработчика, одним из самых интересных новшеств является поддержка удаленной отладки `WebView` с использованием Chrome DevTools (которая включается вызовом `WebView.setWebContentsDebuggingEnabled()`).

Начиная с Lollipop (Android 5.0), Chromium-уровень `WebView` автоматически обновляется из магазина Google Play. Пользователям уже не приходится ждать новых выпусков Android для получения обновлений безопасности (и новой функциональности.) Это значительное достижение.

В еще более поздних версиях, с версии Nougat (Android 7.0) Chromium-уровень `WebView` размещается непосредственно в APK-файле Chrome, что приводит к снижению затрат памяти и ресурсов. Приятно сознавать, что Google следит за своевременным обновлением компонентов `WebView`.

Упражнение. Использование кнопки Back для работы с историей просмотра

Возможно, вы заметили, что после запуска `PhotoPageActivity` вы можете перемещаться по другим ссылкам в `WebView`. Однако сколько бы ссылок вы ни открывали, кнопка Back всегда возвращает вас прямо к `PhotoGalleryActivity`. А если вы хотите, чтобы кнопка Back работала с историей просмотра в `WebView`?

Реализуйте это поведение переопределением метода `Activity.onBackPressed()`. В этом методе для выполнения правильной операции используется комбинация методов `WebView` для работы с историей просмотра (`WebView.canGoBack()` и `WebView.goBack()`). Если история просмотра `WebView` не пуста, вернитесь к предыдущему элементу. В противном случае кнопка Back должна работать как обычно, — вызовите `super.onBackPressed()`.

Упражнение. Поддержка других ссылок

В ходе экспериментов с `WebView` в `PhotoPageFragment` вам могут попасться ссылки с протоколами, отличными от HTTP. Например, на момент написания книги на странице с подробной информацией Flickr отображается кнопка Open an App. Предполагается, что кнопка запустит приложение Flickr, если оно установлено в системе. Если приложение не установлено, открывается магазин Google Play, из которого это приложение можно установить.

Однако при нажатии кнопки Open in App виджет `WebView` выдает сообщение об ошибке (рис. 30.4).

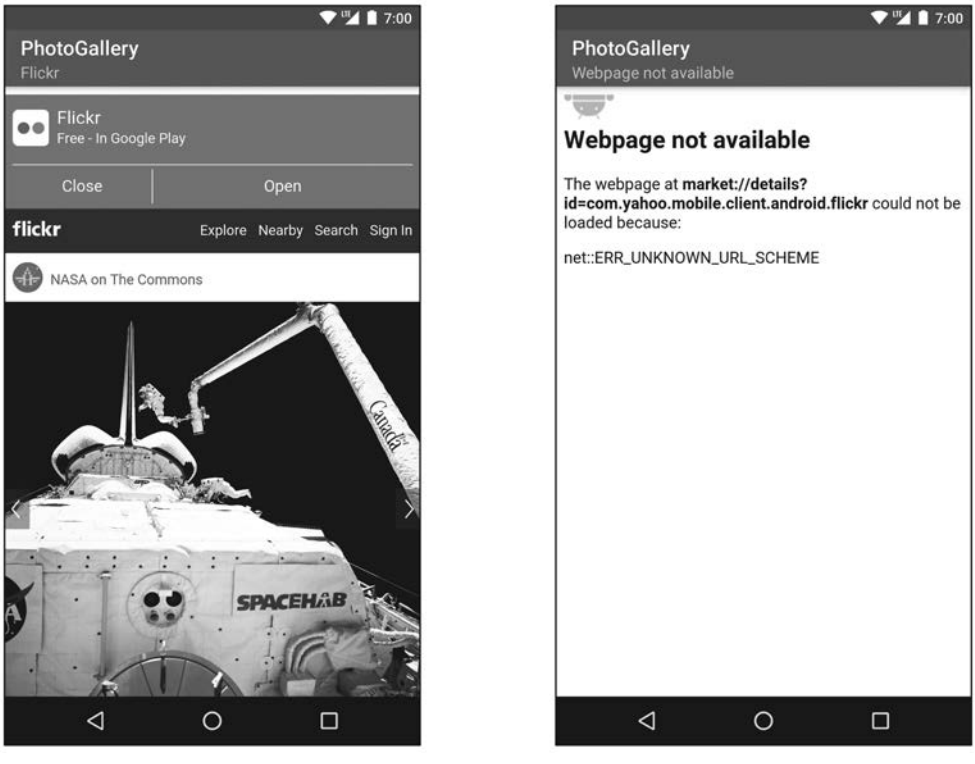


Рис. 30.4. Ошибка кнопки Open an App

Дело в том, что мы предоставили реализацию `WebViewClient`, которая приказывает `WebView` всегда пытаться загружать URI самостоятельно, даже если схема URI не поддерживается `WebView`.

Для решения этой проблемы необходимо организовать обработку URI с протоколом, отличным от HTTP, приложениями, которые лучше всего подходят для этих URI. Перед загрузкой URI проверьте схему. Если схема отлична от HTTP или HTTPS, выдайте действие `Intent.ACTION_VIEW` для URI.

31 Пользовательские представления и события касания

В этой главе мы займемся обработкой событий касания. Для этого мы создадим субкласс `View` с именем `BoxDrawingView`, который займет центральное место в новом проекте DragAndDraw. На этом представлении пользователь рисует прямоугольники, прикасаясь к экрану и перемещая палец. Результат выглядит примерно так, как показано на рис. 31.1.

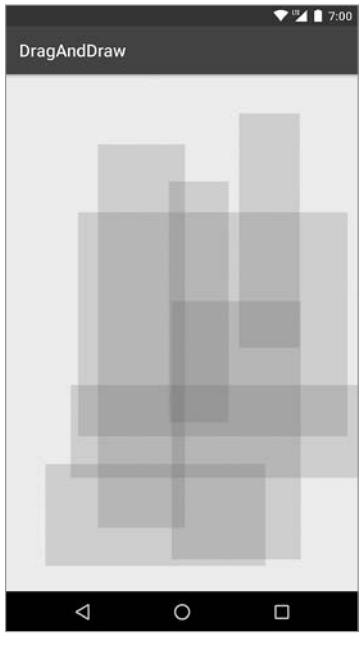


Рис. 31.1. Прямоугольники разных форм и размеров

Создание проекта DragAndDraw

Создайте новый проект с именем «DragAndDraw». Выберите в поле минимального SDK уровень API 19 и создайте пустую активность. Присвойте ей имя `DragAndDrawActivity`.

Класс `DragAndDrawActivity` представляет собой субкласс `SingleFragmentActivity`, который заполняет обычный макет с одним фрагментом. Скопируйте файл SingleFragmentActivity.java и его файл макета activity_fragment.xml в проект DragAndDraw.

В файле DragAndDrawActivity.java объявите `DragAndDrawActivity` субклассом `SingleFragmentActivity`. Этот класс должен создавать экземпляр `DragAndDrawFragment` (класс, который будет создан следующим).

Листинг 31.1. Изменение активности (DragAndDrawActivity.java)

```
public class DragAndDrawActivity extends AppCompatActivity SingleFragmentActivity
{
    @Override
    protected Fragment createFragment() {
        return DragAndDrawFragment.newInstance();
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
    }
}
```

Чтобы подготовить макет `DragAndDrawFragment`, переименуйте файл activity_drag_and_draw.xml в fragment_drag_and_draw.xml.

Макет `DragAndDrawFragment` в конечном итоге будет состоять из `BoxDrawingView` — пользовательского представления, которое мы собираемся написать. Весь графический вывод и обработка событий касания будут реализованы в `BoxDrawingView`.

Создайте новый класс с именем `DragAndDrawFragment` и назначьте его суперклассом `android.support.v4.app.ListFragment`. Переопределите метод `onCreateView(…)`, чтобы он заполнял макет fragment_drag_and_draw.xml.

Листинг 31.2. Создание фрагмента (DragAndDrawFragment.java)

```
public class DragAndDrawFragment extends Fragment {

    public static DragAndDrawFragment newInstance() {
        return new DragAndDrawFragment();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_drag_and_draw, container,
                 false);
        return v;
    }
}
```

Запустите приложение DragAndDraw и убедитесь в том, что настройка была выполнена правильно (рис. 31.2).

Рис. 31.2. DragAndDraw с макетом по умолчанию

Создание нестандартного представления

Android предоставляет много превосходных стандартных представлений и виджетов, но иногда требуется создать нестандартное представление с визуальным оформлением, полностью уникальным для вашего приложения.

Все многообразие нестандартных представлений можно условно разделить на две общие категории:

- *простые представления* — простое представление может быть устроено достаточно сложно; «простым» оно называется только потому, что не имеет дочерних представлений. Простое представление почти всегда также выполняет нестандартную прорисовку;
- *составные представления* — состоят из других объектов представлений. Составные представления обычно управляют дочерними представлениями, но не занимаются своей прорисовкой. Вместо этого каждому дочернему представлению делегируется своя часть работы по прорисовке.

Создание нестандартного представления состоит из трех шагов:

1. Выбор суперкласса. Для простого нестандартного представления View предоставляет пустой «холст» для рисования, поэтому этот выбор является наиболее распространенным. Для составных нестандартных представлений выберите подходящий класс макета (например, `FrameLayout`).

2. Субклассируйте выбранный класс и переопределите как минимум один конструктор из суперкласса или создайте собственный конструктор, вызывающий один из конструкторов суперкласса.
3. Переопределите другие ключевые методы для настройки поведения.

Создание класса BoxDrawingView

Класс `BoxDrawingView` относится к категории простых представлений и является прямым субклассом `View`.

Создайте новый класс с именем `BoxDrawingView` и назначьте `View` его суперклассом. Добавьте в файл BoxDrawingView.java два конструктора.

Листинг 31.3. Исходная реализация BoxDrawingView (BoxDrawingView.java)

```
public class BoxDrawingView extends View {

    // Используется при создании представления в коде
    public BoxDrawingView(Context context) {
        this(context, null);
    }

    // Используется при заполнении представления по разметке XML
    public BoxDrawingView(Context context, AttributeSet attrs) {
        super(context, attrs);
    }
}
```

Два конструктора нужны потому, что экземпляр представления может создаваться как в коде, так и по файлу разметки. Представления, созданные на базе файла макета, получают экземпляр `AttributeSet` с атрибутами XML, заданными в XML. Даже если вы не собираетесь использовать оба конструктора, их рекомендуется включить.

Затем обновите файл макета fragment_drag_and_draw.xml, чтобы в нем использовалось новое представление.

Листинг 31.4. Включение BoxDrawingView в макет (fragment_drag_and_draw.xml)

```
<android.support.constraint.ConstraintLayout
    android:id="@+id/activity_drag_and_draw"
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.bignerdranch.android.draganddraw.DragAndDrawActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="@+id/activity_drag_and_draw"
        app:layout_constraintLeft_toLeftOf="@+id/activity_drag_and_draw"
        app:layout_constraintRight_toRightOf="@+id/activity_drag_and_draw"
        app:layout_constraintTop_toTopOf="@+id/activity_drag_and_draw"/>

</android.support.constraint.ConstraintLayout>
<com.bignerdranch.android.draganddraw.BoxDrawingView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Чтобы заполнитель макетов нашел класс `BoxDrawingView`, вы должны использовать полностью уточненное имя. Заполнитель просматривает файл макета, создавая экземпляры `View`. Если имя элемента будет неполным, то заполнитель ищет класс с указанным именем в пакетах `android.view` и `android.widget`. Если класс находится в другом месте, заполнитель его не найдет, и в приложении произойдет сбой.

По этой причине для классов, не входящих в `android.view` и `android.widget`, необходимо всегда задавать полностью уточненное имя.

Запустите приложение DragAndDraw и убедитесь в том, что настройка была выполнена правильно. Правда, пока на экране нет ничего, кроме пустого представления (рис. 31.3).

Рис. 31.3. BoxDrawingView без прямоугольников

На следующем этапе мы научим BoxDrawingView прослушивать события касания и использовать содержащуюся в них информацию для рисования прямоугольников на экране.

Обработка событий касания

Для прослушивания событий касания можно назначить слушателя события при помощи следующего метода класса View:

```
public void setOnTouchListener(View.OnTouchListener l)
```

Этот метод работает почти так же, как setOnClickListener(View.OnClickListener). Вы предоставляете реализацию View.OnTouchListener, а слушатель вызывается каждый раз, когда происходит событие касания.

Но поскольку мы субклассируем View, можно пойти по сокращенному пути и переопределить следующий метод класса View:

```
public boolean onTouchEvent(MotionEvent event)
```

Этот метод получает экземпляр MotionEvent — класса, описывающего событие касания, включая его позицию и *действие*. Действие описывает стадию события.

Константы действий	Описание
ACTION_DOWN	Пользователь прикоснулся к экрану
ACTION_MOVE	Пользователь перемещает палец по экрану
ACTION_UP	Пользователь отводит палец от экрана
ACTION_CANCEL	Родительское представление перехватило событие касания

В своей реализации onTouchEvent(MotionEvent) для проверки действия можете воспользоваться следующим методом класса MotionEvent:

```
public final int getAction()
```

Добавьте в файл BoxDrawingView.java тег для журнала и реализацию onTouchEvent(MotionEvent), которая регистрирует в журнале информацию о каждом из четырех разных действий.

Листинг 31.5. Реализация BoxDrawingView (BoxDrawingView.java)

```
public class BoxDrawingView extends View {
    private static final String TAG = "BoxDrawingView";
    ...
    @Override
    public boolean onTouchEvent(MotionEvent event) {
        PointF current = new PointF(event.getX(), event.getY());
        String action = "";
```

```
        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                action = "ACTION_DOWN";
                break;
            case MotionEvent.ACTION_MOVE:
                action = "ACTION_MOVE";
                break;
            case MotionEvent.ACTION_UP:
                action = "ACTION_UP";
                break;
            case MotionEvent.ACTION_CANCEL:
                action = "ACTION_CANCEL";
                break;
        }

        Log.i(TAG, action + " at x=" + current.x +
                ", y=" + current.y);

        return true;
    }
}
```

Обратите внимание: координаты *X* и *Y* упаковываются в объекте `PointF`. В оставшейся части этой главы эти два значения обычно будут передаваться вместе. `PointF` — предоставленный Android класс-контейнер, который решает эту задачу за вас.

Запустите приложение DragAndDraw и откройте LogCat. Прикоснитесь к экрану и проведите пальцем. Вы увидите в журнале сообщения с координатами *X* и *Y* каждого действия касания, полученного `BoxDrawingView`.

Отслеживание перемещений между событиями

Класс `BoxDrawingView` должен рисовать прямоугольники, а не регистрировать координаты. Для этого необходимо решить ряд задач.

Прежде всего для определения прямоугольника нам понадобятся две точки: базовая (в которой было сделано исходное касание) и текущая (в которой находится палец).

Следовательно, для определения прямоугольника необходимо отслеживать данные от нескольких событий `MotionEvent`. Данные будут храниться в объекте `Box`.

Создайте класс с именем `Box` для хранения данных, определяющих прямоугольник.

Листинг 31.6. Класс Box (Box.java)

```
public class Box {
    private PointF mOrigin;
    private PointF mCurrent;

    public Box(PointF origin) {
        mOrigin = origin;
        mCurrent = origin;
```

```
    }

    public PointF getCurrent() {
        return mCurrent;
    }

    public void setCurrent(PointF current) {
        mCurrent = current;
    }

    public PointF getOrigin() {
        return mOrigin;
    }
}
```

Когда пользователь прикасается к `BoxDrawingView`, новый объект `Box` создается и включается в массив существующих прямоугольников (рис. 31.4).

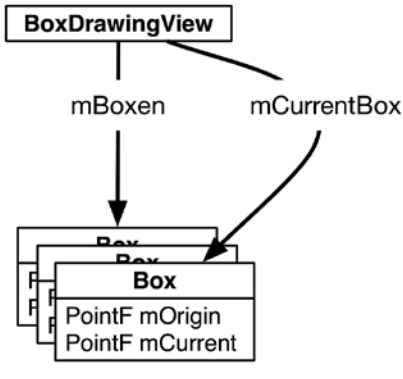


Рис. 31.4. Объекты в DragAndDraw

Добавьте в `BoxDrawingView` код, использующий новый объект `Box` для отслеживания текущего состояния рисования.

Листинг 31.7. Добавление методов жизненного цикла событий касания (BoxDrawingView.java)

```
public class BoxDrawingView extends View {
    private static final String TAG = "BoxDrawingView";

    private Box mCurrentBox;
    private List<Box> mBoxen = new ArrayList<>();
    ...
    @Override
    public boolean onTouchEvent(MotionEvent event) {
        PointF current = new PointF(event.getX(), event.getY());
        String action = "";

        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                action = "ACTION_DOWN";
```

```
                // Сброс текущего состояния
                mCurrentBox = new Box(current);
                mBoxen.add(mCurrentBox);
                break;
            case MotionEvent.ACTION_MOVE:
                action = "ACTION_MOVE";
                if (mCurrentBox != null) {
                    mCurrentBox.setCurrent(current);
                    invalidate();
                }
                break;
            case MotionEvent.ACTION_UP:
                action = "ACTION_UP";
                mCurrentBox = null;
                break;
            case MotionEvent.ACTION_CANCEL:
                action = "ACTION_CANCEL";
                mCurrentBox = null;
                break;
        }

        Log.i(TAG, action + " at x=" + current.x +
                ", y=" + current.y);

        return true;
    }
}
```

При каждом получении события `ACTION_DOWN` в поле `mCurrentBox` сохраняется новый объект `Box` с базовой точкой, соответствующей позиции события. Этот объект `Box` добавляется в массив прямоугольников (в следующем разделе, когда мы займемся прорисовкой, `BoxDrawingView` будет выводить каждый объект `Box` из массива.)

В процессе перемещения пальца по экрану приложение обновляет `mCurrentBox.mCurrent`. Затем, когда касание отменяется или палец не касается экрана, поле `mCurrentBox` обнуляется для завершения операции. Объект `Box` завершен; он сохранен в массиве и уже не будет обновляться событиями перемещения.

Обратите внимание на вызов `invalidate()` в случае `ACTION_MOVE`. Он заставляет `BoxDrawingView` перерисовать себя, чтобы пользователь видел прямоугольник в процессе перетаскивания. Мы подошли к следующему шагу: рисованию прямоугольников на экране.

Рисование внутри onDraw(Canvas)

При запуске приложения все его представления *недействительны* (invalid). Это означает, что они ничего не вывели на экран. Для исправления ситуации Android вызывает метод `draw()` объекта `View` верхнего уровня. В результате представление перерисовывает себя, что заставляет его потомков перерисовать себя. Затем потомки этих потомков перерисовывают себя и так далее вниз по иерархии. Когда все представления в иерархии перерисуют себя, объект `View` верхнего уровня перестает быть недействительным.

Чтобы вмешаться в процесс прорисовки, следует переопределить следующий метод `View`:

