Министерство образования и науки Российской Федерации
Ростовский государственный экономический университет (РИНХ)

С. А. Глушенко
А. И. Долженко

Разработка мобильных приложений

Учебное пособие

Ростов-на-Дону
2018

УДК 004.4(075)
Д 64

Глушенко, С. А., Долженко, А. И. Разработка мобильных приложений: Учебное пособие – Ростов-на-Дону: издательство РГЭУ (РИНХ), 2018. – 221 с.

ISBN

Данный курс имеет цель представить основные программные компоненты и решения для разработки мобильных приложений под операционную систему Android. Технологии разработки приложений иллюстрируются детальным разбором практических примеров, приводятся XML-описания компонентов приложений, коды на языке Java и экранные формы, которые используются как при проектировании, так и при представлении результатов работы приложений.

Курс ориентирован на студентов, имеющих небольшой опыт программирования на современных языках и желающих научиться основам программирования для операционной системы Android.

Предназначено для бакалавров и магистров, обучающихся по направлениям 09.03.01 «Программная инженерия», 09.03.02 «Информационные системы и технологии», 09.03.03 «Прикладная информатика», 080500 «Бизнес-информатика».

Рецензенты:

Е. Д. Стрельцова, профессор кафедры программного обеспечения вычислительной техники Южно-Российского государственного поличехнического университета (НПИ), д.э.н.

С. М. Щербаков, профессор кафедры информационных систем и прикладной информатики Ростовского государственного экономического университета (РИНХ), д.э.н.

Утверждено в качестве учебного пособия редакционно-издательским советом Ростовского государственного экономического университета (РИНХ).

ISBN

СОЖЕРЖАНИЕ

1. ВВЕДЕНИЕ В РАЗРАБОТКУ ANDROID-ПРИЛОЖЕНИЙ

История Android

Android – открытая операционная система для мобильных телефонов, смартфонов, коммуникаторов, планшетных компьютеров, электронных книг, цифровых проигрывателей, наручных часов, нетбуков и смартбуков, основанная на ядре Linux.

Изначально разрабатывалась компанией Android Inc., которую затем (июль, 2005) купила Google. Впоследствии (ноябрь, 2007) Google инициировала создание бизнес-альянса Open Handset Alliance (в его состав вошли: Google, HTC, Intel, Motorola, Nvidia и другие компании), который и занимается сейчас поддержкой и дальнейшим развитием платформы.

С момента выхода первой версии (сентябрь, 2008) произошло несколько обновлений системы. Эти обновления, как правило, касаются исправления обнаруженных ошибок и добавления нового функционала в систему.

Изначально Google рассчитывала давать версиям Android имена известных роботов, но отказалась из-за проблем с авторскими правами. Каждая версия системы, начиная с версии 1.5, получает собственное кодовое имя на тему сладостей. Кодовые имена присваиваются в алфавитном порядке латинского алфавита. В настоящий момент актуальной пользовательской версией является Android O (Oreo), окончательный официальный релиз которой состоялся 21 августа 2017 г.

Инструментарий разработчика

Разработка Android-приложений, как правило, осуществляется на языке **Java**, который является объектно-ориентированным языком программирования. Программы на Java транслируются в байт-код, выполняемый виртуальной машиной Java, которая обрабатывает байтовый код и передает инструкции оборудованию как интерпретатор. Достоинство подобного способа выполнения программ заключается в полной независимости байт-кода от операционной системы и оборудования, что позволяет выполнять Java-приложения на любом устройстве, для которого существует соответствующая виртуальная машина. Другой важной особенностью Java является гибкая система безопасности, благодаря тому что исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного доступа к данным или соединения с другим компьютером) вызывают немедленное прерывание.

Для разработки Android-приложений, в первую очередь, необходимо установить **Java Development Kit (JDK)**. JDK – это бесплатно распространяемый комплект разработчика приложений на языке Java, включающий в себя компилятор Java, стандартные библиотеки классов Java, примеры, документацию, различные утилиты и исполнительную систему Java Runtime Environment (JRE).

В состав JDK не входит интегрированная среда разработки (Integrated Development Environment). Поэтому после того как будет установлен JDK, следует установить IDE. Существует несколько популярных сред разработки, но в данном курсе будет использована **Android Studio**.

Android Software Development Kit (SDK) содержит множество инструментов и утилит для создания и тестирования приложений. Например, с помощью SDK Manager можно установить Android API любой версии, а также проверить репозиторий на наличие доступных, но еще не установленных пакетов и архивов.

Архитектура Android

Основные компоненты операционной системы Android (рисунок 1.1).

Рисунок 1.1 – Архитектура Android

Приложения. Android поставляется с набором основных приложений, который включает: календарь, карты, браузер, менеджер контактов и другие. Все перечисленные приложения написаны на Java.

Платформа приложений. Предоставляя открытую платформу разработки, Android дает разработчикам возможность создавать гибкие и инновационные приложения. Разработчики могут использовать аппаратные возможности устройства, получать информацию о местоположении, выполнять задачи в фоновом режиме, устанавливать оповещения и многое другое. Разработчики имеют полный доступ к тем же API, что используются в основных приложениях.

Архитектура приложений разработана с целью упрощения повторного использования компонентов; любое приложение может «публиковать» свои возможности, и любое другое приложение может затем использовать эти возможности (с учетом ограничений безопасности). Этот же механизм позволяет заменять стандартные компоненты на пользовательские.

Библиотеки. Android включает в себя набор C/C++ библиотек, используемых различными компонентами системы. Эти возможности доступны разработчикам в контексте применения Android Aplication Framework. Некоторые основные библиотеки перечислены ниже:

- медиабиблиотеки – эти библиотеки предоставляют поддержку воспроизведения и записи многих популярных аудио, видеоформатов и форматов изображений, в том числе MPEG4, MP3, AAC, AMR, JPG, PNG и других;
- Surface Manager – управляет доступом к подсистеме отображения 2D и 3D графических слоев;
- LibWebCore – современный веб-движок, на котором построен браузер Android;
- SGL – основной графический движок 2D;
- 3D библиотеки – реализованы на основе OpenGL; библиотеки используют либо аппаратное 3D-ускорение (при его наличии), либо включены программно;
- FreeType – поддержка растровых и векторных шрифтов
- SQLite – механизм базы данных, доступной для всех приложений.

Выполняемая среда Android. Android включает в себя набор основных библиотек, которые обеспечивают большинство функций, доступных в библиотеках Java. Каждое приложение Android работает в своем собственном процессе, со своим собственным экземпляром виртуальной машины Dalvik. Dalvik была написана так, что устройство может работать эффективно с несколькими виртуальными машинами одновременно.

Dalvik проектировалась специально под платформу Android. Виртуальная машина оптимизирована для низкого потребления памяти и работы на мобильном аппаратном обеспечении. Dalvik использует собственный байт-код. Android-приложения переводятся компилятором в специальный машинно-независимый низкоуровневый код. И именно Dalvik интерпретирует и выполняет такую программу при выполнении на платформе. Кроме того, с помощью специальной утилиты, входящей в состав Android SDK, Dalvik способна переводить байт-коды Java в коды собственного формата и также исполнять их в своей виртуальной среде.

Ядро Linux. Android основан на Linux версии 2.6 с основными системными службами: – безопасностью, управлением памятью, управлением процессами и моделью драйверов. Разработчики Android модифицировали ядро Linux, добавив поддержку аппаратного обеспечения, используемого в мобильных устройствах и чаще всего недоступного на компьютерах.

Обзор Java-интерфейсов

Основа для прикладного программиста Android – это набор интерфейсов на языке Java. Следует рассмотреть, как он организован. В основе набора – пакеты, входящие в стандарт языка Java, такие как `java.util, java.lang, java.io`. Они есть на любой платформе, где могут быть запущены java-приложения, и неспецифичны для Android. К ним примыкают расширения, которые не входят в стандарт языка – пакеты `javax.net`, `javax.xml` и другие. Но самым большим и интересным является набор интерфейсов, созданных специально для Android. Вот некоторые из его пакетов.

Пакеты `android.view` и `android.widget` отвечают за графический интерфейс пользователя (Graphical User Interface). Они содержат набор встроенных виджетов, таких, как: кнопки и поля ввода, разметки для расположения виджетов на экране. С их помощью можно создать простейшее Android-приложение.

Для работы с примитивами рисования и графическими файлами предназначен пакет `android.graphics`, а с помощью `android.animation` можно создавать несложную анимацию.

Пакет `android.opengl` представляет движок OpenGL, `android.gesture` осуществляет поддержку управления жестами на сенсорном экране, позволяет распознавать жесты и создавать новые.

Большое количество интерфейсов предназначено для коммуникации. Пакет android.net включает стеки сетевых протоколов высокого уровня, таких

как HTTP и SIP, поддержку Wi-Fi. Пакет `android.webkit` – популярный движок веб-браузера, который позволяет легко отображать веб-страницы в приложении. Пакеты `android.bluetooth` и `android.nfc` предоставляют стеки протоколов связи на коротких расстояниях BlueTooth и Near Field Communication соответственно. Пакет `android.telephony` дает доступ к телефонной функциональности – например, отправка SMS.

Для управления прикладными приложениями предназначен пакет `android.app`. Пакет `android.hardware` позволяет обращаться к камере и датчикам, а пакет `android.location` предоставляет информацию о географических координатах устройства, в том числе с помощью датчика GPS.

Пакет `android.media` отвечает за кодирование звуковых и видео потоков, для мобильных устройств это до сих пор вычислительно сложная задача, требующая качественной оптимизации. Пакет `android.database` предоставляет доступ к базам данных.

Структура Android-приложения

Android-приложения могут быть простыми и сложными, но структура приложений всегда будет одинакова. Есть обязательные элементы приложений, а есть опциональные, которые используются по мере необходимости. Android-приложение состоит из нескольких основных компонентов: манифест приложения, набор различных ресурсов и исходный код программы.

Таблица 1.1 демонстрирует обязательные и возможные составляющие структуры Android-приложения.

Таблица 1.1 – Структура Android-приложения

Название	**Описание**	**Необходимость**
gen	Файлы, сгенерированные самой Java. Здесь находится такой важный файл как R.java	Да
AndroidManifest.xml	Файл манифеста AndroidManifest.xml предоставляет системе основную информацию о программе. Каждое приложение должно иметь свой файл манифеста	Да
src	Каталог, в котором содержится исходный код приложения	Да
assets	Произвольное собрание каталогов и файлов	Нет
res	Каталог, содержащий ресурсы приложения. В данном каталоге	Да

	могут находиться подпапки drawable, anim, layout, menu, values, xml и raw (см. ниже)	

Файл манифеста AndroidManifest.xml

Файл манифеста AndroidManifest.xml предоставляет системе основную информацию о программе. Каждое приложение должно иметь свой файл AndroidManifest.xml. Редактировать файл манифеста можно вручную, изменяя XML-код или через визуальный редактор Manifest Editor, который позволяет осуществлять визуальное и текстовое редактирование файла манифеста приложения.

Назначение файла:

- описывает компоненты приложения – Activities, Services, Broadcast receivers и Content providers;
- содержит список необходимых разрешений для обращения к защищенным частям API и взаимодействия с другими приложениями;
- объявляет разрешения, которые сторонние приложения обязаны иметь для взаимодействия с компонентами данного приложения;
- объявляет минимальный уровень API Android, необходимый для работы приложения;
- перечисляет связанные библиотеки.

Корневым элементом манифеста является `<manifest>`. Помимо данного элемента обязательными элементами являются теги `<application>` и `<uses-sdk>`. Элемент `<application>` является основным элементом манифеста и содержит множество дочерних элементов, определяющих структуру и работу приложения. Порядок расположения элементов, находящихся на одном уровне, произвольный. Все значения устанавливаются через атрибуты элементов. Кроме обязательных элементов, упомянутых выше, в манифесте по мере необходимости используются другие элементы. Вот некоторые из них:

- `<manifest>` является корневым элементом манифеста. По умолчанию IDE создает элемент с четырьмя атрибутами:
  - `xmlns:android` определяет пространство имен Android.
  - `package` определяет уникальное имя пакета приложения.
  - `android:versionCode` указывает на внутренний номер версии.
  - `android:versionName` указывает номер пользовательской версии.
- `<permission>` объявляет разрешение, которое используется для ограничения доступа к определенным компонентам или функциональности данного приложения. В этой секции описываются права, которые должны

запросить другие приложения для получения доступа к приложению. Приложение может также защитить свои собственные компоненты (Activities, Services, Broadcast receivers и Content providers) разрешениями. Оно может использовать любое из системных разрешений, определенных Android или объявленных другими приложениями, а также может определить свои собственные разрешения.

- `<uses-permission>` запрашивает разрешения, которые приложению должны быть предоставлены системой для его нормального функционирования. Разрешения предоставляются во время установки приложения, а не во время его работы. Наиболее распространённые разрешения:
    - INTERNET – доступ к Интернету.
    - READ_CONTACTS – чтение (но не запись) данных из адресной книги пользователя.
    - WRITE_CONTACTS – запись (но не чтение) данных в адресную книгу пользователя
    - RECEIVE_SMS – обработка входящих SMS.
    - ACCESS_FINE_LOCATION – точное определение местонахождения при помощи GPS.
- `<uses-sdk>` позволяет объявлять совместимость приложения с указанной версией (или более новыми версиями API) платформы Android. Уровень API, объявленный приложением, сравнивается с уровнем API системы мобильного устройства, на который инсталлируется данное приложение. Атрибуты:
    - `android:minSdkVersion` определяет минимальный уровень API, требуемый для работы приложения. Система Android будет препятствовать тому, чтобы пользователь установил приложение, если уровень API системы будет ниже, чем значение, определенное в этом атрибуте.
    - `android:maxSDKVersion` позволяет определить самую позднюю версию, которую готова поддерживать программа.
    - `targetSDKVersion` позволяет указать платформу, для которой разрабатывалось и тестировалось приложение.
- `<uses-configuration>` указывает требуемую для приложения аппаратную и программную конфигурацию мобильного устройства. Спецификация используется, чтобы избежать инсталляции приложения на устройствах, которые не поддерживают требуемую конфигурацию. Если приложение может работать с различными конфигурациями устройства,

необходимо включить в манифест отдельные элементы <uses-configuration> для каждой конфигурации.

– `<uses-feature>` объявляет определенную функциональность, требующуюся для работы приложения. Таким образом, приложение не будет установлено на устройствах, которые не имеют требуемую функциональность. Например, приложение могло бы определить, что оно требует камеры с автофокусом. Если устройство не имеет встроенную камеру с автофокусом, приложение не будет установлено. Возможные атрибуты:
  - `android.hardware.camera` – требуется аппаратная камера.
  - `android.hardware.camera.autofocus` – требуется камера с автоматической фокусировкой.
– `<supports-screens>` определяет разрешение экрана, требуемое для функционирования приложения. По умолчанию современное приложение с уровнем API 4 или выше поддерживает все размеры экрана и должно игнорировать этот элемент.
– `<application>` один из основных элементов манифеста, содержащий описание компонентов приложения. Содержит дочерние элементы (`<activity>`, `<service>`, `<receiver>`, `<provider>` и другие), которые объявляют каждый из компонентов, входящих в состав приложения. В манифесте может быть только один элемент `<application>`.

Ресурсы

В Android принято хранить такие объекты, как: изображения, строковые константы, цвета, анимацию, стили и тому подобное, за пределами исходного кода. Система поддерживает хранение ресурсов во внешних файлах. Внешние ресурсы легче поддерживать, обновлять и редактировать.

В основном ресурсы хранятся в виде XML-файлов в каталоге res с подкаталогами values, drawable-ldpi, drawable-mdpi, drawable-hdpi, layout. Но также бывают еще два типа ресурсов: raw и assets.

Для удобства система создает идентификаторы ресурсов и использует их в файле **R.java** (класс R, который содержит ссылки на все ресурсы проекта), что позволяет ссылаться на ресурсы внутри кода программы. Статический класс R генерируется на основе заданных ресурсов и создается во время компиляции проекта. Так как файл R генерируется автоматически, то нет смысла его редактировать вручную, потому что все изменения будут утеряны при повторной генерации.

В общем виде ресурсы представляют собой файл (например, изображение) или значение (например, заголовок программы), связанные с создаваемым приложением. Удобство использования ресурсов заключается в

том, что их можно изменять без повторной компиляции или новой разработки приложения. Самыми распространенными ресурсами являются строки (string), цвета (color) и графические рисунки (bitmap).

В таблице 1.2 перечислены основные ресурсы Android-приложения.

Таблица 1.2 – Ресурсы Android-приложения

Тип ресурса	**Размещение**	**Описание**
Цвета	/res/colors/	Идентификатор цвета, указывающий на цветовой код.
Строки	/res/strings/	Строковые ресурсы. В их число также входят строки в формате java и html.
Меню	/res/menus/	Меню в приложении можно задать как XML-ресурсы.
Параметры	/res/values/	Представляет собой параметры или размеры различных элементов.
Изображения	/res/drawable/	Ресурсы-изображения. Поддерживает форматы JPG, GIF, PNG (самый предпочтительный) и другие. Каждое изображение является отдельным файлом. Система также поддерживает stretchable images, в которых можно менять масштаб отдельных элементов, а другие элементы оставлять без изменений.
Отрисовываемые цвета	/res/values/ или /res/drawable/	Представляет цветные прямоугольники, которые используются в качестве фона основных отрисовываемых объектов, например, точечных рисунков.
Анимация	/res/anim/	Android может выполнить простую анимацию на графике или на серии графических изображений.
Произвольные XML-файлы	/res/xml/	В Android в качестве ресурсов могут использоваться произвольные XML-файлы.
Произвольные необработанные ресурсы	/res/raw/	Любые нескомпилированные двоичные или текстовые файлы, например, видео.

В Android имеется еще один каталог, в котором могут храниться файлы, предназначенные для включения в пакет – /assets. Это не ресурсы, а просто необработанные файлы. Этот каталог находится на том же уровне, что и /res. Для файлов, располагающихся в /assets, в R.java не генерируются идентификаторы ресурсов. Для их считывания необходимо указать путь к файлу. Путь к файлу является относительным и начинается с /assets. Этот

каталог, в отличие от подкаталога res/, позволяет задавать произвольную глубину подкаталогов и произвольные имена файлов.

Разметка

В Android-приложениях пользовательский интерфейс построен на View и ViewGroup объектах. Класс ViewGroup является основой для подкласса Layout (разметка).

Разметка (также используются термины: компоновка или макет) хранится в виде XML-файла в папке /res/layout. Это сделано для того, чтобы отделить код от дизайна, как это принято во многих технологиях (HTML и CSS, Visual Studio и Expression Blend). Кроме основной компоновки для всего экрана, существуют дочерние компоновки для группы элементов. По сути, компоновка – это некий визуальный шаблон для пользовательского интерфейса приложения, который позволяет управлять элементами, их свойствами и расположением. В данной практике придется познакомиться со всеми способами размещения.

Android Studio включает в себя специальный редактор для создания разметки двумя способами. Редактор имеет две вкладки: одна позволяет увидеть, как будут отображаться элементы управления, а вторая – создавать XML-разметку вручную (рисунок 1.2).

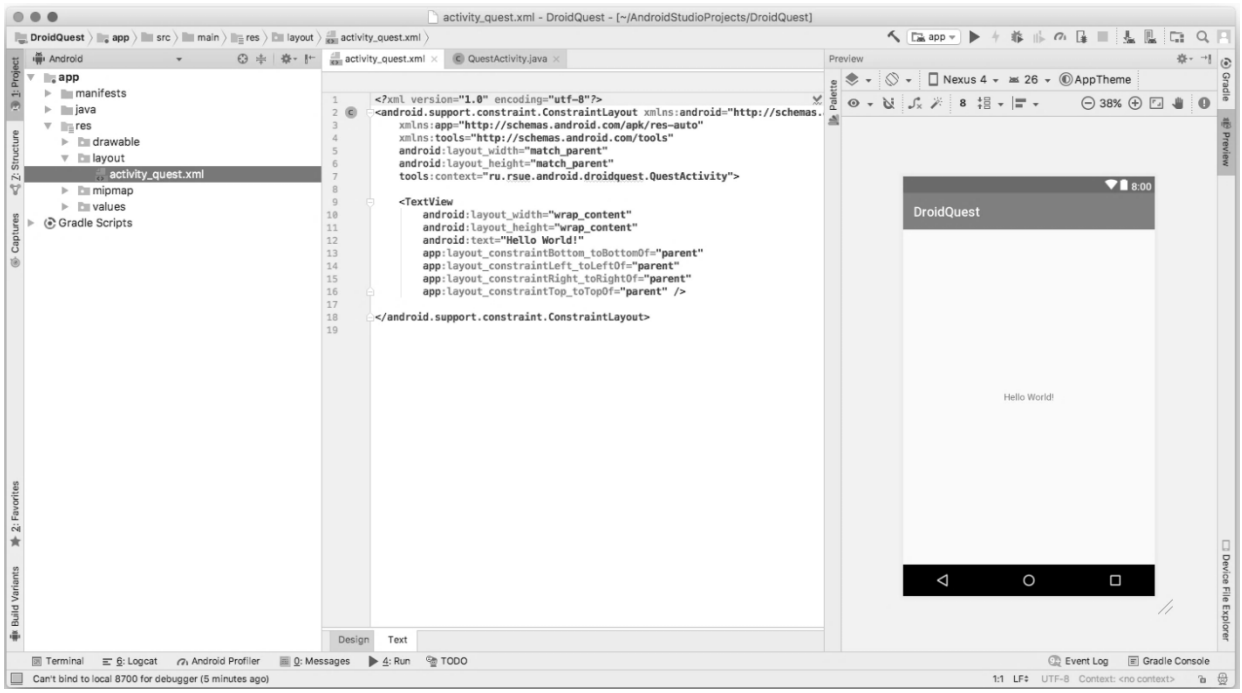

Рисунок 1.2 – Редактор для создания XML-разметки

Создавая пользовательский интерфейс в XML-файле, можно отделить дизайн приложения от программного кода. Можно изменять пользовательский интерфейс в файле разметки без необходимости изменения программного кода. Например, можно создавать XML-разметки для

различных ориентаций экрана мобильного устройства (portrait, landscape), размеров экрана и языков интерфейса. Впрочем, элементы интерфейса можно создавать и программно, когда это необходимо.

Каждый файл разметки должен содержать только один корневой элемент компоновки, который должен быть объектом View или ViewGroup. Внутри корневого элемента можно добавлять дополнительные объекты разметки или дочерние элементы интерфейса, чтобы постепенно формировать иерархию элементов, которую определяет создаваемая разметка.

Существует несколько стандартных типов разметок:

- `FrameLayout` является самым простым типом разметки. Обычно это пустое пространство на экране, которое можно заполнить только дочерним объектом `View` или `ViewGroup`. Все дочерние элементы `FrameLayout` прикрепляются к верхнему левому углу экрана. В разметке `FrameLayout` нельзя определить различное местоположение для дочернего объекта `View`. Последующие дочерние объекты `View` будут просто рисоваться поверх предыдущих представлений, частично или полностью затеняя их, если находящийся сверху объект непрозрачен;
- `LinearLayout` выравнивает все дочерние объекты в одном направлении – вертикально или горизонтально. Направление задается при помощи атрибута ориентации `android:orientation`. Все дочерние элементы помещаются в стек один за другим, так что вертикальный список представлений будет иметь только один дочерний элемент в строке независимо от того, насколько широким он является. Горизонтальное расположение списка будет размещать элементы в одну строку с высотой, равной высоте самого высокого дочернего элемента списка;
- `TableLayout` позиционирует свои дочерние элементы в строки и столбцы. `TableLayout` не отображает линии обрамления для рядов, столбцов или ячеек. `TableLayout` может иметь ряды с разным количеством ячеек. При формировании разметки таблицы некоторые ячейки при необходимости можно оставлять пустыми. `TableLayout` удобно использовать, например, при создании логических игр типа Судоку, Крестики-Нолики и тому подобных;
- `RelativeLayout` позволяет дочерним элементам определять свою позицию относительно родительского представления или относительно соседних дочерних элементов.

Все описываемые разметки являются подклассами `ViewGroup` и наследуют свойства, определенные в классе `View`.

Разметки ведут себя как элементы управления, и их можно группировать. Расположение элементов управления может быть вложенным.

Например, можно использовать `RelativeLayout` в `LinearLayout` и так далее. Однако, слишком большая вложенность элементов управления вызывает проблемы с производительностью.

Компоненты Android-приложения

Каждое Android-приложение запускается в своем собственном процессе. Поэтому приложение изолировано от других запущенных приложений, и неправильно работающее приложение не может беспрепятственно навредить другим Android-приложениям.

Тем не менее главным параметром Android-приложения является возможность использовать компоненты других приложений, если они дают на это соответствующие права. Если нужен некий компонент с прокруткой для отображения текста, и похожий компонент уже реализован в другом приложении, тогда у нас есть возможность использовать реализованный компонент. В этом случае данное приложение не копирует необходимый код к себе и не создает ссылку на него. Вместо этого приложение делает запрос на исполнение части кода другого приложения, где есть нужный нам компонент.

В Android существуют четыре типа компонентов: Activities, Services, Broadcast receivers и Content providers.

Также важно отметить объекты Intents, в Android-приложениях почти все работает благодаря им. Intent – это механизм для описания одной операции (выбрать фотографию, отправить письмо, сделать звонок, запустить браузер и перейти по указанному адресу и другие). Наиболее распространенный сценарий использования Intent – запуск другого Activity в своем приложении.

Activities

Activity представляет собой пользовательский интерфейс для одного действия, которое пользователь может совершить. Например, приложение для обмена текстовыми сообщениями может иметь одно Activity для отображения списка контактов, другое – для написания сообщения выбранному контакту, третье – для просмотра сообщений и еще одно для изменения настроек. Все эти Activities формируют единый пользовательский интерфейс, но не зависят друг от друга.

Activity может находиться в одном из трех состояний:

- active или running – находится на переднем плане и имеет фокус для взаимодействия с пользователем;
- paused – потеряло фокус, но все еще видно пользователю. Сверху находится другое Activity, которое или прозрачно, или закрывает не весь

экран. Приостановленное Activity полностью «живое» (его состояние сохранено), но может быть уничтожено системой в случае нехватки памяти;

- stopped – полностью перекрыто другим Activity. Оно больше не видно пользователю и будет уничтожено системой, когда понадобится память.

Если Activity приостановлено или остановлено, система может удалить его из памяти, либо послать запрос на его завершение, или просто уничтожить его процесс. Когда Activity снова отображается пользователю, его состояние полностью восстанавливается.

Переходя от состояния к состоянию, Activity уведомляет об этом, вызывая следующие методы (рисунок 1.3):

- `void onCreate();`
- `void onStart();`
- `void onRestart();`
- `void onResume();`
- `void onPause();`
- `void onStop();`
- `void onDestroy()`.

Типы процессов в Android-приложении

Жизненный цикл приложения тесно связан с жизненным циклом его процесса. Также он зависит от текущего состояния системы. В случае нехватки памяти Android убивает наименее значимые процессы. Значимость процесса зависит от его типа. Типы процессов, в зависимости от важности, выглядят следующим образом (от наиболее до наименее важных):

- Процесс переднего плана – процесс приложения, с которым пользователь взаимодействует в данный момент. Процесс считается таковым, если его Activity находится на вершине Activity-стека (была вызвана функция `onResume()`, или его Broadcast Receiver работает в настоящее время (в данный момент исполняется приложением `onReceive()`), или же его Service выполняет callback-методы, такие как `onCreate()`, `onStart()` или `onDestroy()`. Как правило, таких процессов очень мало, и они закрываются в самую последнюю очередь.
- Видимый процесс — процесс, который имеет Activity, видимый конечному пользователю в данный момент времени. Процессов, которые выводятся на экран, очень мало, поэтому их работа прерывается только в крайнем случае, если не хватает ресурсов для активных приложений.

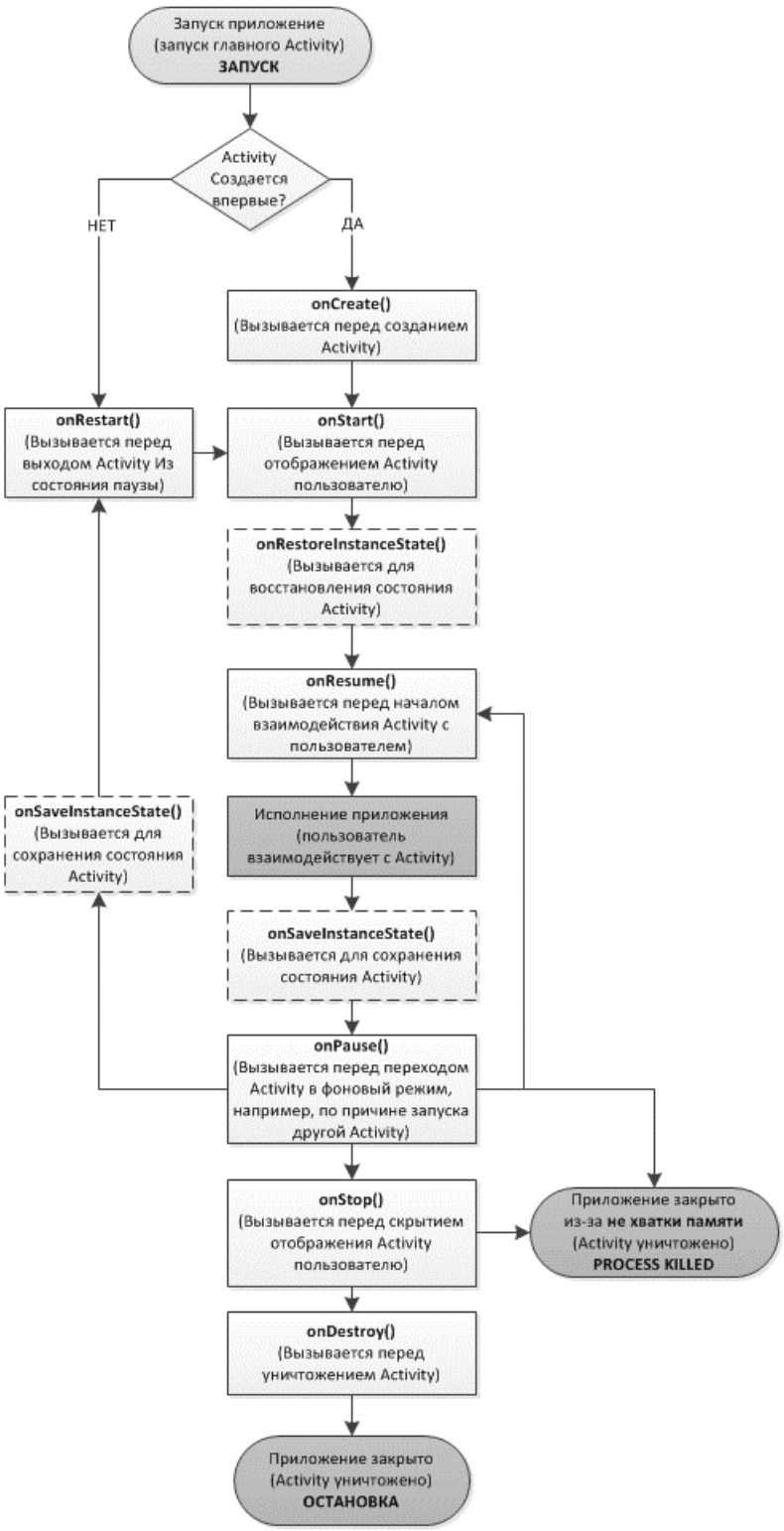

Рисунок 1.3 – Жизненный цикл Activity

– Служебный процесс – процесс, содержащий Service, для которого была вызвана функция `startService()`, при условии, что данный Service сейчас работает.

– Процесс заднего фона — процесс не имеющий видимых пользователю Activity (была вызвана функция `onStop()`). Как правило, существует множество фоновых процессов, работа которых завершается по

принципу «последний запущенный закрывается последним», чтобы освободить ресурсы для приложений, работающих на переднем плане.

Services

Service – это некий процесс, который запускается в фоновом режиме. Как пример, Service может получать данные по сети, выполнять какие-либо длительные вычисления. Хорошим примером Service служит проигрыватель музыки. Пользователь может выбрать любую песню в проигрывателе, включить ее и закрыть плеер, занявшись чем-нибудь другим. Музыка будет проигрываться в фоновом процессе. Service проигрывания музыки будет работать, даже если Activity плеера закрыта.

Подобно Activity, Service имеет свои методы жизненного цикла:

- `void onCreate();`
- `void onStart(Intent intent);`
- `void onDestroy().`

Как и Activities, Services запускаются в главном потоке процесса приложения. По этой причине их следует запускать в отдельном потоке, чтобы они не блокировали другие компоненты или пользовательский интерфейс.

Broadcast receivers

Broadcast receiver – это компонент, который ничего не делает, кроме того, что рассылает и реагирует на широковещательные сообщения. Примером широковещательных компонентов могут быть: сообщения об переходе на летнее/зимнее время, сообщения об минимальном заряде батареи и так далее.

Broadcast receiver не отображает пользовательский интерфейс, но может запустить Activity на полученное сообщение или использовать `NotificationManager` для привлечения внимания пользователя. Привлечь внимание пользователя можно, например, вибрацией устройства, проигрыванием звука или миганием вспышки.

Приемник широковещательных сообщений имеет единственный метод жизненного цикла: `onReceive()`. Когда широковещательное сообщение прибывает для получателя, Android вызывает его методом `onReceive()` и передает в него объект Intent, содержащий сообщение. Приемник широковещательных сообщений является активным только во время выполнения этого метода. Процесс, который в настоящее время выполняет Broadcast receiver, является приоритетным процессом и будет сохранен, кроме случаев критического недостатка памяти в системе.

Когда программа возвращается из `onReceive()`, приемник становится неактивным, и система полагает, что работа объекта Broadcast receiver закончена. Процесс с активным широковещательным получателем защищен от уничтожения системой. Однако процесс, содержащий неактивные компоненты, может быть уничтожен системой в любое время, когда память, которую он потребляет, будет необходима другим процессам.

Content providers

Content providers предоставляют доступ к данным (чтение, добавление, обновление). Content provider может предоставлять доступ к данным не только своему приложению, но и другим. Данные могут размещаться в файловой системе, в базе данных.

Виджеты

Виджет – это объект View, который служит интерфейсом для взаимодействия с пользователем. Иначе виджеты – это обычные элементы управления: кнопки, текстовые поля, флажки, переключатели, списки.

Стандартные элементы имеют привычные свойства: ширину, высоту, цвет и тому подобные. Еще два важных свойства, которые могут влиять на размер и положение дочерних элементов – важность (weight) и выравнивание (gravity). Weight используется для присвоения элементу показателя важности, отличающего его от других элементов, находящихся в контейнере. Предполагается, что в контейнере находятся три элемента управления: первый имеет важность 1 (максимальное возможное значение), а два других имеют значение 0. В этом случае элемент управления, который имеет значение важности 1, займет в контейнере все свободное пространство. Gravity – это ориентация в контейнере. Например, необходимо выровнять текст надписи по правому краю, тогда свойство gravity будет иметь значение right. Набор значений для gravity ограничен: left, center, right, top, bottom, center_vertical, clip_horizontal и еще некоторые.

TextView предназначен для отображения текста без возможности редактирования его пользователем. TextView один из самых используемых виджетов. С его помощью пользователю удобнее ориентироваться в программе. По сути, TextView служит для представления пользователю описательного текста.

Button (кнопка) – один из самых распространенных элементов управления в программировании. Наследуется от Textview и является базовым классом для класса CompoundButton. От класса CompoundButton, в свою очередь, наследуются такие элементы, как: CheckBox, ToggleButton и

RadioButton. На кнопке располагается текст, и на кнопку нужно нажать, чтобы получить результат.

CheckBox является флажком, с помощью которого пользователь может отметить (поставить галочку) определенную опцию. Очень часто флажки используются в настройках, когда нужно выборочно выбрать определенные пункты, необходимые для комфортной работы пользователю.

RadioButton, главная особенность которого состоит в том, что он не используется в одиночестве. Всегда должно быть два и более переключателя, и только один из них может быть выбранным.

ToggleButton по своей функциональности похож на флажок (checkbox) или переключатель (radiobutton). Это кнопка, которая может находиться в одном из двух состояний: активна (On) или неактивна (Off).

Switch – еще один вид переключателей, представляет собой полоску с двумя состояниями, переключиться между которыми можно сдвиганием ползунка.

Spinner похож на выпадающий список. В закрытом состоянии элемент показывает одну строчку, при раскрытии выводит список в виде диалогового окна с переключателями.

ProgressBar (индикатор прогресса) применяется в тех случаях, когда пользователю нужно показать, что программа не зависла, а выполняет продолжительную работу.

SeekBar – обычный слайдер, чтобы пользователь мог передвигать ползунок пальцем на экране. Также можно передвигать ползунок с помощью клавиш-стрелок.

RatingBar показывает значение рейтинга в виде звездочек. Можно установить рейтинг касанием пальца или с помощью клавиш курсора, используя заранее заданное количество звездочек.

Адаптеры

В Android часто используются адаптеры, которые упрощают связывание данных с элементом управления. Адаптеры используются при работе с виджетами: ListView, Spinner и другими.

ListAdapter наследует базовый класс Adapter и служит мостом между данными и ListView. Часто данные могут быть представлены курсорами, но необязательно. Удобство в том, что ListView может отображать любые данные, лишь бы они были завернуты в ListAdapter. ListAdapter имеет несколько подклассов (ArrayAdapter, BaseAdapter, CursorAdapter и другие), которые предназначены для различных целей:

- ArrayAdapter специально предназначен для работы с элементами списка. Он представляет данные в виде массива и добавляет удобный функционал для работы с ними (добавление, удаление, поиск);
- BaseAdapter – очень простой адаптер, обычно используется для заполнения списка статическими данными (которые могут быть взяты из ресурсов);
- CursorAdapter представляет данные для списка через курсор.

Эмулятор

Эмулятор Android – это важный инструмент разработчика. Необходимо изучить его особенности и использовать его на начальном этапе разработки. Однако следует помнить, что эмулятор лишь моделирует общее поведение реального устройства. Поэтому окончательное тестирование необходимо проводить на настоящем телефоне.

Эмулятор создается при помощи Android Virtual Device Manager (AVD Manager). Создавая новое виртуальное устройство, в окне свойств можно задать произвольное название для эмулятора, указать версию API и установить остальные параметры (например, разрешение, плотность пикселей на экране, емкость SD-карты и другие).

С помощью эмулятора можно иметь полноценный доступ к Интернету, настраивать скорость и латентность соединения. Также можно имитировать входящие и исходящие телефонные звонки, и SMS-сообщения. Но в то же время эмулятор не поддерживает виброзвонок, светодиоды, камеру, акселерометр и работу с компасом.

2. ПЕРВОЕ ANDROID-ПРИЛОЖЕНИЕ

В этом разделе будет построено приложение, которое называется DriodQuest. Оно представляет собой викторину на тему: «Хорошо ли пользователь знает Android». Пользователь отвечает на вопрос, нажимая кнопку «Да» или «Нет», а DriodQuest мгновенно сообщает ему результат.

На рисунке 2.1 показан результат нажатия кнопки «Да».

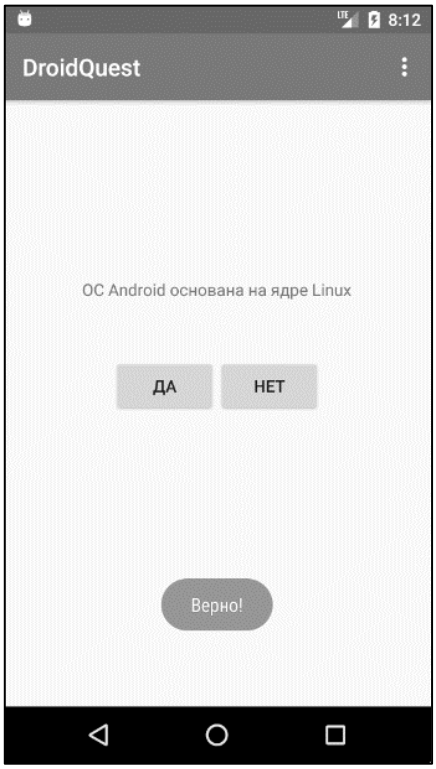

Рисунок 2.1 – Приложение DriodQuest

Основы построения приложения

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

Активность представлена экземпляром Activity — класса 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.

DriodQuest — простое приложение, поэтому в нем используется всего один субкласс Activity с именем QuestActivity. Класс QuestActivity управляет пользовательским интерфейсом, изображенным на рисунке 2.1.

Макет определяет набор объектов пользовательского интерфейса и их расположение на экране. Приложение DriodQuest включает файл макета с именем activity_quest.xml. Разметка XML в этом файле определяет пользовательский интерфейс, изображенный на рисунке 2.1.

Отношения между QuestActivity и activity_quest.xml изображены на рисунке 2.2.

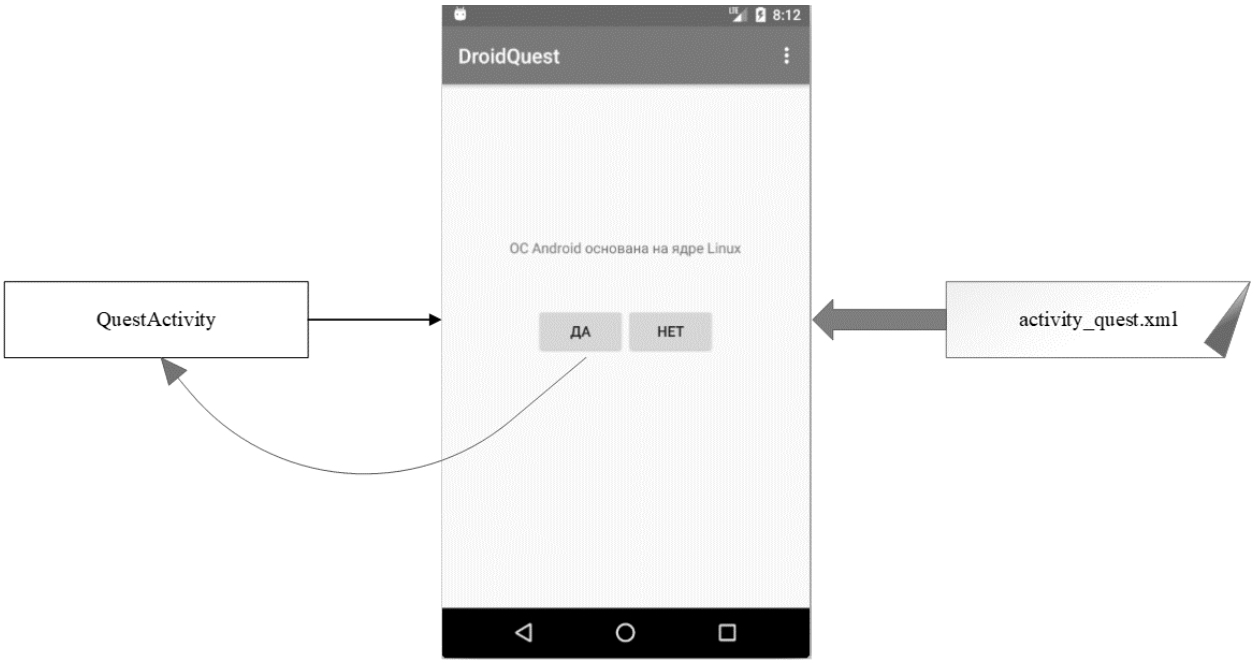

Рисунок 2.2 – QuestActivity управляет интерфейсом, определяемым в файле activity_quest.xml

Создание проекта Android

Проект Android содержит файлы, из которых состоит приложение. Чтобы создать новый проект, надо открыть Android Studio. Если Android Studio запускается на компьютере впервые, то на экране появляется диалоговое окно с приветствием (рисунок 2.3).