```
protected void onDraw(Canvas canvas)
```

Вызов `invalidate()`, выполняемый в ответ на действие `ACTION_MOVE` в `onTouchEvent(MotionEvent)`, снова делает объект `BoxDrawingView` недействительным. Это заставляет его перерисовать себя и приводит к повторному вызову `onDraw(Canvas)`.

Обратите внимание на параметр `Canvas`. `Canvas` и `Paint` — два главных класса, используемых при рисовании в Android.

- Класс `Canvas` содержит все выполняемые операции графического вывода. Методы, вызываемые для объекта Canvas, определяют, где и что выводится — линия, круг, слово или прямоугольник.
- Класс `Paint` определяет, как будут выполняться эти операции. Методы, вызываемые для объекта `Paint`, определяют характеристики вывода: должны ли фигуры заполняться, каким шрифтом должен выводиться текст, каким цветом должны выводиться линии и т. д.

В файле BoxDrawingView.java создайте два объекта `Paint` в конструкторе `BoxDrawingView` для XML.

Листинг 31.8. Создание объектов Paint (BoxDrawingView.java)

```
public class BoxDrawingView extends View {
    private static final String TAG = "BoxDrawingView";

    private Box mCurrentBox;
    private List<Box> mBoxen = new ArrayList<>();
    private Paint mBoxPaint;
    private Paint mBackgroundPaint;
    ...
    // Используется при заполнении представления по разметке XML
    public BoxDrawingView(Context context, AttributeSet attrs) {
        super(context, attrs);

        // Прямоугольники рисуются полупрозрачным красным цветом (ARGB)
        mBoxPaint = new Paint();
        mBoxPaint.setColor(0x22ff0000);

        // Фон закрашивается серовато-белым цветом
        mBackgroundPaint = new Paint();
        mBackgroundPaint.setColor(0xfff8efe0);
    }
}
```

После создания объектов `Paint` можно переходить к рисованию прямоугольников на экране.

Листинг 31.9. Переопределение onDraw(Canvas) (BoxDrawingView.java)

```
public BoxDrawingView(Context context, AttributeSet attrs) {
    ...
}

@Override
```

```
protected void onDraw(Canvas canvas) {
    // Заполнение фона
    canvas.drawPaint(mBackgroundPaint);

    for (Box box : mBoxen) {
        float left = Math.min(box.getOrigin().x, box.getCurrent().x);
        float right = Math.max(box.getOrigin().x, box.getCurrent().x);
        float top = Math.min(box.getOrigin().y, box.getCurrent().y);
        float bottom = Math.max(box.getOrigin().y, box.getCurrent().y);
        canvas.drawRect(left, top, right, bottom, mBoxPaint);
    }
}
```

Первая часть кода тривиальна: используя серовато-белый цвет, мы заполняем «холст» задним фоном для вывода прямоугольников.

Затем для каждого прямоугольника в списке мы определяем значения `left`, `right`, `top` и `bottom` по двум точкам. Значения `left` и `top` будут минимальными, а `bottom` и `right` — максимальными.

После вычисления параметров вызов метода `Canvas.drawRect(…)` рисует красный прямоугольник на экране.

Запустите приложение DragAndDraw и нарисуйте несколько прямоугольников (рис. 31.5).

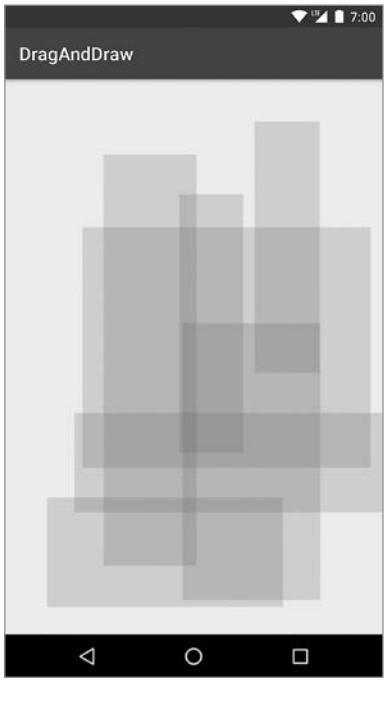


Рис. 31.5. Приложение с нарисованными прямоугольниками

Мы создали представление, которое обрабатывает свои события касания и выполняет прорисовку.

Упражнение. Сохранение состояния

Подумайте, как обеспечить сохранение состояния прямоугольников при изменении ориентации из `View`. В этом вам могут помочь следующие методы `View`:

```
protected Parcelable onSaveInstanceState()
protected void onRestoreInstanceState(Parcelable state)
```

Эти методы работают не так, как метод `onSaveInstanceState(Bundle)` классов `Activity` и `Fragment`. Вместо объекта `Bundle` они возвращают и обрабатывают объект, реализующий интерфейс `Parcelable`. Мы рекомендуем использовать `Bundle` в качестве `Parcelable` вместо того, чтобы писать реализацию `Parcelable` самостоятельно. (Реализация интерфейса `Parcelable` весьма сложна. Лучше избегать ее там, где это возможно.)

Наконец, вы также должны поддерживать сохраненное состояние родителя `BoxDrawingView`, класса `View`. Сохраните результат `super.onSaveInstanceState()` в новом объекте `Bundle` и передайте его суперклассу при вызове `super.onRestoreInstanceState(Parcelable)`.

Упражнение. Повороты

Еще одно, более сложное упражнение: реализуйте возможность вращения прямоугольников вторым пальцем. Для этого вам потребуется отслеживать операции с несколькими указателями в коде обработки `MotionEvent`. Также придется отрабатывать повороты `Canvas`.

При работе с множественными касаниями вам понадобятся:

- *индекс указателя* — сообщает, к какому указателю в текущем наборе относится событие;
- *идентификатор указателя* — обеспечивает однозначную идентификацию конкретного пальца в жесте.

Индекс указателя может изменяться, но идентификатор остается неизменным.

За дополнительной информацией обращайтесь к документации по следующим методам `MotionEvent`:

```
public final int getActionMasked()
public final int getActionIndex()
public final int getPointerId(int pointerIndex)
public final float getX(int pointerIndex)
public final float getY(int pointerIndex)
```

Также посмотрите документацию по константам `ACTION_POINTER_UP` и `ACTION_POINTER_DOWN`.

32 Анимация свойств

Чтобы приложение было работоспособным, достаточно правильно написать код. Но чтобы работа с приложением была настоящим удовольствием, этого недостаточно. Приложение должно восприниматься как реальное, физическое явление, происходящее на экране телефона или планшета.

Как известно, реальные явления двигаются. Чтобы элементы пользовательского интерфейса двигались, к ним применяется *анимация*.

В этой главе мы напишем приложение, которое изображает сцену с солнцем в небе. При нажатии солнце опускается за горизонт, а небо окрашивается в закатный цвет.

Построение сцены

Все начинается с построения сцены, к которой будет применяться анимация. Создайте новый проект с именем Sunset. Убедитесь в том, что `minSdkVersion` присвоено значение 19. Присвойте главной активности имя `SunsetActivity`, добавьте в проект файлы SingleFragmentActivity.java и activity_fragment.xml.

Теперь постройте сцену. Закат у моря полон красок, поэтому для удобства мы присвоим названия некоторым цветам. Добавьте в папку res/values файл colors.xml и включите в него следующие значения:

Листинг 32.1. Добавление цветов (res/values/colors.xml)

```
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>

    <color name="bright_sun">#fcfcb7</color>
    <color name="blue_sky">#1e7ac7</color>
    <color name="sunset_sky">#ec8100</color>
    <color name="night_sky">#05192e</color>
    <color name="sea">#224869</color>
</resources>
```

Прямоугольные представления неплохо подойдут для моря и неба. Однако вряд ли кто-нибудь оценит квадратное солнце, какие бы доводы вы ни выдвигали

в пользу технической простоты такого решения. Добавьте в папку res/drawable/ круглый графический объект sun.xml.

Листинг 32.2. Добавление графического объекта XML для солнца (res/values/colors.xml)

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="oval">
    <solid android:color="@color/bright_sun" />
</shape>
```

Если вывести эллипс в квадратном представлении, получится круг. Зрители одобрительно кивнут при виде столь убедительной имитации.

Вся сцена будет построена в файле макета. Этот макет будет использоваться фрагментом `SunsetFragment`, который мы вскоре построим; присвойте ему имя fragment_sunset.xml.

Листинг 32.3. Создание макета (res/layout/fragment_sunset.xml)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <FrameLayout
        android:id="@+id/sky"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="0.61"
        android:background="@color/blue_sky">
        <ImageView
            android:id="@+id/sun"
            android:layout_width="100dp"
            android:layout_height="100dp"
            android:layout_gravity="center"
            android:src="@drawable/sun" />
    </FrameLayout>

    <View
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="0.39"
        android:background="@color/sea" />
</LinearLayout>
```

Проверьте предварительное изображение макета. У вас должна получиться дневная сцена с солнцем в синем небе над темно-синим морем.

А теперь пора заняться отображением этой сцены на устройстве. Создайте фрагмент с именем `SunsetFragment` и добавьте метод `newInstance(…)`. В коде `onCreateView(…)` заполните файл макета fragment_sunset и верните полученное представление.

Листинг 32.4. Создание SunsetFragment (SunsetFragment.java)

```
public class SunsetFragment extends Fragment {

    public static SunsetFragment newInstance() {
        return new SunsetFragment();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_sunset, container,
false);

        return view;
    }
}
```

Теперь преобразуйте `SunsetActivity` в субкласс `SingleFragmentActivity`, в котором отображается ваш фрагмент.

Листинг 32.5. Отображение SunsetFragment (SunsetActivity.java)

```
public class SunsetActivity extends SingleFragmentActivity {

    @Override
    protected Fragment createFragment() {
        return SunsetFragment.newInstance();
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
    }
}
```

Прежде чем двигаться дальше, запустите Sunset и убедитесь в том, что все связи созданы правильно. Приложение должно выглядеть так, как показано на рис. 32.1.

Простая анимация свойств

Итак, сцена подготовлена; теперь нужно привести ее составляющие в движение. Анимация будет использована для перемещения солнца под линию горизонта.

Но прежде чем браться за анимацию, стоит подготовить кое-какие данные в фрагменте. В методе `onCreateView(…)` занесите пару представлений в поля `SunsetFragment`.

Листинг 32.6. Получение ссылок на представления (SunsetActivity.java)

```
public class SunsetFragment extends Fragment {

    private View mSceneView;
    private View mSunView;
    private View mSkyView;
```

```
    public static SunsetFragment newInstance() {
        return new SunsetFragment();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_sunset, container, false);

        mSceneView = view;
        mSunView = view.findViewById(R.id.sun);
        mSkyView = view.findViewById(R.id.sky);

        return view;
    }
}
```

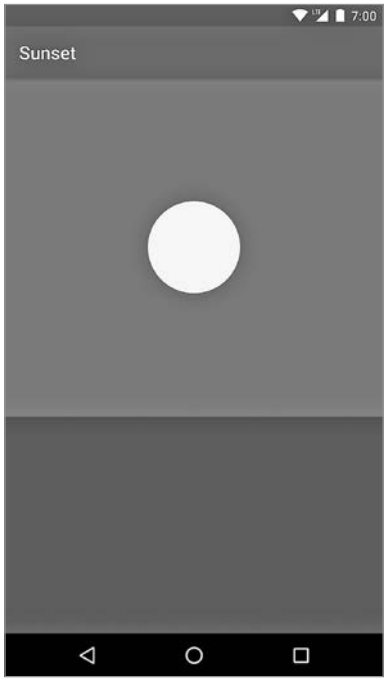


Рис. 32.1. Перед закатом

После завершения подготовки можно переходить к написанию кода анимации. План выглядит так: `mSunView` плавно перемещается так, чтобы верхний край представления совпал с верхним краем моря. Для этого к позиции верхнего края `mSunView` будет применен *сдвиг* до нижнего края родителя.

Прежде всего следует определить начальное и конечное состояния анимации. Этот первый шаг будет реализован в новом методе с именем `startAnimation()`.

Листинг 32.7. Получение верхних координат представлений (SunsetFragment.java)

```
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        ...
    }

    private void startAnimation() {
        float sunYStart = mSunView.getTop();
        float sunYEnd = mSkyView.getHeight();
    }
```

Метод `getTop()` — один из четырех методов `View`, возвращающих прямоугольник локального макета для этого представления: `getTop()`, `getBottom()`, `getRight()` и `getLeft()`. Прямоугольник локального макета представления определяет позицию и размер этого представления относительно его родителя на момент включения представления в макет. В принципе, положение представления на экране можно варьировать, меняя эти значения, но делать это не рекомендуется. Эти значения сбрасываются при каждом проходе обработки макета, поэтому присвоенные значения обычно долго не держатся.

В любом случае анимация будет начинаться от верха текущего положения представления и заканчиваться в состоянии, при котором верх находится у нижнего края родителя `mSunView`, то есть `mSkyView`. Для перехода в новое состояние потребуется смещение на величину, равную высоте `mSkyView`, которая может быть определена вызовом `getHeight()`. Метод `getHeight()` возвращает результат, который также может быть получен по формуле `getTop() - getBottom()`.

Теперь, когда вы знаете, где должна начинаться и завершаться анимация, создайте и запустите экземпляр `ObjectAnimator` для ее выполнения.

Листинг 32.8. Создание анимации солнца (SunsetFragment.java)

```
private void startAnimation() {
    float sunYStart = mSunView.getTop();
    float sunYEnd = mSkyView.getHeight();

    ObjectAnimator heightAnimator = ObjectAnimator
            .ofFloat(mSunView, "y", sunYStart, sunYEnd)
            .setDuration(3000);

    heightAnimator.start();
}
```

Затем подключите метод `startAnimation()`, чтобы он выполнялся каждый раз, когда пользователь касается любой точки сцены.

Листинг 32.9. Запуск анимации по нажатии (SunsetFragment.java)

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_sunset, container, false);
```

```
    mSceneView = view;
    mSunView = view.findViewById(R.id.sun);
    mSkyView = view.findViewById(R.id.sky);

    mSceneView.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            startAnimation();
        }
    });

    return view;
}
```

Запустите приложение Sunset и коснитесь любой точки сцены, чтобы запустить анимацию (рис. 32.2).

Вы увидите, как солнце опускается за горизонт.

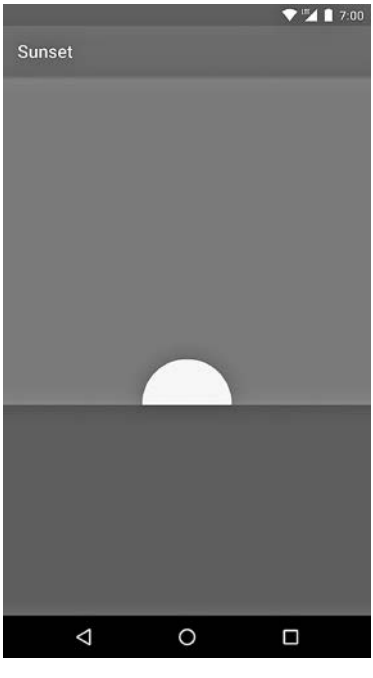


Рис. 32.2. Заход солнца

Как же работает это решение? `ObjectAnimator` называется *аниматором свойства*. Ничего не зная о том, как перемещать представление по экрану, аниматор свойства многократно вызывает `set`-методы свойства с разными значениями.

Объект `ObjectAnimator` создается следующим вызовом метода:

```
ObjectAnimator.ofFloat(mSunView, "y", 0, 1)
```

При запуске `ObjectAnimator` метод `mSunView.setY(float)` многократно вызывается с постепенно увеличивающимися значениями, начиная с 0:

```
mSunView.setY(0);
mSunView.setY(0.02);
mSunView.setY(0.04);
mSunView.setY(0.06);
mSunView.setY(0.08);
...
```

...и так далее, пока не будет вызван метод `mSunView.setY(1)`. Процесс вычисления значений между начальной и конечной точками называется *интерполяцией*. Между каждой интерполированной парой проходит небольшой промежуток времени; так создается иллюзия перемещения представления.

Свойства преобразований

Аниматоры свойств весьма удобны, но, пользуясь только ими, было бы невозможно организовать анимацию представлений настолько легко, как мы это сделали. Современная анимация свойств в Android работает в сочетании со свойствами преобразований.

С нашим представлением связывается прямоугольник локального макета, позиция и размер которого назначаются в процессе макета. Вы можете перемещать представление, задавая дополнительные свойства представления, называемые *свойствами преобразований* (transformation properties). В вашем распоряжении три свойства для выполнения поворотов (`rotation`, `pivotX` и `pivotY`, рис. 32.3), два свойства для масштабирования представления по вертикали и по горизонтали (`scaleX` и `scaleY`, рис. 32.4), два свойства для сдвига представлений (`translationX` и `translationY`, рис. 32.5).

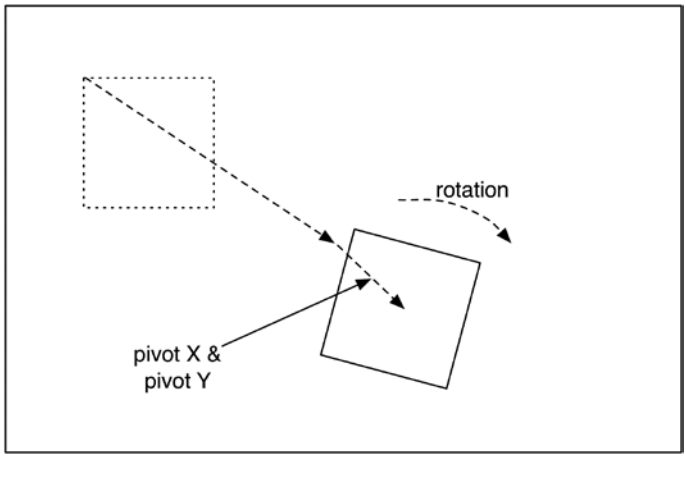


Рис. 32.3. Поворот представления

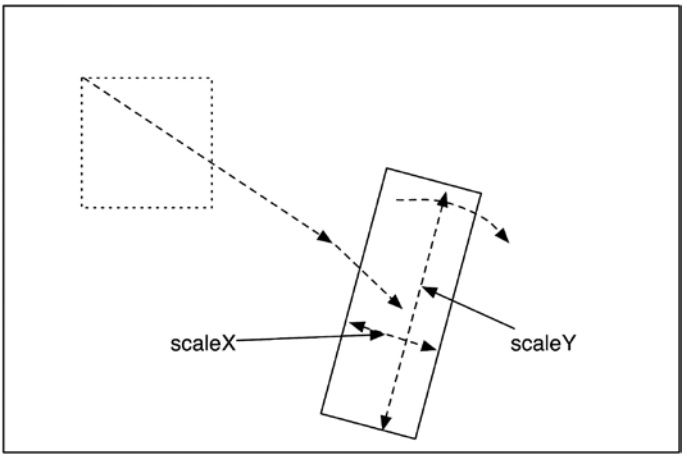


Рис. 32.4. Масштабирование представлений

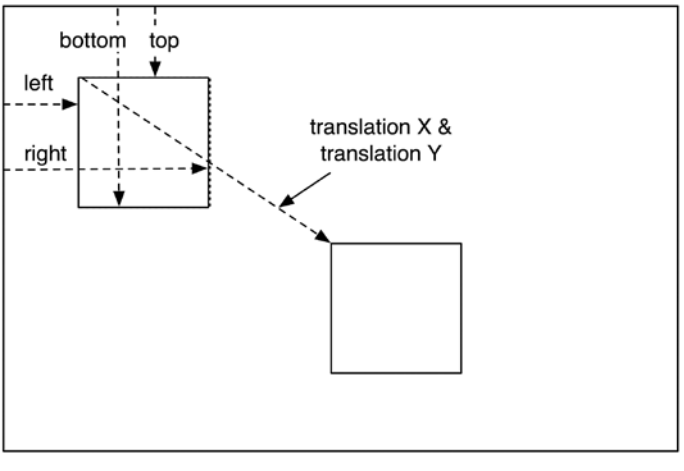


Рис. 32.5. Сдвиг представления

У всех таких свойств существуют `get`- и `set`-методы. Например, для получения текущего значения `translationX` можно вызвать метод `getTranslationX()`, а если вы хотите задать свойству новое значение, вызовите `setTranslationX(float)`.

Как работает свойство `y`? Вспомогательные свойства `x` и `y` созданы на базе локальных координат макета и свойств преобразований. С ними вы можете писать код, который означает: «Разместить это представление в точке с такими координатами *X* и *Y*». Во внутренней реализации эти свойства изменяют `translationX` или `translationY`, чтобы разместить представление в нужной точке. Это означает, что вызов `mSunView.setY(50)` в действительности эквивалентен:

```
mSunView.setTranslationY(50 - mSunView.getTop())
```

Выбор интерполятора

Наша анимация хорошо смотрится, но выполняется слишком резко. Если солнце неподвижно, ему понадобится некоторое время для того, чтобы набрать скорость. Чтобы смоделировать это ускорение, достаточно воспользоваться объектом `TimeInterpolator`. Класс `TimeInterpolator` решает всего одну задачу: он изменяет способ перехода анимации от точки A к точке B.

Добавьте в `startAnimation()` строку кода, которая обеспечит небольшое ускорение солнца в начале анимации с использованием объекта `AccelerateInterpolator`.

Листинг 32.10. Реализация ускорения (SunsetFragment.java)

```
private void startAnimation() {
    float sunYStart = mSunView.getTop();
    float sunYEnd = mSkyView.getHeight();

    ObjectAnimator heightAnimator = ObjectAnimator
            .ofFloat(mSunView, "y", sunYStart, sunYEnd)
            .setDuration(3000);
    heightAnimator.setInterpolator(new AccelerateInterpolator());

    heightAnimator.start();
}
```

Снова запустите приложение Sunset и коснитесь экрана, чтобы просмотреть анимацию. Солнце начинает медленно двигаться и ускоряется при подходе к горизонту.

Существует много разных видов движения, которые могут использоваться в приложениях, поэтому существует много разновидностей `TimeInterpolator`. Полный список всех интерполяторов, входящих в поставку Android, приведен в разделе «Known Indirect Subclasses» справочной документации `TimeInterpolator`.

Изменение цвета

Теперь давайте окрасим небо в закатный цвет. В методе `onCreateView(…)` загрузите все цвета, определенные в colors.xml, в переменные экземпляров.

Листинг 32.11. Загрузка закатных цветов (SunsetFragment.java)

```
public class SunsetFragment extends Fragment {
    ...
    private View mSkyView;

    private int mBlueSkyColor;
    private int mSunsetSkyColor;
    private int mNightSkyColor;
    ...
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
```

```
        ...
        mSkyView = view.findViewById(R.id.sky);

        Resources resources = getResources();
        mBlueSkyColor = resources.getColor(R.color.blue_sky);
        mSunsetSkyColor = resources.getColor(R.color.sunset_sky);
        mNightSkyColor = resources.getColor(R.color.night_sky);

        mSceneView.setOnClickListener(new View.OnClickListener() {
            ...
        });

        return view;
    }
```

Включите в `startAnimation()` дополнительную анимацию цвета неба от `mBlueSkyColor` до `mSunsetSkyColor`.

Листинг 32.12. Анимация цвета неба (SunsetFragment.java)

```
private void startAnimation() {
    float sunYStart = mSunView.getTop();
    float sunYEnd = mSkyView.getHeight();

    ObjectAnimator heightAnimator = ObjectAnimator
            .ofFloat(mSunView, "y", sunYStart, sunYEnd)
            .setDuration(3000);
    heightAnimator.setInterpolator(new AccelerateInterpolator());

    ObjectAnimator sunsetSkyAnimator = ObjectAnimator
            .ofInt(mSkyView, "backgroundColor", mBlueSkyColor, mSunsetSkyColor)
            .setDuration(3000);

    heightAnimator.start();
    sunsetSkyAnimator.start();
}
```

Вроде бы все делается правильно, но, запустив приложение, вы увидите, что что-то сделано не так. Вместо плавного перехода от синего цвета к оранжевому цвета хаотично меняются, словно в калейдоскопе.

Это происходит из-за того, что целочисленное представление цвета — это не просто число, а четыре меньших числа, объединенных в одно значение `int`. Таким образом, чтобы объект `ObjectAnimator` мог правильно вычислить промежуточный цвет на пути от синего к оранжевому, он должен знать, как это делать.

Когда обычного умения `ObjectAnimator` по вычислению промежуточных значений между начальной и конечной точками оказывается недостаточно, вы можете определить субкласс `TypeEvaluator` для решения проблемы. `TypeEvaluator` — объект, который сообщает `ObjectAnimator`, какое значение находится, допустим, на четверти пути от начальной до конечной точки. Android предоставляет субкласс `TypeEvaluator` с именем `ArgbEvaluator`, который позволит добиться желаемого результата.

Листинг 32.13. Назначение ArgbEvaluator (SunsetFragment.java)

```
private void startAnimation() {
    float sunYStart = mSunView.getTop();
    float sunYEnd = mSkyView.getHeight();

    ObjectAnimator heightAnimator = ObjectAnimator
            .ofFloat(mSunView, "y", sunYStart, sunYEnd)
            .setDuration(3000);
    heightAnimator.setInterpolator(new AccelerateInterpolator());

    ObjectAnimator sunsetSkyAnimator = ObjectAnimator
            .ofInt(mSkyView, "backgroundColor", mBlueSkyColor, mSunsetSkyColor)
            .setDuration(3000);
    sunsetSkyAnimator.setEvaluator(new ArgbEvaluator());

    heightAnimator.start();
    sunsetSkyAnimator.start();
}
```

Запустите анимацию еще раз, и вы увидите, как небо окрашивается в красивый оранжевый цвет (рис. 32.6).

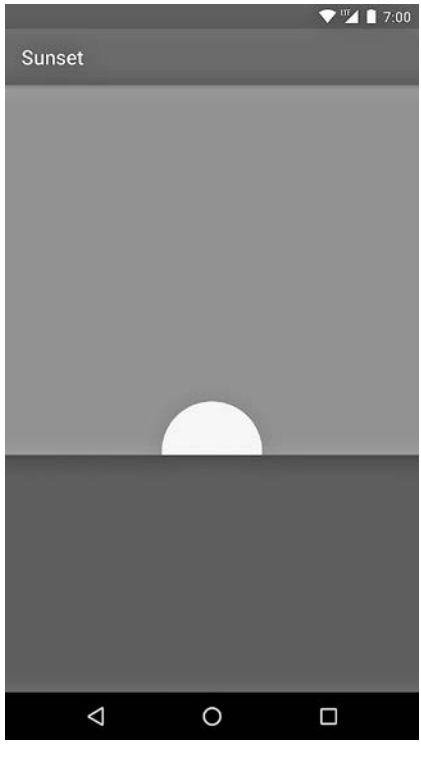


Рис. 32.6. Изменение цвета закатного неба

Одновременное воспроизведение анимаций

Если вам требуется всего лишь воспроизвести несколько анимаций одновременно, то ваша задача проста: одновременно вызовите для них `start()`. Все анимации будут выполняться синхронно.

В ситуациях с более сложными анимациями этого недостаточно. Например, чтобы завершить иллюзию заката, было бы неплохо окрасить оранжевое небо в полуночный синий цвет после захода солнца.

Задача решается при помощи объекта `AnimatorListener`. `AnimatorListener` сообщает о завершении анимации; таким образом, вы можете написать слушателя, который ожидает завершения первой анимации, а потом запустить вторую анимацию ночного неба. Тем не менее такое решение чрезвычайно хлопотно и требует слишком большого количества слушателей. Намного проще воспользоваться `AnimatorSet`.

Сначала постройте анимацию ночного неба и удалите старый код начала анимации.

Листинг 32.14. Построение ночной анимации (SunsetFragment.java)

```
private void startAnimation() {
    ...
    sunsetSkyAnimator.setEvaluator(new ArgbEvaluator());

    ObjectAnimator nightSkyAnimator = ObjectAnimator
            .ofInt(mSkyView, "backgroundColor", mSunsetSkyColor, mNightSkyColor)
            .setDuration(1500);
    nightSkyAnimator.setEvaluator(new ArgbEvaluator());

    heightAnimator.start();
    sunsetSkyAnimator.start();
}
```

Затем постройте и запустите `AnimatorSet`.

Листинг 32.15. Построение AnimatorSet (SunsetFragment.java)

```
private void startAnimation() {
    ...
    ObjectAnimator nightSkyAnimator = ObjectAnimator
            .ofInt(mSkyView, "backgroundColor", mSunsetSkyColor, mNightSkyColor)
            .setDuration(1500);
    nightSkyAnimator.setEvaluator(new ArgbEvaluator());

    AnimatorSet animatorSet = new AnimatorSet();
    animatorSet
            .play(heightAnimator)
            .with(sunsetSkyAnimator)
            .before(nightSkyAnimator);
    animatorSet.start();
}
```

Объект `AnimatorSet` представляет набор анимаций, которые могут воспроизводиться совместно. Существует несколько способов построения таких объектов; проще всего воспользоваться методом `play(Animator)`, который использовался выше.

При вызове `play(Animator)` вы получаете объект `AnimatorSet.Builder`, который позволяет построить цепочку инструкций. Объект `Animator`, передаваемый `play(Animator)`, является «субъектом» цепочки. Таким образом, написанная нами цепочка вызовов может быть описана в виде «Воспроизвести `heightAnimator` с `sunsetSkyAnimator`; также воспроизвести `heightAnimator` до `nightSkyAnimator`». Возможно, в сложных разновидностях `AnimatorSet` потребуется вызвать `play(Animator)` несколько раз; это вполне нормально.

Запустите приложение еще раз и оцените созданный вами умиротворяющий закат. Волшебно.

Для любознательных: другие API для анимации

Анимация свойств — наиболее универсальный механизм в инструментарии анимации, но не единственный. Полезно знать и о других возможностях независимо от того, используете вы их или нет.

Старые средства анимации

Еще одну группу составляют классы из пакета `android.view.animation` (не путайте с более новым пакетом `android.animation`, появившимся в Honeycomb).

Это устаревшая инфраструктура анимации, о которой следует знать в основном для того, чтобы избегать ее. Если в имени класса присутствует слово «animaTION» вместо «animaTOR», это верный признак того, что перед вами старый инструмент и пользоваться им не следует.

Переходы

В Android 4.4 появилась новая инфраструктура, которая позволяет создавать эффектные переходы (transitions) между иерархиями представлений. Например, можно определить переход, при котором маленькое представление в одной активности «разворачивается» в увеличенную версию этого представления в другой активности.

Основной принцип механизма переходов — определение сцен, представляющих состояние иерархии представлений в некоторой точке, и переходов между этими сценами. Сцены могут описываться в XML-файлах макетов, а переходы — в XML-файлах анимации.

Когда активность уже работает, как в этой главе, механизм переходов особой пользы не принесет. В таких ситуациях эффективно работает механизм анимации свойств. С другой стороны, механизм анимации свойств не столь удобен для анимации макета в процессе его появления на экране.

Для примера возьмем фотографии места преступления из приложения CriminalIntent. Если вы попытаетесь реализовать анимацию «увеличения» в диалоговом окне изображения, вам придется самостоятельно рассчитывать, где находится исходное изображение и где будет находиться новое изображение в диалоговом окне. С `ObjectAnimator` реализация таких эффектов требует значительного объема работы. В таких случаях лучше воспользоваться инфраструктурой переходов.

Упражнения

Для начала добавьте возможность «обратить» закат солнца после его завершения: при первом нажатии солнце заходит, а при втором происходит восход. Для этого вам придется построить новый объект `AnimatorSet` — эти объекты не могут выполняться в обратном направлении.

Затем добавьте к солнцу вторичную анимацию: заставьте его дрожать от жара или создайте вращающийся ореол (для повторения анимации можно воспользоваться методом `setRepeatCount(int)` класса `ObjectAnimator`).

Еще одно интересное упражнение — изобразить отражение солнца в воде.

На последнем шаге добавьте возможность обращения сцены заката по нажатии во время ее выполнения. Иначе говоря, если пользователь нажимает на сцене, пока солнце находится на середине пути, оно снова поднимается на небо. Аналогичным образом при нажатии во время наступления темноты небо снова должно окрашиваться в цвет зари.

33 Отслеживание местоположения устройства

В этой главе мы начнем работу над новым приложением с именем Locatr, предназначенным для выполнения геопоиска на сервисе Flickr. Оно определяет текущее местоположение пользователя, после чего ищет фотографии окрестностей (рис. 33.1). В следующей главе мы займемся выводом найденных изображений на карте.

Рис. 33.1. Приложение Locatr к концу главы

Оказывается, даже эта простая задача — определение текущего местоположения — интереснее, чем можно было бы ожидать. Она требует интеграции с Google Play Services — семейством библиотек Google, не входящих в стандартный набор.

Местоположение и библиотеки

Чтобы такое положение дел стало более понятным, немного поговорим о том, какую информацию может получать среднее устройство Android и какие инструменты предоставляет Android для получения этой информации.

В стандартной комплектации Android предоставляет базовый вариант Location API, позволяющий получать данные местоположения от разных источников. На большинстве телефонов такими источниками являются точные данные от приемника GPS и приближенные данные от вышек сотовой связи или сетей WiFi. Такие API существуют с первых версий Android. Вы найдете их в пакете `android.location`.

Да, API `android.location` существуют, но они не идеальны. Реальные приложения выдают запросы типа «Я хочу получить максимум точности, сколько бы заряда аккумулятора для этого ни потребовалось» или «Я хочу получить данные местоположения, но по возможности сэкономить заряд». Запросы вида «Запустите приемник GPS и передайте полученные от него данные» встречаются крайне редко.

При перемещении устройств такой подход начинает создавать проблемы. Если вы находитесь «на природе» — GPS лучше всего. Если сигнал GPS недоступен, то решение с данными от вышек сотовой связи может быть оптимальным. Если ни один из этих сигналов не доступен, даже позиционирование с акселерометром и гироскопом все же лучше, чем полное отсутствие данных.

В прошлом качественным приложениям приходилось специально подписываться на все эти разнородные источники данных и переключаться между ними по мере надобности. Такое решение не назовешь ни простым, ни удобным.

Google Play Services

Возникла необходимость в более совершенных API. Но если бы они были добавлены в стандартную библиотеку, то в лучшем случае прошла бы пара лет, прежде чем все разработчики смогли использовать их. И это было неприятно, потому что у ОС было все необходимое для улучшенных API: GPS, приближенное позиционирование и т. д.

К счастью, стандартная библиотека не единственный способ использования кода. Кроме стандартной библиотеки, Google предоставляет Play Services — набор стандартных сервисных функций, устанавливаемых вместе с приложением магазина Google Play. Для решения проблемы позиционирования компания Google опубликовала в Play Services новую версию сервиса позиционирования Fused Location Provider.

Так как эти библиотеки существуют в другом приложении, это приложение должно быть установлено в системе. Это означает, что приложение может использоваться только на устройствах с установленным и обновленным приложением Play Store. Также это почти наверняка означает, что приложение будет распространяться через Play Store. Если же ваше приложение недоступно через Play Store, значит, вам не повезло и стоит подыскать другой API.

Если вы будете тестировать приложение этой главы на физическом устройстве, убедитесь в том, что на нем установлено обновленное приложение Play Store.

А если приложение тестируется в эмуляторе? Не тревожьтесь — мы рассмотрим эту тему далее в этой главе.

Создание Locatr

Пора браться за дело. Создайте в Android Studio новый проект с именем Locatr. Присвойте главной активности имя `LocatrActivity`. Как и в других приложениях, задайте параметр `minSdkVersion` равным 19 и скопируйте класс `SingleFragmentActivity` с activity_fragment.xml.

Также вам понадобится дополнительный код из PhotoGallery. Мы снова будем обращаться с запросами к Flickr, и готовый код упростит задачу. Откройте решение PhotoGallery (подойдет любая версия после главы 26), выделите файлы FlickrFetchr.java и GalleryItem.java, щелкните правой кнопкой мыши и скопируйте их. Вставьте файлы в раздел кода Java в Locatr.

Вскоре мы возьмемся за построение пользовательского интерфейса. Если вы работаете с эмулятором, прочитайте следующий раздел, чтобы вы могли протестировать весь код, который мы собираемся написать. Если нет — можете переходить сразу к разделу «Построение интерфейса Locatr».

Play Services и тестирование в эмуляторах

Если вы используете эмулятор AVD, сначала убедитесь в том, что образы эмулятора обновлены.

Для этого откройте SDK Manager (Tools ▸ Android ▸ SDK Manager), перейдите к версии Android, которую вы собираетесь использовать для своего эмулятора, и убедитесь в том, что образы (Google APIs System Image) установлены и актуальны (рис. 33.2). Если для образа доступно обновление, щелкните на кнопке, чтобы установить его, и дождитесь завершения установки.

Эмулятор AVD также должен иметь целевую версию ОС с поддержкой Google APIs. При создании эмулятора эти целевые версии ОС можно отличить по строке «Google APIs» в правом столбце. Выберите версию с API уровня 21 и выше (рис. 33.3).

Если подходящий эмулятор уже имеется, но ранее вам пришлось обновить образы из SDK, возможно, эмулятор придется перезапустить.

Фиктивные позиционные данные

Для работы эмулятора вам понадобятся фиктивные обновления позиционных данных. В Android Studio имеется панель управления эмулятора, с которой можно передавать эмулятору позиционные данные. Такое решение отлично работает для старых средств позиционирования, но совершенно не поможет для нового механизма Fused Location Provider. Фиктивные позиционные данные придется публиковать на программном уровне.