Выберитается в диалоговом окне команда *Start a new Android Studio project*. Если диалоговое окно не отображается при запуске, значит, ранее уже создавались другие проекты. В таком случае выполняется команду File→New Project....

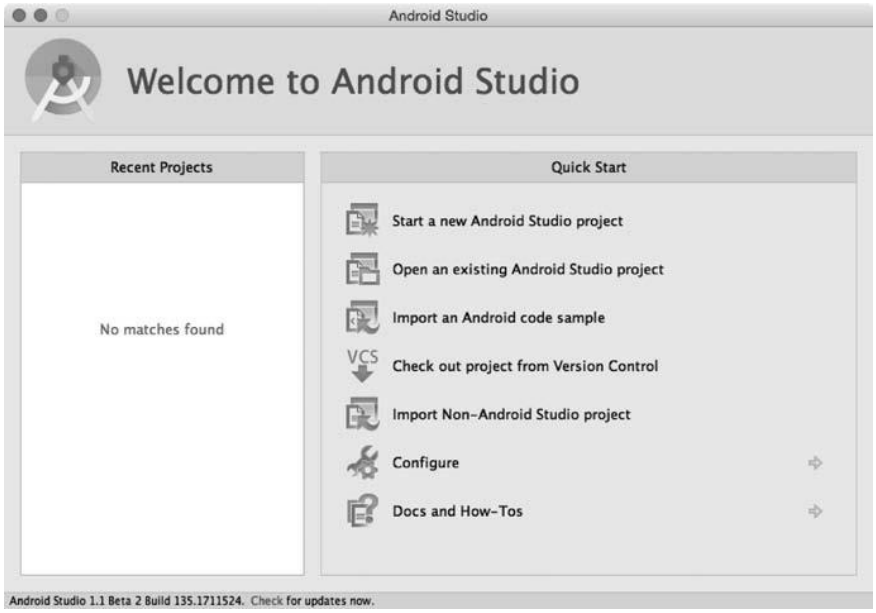

Рисунок 2.3 – Добро пожаловать в Android Studio

Открывается мастер создания проекта. На первом экране мастера вводится имя приложения **DriodQuest** (рисунок 2.4). В поле Company Domain введите строку ***android.rsue.ru*** (*android* – фамилия студента на латинице); сгенерированное имя пакета (Package Name), при этом автоматически меняется на *ru.rsue.android.driodquest*. В поле Project location вводится любая папка файловой системы.

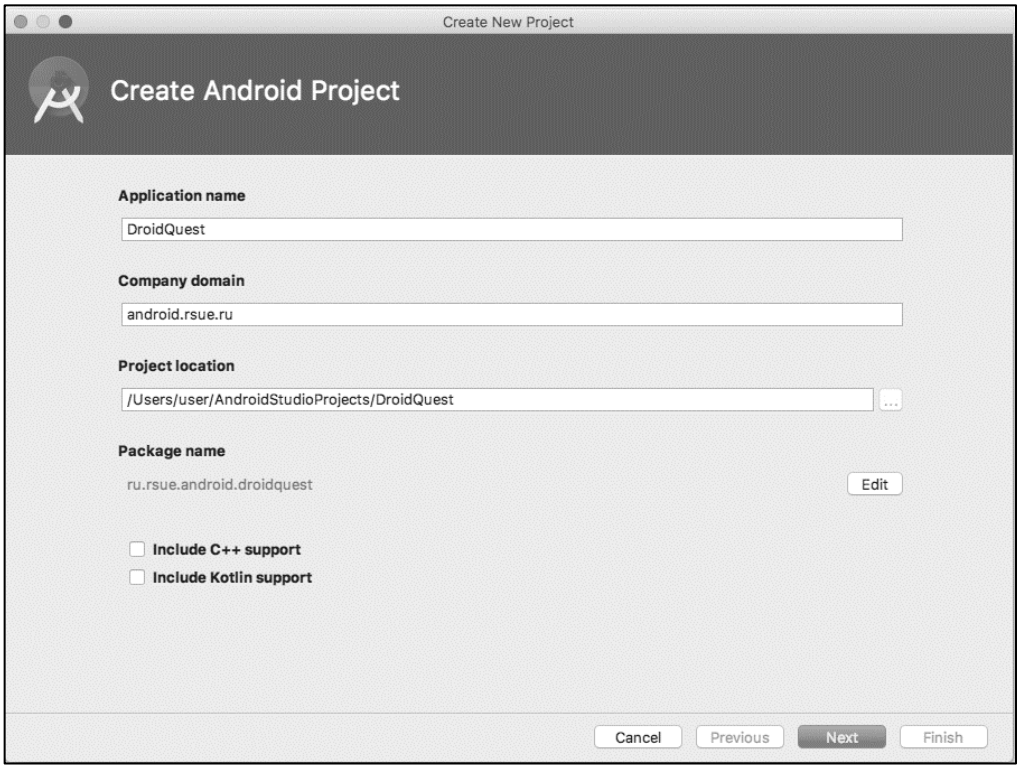

Рисунок 2.4 – Создание нового проекта

В имени пакета используется схема «обратного DNS», согласно которой доменное имя 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.

Щёлкнуть на кнопке Next. На следующем экране можно ввести дополнительную информацию об устройствах, которые необходимо поддерживать. Приложение DriodQuest будет поддерживать только телефоны, поэтому устанавливается только флажок Phone and Tablet. Выбирается в списке минимальная версия SDK API 21: Android 5.0 (Lollipop) (рисунок 2.5).

Рисунок 2.5 – Определение поддерживаемых устройств

Щелкнуть на кнопке Next.

На следующем экране необходимо выбрать шаблон первого экрана DriodQuest (рисунок 2.6). Выбрать пустую активность (Blank Activity / Empty Activity) и щелкнуть на кнопке Next.

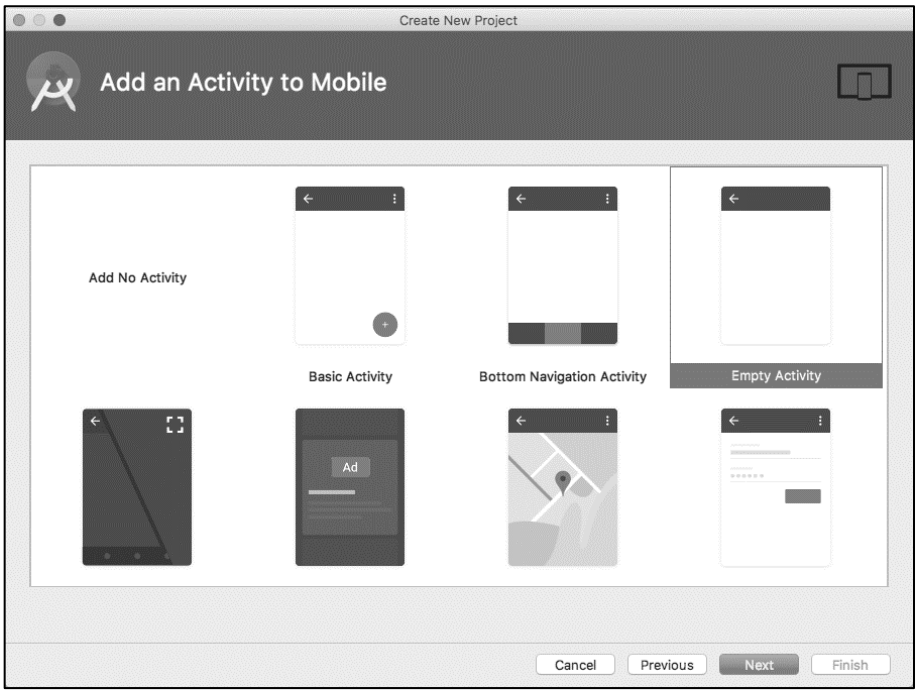

Рисунок 2.6 – Выбор типа активности

В последнем диалоговом окне мастера вводится имя субкласса активности QuestActivity (рисунок 2.7). Суффикс Activity в имени класса. Его присутствие не обязательно, но это очень полезное соглашение, которое стоит соблюдать.

Имя макета автоматически заменяется на activity_quest в соответствии с переименованием активности. Имя макета записывается в порядке, обратном имени активности; в нем используются символы нижнего регистра, а слова разделяются символами подчеркивания.

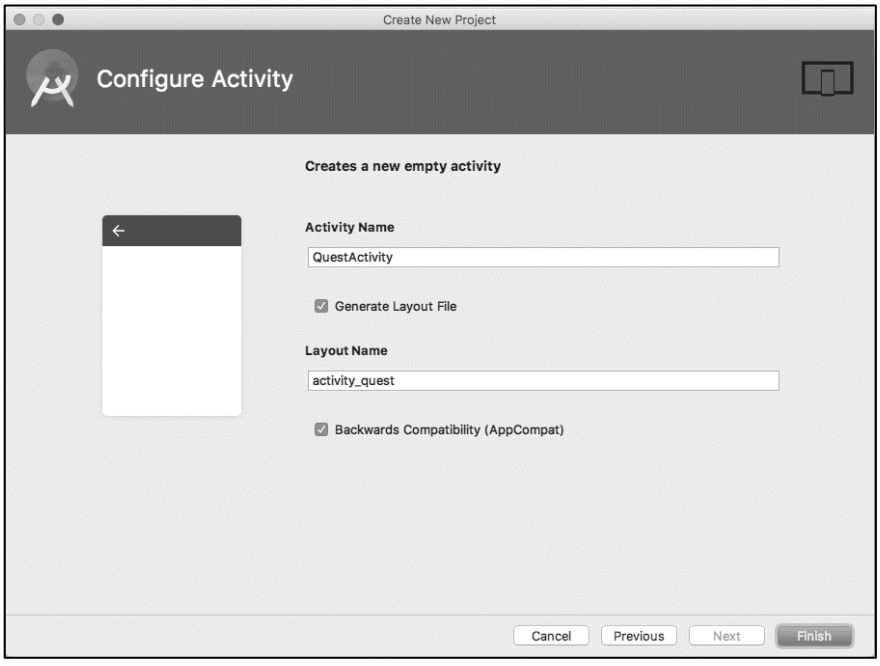

Рисунок 2.7 – Настройка новой активности

Щёлкнуть на кнопке Finish. Среда Android Studio создает и открывает новый проект.

Навигация в Android Studio

Среда Android Studio открывает созданный проект в окне, изображенном на рисунке 2.8.

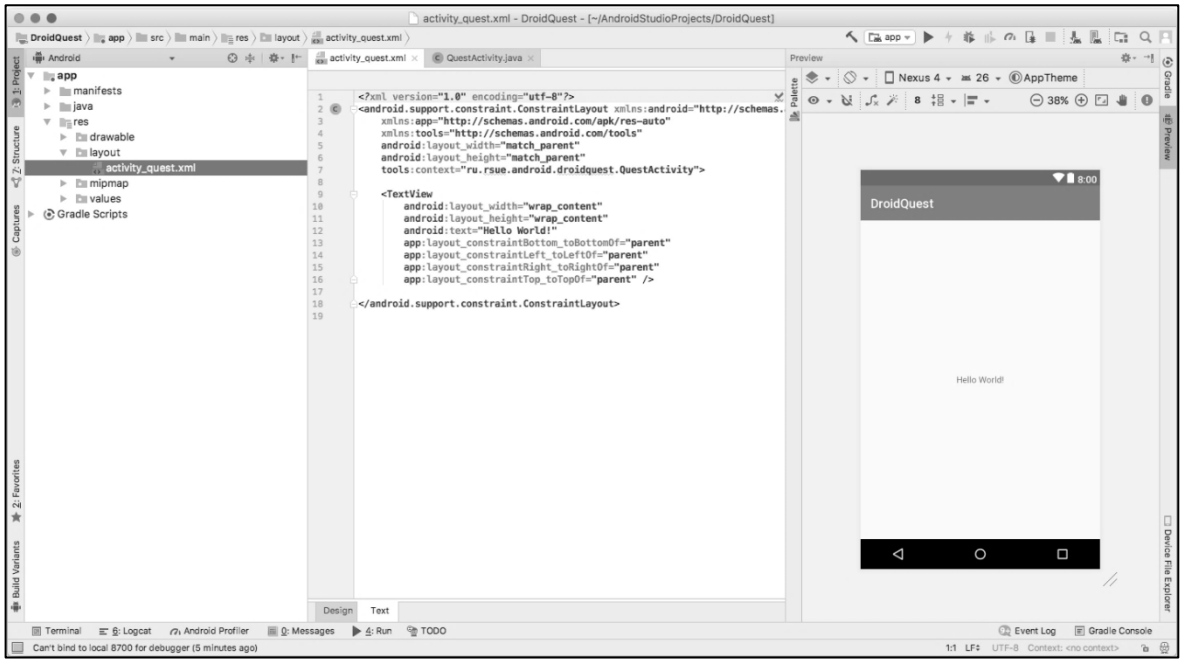

Рисунок 2.8 – Окно нового проекта

Различные части окна проекта называются *панелями*.

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

В середине находится панель *редактора*, в котором Android Studio открывает в файл activity_quest.xml. (Если в редакторе выводится изображение, щёлкнуть на вкладке Text в нижней части панели.) На панели в правой части окна показано, как выглядит содержимое файла в режиме предварительного просмотра.

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, щёлкнуть на серой квадратной кнопке в левом нижнем углу главного окна или выполнить команду View→Tool Buttons.

Построение макета пользовательского интерфейса

На данный момент файл activity_quest.xml определяет разметку для активности по умолчанию. Разметка шаблона часто изменяется, но XML будет выглядеть примерно так, как показано в листинге 2.1.

Листинг 2.1 – Разметка для активности

```
<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=".QuestActivity">
<TextView
    android:text="@string/hello_world"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
</RelativeLayout>
```

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

На рисунке 2.9 показано, как выглядят на экране виджеты RelativeLayout и TextView, определенные в листинге 2.1.

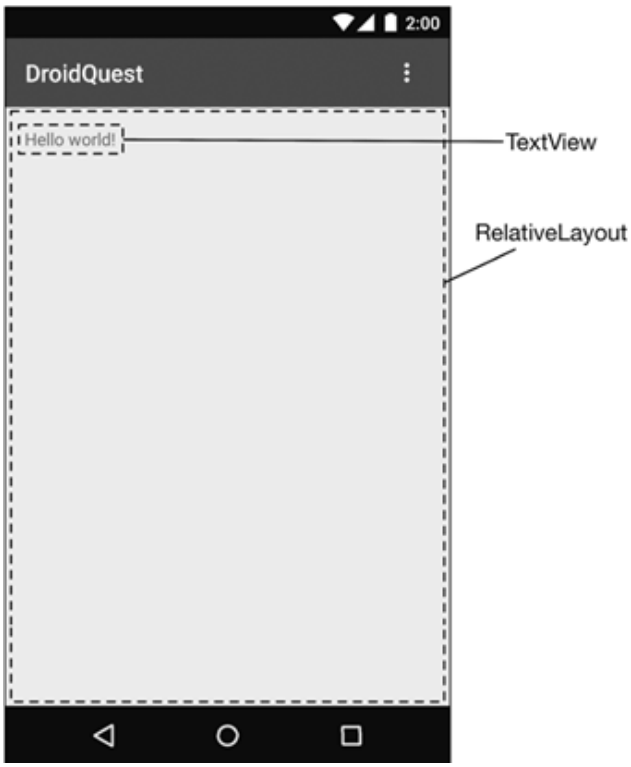

Рисунок 2.9 – Виджеты по умолчанию на экране

В интерфейсе QuestActivity будут задействованы пять виджетов: вертикальный виджет LinearLayout; TextView; горизонтальный виджет LinearLayout; две кнопки Button.

На рисунке 2.10 показано, как из этих виджетов образуется интерфейс QuestActivity.

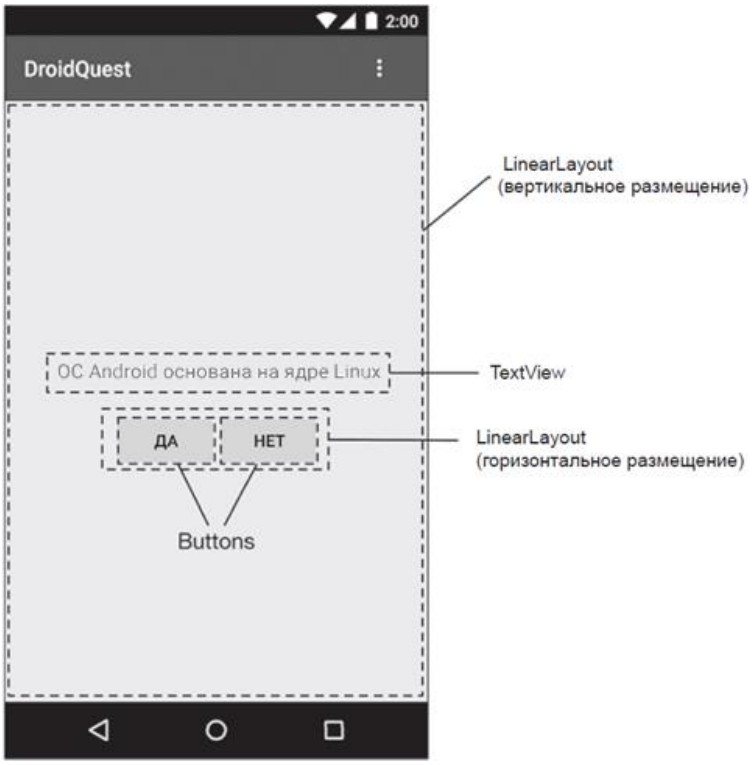

Рисунок 2.10 – Запланированное расположение виджетов на экране

Теперь нужно определить эти виджеты в файле activity_quest.xml.

Внести в файл activity_quest.xml изменения, представленные в листинге 2.2 – Разметка XML, которую нужно удалить, выделена перечеркиванием, а добавляемая разметка XML выделена жирным шрифтом.

Листинг 2.2 – Определение виджетов в XML (activity_quest.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=".QuestActivity">
    <TextView
        android:text="@string/hello_world"
        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>
```

Сравнить XML с пользовательским интерфейсом, изображенным на рисунке 2.10 – 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.

Иерархия представлений

Виджеты входят в иерархию объектов View, называемую *иерархией представлений*. На рисунке 2.11 изображена иерархия виджетов для разметки XML из листинга 2.2.

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 необходимо 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 имеются два собственных потомка Button.

Атрибуты виджетов

Рассмотрим некоторые атрибуты, используемые для настройки виджетов.

`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 задаются значения `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, которое предоставляет Android для размещения иерархии представлений данного приложения.

У других виджетов макета ширине и высоте задается значение wrap_content. На рисунке 2.10 показано, как в этом случае определяются их размеры.

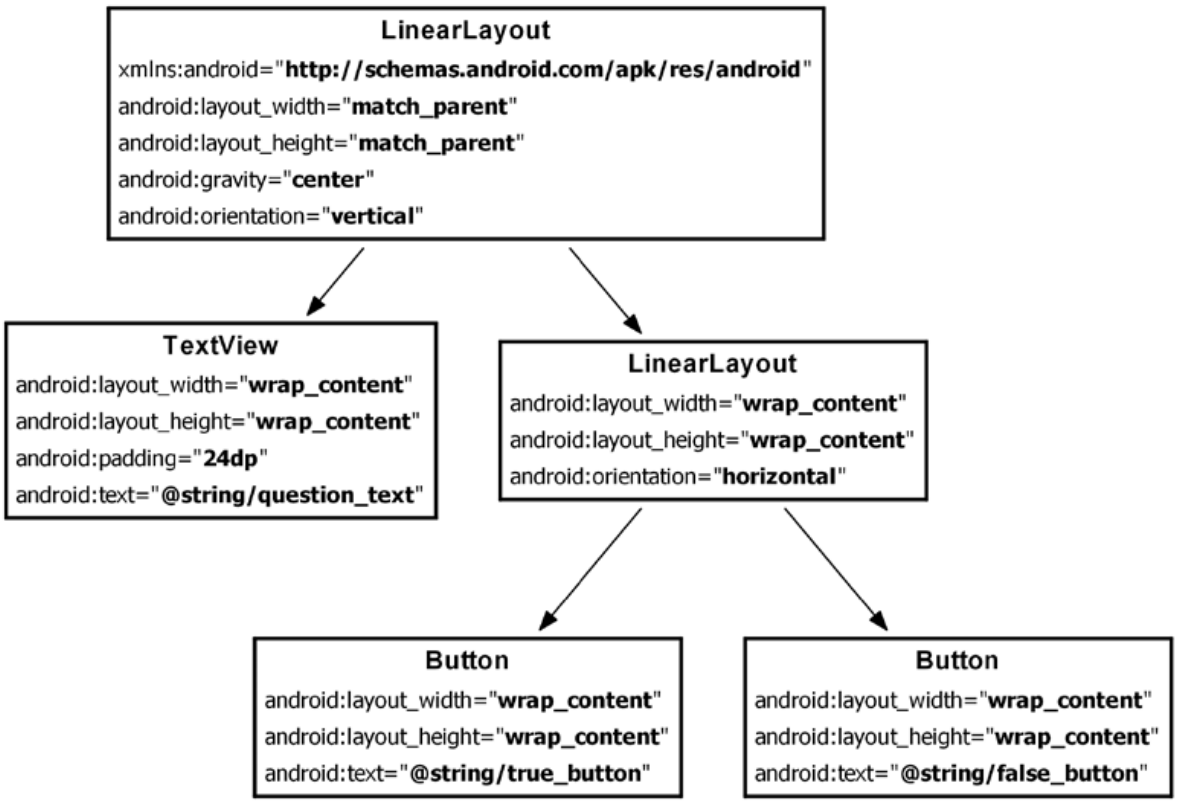

Рисунок 2.11 – Иерархия виджетов и атрибутов

Виджет TextView содержит чуть больше 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 (dp — это пикселы, не зависящие от плотности (density-independent pixels)).

`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 определяет 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 этом 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.

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 ссылается activity_quest.xml, еще не существуют. Это необходимо исправить.

Создание строковых ресурсов

Каждый проект включает строковый файл по умолчанию с именем strings.xml. Найти в окне Project каталог app/res/values, раскрыть его и открыть файл strings.xml.

В шаблон уже включено несколько строковых ресурсов. Удалить неиспользуемую строку с именем hello_world и добавить три новые строки для макета.

Листинг 2.3 – Добавление строковых ресурсов (strings.xml)

```
<resources>
      <string name="app_name">DriodQuest</string>
      <string name="hello_world">Hello world!</string>
      <string name="question_text">ОС Android основана на ядре
            Linux</string>
      <string name="true_button">Да</string>
      <string name="false_button">Нет</string>
      <string name="action_settings">Settings</string>
</resources>
```

Теперь по ссылке @string/false_button в любом файле XML проекта DriodQuest надо получать строковый литерал "Нет" на стадии выполнения.

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

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

Предварительный просмотр макета

Макет готов, и его можно просмотреть в графическом конструкторе (рисунок 2.12). Прежде всего надо убедиться в том, что файлы сохранены и не содержат ошибок. Затем вернуться к файлу activity_quest.xml и открыть панель Preview при помощи вкладки в правой части редактора.

От разметки XML к объектам View

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

В окне инструментов Project открыть каталог app/java, а затем содержимое пакета ru.rsue.android.driodquest. Открыть файл QuestActivity.java и просмотреть его содержимое (листинг 2.4).

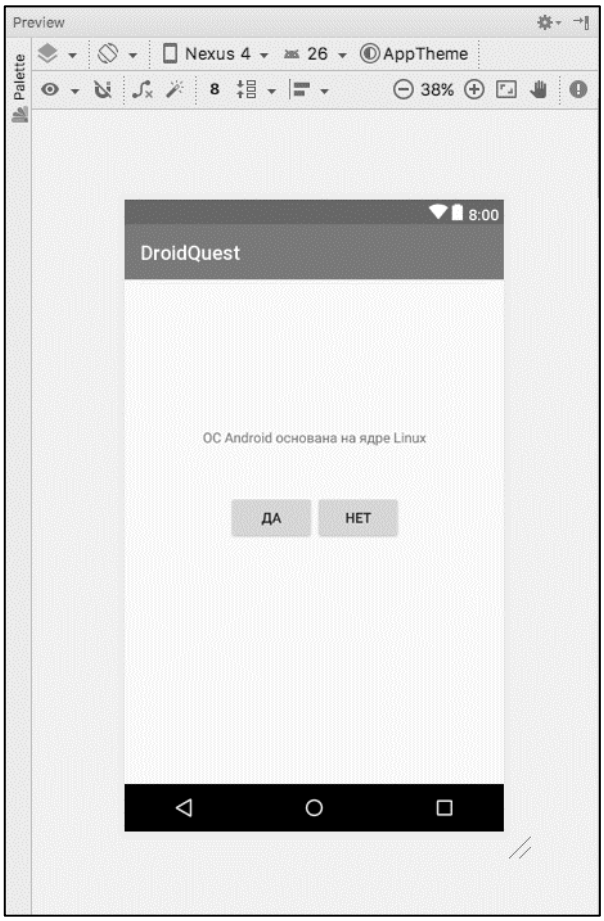

Рисунок 2.12 – Предварительный просмотр в графическом конструкторе макетов (activity_quest.xml)

Листинг 2.4 – Файл класса QuestActivity по умолчанию (QuestActivity.java)

```
package ru.rsue.android.driodquest;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
public class QuestActivity extends AppCompatActivity {
      @Override
      protected void onCreate(Bundle savedInstanceState) {
           super.onCreate(savedInstanceState);
           setContentView(R.layout.activity_quest);
      }
}
```

(AppCompatActivity — это субкласс, наследующий от класса Android Activity и обеспечивающий поддержку старых версий Android.)

(Если не все директивы import отображены, щелкнуть на знаке ⊕ слева от первой директивы import, чтобы раскрыть список.)

Файл может содержать три метода Activity: `onCreate(Bundle)`, `onCreateOptionsMenu(Menu)` и `onOptionsItemSelected(MenuItem)`.

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 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, необходимо передать идентификатор ресурса макета.

Ресурсы и идентификаторы ресурсов

Для обращения к ресурсу в коде используется его идентификатор ресурса. Макету приложения назначен идентификатор ресурса R.layout.activity_quest.

Чтобы просмотреть текущие идентификаторы ресурсов проекта DriodQuest, необходимо сначала изменить режим представления проекта. По умолчанию Android Studio использует режим представления Android (рисунок 2.13). В этом режиме истинная структура каталогов проекта Android скрывается, чтобы можно было сосредоточиться на тех файлах и папках, которые чаще всего нужны программисту.

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

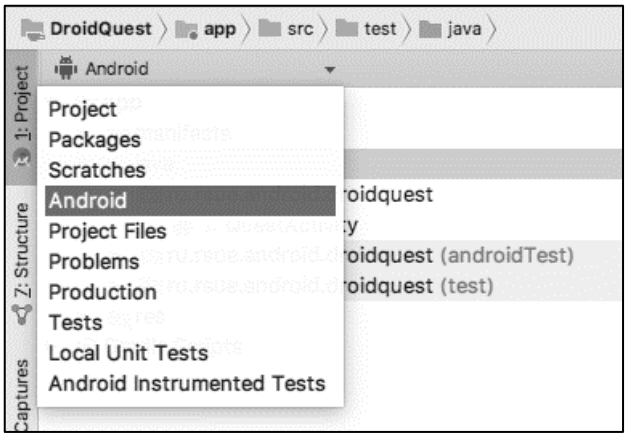

Рисунок 2.13 – Изменение режима представления проекта

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

Листинг 2.5 – Текущие идентификаторы DriodQuest

```
/* AUTO-GENERATED FILE. DO NOT MODIFY.
... */
package ru.rsue.android.driodquest;
public final class R {
      public static final class anim {
          ...
      }
      ...
      public static final class id {
          ...
      }
      public static final class layout {
          ...
          public static final int activity_quest=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 correct_toast=0x7f0a0011;
          public static final int false_button=0x7f0a0012;
          public static final int incorrect_toast=0x7f0a0013;
          public static final int question_text=0x7f0a0014;
          public static final int true_button=0x7f0a0015;
      }
}
```

Файл R.java может быть довольно большим; в листинге 2.5 значительная часть его содержимого не показана.

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

Строкам также назначаются идентификаторы ресурсов. В приложении еще не было ссылок 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_quest.xml. Не каждому виджету нужен идентификатор ресурса.

Прежде чем генерировать идентификаторы ресурсов, переключитесь обратно в режим представления Android. Чтобы сгенерировать идентификатор ресурса для виджета, включите в определение виджета

атрибут android:id. В файле activity_quest.xml добавьте атрибут android:id для каждой кнопки.

Листинг 2.6 – Добавление идентификаторов кнопок (activity_quest.xml)

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

Знак + присутствует в значениях android:id, но не в значениях android:text. Это связано с тем, что идентификаторы *создаются*, а на строки только *ссылаемся*.

Подключение виджетов к программе

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

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

Листинг 2.7 – Добавление полей (QuestActivity.java)

```
public class QuestActivity extends AppCompatActivity {
      private Button mTrueButton;
      private Button mFalseButton;
      @Override
      protected void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_quest);
    }
}
```

Обратить внимание на префикс **m** у имен двух полей (переменных экземпляров). Этот префикс соответствует схеме формирования имен Android.

Навести 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 QuestActivity.java. Ввести следующую директиву импортирования в начале файла:

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

А можно пойти по простому пути и поручить эту работу Android Studio. Нажать Alt+Enter — волшебство IntelliJ приходит на помощь. Новая директива import теперь появляется под другими директивами в начале файла. Этот прием часто бывает полезным, если код работает не так, как положено.

Теперь можно подключить 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 QuestActivity.java по идентификаторам ресурсов кнопок можно получить заполненные объекты и присвоить их полям. Возвращенный объект View перед присваиванием необходимо преобразовать в Button.

Листинг 2.8 – Получение ссылок на виджеты (QuestActivity.java)

```
public class QuestActivity extends AppCompatActivity {
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_quest);
        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 приложение ожидает наступления конкретного события, 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 не придется писать собственные реализации. В данном случае прослушиваемым событием является «щелчок» на кнопке, поэтому слушатель должен реализовать интерфейс View.OnClickListener.

В файле QuestActivity.java включить следующий фрагмент кода в метод onCreate(…) непосредственно после присваивания.

Листинг 2.9 – Назначение слушателя для кнопки True (QuestActivity.java)

```
...
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_quest);
    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 появится ошибка «View cannot be resolved to a type», надо воспользоваться комбинацией Alt+Enter для импортирования класса View.)

В листинге 2.9 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 будут реализованы в виде анонимных внутренних классов. В этом случае реализация методов слушателя находится непосредственно там, где это необходимо, и уменьшаются затраты 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)`. Интерфейс слушателя требует, чтобы метод `onClick(View)` был реализован, но не устанавливает никаких правил относительно того, *как именно* он будет реализован.

Следует назначить аналогичного слушателя для кнопки False.

Листинг 2.10 – Назначение слушателя для кнопки False (QuestActivity.java)

```
mTrueButton.setOnClickListener(new View.OnClickListener() {
          ...
      });
      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

В данном 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 (рисунок 2.14).

Рисунок 2.14 – Уведомление с информацией для пользователя

Уведомления будут сообщать пользователю, правильно ли он ответил на вопрос.

Для начала следует вернуться к файлу strings.xml и добавить строковые ресурсы, которые будут отображаться в уведомлении.

Листинг 2.11 – Добавление строк уведомлений (strings.xml)

```
<resources>
    ...
    <string name="false_button">Нет</string>
    <string name="correct_toast">Верно!</string>
    <string name="incorrect_toast">Не верно!</string>
    <string name="menu_settings">Settings</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, вызвать Toast.show(), чтобы уведомление появилось на экране.

В классе QuestActivity вызов `makeText(…)` будет присутствовать в слушателе каждой кнопки (листинг 2.12).

Автозавершение

Начать вводить новый код из листинга 2.12 – Когда будет набрана точка после класса Toast, на экране появляется список методов и констант класса Toast.

Чтобы выбрать одну из рекомендаций, надо использовать клавиши ↑ или ↓ и нажать клавиши Tab или Return/Enter. Выбрать в списке рекомендаций метод `makeText(Context context, int resID, int duration)`. Механизм автозавершения добавляет полный вызов метода.

Задать параметры метода makeText так, как показано в листинге 2.12.

Листинг 2.12 – Создание уведомлений (QuestActivity.java)

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

В вызове `makeText(…)` экземпляр QuestActivity передается в аргументе Context. Но просто передать this нельзя. В этом месте кода определяется анонимный класс, где `this` обозначает `View.OnClickListener`.

Благодаря использованию автозавершения не придется ничего специально делать для импортирования класса Toast. Когда происходит соглашение на рекомендацию автозавершения, необходимые классы импортируются автоматически.

Сохраните внесенные изменения.

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), выполнить команду Tools→Android→AVD Manager. Когда на экране появится окно AVD Manager, щелкнуть на кнопке Create Virtual Device... в левой части этого окна.

Открывается диалоговое окно с многочисленными параметрами настройки виртуального устройства. Выбрать эмуляцию устройства Nexus 5, как показано на рисунке 2.15. Щёлкнуть на кнопке Next.

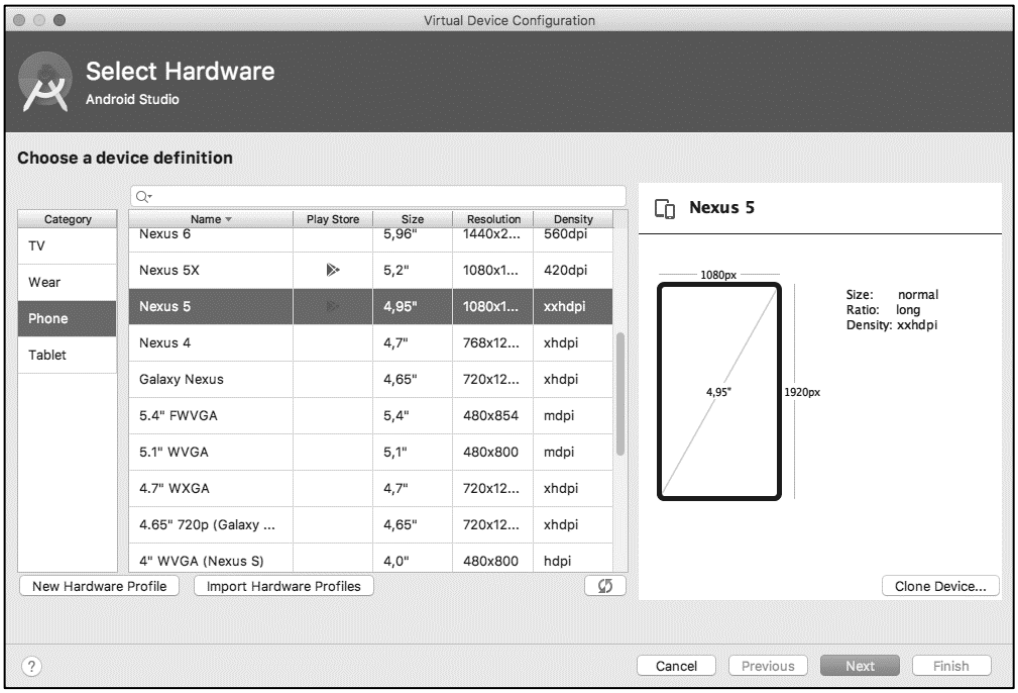

Рисунок 2.15 – Выбор виртуального устройства

На следующем экране выбрать образ системы, на основе которого будет работать эмулятор. Выберите эмулятор x86 Marshmallow и щелкнуть на кнопке Next (рисунок 2.16).

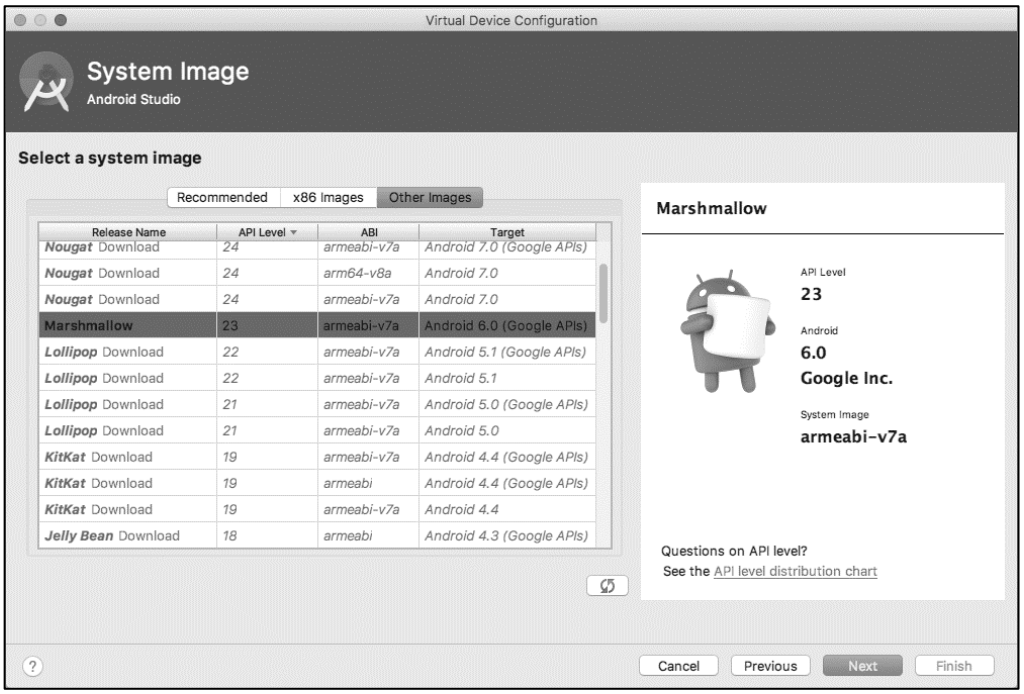

Рисунок 2.16 – Выбор образа системы

Наконец, просмотреть и при необходимости изменить свойства эмулятора (впрочем, свойства существующего эмулятора также можно изменить позднее). Пока необходимо присвоить эмулятору имя, и щёлкнуть на кнопке Finish (Рисунок 2.17).

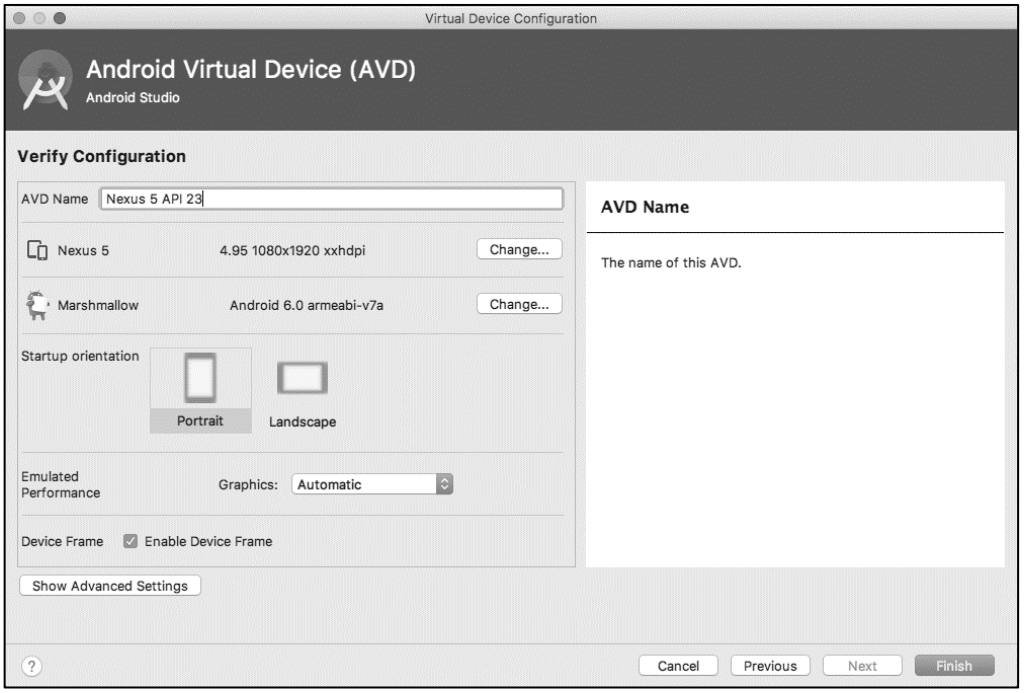

Рисунок 2.17 – Настройка свойств эмулятора

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

Возможно, запуск эмулятора потребует некоторого времени, но вскоре приложение DriodQuest запустится на созданном виртуальном устройстве. Понажимать кнопки и оценить уведомления. (Если приложение запускается, когда вас нет поблизости, возможно, вам придется разблокировать AVD после возвращения. AVD работает как настоящее устройство и блокируется после некоторого времени бездействия.)

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

Сравнить свой код с кодом раздела и попытаться найти источник проблемы. Затем снова запустите приложение.

Не закрывать эмулятор; не стоит ждать, пока он загружается, при каждом запуске, а можно остановить приложение кнопкой Back (∨-образная стрелка), а затем снова запустить приложение из Android Studio, чтобы протестировать изменения.

3. ANDROID И МОДЕЛЬ MVC

В данном разделе приложение DriodQuest будет обновлено путем включения в него дополнительных вопросов. Для этого в проект DriodQuest будет добавлен класс Question. Экземпляр этого класса инкапсулирует один вопрос с ответом «да/нет». Затем будет создан массив объектов Question, с которым будет работать класс QuestActivity. На рисунке 3.1 показан результат работы доработанного приложения.

Рисунок 3.1 – Доработанное приложение DriodQuest

Создание нового класса

В окне инструментов Project щёлкнуть правой кнопкой мыши на пакете ru.rsue.android.driodquest и выбрать команду New→Java Class. Ввести имя класса Question и щелкнуть на кнопке OK (рисунок 3.2).

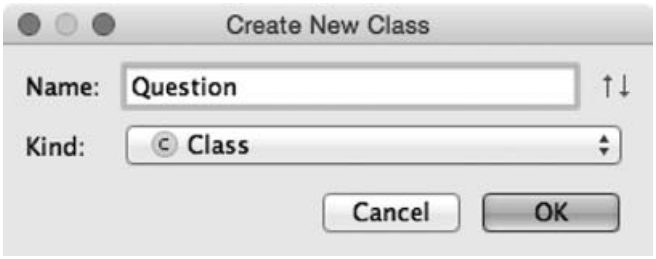

Рисунок 3.2 – Создание класса Question

Добавить в файл Question.java два поля и конструктор:

Листинг 3.1 – 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).

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.

Открыть окно настроек Android Studio (меню File→Settings). Открыть раздел Editor, затем раздел Code Style. Выбрать категорию Java и перейти на вкладку Code Generation.

В таблице Naming найти строку Field (рисунок 3.3) и в поле Naming Prefix ввести префикс m для полей. Затем добавить префикс s для статических полей в строке Static field.

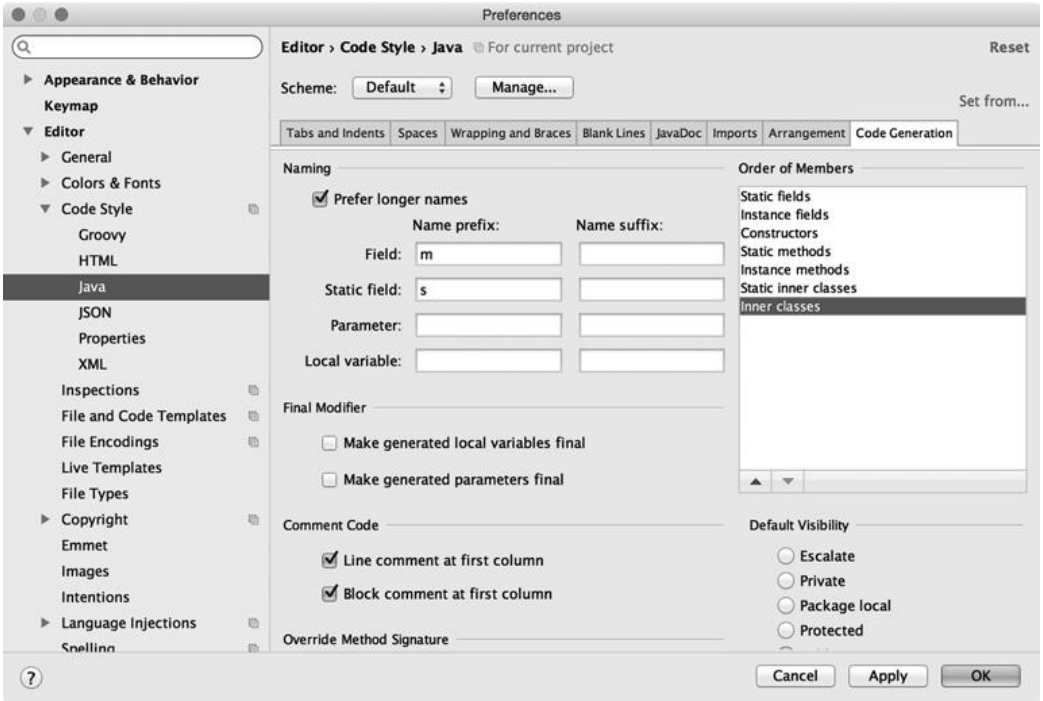

Рисунок 3.3 – Настройка стиля оформления кода Java

Щелкнуть на кнопке OK.

Вернуться к файлу Question.java, щелкнуть правой кнопкой мыши после конструктора и выбрать команду Generate…→Getter And Setter. Выбрать поля mTextResID и mAnswerTrue, затем щелкнуть на кнопке OK, чтобы сгенерировать get- и set-метод для каждой переменной.

Листинг 3.2 – 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.

Обновление уровня представления

Обновить уровень представления DriodQuest и включить в него кнопку «Далее».

В Android объекты уровня представления обычно заполняются на основе разметки XML в файле макета. Весь макет DriodQuest определяется в файле activity_quest.xml. В него следует внести изменения, представленные на рисунке 3.4. (Для экономии места на рисунке не показаны атрибуты виджетов, оставшихся без изменений.)

Итак, на уровне представления необходимо внести следующие изменения:

– удалить атрибут android:text из TextView. Жестко запрограммированный текст вопроса не должен присутствовать в определении;

– назначить TextView атрибут android:id. Идентификатор ресурса необходим виджету, для того чтобы задать его текст в коде QuestActivity;

– добавить новый виджет Button как потомка корневого элемента LinearLayout.

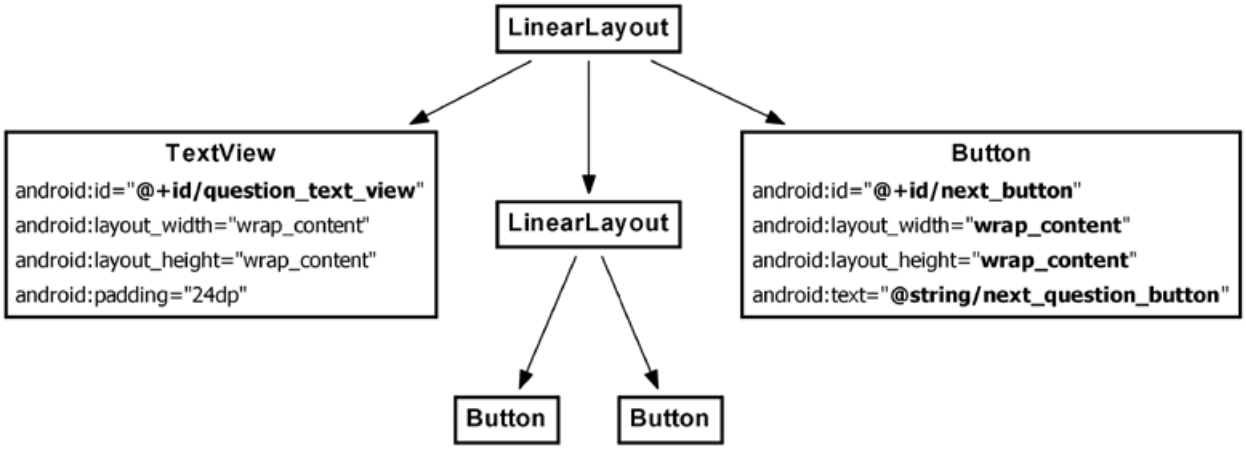

Рисунок 3.4 – Добавление новой кнопки

Вернуться к файлу activity_quest.xml и выполнить все перечисленные действия.

Листинг 3.3 – Новая кнопка и изменения в TextView (activity_quest.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>
```

Сохранить файл activity_quest.xml. Возможно, на экране появится ошибка с сообщением об отсутствующем строковом ресурсе.

Вернуться к файлу res/values/strings.xml. Удалить строку вопроса и добавить строку для новой кнопки.

Листинг 3.4 – Обновленные строки (strings.xml)

```
...
    <string name="app_name">DriodQuest</string>
    ~~<string name="question_text">ОС Android основана на ядре Linux
        </string>~~
    <string name="true_button">Да</string>
```

```
    <string name="false_button">Нет</string>
    <string name="next_button">Далее</string>
    <string name="correct_toast">Верно!</string>
...
```

Далее в файл strings.xml необходимо добавить строки с вопросами по ОС Android, которые будут предлагаться пользователю.

Листинг 3.5 – Добавление строк вопросов (strings.xml)

```
...
    <string name="incorrect_toast">Не верно!</string>
    <string name="action_settings">Settings</string>
    <string name="question_android">ОС Android основана на ядре
        Linux.</string>
    <string name="question_linear">Разметка LinearLayout позиционирует
        свои дочерние элементы в строки и столбцы.</string>
    <string name="question_service">Компонент Android-приложения Service
        предоставляют доступ к данным.</string>
    <string name="question_res">Ресурсы приложения содержатся в
отдельном
        файле XML.</string>
    <string name="question_manifest">AndroidManifest.xml предоставляет
        системе основную информацию о программе.</string>
...
```

Сохранить файлы. Вернуться к файлу activity_quest.xml и ознакомиться с изменениями макета в графическом конструкторе.

Обновление уровня контроллера

В предыдущей работе в контроллере DriodQuest — QuestActivity — не происходило почти ничего. Он отображал макет, определенный в файле activity_quest.xml, назначал слушателей для двух кнопок и организовывал выдачу уведомлений.

Теперь, когда появились дополнительные вопросы, классу QuestActivity придется приложить дополнительные усилия для связывания уровней модели и представления DriodQuest.

Открыть файл QuestActivity.java. Добавить переменные для TextView и новой кнопки Button. Также создать массив объектов Question и переменную для индекса массива.

Листинг 3.6 – Добавление переменных и массива Question (QuestActivity.java)

```
public class QuestActivity extends AppCompatActivity {
    private Button mTrueButton;
    private Button mFalseButton;
    private Button mNextButton;
    private TextView mQuestionTextView;
```

```
    private Question[] mQuestionBank = new Question[] {
        new Question(R.string.question_android, true),
        new Question(R.string.question_linear, false),
        new Question(R.string.question_service, false)
        new Question(R.string.question_res, true),
        new Question(R.string.question_manifest, true),
    };
    private int mCurrentIndex = 0;
...
```

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

Следует использовать mQuestionBank, mCurrentIndex и методы доступа Question для вывода на экран серии вопросов.

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

Листинг 3.7 – Подключение виджета TextView (QuestActivity.java)

```
public class QuestActivity extends AppCompatActivity {
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_quest);
        mQuestionTextView =
        (TextView)findViewById(R.id.question_text_view);
        int question = mQuestionBank[mCurrentIndex].getTextResID();
        mQuestionTextView.setText(question);
        mTrueButton = (Button)findViewById(R.id.true_button);
        ...
    }
}
```

Сохранить файлы и проверить возможные ошибки. Запустить программу DriodQuest. Первый вопрос из массива должен отображаться в виджете TextView.

Получить ссылку на кнопку, назначить ей слушателя `View.OnClickListener`. Этот слушатель будет увеличивать индекс и обновлять текст TextView.

Листинг 3.8 – Подключение новой кнопки (QuestActivity.java)

```
public class QuestActivity extends AppCompatActivity {
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_quest);
        ...
```

```
        mFalseButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(QuestActivity.this,
                    R.string.correct_toast,
                Toast.LENGTH_SHORT).show();
            }
        });
        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 3.9, и вызвать этот метод в слушателе mNextButton и в конце onCreate(Bundle) для исходного заполнения текста в представлении активности.

Листинг 3.9 – Инкапсуляция в методе updateQuestion() (QuestActivity.java)

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

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

Запустить DriodQuest и протестировать новую кнопку «Далее».

В текущем состоянии приложение DriodQuest считает, что на все вопросы ответ должен быть отрицательным; исправить этот недостаток. Для этого будет реализован закрытый метод для инкапсуляции кода вместо того, чтобы вставлять одинаковый код в двух местах.

Сигнатура метода, который будет добавлен в QuestActivity, 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.

Включить в файл QuestActivity.java реализацию checkAnswer(boolean), приведенную в листинге 3.10.

Листинг 3.10 – Добавление метода checkAnswer(boolean) (QuestActivity.java)

```
public class QuestActivity extends AppCompatActivity {
    ...
    private void updateQuestion() {
        ...
    }
    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();
    }
    ...
}
```

Включить в слушателя кнопки вызов checkAnswer(boolean), как показано в листинге 3.11.

Листинг 3.11 – Вызов метода checkAnswer(boolean) (QuestActivity.java)

```
public class QuestActivity 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(QuestActivity.this,~~
            ~~R.string.incorrect_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(QuestActivity.this, R.string.correct_toast,~~
            ~~Toast.LENGTH_SHORT).show();~~
            checkAnswer(false);
        }
    });
    mNextButton = (Button)findViewById(R.id.next_button);
    ...
    }
}
```

Программа DriodQuest снова готова к работе. Запустить ее на реальном устройстве.

Запуск на устройстве

В этом разделе будет выполнена настройка системы, устройства и приложения для выполнения DriodQuest на физическом устройстве.

Подключение устройства

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

Настройка устройства для разработки

Чтобы тестировать приложения на своем устройстве, необходимо разрешить для устройства отладку USB следующим образом:

– на устройствах с Android 4.2 и выше флажок Developer options по умолчанию не отображается. Чтобы включить его, надо открыть меню Settings→About Tablet/Phone и нажать Build Number семь раз. После этого вернуться в меню Settings, найдите раздел Developer options и установить флажок USB debugging;

– на устройствах с Android 4.0 или 4.1 открыть меню Settings→Developer options;

– на устройствах с версиями Android до 4.0 открыть меню Settings→Applications→Development и найти флажок USB debugging.

Как видно, настройки серьезно различаются между устройствами. Если все же возникнут проблемы с включением отладки на устройстве, следует обратиться за помощью по адресу *http://developer.android.com/tools/device.html*.

Чтобы убедиться в том, что устройство было успешно опознано, открыть режим представления Devices. Для этого проще всего выбрать панель Android в нижней части окна Android Studio. На панели должен находиться раскрывающийся список подключенных устройств (рисунок 3.5). В списке должны присутствовать как AVD, так и физическое устройство.

Рисунок 3.5 – Просмотр списка подключенных устройств

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

Запустить DriodQuest так, как это делалось ранее. Android Studio предлагает выбор между запуском на виртуальном устройстве и на физическом устройстве, подключенном к системе. Выбрать физическое устройство. DriodQuest запускается на выбранном устройстве.

Если Android Studio не предлагает выбрать устройство, а DriodQuest запускается в эмуляторе, проверить описанную ранее процедуру и убедитесь в том, что устройство подключено. Затем проверить правильность

конфигурации запуска; чтобы изменить параметры конфигурации, выбрать раскрывающийся список в верхней части окна (рисунок 3.6).

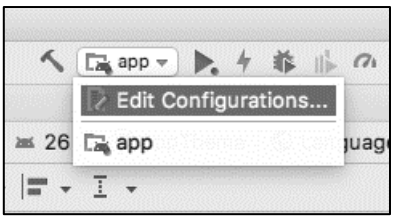

Рисунок 3.6 – Конфигурации запуска

Выбрать в списке строку Edit Configurations; открывается новое окно с подробной информацией о текущей конфигурации (рисунок 3.7).

Выбрать на левой панели строку app и убедиться в том, что в разделе Target Device установлен переключатель Show chooser dialog. Щёлкнуть на кнопке OK и запустить приложение заново. На этот раз будет предложено выбрать устройство для запуска.

Добавление ресурсов в проект

Приложение DriodQuest работает, но пользовательский интерфейс смотрелся бы более привлекательно, если бы на кнопке «Далее» была изображена стрелка, обращенная направо.

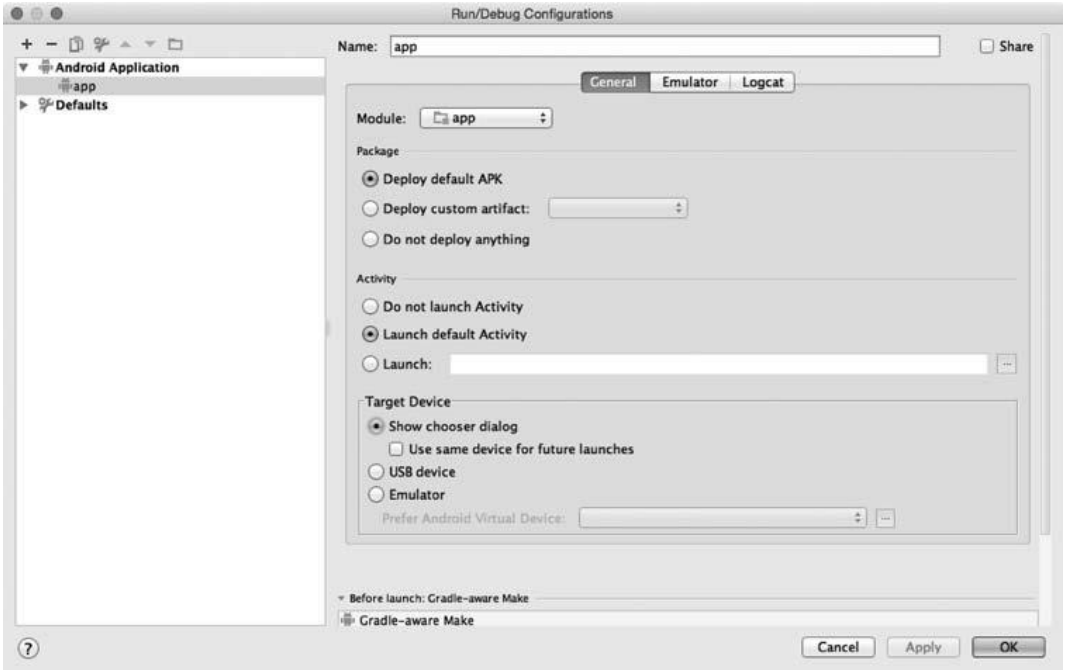

Рисунок 3.7 – Свойства конфигурации запуска

Изображение такой стрелки можно найти на сайте *https://material.io/icons/* (рисунок 3.8).

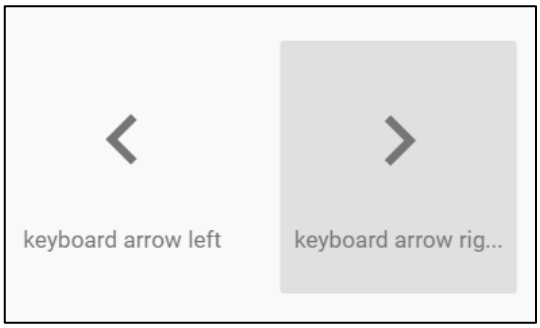

Рисунок 3.8 – Значки навигации

Загрузить файл и открыть каталог проекта /DriodQuest/app/src/main/res. Найти в нем подкаталоги drawable-hdpi, res/drawable-mdpi, drawable-xhdpi и drawable-xxhdpi.

Суффиксы имен каталогов обозначают экранную плотность пикселов устройства (таблица 3.1).

Таблица 3.1 – Обозначение экранной плотности пикселов устройства

mdpi	Средняя плотность (~160 dpi)
hdpi	Высокая плотность (~240 dpi)
xhdpi	Сверхвысокая плотность (~360 dpi)
xxhdpi	Сверхсверхвысокая плотность (~480 dpi)

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

В полноценных приложениях важно включить в проект графику для разных плотностей пикселов, поскольку это приводит к сокращению артефактов от масштабирования изображений. Вся графика в проекте устанавливается с приложением, а ОС выбирает наиболее подходящий вариант для конкретного устройства.

В решения DriodQuest включаются 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 DriodQuest — 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 необязательно 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.

Добавление значка

Следующим шагом станет включение графических файлов в ресурсы приложения DriodQuest.

Для начала следует убедиться в том, что проект содержит все необходимые папки drawable. Проверить, что в окне инструментов Project отображается представление Project (выбрать в раскрывающемся списке в верхней части окна инструментов Project строку *Project*. Раскрыть содержимое узла DriodQuest/app/src/main/res. В нем должны присутствовать папки с именами drawable-hdpi, drawable-mdpi, drawable-xhdpi и drawable-xxhdpi, изображенные на рисунке 3.9.

Если какие-либо из перечисленных папок drawable отсутствуют, создать их перед добавлением графических ресурсов. Щёлкнуть правой кнопкой мыши на каталоге res и выберите команду New→Directory. Ввести имя отсутствующего каталога (например, drawable-mdpi) и щёлкнуть на кнопке OK (рисунок 3.10).

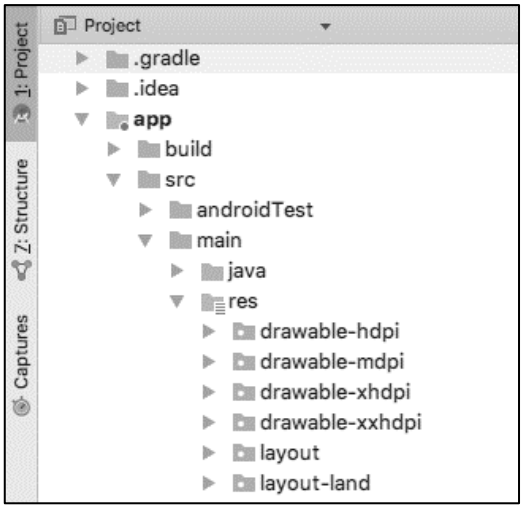

Рисунок 3.9 – Проверка существования каталогов drawable

Рисунок 3.10 – Создание каталога drawable

Созданный каталог drawable-mdpi должен появиться в представлении Project окна инструментов Project. (Если новый каталог не виден, вероятно,

работа осуществляется в режиме представления Android. Переключитесь в режим Project так, как описано выше.)

Повторить описанные действия для создания каталогов drawable-hdpi, drawable-xhdpi и drawable-xxhdpi.

Когда все каталоги будут созданы, для каждого каталога drawable в файле решений скопироапть файлы arrow_left.png и arrow_right.png и вставить их в соответствующий каталог drawable проекта.

После того как все изображения будут скопированы, файлы arrow_left.png и arrow_right.png появляются в окне инструментов Project (рисунок 3.11).

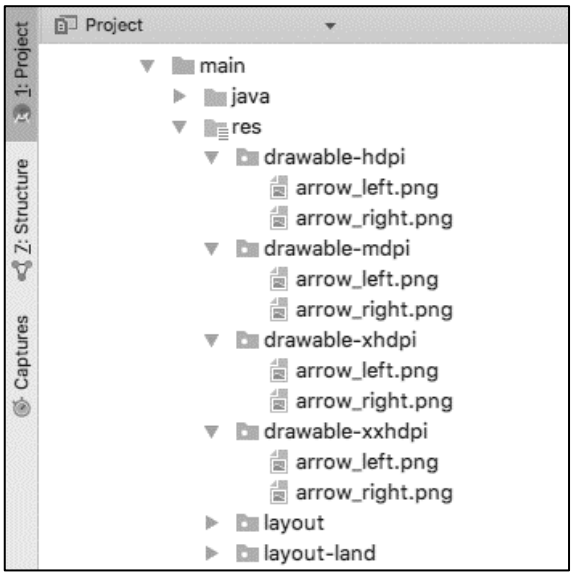

Рисунок 3.11 – Изображения стрелок в каталогах drawable проекта DriodQuest

Переключив окно инструментов Project обратно в режим Android, можно увидеть сводку добавленных файлов (рисунок 3.12).

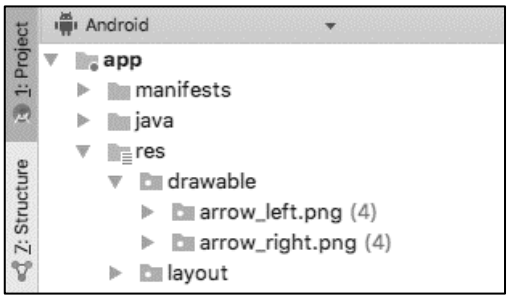

Рисунок 3.12 – Сводка изображений со стрелками в каталогах drawable приложения DriodQuest

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

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

Ссылки на ресурсы в XML

Для ссылок на ресурсы в коде используются идентификаторы ресурсов. Но в приложении необходимо настроить кнопку «Далее» так, чтобы в определении макета отображалась стрелка. Для включения ссылки на ресурс в разметку XML открыть файл activity_quest.xml и добавить два атрибута в определение виджета Button.

Листинг 3.12 – Включение графики в кнопку «Далее» (activity_quest.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 установлена ссылка на другой ресурс по его типу и имени. Ссылка на строку начинается с префикса @string/. Ссылка на графический ресурс начинается с префикса @drawable/.

Сохранить приложение DriodQuest и запустите его.

Однако в приложении DriodQuest скрывается ошибка. Во время выполнения DriodQuest нажать кнопку «Далее», чтобы перейти к следующему вопросу, а затем повернуть устройство. (Если программа выполняется в эмуляторе, нажат Ctrl+F12.)

Самостоятельные задания.

Задание 1. Добавление слушателя для TextView

Кнопка «Далее» удобна, но было бы неплохо сделать так, чтобы пользователь мог перейти 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.

Задание 2. Добавление кнопки возврата

Добавить кнопку для возвращения к предыдущему вопросу. Пользовательский интерфейс должен выглядеть примерно так, как показано на рисунке 3.13.

Задание 3. От Button к ImageButton

Возможно, пользовательский интерфейс будет смотреться еще лучше, если на кнопках будут отображаться *только* значки, как на рисунке 3.14.

Рисунок 3.13 – Теперь с кнопкой возврата!

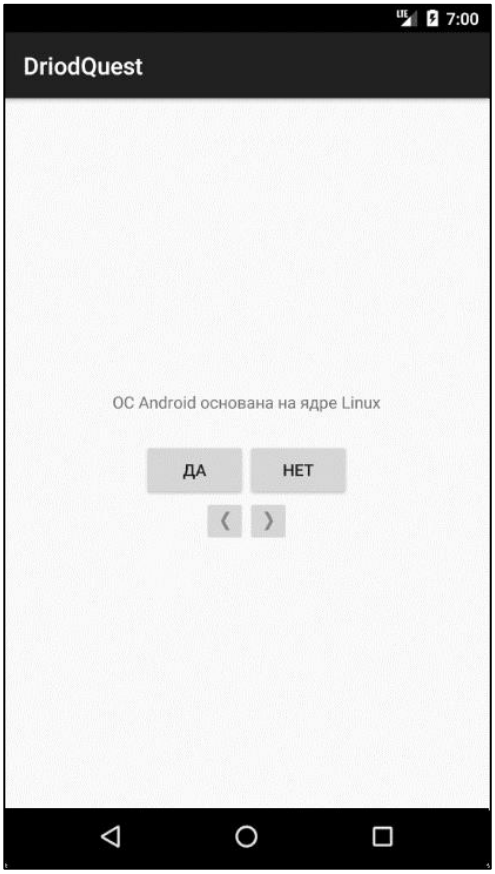

Рисунок 3.14 – Кнопки только со значками

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 рисунке 3.15.

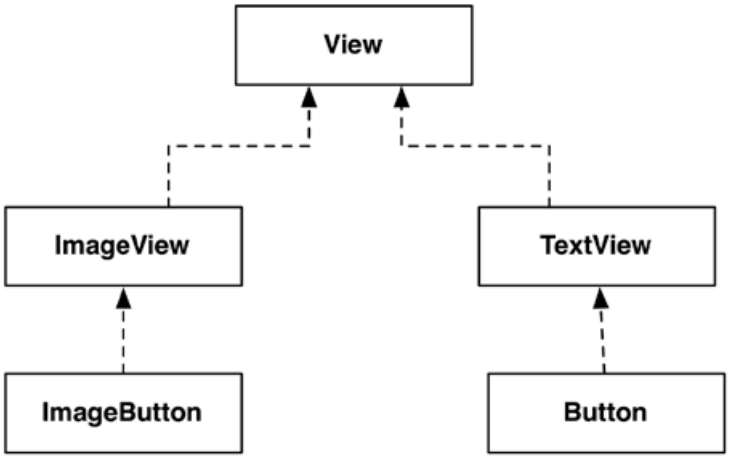

Рисунок 3.15 – Диаграмма наследования ImageButton и Button

Атрибуты text и drawable кнопки «Далее» 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_question_button"
    android:drawableRight="@drawable/arrow_right"
    android:drawablePadding="4dp"
    android:src="@drawable/arrow_right"
/>
```

Также необходимо внести изменения в QuestActivity, чтобы этот класс работал с ImageButton.

После того как будет выполнена замена кнопки кнопками 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).

Добавить атрибут `android:contentDescription` в каждый элемент ImageButton.

4. ЖИЗНЕННЫЙ ЦИКЛ АКТИВНОСТИ

В этом разделе будет рассмотрен жизненный цикл (ЖЦ) Activity, который имеется у каждого экземпляра. В процессе ЖЦ активность переходит между тремя возможными состояниями: выполнением, приостановкой и остановкой. Для каждого перехода у Activity существует метод, который оповещает активность об изменении состояния. На рисунке 4.1 изображен жизненный цикл активности, состояния и методы.

Рисунок 4.1 – Диаграмма состояний Activity

Субклассы Activity могут использовать методы, представленные на рисунке 4.1, для выполнения необходимых действий во время критических переходов жизненного цикла активности. Однако важно понимать, что методы жизненного цикла Activity никогда не вызываются в приложениях: они переопределяются в субклассах активности, а Android вызывает их в нужный момент времени.

Регистрация событий жизненного цикла Activity

Далее методы ЖЦ будут переопределены, чтобы отследить основные переходы жизненного цикла QuestActivity. Реализации ограничиваются регистрацией в журнале сообщения о вызове метода.

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 для этих целей: `public static int d(String tag, String msg);`

Имя «d» означает «debug» (отладка) и относится к уровню регистрации сообщений. Первый параметр определяет источник сообщения, а второй — его содержимое.

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

В файле QuestActivity.java добавить константу TAG в QuestActivity.

Листинг 4.1 – Добавление константы TAG (QuestActivity.java)

```
public class QuestActivity extends AppCompatActivity {
      private static final String TAG = "QuestActivity";
      ...
}
```

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

Листинг 4.2 – Включение команды регистрации сообщения в OnCreate(…) (QuestActivity.java)

```
public class QuestActivity extends AppCompatActivity {
      ...
      @Override
      public void onCreate(Bundle savedInstanceState) {
           super.onCreate(savedInstanceState);
           Log.d(TAG, "onCreate(Bundle) вызван");
           setContentView(R.layout.activity_quest);
           ...
      }
}
```

Теперь переопределить еще пять методов в QuestActivity; для этого добавить следующие определения после `onCreate(Bundle)`, но до `onCreateOptionsMenu(Menu)`:

Листинг 4.3 – Переопределение методов жизненного цикла (QuestActivity.java)

```
public class QuestActivity extends AppCompatActivity {
      ...
      @Override
      public void onStart() {
           super.onStart();
           Log.d(TAG, "onStart() вызван");
      }
```

```
    @Override
    public void onPause() {
        super.onPause();
        Log.d(TAG, "onPause() вызван");
    }
    @Override
    public void onResume() {
        super.onResume();
        Log.d(TAG, "onResume() вызван");
    }
    @Override
    public void onStop() {
        super.onStop();
        Log.d(TAG, "onStop() вызван");
    }
    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.d(TAG, "onDestroy() вызван");
    }
    ...
}
```

Следует обратить внимание на вызовы реализаций суперкласса перед регистрацией сообщений. Эти вызовы являются обязательными, причем в onCreate(…) реализация суперкласса должна вызываться ***до*** выполнения каких-либо критических операций; в других методах порядок менее важен.

Использование LogCat

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

При запуске DriodQuest данные LogCat появляются в нижней части окна Android Studio (рисунок 4.2). Если панель LogCat не видна, выбрать панель Android в нижней части окна и перейдите на вкладку Devices|logcat.

Запустить DriodQuest; на панели LogCat начнут быстро появляться сообщения. По умолчанию выводятся журнальные сообщения, сгенерированные с именем пакета приложения. В LogCat появятся сообщения программы, а также некоторые системные сообщения.

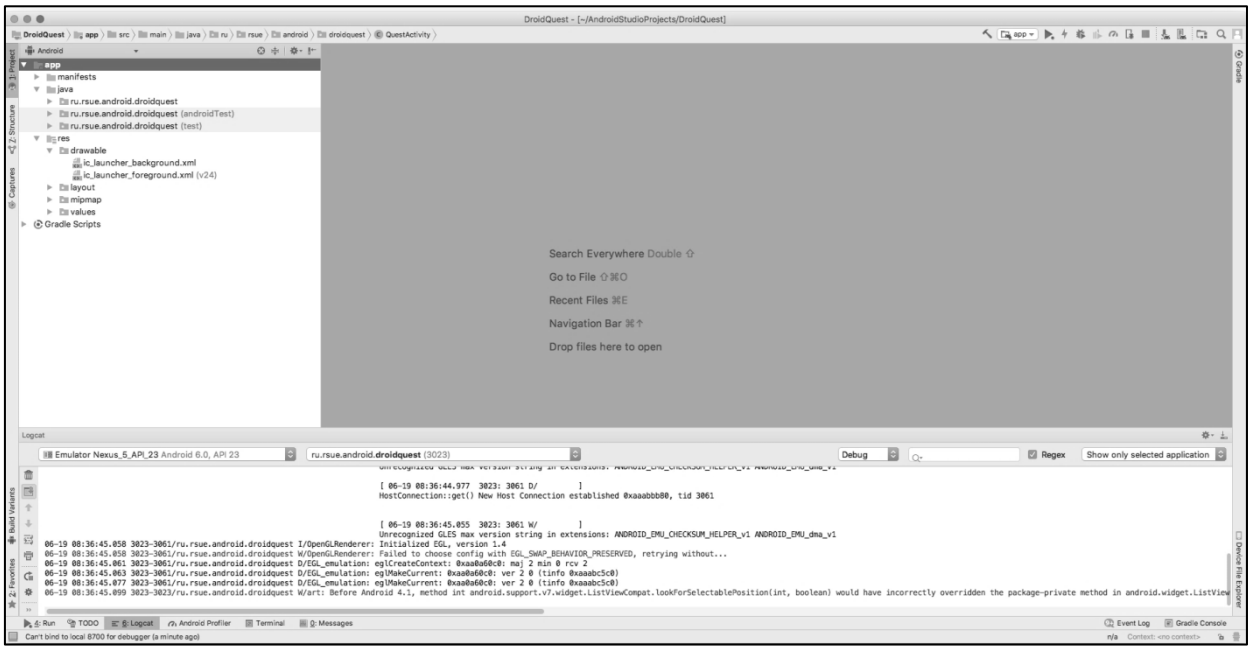

Рисунок 4.2 – Android Studio с выводом LogCat

Чтобы упростить поиск сообщений, можно отфильтровать вывод по константе TAG. В LogCat щёлкнуть на раскрывающемся списке фильтра в правом верхнем углу панели. Обратить внимание на существующий фильтр, настроенный на вывод сообщений только от данного приложения. Если выбрать вариант No Filters, будут выводиться журнальные сообщения, сгенерированные в масштабах всей системы.

Выбрать в раскрывающемся списке пункт Edit Filter Configuration. Нажать кнопку + для создания нового фильтра. Ввести имя фильтра QuestActivity и ввести строку QuestActivity в поле by Log Tag: (рисунок 4.3).

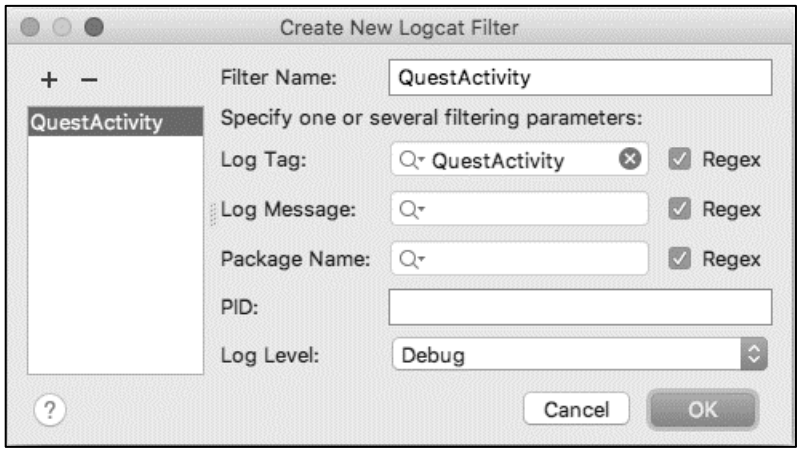

Рисунок 4.3 – Создание фильтра в LogCat

Щёлкнуть на кнопке OK; после этого будут видны только сообщения с тегом QuestActivity (рисунок 4.4). Как видно, после запуска DriodQuest были вызваны три метода жизненного цикла и был создан исходный экземпляр QuestActivity.

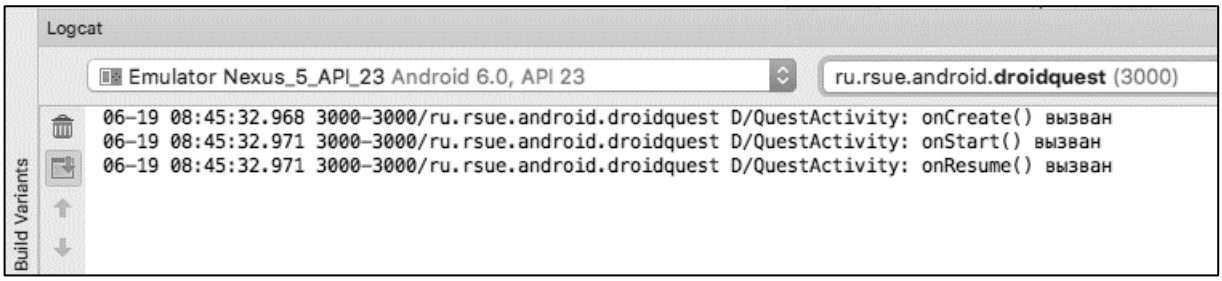

Рисунок 4.4 – При запуске DriodQuest происходит создание, запуск и продолжение активности

Эксперемент. Необходимо нажать на устройстве кнопку Back, а затем проверить вывод LogCat. Активность приложения получила вызовы onPause(), onStop() и onDestroy() (рисунок 4.5).

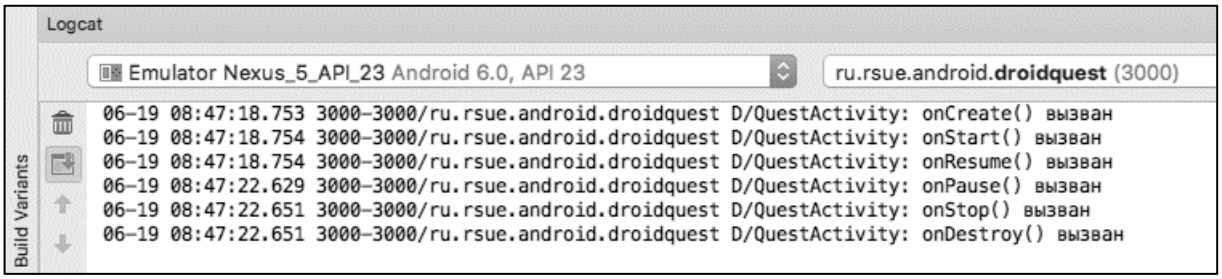

Рисунок 4.5 – Нажатие кнопки Back приводит к уничтожению активности

Нажимая кнопку Back, пользователь сообщает Android: «Я завершил работу с активностью, и она мне больше не нужна». Android уничтожает активность, чтобы избежать неэффективного расходования ограниченных ресурсов устройства.

Перезапустить приложение DriodQuest. Нажмите кнопку Home и проверьте вывод LogCat. Активность получила вызовы onPause() и onStop(), но не вызов onDestroy() (рисунок 4.6).

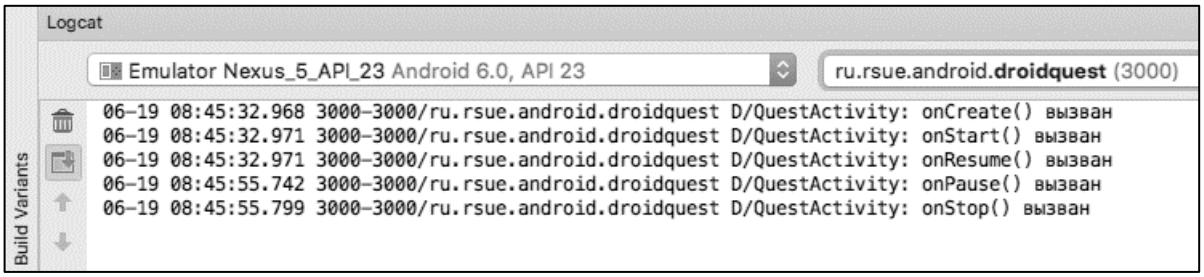

Рисунок 4.6 – Нажатие кнопки Back приводит к уничтожению активности

Вызвать на устройстве диспетчер задач. На новых устройствах для этого следует нажать кнопку Recents рядом с кнопкой Home. На устройствах без кнопки Recents выполнить долгое нажатие кнопки Home.

В диспетчере задач нажать на приложении DriodQuest и проверить вывод LogCat. Активность запускается и продолжает работу, но создавать ее не нужно.

Нажатие кнопки Home сообщает Android: «Я сейчас займусь другим делом, но потом могу вернуться». Android приостанавливает активность, но старается не уничтожать ее на случай возвращения.

Тем не менее существование остановленной активности не гарантировано. Если системе потребуется занятая память, она уничтожает остановленные активности.

Повороты и жизненный цикл активности

Следует вернуться к ошибке приложения, которая была обнаружена в предыдущем разделе. Запустить DriodQuest, нажать кнопку «Далее» для перехода к следующему вопросу, а затем повернуть устройство. (Чтобы имитировать поворот в эмуляторе, следует нажать Ctrl+F12.)

После поворота DriodQuest снова выводит первый вопрос. Чтобы понять, почему это произошло, необходимо просмотреть вывод LogCat. Он выглядит примерно так, как показано на рисунке 4.7.

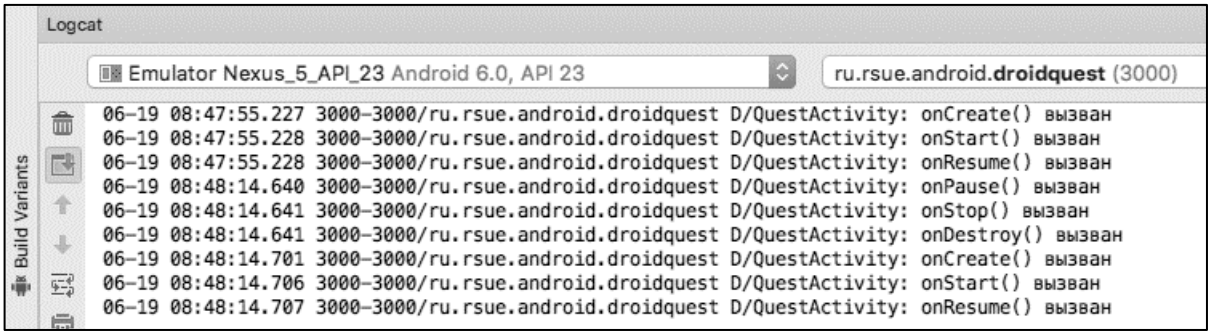

Рисунок 4.7 – QuestActivity умирает и возрождается

Когда пользователь поворачивает устройство, экземпляр QuestActivity уничтожается, и вместо него создается новый экземпляр. Снова поверните устройство — происходит еще один цикл уничтожения и возрождения. Из-за этого и возникает ошибка. При каждом создании нового экземпляра QuestActivity переменная mCurrentIndex инициализируется 0, а пользователь начинает с первого вопроса.