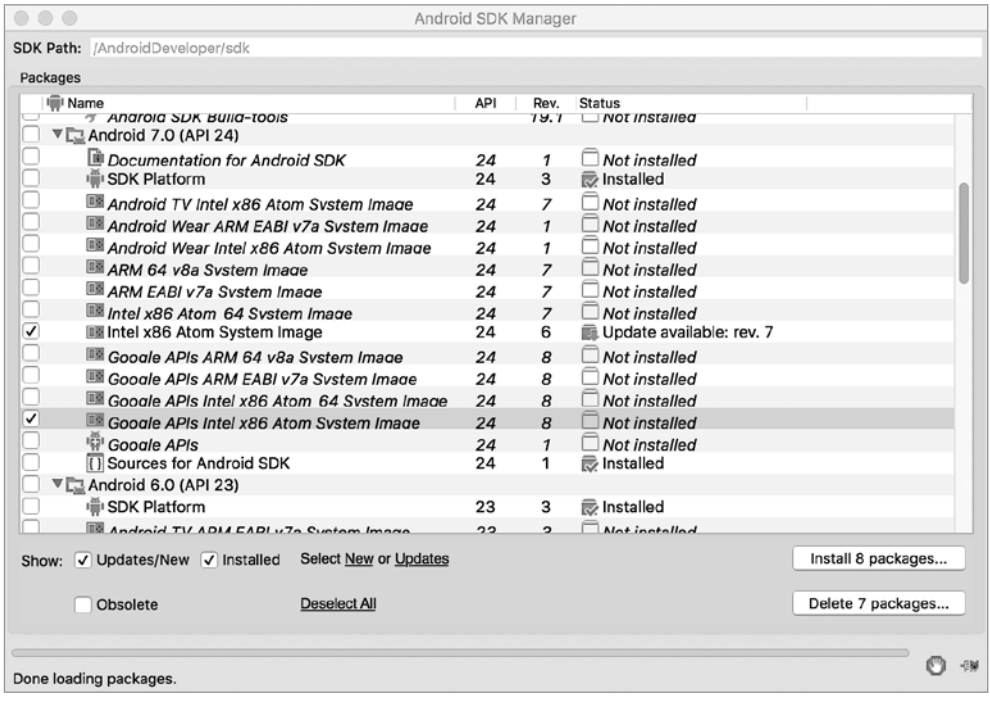


Рис. 33.2. Проверка обновления эмулятора

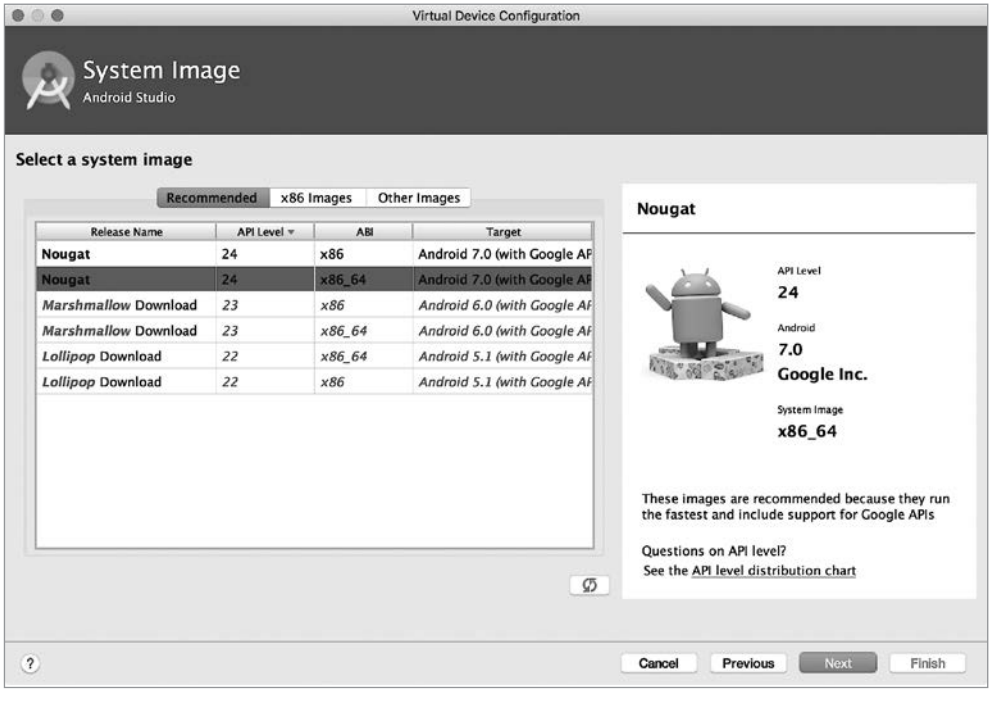


Рис. 33.3. Выбор образа с поддержкой Google APIs

Мы, сотрудники Big Nerd Ranch, любим объяснять интересные темы во всех подробностях. Тем не менее на этот раз вместо того, чтобы объяснять все тонкости кода, генерирующего фиктивные позиционные данные, мы написали для вас отдельное приложение MockWalker. Чтобы использовать его, загрузите и установите APK по следующему URL-адресу: *https://www.bignerdranch.com/solutions/MockWalker.apk*.

Для этого проще всего открыть браузер в эмуляторе и ввести URL (рис. 33.4).

Когда это будет сделано, коснитесь элемента оповещения загрузки на панели инструментов, чтобы открыть APK (рис. 33.5).

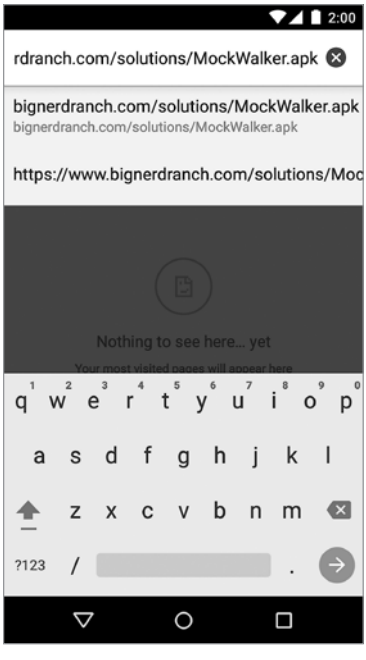


Рис. 33.4. Ввод URL-адреса

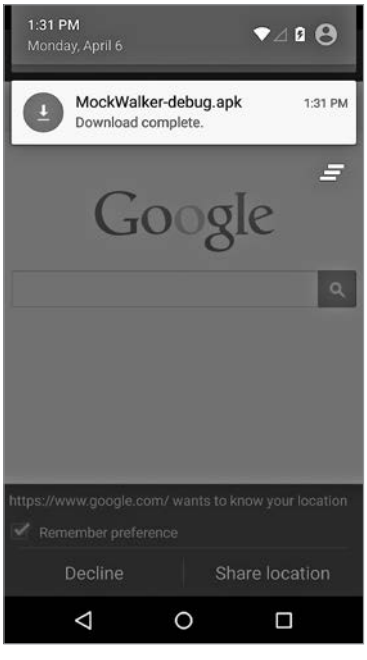


Рис. 33.5. Открытие загруженного пакета

MockWalker использует службу для отправки фиктивных позиционных данных Fused Location Provider. Приложение имитирует перемещение по кругу в окрестностях Кирквуда (штат Атланта). Пока служба работает, каждый раз, когда приложение Locatr запрашивает у Fused Location Provider данные позиционирования, оно получает данные от MockWalker.

Запустите MockWalker и нажмите кнопку Start (рис. 33.6). Служба начинает работу после выхода из приложения. (Не закрывайте эмулятор — он должен выполняться, пока вы работаете над Locatr.) Когда необходимость в фиктивных данных отпадет, снова откройте MockWalker и нажмите кнопку Stop.

Если вы хотите знать, как работает приложение MockWalker, просмотрите исходный код в папке решений этой главы (также см. раздел «Добавление значка» главы 2). В нем встречается ряд интересных аспектов: RxJava и закрепляемые

Рис. 33.6. Выполнение MockWalker

(sticky) службы для управления обновлениями позиционных данных. Если вас заинтересует эта тема, изучите ее самостоятельно.

Построение интерфейса Locatr

Перейдем к созданию интерфейса. Начните с добавления строки с текстом кнопки поиска в файл res/values/strings.xml.

Листинг 33.1. Добавление текста кнопки поиска (res/values/strings.xml)

```
<resources>
    <string name="app_name">Locatr</string>

    <string name="search">Find an image near you</string>
</resources>
```

Как обычно, в приложении будет использоваться фрагмент, поэтому переименуйте activity_locatr.xml в fragment_locatr.xml. Включите в `RelativeLayout` виджет `ImageView` для отображения найденного изображения (рис. 33.7). (Значения атрибутов `padding` генерируются шаблонным кодом на момент написания книги. Они не важны, оставьте или удалите их на свое усмотрение.)

Также понадобится кнопка для включения поиска; ее можно разместить на панели инструментов. Создайте файл res/menu/fragment_locatr.xml и добавьте элемент

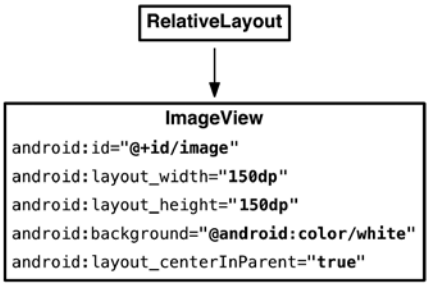


Рис. 33.7. Макет Locatr (res/layout/fragment_locatr.xml)

меню для отображения значка позиционирования. (Да, файл с таким же именем, как у res/layout/fragment_locatr.xml. Никаких проблем это не создает: ресурсы меню существуют в другом пространстве имен.)

Листинг 33.2. Настройка меню Locatr (res/menu/fragment_locatr.xml)

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item android:id="@+id/action_locate"
        android:icon="@android:drawable/ic_menu_compass"
        android:title="@string/search"
        android:enabled="false"
        app:showAsAction="ifRoom"/>
</menu>
```

По умолчанию кнопка блокируется в разметке XML. Позднее блокировка снимается при подключении к Play Services.

Теперь создайте субкласс `Fragment` с именем `LocatrFragment`, который связывается с макетом и получает ссылку на `ImageView`.

Листинг 33.3. Создание LocatrFragment (LocatrFragment.java)

```
public class LocatrFragment extends Fragment {
    private ImageView mImageView;

    public static LocatrFragment newInstance() {
        return new LocatrFragment();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_locatr, container, false);

        mImageView = (ImageView) v.findViewById(R.id.image);

        return v;
    }
}
```

Также подключите элемент меню. Сохраните ссылку на него в отдельной переменной экземпляра, чтобы снять блокировку с кнопки в будущем.

Листинг 33.4. Добавление меню в фрагмент (LocatrFragment.java)

```
public class LocatrFragment extends Fragment {
    private ImageView mImageView;

    public static LocatrFragment newInstance() {
        return new LocatrFragment();
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setHasOptionsMenu(true);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        ...
    }

    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
        super.onCreateOptionsMenu(menu, inflater);
        inflater.inflate(R.menu.fragment_locatr, menu);
    }
}
```

Свяжите все компоненты в `LocatrActivity`. Удалите весь текущий код класса и замените его:

Листинг 33.5. Подключение фрагмента Locatr (LocatrActivity.java)

```
public class LocatrActivity extends SingleFragmentActivity {
    @Override
    protected Fragment createFragment() {
        return LocatrFragment.newInstance();
    }
}
```

Теперь все готово к предстоящим испытаниям.

Настройка Google Play Services

Для получения позиционных данных от Fused Location Provider необходимо использовать Google Play Services. Для этого следует выполнить несколько стандартных действий.

Прежде всего добавьте зависимость для библиотеки Google Play Services. Сами службы находятся в приложении Play, но библиотека Play Services содержит весь код взаимодействия с ними.

Откройте окно настроек модуля app (File ▸ Project Structure). Перейдите к списку зависимостей и добавьте зависимость для библиотеки. Введите следующее имя зависимости: com.google.android.gms:play-services-location:10.0.1. (На момент написания книги эта зависимость не отображалась в результатах поиска, так что будьте внимательны при вводе.) Здесь находится позиционная часть Play Services.

Со временем номер версии библиотеки изменится. Если вы хотите узнать последнюю версию, проведите поиск библиотечных зависимостей для `play-services`. В результатах появится зависимость `com.google.android.gms:play-services` с номером версии. Эта зависимость включает все содержимое Play Services. Если вы хотите использовать новейшую версию библиотеки, номер версии из `play-services` может использоваться и в более ограниченной библиотеке `play-services-location`.

Какой же номер версии следует использовать вам? По нашему опыту, всегда лучше выбрать самую последнюю из всех существующих версий. Но мы не можем гарантировать, что код из этой главы будет так же работать и в будущих версиях. Итак, в этой главе лучше использовать ту версию, для которой был написан наш код: 10.0.1.

Затем следует проверить доступность Play Services. Так как все рабочие компоненты находятся в другом приложении на вашем устройстве, работоспособность библиотеки Play Services не гарантирована. Сама библиотека позволяет легко выполнить необходимую проверку. Внесите соответствующее изменение в главную активность:

Листинг 33.6. Добавление проверки Play Services (LocatrActivity.java)

```
public class LocatrActivity extends SingleFragmentActivity {
    private static final int REQUEST_ERROR = 0;

    @Override
    protected Fragment createFragment() {
        return LocatrFragment.newInstance();
    }

    @Override
    protected void onResume() {
        super.onResume();

        GoogleApiAvailability apiAvailability = GoogleApiAvailability.
getInstance();
        int errorCode = apiAvailability.isGooglePlayServicesAvailable(this);

        if (errorCode != ConnectionResult.SUCCESS) {
            Dialog errorDialog = apiAvailability
                    .getErrorDialog(this, errorCode, REQUEST_ERROR,
                            new DialogInterface.OnCancelListener() {

                                @Override
```

```
                                public void onCancel(DialogInterface dialog) {
                                    // Выйти, если сервис недоступен.
                                    finish();
                                }
                            });

            errorDialog.show();
        }
    }
}
```

В данном случае защищаться от проблем с поворотами не нужно. Значение `errorCode` останется неизменным и при повороте, так что диалоговое окно появится снова.

Разрешения

Также для работы приложения необходимо добавить некоторые разрешения. Для нас важны два разрешения: `android.permission.ACCESS_FINE_LOCATION` и `android.permission.ACCESS_COARSE_LOCATION`. Первое соответствует получению данных от приемника GPS, а второе — получению данных от вышек сотовой связи или точек доступа WiFi.

В этой главе будут запрашиваться позиционные данные высокой точности, поэтому нам определенно потребуется разрешение `ACCESS_FINE_LOCATION`. Однако заодно будет полезно запросить и `ACCESS_COARSE_LOCATION` — если источник точных данных недоступен, по крайней мере у вас будет возможность переключиться на запасной источник приближенных данных.

Включите эти разрешения в манифест. Заодно добавьте и разрешение на доступ к Интернету, чтобы обращаться с запросами к Flickr.

Листинг 33.7. Добавление разрешений (AndroidManifest.xml)

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bignerdranch.android.locatr" >

    <uses-permission
        android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission
        android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission
        android:name="android.permission.INTERNET" />
    ...
</manifest>
```

И `FINE_LOCATION`, и `COARSE_LOCATION` — небезопасные разрешения. Это означает, что, в отличие от разрешений `INTERNET`, кода в манифесте недостаточно: также необходимо подтвердить их запросом на стадии выполнения.

Для решения этой проблемы позднее в этой главе мы напишем код, связанный с разрешениями. А пока перейдем к интеграции Google Play Services.

Использование Google Play Services

Чтобы использовать Play Services, необходимо создать клиента — экземпляр класса `GoogleApiClient`. Документация по этому классу (и всем остальным классам Play Services, которые будут использоваться в этих двух главах) представлена в справочном разделе Play Services: *developer.android.com/reference/gms-packages.html*.

Чтобы создать клиента, создайте экземпляр `GoogleApiClient.Builder` и настройте его. Как минимум необходимо включить в экземпляр информацию о конкретных API, которые вы собираетесь использовать. Затем вызовите `build()` для создания экземпляра.

В методе `onCreate(Bundle)` создайте экземпляр `GoogleApiClient.Builder` и добавьте в него Location Services API.

Листинг 33.8. Создание GoogleApiClient (LocatrFragment.java)

```
public class LocatrFragment extends Fragment {
    private ImageView mImageView;
    private GoogleApiClient mClient;

    public static LocatrFragment newInstance() {
        return new LocatrFragment();
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setHasOptionsMenu(true);

        mClient = new GoogleApiClient.Builder(getActivity())
                .addApi(LocationServices.API)
                .build();
    }
```

Когда клиент будет создан, к нему необходимо подключиться. Google рекомендует всегда подключаться к клиенту в методе `onStart()` и отключаться в `onStop()`. Вызов `connect()` для клиента также изменит возможности кнопки меню, поэтому мы вызовем `invalidateOptionsMenu()` для обновления ее визуального состояния. (Позднее этот метод будет вызван еще один раз: после того, как мы получим информацию о создании подключения.)

Листинг 33.9. Подключение и отключение (LocatrFragment.java)

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    ...
}

@Override
public void onStart() {
    super.onStart();

    getActivity().invalidateOptionsMenu();
    mClient.connect();
```

```
}

@Override
public void onStop() {
    super.onStop();

    mClient.disconnect();
}

@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
```

Если клиент не подключен, приложение ничего не сможет сделать. Соответственно, на следующем шаге мы устанавливаем или снимаем блокировку кнопки в зависимости от состояния подключения клиента.

Листинг 33.10. Обновление кнопки меню (LocatrFragment.java)

```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_locatr, menu);

    MenuItem searchItem = menu.findItem(R.id.action_locate);
    searchItem.setEnabled(mClient.isConnected());
}
```

Добавьте еще один вызов `getActivity().invalidateOptionsMenu()` для обновления состояния элемента меню при получении информации о подключении. Информация о состоянии подключения передается через два интерфейса обратного вызова: `ConnectionCallbacks` и `OnConnectionFailedListener`. Назначьте слушателя `ConnectionCallbacks` в `onCreate(Bundle)`, чтобы перерисовать панель инструментов при подключении.

Листинг 33.11. Прослушивание событий подключения (LocatrFragment.java)

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setHasOptionsMenu(true);

    mClient = new GoogleApiClient.Builder(getActivity())
            .addApi(LocationServices.API)
            .addConnectionCallbacks(new GoogleApiClient.ConnectionCallbacks() {
                @Override
                public void onConnected(Bundle bundle) {
                    getActivity().invalidateOptionsMenu();
                }

                @Override
                public void onConnectionSuspended(int i) {
                }
            })
            .build();
}
```

Если вам интересно, подключите `OnConnectionFailedListener` и посмотрите, о чем он сообщит. Впрочем, это не обязательно.

После всего, что было сделано, подготовка к использованию Google Play Services завершена.

Геопоиск Flickr

Следующим шагом станет возможность поиска географического местоположения на сайте Flickr. Для этого проводится обычный поиск, но с указанием широты и долготы.

В Android эта информация передается позиционным API в объектах Location. Напишите новое переопределение `buildUrl(…)`, которое получает один из таких объектов `Location` и строит соответствующий поисковый запрос.

Листинг 33.12. Новая версия buildUrl(Location) (FlickrFetchr.java)

```
private String buildUrl(String method, String query) {
    ...
}

private String buildUrl(Location location) {
    return ENDPOINT.buildUpon()
            .appendQueryParameter("method", SEARCH_METHOD)
            .appendQueryParameter("lat", "" + location.getLatitude())
            .appendQueryParameter("lon", "" + location.getLongitude())
            .build().toString();
}
```

Затем напишите соответствующий метод `searchPhotos(Location)`.

Листинг 33.13. Новая версия searchPhotos(Location) (FlickrFetchr.java)

```
public List<GalleryItem> searchPhotos(String query) {
    ...
}

public List<GalleryItem> searchPhotos(Location location) {
    String url = buildUrl(location);
    return downloadGalleryItems(url);
}
```

Получение позиционных данных

Итак, все готово к получению позиционных данных. Для взаимодействия с Fused Location Provider API используется класс с подходящим именем `FusedLocationProviderApi`. Этот класс наличествует в единственном экземпляре: это синглетный объект, существующий в LocationServices под именем `FusedLocationApi`.

Чтобы получить позиционные данные от API, необходимо построить запрос. Запросы к Fused Location представляются объектами `LocationRequest`. Создайте такой объект и настройте его в новом методе с именем `findImage()`. (Существуют два разных класса `LocationRequest`; используйте версию с полным именем `com.google.android.gms.location.LocationRequest`.)