Создание макета для альбомной ориентации

В окне инструментов Project щёлкнуть правой кнопкой мыши на каталоге res и выбрать команду New→Android resource directory. Открывается окно со списками типов ресурсов и квалификаторами этих типов (наподобие изображенного на рисунке 4.8). Выбрать в раскрывающемся списке Resource type строку layout. Оставить в списке Source set значение main.

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

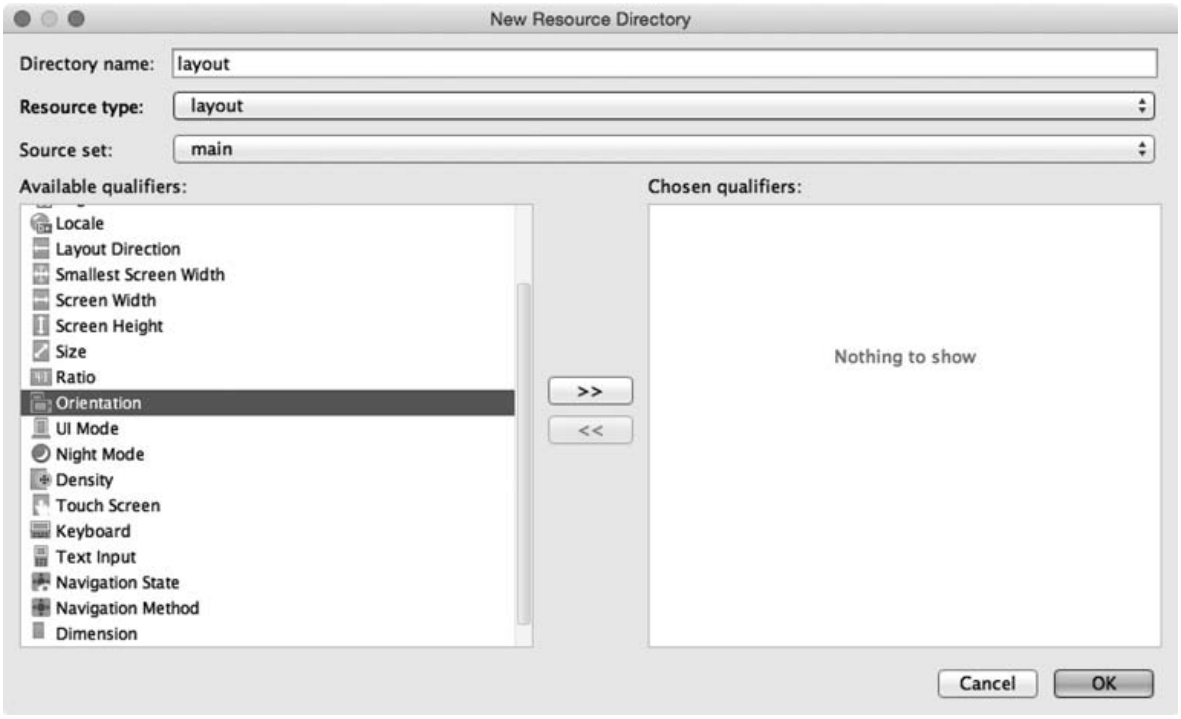

Рисунок 4.8 – Создание нового каталога ресурсов

Наконец, надо убедиться в том, что в списке Screen Orientation выбрано значение Landscape (рисунок 4.9).

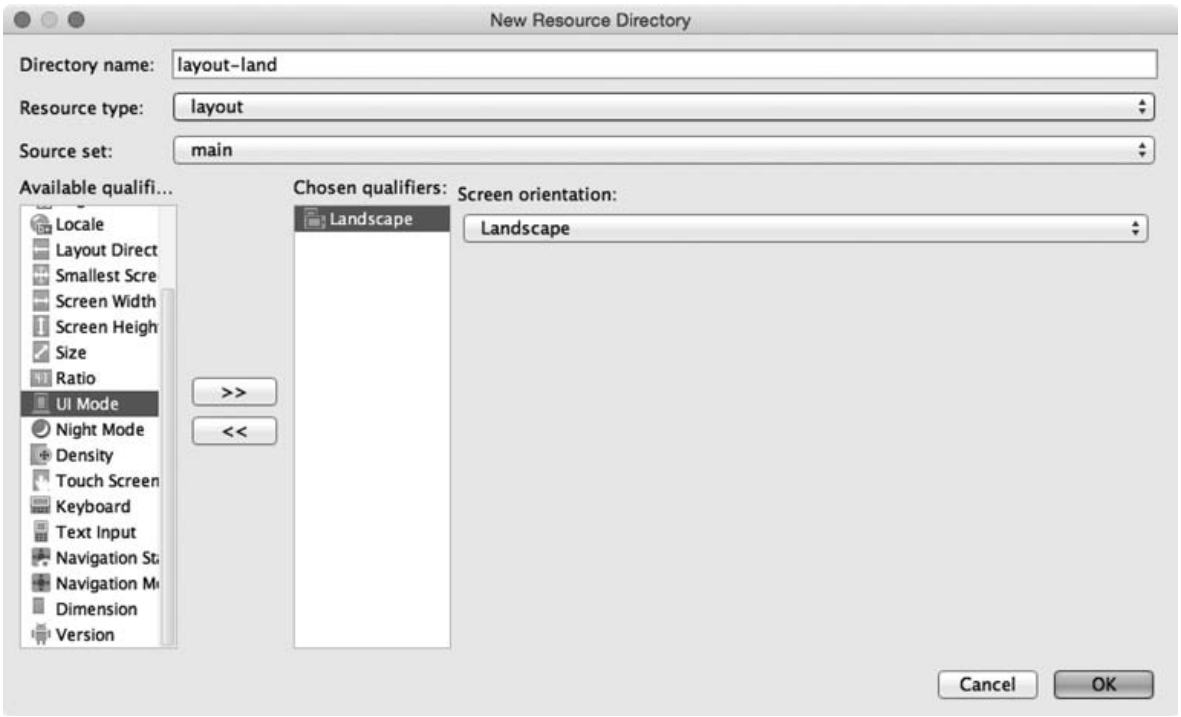

Рисунок 4.9 – Создание каталога res/layout-land/

Проверить, что в поле Directory name теперь указано имя каталога layout-land. Хотя окно выглядит нетривиально, его цель — задать имя каталога. Щёлкнуть на кнопке OK; Android Studio создает папку 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 конфигурационных квалификаторов, поддерживаемых Android, и обозначаемых ими компонентов конфигурации устройств находится по адресу *http://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/.

Скопировать файл activity_quest.xml из res/layout/ в res/layout-land/. Теперь в приложении имеются два макета: макет для альбомной ориентации и макет по умолчанию. Оставить имя файла без изменения. Два файла макетов должны иметь одинаковые имена, чтобы на них можно было ссылаться по одному идентификатору ресурса.

Теперь вносятся некоторые изменения в альбомный макет, чтобы он отличался от макета по умолчанию. Сводка этих изменений представлена на рисунке 4.10.

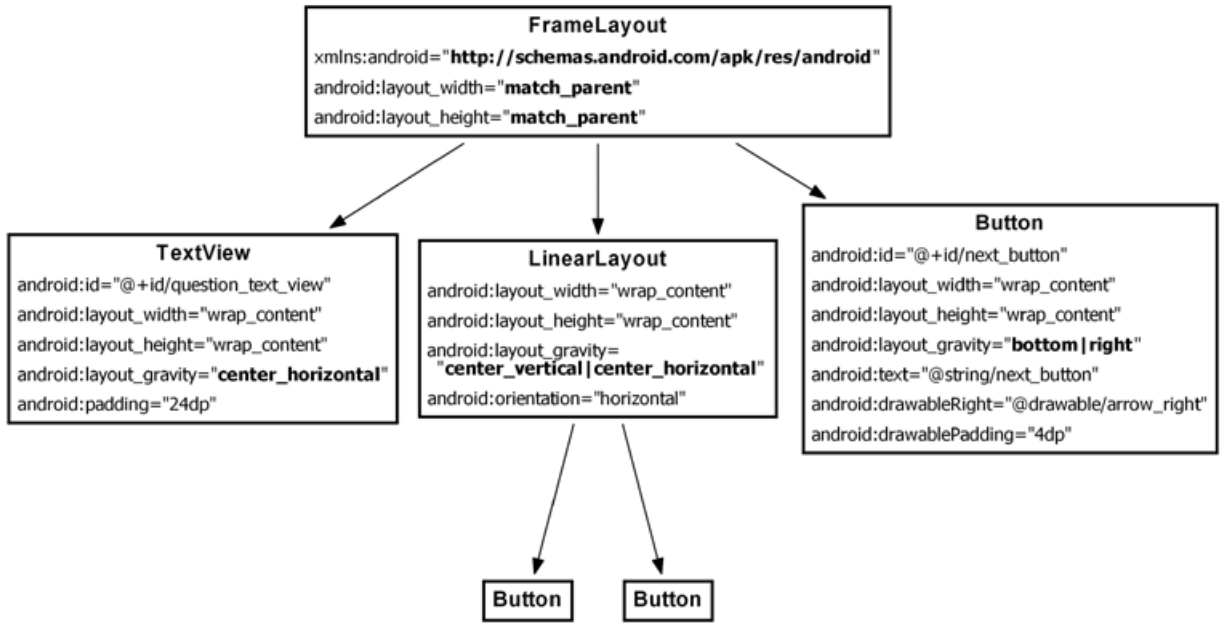

Рисунок 4.10 – Альтернативный макет для альбомной ориентации

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.

Открыть файл layout-land/activity_quest.xml и внести необходимые изменения, руководствуясь рисунком 4.10 – Проверить результат своей работы по листингу 4.4.

Листинг 4.4 – Настройка альбомного макета (layout-land/activity_quest.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 запустить DriodQuest. Повернуть устройство в альбомную ориентацию, чтобы увидеть новый макет (рисунок 4.11). Конечно, в программе используется не только новый макет, но и новый экземпляр QuestActivity.

Рисунок 4.11 – QuestActivity в альбомной ориентации

Повернуть устройство, чтобы переключиться в книжную ориентацию. Отобразится новый макет — и новый экземпляр QuestActivity.

Android выбирает наиболее подходящий ресурс, но для этого он создает новую активность с нуля. Чтобы класс QuestActivity вывел новый макет, необходимо снова вызвать метод `setContentView(R.layout.activity_quest)`. А это не произойдет без повторного вызова `QuestActivity.onCreate(…)`. Соответственно Android при повороте уничтожает текущий экземпляр QuestActivity и начинает все заново, чтобы обеспечить оптимальный подбор ресурсов для новой конфигурации.

Android уничтожает текущую активность и создает новую при каждом изменении конфигурации времени выполнения. Также в процессе выполнения могут происходить и другие изменения (например, изменение языка или доступности клавиатуры), но изменение в ориентации экрана является самым частым.

Сохранение данных между поворотами

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

Чтобы исправить эту ошибку, экземпляр QuestActivity, созданный после поворота, должен знать старое значение mCurrentIndex. 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);
```

Обычно этот метод вызывается системой перед `onPause()`, `onStop()` и `onDestroy()`.

Реализация по умолчанию onSaveInstanceState(…) приказывает всем представлениям активности сохранить свое состояние в данных объекта Bundle — структуры, связывающей строковые ключи со значениями некоторых ограниченных типов.

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

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

При переопределении onCreate(…) вызывается реализации onCreate(…) суперкласса активности и ей передается только что полученный объект Bundle. В реализации суперкласса сохраненное состояние представлений извлекается и используется для воссоздания иерархии представлений активности.

Переопределение onSaveInstanceState(Bundle)

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

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

Листинг 4.5 – Добавление ключа для сохраняемого значения (QuestActivity.java)

```
public class QuestActivity extends AppCompatActivity {
private static final String TAG = "QuestActivity";
private static final String KEY_INDEX = "index";
Button mTrueButton;
...
```

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

Листинг 4.6 – Переопределение onSaveInstanceState(…) (QuestActivity.java)

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

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

Наконец, в методе `onCreate(…)` следует проверить это значение, и если оно присутствует, присвоить его mCurrentIndex.

Листинг 4.7 – Проверка сохраненных данных в onCreate(…) (QuestActivity.java)

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

Запустить DriodQuest и нажмать кнопку «Далее». При каждом выполненном повороте устройства вновь созданный экземпляр QuestActivity «вспоминает» текущий вопрос.

Для сохранения и восстановления из Bundle подходят примитивные типы и объекты, 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.

Методы и уровни регистрации

Когда используется класс `android.util.Log` для регистрации сообщений в журнале, известно не только содержимое сообщения, но и *уровень регистрации*, определяющий важность сообщения. Android поддерживает пять уровней регистрации (таблица 4.1). Каждому уровню соответствует свой метод класса Log. Регистрация данных в журнале сводится к вызову соответствующего метода Log.

Таблица 4.1 – 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 приложением. В листинге 4.8 представлены примеры сигнатуры методов журнала. Для сборки строк сообщений используются стандартные средства конкатенации строк Java — или String.format, если их окажется недостаточно.

Листинг 4.8 – 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);
}
```

Самостоятельные задания.

Задание. Добавление вопросов

Самостоятельно необходимо добавить еще пять вопросов по ОС Android, которые будут предлагаться пользователю.

5. ДОБАВЛЕНИЕ ВТОРОЙ АКТИВНОСТИ

В данном разделе в приложение DriodQuest будет добавлена вторая активность. 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 QuestActivity и отвечает на вопрос, он получает новое сообщение.

Рисунок 5.1 – DeceitActivity позволяет подсмотреть ответ на вопрос

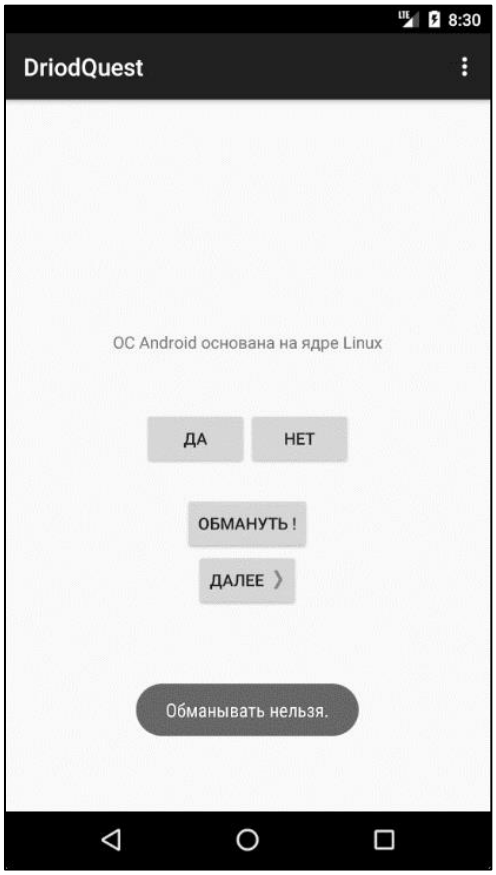

Рисунок 5.2 – QuestActivity выводит новое сообщение

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, то Android-приложение будет протестовать. Чтобы избежать возможных проблем, следует воспользоваться мастером New Activity среды Android Studio.

Запустить мастер New Activity: щёлкнуть правой кнопкой мыши на пакете ru.rsue.android.driodquest в окне инструментов Project и выбрать команду New→ Activity→Blank Activity (Empty Activity).

На экране появляется диалоговое окно, изображенное на рисунке 5.3 – Ввести в поле Activity Name строку DeceitActivity — это имя субкласса Activity. В поле Layout Name автоматически генерируется имя activity_deceit. Оно становится базовым именем файла макета, создаваемого мастером. Поле Title также автоматически заполняется текстом DeceitActivity, но поскольку этот текст будет виден пользователю, оставить в поле только «Deceit».

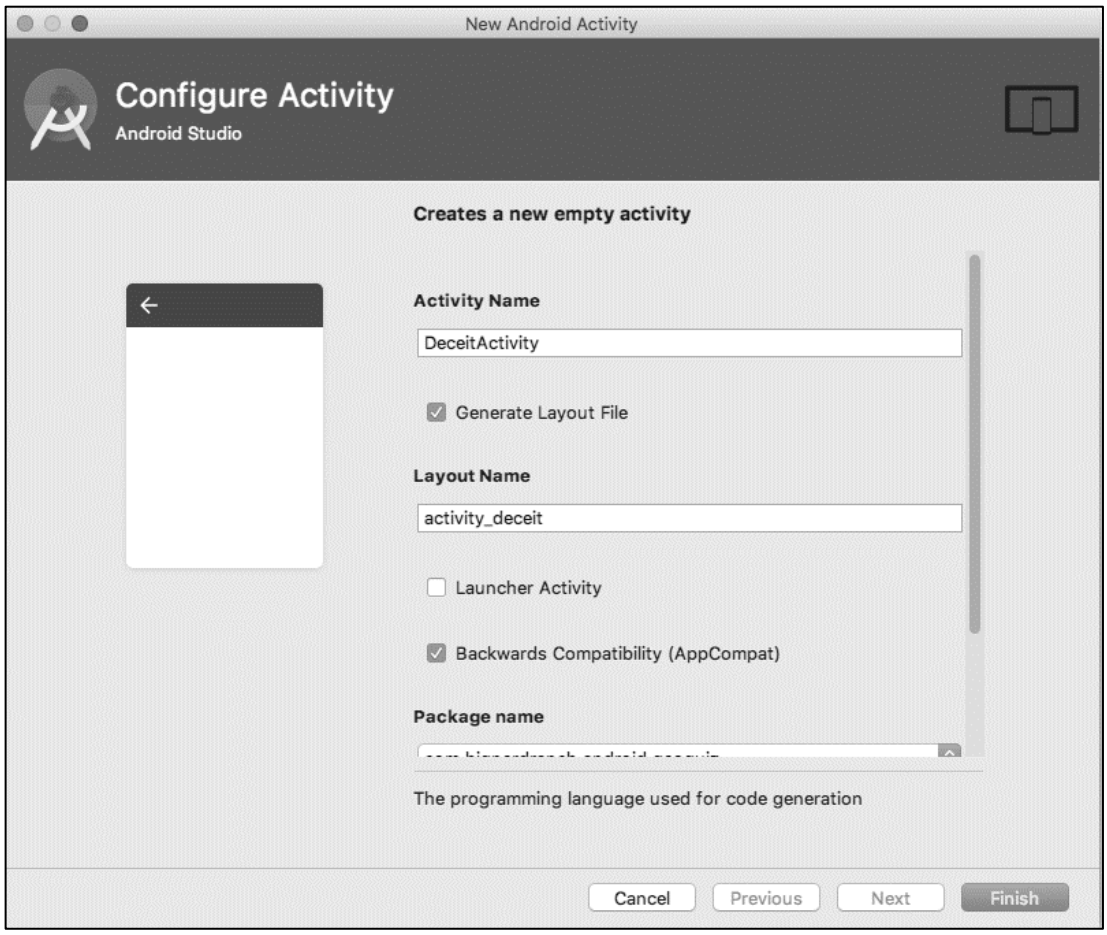

Рисунок 5.3 – Мастер New Blank Activity

Остальным полям можно оставить значения по умолчанию, но надо убедиться в том, что имя пакета выглядит так, как предполагается. Оно определяет местонахождение DeceitActivity.java в файловой системе. Щёлкнуть на кнопке Finish, чтобы мастер завершил свою работу.

Теперь можно обратиться к пользовательскому интерфейсу. Снимок экрана в начале работы показывает, как должна выглядеть активность DeceitActivity. На рисунке 5.4 приведены определения виджетов.

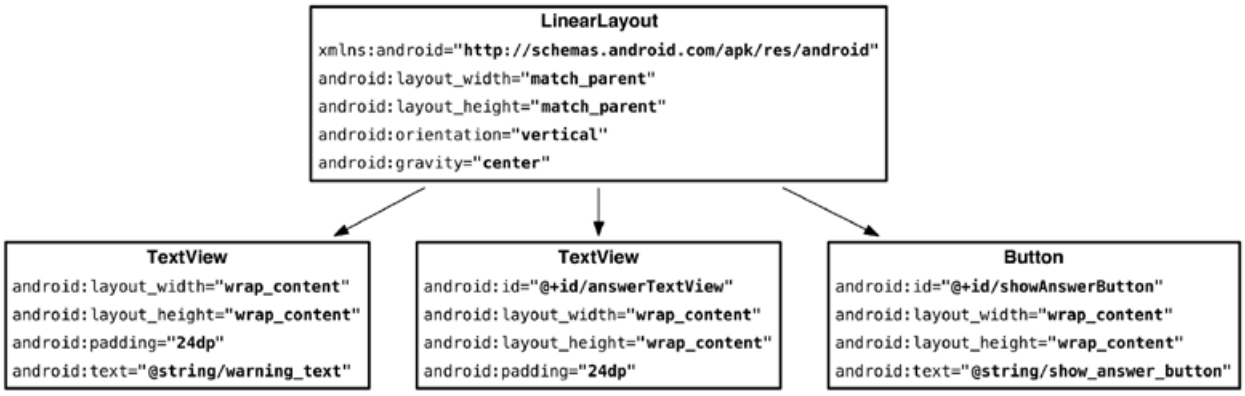

Рисунок 5.4 – Схема макета DeceitActivity

После завершения мастера должен открыться файл activity_deceit.xml из каталога layout. Если этого не произошло, надо открыть файл самостоятельно и перейти в режим представления Text (XML).

Создать разметку XML для макета по образцу рисунка 5.4.

Листинг 5.1 – Заполнение макета второй активности (activity_deceit.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:gravity="center"
      android:orientation="vertical"
      tools:context="ru.rsue.android.driodquest.DeceitActivity">
      <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="Ответ"/>
      <Button
           android:id="@+id/show_answer_button"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:text="@string/show_answer_button"/>
</LinearLayout>
```

Следует обратить 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, которому можно присвоить фиктивное литеральное значение, чтобы знать, как виджет будет выглядеть во время выполнения. Значение «Ответ» никогда не появится в реальном приложении.

Далее открыть файл strings.xml и добавить необходимые строки.

Листинг 5.2 – Добавление строк (strings.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    ...
    <string name="question_manifest">AndroidManifest.xml предоставляет
      системе основную информацию о программе.</string>
    <string name="warning_text">Вы уверены, что хотите это
      сделать?</string>
    <string name="show_answer_button">Показать ответ</string>
    <string name="deceit_button">Обмануть !</string>
    <string name="judgment_toast">Обманывать нельзя.</string>
</resources>
```

Не надо создавать для activity_deceit.xml альтернативный макет с альбомной ориентацией, однако существует возможность увидеть, как макет по умолчанию будет отображаться в альбомном режиме.

В окне предварительного просмотра найти на панели инструментов над панелью предварительного просмотра кнопку, на которой изображено устройство с изогнутой стрелкой. Щёлкнуть на этой кнопке, чтобы изменить ориентацию макета (рисунок 5.5).

Рисунок 5.5 – Просмотр макета activity_deceit.xml в альбомной ориентации

Макет по умолчанию достаточно хорошо смотрится в обеих ориентациях.

Объявление активностей в манифесте

Манифест (manifest) представляет собой файл XML с метаданными, описывающими разрабатываемое приложение для ОС Android. Файл

манифеста всегда называется AndroidManifest.xml и располагается в каталоге app/manifests проекта.

В окне инструментов Project найти и открыть AndroidManifest.xml. Также можно воспользоваться диалоговым окном Android Studio Quick Open — нажать Command+Shift+O (Ctrl+Shift+N) и начать вводить имя файла. Когда в окне будет предложен правильный файл, необходимо нажать Return (Enter), чтобы открыть этот файл.

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

Когда использовался мастер новых приложений для создания QuestActivity, мастер объявил активность сомостоятельно. Аналогичным образом мастер New Activity объявил DeceitActivity, добавив разметку XML, выделенную в листинге 5.3.

Листинг 5.3 – Объявление DeceitActivity в манифесте (AndroidManifest.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="ru.rsue.android.driodquest"
        <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".QuestActivity"
            android:label="@string/app_name" >
            ...
        </activity>
        <activity
            android:name=".DeceitActivity"
            android:label="@string/title_activity_deceit" >
        </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.

Добавление кнопки Deceit в QuestActivity

Итак, пользователь должен нажать кнопку в QuestActivity, чтобы вызвать на экран экземпляр DeceitActivity. Следовательно, необходимо

включить новые кнопки в layout/activity_quest.xml и layout-land/activity_quest.xml.

В макете по умолчанию добавить новую кнопку как прямого потомка корневого элемента LinearLayout. Ее определение должно непосредственно предшествовать кнопке «Далее».

Листинг 5.4 – Добавление кнопки Deceit! в макет по умолчанию (layout/activity_quest.xml)

```
...
</LinearLayout>
    <Button
        android:id="@+id/deceit_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/deceit_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 Deceit! в альбомный макет (layout-land/activity_quest.xml)

```
    ...
    </LinearLayout>
    <Button
        android:id="@+id/deceit_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|center"
        android:text="@string/deceit_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>
```

Сохранить файлы макетов и открыть QuestActivity.java. Добавить переменную, получить ссылку и назначить заглушку `View.onClickListener` для кнопки Deceit!.

Листинг 5.6 – Подключение кнопки Deceit! (QuestActivity.java)

```
public class QuestActivity extends AppCompatActivity {
    ...
    private Button mNextButton;
    private Button mDeceitButton;
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        mDeceitButton = (Button)findViewById(R.id.deceit_button);
        mDeceitButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Запуск DeceitActivity
            }
        });
        if (savedInstanceState != null) {
            mCurrentIndex = savedInstanceState.getInt(KEY_INDEX, 0);
        }
        updateQuestion();
    }
    ...
}
```

Теперь можно переходить к запуску DeceitActivity.

Запуск активности

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

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

В слушателе mDeceitButton создать объект Intent, включающий класс DeceitActivity, а затем передать его при вызове startActivity(Intent) (листинг 5.7).

Листинг 5.7 – Запуск DeceitActivity (QuestActivity.java)

```
...
mDeceitButton = (Button)findViewById(R.id.deceit_button);
mDeceitButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Запуск DeceitActivity
        Intent i = new Intent(QuestActivity.this, DeceitActivity.class);
        startActivity(i);
```

```
    }
});
```

Запустить приложение DriodQuest. Нажать кнопку *«Обмануть!»*; на экране появляется новая активность. Теперь следует нажать кнопку Back. Активность DeceitActivity уничтожается, а приложение возвращается к QuestActivity.

Передача данных между активностями

Чтобы сообщить DeceitActivity ответ на текущий вопрос, будет передано значение

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

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

Для включения дополнений в интент используется метод:

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

Метод `Intent.putExtra(…)` 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.

Добавить в DeceitActivity.java ключ для дополнения.

Листинг 5.8 – Добавление константы (DeceitActivity.java)

```
public class DeceitActivity extends AppCompatActivity {
    public static final String EXTRA_ANSWER_IS_TRUE =
            "ru.rsue.android.driodquest.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 включить дополнение в интент. Эту работу правильнее инкапсулировать в методе `newIntent(…)`.

Создайте этот метод в DeceitActivity:

Листинг 5.9 – Метод newIntent(...) (DeceitActivity.java)

```
public class DeceitActivity extends AppCompatActivity {
  private static final String EXTRA_ANSWER_IS_TRUE =
      "ru.rsue.android.driodquest.answer_is_true";
  public static Intent newIntent(Context packageContext, boolean
  answerIsTrue) {
      Intent i = new Intent(packageContext, DeceitActivity.class);
      i.putExtra(EXTRA_ANSWER_IS_TRUE, answerIsTrue);
      return i;
```

```
  }
...
```

Этот статический метод позволяет создать объект Intent, настроенный дополнениями, необходимыми для DeceitActivity. Логический аргумент answerIsTrue помещается в интент под закрытым именем с использованием константы EXTRA_ANSWER_IS_TRUE.

Используется этот метод в слушателе кнопки Deceit!:

Листинг 5.10 – Запуск DeceitActivity.java с дополнением (QuestActivity.java)

```
...
mDeceitButton.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
         // Запуск DeceitActivity
         Intent i = new Intent(QuestActivity.this,
               DeceitActivity.class);
         boolean answerIsTrue = mQuestionBank[mCurrentIndex]
         .isAnswerTrue();
         Intent i = DeceitActivity.newIntent(QuestActivity.this,
         answerIsTrue);
         startActivity(i);
      }
   });
   updateQuestion();
}
```

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 DeceitActivity прочитать значение из дополнения в `onCreate(Bundle)` и сохранить его в переменной.

Листинг 5.11 – Использование дополнения (DeceitActivity.java)

```
public class DeceitActivity extends AppCompatActivity {
   ...
   private boolean mAnswerIsTrue;
   ...
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_deceit);
      mAnswerIsTrue = getIntent().getBooleanExtra(EXTRA_ANSWER_IS_TRUE,
          false);
```

```
    }
    ...
}
```

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

Наконец, добавить в DeceitActivity код, обеспечивающий использование прочитанного значения в виджете TextView ответа и кнопке Show Answer.

Листинг 5.12 – Добавление функциональности подсматривания ответов (DeceitActivity.java)

```
public class DeceitActivity extends AppCompatActivity {
  ...
  private boolean mAnswerIsTrue;
  private TextView mAnswerTextView;
  private Button mShowAnswer;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
      ...
      mAnswerIsTrue = getIntent().getBooleanExtra(EXTRA_ANSWER_IS_TRUE,
      false);
      mAnswerTextView = (TextView)findViewById(R.id.answer_text_view);
      mShowAnswer = (Button)findViewById(R.id.show_answer_button);
      mShowAnswer.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);
            }
          }
      });
  }
}
```

В программном коде происходит следующее: задается текст TextView при помощи метода `TextView.setText(int)`. Метод `TextView.setText(…)` существует в нескольких вариантах; здесь используется вариант, получающий идентификатор строкового ресурса.

Запустить приложение DriodQuest. Нажать кнопку Deceit!, чтобы перейти к DeceitActivity. Затем нажать кнопку Show Answer, чтобы открыть ответ на текущий вопрос.

Получение результата от дочерней активности

В текущей версии приложения пользователь может беспрепятственно жульничать. Исправить этот недостаток; для этого DeceitActivity надо сообщать QuestActivity, подсмотрел ли пользователь ответ.

Чтобы получить информацию от дочерней активности, внадо вызвать 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.

В классе QuestActivity изменить слушателя mDeceitButton и включить в него вызов `startActivityForResult(Intent, int)`.

Листинг 5.13 – Вызов startActivityForResult(…) (QuestActivity.java)

```
public class QuestActivity extends AppCompatActivity {
    ...
    private static final int REQUEST_CODE_DECEIT = 0;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        mDeceitButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            ...
            Intent i = DeceitActivity.newIntent(QuestActivity.this,
            answerIsTrue);
            ~~startActivity(i);~~
            startActivityForResult(i, REQUEST_CODE_DECEIT);
        }
    });
...
}
```

Передача результата

Существуют 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.

Возвращение интента

В данной реализации дочерняя активность должна вернуть QuestActivity некоторые данные. Соответственно будет создан объект Intent,

в который будет помещено дополнение, а затем будет вызван Activity.setResult(int, Intent) для передачи этих данных QuestActivity.

Добавить в DeceitActivity константу для ключа дополнения и закрытый метод, который выполняет необходимую работу. Затем включить вызов этого метода в слушателя кнопки Show Answer.

Листинг 5.14 – Назначение результата (DeceitActivity.java)

```
public class DeceitActivity extends AppCompatActivity {
    public static final String EXTRA_ANSWER_IS_TRUE =
    "ru.rsue.android.driodquest.answer_is_true";
    public static final String EXTRA_ANSWER_SHOWN =
    "ru.rsue.android.driodquest.answer_shown";
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        mShowAnswer.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, DeceitActivity упаковывает код результата и интент в вызове `setResult(int, Intent)`.

Затем, когда пользователь нажимает кнопку Back для возвращения к QuestActivity, ActivityManager вызывает следующий метод родительской активности:

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

В параметрах передается исходный код запроса от QuestActivity, код результата и интент, переданный `setResult(…)`.

Остается последний шаг: переопределение `onActivityResult(int, int, Intent)` в QuestActivity для обработки результата. Но поскольку содержимое интента результата также относится к подробностям реализации DeceitActivity, добавить еще один метод для упрощения декодирования дополнения к виду, который может использоваться QuestActivity.

Листинг 5.15 – Декодирование интента результата (DeceitActivity.java)

```
public static Intent newIntent(Context packageContext,
      boolean answerIsTrue) {
    ...
}
public static boolean wasAnswerShown(Intent result) {
    return result.getBooleanExtra(EXTRA_ANSWER_SHOWN, false);
}
@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
}
```

Обработка результата

Добавить в QuestActivity.java новую переменную для хранения значения, возвращаемого DeceitActivity. Затем включить в переопределение onActivityResult(…) код его получения, проверки кода запроса и кода результата, чтобы быть уверенным в том, что они соответствуют ожиданиям.

Листинг 5.16 – Реализация onActivityResult(…) (QuestActivity.java)

```
public class QuestActivity extends AppCompatActivity {
    ...
    private int mCurrentIndex = 0;
    private boolean mIsDeceiter;
    ...
    @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_DECEIT) {
            if (data == null) {
                return;
            }
```

```
        mIsDeceiter = DeceitActivity.wasAnswerShown(data);
        }
    }
    ...
```

Изменить метод checkAnswer(boolean) в QuestActivity. Он должен проверять, подсмотрел ли пользователь ответ, и реагировать соответствующим образом.

Листинг 5.17 – Изменение уведомления в зависимости от значения mIsDeceiter (QuestActivity.java)

```
private void checkAnswer(boolean userPressedTrue) {
    boolean answerIsTrue = mQuestionBank[mCurrentIndex].isAnswerTrue();
    int messageResId = 0;
    if (mIsDeceiter) {
        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();
}
@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;
            mIsDeceiter = false;
            updateQuestion();
        }
    });
    ...
}
```

Запустить приложение DriodQuest и попробовать подсмотреть ответ.

Самостоятельные задания.

Задание. Устранение логических ошибок

Приложение DriodQuest содержит ряд «лазеек», которыми можно воспользоваться. В этом задании необходимо устранить эти недостатки. Основные дефекты приложения перечислены ниже по возрастанию сложности, от самых простых к самым сложным:

– подсмотрев ответ, пользователь может повернуть DeceitActivity, чтобы сбросить результат;

– после возвращения пользователь может повернуть QuestActivity, чтобы сбросить флаг mIsDeceiter;

– пользователь может нажимать кнопку «Далее» до тех пор, пока вопрос, ответ на который был подсмотрен, снова не появится на экране.

6. UI-ФРАГМЕНТЫ

В данном разделе начинается разработка приложения BookDepository. Оно предназначено для учета прочтенных книг. В приложении BookDepository пользователь создает запись о книге, которую планирует прочитать, с заголовком, датой и фотографией обложки. Также можно отправить отзыв о книге по электронной почте, опубликовать его в Twitter, Facebook или другом приложении. Сообщая о прочтенных книгах, пользователь мотивирует других больше читать.

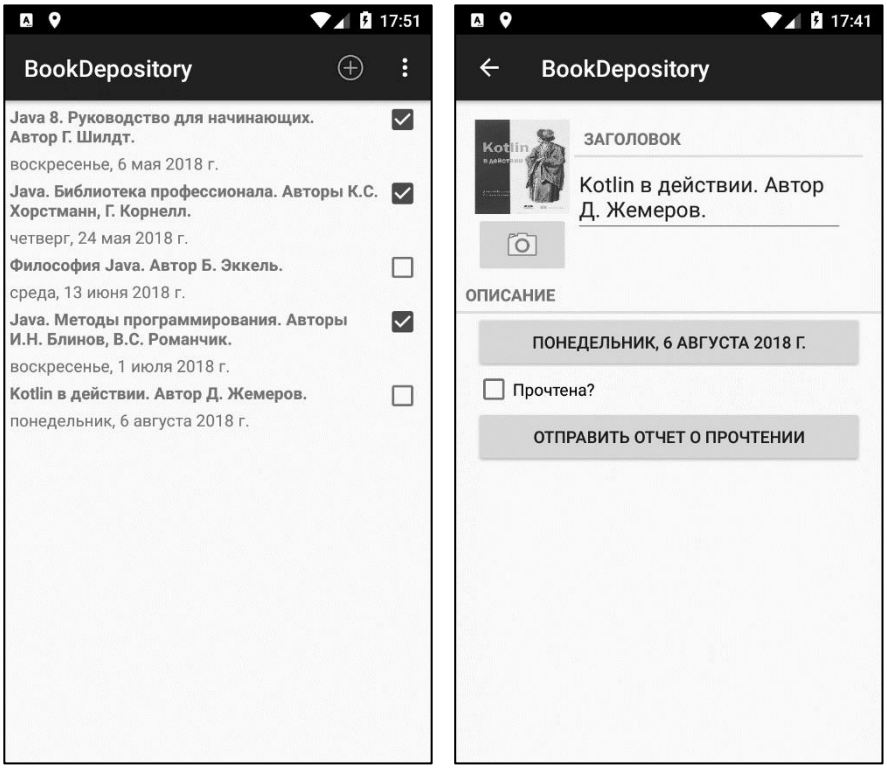

Рисунок 6.1 – Приложение BookDepository

BookDepository — сложное приложение, в нем используется интерфейс типа «список/детализация»: на главном экране выводится список желаемых к прочтению книг. Пользователь может добавить новую или выбрать существующую книгу для просмотра и редактирования информации.

Начало работы над BookDepository

На первом этапе будет реализовано представление детализации BookDepository. На рисунке 6.2 показано, как будет выглядеть приложение BookDepository в конце этого раздела.

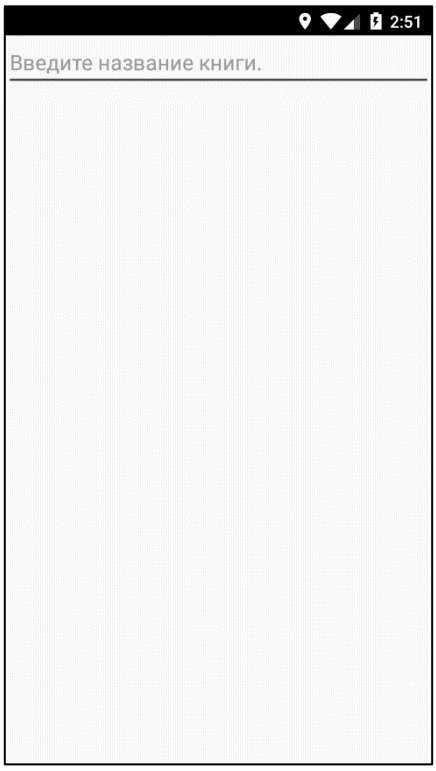

Рисунок 6.2 – Приложение BookDepository в конце раздела 6

Экраном, показанным на рисунке 6.2, будет управлять UI-фрагмент с именем BookFragment. *Хостом* (host) экземпляра BookFragment является активность с именем BookActivity.

Хост предоставляет позицию в иерархии представлений, в которой фрагмент может разместить свое представление (рисунок 6.3). Фрагмент не может вывести представление на экран сам по себе. Его представление отображается только при размещении в иерархии активности.

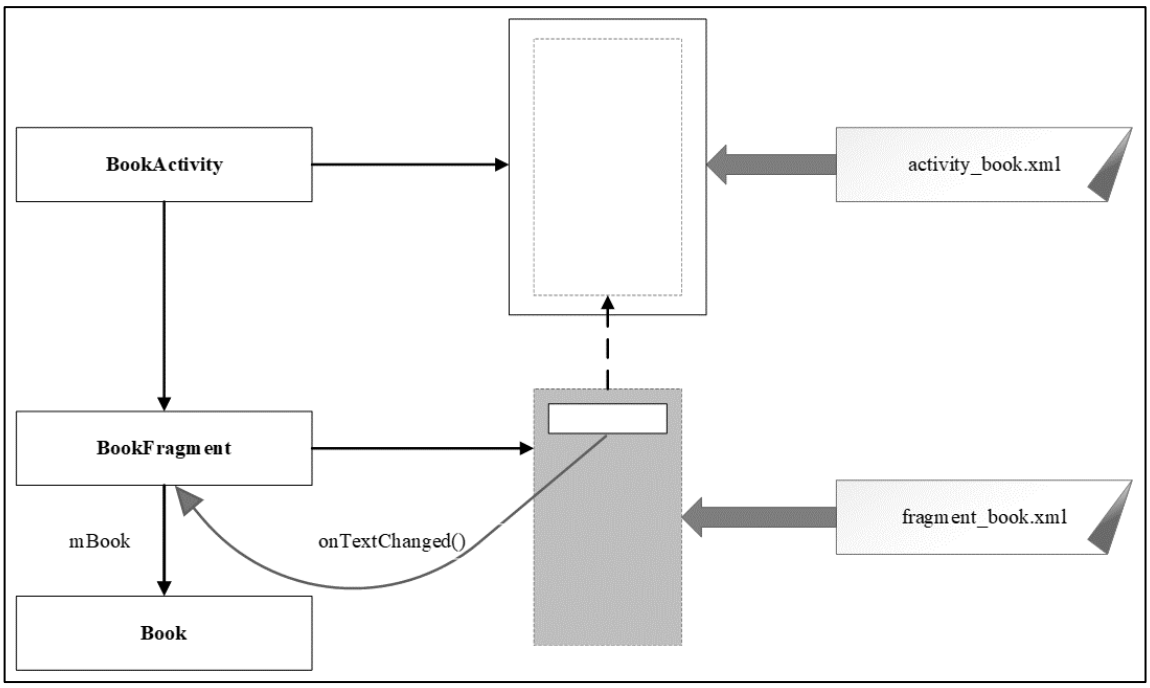

Рисунок 6.3 – BookActivity как хост BookFragment

На рисунке 6.4 изображена общая структура BookDepository. Класс BookFragment создает пользовательский интерфейс и управляет им, а также обеспечивает взаимодействие с объектами модели. В проект будут добавлены три класса, изображенных на рисунке 6.4: Book, BookFragment и BookActivity.

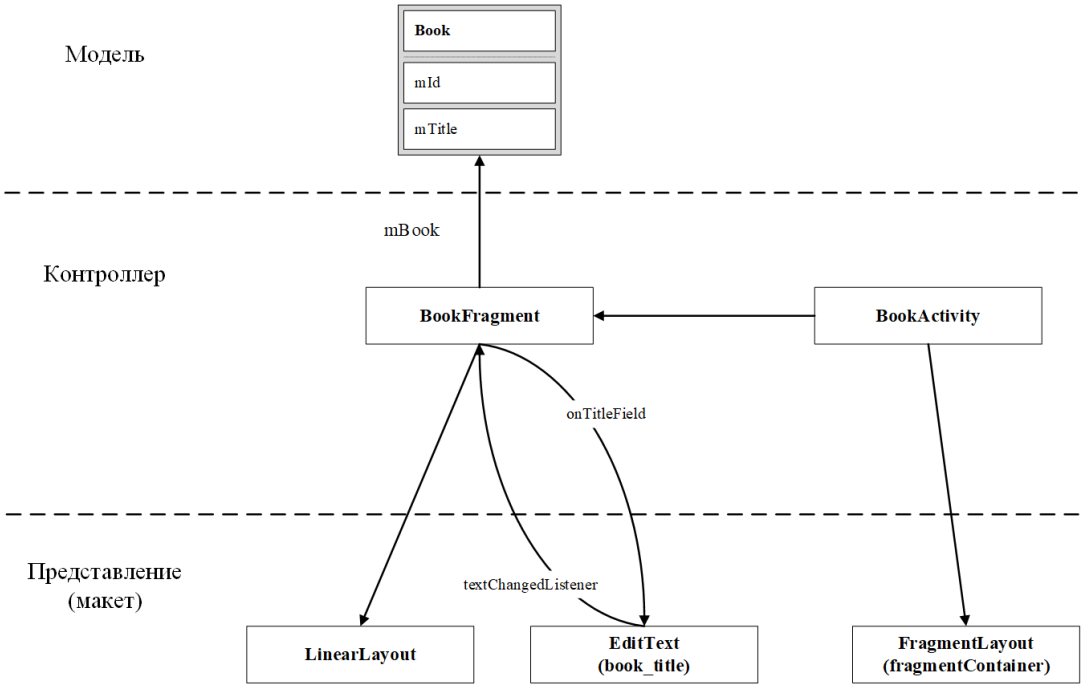

Рисунок 6.4 – Диаграмма объектов BookDepository

Экземпляр Book представляет одну книгу. В этом разделе описание книги будет состоять только из заголовка и идентификатора. Заголовок содержит текст (например, «Война и мир. Том 1.» или «Горе от ума.»), а идентификатор однозначно идентифицирует экземпляр Book.

На данном этапе для простоты будет использоваться один экземпляр Book. В класс BookFragment включается поле (mBook) для хранения этой отдельной книги.

Представление BookActivity состоит из элемента FrameLayout, определяющего место, в котором будет отображаться представление BookFragment.

Представление BookFragment будет состоять из элементов LinearLayout и EditText. BookFragment определяет поле для виджета EditText (mTitleField) и назначает для него слушателя, обновляющего уровень модели при изменении текста.

Создание нового проекта

Создать новое приложение Android (File→New Project…). Ввести имя приложения **BookDepository** и имя пакета ***ru.rsue.android*** (*android* – фамилия студента на латинице).

Щёлкнуть на кнопке Next и задать минимальный уровень SDK: API 19: Android 4.4 – Также проследить за тем, чтобы для типов приложений были установлены только флажки Phone и Tablet.

Снова щёлкнуть на кнопке Next, чтобы выбрать разновидность добавляемой активности. Выбрать пустую активность (**Blank** или **Empty Activity**) и продолжать работу с мастером.

На последнем шаге мастера ввести имя активности **BookActivity** и щёлкнуть на кнопке Finish.

Библиотека поддержки

Фрагменты появились в API уровня 11 вместе с первыми планшетами на базе Android и потребностью в гибкости пользовательского интерфейса. *Библиотека поддержки* (support library) включает полную реализацию фрагментов вплоть до API уровня 4. В нее включены два ключевых класса: `Fragment (android.support.v4.app.Fragment)` и `FragmentActivity (android.support.v4.app.FragmentActivity)`.

Чтобы использовать библиотеку поддержки, необходимо включить ее в состав зависимостей проекта. Для этого надо перейти к структуре проекта (File→Project Structure...). Выбрать в левой части модуль app, а в нем перейти на вкладку **Dependencies**. На этой вкладке перечислены зависимости модуля app. Нажать кнопку «+» и выбрать зависимость Library, чтобы создать новую зависимость. Выбрать в списке библиотеку **support-v4** и щёлкнуть на кнопке OK.

Открыть файл **build.gradle** из модуля app. Проект содержит два файла build.gradle: один для проекта в целом, другой для модуля app. Добавленная библиотека появляется в списке.

Создание уровня модели BookDepository

В окне инструментов Project найти и открыть файл BookActivity.java. Выбрать FragmentActivity суперклассом BookActivity (листинг 6.1).

Листинг 6.1 – Настройка шаблонного кода (BookActivity.java)