Листинг 33.14. Построение запроса позиционных данных (LocatrFragment.java)

```
    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
        ...
    }

    private void findImage() {
        LocationRequest request = LocationRequest.create();
        request.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
        request.setNumUpdates(1);
        request.setInterval(0);
    }
}
```

Объекты `LocationRequest` задают разнообразные параметры запроса:

- *интервал* — как часто должны обновляться позиционные данные;
- *количество обновлений* — сколько раз должны обновляться позиционные данные;
- *приоритет* — как следует поступать в ситуации выбора между расходом заряда аккумулятора и точностью выполнения запроса;
- *срок действия* — ограничен ли срок действия запроса, и если да, когда он истекает;
- *минимальное смещение* — при каком минимальном смещении устройства (в метрах) должно инициироваться обновление местоположения.

При исходном создании объект `LocationRequest` настраивается с точностью в пределах городского квартала и бесконечными медленными обновлениями. В коде мы переключимся в режим однократного получения высокоточных позиционных данных, изменяя приоритет и количество обновлений. Интервалу будет присвоено значение 0, показывающее, что обновление позиционных данных должно происходить как можно быстрее.

Следующий шаг — отправка запроса и прослушивание возвращаемых объектов `Location`. Для этого будет добавлен объект `LocationListener`. Существуют две версии `LocationListener`, которые вы можете импортировать; выберите версию `com.google.android.gms.location.LocationListener`. Добавьте в `findImage()` еще один вызов метода.

Листинг 33.15. Отправка LocationRequest (LocatrFragment.java)

```
public class LocatrFragment extends Fragment {
    private static final String TAG = "LocatrFragment";
    ...
    private void findImage() {
```

```
        LocationRequest request = LocationRequest.create();
        request.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
        request.setNumUpdates(1);
        request.setInterval(0);
        LocationServices.FusedLocationApi
                .requestLocationUpdates(mClient, request, new LocationListener() {
                    @Override
                    public void onLocationChanged(Location location) {
                        Log.i(TAG, "Got a fix: " + location);
                    }
                });
    }
```

Для запроса с более долгим сроком жизни следовало бы сохранить слушателя и позднее вызвать `removeLocationUpdates(…)` для отмены запроса. Но поскольку мы вызвали `setNumUpdates(1)`, достаточно отправить запрос и забыть о нем.

(Вызов `requestLocationUpdates(…)` будет помечен красным индикатором ошибки. Вызовет ли это проблемы? Да, вызовет – но пока не обращайте внимания; вскоре мы вернемся к проблеме.)

Наконец, для отправки запроса следует связать его с кнопкой поиска. Переопределите метод `onOptionsItemSelected(…)` и добавьте в него вызов `findImage()`.

Листинг 33.16. Подключение кнопки поиска (LocatrFragment.java)

```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    ...
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_locate:
            findImage();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Запустите приложение и нажмите кнопку поиска. В этот момент заявляет о себе красный индикатор ошибки: на экране появляется сообщение об остановке Locatr (рис. 33.8).

Проверка вывода LogCat показывает, что в ходе выполнения было выдано исключение `SecurityException`:

```
FATAL EXCEPTION: main
Process: com.bignerdranch.android.locatr, PID: 7892
java.lang.SecurityException: Client must have ACCESS_FINE_LOCATION permission to
request PRIORITY_HIGH_ACCURACY locations.
    at android.os.Parcel.readException(Parcel.java:1684)
    at android.os.Parcel.readException(Parcel.java:1637)
```

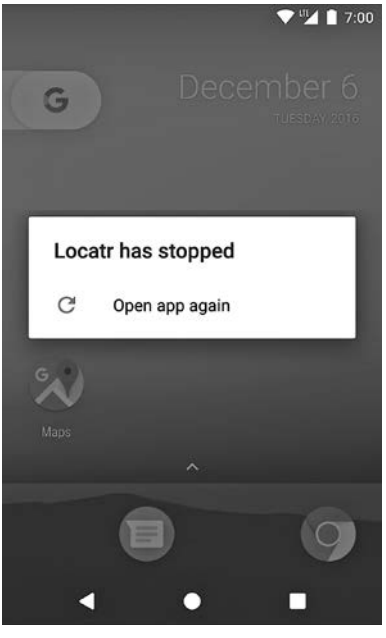


Рис. 33.8. Разрешение отсутствует

```
...
at com.google.android.gms.location.internal.zzd
    .requestLocationUpdates(Unknown Source)
at com.bignerdranch.android.locatr.LocatrFragment
    .findImage(LocatrFragment.java:102)
at com.bignerdranch.android.locatr.LocatrFragment
    .onOptionsItemSelected(LocatrFragment.java:89)
at android.support.v4.app.Fragment.performOptionsItemSelected
                  (Fragment.java:2212)
...
at java.lang.reflect.Method.invoke(Native Method)
at com.android.internal.os.ZygoteInit$MethodAndArgsCaller
    .run(ZygoteInit.java:886)
at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:776)
```

Чтобы вызов `requestLocationUpdates(…)` успешно работал, сначала необходимо запросить разрешение.

Запрос разрешения во время выполнения

Для работы с разрешениями на стадии выполнения необходимо сделать три вещи:

- проверить наличие разрешения перед выполнением операции;
- запросить разрешение, если это не было сделано ранее;
- прослушать ответ на запрос разрешения.

При запросе разрешения на экране отображается стандартный интерфейс для получения разрешения. Если диалоговое окно отображается в точке приложения, в которой пользователю будет понятно, что происходит, то дополнительно делать ничего не придется.

Впрочем, иногда причина не столь очевидна. Пользователи часто отклоняют непонятные запросы, поэтому в таких случаях приходится выводить пояснение. Для таких приложений потребуется четвертый шаг:

- проверить, нужно ли в вашем приложении вывести пояснение для пользователя.

Так как задача Locatr вполне очевидна, выводить пояснение не нужно — по крайней мере до выполнения упражнения в конце главы.

Проверка разрешений

Первым шагом должно стать включение информации о разрешениях в код Java. Для этого добавьте в начало `LocatrFragment` массив констант со списком всех необходимых разрешений.

Листинг 33.17. Добавление констант разрешений (LocatrFragment.java)

```
public class LocatrFragment extends Fragment {
    private static final String TAG = "LocatrFragment";
    private static final String[] LOCATION_PERMISSIONS = new String[]{
            Manifest.permission.ACCESS_FINE_LOCATION,
            Manifest.permission.ACCESS_COARSE_LOCATION,
    };

    private ImageView mImageView;
    private GoogleApiClient mClient;
```

Все стандартные разрешения Android объявляются в классе `Manifest.permission` для использования в программном коде. Эти две константы обозначают те же строковые значения, которые вы использовали в файле AndroidManifest.xml. После объявления массива разрешений следующим шагом должен стать запрос необходимых разрешений.

Небезопасные разрешения предоставляются на уровне групп, а не на уровне отдельных разрешений. Группа содержит несколько разрешений, относящихся к одной области. Например, разрешения `ACCESS_FINE_LOCATION` и `ACCESS_COARSE_LOCATION` входят в группу `LOCATION`.

Таблица 33.1. Группы разрешений

Группа	Разрешения
CALENDAR	READ_CALENDAR, WRITE_CALENDAR
CAMERA	CAMERA
CONTACTS	READ_CONTACTS, WRITE_CONTACTS, GET_ACCOUNTS

Группа	Разрешения
LOCATION	ACCESS_FINE_LOCATION, ACCESS_COARSE_LOCATION
MICROPHONE	RECORD_AUDIO
PHONE	READ_PHONE_STATE, CALL_PHONE, READ_CALL_LOG, WRITE_CALL_LOG, ADD_VOICEMAIL, USE_SIP, PROCESS_OUTGOING_CALLS
SENSORS	BODY_SENSORS
SMS	SEND_SMS, RECEIVE_SMS, READ_SMS, RECEIVE_WAP_PUSH, RECEIVE_MMS
STORAGE	READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE

Разрешение предоставляется не для отдельных элементов, а сразу для всей группы. Так как оба наших разрешения входят в группу `LOCATION`, достаточно проверить наличие только одного из них.

Напишите метод для проверки доступности первого разрешения в массиве `LOCATION_PERMISSIONS`.

Листинг 33.18. Проверка разрешения (LocatrFragment.java)

```
    private void findImage() {
        ...
    }

    private boolean hasLocationPermission() {
        int result = ContextCompat
                .checkSelfPermission(getActivity(), LOCATION_PERMISSIONS[0]);
        return result == PackageManager.PERMISSION_GRANTED;
    }
}
```

Так как метод `checkSelfPermission(…)` для `Activity` появился относительно недавно (в Marshmallow), используйте версию `checkSelfPermission(…)` из `ContextCompat`, чтобы избежать громоздкого кода с проверкой условий. Она обеспечит совместимость автоматически.

Добавьте перед вызовом `findImage()` проверку наличия разрешений.

Листинг 33.19. Добавление проверки разрешения (LocatrFragment.java)

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_locate:
            if (hasLocationPermission()) {
                findImage();
            }
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

А теперь следует обработать ситуацию с *отсутствием* разрешения. В таком случае следует вызвать метод `requestPermissions(…)`.

Листинг 33.20. Запрос разрешения (LocatrFragment.java)

```
public class LocatrFragment extends Fragment {
    private static final String TAG = "LocatrFragment";
    private static final String[] LOCATION_PERMISSIONS = new String[]{
            Manifest.permission.ACCESS_FINE_LOCATION,
            Manifest.permission.ACCESS_COARSE_LOCATION,
    };
    private static final int REQUEST_LOCATION_PERMISSIONS = 0;
    ...
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.action_locate:
                if (hasLocationPermission()) {
                    findImage();
                } else {
                    requestPermissions(LOCATION_PERMISSIONS,
                            REQUEST_LOCATION_PERMISSIONS);
                }
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }
```

Метод `requestPermissions(…)` выполняется как собой асинхронный запрос. При его вызове Android отображает системное диалоговое окно разрешений с сообщением, соответствующим запрашиваемому разрешению.

Для обработки реакции на системное диалоговое окно вы пишете соответствующую реализацию `onRequestPermissionsResult(…)`. Android вызывает этот метод обратного вызова при нажатии пользователем кнопки ALLOW или DENY. Напишите реализацию, которая снова проверяет разрешение и вызывает `findImage()`, если разрешение было предоставлено.

Листинг 33.21. Обработка результата запроса разрешения (LocatrFragment.java)

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    ...
}

@Override
public void onRequestPermissionsResult(int requestCode, String[] permissions,
                                       int[] grantResults) {
    switch (requestCode) {
        case REQUEST_LOCATION_PERMISSIONS:
            if (hasLocationPermission()) {
                findImage();
```

```
            }
        default:
            super.onRequestPermissionsResult(requestCode, permissions,
                                             grantResults);
    }
}

private void findImage() {
    LocationRequest request = LocationRequest.create();
    request.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
    request.setNumUpdates(1);
```

Обратный вызов `onRequestPermissionsResult(int,String[],int[])` включает параметр с именем `grantResults`. При желании можно проверить этот параметр, чтобы узнать, было ли предоставлено разрешение.

Мы воспользуемся более простым способом: когда вы вызываете `hasLocationPermission()`, вызов `checkSelfPermission(…)` также сообщает, было ли получено разрешение. Итак, повторный вызов `hasLocationPermission()` позволяет добиться нужного результата с меньшим объемом кода, чем при проверке содержимого `grantResults`.

Запустите приложение Locatr и снова выберите элемент меню. На этот раз появится системное диалоговое окно разрешений (рис. 33.9).

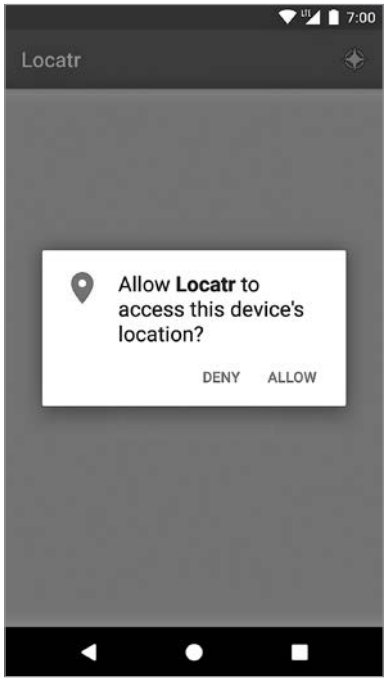


Рис. 33.9. Диалоговое окно группы разрешений LOCATION

Если нажать кнопку ALLOW, то разрешение будет предоставлено до момента удаления приложения или отключения устройства. Если нажать кнопку DENY, то разрешение отклоняется временно — при следующем нажатии кнопки диалоговое окно появится снова. (В ходе отладки мы предпочитаем сбрасывать состояние разрешений, удаляя приложение.)

Когда все будет сделано, убедитесь в том, что поиск работает так, как ожидалось. Не забудьте запустить MockWalker, если приложение выполняется в эмуляторе. (Если у вас возникнут проблемы с меню, вернитесь к главе 13 за информацией об интеграции библиотеки AppCompat.) В журнале появится строка следующего вида:

```
...D/libEGL: loaded /system/lib/egl/libGLESv2_MRVL.so
...D/GC: <tid=12423> OES20 ===> GC Version   : GC Ver rls_pxa988_KK44_GC13.24
...D/OpenGLRenderer: Enabling debug mode 0
...I/LocatrFragment: Got a fix: Location[fused 33.758998,-84.331796 acc=38 et=...]
```

В переданной информации содержится широта и долгота, точность и оценка времени получения позиционных данных. Передав пару «широта/долгота» Google Maps, вы увидите свое текущее местоположение на карте (рис. 31.10).

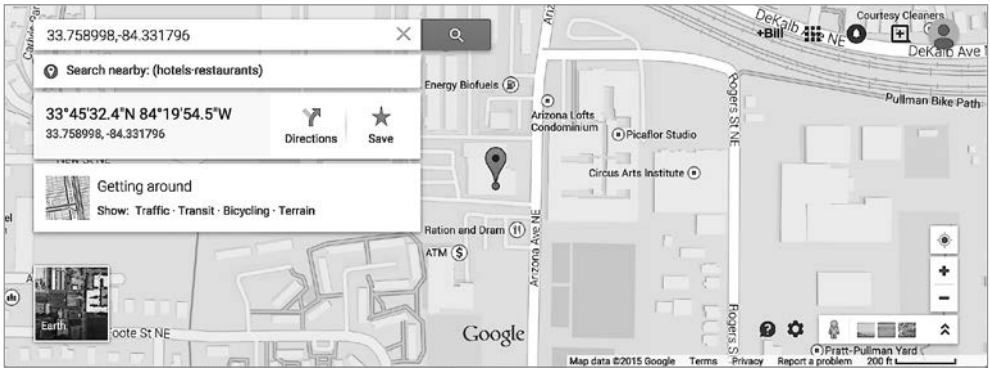


Рис. 33.10. Текущее местоположение

Поиск и вывод изображений

Позиционные данные успешно получены, теперь их нужно использовать. Напишите задачу `AsyncTask` для получения объекта `GalleryItem` поблизости от текущей позиции, загрузки связанного с ним изображения и вывода его в приложении.

Новый код размещается в новом внутреннем классе `AsyncTask` с именем `SearchTask`. Начните с выполнения поиска и выбора первого полученного объекта `GalleryItem`.

Листинг 33.22. Реализация SearchTask (LocatrFragment.java)

```
private void findImage() {
    ...
    LocationServices.FusedLocationApi
            .requestLocationUpdates(mClient, request, new LocationListener() {
```

```
                @Override
                public void onLocationChanged(Location location) {
                    Log.i(TAG, "Got a fix: " + location);
                    new SearchTask().execute(location);
                }
            });
}

private boolean hasLocationPermission() {
    int result = ContextCompat
            .checkSelfPermission(getActivity(), LOCATION_PERMISSIONS[0]);
    return result == PackageManager.PERMISSION_GRANTED;
}

private class SearchTask extends AsyncTask<Location,Void,Void> {
    private GalleryItem mGalleryItem;

    @Override
    protected Void doInBackground(Location... params) {
        FlickrFetchr fetchr = new FlickrFetchr();
        List<GalleryItem> items = fetchr.searchPhotos(params[0]);

        if (items.size() == 0) {
            return null;
        }

        mGalleryItem = items.get(0);

        return null;
    }
}
```

Сохранение `GalleryItem` в переменной экземпляра пока ничего не дает. Тем не менее оно избавит вас от лишней работы в следующей главе.

Затем загрузите данные изображения, связанные с `GalleryItem`, и декодируйте их. Выведите изображение в `mImageView` в методе `onPostExecute(Void)`.

Листинг 33.23. Загрузка и вывод изображения (LocatrFragment.java)

```
private class SearchTask extends AsyncTask<Location,Void,Void> {
    private GalleryItem mGalleryItem;
    private Bitmap mBitmap;

    @Override
    protected Void doInBackground(Location... params) {
        ...
        mGalleryItem = items.get(0);

        try {
            byte[] bytes = fetchr.getUrlBytes(mGalleryItem.getUrl());
            mBitmap = BitmapFactory.decodeByteArray(bytes, 0, bytes.length);
        } catch (IOException ioe) {
            Log.i(TAG, "Unable to download bitmap", ioe);
        }
```

```
        return null;
    }

    @Override
    protected void onPostExecute(Void result) {
        mImageView.setImageBitmap(mBitmap);
    }
}
```

После этого приложение будет успешно находить ближайшее изображение на сайте Flickr (рис. 33.11). Запустите Locatr и нажмите кнопку поиска.

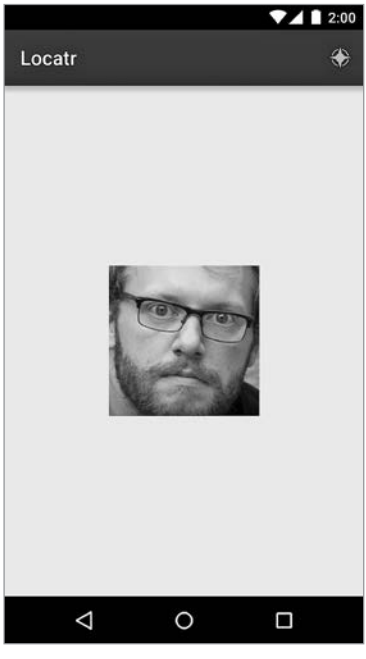


Рис. 33.11. Результат

Упражнение: обоснование разрешений

Как упоминалось выше, системное диалоговое окно разрешений во многих ситуациях недостаточно содержательно. В таких случаях следует объяснить пользователю, для чего запрашивается разрешение.

В системе Android существует рекомендованная последовательность действий для вывода таких пояснений. На момент написания книги она выглядела так:

1. При первом запросе пользователя выводится системное диалоговое окно.
2. При всех последующих запросах выводится диалоговое окно с вашим пояснением, а потом снова системное диалоговое окно.

3. Если пользователь потребует отказать в разрешении навсегда, ни пояснение, ни системное диалоговое окно никогда более не выводится.

На первый взгляд схема выглядит излишне сложной. К счастью, Android предоставляет метод, упрощающий ее реализацию: `shouldShowRequestPermissionRationale(…)` из класса `ActivityCompat`.

Метод `shouldShowRequestPermissionRationale(…)` возвращает `false` перед тем, как пользователь впервые запросит разрешение, `true` после первого отказа и затем `false`, если пользователь решит навсегда отказать в предоставлении разрешения.