```
public class BookActivity extends ~~AppCompatActivity~~ FragmentActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_book);
    }
}
```

В окне инструментов Project щёлкнуть правой кнопкой мыши на пакете ru.rsue.android и выбрать команду New→Java Class. Ввести имя класса **Book** и щёлкнуть на кнопке OK.

Добавьте в Book.java следующий код.

Листинг 6.2 – Добавление кода в класс Book (Book.java)

```
public class Book {
    private UUID mId;
    private String mTitle;
    public Book() {
        mId = UUID.randomUUID(); //Генерирование уникального идентификатора
    }
}
```

Далее следует настроить Android Studio на распознавание префикса **m** в полях классов. Открыть окно настроек Android Studio (команда File→Settings). Открыть раздел Editor, затем раздел Code Style. Выбрать категорию Java и перейти на вкладку Code Generation. В таблице Naming найти строку Field и в поле Naming Prefix ввести префикс **m** для полей. Затем добавить префикс **s** для статических полей в строке Static field.

После этого для свойства mId, доступного только для чтения, необходимо сгенерировать только get-метод, а для свойства mTitle — get- и set-методы.

Листинг 6.3 – Сгенерированные get- и set-методы (Book.java)

```
public class Book {
    private UUID mId;
    private String mTitle;
    public Book() {
        mId = UUID.randomUUID();
    }
    public UUID getId() {
        return mId;
    }
    public String getTitle() {
        return mTitle;
    }
    public void setTitle(String title) {
        mTitle = title;
    }
}
```

Итак, был создан уровень модели и активность, которая выполняет хостинг фрагмента.

Определение контейнерного представления

UI-фрагмент будет добавлен в коде активности-хоста, но все равно необходимо найти место для представления фрагмента в иерархии представлений активности. В макете BookActivity этим местом будет элемент FrameLayout, изображенный на рисунке 6.5.

Элемент FrameLayout станет *контейнерным представлением* для BookFragment. Контейнерное представление абсолютно универсально; оно не привязывается к классу BookFragment. Один макет будет использовать для хостинга разных фрагментов.

Найти макет BookActivity в файле res/layout/activity_book.xml. Открыть файл и заменить макет по умолчанию элементом FrameLayout.

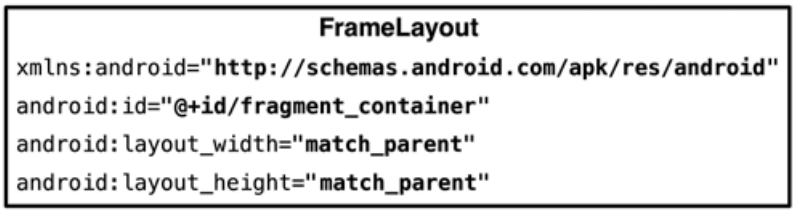

Рисунок 6.5 – Хостинг фрагмента для BookActivity

Разметка XML должна совпадать с приведенной в листинге 6.6.

Листинг 6.4 – Создание контейнера фрагмента (activity_book.xml)

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/fragmentContainer"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
/>
```

Запустить BookDepository, чтобы проверить свой код. Отобразиться только пустой элемент FrameLayout, потому что BookActivity еще не выполняет функции хоста фрагмента (рисунок 6.6).

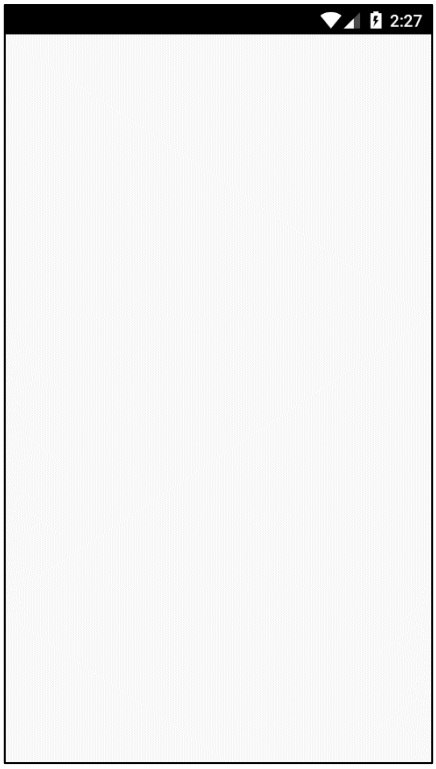

Рисунок 6.6 – Пустой элемент FrameLayout

Создание UI-фрагмента

Представление BookFragment будет отображать информацию, содержащуюся в экземпляре Book. На рисунке 6.7 изображен макет представления BookFragment. Он состоит из вертикального элемента LinearLayout, содержащего EditText — виджет с областью для ввода и редактирования текста.

Чтобы создать файл макета, щёлкнуть правой кнопкой мыши на папке res/layout в окне инструментов Project и выбрать команду New→Layout resource file. Присвойть файлу фрагмента имя fragment_book.xml. Выбрать корневым элементом LinearLayout и щёлкнуть на кнопке OK; Android Studio сгенерирует файл самостоятельно.

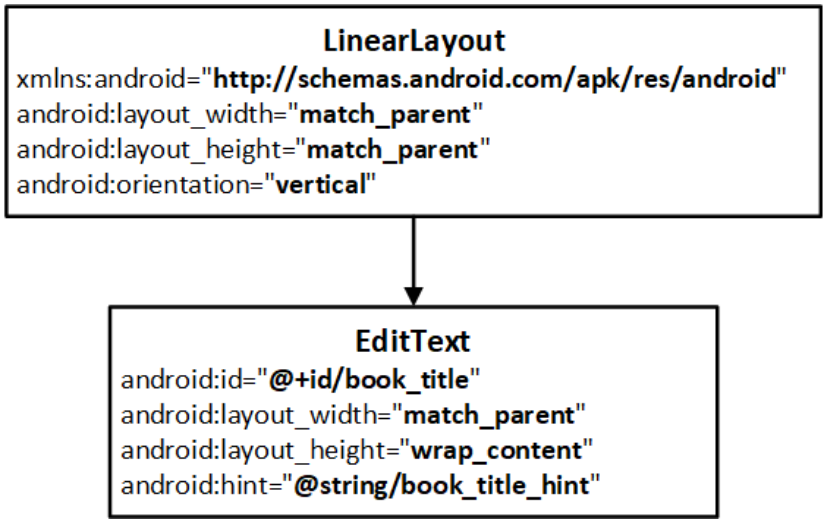

Рисунок 6.7 – Исходный макет BookFragment

Когда файл откроется, следует перейти к разметке XML. Мастер уже добавил элемент LinearLayout. Руководствуясь рисунком 6.7, внести необходимые изменения в fragment_book.xml. Проверить результаты по листингу 6.5.

Листинг 6.5 – Файл макета для представления фрагмента (fragment_book.xml)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical">
  <EditText android:id="@+id/book_title"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="@string/book_title_hint"
  />
</LinearLayout>
```

Открыть файл res/values/strings.xml, добавьте строковый ресурс book_title_hint.

Листинг 6.6 – Добавление и удаление строк (res/values/strings.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">BookDepository</string>
  <string name="hello_world">Hello world!</string>
  <string name="menu_settings">Settings</string>
  <string name="title_activity_book">BookActivity</string>
  <string name="book_title_hint">Введите название книги.</string>
</resources>
```

Сохранить файлы. Вернуться к fragment_book.xml, чтобы увидеть предварительное изображение представления фрагмента.

Создание класса BookFragment

Щёлкнуть правой кнопкой мыши на пакете ru.rsue.android и выбрать команду New→Java Class. Ввести имя класса BookFragment и щёлкнуть на кнопке OK, чтобы сгенерировать класс.

Теперь этот класс нужно преобразовать во фрагмент. Изменить класс BookFragment так, чтобы он субклассировал **Fragment**.

Листинг 6.7 – Субклассирование класса Fragment (BookFragment.java)

```
public class BookFragment extends Fragment {
}
```

При субклассировании Fragment 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. В приложении будет использована версия библиотеки поддержки, поэтому при появлении диалогового окна выбрать версию класса Fragment из android.support.v4.app (рисунок 6.8).

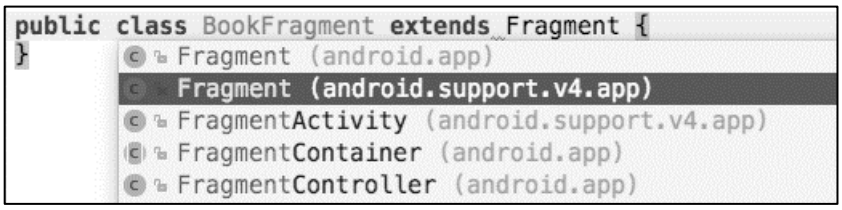

Рисунок 6.8 – Выбор класса Fragment из библиотеки поддержки

Код должен выглядеть так, как показано в листинге 6.8.

Листинг 6.8 – Импортирование Fragment (BookFragment.java)

```
package ru.rsue.android.bookdepository;
import android.support.v4.app.Fragment;
public class BookFragment extends Fragment {}
```

Если файл содержит директиву import для android.app.Fragment, удалить эту строку кода. Импортировать правильный класс Fragment комбинацией клавиш Alt+Enter.

Реализация методов жизненного цикла фрагмента

BookFragment — контроллер, взаимодействующий с объектами модели и представления. Его задача — выдача подробной информации о конкретной книге и ее обновление при модификации пользователем.

В приложении BookDepository большая часть работы контроллера будет выполняться фрагментами в методах жизненного цикла фрагментов.

В файле BookFragment.java добавьте переменную для экземпляра Book и реализацию Fragment.onCreate(Bundle).

Листинг 6.9 – Переопределение Fragment.onCreate(Bundle) (BookFragment.java)

```
public class BookFragment extends Fragment {
    private Book mBook;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mBook = new Book();
    }
}
```

В файле BookFragment.java добавить реализацию `onCreateView(…)`, которая заполняет разметку fragment_book.xml.

Листинг 6.10 – Переопределение onCreateView(…) (BookFragment.java)

```
public class BookFragment extends Fragment {
    private Book mBook;
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
    container, Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_book, container,
        false);
        return v;
    }
}
```

В методе `onCreateView(…)` явно заполняется 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. В данном случае передается false, потому что представление будет добавлено в коде активности.

Подключение виджетов во фрагменте

В методе `onCreateView(…)` также настраивается реакция виджета EditText на ввод пользователя. После того как представление будет заполнено, метод получает ссылку на EditText и добавляет слушателя.

Листинг 6.11 – Настройка виджета EditText (BookFragment.java)

```
public class BookFragment extends Fragment {
    private Book mBook;
    private EditText mTitleField;
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
    container, Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_book, parent, false);
        mTitleField = (EditText)v.findViewById(R.id.book_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) {
                mBook.setTitle(s.toString());
            }
            @Override
            public void afterTextChanged(Editable c) {
                // 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 был использован ранее, является вспомогательным методом, вызывающим `View.findViewById(int)` в своей внутренней реализации. У класса Fragment

аналогичного вспомогательного метода нет, поэтому следует вызывать основной метод.

Назначение слушателей во фрагменте работает точно так же, как в активности. В листинге 6.11 создается анонимный класс, который реализует интерфейс слушателя TextWatcher. Этот интерфейс содержит три метода, но в данный момент интерес представляет только один: `onTextChanged(…)`.

В методе `onTextChanged(…)` вызывается `toString()` для объекта CharSequence, представляющего ввод пользователя. Этот метод возвращает строку, которая затем используется для задания заголовка Book.

Код BookFragment готов.

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 (рисунок 6.9).

FragmentManager управляет двумя структурами: списком фрагментов и стеком транзакций фрагментов.

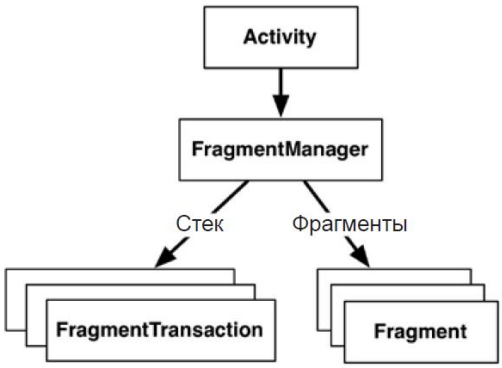

Рисунок 6.9 – FragmentManager

Чтобы добавить фрагмент в активность в коде, следует обратиться с вызовом к объекту FragmentManager активности. Прежде всего необходимо получить сам объект FragmentManager. В BookActivity.java включите следующий код в onCreate(…).

Листинг 6.12 – Получение объекта FragmentManager (BookActivity.java)

```
public class BookActivity extends FragmentActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_book);
        FragmentManager fm = getSupportFragmentManager();
    }
}
```

Метод getSupportFragmentManager() вызывается, потому что в приложении используется библиотека поддержки и класс FragmentActivity. Если бы библиотека поддержки не использовалась, то вместо FragmentActivity можно было бы субклассировать Activity и вызвать getFragmentManager().

Транзакции фрагментов

После получения объекта FragmentManager добавить следующий код, который передает ему фрагмент для управления.

Листинг 6.13 – Добавление BookFragment (BookActivity.java)

```
public class BookActivity extends FragmentActivity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_book);
       FragmentManager fm = getSupportFragmentManager();
       Fragment fragment = fm.findFragmentById(R.id.fragmentContainer);
       if (fragment == null) {
           fragment = new BookFragment();
           fm.beginTransaction()
           .add(R.id.fragmentContainer, fragment)
           .commit();
       }
   }
}
```

Следует обратить внимание на метод add(…) и окружающий его код. Этот код создает и закрепляет *транзакцию фрагмента*.

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 можно 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 листинге 6.13 означает: «Создать новую транзакцию фрагмента, включить в нее одну операцию add, а затем закрепить».

Теперь BookActivity является хостом для BookFragment. Чтобы убедиться в этом, запустить приложение BookDepository. На экране отображается представление из файла fragment_book.xml (рисунок 6.2).

7. МАКЕТЫ И ВИДЖЕТЫ

В данном разделе будут более подробно рассмотрены макеты и виджеты, а также в BookDepository будет включено хранение даты и статуса.

Обновление Book

Открыть файл Book.java и добавить два новых поля. Поле типа Date представляет дату, к которой следует прочитать книгу, а поле типа boolean — признак того, что книга была прочитана.

Листинг 7.1 – Добавление полей в класс Book (Book.java)

```
public class Book {
private UUID mId;
private String mTitle;
private Date mDate;
private boolean mReaded;

public Book() {
        mId = UUID.randomUUID();
        mDate = new Date();
    }
    ...
}
```

Возможно, Android Studio найдет два класса с именем Date. Использовать комбинацию клавиш Alt+Enter, чтобы импортировать класс вручную. Когда будет предложено выбрать импортируемую версию класса Date, надо выбрать версию `java.util.Date`.

Инициализация переменной Date конструктором Date по умолчанию присваивает mDate текущую дату.

Затем сгенерируйте get- и set-методы для новых полей (щёлкнуть правой кнопкой мыши на файле и выбрать команду Generate…→Getter and Setter).

Листинг 7.2 – Сгенерированные get- и set-методы (Book.java)

```
public class Book {
    ...
    public void setTitle(String title) {
        mTitle = title;
    }
    public Date getDate() {
        return mDate;
    }
    public void setDate(Date date) {
        mDate = date;
    }
    public boolean isReaded() {
```

```
        return mReaded;
    }
    public void setReaded(boolean readed) {
        mReaded = readed;
    }
}
```

Далее будет выполнено обновление макета в fragment_book.xml новыми виджетами и их связывание с виджетами в BookFragment.java.

Обновление макета

На рисунке 7.1 представлено, каким образом представление BookFragment будет выглядеть в конце раздела.

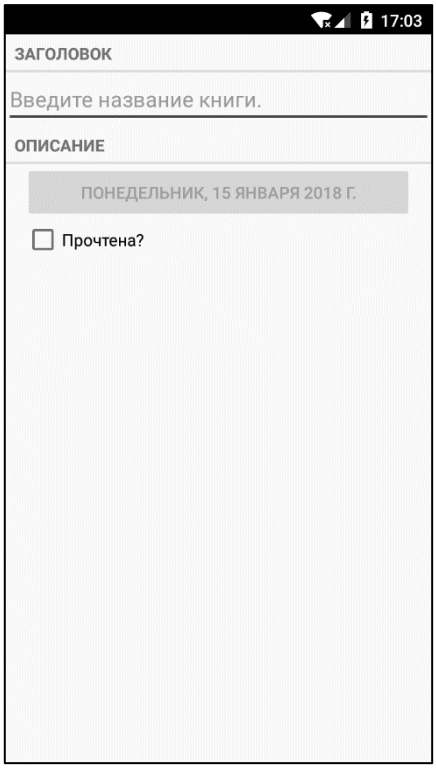

Рисунок 7.1 – Приложение BookDepository в конце раздела 7

Чтобы добиться этого экрана, в макет BookFragment будет добавлено четыре виджета: два виджета TextView, Button и CheckBox.

Открыть файл fragment_book.xml и внести изменения, представленные в листинге 7.3. Возможно, возникнут ошибки отсутствия строковых ресурсов — вскоре они будут созданы.

Листинг 7.3 – Добавление новых виджетов (fragment_book.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=
        "http://schemas.android.com/apk/res/android"
    ...
    >
    <TextView
        android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:text="@string/book_title_label"
        style="?android:listSeparatorTextViewStyle"
    />
    <EditText android:id="@+id/book_title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="16dp"
        android:layout_marginRight="16dp"
        android:hint="@string/book_title_hint"
    />
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/book_details_label"
        style="?android:listSeparatorTextViewStyle"
    />
    <Button android:id="@+id/book_date"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="16dp"
        android:layout_marginRight="16dp"
    />
    <CheckBox android:id="@+id/book_readed"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="16dp"
        android:layout_marginRight="16dp"
        android:text="@string/book_readed_label"
    />
</LinearLayout>
```

Обратить внимание: для виджета Button не был указан атрибут `android:text`. Кнопка будет выводить дату, хранящуюся в Book, а ее текст будет задаваться в коде.

Вернуться к файлу res/values/strings.xml и добавить необходимые строковые ресурсы.

Листинг 7.4 – Добавление строковых ресурсов (strings.xml)

```
<resources>
    <string name="app_name">BookDepository</string>
    ...
    <string name="book_title_label">Заголовок</string>
    <string name="book_details_label">Описание</string>
    <string name="book_readed_label">Прочтена?</string>
</resources>
```

Сохранить файлы и проверить возможные опечатки.

Подключение виджетов

Виджет CheckBox должен показывать, прочтена ли книга. Изменение состояния CheckBox также должно приводить к обновлению поля mReaded класса Book. От Button пока что требуется только вывод даты из поля mDate объекта Book.

В файле BookFragment.java добавить две переменные экземпляра.

Листинг 7.5 – Добавление переменных экземпляра для виджетов (BookFragment.java)

```
public class BookFragment extends Fragment {
private Book mBook;
private EditText mTitleField;
private Button mDateButton;
private CheckBox mReadedCheckBox;
@Override
public void onCreate(Bundle savedInstanceState) {
...
```

Затем в `onCreateView(…)` получите ссылку на новую кнопку, задайте в тексте кнопки дату прочтения книги и заблокируйте ее.

Листинг 7.6 – Назначение текста Button (BookFragment.java)

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
Bundle savedInstanceState) {
   View v = inflater.inflate(R.layout.fragment_book, parent, false);
   ...
   mTitleField.addTextChangedListener(new TextWatcher() {
       ...
   });
   mDateButton = (Button)v.findViewById(R.id.book_date);
   mDateButton.setText(mBook.getDate().toString());
   mDateButton.setEnabled(false);
   return v;
}
```

Блокировка кнопки гарантирует, что она не будет реагировать на нажатия. При этом изменяется оформление кнопки, чтобы сообщить пользователю о заблокированном состоянии. В дальнейшем блокировка кнопки будет снята при назначении слушателя.

Теперь можно заняться CheckBox: необходимо получить ссылку и назначить слушателя, который будет обновлять поле mReaded объекта Book.

Листинг 7.7 – Назначение слушателя для изменений CheckBox (BookFragment.java)

```
...
   mDateButton = (Button)v.findViewById(R.id.book_date);
   mDateButton.setText(mBook.getDate().toString());
   mDateButton.setEnabled(false);
   mReadedCheckBox = (CheckBox)v.findViewById(R.id.book_readed);
   mReadedCheckBox.setOnCheckedChangeListener(new
            OnCheckedChangeListener() {
       @Override
       public void onCheckedChanged(CompoundButton buttonView, boolean
                isChecked) {
                // Назначение флага прочтения книги
                mBook.setReaded(isChecked);
       }
   });
   return v;
}
```

При создании `OnCheckedChangeListener` будет предложено выбрать один из двух вариантов импортирования. Следует выбрать версию `android.widget.CompoundButton`.

Запустить BookDepository. Переключить состояние CheckBox и убедиться в том, что кнопка заблокирована, и на ней отображается текущая дата.

Поля и отступы

В файле fragment_book.xml виджетам назначаются атрибуты margin и padding. 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. Можно допустить, необходимо заметно увеличить размеры кнопки даты без изменения размера текста. Добавить в Button следующий атрибут, сохранить макет и запустить приложение заново (рисунок 7.3).

Листинг 7.8 – Назначение отступов

```
<Button android:id="@+id/book_date"
   ...
   android:layout_marginLeft="16dp"
   android:layout_marginRight="16dp"
   android:padding="80dp" />
```

Следует удалить этот атрибут, прежде чем продолжать работу.

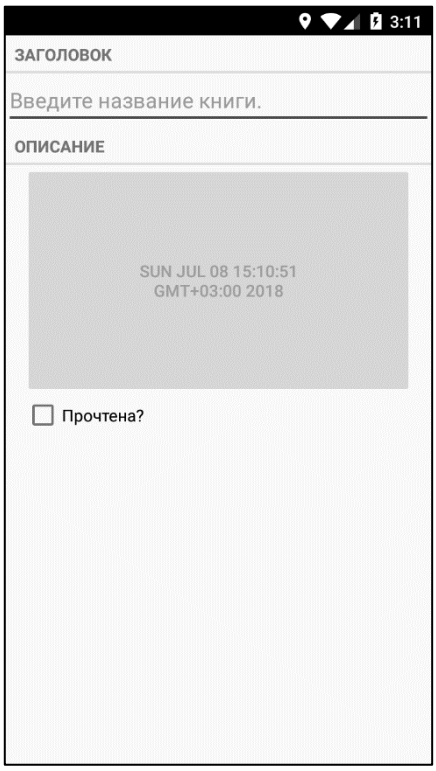

Рисунок 7.3 – Для любителей больших кнопок

Использование графического конструктора

До настоящего момента макеты создавались, вводя разметку XML. В этом разделе будет использован графический конструктор для построения альтернативного альбомного макета BookFragment.

Большинство встроенных классов макетов, таких, как: `LinearLayout`, — автоматически изменяют размеры себя и своих потомков при поворотах. Однако в некоторых случаях изменение размеров по умолчанию недостаточно эффективно использует свободное пространство.

Запустить приложение BookDepository и повернуть устройство, чтобы увидеть макет BookFragment в альбомной ориентации (рисунок 7.4).

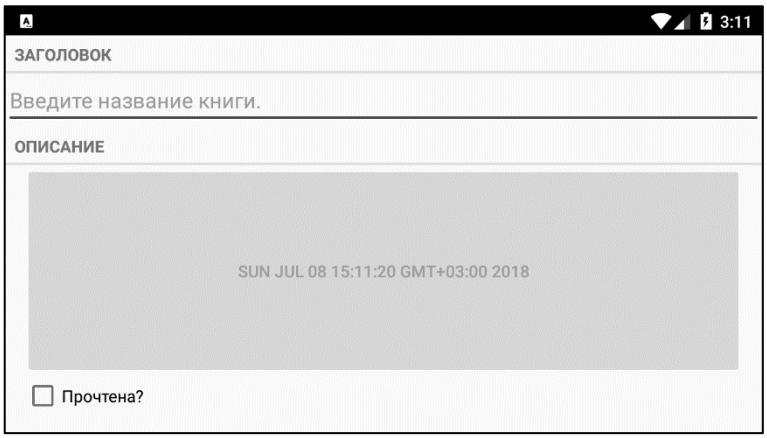

Рисунок 7.4 – BookFragment в альбомной ориентации

Кнопка даты становится слишком длинной; было бы лучше, если бы в альбомной ориентации кнопка и флажок располагались рядом друг с другом.

Чтобы внести эти изменения, переключить в графический конструктор. Открыть файл fragment_book.xml и выбрать вкладку Design в нижней части панели.

Графический конструктор может сгенерировать альбомную версию файла макета. Найти кнопку с изображением листа бумаги в правом верхнем углу (рисунок 7.5). Щёлкнуть на этой кнопке и выбрать команду Create Landscape Variation.

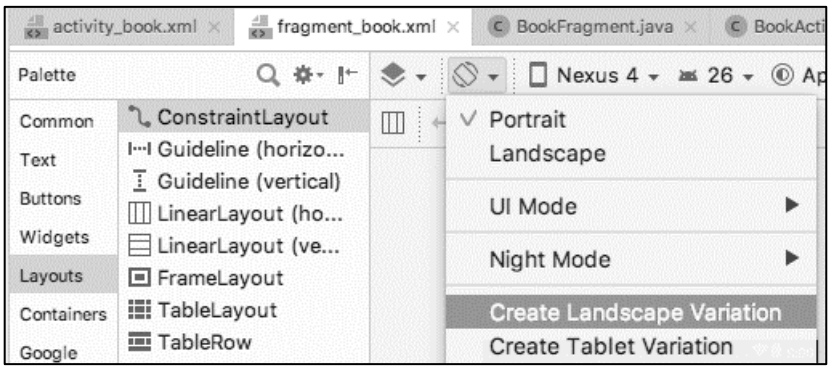

Рисунок 7.5 – Создание альтернативного макета в графическом конструкторе

На экране появляется новый макет. При этом автоматически создается каталог res/layout-land, а существующий файл макета fragment_book.xml копируется в новый каталог.

Следующие изменения необходимо внести в альбомный макет (рисунок 7.6).

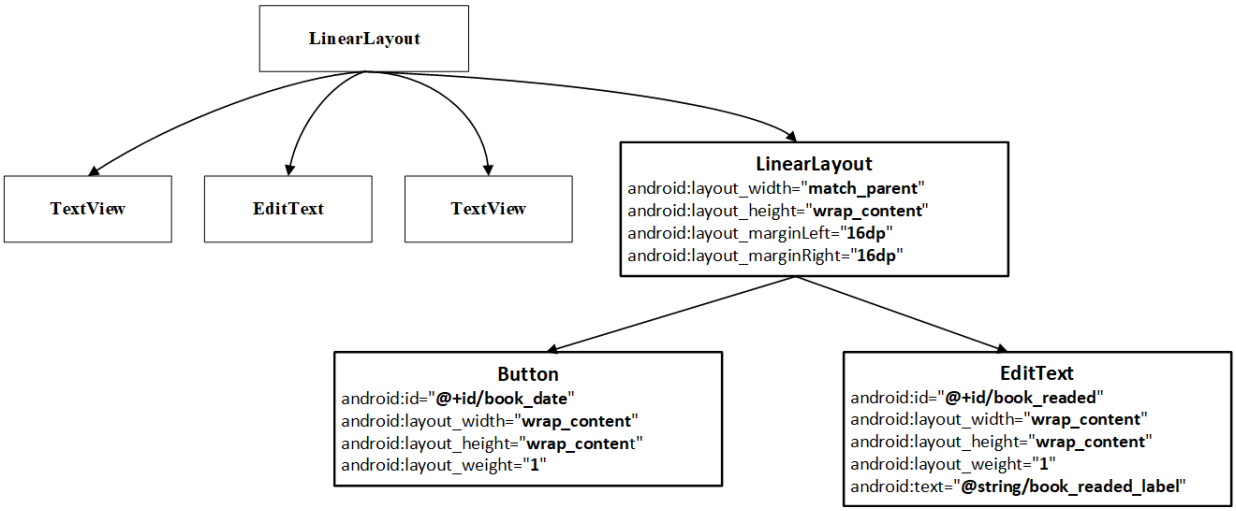

Рисунок 7.6 – Альбомный макет для BookFragment

Изменения можно разделить на четыре группы:

1. Добавление виджета LinearLayout в макет.
2. Редактирование атрибутов LinearLayout.

3. Назначение виджетов Button и CheckBox потомками LinearLayout.
4. Обновление параметров макета Button и CheckBox.

Добавление нового виджета

Чтобы добавить виджет, можно выделить его в палитре и перетащить на дерево компонентов. Щёлкнуть на категории Layouts в палитре, если она еще не раскрыта. Выбрать пункт LinearLayout (Horizontal) и перетащить его на дерево компонентов. Отпустить перетаскиваемый объект над кнопкой даты. Убедиться в том, что новый виджет LinearLayout является потомком корневого элемента LinearLayout, как показано на рисунке 7.7.

Виджеты также можно добавлять перетаскиванием из палитры в область предварительного просмотра. Однако виджеты Layout часто пусты или закрыты другими представлениями, поэтому может быть трудно понять, где следует разместить виджет в области предварительного просмотра для получения нужной иерархии. Перетаскивание на дерево компонентов существенно упрощает выполнение этой операции.

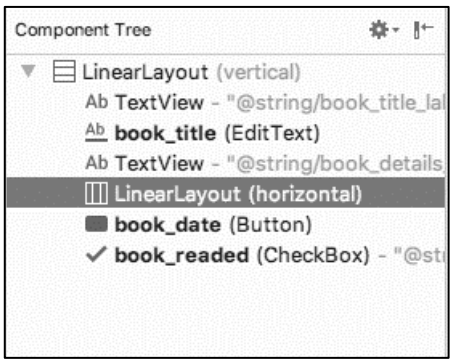

Рисунок 7.7 – Добавление LinearLayout в fragment_book.xml

Редактирование атрибутов в свойствах

Выбрать новый виджет `LinearLayout` в дереве компонентов, чтобы отобразить его атрибуты на панели свойств. Найти атрибуты `layout:width` и `layout:height`.

Присвоить атрибуту `layout:width` значение `match_parent`, а атрибуту `layout:height` — значение `wrap_content`, как показано на рисунке 7.8 – Теперь виджет LinearLayout заполняет всю доступную ширину и занимает столько места по вертикали, сколько необходимо для отображения CheckBox и Button.

Необходимо, чтобы поля нового виджета `LinearLayout` совпадали с полями других виджетов. Открыть атрибут `layout:margin`, выделить поле рядом с Left и ввести значение 16dp. Сделать то же самое для правого (Right) поля (рисунок 7.9).

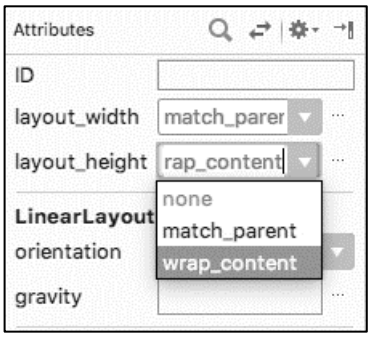

Рисунок 7.8 – Изменение ширины и высоты LinearLayout

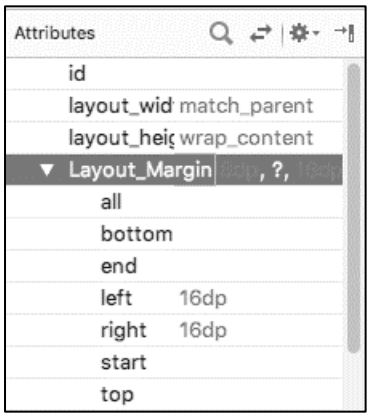

Рисунок 7.9 – Назначение полей в режиме свойств

Сохранить файл макета и переключиться на разметку XML при помощи вкладки text в нижней части области предварительного просмотра. Только что добавленный элемент `LinearLayout` с атрибутами полей будет доступен для просмотра.

Реорганизация виджетов в дереве компонентов

Следующий шаг — назначение виджетов Button и CheckBox потомками нового виджета `LinearLayout`. Вернуться к графическому конструктору, выбрать виджет Button и перетащите его на `LinearLayout`.

В дереве компонентов отражается тот факт, что Button теперь является потомком нового виджета `LinearLayout` (рисунок 7.10). Проделайте то же самое с CheckBox.

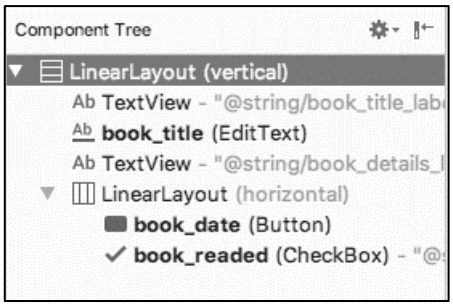

Рисунок 7.10 – Button и Checkbox становятся потомками нового виджета LinearLayout

Если потомки размещаются не в том порядке, их можно переупорядочить посредством перетаскивания. Также в дереве компонентов можно удалять виджеты из макета, но: удаление виджета приводит к удалению его потомков.

В области предварительного просмотра виджет CheckBox не виден — его скрывает Button. Виджет `LinearLayout` проверил ширину (match_parent) своего первого потомка (Button) и выделил ему все пространство, ничего не оставив CheckBox (рисунок 7.11).

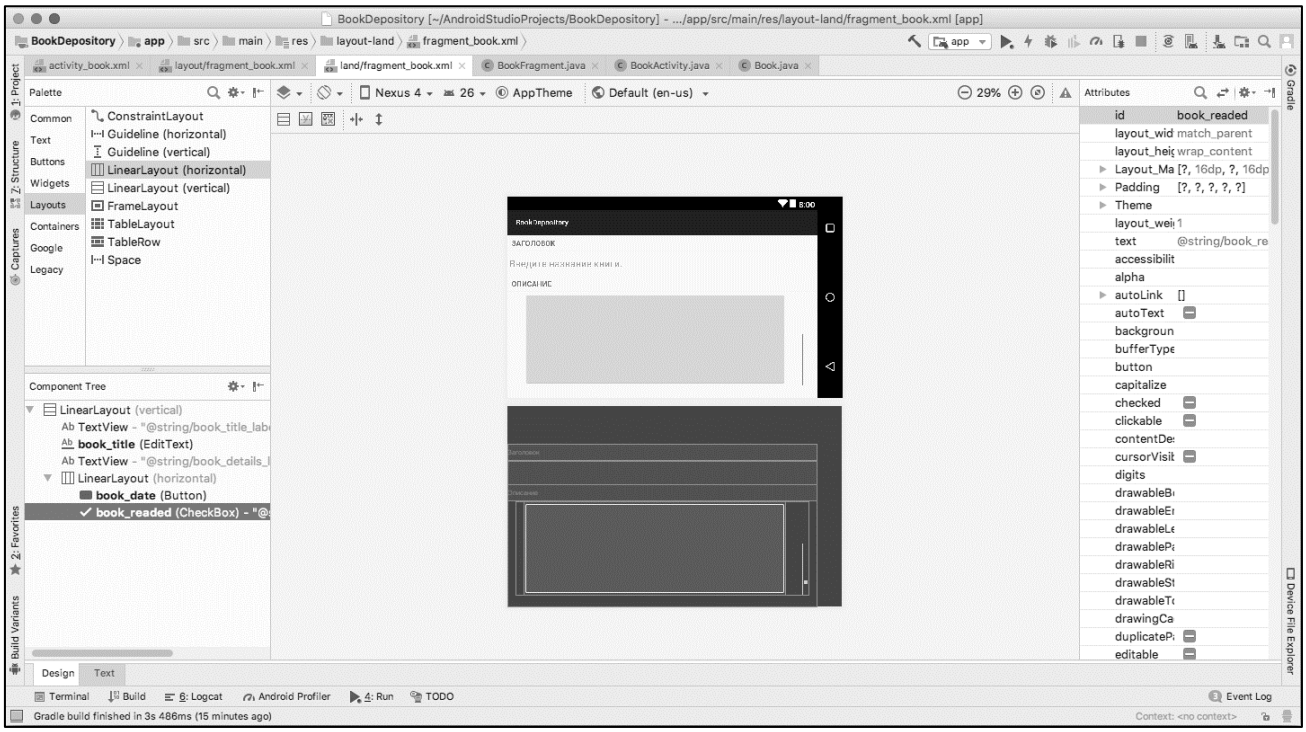

Рисунок 7.11 – Виджет Button, который был определен первым, скрывает CheckBox

Чтобы восстановить равноправие потомков LinearLayout, следует изменить параметры макета потомков.

Обновление параметров макета потомков

Сначала выделить кнопку даты в дереве компонентов. На панели свойств щёлкнуть на текущем значении `layout:width` и заменить его значением wrap_content. Удалить оба значения 16dp полей кнопки. Теперь, когда кнопка находится внутри LinearLayout, поля ей не нужны. Наконец, найти поле `layout:weight` в разделе Layout Parameters и задать ему значение 1. Это поле соответствует атрибуту `android:layout_weight` на рисунке 7.6.

Выбрать виджет CheckBox в дереве компонентов и внести те же изменения: атрибут `layout_width` должен содержать `wrap_content`, атрибуты полей должны быть пустыми, а атрибут Weight должен быть равен 1.

В области предварительного просмотра надо убедиться в том, что оба виджета теперь видны. Сохранить файл и вернуться к XML, чтобы подтвердить изменения. В листинге 7.9 приведена соответствующая разметка XML.

Листинг 7.9 – Разметка XML макета, созданного в графическом конструкторе (layout-land/fragment_book.xml)

```
...
    <TextView
        android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:text="@string/book_details_label"
        style="?android:listSeparatorTextViewStyle"
    />
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="16dp"
        android:layout_marginRight="16dp" >
        <Button
            android:id="@+id/book_date"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1" />
        <CheckBox
            android:id="@+id/book_readed"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="@string/book_readed_label" />
    </LinearLayout>
</LinearLayout>
```

Запустить BookDepository, повернуть устройство в альбомную ориентацию и убедиться в том, что макет для новой конфигурации устройства успешно оптимизирован.

Самостоятельные задания.

Задание. Форматирование даты

На текущий момент объект Date больше напоминает временную метку (timestamp), чем традиционную дату. При вызове `toString()` для Date возвращается именно временная метка, которая отображается на кнопке. Временные метки хорошо подходят для отчетов, но на кнопке было бы лучше выводить дату в формате, более привычном для человека (например, «15 января 2018»). Для этого можно воспользоваться экземпляром класса `android.text.format.DateFormat`. Для выполнения этого задания надо ознакомиться с описанием этого класса в документации Android.

Использовать методы класса DateFormat для формирования строки в стандартном формате или же подготовить собственную форматную строку. Также можно создать форматную строку для вывода дня недели («Понедельник, 15 января 2018»). Например, так:

```
mDateButton.setText(DateFormat.getDateInstance(...).format(mBook
    .getmDate()));
```

8. ВЫВОД СПИСКОВ И LISTFRAGMENT

На данный момент уровень модели BookDepository состоит только из одного единственного экземпляра Book. В этом разделе приложение BookDepository будет обновлено так, чтобы оно поддерживало списки. В списке для каждой книги будет отображаться краткое описание и дата, а также признак её прочтения (рисунок 8.1).

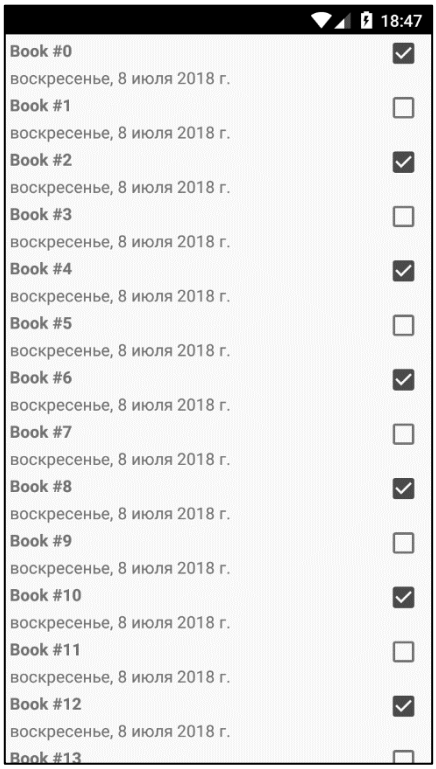

Рисунок 8.1 – Список книг

На рисунке 8.2 показана общая структура приложения BookDepository для этого раздела.

На уровне модели появляется новый объект BookLab, который представляет собой централизованное хранилище для объектов Book.

Для отображения списка на уровне контроллера BookDepository появляется новая активность и новый фрагмент: BookListActivity и BookListFragment.

Обновление уровня модели BookDepository

Прежде всего необходимо преобразовать уровень модели BookDepository из одного объекта Book в список List объектов Book.

Для хранения массива-списка книг будет использоваться *синглетный* (singleton) класс. Такие классы допускают создание только одного экземпляра.

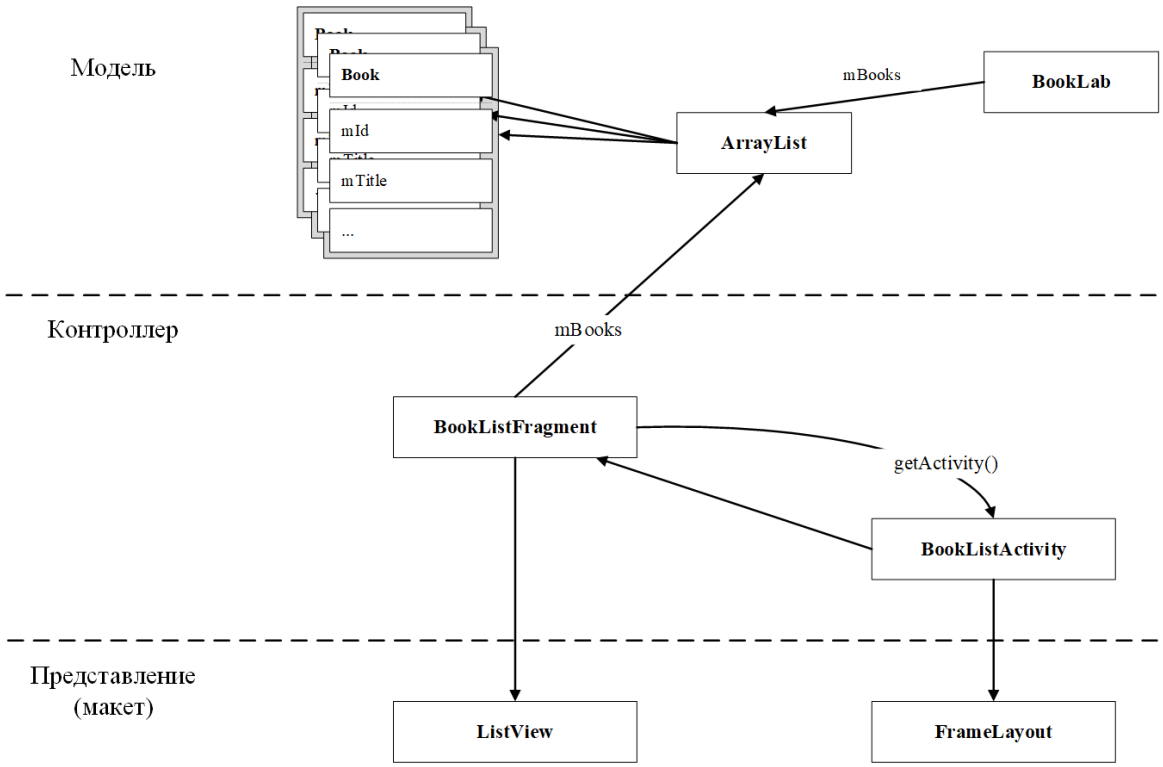

Рисунок 8.2 – Приложение BookDepository со списком

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 жизненными циклами. Синглетные классы будут уничтожаться, когда Android удаляет приложение из памяти. Синглет BookLab не подходит для долгосрочного хранения данных, но позволяет приложению назначить одного владельца данных и предоставляет возможность простой передачи этой информации между классами-контроллерами.

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.

Щёлкнуть правой кнопкой мыши на пакете ru.rsue.android и выбрать команду New→Java Class. Ввести имя класса BookLab и щёлкнуть на кнопке Finish.

В файле BookLab.java реализовать BookLab как синглетный класс с закрытым конструктором и методом `get()`.

Листинг 8.1 – Синглетный класс (BookLab.java)

```
public class BookLab {
    private static BookLab sBookLab;
    public static BookLab get(Context context) {
        if (sBookLab == null) {
            sBookLab = new BookLab(context);
```

```
        }
        return sBookLab;
    }
    private BookLab(Context context) {
    }
}
```

В реализации BookLab есть несколько моментов, заслуживающих внимания. Во-первых, префикс **s** у переменной `sBookLab`. Это условное обозначение используется Android, чтобы показать, что переменная `sBookLab` является статической.

Следует обратить внимание на закрытый конструктор `BookLab`. Другие классы не смогут создать экземпляр `BookLab` в обход метода `get()`.

Для начала предоставить `BookLab` несколько объектов Book для хранения. В конструкторе `BookLab` создайть пустой список List объектов Book. Добавить два метода: `getBooks()` возвращает List, а `getBook(UUID)` возвращает объект Book с заданным идентификатором (листинг 8.2).

Листинг 8.2 – Создание списка List объектов Book (BookLab.java)

```
public class BookLab {
    private static BookLab sBookLab;
    private List<Book> mBooks;
    public static BookLab get(Context context) {
        ...
    }
    private BookLab(Context context) {
        mBooks = new ArrayList<>();
    }
    public List<Book> getBooks() {
        return mBooks;
    }
    public Book getBook(UUID id) {
        for (Book book : mBooks) {
            if (book.getId().equals(id)) {
                return book;
            }
        }
        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.

Со временем List будет содержать объекты Book, созданные пользователем, которые будут сохраняться и загружаться повторно. А пока заполнить массив 100 однообразных объектов Book (листинг 8.3).

Листинг 8.3 – Генерирование тестовых объектов (BookLab.java)

```
private BookLab(Context Context) {
      mBooks = new ArrayList<>();
      for (int i = 0; i < 100; i++) {
           Book book = new Book();
           book.setTitle("Book #" + i);
           book.setReaded(i % 2 == 0); // Для каждого второго объекта
           mBooks.add(book);
      }
}
```

Теперь в приложении имеется полностью загруженный уровень модели и 100 книг для вывода на экран.

Абстрактная активность для хостинга фрагмента

Далее будет создадан класс BookListActivity, предназначенный для выполнения функций хоста для BookListFragment. Но сначала будет создано представление для BookListActivity.

Для BookListActivity можно просто воспользоваться макетом, определенным в файле activity_book.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_book.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_book.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 щёлкнуть правой кнопкой мыши на файле res/layout/activity_book.xml. (Щелкнуть нужно на activity_book.xml, а не на fragment_book.xml.)

Выбрать в контекстном меню команду Refactor→Rename.... Ввести имя activity_fragment.xml. При переименовании ресурса ссылки на него обновляются автоматически.

Среда Android Studio должна автоматически обновить ссылки на новый файл activity_fragment.xml. Если будет получено сообщение об ошибке в BookActivity.java, то придется вручную обновить ссылку BookActivity, как показано в листинге 8.5.

Листинг 8.5 – Обновление файла макета для BookActivity (BookActivity.java)

```
public class BookActivity extends FragmentActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       ~~setContentView(R.layout.activity_book);~~
       setContentView(R.layout.activity_fragment);
       FragmentManager fm = getSupportFragmentManager();
       Fragment fragment =
                fm.findFragmentById(R.id.fragment_container);
       if (fragment == null) {
            ...
       }
    }
}
```

Абстрактный класс Activity

Для создания класса BookListActivity можно повторно использовать код BookActivity. Код, написанный для BookActivity (листинг 8.5): прост и практически универсален. Собственно в нем есть всего одно неуниверсальное место: создание экземпляра BookFragment перед его добавлением в FragmentManager.

Листинг 8.6 – Класс BookActivity почти универсален (BookActivity.java)

```
public class BookActivity extends FragmentActivity {
    @Override
    public 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 BookFragment();
            fm.beginTransaction()
            .add(R.id.fragment_container, fragment)
            .commit();
       }
```

```
    }
}
```

Почти в каждой активности, которая будет создаваться в этом приложении, будет присутствовать такой же код. Чтобы избежать повторений, следует выделить его в абстрактный класс.

Создать новый класс с именем `SingleFragmentActivity` в пакете BookDepository. Сделать его субклассом `FragmentActivity` и объявить абстрактным классом.

Листинг 8.7 – Создание абстрактной активности (SingleFragmentActivity.java)

```
public abstract class SingleFragmentActivity extends FragmentActivity {
}
```

Теперь включить следующий фрагмент в SingleFragmentActivity.java. Не считая выделенных частей, он идентичен старому коду BookActivity.

Листинг 8.8 – Добавление обобщенного суперкласса (SingleFragmentActivity.java)

```
public abstract class SingleFragmentActivity extends FragmentActivity {
    protected abstract Fragment createFragment();
    @Override
    public 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 осуществляется поиск фрагмента в FragmentManager этого контейнера, создавая и добавляя его, если он не существует.

Код в листинге 8.8 отличается от кода BookActivity 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

Открыть класс с именем BookActivity, назначить его суперклассом SingleFragmentActivity, удалить реализацию `onCreate(Bundle)` и реализовать метод `createFragment()` так, как показано в листинге 8.9.

Листинг 8.9 – Переработка BookActivity (BookActivity.java)

```
public class BookActivity extends FragmentActivity
        SingleFragmentActivity {
    @Override
    public 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 BookFragment();
            fm.beginTransaction()
            .add(R.id.fragment_container, fragment)
            .commit();
        }
    }
    @Override
    protected Fragment createFragment() {
        return new BookFragment();
    }
}
```

Создание контроллеров для вывода списков

Далее будут создаданы два новых класса-контроллера: BookListActivity и BookListFragment.

Щёлкнуть правой кнопкой мыши на пакете ru.rsue.android, выбрать команду New→Java Class и присвоить классу имя BookListActivity.

Изменить новый класс BookListActivity так, чтобы он тоже субклассировал SingleFragmentActivity и реализовал метод createFragment().

Листинг 8.10 – Реализация BookListActivity (BookListActivity.java)

```
public class BookListActivity extends SingleFragmentActivity {
    @Override
    protected Fragment createFragment() {
        return new BookListFragment();
    }
}
```

Если класс BookListActivity содержит другие методы, такие, как: onCreate — удалить их. Пусть класс SingleFragmentActivity выполняет свою работу, а реализация BookListActivity будет по возможности простой.

Для создания класса BookListFragment снова щёлкнуть правой кнопкой мыши на пакете ru.rsue.android, выбрать команду New→Java Class и присвоить классу имя BookListFragment.

Листинг 8.11 – Реализация BookListFragment (BookListActivity.java)

```
public class BookListFragment extends Fragment {
   // Пока пусто
}
```

Пока BookListFragment остается пустой оболочкой фрагмента. Он будет использован позднее.

Объявление BookListActivity

Теперь, когда класс BookListActivity создан, его следует объявить в манифесте. Кроме того, список книг должен выводиться на первом экране, который виден пользователю после запуска BookDepository; следовательно, активность BookListActivity должна быть активностью лаунчера.

Включить в манифест объявление BookListActivity и переместить фильтр интентов из объявления BookActivity в объявление BookListActivity, как показано в листинге 8.12.

Листинг 8.12 – Объявление BookListActivity активностью лаунчера (AndroidManifest.xml)

```
...
   <application
   ...
   android:label="@string/app_name"
   android:theme="@style/AppTheme" >
      <activity android:name=".BookListActivity">
         <intent-filter>
         <action android:name="android.intent.action.MAIN" />
         <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
      <activity android:name=".BookActivity"
         android:label="@string/app_name">
         <intent-filter>
         <action android:name="android.intent.action.MAIN" />
         <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
   </application>
</manifest>
```

BookListActivity теперь является активностью лаунчера. Запустить BookDepository; на экране появляется виджет FrameLayout из BookListActivity, содержащий пустой фрагмент BookListFragment (рисунок 8.3).

RecyclerView

Итак, в BookListFragment должен отображаться список. Для этого следует воспользоваться 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.

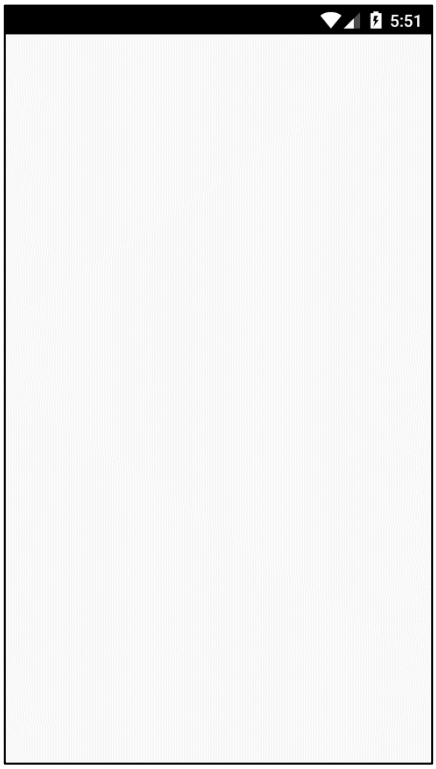

Рисунок 8.3 – Пустой экран BookListActivity

Первая реализация передачи данных для отображения будет очень простой: в элементе списка будет отображаться только краткое описание объекта Book, а объект View представляет собой простой виджет TextView (рисунок 8.4).

На рисунке 8.4 изображены 12 виджетов TextView. Позднее пользователь сможет провести по экрану BookDepository, чтобы прокрутить 100 виджетов TextView для просмотра всех объектов Book.

Единственная обязанность RecyclerView — повторное использование виджетов TextView и их позиционирование на экране. Чтобы обеспечить их

исходное размещение, он работает с двумя классами: субклассом Adapter и субклассом ViewHolder.

Класс RecyclerView находится в одной из многочисленных библиотек поддержки Google. Первым шагом в использовании RecyclerView станет добавление библиотеки RecyclerView к зависимостям приложения.

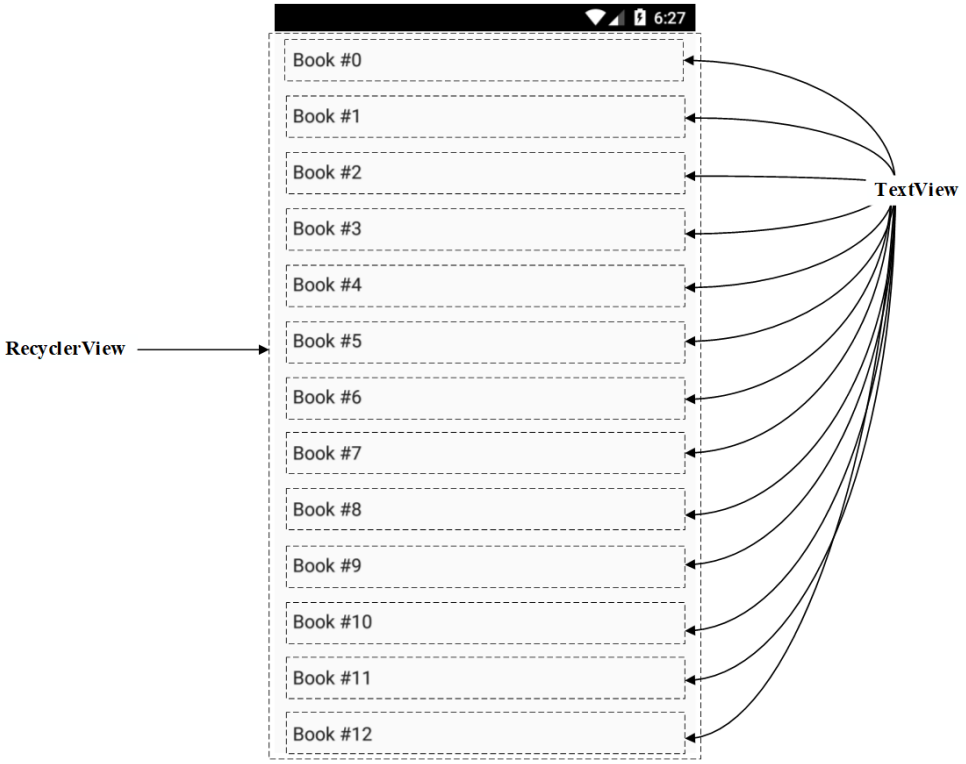

Рисунок 8.4 – RecyclerView с дочерними виджетами TextView

Открыть окно структуры проекта командой File→Project Structure.... Выбрать модуль app слева, перейти на вкладку Dependencies. Щёлкнуть на кнопке + и выбрать Library dependency, чтобы добавить зависимость.

Найти и выделить библиотеку recyclerview-v7; щёлкнуть на кнопке OK, чтобы добавить библиотеку как зависимость (рисунок 8.5).

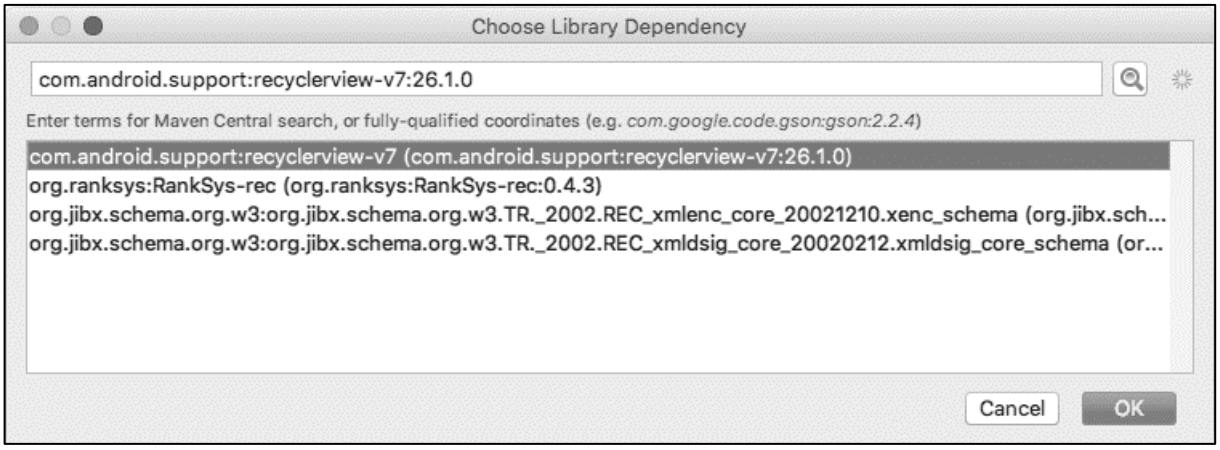

Рисунок 8.5 – Добавление зависимости для RecyclerView

Виджет RecyclerView будет находиться в файле макета BookListFragment. Сначала необходимо создать файл макета: щёлкнуть правой кнопкой мыши на каталоге res/layout и выбрать команду New→Layout resource file. Ввести имя fragment_book_list и щёлкнуть на кнопке OK, чтобы создать файл.

Открыть только что созданный файл fragment_book_list, заменить его корневое представление на RecyclerView и присвойть ему идентификатор.

Листинг 8.13 – Включение RecyclerView в файл макета (fragment_book_list.xml)

```
<android.support.v7.widget.RecyclerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/book_recycler_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

Представление BookListFragment готово; теперь его нужно связать с фрагментом. Изменить класс BookListFragment так, чтобы он использовал этот файл макета и находил RecyclerView в файле макета, как показано в листинге 8.14.

Листинг 8.14 – Подготовка представления для BookListFragment (BookListFragment.java)

```
public class BookListFragment extends Fragment {
 private RecyclerView mBookRecyclerView;
 @Override
 public View onCreateView(LayoutInflater inflater, ViewGroup container,
             Bundle savedInstanceState) {
   View view = inflater.inflate(R.layout.fragment_book_list, container,
       false);
   mBookRecyclerView = (RecyclerView) view
       .findViewById(R.id.book_recycler_view);
   mBookRecyclerView.setLayoutManager(new LinearLayoutManager
       (getActivity()));
   return view;
 }
}
```

Сразу же после создания виджета RecyclerView ему назначается другой объект LayoutManager. Это необходимо для работы виджета RecyclerView. Если ему не будет предоставлен объект LayoutManager, то возникнет ошибка.

Запустить приложение. Снова отобразится пустой экран, но сейчас перед пользователем пустой виджет RecyclerView. Объекты Book остаются невидимыми до тех пор, пока не будут определены реализации Adapter и ViewHolder.

Реализация адаптера и ViewHolder

На первом этапе следует реализовать определение ViewHolder как внутреннего класса BookListFragment.

Листинг 8.15 – Простая реализация ViewHolder (BookListFragment.java)

```
public class BookListFragment extends Fragment {
    ...
    private class BookHolder extends RecyclerView.ViewHolder {
        public TextView mTitleTextView;
        public BookHolder(View itemView) {
            super(itemView);
            mTitleTextView = (TextView) itemView;
        }
    }
}
```

В своем текущем виде ViewHolder хранит ссылку на одно представление: виджет TextView для заголовка. Ожидается, что itemView относится к типу TextView, в противном случае при выполнении кода произойдет ошибка.

После определения ViewHolder создать адаптер.

Листинг 8.16 – Начало работы над адаптером (BookListFragment.java)

```
public class BookListFragment extends Fragment {
    ...
    private class BookAdapter extends RecyclerView.Adapter<BookHolder> {
        private List<Book> mBooks;
        public BookAdapter(List<Book> books) {
            mBooks = books;
        }
    }
}
```

(Код в листинге 8.16 не компилируется. Вскоре этот недостаток будет исправлен.)

Класс RecyclerView взаимодействует с адаптером, когда потребуется создать объект ViewHolder или связать его с объектом Book. Сам виджет RecyclerView ничего не знает об объекте Book, но адаптер располагает полной информацией о Book.

Далее следует реализовать три метода BookAdapter.

Листинг 8.17 – Реализация методов BookAdapter (BookListFragment.java)

```
private class BookAdapter extends RecyclerView.Adapter<BookHolder> {
    ...
    @Override
    public BookHolder 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 BookHolder(view);
    }
    @Override
    public void onBindViewHolder(BookHolder holder, int position) {
        Book book = mBooks.get(position);
        holder.mTitleTextView.setText(book.getTitle());
    }
    @Override
    public int getItemCount() {
        return mBooks.size();
    }
}
```

Итак, адаптер готов; остается связать его с RecyclerView. Следует реализовать метод updateUI, который настраивает пользовательский интерфейс BookListFragment. Пока он создает объект BookAdapter и назначает его RecyclerView.

Листинг 8.18 – Подготовка адаптера (BookListFragment.java)

```
public class BookListFragment extends Fragment {
    private RecyclerView mBookRecyclerView;
    private BookAdapter mAdapter;
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
                container, Bundle savedInstanceState) {
        ...
        mBookRecyclerView.setLayoutManager(new LinearLayoutManager
            (getActivity()));
        updateUI();
        return view;
    }
    private void updateUI() {
        BookLab bookLab = BookLab.get(getActivity());
        List<Book> books = bookLab.getBooks();
        mAdapter = new BookAdapter(books);
        mBookRecyclerView.setAdapter(mAdapter);
    }
    ...
}
```

Запустить приложение BookDepository и прокрутить новый список RecyclerView, который должен выглядеть примерно так, как показано на рисунке 8.6.

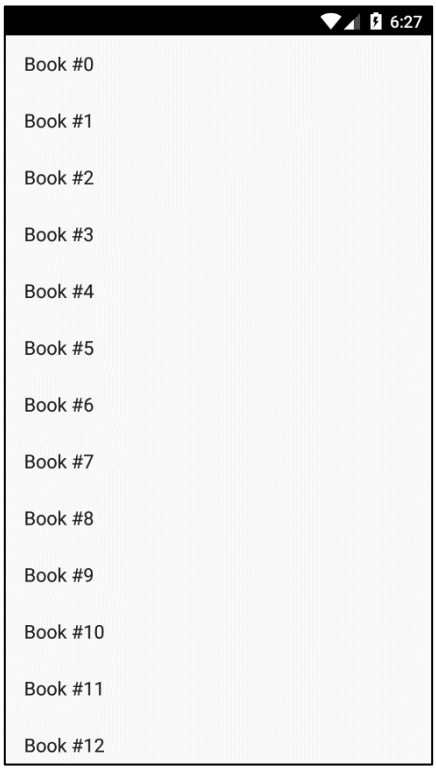

Рисунок 8.6 – Список объектов Book

Создание макета элемента списка

В приложении BookDepository макет элемента списка должен включать краткое описание книги, дату и признак прочтения (рисунок 8.7). Такой макет состоит из двух виджетов — TextView и CheckBox.

Рисунок 8.7 – Список с пользовательским макетом элементов

Новый макет элемента списка создается точно так же, как для представления активности или фрагмента. В окне инструментов Project щёлкнуть правой кнопкой мыши на каталоге res/layout и выбрать команду New→Layout resource file. В открывшемся диалоговом окне ввести имя файла list_item_book, выбрать корневой элемент RelativeLayout и щёлкнуть на кнопке OK.

Листинг 8.19 – Макет пользовательского элемента списка (list_item_book.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android=
                "http://schemas.android.com/apk/res/android"
                android:layout_width="match_parent"
                android:layout_height="wrap_content">
    <CheckBox
            android:id="@+id/list_item_book_readed_check_box"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentRight="true"
            android:padding="4dp"/>

    <TextView
            android:id="@+id/list_item_book_title_text_view"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_toLeftOf=
           "@id/list_item_book_readed_check_box"
            android:textStyle="bold"
            android:padding="4dp"
            tools:text="Заголовок книги"/>

    <TextView
            android:id="@+id/list_item_book_date_text_view"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_toLeftOf=
           "@id/list_item_book_readed_check_box"
            android:layout_below="@id/list_item_book_title_text_view"
            tools:text="Дата прочтения книги"/>
</RelativeLayout>
```

Использование нового представления элемента списка

Теперь внести изменения в класс BookAdapter, чтобы использовать новый файл макета list_item_book.

Листинг 8.20 – Заполнение пользовательского макета (BookListFragment.java)

```
private class BookAdapter extends RecyclerView.Adapter<BookHolder> {
   ...
   @Override
   public BookHolder onCreateViewHolder(ViewGroup parent, int viewType)
   {
      LayoutInflater layoutInflater =
                LayoutInflater.from(getActivity());
```

```
        View view = layoutInflater
            .inflate(android.R.layout.simple_list_item_1
                R.layout.list_item_book, parent, false);
        return new BookHolder(view);
    }
    ...
}
```

Наконец, пора наделить BookHolder новыми обязанностями. Внести изменения в класс BookHolder, чтобы он получал виджет TextView с кратким описанием, TextView с датой и CheckBox с признаком прочтения книги.

Листинг 8.21 – Поиск представлений в BookHolder (BookListFragment.java)

```
private class BookHolder extends RecyclerView.ViewHolder {
    public TextView mTitleTextView;
    private TextView mTitleTextView;
    private TextView mDateTextView;
    private CheckBox mReadedCheckBox;
    public BookHolder(View itemView) {
        super(itemView);
        mTitleTextView = (TextView) itemView;
        mTitleTextView = (TextView)
            itemView.findViewById(R.id.list_item_book_title_text_view);
        mDateTextView = (TextView)
            itemView.findViewById(R.id.list_item_book_date_text_view);
        mReadedCheckBox = (CheckBox)
            itemView.findViewById(R.id.list_item_book_readed_check_box)
    }
}
```

Добавить в BookHolder метод bindBook(Book), чтобы немного упростить код.

Листинг 8.22 – Связывание представлений в BookHolder (BookListFragment.java)

```
private class BookHolder extends RecyclerView.ViewHolder {
    private Book mBook;
    ...
    public void bindBook(Book book) {
        mBook = book;
        mTitleTextView.setText(mBook.getTitle());
        mDateTextView.setText(mBook.getDate().toString());
        mReadedCheckBox.setChecked(mBook.isReaded());
    }
}
```

Получив объект Book, объект BookHolder обновляет TextView с кратким описанием, TextView с датой и CheckBox с признаком прочтения книги в соответствии с содержимым Book.

Теперь у BookHolder есть все необходимое для выполнения его работы. BookAdapter достаточно использовать новый метод bindBook.

Листинг 8.23 – Связывание адаптера с BookHolder (BookListFragment.java)

```
private class BookAdapter extends RecyclerView.Adapter<BookHolder> {
    ...
    @Override
    public void onBindViewHolder(BookHolder holder, int position) {
        Book book = mBooks.get(position);
        holder.mTitleTextView.setText(book.getTitle());
        holder.bindBook(book);
    }
    ...
}
```

Запустить BookDepository и понаблюдать за новым макетом list_item_book в действии (рисунок 8.9).

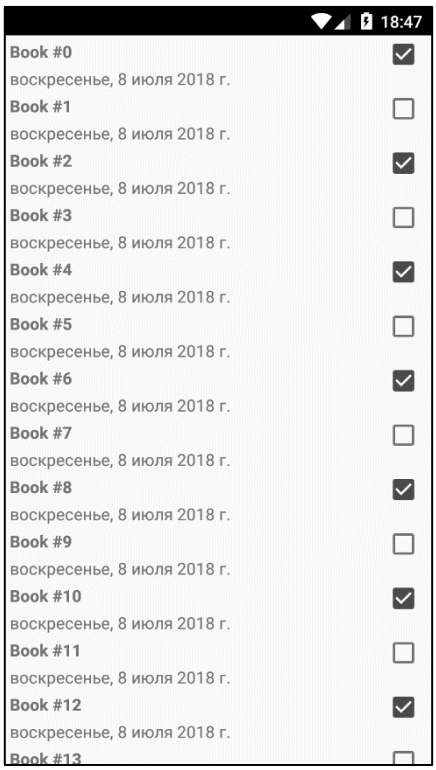

Рисунок 8.9 – А теперь с новыми макетами элементов!

Далее будет реализовано отображение простого уведомления Toast при касании объекта Book в списке. Так как каждое представление View связывается с некоторым ViewHolder, то необходимо сделать объект ViewHolder реализацией OnClickListener для своего View.

Внести изменения в класс BookHolder для обработки касаний в строках.

Листинг 8.24 – Обработка касаний в BookHolder (BookListFragment.java)

```
private class BookHolder extends RecyclerView.ViewHolder
        implements View.OnClickListener {
    ...
    public BookHolder(View itemView) {
        super(itemView);
        itemView.setOnClickListener(this);
        ...
    }
    ...
    @Override
    public void onClick(View v) {
        Toast.makeText(getActivity(),
            mBook.getTitle() + " clicked!", Toast.LENGTH_SHORT)
                .show();
    }
}
```

В листинге 8.24 объект BookHolder реализует интерфейс OnClickListener. Для itemView — представления View всей строки — BookHolder назначается получателем событий щелчка.

Запустить приложение BookDepository и коснуться строки в списке. На экране появляется уведомление Toast с сообщением о касании.

9. АРГУМЕНТЫ ФРАГМЕНТОВ

В данном разделе в приложении BookDepository будет реализована совместная работа списка и детализации. Когда пользователь щелкает на элементе списка книг, на экране возникает новый экземпляр BookActivity, который является хостом для экземпляра BookFragment с подробной информацией о конкретном экземпляре Book (рисунок 9.1).

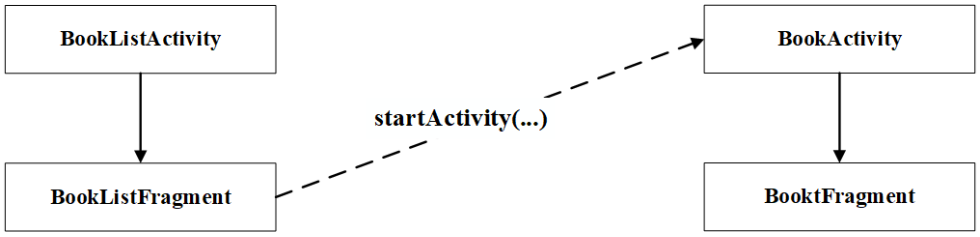

Рисунок 9.1 – Запуск BookActivity из BookListActivity

Запуск активности из фрагмента

Запуск активности из фрагмента осуществляется практически так же, как запуск активности из другой активности. Будет вызван метод `Fragment.startActivity(Intent)`, который вызывает соответствующий метод Activity во внутренней реализации.

В реализации `onListItemClick(…)` из BookListFragment заменить уведомление кодом, запускающим экземпляр BookActivity.

Листинг 9.1 – Запуск BookActivity (BookListFragment.java)

```
private class BookHolder extends RecyclerView.ViewHolder
implements View.OnClickListener {
    ...
    @Override
    public void onClick(View v) {
        ~~Toast.makeText(getActivity(),~~
            ~~mBook.getTitle() + " clicked!", Toast.LENGTH_SHORT)~~
            ~~.show();~~

        Intent intent = new Intent(getActivity(), BookActivity.class);
        startActivity(intent);
    }
}
```

Класс BookListFragment создает явный интент с указанием класса BookActivity. BookListFragment использует метод `getActivity()` для передачи активности-хоста как объекта Context, необходимого конструктору Intent.

Запустить приложение BookDepository.

Щёлкнуть на любой строке списка; открывается новый экземпляр BookActivity, управляющий фрагментом BookFragment (рисунок 9.2).

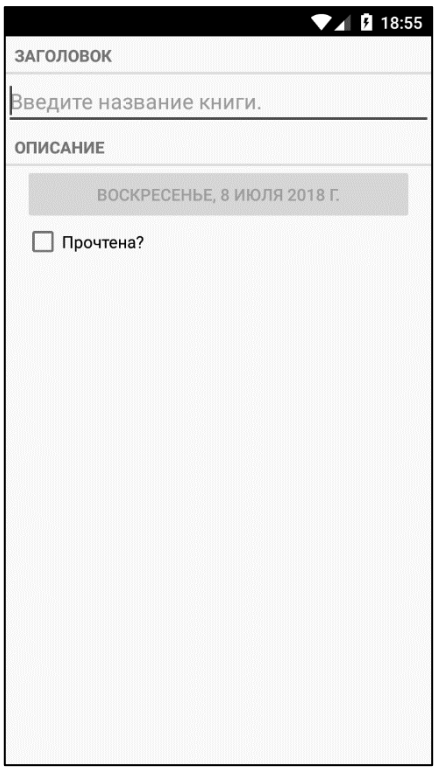

Рисунок 9.2 – Запуск пустого экземпляра BookFragment

Экземпляр BookFragment еще не содержит данных конкретного объекта Book, потому что он не получает сведений, какой именно объект Book следует отображать.

Использование дополнений

Включение дополнения

Чтобы сообщить BookFragment, какой объект Book следует отображать, можно передать идентификатор в дополнении (extra) объекта Intent при запуске BookActivity.

На первом этапе следует создать новый метод newIntent в BookActivity.

Листинг 9.2 – Создание нового метода newIntent (BookActivity.java)

```
public class BookActivity extends SingleFragmentActivity {
    public static final String EXTRA_BOOK_ID =
        "ru.rsue.android.bookdepository.book_id";
    public static Intent newIntent(Context packageContext, UUID bookId)
    {
        Intent intent = new Intent(packageContext, BookActivity.class);
        intent.putExtra(EXTRA_BOOK_ID, bookId);
        return intent;
    }
    ...
}
```

После создания явного интента вызывается метод putExtra(…), передавая строковый ключ и связанное с ним значение (bookId). В данном случае вызывается версия putExtra(String, Serializable), потому что UUID является объектом Serializable.

Затем необходимо обновить класс BookHolder, чтобы он использовал метод newIntent с передачей идентификатора книги.

Листинг 9.3 – Сохранение и передача Book (BookListFragment.java)

```
private class BookHolder extends RecyclerView.ViewHolder
        implements View.OnClickListener {
    ...
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(getActivity(), BookActivity.class);
        Intent intent = BookActivity.newIntent(getActivity(),
            mBook.getId());
        startActivity(intent);
    }
}
```

Чтение дополнения

Идентификатор книги сохранен в интенте, принадлежащем BookActivity, однако прочитать и использовать эти данные должен класс BookFragment.

Существуют два способа, которыми фрагмент может обратиться к данным из интента активности: простое и прямолинейное обходное решение и сложная, гибкая полноценная реализация. Сначала будет использован первый способ, а потом реализуовано сложное гибкое решение с *аргументами фрагментов*.

В простом решении BookFragment просто использует метод getActivity() для прямого обращения к интенту BookActivity. Вернуться к классу BookFragment, прочитать дополнение из интента BookActivity и использовать его для получения данных Book.

Листинг 9.4 – Сохранение и передача Book (BookFragment.java)

```
public class BookFragment extends Fragment {
    ...
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mBook = new Book();
        UUID bookId = (UUID) getActivity().getIntent()
            .getSerializableExtra(BookActivity.EXTRA_BOOK_ID);
        mBook = BookLab.get(getActivity()).getBook(bookId);
    }
    ...
```

Если не считать вызова `getActivity()`, листинг 9.4 практически не отличается от кода выборки дополнения из кода активности. Метод `getIntent()` возвращает объект Intent, используемый для запуска BookActivity.

Метод `getSerializableExtra(String)` вызывается для Intent, чтобы извлечь UUID в переменную. После получения идентификатора, он используется для получения объекта Book от BookLab.

Обновление представления BookFragment данными Book

Теперь, когда фрагмент BookFragment получает объект Book, его представление может отобразить данные Book. Обновите метод `onCreateView(…)`, чтобы он выводил краткое описание книги и признак прочтения (код вывода даты уже имеется).

Листинг 9.5 – Обновление объектов представления (BookFragment.java)

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
Bundle savedInstanceState) {
  ...
  mTitleField = (EditText)v.findViewById(R.id.book_title);
  mTitleField.setText(mBook.getTitle());
  mTitleField.addTextChangedListener(new TextWatcher() {
      ...
  });
  ...
  mReadedCheckBox = (CheckBox)v.findViewById(R.id.book_readed);
  mReadedCheckBox.setChecked(mBook.isReaded());
  mReadedCheckBox.setOnCheckedChangeListener(new
          OnCheckedChangeListener() {
      ...
  });
  ...
  return v;
}
```

Запустить приложение BookDepository. Выбрать строку Book #4 и убедиться в том, что на экране появится экземпляр BookFragment с правильными данными книги (рисунок 9.3).

Недостаток прямой выборки

Обращение из фрагмента к интенту, принадлежащему активности-хосту, упрощает код. С другой стороны, оно нарушает инкапсуляцию фрагмента. Класс BookFragment 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 ru.rsue.android.bookdepository.book_id.

Рисунок 9.3 – Книга, выбранная в списке

Возможно, для BookFragment такое предположение разумно, но оно означает, что класс BookFragment в своей текущей реализации не может использоваться с произвольной активностью.

Другое, более правильное решение — сохранение идентификатора в месте, принадлежащем BookFragment (вместо хранения его в личном пространстве BookActivity). В этом случае объект BookFragment может прочитать данные, не полагаясь на присутствие конкретного дополнения в интенте активности. Такое «место», принадлежащее фрагменту, называется *пакетом аргументов* (arguments bundle).

Присоединение аргументов к фрагменту

Чтобы присоединить пакет аргументов к фрагменту, надо вызвать 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.

Включить в BookFragment метод newInstance(UUID), который получает UUID, создает пакет аргументов, создает экземпляр фрагмента, а затем присоединяет аргументы к фрагменту.

Листинг 9.6 – Метод newInstance(UUID) (BookFragment.java)

```
public class BookFragment extends Fragment {
    private static final String ARG_BOOK_ID = "book_id";
    private Book mBook;
    private EditText mTitleField;
    private Button mDateButton;
    private CheckBox mReadedCheckbox;

    public static BookFragment newInstance(UUID bookId) {
        Bundle args = new Bundle();
        args.putSerializable(ARG_BOOK_ID, bookId);
        BookFragment fragment = new BookFragment();
        fragment.setArguments(args);
        return fragment;
    }
    ...
}
```

Теперь класс BookActivity должен вызывать BookFragment.newInstance(UUID) каждый раз, когда ему потребуется создать BookFragment. При вызове передается значение UUID, полученное из дополнения. Вернуться к классу BookActivity, в методе createFragment() получить дополнение из интента BookActivity и передать его BookFragment.newInstance(UUID).

Константу EXTRA_BOOK_ID также можно сделать закрытой, потому что ни одному другому классу не потребуется работать с этим дополнением.

Листинг 9.7 – Использование newInstance(UUID) (BookActivity.java)

```
public class BookActivity extends SingleFragmentActivity {
    private static final String EXTRA_BOOK_ID =
        "ru.rsue.android.bookdepository.book_id";
    ...
    @Override
    protected Fragment createFragment() {
        return new BookFragment();
        UUID bookId = (UUID) getIntent()
            .getSerializableExtra(EXTRA_BOOK_ID);
        return BookFragment.newInstance(bookId);
    }
}
```

Потребность в независимости не является двусторонней. Класс BookActivity должен многое знать о классе BookFragment — 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 необходимо 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 `BookFragment.onCreate(…)` заменить код упрощенного решения выборкой UUID из аргументов фрагмента.