Реализуйте `DialogFragment` для вывода короткого сообщения: «Locatr использует позиционные данные для поиска близлежащих изображений на Flickr». Используйте метод `shouldShowRequestPermissionRationale(…)` для проверки того, нужно ли выводить пояснение, перед вызовом `requestPermission(…)`. Если пояснение находится на экране, приложение Locatr должно дождаться, пока оно будет закрыто пользователем, прежде чем вызывать `requestPermission(…)`. (Подсказка: для проверки закрытия можно переопределить `DialogFragment.onCancel(…)`.)

Упражнение. Индикатор прогресса

Было бы неплохо дополнить пользовательский интерфейс этого простого приложения обратной связью. Пока при нажатии на кнопку ничто не свидетельствует о том, что в приложении происходит что-то полезное.

Измените приложение Locatr так, чтобы оно немедленно реагировало на нажатие кнопки и отображало индикатор прогресса. Класс `ProgressDialog` создает вращающийся индикатор, который отлично справляется с задачей. Также необходимо отслеживать время выполнения `SearchTask`, чтобы индикатор прогресса можно было убрать в нужный момент.

34 Карты

В этой главе мы сделаем следующий шаг в работе над LocatrFragment. Кроме поиска ближайшего изображения мы найдем его широту и долготу и нанесем местонахождение на карту.

Импортирование Play Services Maps

Прежде чем браться за работу, необходимо импортировать картографическую библиотеку. Она также входит в состав библиотек Play Services. Откройте окно структуры проекта и добавьте в модуль `app` следующую зависимость: `com.google.android.gms:play-services-maps:7.0.0`. Как и в предыдущей главе, обратите внимание на изменение фактического номера версии со временем. Используйте последний номер версии «простой» зависимости `play-services`.

Работа с картами в Android

Как бы впечатляюще ни выглядели данные, сообщающие текущее местоположение телефона, они просто-таки напрашиваются на визуальное представление. Вероятно, картографические приложения стали первым бестселлером среди приложений для смартфонов; вот почему работа с картами поддерживается в Android с самых первых дней.

Данные карт велики, сложны и требуют поддержки целой системы серверов, предоставляющих базовые картографические данные. Большая часть системы Android может существовать самостоятельно как часть Android Open Source Project. Тем не менее о картах этого сказать нельзя.

Итак, хотя карты всегда существовали в Android, они также всегда были отделены от остальных API системы Android. Текущая версия Maps API, версия 2, существует в Google Play Services наряду с Fused Location Provider. А следовательно, их использование требует выполнения тех же условий, которые были описаны в разделе «Google Play Services» главы 33: необходимо либо устройство с установленным приложением Play Store, либо эмулятор с Google APIs.

Если вы сразу перешли к этой главе из-за интереса к теме, выполните перед началом следующие действия:

1. Убедитесь в том, что устройство поддерживает Play Services.
2. Импортируйте нужную библиотеку Play Services.
3. При помощи `GoogleApiAvailability` убедитесь в том, что у вас установлена новейшая версия приложения Play Store.

Получение ключа Maps API

Использование Maps API также потребует объявления ключа API в манифесте, а для этого необходимо получить собственный ключ API. Этот ключ позволит приложению использовать картографический сервис Google.

Конкретная процедура получения ключа время от времени меняется, поэтому мы рекомендуем обратиться за инструкциями к документации Google: *developers.google.com/maps/documentation/android/start*.

На момент написания книги в этих инструкциях предлагается создать новый проект. Это выглядит довольно странно, когда у вас уже имеется полноценный проект с большим объемом кода. Вместо этого мы рекомендуем щелкнуть правой кнопкой мыши на имени пакета com.bignerdranch.android.locatr и выбрать команду New ▸ Activity ▸ Gallery..., а затем выбрать Google Maps Activity для создания новой шаблонной активности. Оставьте новой активности имя по умолчанию.

После выполнения этих инструкций в приложении появится новый код: несколько новых записей в манифесте, новая строка в новом файле с именем values/google_maps_api.xml и новая активность с именем `MapsActivity`. Выполните инструкции в файле google_maps_api.xml, чтобы получить работоспособный ключ API. Строка `google_maps_key` в файле google_maps_api.xml должна выглядеть примерно так (с другим значением ключа):

```
<!-- Наш ключ цифровой подписи (у вас он будет другим) -->
<string name="google_maps_key" templateMergeStrategy="preserve"
translatable="false">
        AIzaSyClrnnYZEx0iYmJkIc0K4rdObrXcFkLl-U</string>
```

Этот ключ связан с ключом, использованным при построении приложения. Наш отладочный ключ отличается от вашего, поэтому вам он не подойдет. (Если вы потратили время на то, чтобы ввести его вручную — примите наши извинения.)

Добавление `MapsActivity` было необходимо для построения шаблонной разметки XML, но сам класс не нужен — мы адаптируем `LocatrFragment`. Удалите `MapsActivity` из приложения и исключите его запись из манифеста:

Листинг 34.1. Удаление записи MapsActivity из манифеста (AndroidManifest.xml)

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bignerdranch.android.locatr">
    ...
    <application ...>
        ...
        <meta-data
```

```
            android:name="com.google.android.geo.API_KEY"
            android:value="@string/google_maps_key"/>

        <activity
            android:name=".MapsActivity"
            android:label="@string/title_activity_maps">
        </activity>
    </application>

</manifest>
```

На момент написания книги шаблон также автоматически добавлял в набор зависимостей артефакт `com.google.android.gms:playservices`. К сожалению, гигантское количество методов в этом артефакте приведет к тому, что ваше приложение не будет строиться. (APK-файл базового приложения Android содержит не более 65 536 методов. Если вам понадобится большее количество методов, возможно и это — следует использовать специальную конфигурацию multidex, но эта тема выходит за рамки книги.)

Проблема решается удалением зависимости (но зависимости `play-services-location` и `play-services-maps` следует оставить).

Листинг 34.2. Удаление зависимости Play Services (app/build.gradle)

```
dependencies {
    compile fileTree(include: ['*.jar'], dir: 'libs')
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
    compile 'com.android.support:appcompat-v7:25.0.1'
    compile 'com.google.android.gms:play-services-location:10.0.1'
    compile 'com.google.android.gms:play-services-maps:10.0.1'
    compile 'com.google.android.gms:play-services:10.0.1'
    testCompile 'junit:junit:4.12'
}
```

На этом подготовку можно считать завершенной.

Создание карты

Предварительная настройка Maps API выполнена — теперь нужно создать карту. Карты отображаются в виджете `MapView`. Этот виджет похож на остальные виджеты представлений, не считая одного отличия: для его правильной работы необходимо вручную передавать все события жизненного цикла:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    mMapView.onCreate(savedInstanceState);
}
```

И это основательно раздражает. Намного проще поручить эту работу SDK: для этого используется класс `MapFragment` или, при использовании фрагментов из библиотеки поддержки, — `SupportMapFragment`. `MapFragment` создает `MapView` и выполняет функции хоста, включая соответствующие передачи обратных вызовов жизненного цикла.

Для начала полностью уничтожьте старый пользовательский интерфейс и замените его `SupportMapFragment`. Это не так сложно, как может показаться. Все, что от вас требуется, — переключиться на использование `SupportMapFragment`, удалить старый метод `onCreateView(…)` и удалить весь код, в котором используется ваш виджет `ImageView`.

Листинг 34.3. Переключение на SupportMapFragment (LocatrFragment.java)

```
public class LocatrFragment extends SupportMapFragment Fragment{
    private static final String TAG = "LocatrFragment";
    private static final String[] LOCATION_PERMISSIONS = new String[]{
            Manifest.permission.ACCESS_FINE_LOCATION,
            Manifest.permission.ACCESS_COARSE_LOCATION,
    };
    private static final int REQUEST_LOCATION_PERMISSIONS = 0;

    private ImageView mImageView;
    private GoogleApiClient mClient;
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_locatr, container, false);

        mImageView = (ImageView) v.findViewById(R.id.image);

        return v;
    }
    ...
    private class SearchTask extends AsyncTask<Location,Void,Void> {
        ...
        @Override
        protected void onPostExecute(Void result) {
            mImageView.setImageBitmap(mBitmap);
        }
    }
}
```

`SupportMapFragment` использует собственное переопределение `onCreateView(…)`, так что все должно быть готово. Запустите Locatr; в приложении должна появиться карта (рис. 34.1).

Получение расширенных позиционных данных

Чтобы нанести изображение на карту, необходимо знать, к какой точке оно относится. Включите дополнительный параметр `extra` в запрос Flickr API для получения пары «широта—долгота» для вашего объекта `GalleryItem`.

Рис. 34.1. Обычная карта

Листинг 34.4. Включение данных широты и долготы в запрос (FlickrFetchr.java)

```
private static final String API_KEY = "ваш_ключ_API";
private static final String FETCH_RECENTS_METHOD = "flickr.photos.getRecent";
private static final String SEARCH_METHOD = "flickr.photos.search";
private static final Uri ENDPOINT = Uri
        .parse("https://api.flickr.com/services/rest/")
        .buildUpon()
        .appendQueryParameter("api_key", API_KEY)
        .appendQueryParameter("format", "json")
        .appendQueryParameter("nojsoncallback", "1")
        .appendQueryParameter("extras", "url_s,geo")
        .build();
```

Также добавьте широту и долготу в `GalleryItem`.

Листинг 34.5. Добавление свойств для широты и долготы (GalleryItem.java)

```
public class GalleryItem {
    private String mCaption;
    private String mId;
    private String mUrl;
    private double mLat;
    private double mLon;
    ...
    public Uri getPhotoPageUri() {
        return Uri.parse("http://www.flickr.com/photos/")
```

```
                .buildUpon()
                .appendPath(mOwner)
                .appendPath(mId)
                .build();
    }

    public double getLat() {
        return mLat;
    }

    public void setLat(double lat) {
        mLat = lat;
    }

    public double getLon() {
        return mLon;
    }

    public void setLon(double lon) {
        mLon = lon;
    }

    @Override
    public String toString() {
        return mCaption;
    }
}
```

Наконец, извлеките полученные данные из ответа JSON.

Листинг 34.6. Извлечение данных из ответа Flickr JSON (FlickrFetchr.java)

```
private void parseItems(List<GalleryItem> items, JSONObject jsonBody)
        throws IOException, JSONException {

    JSONObject photosJsonObject = jsonBody.getJSONObject("photos");
    JSONArray photoJsonArray = photosJsonObject.getJSONArray("photo");

    for (int i = 0; i < photoJsonArray.length(); i++) {
        JSONObject photoJsonObject = photoJsonArray.getJSONObject(i);

        GalleryItem item = new GalleryItem();
        item.setId(photoJsonObject.getString("id"));
        item.setCaption(photoJsonObject.getString("title"));

        if (!photoJsonObject.has("url_s")) {
            continue;
        }

        item.setUrl(photoJsonObject.getString("url_s"));
        item.setOwner(photoJsonObject.getString("owner"));
        item.setLat(photoJsonObject.getDouble("latitude"));
        item.setLon(photoJsonObject.getDouble("longitude"));

        items.add(item);
    }
}
```

После того как позиционные данные будут получены, добавьте в главный фрагмент дополнительные поля для хранения текущего состояния поиска. Добавьте одно поле для объекта `Bitmap`, который будет отображаться, другое — для объекта `GalleryItem`, с которым этот объект связан, и третье — для текущей позиции `Location`.

Листинг 34.7. Добавление данных для карты (LocatrFragment.java)

```
public class LocatrFragment extends SupportMapFragment {
    ...
    private static final int REQUEST_LOCATION_PERMISSIONS = 0;
    private Bitmap mMapImage;
    private GalleryItem mMapItem;
    private Location mCurrentLocation;
    ...
```

Присвойте значения этих полей на основании данных из `SearchTask`.

Листинг 34.8. Сохранение результатов запроса (LocatrFragment.java)

```
private class SearchTask extends AsyncTask<Location,Void,Void> {
    private Bitmap mBitmap;
    private GalleryItem mGalleryItem;
    private Location mLocation;

    @Override
    protected Void doInBackground(Location... params) {
        mLocation = params[0];
        FlickrFetchr fetchr = new FlickrFetchr();
        ...
    }

    @Override
    protected void onPostExecute(Void result) {
        mMapImage = mBitmap;
        mMapItem = mGalleryItem;
        mCurrentLocation = mLocation;
    }
}
```

Итак, все необходимые данные получены; теперь можно построить карту для их отображения.

Работа с картой

Наш фрагмент `SupportMapFragment` создает виджет `MapView`, который в свою очередь становится хостом для объекта, выполняющего настоящую работу: `GoogleMap`. Следовательно, работа должна начаться с получения ссылки на этот «главный» объект. Для этого вызовите `getMapAsync(OnMapReadyCallback)`.

Листинг 34.9. Получение объекта GoogleMap (LocatrFragment.java)

```
public class LocatrFragment extends SupportMapFragment {
    ...
    private static final int REQUEST_LOCATION_PERMISSIONS = 0;

    private GoogleApiClient mClient;
    private GoogleMap mMap;
    private Bitmap mMapImage;
    private GalleryItem mMapItem;
    private Location mCurrentLocation;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setHasOptionsMenu(true);
        mClient = new GoogleApiClient.Builder(getActivity())
                ...
                .build();

        getMapAsync(new OnMapReadyCallback() {
            @Override
            public void onMapReady(GoogleMap googleMap) {
                mMap = googleMap;
            }
        });
    }
```

Назначение метода `SupportMapFragment.getMapAsync(…)` полностью соответствует его имени: метод асинхронно получает объект карты. Если вызвать его из `onCreate(Bundle)`, то вы получите ссылку на уже созданный и инициализированный объект `GoogleMap`.

С объектом `GoogleMap` мы можем обновить внешний вид карты в соответствии с текущим состоянием `LocatrFragment`. Первое, что нужно сделать, — увеличить зону, представляющую интерес. Эта зона должна быть окружена полями; добавьте в файл размеров значение для величины полей.

Листинг 34.10. Добавление полей (res/values/dimens.xml)

```
<resources>
    <!-- Поля по умолчанию согласно стилевым рекомендациям Android. -->
    <dimen name="activity_horizontal_margin">16dp</dimen>
    <dimen name="activity_vertical_margin">16dp</dimen>
    <dimen name="map_inset_margin">100dp</dimen>
</resources>
```

Затем добавьте реализацию `updateUI()` для выполнения масштабирования.

Листинг 34.11. Увеличение карты (LocatrFragment.java)

```
private boolean hasLocationPermission() {
    ...
}
```

```
private void updateUI() {
    if (mMap == null || mMapImage == null) {
        return;
    }

    LatLng itemPoint = new LatLng(mMapItem.getLat(), mMapItem.getLon());
    LatLng myPoint = new LatLng(
            mCurrentLocation.getLatitude(), mCurrentLocation.getLongitude());

    LatLngBounds bounds = new LatLngBounds.Builder()
            .include(itemPoint)
            .include(myPoint)
            .build();

    int margin = getResources().getDimensionPixelSize(R.dimen.map_inset_margin);
    CameraUpdate update = CameraUpdateFactory.newLatLngBounds(bounds, margin);
    mMap.animateCamera(update);
}

private class SearchTask extends AsyncTask<Location,Void,Void> {
```

Вкратце, здесь происходит следующее: чтобы перемещать `GoogleMap` по карте, мы строим объект `CameraUpdate`. Класс `CameraUpdateFactory` содержит разнообразные статические методы для построения различных видов объектов `CameraUpdate`, которые изменяют позицию, уровень увеличения и другие свойства участка, отображаемого на карте.

В данном случае мы создаем обновление, которое наводит камеру на конкретный объект `LatLngBounds`. Считайте, что объект `LatLngBounds` представляет прямоугольник, заключающий множество точек. Прямоугольник также можно определить явно, указав координаты его юго-западного и северо-восточного углов.

Впрочем, чаще бывает удобнее предоставить список точек, которые этот прямоугольник должен включать. С классом `LatLngBounds.Builder` это делается легко: создайте `LatLngBounds.Builder` и вызовите `.include(LatLng)` для каждой точки, которая должна быть включена в `LatLngBounds` (точки представляются объектами `LatLng`). Когда это будет сделано, остается вызвать `build()` для получения правильно настроенного объекта `LatLngBounds`.

Далее карта обновляется одним из двух способов: методом `moveCamera(CameraUpdate)` или `animateCamera(CameraUpdate)`. Второй метод (с анимацией) интереснее, поэтому естественно, что мы использовали именно его.

Подключите свой метод `updateUI()` в двух местах: при исходном получении карты и при завершении поиска.

Листинг 34.12. Подключение updateUI() (LocatrFragment.java)

```
@Override
public void onCreate(Bundle savedInstanceState) {
    ...
    getMapAsync(new OnMapReadyCallback() {
        @Override
```

```
        public void onMapReady(GoogleMap googleMap) {
            mMap = googleMap;
            updateUI();
        }
    });
}
...
private class SearchTask extends AsyncTask<Location,Void,Void> {
    ...
    @Override
    protected void onPostExecute(Void result) {
        mMapImage = mBitmap;
        mMapItem = mGalleryItem;
        mCurrentLocation = mLocation;

        updateUI();
    }
}
```

Запустите приложение Locatr и нажмите кнопку поиска. На карте появляется увеличенное изображение области, включающей ваше текущее местоположение (рис. 34.2). (У пользователей эмуляторов для получения позиционных данных должно работать приложение MockWalker.)

Рис. 34.2. Карта с увеличением

Рисование на карте

Карта получилась симпатичная, но не слишком наглядная. Где-то здесь находитесь вы, где-то рядом находится фотография Flickr... Но где именно? Добавим ясности при помощи маркеров.

Рисование на карте отличается от рисования на обычном представлении. Собственно, оно проще: вместо того, чтобы рисовать пикселы на экране, вы «рисуете» особенности географической области. Под «рисованием» мы имеем в виду «построение маленьких объектов и добавление их к объекту `GoogleMap`, чтобы он мог нарисовать их за вас».

Вообще-то и эта формулировка не совсем точна — эти объекты создает объект `GoogleMap`, а не вы. А вы создаете описания того, что же должен создать объект `GoogleMap`, — так называемые *варианты* (options).

Добавьте на карту два маркера: создайте объекты `MarkerOptions` и вызовите `mMap.addMarker(MarkerOptions)`.

Листинг 34.13. Добавление маркеров (LocatrFragment.java)

```
private void updateUI() {
    ...
    LatLng itemPoint = new LatLng(mMapItem.getLat(), mMapItem.getLon());
    LatLng myPoint = new LatLng(
            mCurrentLocation.getLatitude(), mCurrentLocation.getLongitude());

    BitmapDescriptor itemBitmap = BitmapDescriptorFactory.fromBitmap(mMapImage);
    MarkerOptions itemMarker = new MarkerOptions()
            .position(itemPoint)
            .icon(itemBitmap);
    MarkerOptions myMarker = new MarkerOptions()
            .position(myPoint);

    mMap.clear();
    mMap.addMarker(itemMarker);
    mMap.addMarker(myMarker);

    LatLngBounds bounds = new LatLngBounds.Builder()
    ...
}
```

При вызове `addMarker(MarkerOptions)` объект `GoogleMap` строит экземпляр `Marker` и добавляет его на карту. Если в будущем вам потребуется удалить или изменить маркер, сохраните этот экземпляр. В нашем примере карта должна очищаться при каждом обновлении, поэтому сохранять объекты `Marker` не нужно.

Запустите приложение Locatr и нажмите кнопку поиска — вы увидите, что на карте отображаются два маркера (рис. 34.3).