Листинг 9.8 – Получение идентификатора книги из аргументов (BookFragment.java)

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    UUID bookId = (UUID) getActivity().getIntent()
        .getSerializableExtra(BookActivity.EXTRA_BOOK_ID);
    UUID bookId = (UUID) getArguments().getSerializable(ARG_BOOK_ID);
    mBook = BookLab.get(getActivity()).getBook(bookId);
}
```

Запустить приложение BookDepository. Оно работает точно так же, но реализует архитектуру с независимостью BookFragment.

Перезагрузка списка

Запустить приложение BookDepository, щёлкнуть на элементе списка и внести изменения в подробную информацию о книге. Эти изменения сохраняются в модели, но при возвращении к списку содержимое RecyclerView остается неизменным.

Адаптеру RecyclerView необходимо сообщить, 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 BookListFragment запускает экземпляр BookActivity, последний помещается на вершину стека. При этом экземпляр BookActivity, который до этого находился на вершине, приостанавливается и останавливается.

Когда пользователь нажимает кнопку Back для возвращения к списку, экземпляр BookActivity извлекается из стека и уничтожается. В этот момент BookListActivity запускается и продолжает выполнение (рисунок 9.4).

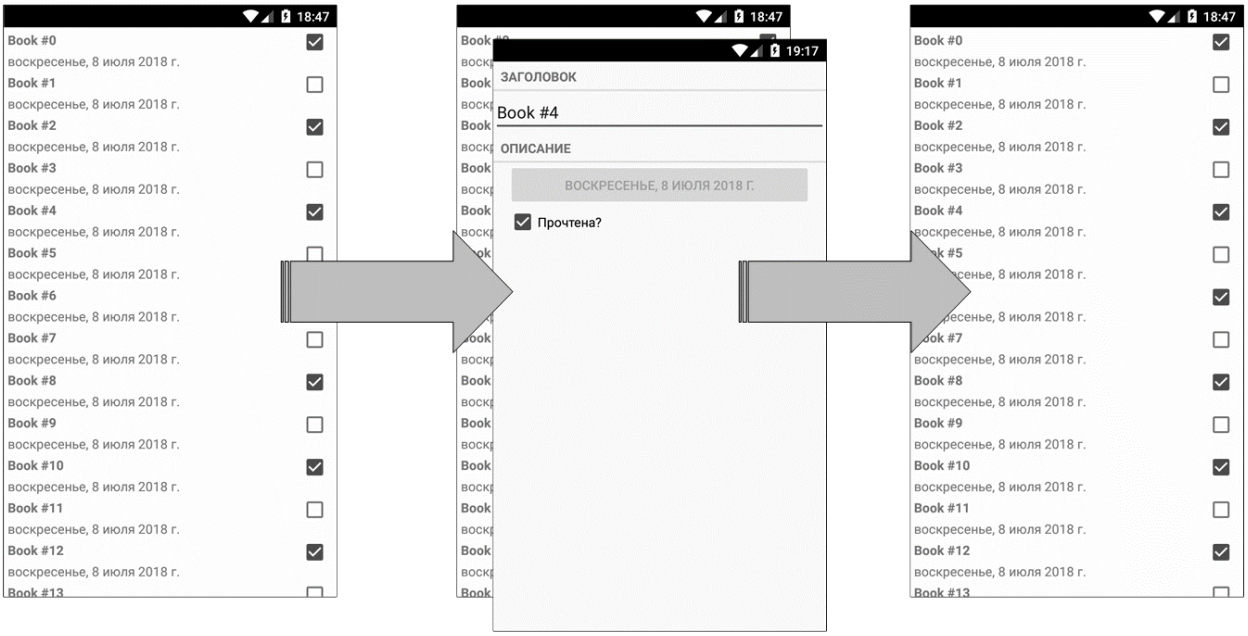

Рисунок 9.4 – Стек возврата BookDepository

Когда экземпляр BookListActivity продолжает выполнение, он получает вызов `onResume()` от ОС. При получении этого вызова BookListActivity его экземпляр FragmentManager вызывает `onResume()` для фрагментов, хостом которых в настоящее время является активность. В данном случае это единственный фрагмент BookListFragment.

В классе BookListFragment переопределить `onResume()` и инициировать вызов updateUI() для перезагрузки списка. Изменить метод `updateUI()` для вызова `notifyDataSetChanged()`, если объект BookAdapter уже создан.

Листинг 9.9 – Перезагрузка списка в onResume() (BookListFragment.java)

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    ...
}

@Override
public void onResume() {
    super.onResume();
    updateUI();
}

private void updateUI() {
    BookLab bookLab = BookLab.get(getActivity());
    List<Book> books = bookLab.getBooks();
    if (mAdapter == null) {
```

```
            mAdapter = new BookAdapter(books);
            mBookRecyclerView.setAdapter(mAdapter);
        } else {
            mAdapter.notifyDataSetChanged();
        }
    }
```

Запустить приложение BookDepository. Выбрать книгу в списке и изменить её подробную информацию. Вернувшись к списку, пользователь немедленно увидит свои изменения.

Самостоятельные задания.

Задание. Эффективная перезагрузка RecyclerView

Метод `notifyDataSetChanged` адаптера хорошо подходит, для того чтобы приказать RecyclerView перезагрузить все элементы, видимые в настоящее время.

В BookDepository этот метод неэффективен, потому что при возвращении к BookListFragment заведомо изменилось не более одного объекта Book.

Использовать 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. Для этого использовать метод `getAdapterPosition()`.

10. VIEWPAGER

В данном разделе будет создана новая активность, которая станет хостом для BookFragment. Макет активности будет состоять из экземпляра ViewPager. Включение виджета ViewPager в пользовательский интерфейс позволит «листать» элементы списка, проводя пальцем по экрану (рисунок 10.1).

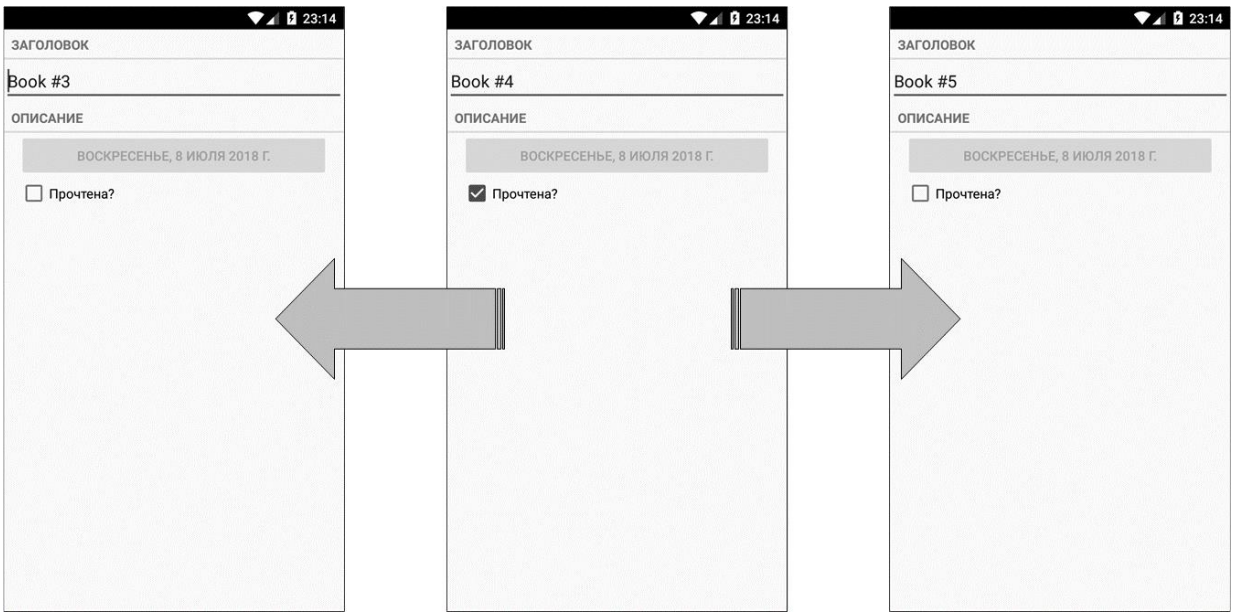

Рисунок 10.1 – Листание страниц

На рисунке 10.2 представлена обновленная диаграмма BookDepository.

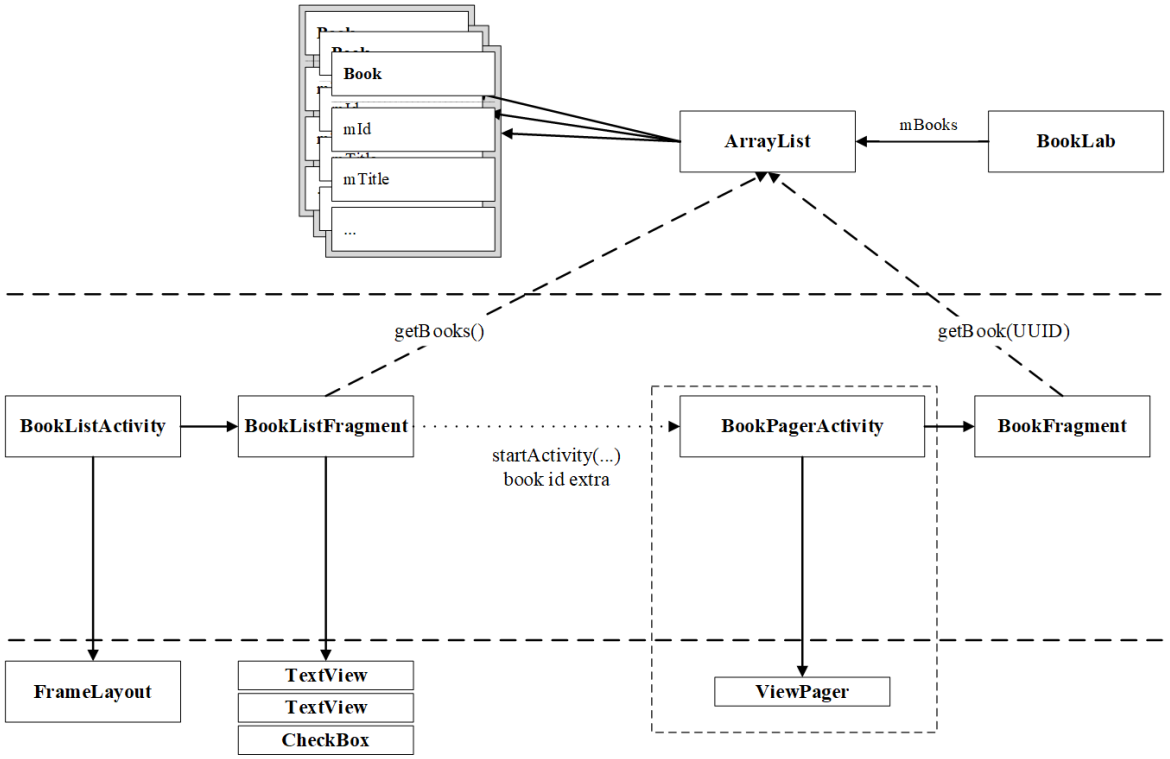

Рисунок 10.2 – Диаграмма объектов BookPagerActivity

Новая активность с именем BookPagerActivity займет место BookActivity. 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 BookDepository ничего другого менять не придется. В частности, класс BookFragment останется неизменным благодаря той работе по обеспечению независимости BookFragment, которая была проведена в предыдущем разделе.

Создание BookPagerActivity

Класс BookPagerActivity будет субклассом FragmentActivity. Он создает экземпляр и управляет ViewPager. Создайте новый класс с именем BookPagerActivity. Назначьте его суперклассом FragmentActivity и создать представление для активности.

Листинг 10.1 – Создание ViewPager (BookPagerActivity.java)

```
public class BookPagerActivity extends FragmentActivity {
      @Override
      protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_book_pager);
      }
}
```

Файл макета еще не существует. Создать новый файл макета в res/layout/ и присвойте ему имя activity_book_pager. Назначить его корневым представлением ViewPager и присвойть ему атрибуты, показанные на рисунке 10.3. Обратить внимание на необходимость использования полного имени пакета ViewPager (`android.support.v4.view.ViewPager`).

```
android.support.v4.view.ViewPager
xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/activity_book_pager_view_pager"
android:layout_width="match_parent"
android:layout_height="match_parent"
```

Рисунок 10.3 – Определение ViewPage в BookPagerActivity (activity_book_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 также необходим адаптер PagerAdapter.

Однако взаимодействие между ViewPager и PagerAdapter намного сложнее взаимодействия между RecyclerView и Adapter. Но можно 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 книг следует вернуть объект BookFragment, настроенный для вывода информации объекта в заданной позиции.

В классе BookPagerActivity Добавить следующий код для назначения PagerAdapter класса ViewPager и реализации его методов `getCount()` и `getItem(int)`.

Листинг 10.2 – Назначение PagerAdapter (BookPagerActivity.java)

```
public class BookPagerActivity extends FragmentActivity {
    private ViewPager mViewPager;
    private List<Book> mBooks;

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

        mViewPager = (ViewPager) findViewById(
                R.id.activity_book_pager_view_pager);
        mBooks = BookLab.get(this).getBooks();
        FragmentManager fragmentManager = getSupportFragmentManager();
        mViewPager.setAdapter(new
                        FragmentStatePagerAdapter(fragmentManager) {
            @Override
            public Fragment getItem(int position) {
                Book book = mBooks.get(position);
                return BookFragment.newInstance(book.getId());
            }
            @Override
            public int getCount() {
                return mBooks.size();
            }
        });
    }
}
```

После поиска ViewPager в представлении активности получается от BookLab набор данных — контейнер List объектов Book. Затем получается экземпляр FragmentManager для активности.

На следующем шаге адаптером назначается безымянный экземпляр FragmentStatePagerAdapter. Для создания FragmentStatePagerAdapter необходим объект FragmentManager. FragmentStatePagerAdapter — 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. Для этого необходим экземпляр FragmentManager.

Интеграция контроллера

Теперь можно переходить к устранению класса BookActivity и замене его классом BookPagerActivity.

Начать нужно с добавления метода newIntent в BookPagerActivity вместе с дополнением для идентификатора книги.

Листинг 10.3 – Создание newIntent (BookPagerActivity.java)

```
public class BookPagerActivity extends FragmentActivity {
    private static final String EXTRA_BOOK_ID =
            "ru.rsue.android.bookdepository.book_id";

    private ViewPager mViewPager;
    private List<Book> mBooks;

    public static Intent newIntent(Context packageContext, UUID bookId)
    {
        Intent intent = new Intent(packageContext,
                BookPagerActivity.class);
        intent.putExtra(EXTRA_BOOK_ID, bookId);
        return intent;
    }
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_book_pager);
        UUID bookId = (UUID) getIntent()
                .getSerializableExtra(EXTRA_BOOK_ID);
        ...
    }
}
```

Теперь нужно сделать так, чтобы при выборе элемента списка в BookListFragment запускался экземпляр BookPagerActivity вместо BookActivity.

Вернуться к файлу BookListFragment.java и изменить метод BookHolder.onClick(…), чтобы он запускал BookPagerActivity.

Листинг 10.4 – Запуск активности (BookListFragment.java)

```
private class BookHolder extends RecyclerView.ViewHolder
implements View.OnClickListener {
    ...
    @Override
    public void onClick(View v) {
        Intent intent = BookActivity.newIntent(getActivity(),
                        mBook.getId());
        Intent intent = BookPagerActivity.newIntent(getActivity(),
                        mBook.getId());
        startActivity(intent);
    }
}
```

Необходимо добавить BookPagerActivity в манифест, чтобы ОС могла запустить эту активность. Для этого достаточно заменить в манифесте BookActivity на BookPagerActivity.

Листинг 10.5 – Добавление BookPagerActivity в манифест (AndroidManifest.xml)

```
<application ...>
    ...
    <activity
        android:name=".BookActivity"
        android:name=".BookPagerActivity"
        android:label="@string/app_name" >
    </activity>
    ...
</application>
```

Наконец, чтобы не загромождать проект, удалить BookActivity.java в окне инструментов Project.

Запустить приложение BookDepository. Нажать на строке Book #0, чтобы просмотреть подробную информацию. Провести по экрану влево или вправо, чтобы просмотреть другие элементы списка. 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)`.

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 `BookPagerActivity.onCreate(…)` найти индекс отображаемой книги; для этого перебрать и проверить идентификаторы всех книг. Когда будет найден экземпляр Book, у которого поле mId совпадает с bookId в дополнении интента, изменить текущий элемент по индексу найденного объекта Book.

Листинг 10.6 – Назначение исходного элемента (BookPagerActivity.java)

```
public class BookPagerActivity extends FragmentActivity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
      ...
      FragmentManager fragmentManager = getSupportFragmentManager();
      mViewPager.setAdapter(new
               FragmentStatePagerAdapter(fragmentManager) {
         ...
      });
      for (int i = 0; i < mBooks.size(); i++) {
            if (mBooks.get(i).getId().equals(bookId)) {
            mViewPager.setCurrentItem(i);
            break;
         }
      }
   }
}
```

Запустить приложение BookDepository. При выборе любого элемента списка должна отображаться подробная информация правильного объекта Book. Теперь экземпляр ViewPager полностью готов к работе.

11. ДИАЛОГОВЫЕ ОКНА

В данном разделе будет добавлено диалоговое окно, в котором пользователь может изменить дату начала прочтения книги. Диалоговые окна требуют от пользователя внимания и ввода данных. Обычно они используются для принятия решений или отображения важной информации.

При нажатии кнопки даты в BookFragment будет открываться диалоговое окно, показанное на рисунке 11.1.

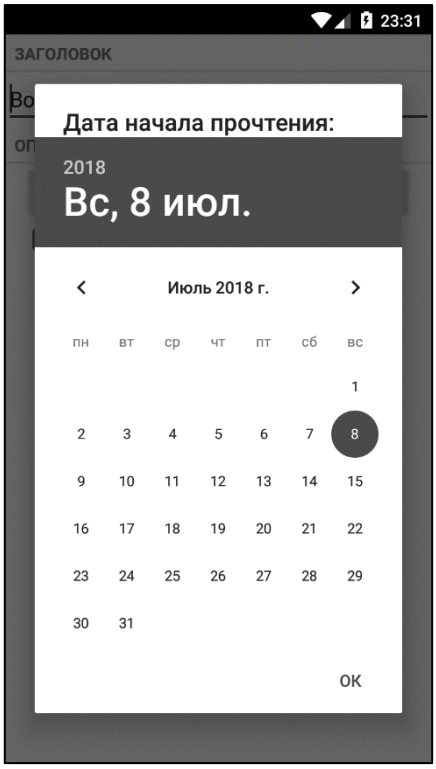

Рисунок 11.1 – Диалоговое окно для выбора даты

Диалоговое окно на рисунке 11.1 является экземпляром **AlertDialog** — субкласса **Dialog**. Именно этот многоцелевой субкласс Dialog будет чаще всего использован в программах.

Начиная с версии Lollipop, диалоговые окна прошли визуальную переработку. AlertDialog в Lollipop и выше автоматически используют новый стиль. В более ранних версиях Android окна AlertDialog возвращаются к старому стилю. Для того чтобы диалоговые окна всегда отображались в новом стиле независимо от версии Android на устройстве пользователя, необходимо использовать библиотеку **AppCompat**.

AppCompat — библиотека совместимости, разработанная компанией Google, которая реализует некоторые возможности новых версий Android на старых устройствах. В приложении библиотека AppCompat будет использована для создания стабильного оформления диалоговых окон во всех поддерживаемых версиях Android.

Библиотека AppCompat

Чтобы использовать библиотеку AppCompat, сначала необходимо добавить ее в число зависимостей. Возможно, она там уже есть; это зависит от того, как создавался проект.

Открыть окно Project Structure (File→Project Structure...), выбрать модуль app и щёлкнуть на вкладке Dependencies. Если библиотека AppCompat не входит в список, добавить ее — щёлкнуть на кнопке + и выбрать зависимость appcompat-v7 в списке (рисунок 11.2).

Рисунок 11.2 – Выбор зависимости AppCompat

Библиотека AppCompat содержит собственный класс AlertDialog, который будет использоваться в приложении. Эта версия AlertDialog очень похожа на ту, которая включена в ОС Android. Чтобы использовать именно ту версию, которая нужна, необходимо импортировать правильную версию AlertDialog. В приложении будет использоваться `android.support.v7.app.AlertDialog`.

Использование DialogFragment

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 BookDepository будет создан субкласс DialogFragment с именем **DatePickerFragment**. В коде DatePickerFragment создается и

настраивается экземпляр AlertDialog, отображающий виджет DatePicker. В качестве хоста DatePickerFragment используется экземпляр BookPagerActivity.

На рисунке 11.3 изображена схема этих отношений.

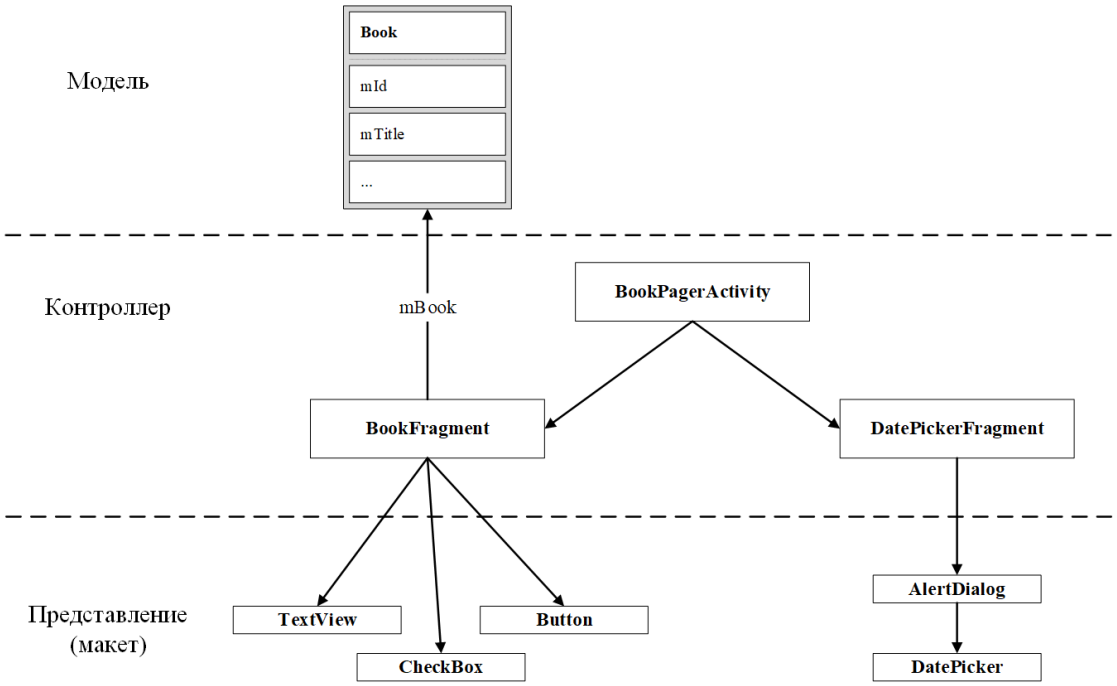

Рисунок 11.3 – Диаграмма объектов для двух фрагментов с хостом BookPagerActivity

В этом разделе предстоит реализовать следующие задачи:

- создать класс DatePickerFragment;
- построить AlertDialog;
- вывести диалоговое окно на экран с использованием FragmentManager.

Перед началом работы, добавить строковый ресурс (листинг 11.1).

Листинг 11.1 – Добавление строки заголовка диалогового окна (values/strings.xml)

```
<resources>
    ...
    <string name="book_readed_label">Прочтена?</string>
    <string name="date_picker_title">Дата начала прочтения:</string>
</resources>
```

Создать новый класс с именем **DatePickerFragment** и назначить его суперклассом **DialogFragment**. Обязательно выбрать 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.

Добавить в файл DatePickerFragment.java реализацию `onCreateDialog(…)`, которая создает AlertDialog с заголовком и одной кнопкой OK. (Виджет DatePicker будет добавлен позднее.)

Проследите за тем, чтобы импортировалась версия AlertDialog из AppCompat: `android.support.v7.app.AlertDialog`.

Листинг 11.2 – 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 передается 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 11.2 передается константа Android для кнопки OK и null вместо слушателя. Слушатель будет реализован позднее.

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.

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 разработчика: если передать FragmentTransaction, за создание и закрепление транзакции отвечает разработчик. При передаче FragmentManager транзакция автоматически создается и закрепляется для разработчика.

Добавить в BookFragment константу для метки DatePickerFragment. Затем в методе `onCreateView(…)` удалить код, блокирующий кнопку даты, и назначить слушателя `View.OnClickListener`, который отображает DatePickerFragment при нажатии кнопки даты.

Листинг 11.3 – Отображение DialogFragment (BookFragment.java)

```
public class BookFragment extends Fragment {
   private static final String ARG_BOOK_ID = "book_id";
   private static final String DIALOG_DATE = "DialogDate";
   ...
   @Override
   public View onCreateView(LayoutInflater inflater, ViewGroup
               container, Bundle savedInstanceState) {
      ...
      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);
         }
      });
      mReadedCheckBox = (CheckBox) v.findViewById(R.id.book_readed);
      ...
      return v;
```

```
}
...
```

Запустить приложение BookDepository и нажать кнопку даты, чтобы диалоговое окно появилось на экране (рисунок 11.4).

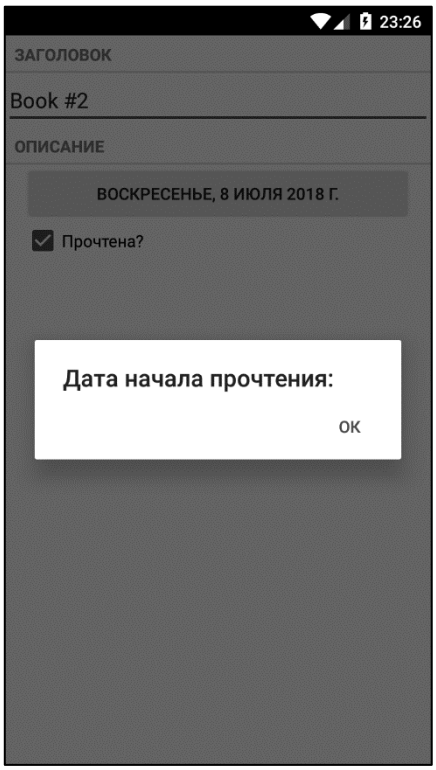

Рисунок 11.4 – AlertDialog с заголовком и кнопкой

Далее будет включен 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 создать новый файл макета с именем dialog_date.xml и назначить его корневым элементом DatePicker. Макет будет состоять из одного объекта View (DatePicker), который будет заполнен и передан `setView(…)`.

Настройть макет DatePicker так, как показано на рисунке 11.5.

DatePicker

```
xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/dialog_date_date_picker"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:calendarViewShown="false"
```

Рисунок 11.5 – Макет DatePicker (layout/dialog_date.xml)

В методе `DatePickerFragment.onCreateDialog(…)` заполнить представление и назначить его диалоговому окну.

Листинг 11.4 – 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();
}
```

Запустить приложение BookDepository. Нажать кнопку даты и убедиться в том, что в диалоговом окне теперь отображается DatePicker. На устройствах с версией Lollipop и выше отображается календарный виджет (рисунок 11.6).

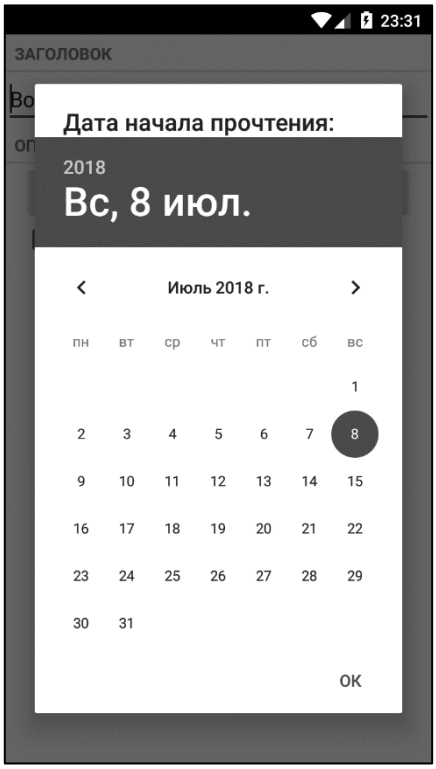

Рисунок 11.6 – DatePicker в Lollipop

Передача данных между фрагментами

В предыдущих разделах осуществлялась передача данных между двумя активностями, между двумя фрагментными активностями, а теперь необходимо передать данные между двумя фрагментами, хостом которых

является одна активность, — BookFragment и DatePickerFragment (рисунок 11.7).

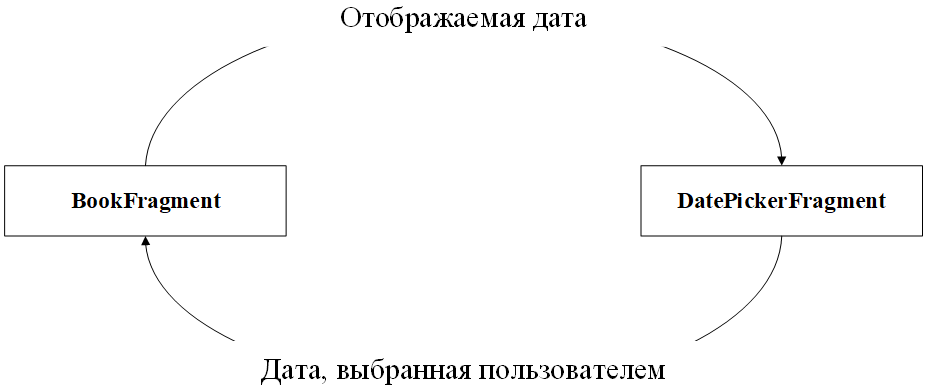

Рисунок 11.7 – Взаимодействие между BookFragment и DatePickerFragment

Чтобы передать дату начала прочтения книги DatePickerFragment, будет создан метод `newInstance(Date)` и объект Date сделан аргументом фрагмента.

Чтобы вернуть новую дату фрагменту BookFragment для обновления уровня модели и его собственного представления, необходимо упаковать ее как дополнение объекта Intent и передать этот объект Intent в вызове `BookFragment.onActivityResult(…)` (рисунок 11.8).

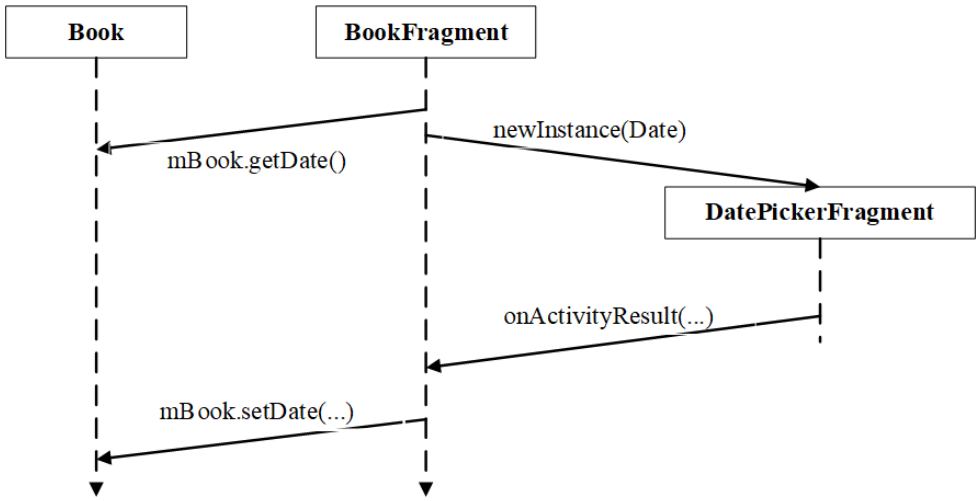

Рисунок 11.8 – Последовательность событий взаимодействия между BookFragment и DatePickerFragment

Передача данных DatePickerFragment

Чтобы получить данные в DatePickerFragment, дата будет сохраняться 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. Добавить в файл DatePickerFragment.java метод `newInstance(Date)`.

Листинг 11.5 – 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 BookFragment удалить вызов конструктора DatePickerFragment и заменить его вызовом `DatePickerFragment.newInstance(Date)`.

Листинг 11.6 – Добавление вызова newInstance() (BookFragment.java)

```
    @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 = new DatePickerFragment();
                DatePickerFragment dialog = DatePickerFragment
                        .newInstance(mBook.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 можно получить нужную информацию из Calendar.

В методе `onCreateDialog(…)` получить объект Date из аргументов и использовать его с Calendar для инициализации DatePicker.

Листинг 11.7 – 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_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 BookFragment успешно сообщает DatePickerFragment, какую дату следует отобразить. Запустить приложение BookDepository и убедиться в том, что все работает так же, как прежде.

Возвращение данных BookFragment

Чтобы экземпляр BookFragment получал данные от DatePickerFragment, необходимо каким-то образом отслеживать отношения между двумя фрагментами.

С активностями вызывается `startActivityForResult(…)`, а ActivityManager отслеживает отношения между родительской и дочерней активностью. Когда дочерняя активность прекращает существование, ActivityManager знает, какая активность должна получить результат.

Для создания аналогичной связи можно назначить BookFragment *целевым фрагментом* (target fragment) для DatePickerFragment. Эта связь будет автоматически восстановлена после того, как и BookFragment, и

DatePickerFragment уничтожаются и заново создаются 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, надо вызвать `getTargetFragment()` и `getTargetRequestCode()` для фрагмента, назначившего целевой фрагмент.

В файле BookFragment.java создать константу для кода запроса, а затем назначьте BookFragment целевым фрагментом экземпляра DatePickerFragment.

Листинг 11.8 – Назначение целевого фрагмента (BookFragment.java)

```
public class BookFragment extends Fragment {
   private static final String ARG_BOOK_ID = "book_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(mBook.getDate());
              dialog.setTargetFragment(BookFragment.this,
                           REQUEST_DATE);
              dialog.show(manager, DIALOG_DATE);
         }
      });
      mReadedCheckBox = (CheckBox) v.findViewById(R.id.book_readed);
      ...
      return v;
   }
   ...
}
```

Передача данных целевому фрагменту

Итак, связь между BookFragment и DatePickerFragment создана, и теперь нужно вернуть дату BookFragment. Дата будет включена в объект Intent как дополнение.

В классе DatePickerFragment создать закрытый метод, который создает интент, помещает в него дату как дополнение, а затем вызывает `BookFragment.onActivityResult(…)`.

Листинг 11.9 – Обратный вызов целевого фрагмента (DatePickerFragment.java)

```
public class DatePickerFragment extends DialogFragment {
    public static final String EXTRA_DATE =
            "ru.rsue.android.bookdepository.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 BookFragment. В коде `onCreateDialog(…)` заменить параметр null вызова `setPositiveButton(…)` реализацией `DialogInterface.OnClickListener`, которая возвращает выбранную дату и вызывает `sendResult`.

Листинг 11.10 – 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 BookFragment переопределить метод `onActivityResult(…)`, чтобы он возвращал дополнение, задавал дату в Book и обновлял текст кнопки даты.

Листинг 11.11 – Реакция на получение данных от диалогового окна (BookFragment.java)

```
public class BookFragment 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
      );
                mBook.setDate(date);
                mDateButton.setText(mBook.getDate().toString());
            }
    }
}
```

Код, задающий текст кнопки, идентичен коду из `onCreateView(…)`. Чтобы избежать задания текста в двух местах, необходимо инкапсулировать этот код в закрытом методе `updateDate()`, а затем вызвать его в `onCreateView(…)` и `onActivityResult(…)`.

Это можно сделать вручную или поручить работу Android Studio. Выделить всю строку кода, которая задает текст mDateButton, щёлкнуть на ней правой кнопкой мыши и выбрать команду Refactor→Extract→Method... (рисунок 11.9).

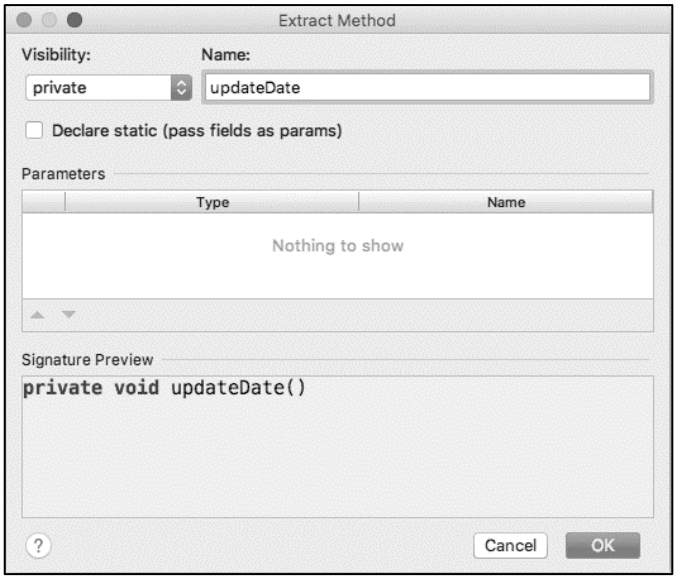

Рисунок 11.9 – Извлечение метода в Android Studio

Выбрать закрытый уровень видимости метода и ввести имя **updateDate**. Щёлкнуть на кнопке OK; среда Android Studio сообщает, что ей удалось найти еще одно место, в котором использовалась эта строка кода. Щёлкнуть на кнопке Yes, чтобы разрешить Android Studio обновить второе вхождение. Убедиться в том, что код был выделен в метод updateDate (листинг 11.12).

Листинг 11.12 – Выделение кода в метод updateDate() (BookFragment.java)

```
public class BookFragment extends Fragment {
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
                container, Bundle savedInstanceState) {
       View v = inflater.inflate(R.layout.fragment_book, container,
                false);
       ...
       mDateButton = (Button) v.findViewById(R.id.book_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);
             mBook.setDate(date);
             updateDate();
        }
    }
    private void updateDate() {
        mDateButton.setText(mBook.getDate().toString());
    }
}
```

Запустить приложение BookDepository и убедиться в том, что пользователь действительно может управлять датой. Изменить дату Book; новая дата должна появиться в представлении BookFragment. Вернуться к списку книг, проверить дату Book и убедиться в том, что уровень модели действительно обновлен.

Самостоятельные задания.

Задание 1. Новые диалоговые окна

Написать еще один диалоговый фрагмент TimePickerFragment для выбора времени начала прочтения книги. Использовать виджет TimePicker, Добавить в BookFragment еще одну кнопку для отображения TimePickerFragment.

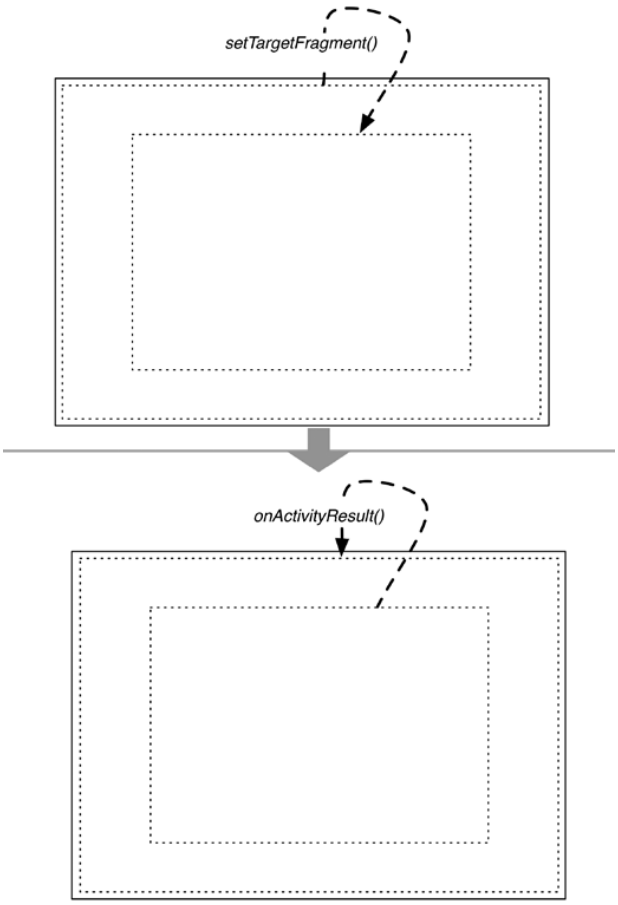

Рисунок 11.10 – Взаимодействие между фрагментами на планшетах

Задание 2. DialogFragment

Необходимо измененить представление DatePickerFragment.

Первая часть — представить представление DatePickerFragment с переопределением onCreateView вместо onCreateDialog. При таком способе создания DialogFragment не будет отображаться со встроенными областями заголовка и кнопок в верхней и нижней части диалогового окна. Необходимо самостоятельно создать кнопку OK в dialog_date.xml.

После того как представление DatePickerFragment будет создано в onCreateView, можно отобразить фрагмент DatePickerFragment как диалоговое окно или встроить его в активность.

Во второй части — создать новый субкласс SingleFragmentActivity и сделать эту активность хостом для DatePickerFragment. При таком представлении DatePickerFragment будет использоваться механизм startActivityForResult для возвращения даты BookFragment. В DatePickerFragment, если целевой фрагмент не существует, использовать метод `setResult(int, intent)` активности-хоста для возвращения даты фрагменту.

12. ПАНЕЛЬ ИНСТРУМЕНТОВ

В данном разделе для приложения BookDepository будет создано меню, которое станет отображаться на панели инструментов. В этом меню будет присутствовать элемент действия (action item) для добавления новой книги. Также будет обеспечена работа кнопки Up на панели инструментов (рисунок 12.1).

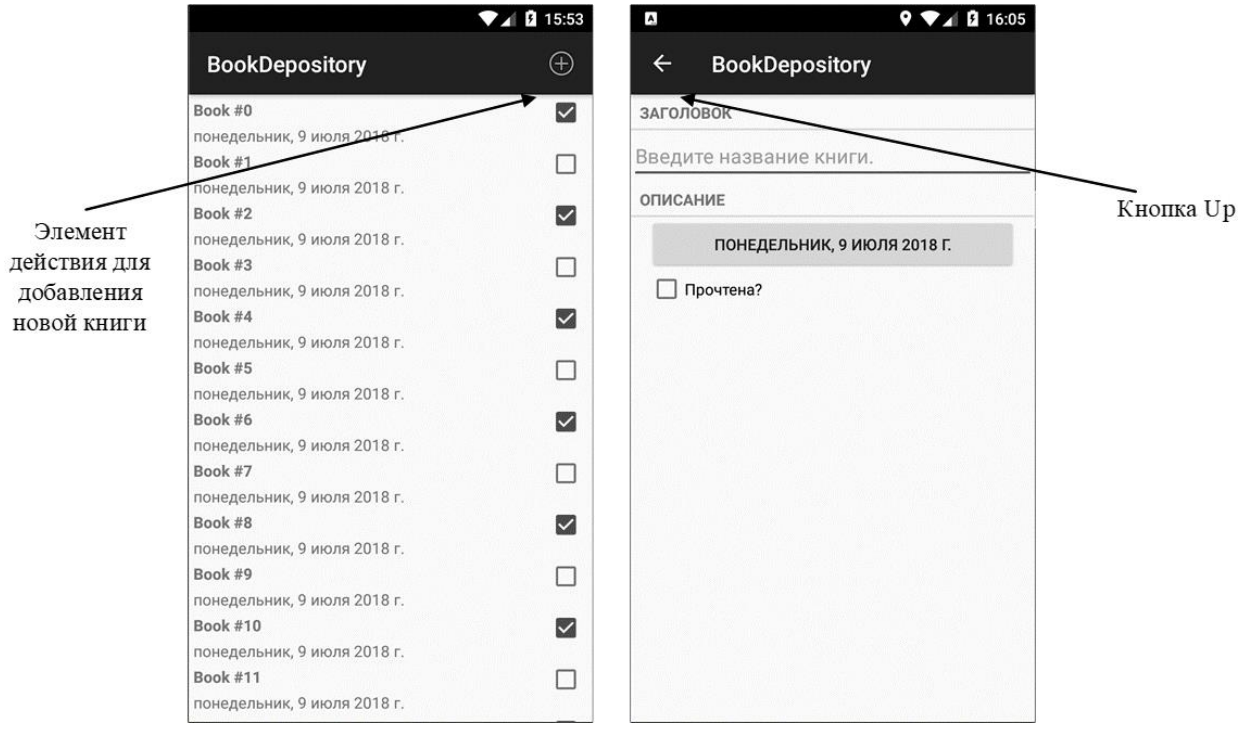

Рисунок 12.1 – Панель инструментов BookDepository

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.

Использование библиотеки AppCompat

В предыдущем разделе в приложение BookDepository была добавлена зависимость для библиотеки AppCompat. Полная интеграция с библиотекой AppCompat требует ряда дополнительных шагов. Возможно, некоторые из этих действий уже выполнены.

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 — убедиться в том, что используется одна 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. Следует обратить внимание на атрибут `android:theme`. Он выглядит примерно так, как показано в листинге 12.1.

Листинг 12.1 – Стандартный манифест (AndroidManifest.xml)

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

AppTheme определяется в файле res/values/styles.xml. В зависимости от того, как создавался исходный проект, он может содержать несколько версий AppTheme в нескольких файлах styles.xml. Надо убедиться в том, что атрибут parent элемента AppTheme соответствует выделенной части в листинге 12.2.

Листинг 12.2 – Использование темы AppCompat (res/values/styles.xml)