Наше приложение для геопоиска изображений завершено. В этой главе вы научились пользоваться двумя API Play Services, определили местоположение телефона, зарегистрировались для использования одной из многочисленных веб-служб API и нанесли все на карту.

Рис. 34.3. Маркеры на карте

Для любознательных: группы и ключи API

Если над приложением с ключом API работают сразу несколько разработчиков, построение отладочных версий усложняется. Ваши удостоверения содержатся в файле хранилища ключей, уникального для вас. В группе каждый разработчик имеет *собственный* файл хранилища ключей и *собственные* удостоверения. Чтобы подключить нового участника к работе над приложением, вы должны запросить у него хеш SHA1, а затем обновить удостоверения вашего ключа API.

По крайней мере, это один из способов организации управления ключом API: управление всеми хешами подписания в проекте. Если вы хотите целенаправленно управлять тем, кто что делает, это может быть верным решением.

Однако существует и другой вариант: создание отладочного хранилища ключей для конкретного проекта. Начните с создания нового отладочного хранилища программой Java *keytool*.

Листинг 34.14. Создание нового хранилища ключей

```
$ keytool -genkey -v -keystore debug.keystore -alias androiddebugkey \
--storepass android -keypass android -keyalg RSA -validity 14600
```

Программа *keytool* задаст серию вопросов. Честно ответьте на них (так как речь идет об отладочном ключе, можно оставить ответы по умолчанию всем вопросам, кроме имени).

```
$ keytool -genkey -v -keystore debug.keystore -alias androiddebugkey \
--storepass android -keypass android -keyalg RSA -validity 14600
What is your first and last name?
  [Unknown]:  Bill Phillips
...
```

После того как файл debug.keystore будет сгенерирован, переместите его в папку модуля app. Затем откройте окно структуры проекта, выберите модуль app и перейдите на вкладку Signing. Щелкните на кнопке +, чтобы добавить новую конфигурацию подписания. Введите в поле Name строку debug, а в поле Store file — строку debug.keystore (рис. 34.4).

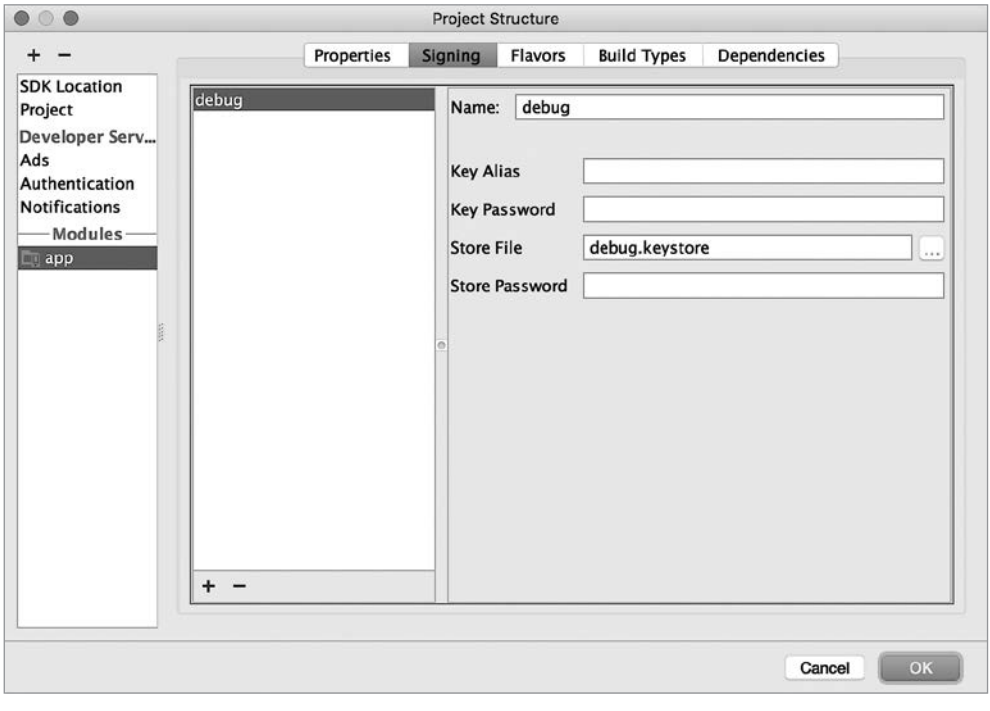


Рис. 34.4. Настройка отладочного ключа подписания

Если вы настроите ключ API для использования нового хранилища, то все участники смогут использовать один и тот же ключ, работая с одним хранилищем. Согласитесь, так удобнее.

Если вы выберете этот вариант, вам придется проявить осмотрительность в отношении распространения нового файла debug.keystore. Если он будет доступен только в закрытом репозитории, все будет нормально. Только не публикуйте его в открытом репозитории, в котором к нему сможет обратиться любой желающий, потому что это позволит посторонним использовать ваш ключ API.

35 Материальный дизайн

Одним из самых значительных новшеств Android за последнее десятилетие стало введение нового стиля дизайна — *материального дизайна* (material design). Этот новый визуальный язык привлек большое внимание при выходе Android 5.0 (Lollipop), а вскоре было выпущено чрезвычайно подробное руководство по стилю.

Конечно, нас, разработчиков, вопросы дизайна касаются только косвенно. Наша задача — заставить программу работать, а как она при этом будет выглядеть — дело второстепенное. Однако материальный дизайн дополняет субъективные факторы дизайна новыми концепциями интерфейса. Если вы познакомитесь с ними, вам будет намного проще реализовать новые решения.

Последняя глава несколько отличается от предыдущих. Считайте, что это очередной раздел «Для любознательных», только очень большой. Здесь нет примеров, а большая часть приведенной информации не входит в обязательный круг чтения.

С дизайнерской точки зрения материальный дизайн выводит на передний план три основные идеи:

- *Материал как метафора*: компоненты приложения действуют как физические, материальные объекты.
- *Яркий, графический, броский характер*: дизайн приложения должен «бросаться в глаза», как качественный дизайн в журнале или в книге.
- *Содержательность движения*: приложение должно использовать анимацию в ответ на действия, выполняемые пользователем.

Единственное, о чем ничего не будет сказано в этой главе, — второй пункт. Все это относится к обязанностям дизайнера. Если вы сами занимаетесь дизайном своего приложения, обратитесь за подробностями к руководствам по созданию материального дизайна (*developer.android.com/design/material/index.html*).

Что касается первого пункта — «материал как метафора», — дизайнерам понадобится ваше содействие в построении материальных поверхностей. Вы должны уметь размещать такие поверхности в трех измерениях с использованием свойств оси *Z*, а также использовать два новых материальных виджета: незакрепленные панели действий и всплывающие уведомления (snackbars).

Наконец, чтобы не нарушать принцип *содержательности движения*, вам придется освоить новый набор средств анимации: аниматоров списков состояний, анимированных списков состояний (да, все правильно — это не одно и то же), круговых раскрытий и переходов между общими элементами. Все эти средства придают дизайну визуальный интерес, так высоко ценимый дизайнерами.

Материальные поверхности

Первая и самая главная концепция материального дизайна, которую должен усвоить разработчик, — это понятие материальных поверхностей. Дизайнеры представляют себе их как карточки толщиной 1dp. Эти карточки обладают волшебными свойствами: они могут увеличиваться, на них могут отображаться анимационная графика и изменяющийся текст (рис. 35.1).

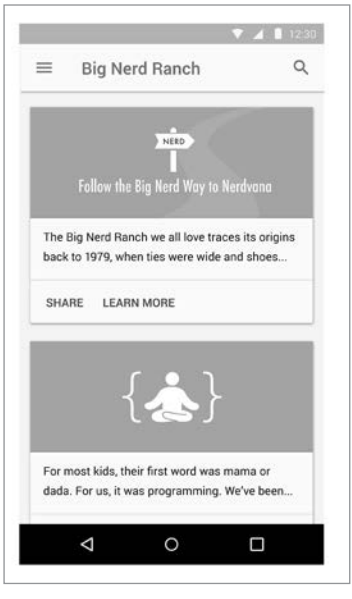


Рис. 35.1. Интерфейс с двумя материальными поверхностями

Но несмотря на все волшебство, поверхности по-прежнему ведут себя как настоящие бумажные карточки. Например, один лист бумаги не может пройти сквозь другой. Та же логика действует и при перемещении материальных поверхностей: они не могут проходить сквозь друг друга.

Поверхности существуют и перемещаются друг относительно друга в трехмерном пространстве. Они могут двигаться как вверх (по направлению к вашему пальцу), так и вниз, удаляясь от него (рис. 35.2).

Чтобы сместить одну поверхность относительно другой, следует приподнять ее и переместить (рис. 35.3).

Возвышение и координата Z

Самый очевидный способ имитации глубины интерфейса — тени, отбрасываемые элементами друг на друга. Кто-то решит, что в идеальном мире дизайнеры должны возиться с прорисовкой теней, пока мы, разработчики, едим пончики. (Впрочем, у кого-то могут быть другие мнения по поводу того, как выглядит идеальный мир.)

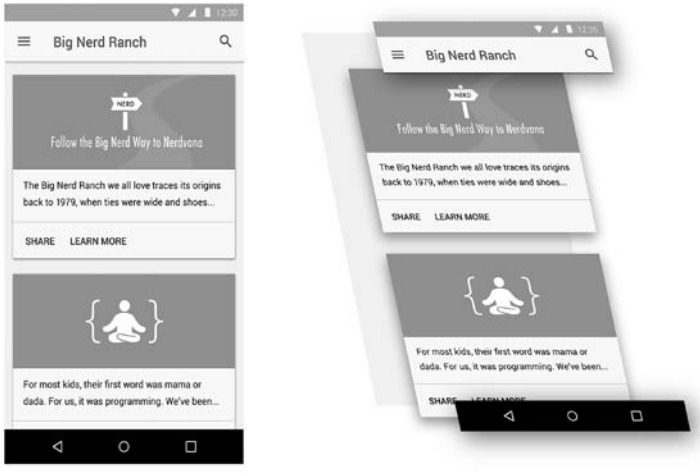


Рис. 35.2. Материальный дизайн в трехмерном пространстве

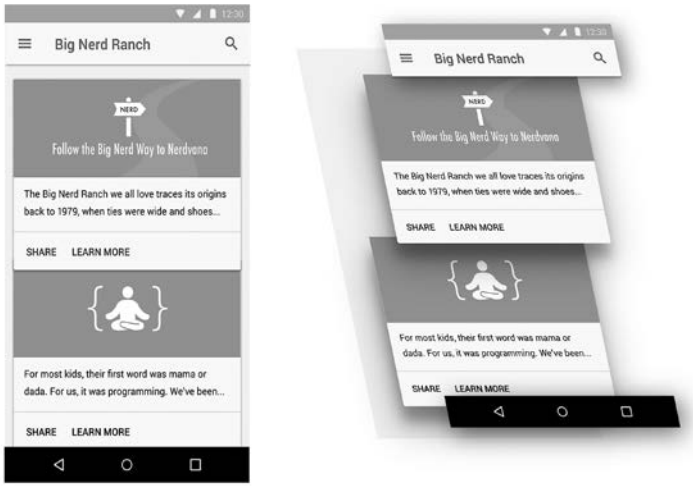


Рис. 35.3. Относительное перемещение поверхностей

Однако прорисовка теней для множества разнообразных поверхностей — притом во время перемещения! — не из тех задач, с которыми дизайнер может справиться самостоятельно. Вместо этого вы поручаете прорисовку теней Android, назначая каждому представлению некоторое *возвышение* (elevation).

В Lollipop в систему макетов была добавлена ось *Z*, задающая положение представления в трехмерном пространстве. Возвышение — своего рода дополнительная координата, назначаемая представлению в макете: представление можно сместить относительно исходной позиции, но основное «место обитания» находится именно там (рис. 35.4).

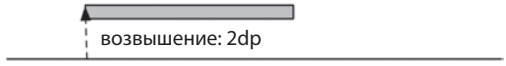


Рис. 35.4. Возвышение по оси Z

Чтобы задать величину возвышения, либо вызовите метод `View.setElevation(float)`, либо задайте нужное значение в файле макета.

Листинг 35.1. Назначение возвышения в файле макета

```
<Button xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:elevation="2dp"/>
```

Так как мы собираемся задать базовое значение по оси *Z*, решение с атрибутом XML является предпочтительным. Кроме того, оно проще вызова `setElevation(float)`, потому что атрибут `elevation` игнорируется в старых версиях Android, и вам не придется беспокоиться о совместимости.

Для изменения возвышения `View` используются свойства `translationZ` и `Z`. Они работают точно так же, как свойства `translationX`, `translationY`, `X` и `Y`, представленные в главе 32.

Значение `Z` всегда равно сумме `elevation` и `translationZ`. Если вы присвоите значение `Z`, система выполнит необходимые вычисления для присваивания нужного значения `translationZ` (рис. 35.5).

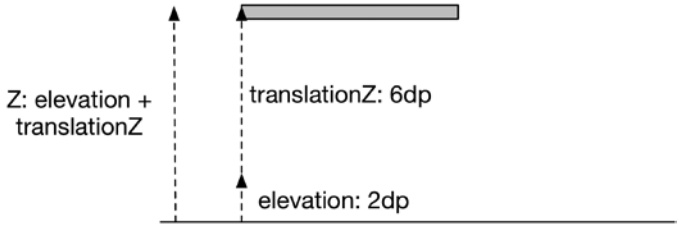


Рис. 35.5. Z и translationZ

Аниматоры списков состояний

В материальных приложениях часто используются многочисленные взаимодействия с пользователем. Чтобы увидеть пример, достаточно нажать кнопку в Lollipop: нажатие кнопки сопровождается анимацией по оси *Z*. Когда вы отпускаете палец, кнопка снова возвращается в прежнее состояние.

Для упрощения реализации подобных анимаций в Lollipop появились *аниматоры списков состояний* — анимационные аналоги графических объектов списков состояний: вместо того, чтобы заменять один графический объект другим, они выполняют анимацию представления в некоторое состояние. Для выполнения

анимации кнопки при нажатии можно определить аниматор списка состояний, который находится в папке res/animator и выглядит так:

Листинг 35.2. Пример аниматора списка состояний

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true">
        <objectAnimator android:propertyName="translationZ"
                        android:duration="100"
                        android:valueTo="6dp"
                        android:valueType="floatType"
                        />
    </item>
    <item android:state_pressed="false">
        <objectAnimator android:propertyName="translationZ"
                        android:duration="100"
                        android:valueTo="0dp"
                        android:valueType="floatType"
            />
    </item>
</selector>
```

Такое решение прекрасно подходит для анимации свойств. Если же вы хотите выполнить покадровую анимацию, понадобится другой инструмент — *анимированный список состояний*.

Термин «анимированный список состояний» создает некоторую путаницу. Он похож на «аниматор списка состояний», но работает совершенно иначе. Анимированные списки состояний позволяют определить изображения для каждого состояния, как и обычные списки состояний, но они также позволяют определять покадровые анимационные переходы между этими состояниями.

В главе 23 мы определили список состояний для кнопок BeatBox. Если бы садист-дизайнер (вроде нашего Кар Лун Вона) захотел, чтобы каждое нажатие кнопки сопровождалось многокадровой анимацией, можно было бы изменить XML в соответствии с листингом 35.3. (Эта версия должна была бы находиться в папке res/drawable-21, потому что эта возможность не поддерживалась до выхода Lollipop.)

Листинг 35.3. Анимированный список состояний

```
<animated-selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/pressed"
          android:drawable="@drawable/button_beat_box_pressed"
          android:state_pressed="true"/>
    <item android:id="@+id/released"
          android:drawable="@drawable/button_beat_box_normal" />

    <transition
        android:fromId="@id/released"
        android:toId="@id/pressed">
        <animation-list>
            <item android:duration="10" android:drawable="@drawable/button_frame_1" />
```

```
            <item android:duration="10" android:drawable="@drawable/button_frame_2" />
            <item android:duration="10" android:drawable="@drawable/button_frame_3" />
            ...
        </animation-list>
    </transition>
</animated-selector>
```

Здесь каждому элементу в селекторе назначается идентификатор. Затем мы определяем переход между идентификаторами для воспроизведения многокадровой анимации. При желании также можно определить анимацию для отпускания кнопки; для этого потребуется другой тег `transition`.

Средства анимации

В материальном дизайне используются модные анимации. Одни из них реализуются легко; другие требуют более значительной работы, но Android предоставляет инструменты, которые вам в этом помогут.

Круговое раскрытие

Анимация кругового раскрытия используется в материальном дизайне для создания эффекта, напоминающего расширяющуюся круговую заливку. Представление или блок контента последовательно раскрываются от некоторой точки — чаще всего точки прикосновения. Рисунок 35.6 помогает понять, как может работать круговое раскрытие.

Рис. 35.6. Круговое раскрытие при нажатии кнопки в приложении BeatBox

Возможно, вы помните упрощенную версию этого эффекта из главы 6, где она использовалась для сокрытия кнопки. Сейчас будет рассмотрен другой, более неординарный способ использования кругового раскрытия.

Для создания анимации кругового раскрытия следует вызвать метод `createCircularReveal(…)` для `ViewAnimationUtils`. Этот метод получает обширный набор параметров:

```
static Animator createCircularReveal(View view, int centerX, int centerY, float
startRadius, float endRadius)
```

В параметре `View` передается представление, которое должно быть раскрыто в ходе анимации. На рис. 35.6 это представление окрашено в красный цвет, а его ширина и высота совпадают с шириной и высотой `BeatBoxFragment`. Если выполнить анимацию от `startRadius=0` до большого значения `endRadius`, это представление в исходном состоянии будет полностью прозрачным, а затем будет постепенно проявляться при расширении круга. Начальная точка анимации (в координатах `View`) будет находиться в точке с координатами `centerX` и `centerY`. Метод возвращает объект `Animator`, который работает точно так же, как объект `Animator` из главы 30.

В руководствах по материальному оформлению говорится, что такие анимации должны начинаться в точке, в которой пользователь прикоснулся к экрану. Следовательно, первым шагом должно быть определение экранных координат точки касания, как показано в листинге 35.4.

Листинг 35.4. Определение экранных координат в слушателе щелчка

```
@Override
public void onClick(View clickSource) {
    int[] clickCoords = new int[2];

    // Нахождение экранных координат clickSource
    clickSource.getLocationOnScreen(clickCoords);

    // Переход к центральной точке представления
    // вместо угловой точки
    clickCoords[0] += clickSource.getWidth() / 2;
    clickCoords[1] += clickSource.getHeight() / 2;

    performRevealAnimation(mViewToReveal, clickCoords[0], clickCoords[1]);
}
```

После этого можно выполнить анимацию (листинг 35.5).

Листинг 35.5. Выполнение анимации раскрытия

```
private void performRevealAnimation(View view, int screenCenterX,
                                    int screenCenterY) {
    // Нахождение центра относительно представления,
    // к которому будет применяться анимация
    int[] animatingViewCoords = new int[2];
    view.getLocationOnScreen(animatingViewCoords);
    int centerX = screenCenterX - animatingViewCoords[0];
    int centerY = screenCenterY - animatingViewCoords[1];
```

```
    // Определение максимального радиуса
    Point size = new Point();
    getActivity().getWindowManager().getDefaultDisplay().getSize(size);
    int maxRadius = size.y;

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
        ViewAnimationUtils.createCircularReveal(view, centerX, centerY,
                                                0, maxRadius)
                .start();
    }
}
```

Важное замечание: чтобы этот метод сработал, представление уже должно находиться в макете.

Переходы между общими элементами

Другая разновидность анимации, новая для материального дизайна, — *переходы между общими элементами*. Такие переходы предназначены для конкретной ситуации: на двух экранах отображаются некоторые одинаковые объекты.

Вспомните, как вы работали над приложением CriminalIntent. В этом приложении отображалась уменьшенная версия сделанного снимка. В одном из упражнений вам было предложено построить другое представление для полноразмерной версии изображения. Ваше решение могло выглядеть примерно так, как показано на рис. 35.7.

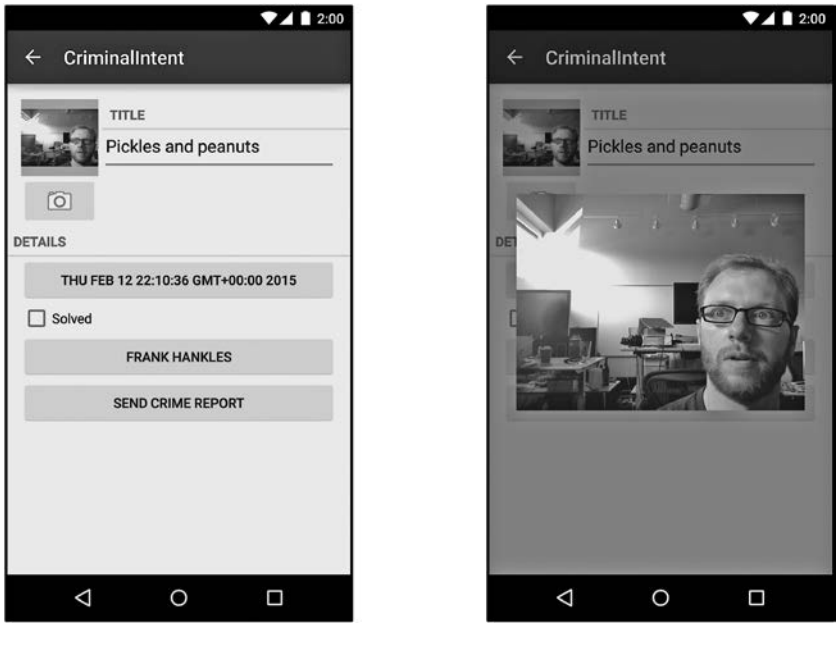


Рис. 35.7. Интерфейс с двумя материальными поверхностями

Это стандартный паттерн из области пользовательского интерфейса: вы нажимаете на одном элементе, а следующее представление открывает дополнительную информацию об этом элементе.

Анимация перехода между общими элементами предназначена для любых ситуаций, в которых вы переходите между двумя экранами, на которых отображаются общие элементы. В данном случае как на большом изображении справа, так и на миниатюре слева выводится одно изображение. Иначе говоря, это изображение является *общим элементом*.

Начиная с Lollipop, система Android предоставляет средства для реализации переходов между активностями или фрагментами. Сейчас мы покажем, как эти средства работают с активностями. Середина анимации выглядит так, как показано на рис. 35.8.

Рис. 35.8. Переход между общими элементами

Для активностей процесс реализации перехода состоит из трех шагов.

1. Включить переходы между активностями;
2. Назначить имена переходов представлениям общего элемента;
3. Открыть следующую активность с объектом `ActivityOptions`, который инициирует переход.

Сначала необходимо включить переходы. Если ваша активность использует тему AppCompat, которая постоянно используется в этой книге, то этот шаг можно пропустить. (AppCompat наследует от темы `Material`, которая включает переходы между активностями за вас.)

В своем примере мы назначаем активности прозрачный фон при помощи конструкции `@android:style/Theme.Translucent.NoTitleBar`. Эта тема не наследует от темы `Material`, поэтому переходы между активностями для нее не включаются по умолчанию. Их приходится включать вручную, что может делаться одним из двух способов. Первый способ: добавьте в активность строку кода, как показано в листинге 35.6.

Листинг 35.6. Включение переходов между активностями в коде

```
@Override
public void onCreate(Bundle savedInstanceState) {
    getWindow().requestFeature(Window.FEATURE_ACTIVITY_TRANSITIONS);
    super.onCreate(savedInstanceState);
    ...
}
```

Второй способ: измените стиль, используемый активностью, и присвойте атрибуту `android:windowActivityTransitions` значение `true`.

Листинг 35.7. Включение переходов между активностями на уровне стиля

```
<resources>
    <style name="TransparentTheme"
        parent="@android:style/Theme.Translucent.NoTitleBar">
        <item name="android:windowActivityTransitions">true</item>
    </style>

</resources>
```

Следующий шаг реализации перехода между общими элементами — пометка представлений, содержащих общий элемент, именем перехода. Для этого используется свойство `View`, появившееся в API 21: `transitionName`. Значение свойства задается либо в XML, либо в коде; в зависимости от обстоятельств тот или иной способ может оказаться предпочтительным. В нашем случае имя перехода для увеличенного изображения задается назначением `android:transitionName` в разметке XML (рис. 35.9).

Затем мы определим статический метод `startWithTransition(…)` (листинг 35.8) для назначения того же имени представлению, с которого должна начинаться анимация.

Листинг 35.8. Метод startWithTransition

```
public static void startWithTransition(Activity activity, Intent intent,
  View sourceView) {
    ViewCompat.setTransitionName(sourceView, "image");
    ActivityOptionsCompat options = ActivityOptionsCompat
        .makeSceneTransitionAnimation(activity, sourceView, "image");

    activity.startActivity(intent, options.toBundle());
}
```

Метод `ViewCompat.setTransitionName(View, String)` предназначен для старых версий Android, в которых класс `View` не имеет реализации `setTransitionName(String)`.

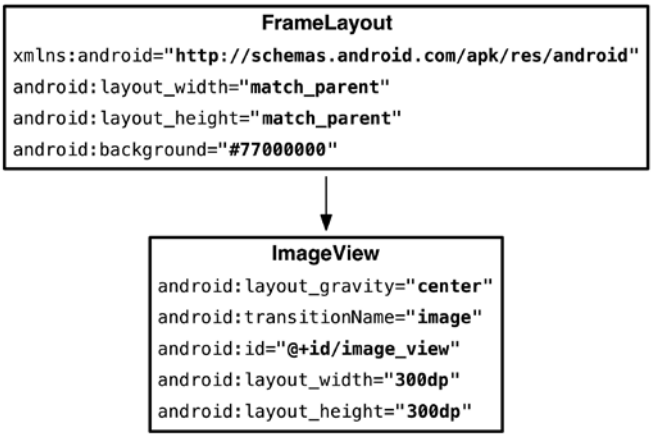


Рис. 35.9. Макет увеличенного изображения

В листинге 35.8 также представлен и последний шаг: создание объекта `ActivityOptions`. Этот объект передает ОС информацию об общих элементах и используемом значении `transitionName`.

О переходах и переходах между общими элементами можно рассказать еще очень много. Например, они могут использоваться для переходов между фрагментами. За дополнительной информацией обращайтесь к документации Google по инфраструктуре переходов: developer.android.com/training/transitions/overview.html.

Компоненты View

В новом руководстве по материальному дизайну из Lollipop описаны новые разновидности компонентов представлений View. Группа разработки Android предоставила реализации многих из этих компонентов. Давайте познакомимся с некоторыми представлениями, которые с большой вероятностью встретятся вам в ходе работы.

Карточки

Первый новый виджет — *карточка* (card) — является контейнером для других виджетов (рис. 35.10).

В карточках размещаются другие виды контента. Они слегка приподняты, за ними отображается тень, а углы слегка закруглены.

Книга, которую вы держите в руках, — не учебник по дизайну, так что мы не сможем ничего посоветовать относительно того, когда и где использовать карточки. (Если вас заинтересует эта тема, обращайтесь к документации Google по материальному дизайну: *www.google.com/design/spec*.) Впрочем, мы можем сказать, как следует создавать карточки: при помощи `CardView`.

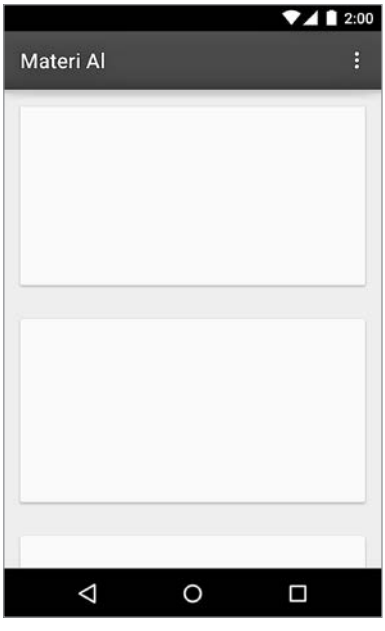


Рис. 35.10. Карточки

Класс `CardView` поставляется в отдельной библиотеке поддержки v7, как и `RecyclerView`. Чтобы включить его в свой проект, добавьте в модуль зависимость `com.android.support:cardview-v7`.

После этого вы сможете использовать `CardView` в макетах по тем же принципам, что и все остальные разновидности `ViewGroup`. Класс является субклассом `FrameLayout`, поэтому к потомкам `CardView` можно применять любые параметры макетирования `FrameLayout`.

Листинг 35.9. Использование CardView в макете

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              xmlns:tools="http://schemas.android.com/tools"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical"
              tools:context=".MainActivity">
    <android.support.v7.widget.CardView
        android:id="@+id/item"
        android:layout_width="match_parent"
        android:layout_height="200dp"
        android:layout_margin="16dp"
        >
        ...
    </android.support.v7.widget.CardView>

</LinearLayout>
```

Так как класс `CardView` находится в библиотеке поддержки, это предоставляет некоторые удобные аспекты совместимости на старых устройствах. В отличие от других виджетов, карточка всегда отбрасывает тень. (На старых устройствах она просто рисует собственную копию — не идеальное решение, но достаточно близкое к идеалу.) За информацией о других мелких визуальных различиях обращайтесь к документации `CardView`.

Плавающие кнопки действий

Еще один компонент, который часто встречается на практике, — *плавающая кнопка действия* (FAB, Floating Action Button). Пример изображен на рис. 35.11.

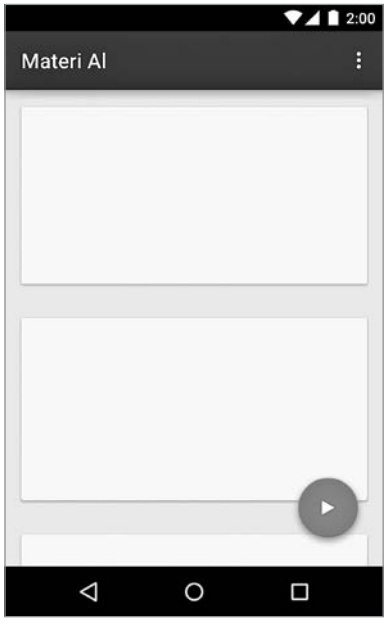


Рис. 35.11. Плавающая кнопка действия

Реализация плавающей кнопки действия доступна в библиотеке поддержки Google. Чтобы включить библиотеку в проект, добавьте зависимость `com.android.support:design:24.2.1`.

Плавающая кнопка действия представляет собой кружок со сплошной заливкой и круглой тенью, предоставляемой классом `OutlineProvider`. Класс `FloatingActionButton`, субкласс `ImageView`, обеспечивает прорисовку круга и тени. Просто включите элемент `FloatingActionButton` в файл макета и задайте его атрибуту `src` изображение, которое должно отображаться на кнопке.

Хотя плавающую кнопку действия можно поместить в `FrameLayout`, в библиотеку поддержки включен удобный класс `CoordinatorLayout` — субкласс `FrameLayout`, изменяющий позицию плавающей кнопки в зависимости от движения других

компонентов. Теперь при отображении уведомления `Snackbar` плавающая кнопка смещается вверх, чтобы панель `Snackbar` ее не закрывала (листинг 35.10).

Листинг 35.10. Включение плавающей кнопки действия в макет

```
<android.support.design.widget.CoordinatorLayout
            xmlns:android="http://schemas.android.com/apk/res/android"
            xmlns:tools="http://schemas.android.com/tools"
            xmlns:app="http://schemas.android.com/apk/res-auto"
            android:layout_width="match_parent"
            android:layout_height="match_parent">
    [... основной контент ...]
    <android.support.design.widget.FloatingActionButton
        android:id="@+id/floating_action_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|right"
        android:layout_margin="16dp"
        android:src="@drawable/play"/>
</android.support.design.widget.CoordinatorLayout>
```

Этот код размещает кнопку над остальным контентом в правом нижнем углу, не нарушая размещения других виджетов.

Всплывающие уведомления

Всплывающие уведомления `Snackbar` сложнее плавающих кнопок действий. Они представляют собой маленькие интерактивные компоненты, появляющиеся у нижнего края экрана (рис. 35.12).

Уведомления `Snackbar` появляются сверху вниз у нижнего края экрана. По истечении заданного периода времени (или после следующего взаимодействия с экраном) они автоматически опускаются за пределы экрана. По своему назначению всплывающие уведомления `Snackbar` напоминают уведомления `Toast`, но в отличие от `Toast` они являются частью собственного интерфейса приложения. Уведомление `Toast` появляется над приложением и остается даже в том случае, если вы переместитесь в другое место. Кроме того, `Snackbar` может предоставить кнопку для выполнения непосредственного действия пользователем.

Реализация уведомлений `Snackbar`, как и реализация плавающих кнопок действий, предоставляется Android в библиотеке поддержки.

По способу построения и отображения виджеты `Snackbar` также имеют много общего с `Toast` (листинг 35.11).

Листинг 35.11. Создание объекта Snackbar

```
Snackbar.make(container, R.string.munch, Snackbar.LENGTH_SHORT).show();
```

При построении `Snackbar` передается представление, в котором будет отображаться уведомление, отображаемый текст и промежуток времени, в течение которого

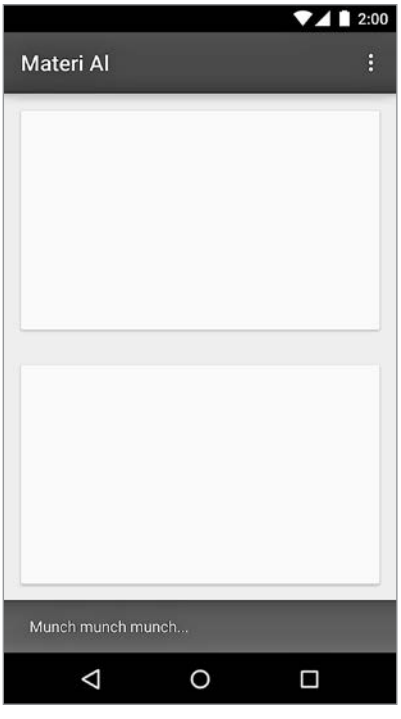


Рис. 35.12. Всплывающее уведомление Snackbar

он будет оставаться на экране. Наконец, вызов метода `show()` отображает уведомление на экране.

В правой части панели `Snackbar` может отображаться необязательный элемент действия. Это может быть удобно, если пользователь выполняет некое деструктивное действие (скажем, удаляет запись из базы) и вы хотите предоставить ему возможность отмены этого действия.

Подробнее о материальном дизайне

В этой главе мы представили некую «солянку» из разнородных инструментов. Если эти инструменты так и пролежат на дальней полке вашего творческого инструментария, никакого проку от них не будет. Не упускайте возможность придать приложению визуальную глубину или применить новую анимацию.

Одним из лучших источников вдохновения станет спецификация материального оформления, в которой представлено немало замечательных идей: *www.google.com/design/spec/material-design/introduction.html*. Также загляните в Google Play, посмотрите, как работают другие приложения, и спросите себя: а как бы я реализовал это в своем приложении? Возможно, ваша программа получится намного более интересной, чем предполагалось изначально.

Послесловие

Поздравляем! Вы добрались до последней страницы этого учебника. Не каждому хватило бы терпения сделать то, что вы сделали, узнать то, что вы узнали. Ваш самоотверженный труд не пропал даром — теперь вы стали разработчиком Android.

Последнее упражнение

У нас осталось еще одно, последнее упражнение для вас: станьте хорошим разработчиком Android. Каждый хороший разработчик хорош по-своему, поэтому с этого момента вы должны найти собственный путь.

С чего начать, спросите вы? Мы можем дать несколько советов.

- *Пишите код*. Начинайте прямо сейчас. Вы быстро забудете то, что узнали в книге, если не будете применять полученные знания. Примите участие в проекте или напишите собственное простое приложение. Что бы вы ни выбрали, не тратьте времени: пишите код.
- *Учитесь*. В этой книге вы узнали много полезной информации. Что-то из этого пробудило ваше воображение? Напишите код, чтобы поэкспериментировать со своими любимыми возможностями. Найдите и почитайте дополнительную документацию по ним — или целую книгу, если она есть. Также загляните на канал Android Developers на YouTube и послушайте подкаст Android Developers Backstage.
- *Встречайтесь с людьми*. Локальные встречи также помогут вам найти единомышленников. Многие первоклассные разработчики Android активно общаются в Twitter и в Google Plus. Посещайте конференции Android, на них вы познакомитесь с другими разработчиками Android (а может, и с нами!).
- *Присоединяйтесь к сообществу разработки с открытым кодом*. Разработка Android процветает на сайте *www.github.com*. Обнаружив полезную библиотеку, посмотрите, в каких еще проектах участвуют его авторы. Делитесь своим кодом — никогда не знаешь, кому он пригодится. Список рассылки Android Weekly поможет вам быть в курсе происходящего в сообществе Android (*androidweekly.net*).

Бессовестная самореклама

Если вам понравилась книга, ознакомьтесь с другими учебниками Big Nerd Ranch по адресу *www.bignerdranch.com/books*. У нас также имеется широкий выбор недельных курсов для разработчиков, на которых вы всего за неделю сможете получить массу полезной информации. И конечно, если вам понадобятся услуги опытных разработчиков, мы также занимаемся контрактным программированием. За подробностями обращайтесь на наш сайт по адресу *www.bignerdranch.com*.

Спасибо

Без таких читателей, как вы, наша работа была бы невозможной. Спасибо вам за то, что купили и прочитали нашу книгу.

Б. Филлипс, К. Стюарт, К. Марсикано

Android. Программирование для профессионалов

3-е издание

Перевел с английского *Е. Матвеев*

Заведующая редакцией	*Ю. Сергиенко*
Ведущий редактор	*Н. Римицан*
Литературный редактор	*Н. Суслова*
Художник	*С. Заматевская*
Корректор	*И. Тимофеева*
Верстка	*Л. Егорова*

Изготовлено в России. Изготовитель: ООО «Прогресс книга».
Место нахождения и фактический адрес: 191123, Россия, город Санкт-Петербург,
улица Радищева, дом 39, корпус Д, офис 415. Тел.: +78127037373.

Дата изготовления: 06.2017. Наименование: книжная продукция. Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12.000 —
Книги печатные профессиональные, технические и научные.

Подписано в печать 26.05.17. Формат 70×100/16. Бумага офсетная. Усл. п. л. 55,470. Тираж 1200. Заказ 0000.

Отпечатано в ОАО «Первая Образцовая типография». Филиал «Чеховский Печатный Двор».
142300, Московская область, г. Чехов, ул. Полиграфистов, 1.
Сайт: www.chpk.ru. E-mail: marketing@chpk.ru
Факс: 8(496) 726-54-10, телефон: (495) 988-63-87