```
<resources>
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    </style>
</resources>
```

Использование AppCompatActivity

Последним шагом становится преобразование всех активностей BookDepository в субклассы AppCompatActivity. До настоящего момента все активности были субклассами FragmentActivity, что позволяло использовать реализацию фрагментов из библиотеки поддержки.

Класс AppCompatActivity сам является субклассом FragmentActivity. Это означает, что по-прежнему можно использовать поддержку фрагментов в AppCompatActivity, что упрощает необходимые изменения в BookDepository.

Преобразовать SingleFragmentActivity и BookPagerActivity в субклассы AppCompatActivity.

Листинг 12.3 – Преобразование в AppCompatActivity (SingleFragmentActivity.java)

```
public abstract class SingleFragmentActivity extends FragmentActivity
        AppCompatActivity {
    ...
}
```

Листинг 12.4 – Преобразование в AppCompatActivity (BookPagerActivity.java)

```
public class BookPagerActivity extends FragmentActivity
        AppCompatActivity {
    ...
}
```

Запустить BookDepository и убедиться в том, что запуск происходит без сбоев. Приложение должно выглядеть примерно так, как показано на рисунке 12.2.

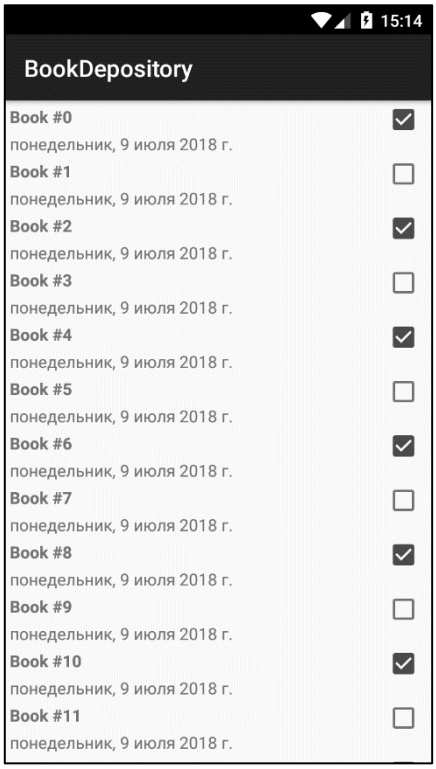

Рисунок 12.2 – Новая панель инструментов

Теперь, когда в приложении BookDepository используется панель инструментов AppCompat, 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. Далее будет добавлен элемент действия, при помощи которого пользователь сможет создать описание новой книги.

Для работы меню потребуется несколько строковых ресурсов. Добавить их в файл strings.xml (листинг 12.5). Пока эти строки выглядят довольно загадочно, но лучше решить эту проблему сразу. Когда они позднее понадобятся, они уже будут на своем месте.

Листинг 12.5 – Добавление строк для меню (res/values/strings.xml)

```
<resources>
    ...
    <string name="date_picker_title">Дата начала прочтения:</string>
    <string name="new_book">Новая книга</string>
    <string name="show_subtitle">Показать подзаголовок</string>
    <string name="hide_subtitle">Скрыть подзаголовок</string>
    <string name="subtitle_format">%1$d книг</string>
</resources>
```

Определение меню в XML

Меню определяются такими же ресурсами, как и макеты. Создается описание меню в XML и файл помещается в каталог res/menu проекта. Android генерирует идентификатор ресурса для файла меню, который затем используется для заполнения меню в коде.

В окне инструментов Project щёлкнуть правой кнопкой мыши на каталоге res и выбрать команду New→Android resource File. Выбрать тип ресурса Menu, присвойте ресурсу меню имя fragment_book_list и щёлкнуть на кнопке OK. Android Studio сгенерирует файл res/menu/fragment_book_list.xml (рисунок 12.3).

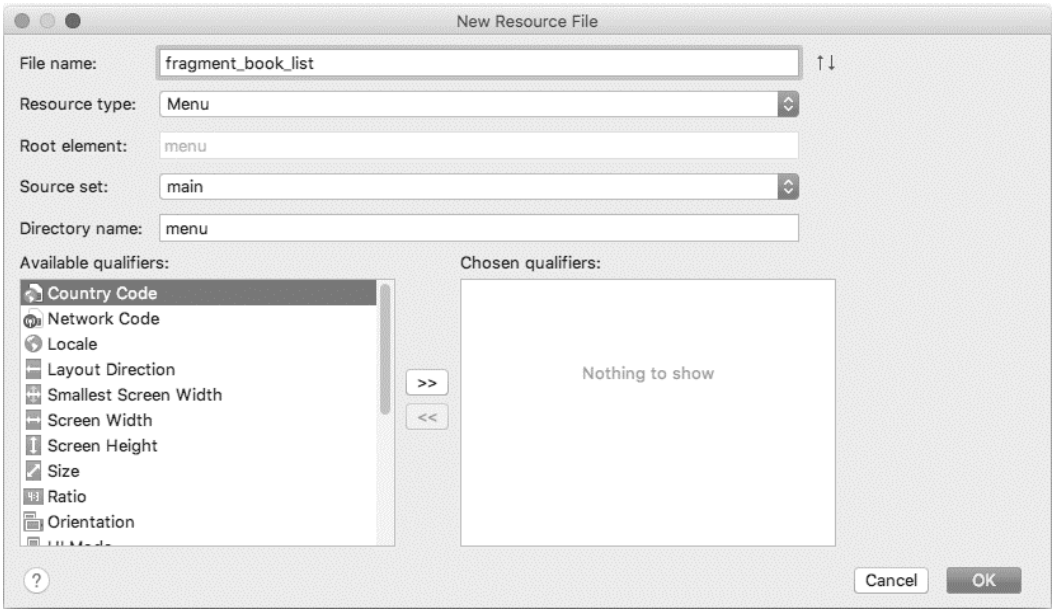

Рисунок 12.3 – Создание файла меню

Добавить в новый файл fragment_book_list.xml элемент item (листинг 12.6).

Листинг 12.6 – Создание ресурса меню BookListFragment (fragment_book_list.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<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_new_book"
        android:icon="@android:drawable/ic_menu_add"
        android:title="@string/new_book"
        app:showAsAction="ifRoom|withText"/>
</menu>
```

Атрибут showAsAction устанавливает, должна ли команда меню отображаться на самой панели инструментов или в *дополнительном меню* (overflow menu). В данном случае два значения объединиены, 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.

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 — найти системные значки, соответствующие потребностям приложения, и скопировать их прямо в графические ресурсы проекта.

Системные значки находятся в каталоге Android SDK. Для получения местонахождения SDK можно открыть окно Project Structure и выбрать категорию SDK Location. В каталоге SDK можно найти разные ресурсы Android, включая ic_menu_add. Эти ресурсы находятся в каталоге /platforms/android-26/data/res, где 26 — уровень API Android-версии.

Самый простой вариант — использовать программу **Android Asset Studio**, включенную в Android Studio. Asset Studio позволяет создать и настроить изображение для использования на панели инструментов.

Щёлкнуть правой кнопкой мыши на каталоге drawable в окне инструментов Project и выбрать команду New→Image Asset. На экране появляется Asset Studio (рисунок 12.4).

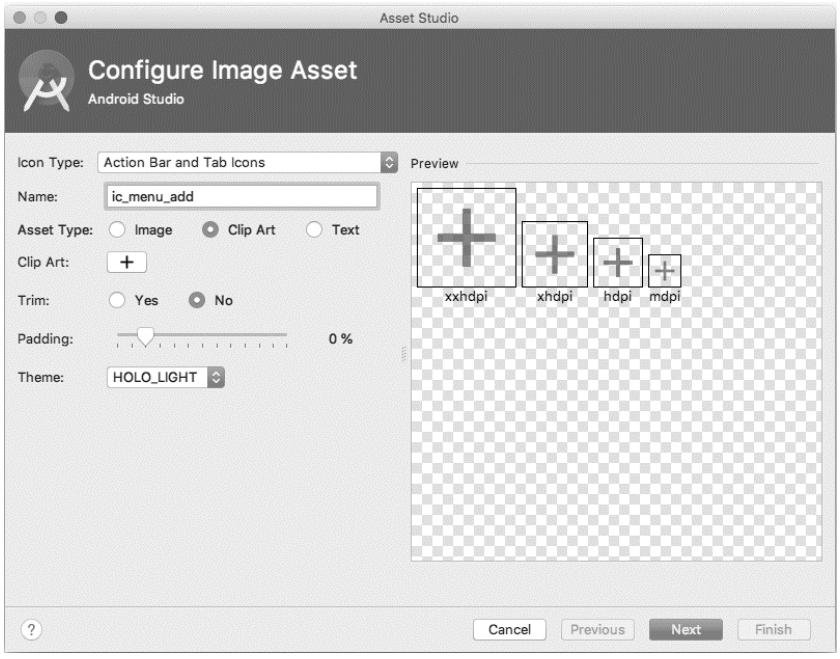

Рисунок 12.4 – Asset Studio

В Asset Studio можно генерировать значки нескольких разных типов. Выбрать в поле Asset Type: вариант Action Bar and Tab Icons. Затем в группе Foreground выбрать переключатель Clipart и нажать кнопку Choose, чтобы выбрать графическую заготовку.

В открывшемся окне выбрать изображение, напоминающее знак + (рисунок 12.5).

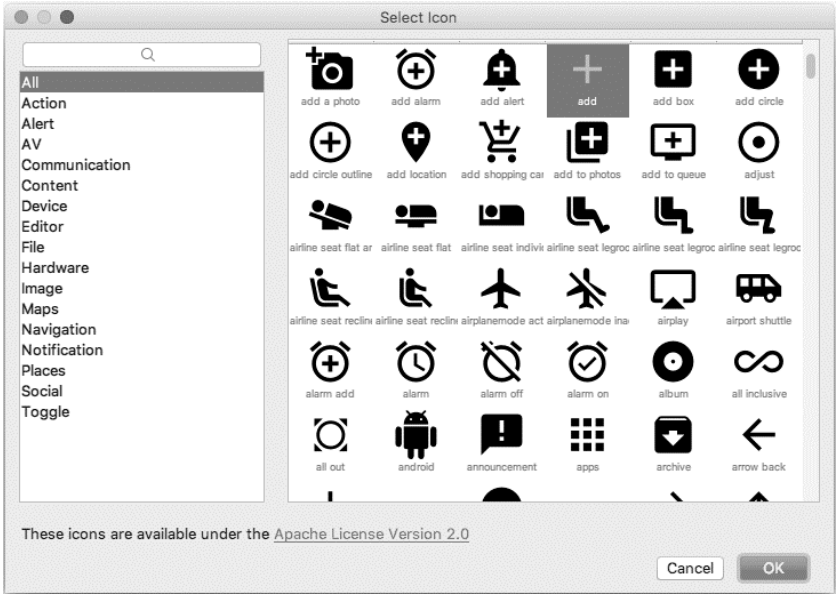

Рисунок 12.5 – Галерея графических заготовок

Наконец, ввести имя ic_menu_add и нажать кнопку Next (рисунок 12.6).

Asset Studio предлагает выбрать модуль и каталог для добавления изображения. Оставить значения по умолчанию, чтобы добавить изображение в модуль app. В окне также выводится план работы, которую выполнит Asset Studio. Значки mdpi, hdpi, xhdpi и xxhdpi будут созданы автоматически.

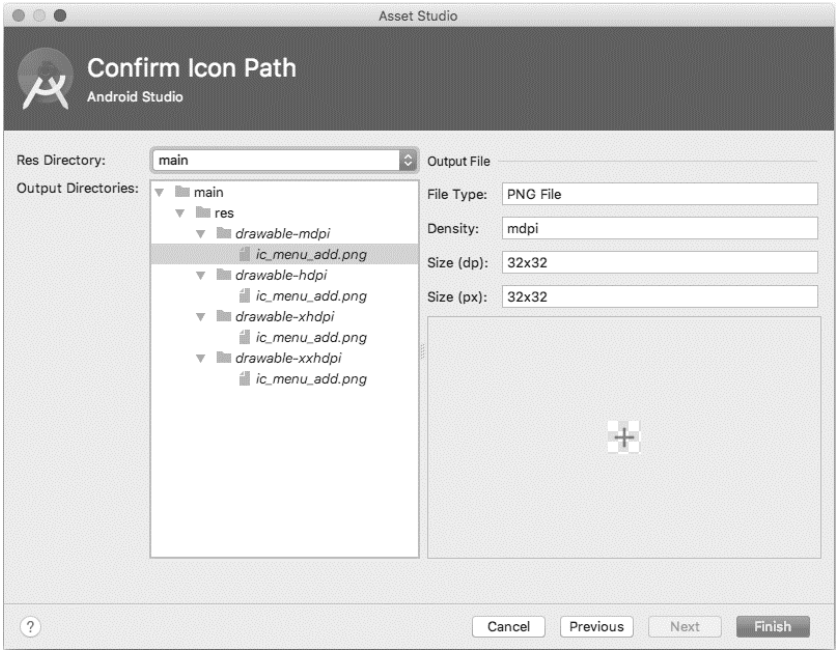

Рисунок 12.6 – Файлы, сгенерированные в Asset Studio

Щёлкнуть на кнопке **Finish**, чтобы сгенерировать изображения.

Затем в файле макета изменить атрибут icon и включить в него ссылку на новый ресурс из данного проекта.

Листинг 12.7 – Ссылка на локальный ресурс (menu/fragment_book_list.xml)

```
<item
    android:id="@+id/menu_item_new_book"
    android:icon="@android:drawable/ic_menu_add"
    android:icon="@drawable/ic_menu_add"
    android:title="@string/new_book"
    app:showAsAction="ifRoom|withText"/>
```

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 приложения требует, чтобы реализация находилась в фрагменте, а не в активности. Класс Fragment содержит собственный набор методов обратного вызова для командных меню, которые будут реализованы в BookListFragment. Для создания меню и обработки выбранных команд используются следующие методы:

```
public void onCreateOptionsMenu(Menu menu,
        MenuInflater inflater)
public boolean onOptionsItemSelected(MenuItem item)
```

В файле BookListFragment.java переопределить метод `onCreateOptionsMenu(Menu, MenuInflater)` так, чтобы он заполнял меню, определенное в файле fragment_book_list.xml.

Листинг 12.8 – Заполнение ресурса меню (BookListFragment.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_book_list, menu);
}
```

В этом методе вызывается метод MenuInflater.inflate(int, Menu) и передается идентификатор ресурса файла меню. Вызов заполняет экземпляр Menu командами, определенными в файле.

FragmentManager отвечает за вызов `Fragment.onCreateOptionsMenu(Menu, MenuInflater)` при получении активностью обратного вызова `onCreateOptionsMenu(…)` от ОС. Поэтому необходимо явно указать FragmentManager, что фрагмент должен получить вызов `onCreateOptionsMenu(…)`. Для этого вызывается следующий метод:

```
public void setHasOptionsMenu(boolean hasMenu)
```

В методе `BookListFragment.onCreate(…)` следует сообщить FragmentManager, что экземпляр BookListFragment должен получать обратные вызовы командного меню.

Листинг 12.9 – Получение обратных вызовов (BookListFragment.java)

```
...
private RecyclerView mBookRecyclerView;
private BookAdapter mAdapter;

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

@Override
public View onCreateView(LayoutInflater inflater, ...){
    ...
}
```

В приложении BookDepository появляется командное меню (рисунок 12.7). У большинства телефонов в книжной ориентации хватает места только для значка. Текст команды открывается долгим нажатием на значке на панели инструментов (рисунок 12.8).

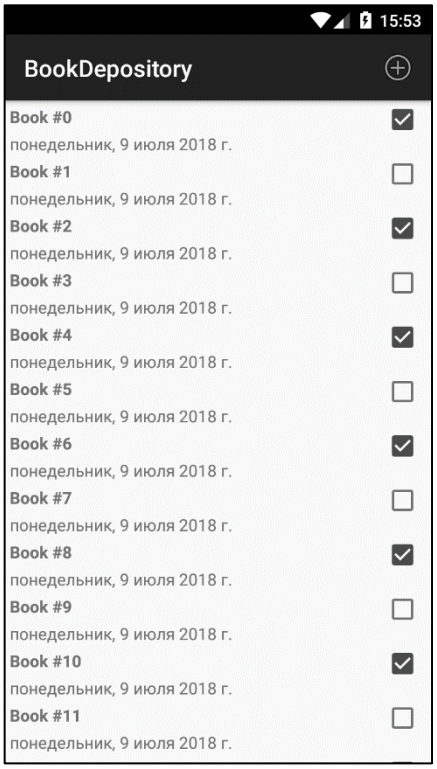

Рисунок 12.7 – Значок команды меню на панели инструментов

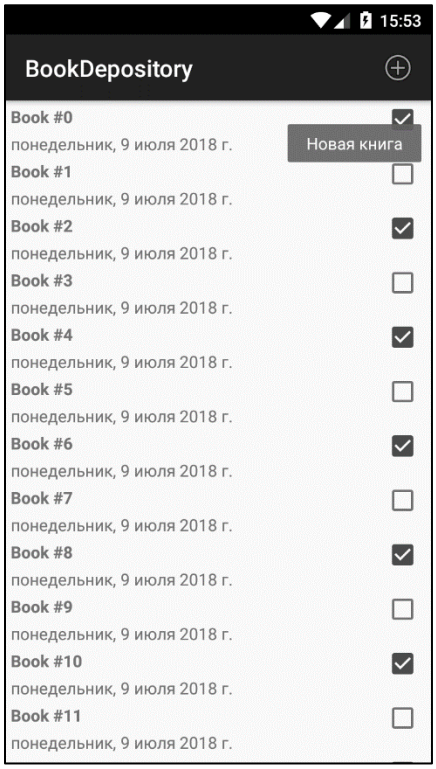

Рисунок 12.8 – Долгое нажатие на значке на панели инструментов выводит текст команды

В альбомной ориентации на панели инструментов хватает места как для значка, так и для текста (рисунок 12.9).

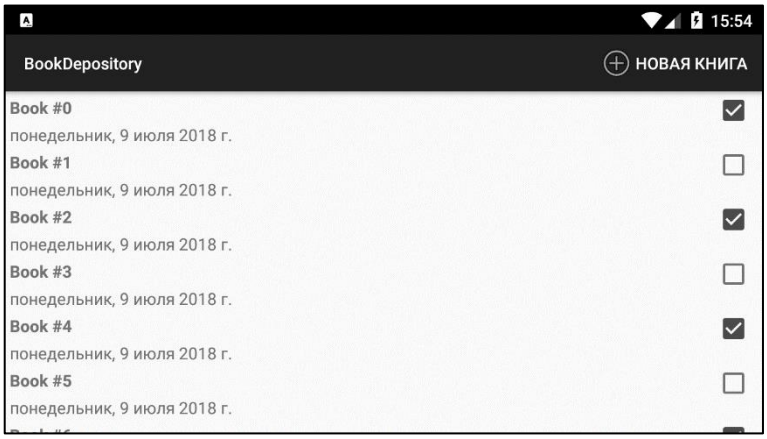

Рисунок 12.9 – Значок и текст на панели инструментов

Реакция на выбор команд

Чтобы отреагировать на выбор пользователем команды *New Book*, понадобится механизм добавления нового объекта Book в список. Включить в файл BookLab.java следующий метод.

Листинг 12.10 – Добавление нового объекта Book (BookLab.java)

```
...
public void addBook(Book b) {
    mBooks.add(b);
}
public List<Book> getBooks() {
    return mBooks;
}
...
```

Теперь, когда есть возможность вводить описания книг самостоятельно, программное генерирование 100 объектов становится лишним. В файле BookLab.java удалить код, генерирующий эти книги.

Листинг 12.11 – Долой случайные книги! (BookLab.java)

```
private BookLab(Context context) {
    mBooks = new ArrayList<>();
    for (int i = 0; i < 100; i++) {
        Book book = new Book();
        book.setTitle("Book #" + i);
        book.setReaded(i % 2 == 0);
        mBooks.add(book);
    }
}
```

Когда пользователь выбирает команду в командном меню, фрагмент получает обратный вызов метода `onOptionsItemSelected(MenuItem)`. Этот метод получает экземпляр MenuItem, описывающий выбор пользователя.

И хотя сознанное меню состоит всего из одной команды, в реальных меню их обычно больше. Чтобы определить, какая команда меню была выбрана, надо проверить идентификатор команды меню и отреагировать соответствующим образом. Этот идентификатор соответствует идентификатору, назначенному команде в файле меню.

В файле BookListFragment.java реализуйте метод `onOptionsItemSelected(MenuItem)`, реагирующий на выбор команды меню. Реализация создает новый объект Book, добавляет его в BookLab и запускает экземпляр BookPagerActivity для редактирования нового объекта Book.

Листинг 12.12 – Реакция на выбор команды меню (BookListFragment.java)

```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    ...
}
@Override
public boolean onOptionsItemSelected(MenuItem item) {
```

```
    switch (item.getItemId()) {
        case R.id.menu_item_new_book:
            Book book = new Book();
            BookLab.get(getActivity()).addBook(book);
            Intent intent = BookPagerActivity
                .newIntent(getActivity(), book.getId());
            startActivity(intent);
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Метод возвращает логическое значение. После того как команда меню будет обработана, он вернет true; тем самым сообщая, что дальнейшая обработка не нужна. Секция default вызывает реализацию суперкласса, если идентификатор команды не известен в данной реализации.

Запустить приложение BookDepository и опробовать новую команду. Добавить несколько книг и отредактировать их.

Иерархическая навигация

До настоящего момента приложение BookDepository использует кнопку Back для навигации по приложению. Кнопка Back возвращает приложение к предыдущему состоянию. 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 BookDepository, добавить атрибут parentActivityName в файл AndroidManifest.xml.

Листинг 12.13 – Включение кнопки Up (AndroidManifest.xml)

```
...
<activity
    android:name=".BookPagerActivity"
    android:label="@string/app_name"
    android:parentActivityName=".BookListActivity">
</activity>
...
```

Запустить приложение BookDepository и добавить новую книгу. Обратить внимание на появление кнопки Up (рисунок 12.10). Нажатие кнопки Up переводит приложение на один уровень вверх в иерархии BookDepository к активности BookListActivity.

Рисунок 12.10 – Кнопка Up в BookPagerActivity

Альтернативная команда меню

Далее будет добавлена команда меню, скрывающая и отображающая подзаголовок в панели инструментов BookListActivity.

В файле res/menu/fragment_book_list.xml добавить элемент действия *Show Subtitle*, который будет отображаться на панели инструментов при наличии свободного места.

Листинг 12.14 – Добавление элемента действия Show Subtitle (res/menu/fragment_book_list.xml)

```
...
    <item
    android:id="@+id/menu_item_new_book"
    ...
    app:showAsAction="ifRoom|withText"/>
    <item
        android:id="@+id/menu_item_show_subtitle"
        android:title="@string/show_subtitle"
        app:showAsAction="ifRoom"/>
</menu>
```

Команда отображает подзаголовок с количеством книг в BookDepository. Создать новый метод `updateSubtitle()`, который будет задавать подзаголовок панели инструментов.

Листинг 12.15 – Назначение подзаголовка панели инструментов (BookListFragment.java)

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    ...
}
private void updateSubtitle() {
    BookLab bookLab = BookLab.get(getActivity());
    int bookCount = bookLab.getBooks().size();
    String subtitle = getString(R.string.subtitle_format, bookCount);
    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 BookListFragment, преобразуется в AppCompatActivity. BookDepository 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 его можно вызвать при нажатии нового элемента действий.

Листинг 12.16 – Обработка элемента действия Show Subtitle (BookListFragment.java)

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_new_book:
        ...
        case R.id.menu_item_show_subtitle:
            updateSubtitle();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Запустить BookDepository, нажать элемент *Показать подзаголовок* и убедиться в том, что в подзаголовке отображается количество книг.

Переключение текста команды

Теперь подзаголовок отображается, но текст команды меню остался неизменным: *Показать подзаголовок*. Было бы лучше, если бы текст команды и функциональность команды меню изменялись в зависимости от текущего состояния подзаголовка.

При вызове `onOptionsItemSelected(…)` в параметре передается объект MenuItem для элемента действия, нажатого пользователем. Текст элемента *Показать подзаголовок* 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 заново использовать код обновления элемента действия как при выборе элемента действия пользователем, так и при повторном создании панели инструментов.

Сначала добавить переменную для хранения признака видимости подзаголовка.

Листинг 12.17 – Хранение признака видимости подзаголовка (BookListFragment.java)

```
public class BookListFragment extends Fragment {
    private RecyclerView mBookRecyclerView;
    private BookAdapter mAdapter;
    private boolean mSubtitleVisible;
    ...
```

Затем изменить подзаголовок в `onCreateOptionsMenu(…)` и инициировать повторное создание элементов действий при нажатии элемента действия *Показать подзаголовок*.

Листинг 12.18 – Обновление MenuItem (BookListFragment.java)

```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_book_list, menu);

    MenuItem subtitleItem = menu.findItem(R.id.menu_item_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.menu_item_new_book:
            ...
        case R.id.menu_item_show_subtitle:
            mSubtitleVisible = !mSubtitleVisible;
            getActivity().invalidateOptionsMenu();
            updateSubtitle();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Наконец, проверить переменную mSubtitleVisible при отображении или сокрытии подзаголовка панели инструментов.

Листинг 12.19 – Отображение или сокрытие подзаголовка (BookListFragment.java)

```
private void updateSubtitle() {
```

```
    BookLab bookLab = BookLab.get(getActivity());
    int bookCount = bookLab.getBooks().size();
    String subtitle = getString(R.string.subtitle_format, bookCount);
    if (!mSubtitleVisible) {
        subtitle = null;
    }
    AppCompatActivity activity = (AppCompatActivity) getActivity();
    activity.getSupportActionBar().setSubtitle(subtitle);
}
```

Запустить приложение BookDepository и убедиться в том, что подзаголовок успешно скрывается и отображается. Следует обратить внимание на изменение текста команды в зависимости от состояния подзаголовка.

Однако в приложении присутствуют две проблемы. Во-первых, при создании новой книги и последующем возвращении к BookListActivity кнопкой Back содержимое подзаголовка не соответствует новому количеству книг. Во-вторых, если отобразить подзаголовок, а потом повернуть устройство, подзаголовок исчезнет.

Начать следует с первой проблемы. Она решается обновлением текста подзаголовка при возвращении к BookListActivity. Инициировать вызов updateSubtitle в onResume. Метод updateUI уже вызывается в onResume и onCreate; добавить вызов updateSubtitle в метод updateUI.

Листинг 12.20 – Вывод обновленного состояния (BookListFragment.java)

```
private void updateUI() {
    BookLab bookLab = BookLab.get(getActivity());
    List<Book> books = bookLab.getBooks();
    ...
    updateSubtitle();
}
```

Запустить BookDepository, отобразить подзаголовок, создайть новую книгу и нажать на устройстве кнопку Back, чтобы вернуться к BookListActivity. На этот раз на панели инструментов будет отображаться правильное количество.

Повторите эти действия, но вместо кнопки Back можно воспользоваться кнопкой Up. Подзаголовок снова становится невидимым. Это происходит, потому что у реализации иерархической навигации в Android имеется неприятный побочный эффект: активность, к которой осуществляется переход кнопкой Up, полностью создается заново, с нуля.

Возможное решение — переопределение механизма перехода. В BookDepository можно вызвать метод `finish()` для `BookPagerActivity`,

чтобы вернуться к предыдущей активности. Реализовать этот механизм САМОСТОЯТЕЛЬНО, для этого в файле BookPagerActivity.java переопределить метод `onOptionsItemSelected(MenuItem)`, реагирующий на выбор команды меню Up. После этого в подзаголовке всегда будет отображаться правильное количество книг.

Для решения проблемы с поворотом следует сохранить переменную экземпляра mSubtitleVisible между поворотами при помощи механизма сохранения состояния экземпляров.

Листинг 12.21 – Сохранение признака видимости подзаголовка (BookListFragment.java)

```
public class BookListFragment 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);
    }
}
```

Запустить приложение BookDepository. Отобразите подзаголовок, поверните устройство. Подзаголовок должен появиться в воссозданном представлении, как и ожидалось.

Самостоятельные задания.

Задание 1. Удаление книг

Приложение BookDepository дает возможность создать новые книги, но не предоставляет средств для их удаления из списка. В этом задании

добавить в BookFragment новый элемент действия для удаления текущей книги. После того как пользователь нажимает элемент удаления, необходимо вернуть его к предыдущей активности вызовом метода `finish()` для активности-хоста BookFragment.

Задание 2. Множественное число в строках

Если список содержит одну книгу, подзаголовок «1 книг» становится грамматически неправильным. В этом задании необходимо исправить текст подзаголовка.

Можно создать несколько разных строк 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 определить в файле strings.xml элемент `plurals`:

```
<plurals name="subtitle_plural">
   <item quantity="one">%1$d книга</item>
   <item quantity="two-four">%1$d книги</item>
   <item quantity="other">%1$d книг</item>
</plurals>
```

Затем использовать метод `getQuantityString(…)` для правильного формирования множественного числа строки:

```
int bookSize = bookLab.getBooks().size();
String subtitle = getResources()
      .getQuantityString(R.plurals.subtitle_plural, bookSize, bookSize);
```

Задание 3. Пустое представление для списка

В настоящее время при запуске BookDepository отображает пустой виджет RecyclerView — большую черную пустоту. Пользователям необходимо предоставить что-то для взаимодействия при отсутствии элементов в списке. В этом задании необходимо в пустом представлении выводить сообщение (например, «Список пуст»).

13. БАЗА ДАННЫХ SQLITE

В данном разделе будет реализовано взаимодействие приложения BookDepository с базой данных SQLite. Будет рассмотрена работа основных классов поддержки SQLite в Android для выполнения операций с базами данных SQLite, а именно операций открытия, чтения и записи.

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. Более подробную информацию можно найти в полной документации SQLite по адресу http://www.sqlite.org.

Определение схемы

Прежде чем создавать базу данных, необходимо решить, какая информация будет в ней храниться. В BookDepository хранится только список книг, поэтому следует определить одну таблицу с именем books (рисунок 13.1).

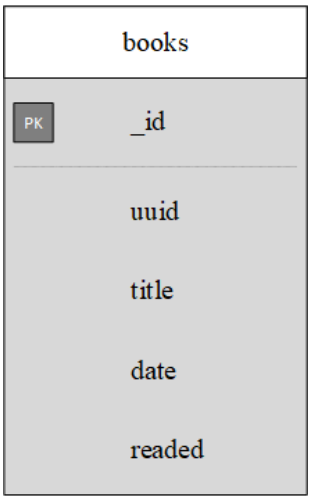

Рисунок 13.1 – Таблица books

Для определения в коде Java упрощенной схемы базы данных, в которой будет храниться имя таблицы с описанием ее столбцов, необходимо создать класс для хранения схемы. Этот класс будет называться **BookDbSchema**, но в диалоговом окне New Class следует ввести имя **database.BookDbSchema**. Файл BookDbSchema.java будет помещен в отдельный пакет database, который будет использоваться для организации всего кода, относящегося к базам данных.

В классе BookDbSchema определить внутренний класс BookTable для описания таблицы.

Листинг 13.1 – Определение BookTable (BookDbSchema.java)

```
public class BookDbSchema {
    public static final class BookTable {
        public static final String NAME = "books";
    }
}
```

Класс BookTable существует только для определения строковых констант, необходимых для описания основных частей определения таблицы. Определение начинается с имени таблицы в базе данных BookTable.NAME, за которым следуют описания столбцов.

Листинг 13.2 – Определение столбцов таблицы (BookDbSchema.java)

```
public class BookDbSchema {
    public static final class BookTable {
        public static final String NAME = "books";
        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 READED = "readed";
        }
    }
}
```

При наличии такого определения можно обращаться к столбцу с именем title в синтаксисе, безопасном для кода Java: BookTable.Cols.TITLE. Такой синтаксис существенно снижает риск изменения программы, если понадобится когда-нибудь 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 BookDbSchema.

4. Если это старая версия, выполнить код преобразования ее в новую версию.

Android предоставляет класс `SQLiteOpenHelper`, который сделает все эти действия. Создать в пакете database класс с именем `BookBaseHelper`.

Листинг 13.3 – Создание BookBaseHelper (BookBaseHelper.java)

```
public class BookBaseHelper extends SQLiteOpenHelper {
    private static final int VERSION = 1;
    private static final String DATABASE_NAME = "bookBase.db";
    public BookBaseHelper(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. Его следует использовать в BookLab для создания базы данных.

Листинг 13.4 – Открытие SQLiteDatabase (BookLab.java)

```
public class BookLab {
    private static BookLab sBookLab;
    private List<Book> mBooks;
    private Context mContext;
    private SQLiteDatabase mDatabase;
    ...
    private BookLab(Context context) {
        mContext = context.getApplicationContext();
        mDatabase = new BookBaseHelper(mContext)
                .getWritableDatabase();
        mBooks = new ArrayList<>();
    }
...
```

При вызове `getWritableDatabase()` класс `BookBaseHelper`:

1) открывает /data/data/ru.rsue.android.bookdepository/databases/ bookBase.db. Если файл базы данных не существует, то он создается;

2) если база данных открывается впервые, вызывает метод `onCreate(SQLiteDatabase)` с последующим сохранением последнего номера версии;

3) если база данных открывается не впервые, проверяет номер ее версии. Если версия базы данных в `BookOpenHelper` выше, то вызывается метод `onUpgrade(SQLiteDatabase, int, int)`.

Код создания исходной базы данных размещается в `onCreate(SQLiteDatabase)`, код обновления – в `onUpgrade(SQLiteDatabase, int, int)`, а дальше все работает само собой.

Пока приложение BookDepository 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 `BookTable`, являющийся внутренним классом `BookDbSchema`.

Процедура импортирования состоит из двух шагов. Сначала запишите начальную часть кода создания SQL:

Листинг 13.5 – Первая часть onCreate(…) (BookBaseHelper.java)

```
@Override
public void onCreate(SQLiteDatabase db) {
   db.execSQL("create table " + BookDbSchema.BookTable.NAME);
}
```

Навести курсор на слово BookTable и нажать Alt+Enter. Затем выбрать первый вариант: `Add import for 'ru.rsue.android.bookdepository.database.BookDbSchema.BookTable'`.

Android Studio генерирует директиву import, которая позволяет ссылаться на строковые константы из BookDbSchema.BookTable в форме BookTable.Cols.UUID (вместо того, чтобы вводить полное имя BookDbSchema.BookTable.Cols.UUID). Использовать это обстоятельство, чтобы завершить ввод кода определения таблицы.

Листинг 13.6 – Создание таблицы (BookBaseHelper.java)

```
@Override
public void onCreate(SQLiteDatabase db) {
   db.execSQL("create table " + BookTable.NAME + "(" +
      " _id integer primary key autoincrement, " +
      BookTable.Cols.UUID + ", " +
      BookTable.Cols.TITLE + ", " +
      BookTable.Cols.DATE + ", " +
      BookTable.Cols.READED + ")"
   );
}
```

Запустить BookDepository; приложение создаст базу данных. Если приложение выполняется в эмуляторе или на «рутованном» устройстве, то можно увидеть созданную базу данных. (На физическом устройстве это невозможно — база данных хранится в закрытой области.) Выполнить

команду Tools→Android→Android Device Monitor и заглянуть в каталог /data/data/ru.rsue.android.bookdepository/databases/ (рисунок 13.2).

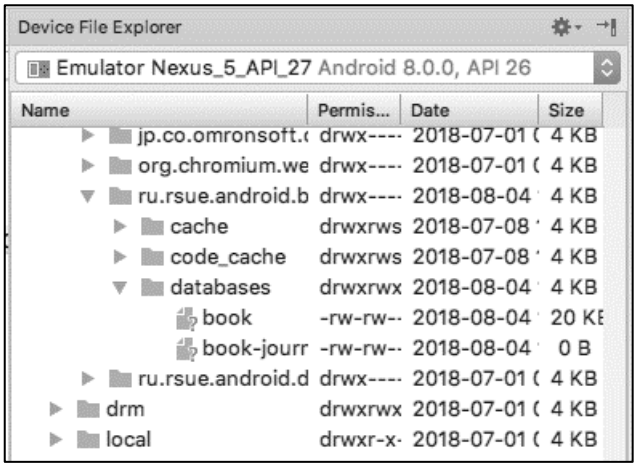

Рисунок 13.2 – Созданная база данных

Решение проблем при работе с базами данных

При написании кода для работы с базами данных SQLite иногда требуется слегка изменить структуру базы данных. Например, добавить новое поле. Для этого в таблицу 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 просто необходимо 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 BookDepository к области Uninstall у верхнего края экрана. (Если на вашем устройстве используется нестандартная версия Android, процесс может выглядеть иначе.)

Следует помнить об этом приеме, если возникнут проблемы при работе с базами данных в этом разделе.

Изменение кода BookLab

Итак, теперь в приложении есть база данных, поэтому для дальнейшей работы следует изменить довольно большой объем кода в BookLab, чтобы хранение данных осуществлялось в базе данных mDatabase, а не в mBooks.

Для начала надо расчистить место для работы. Удалить из BookLab весь код, относящийся к mBooks.

Листинг 13.7 – Удаление лишнего (BookLab.java)

```
public class BookLab {
    private static BookLab sBookLab;
    private List<Book> mBooks;
    private Context mContext;
    private SQLiteDatabase mDatabase;
    public static BookLab get(Context context) {
        ...
    }
    private BookLab(Context context) {
        mContext = context.getApplicationContext();
        mDatabase = new BookBaseHelper(mContext)
        .getWritableDatabase();
        mBooks = new ArrayList<>();
    }
    public void addBook(Book b) {
        mBooks.add(b);
    }
    public List<Book> getBooks() {
        return mBooks;
        return new ArrayList<>();
    }
    public Book getBook(UUID id) {
        for (Book book : mBooks) {
            if (book.getId().equals(id)) {
                return book;
            }
        }
        return null;
    }
}
```

Приложение BookDepository остается в состоянии, которое вряд ли можно назвать работоспособным, но в дальнейшем это будет исправлено.

Запись в базу данных

Работа с SQLiteDatabase начинается с записи данных. Приложение должно вставлять новые записи в существующую таблицу, а также обновлять уже существующие 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. 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 Book в коде BookLab. Добавить закрытый метод, который будет преобразовывать объект Book в ContentValues.

Листинг 13.8 – Создание ContentValues (BookLab.java)

```
...
    public getBook(UUID id) {
        return null;
    }
    private static ContentValues getContentValues(Book book) {
        ContentValues values = new ContentValues();
        values.put(BookTable.Cols.UUID, book.getId().toString());
        values.put(BookTable.Cols.TITLE, book.getTitle());
        values.put(BookTable.Cols.DATE, book.getDate().getTime());
        values.put(BookTable.Cols.READED, book.isReaded() ? 1 : 0);
        return values;
    }
}
```

В качестве ключей использовать 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. Заполнить метод addBook(Book) новой реализацией.

Листинг 13.9 – Вставка записи (BookLab.java)

```
public void addBook(Book b) {
    ContentValues values = getContentValues(b);
    mDatabase.insert(BookTable.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 данном случае BookTable.NAME. Последний аргумент содержит вставляемые данные.

Далее следует добавить метод обновления строк в базе данных.

Листинг 13.10 – Обновление записи (BookLab.java)

```
public Book getBook(UUID id) {
    return null;
}
public void updateBook(Book book) {
    String uuidString = book.getId().toString();
    ContentValues values = getContentValues(book);
    mDatabase.update(BookTable.NAME, values,
    BookTable.Cols.UUID + " = ?",
    new String[] {
        uuidString
    });
}
private static ContentValues getContentValues(Book book) {
    ContentValues values = new ContentValues();
    values.put(BookTable.Cols.UUID, book.getId().toString());
...
```

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[]).

Экземпляры Book, изменяемые в BookFragment, должны быть записаны в базу данных при завершении BookFragment. Добавить переопределение `BookFragment.onPause()`, которое обновляет копию Book из BookLab.

Листинг 13.11 – Запись обновлений (BookFragment.java)

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    UUID bookId = (UUID) getArguments().getSerializable(ARG_BOOK_ID);
    mBook = BookLab.get(getActivity()).getBook(bookId);
}
@Override
public void onPause() {
    super.onPause();
    BookLab.get(getActivity()).updateBook(mBook);
}
```

К сожалению, проверить работоспособность этого кода пока не удастся. С проверкой придется подождать, пока приложение не научится читать обновленные данные. Чтобы убедиться в том, что программа нормально компилируется, необходимо запустить BookDepository 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 различных операций; метод существует в нескольких перегруженных версиях. Версия, которая будет использована, выглядит так:

```
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 нужны, и порядок их извлечения. Наконец, where и whereArgs работают так же, как и в команде `update(…)`.

Создать вспомогательный метод, в котором будет вызываться метод `query(…)` для таблицы BookTable.

Листинг 13.12 – Запрос для получения данных Book (BookLab.java)

```
...
   values.put(BookTable.Cols.DATE, book.getDate().getTime());
   values.put(BookTable.Cols.READED, book.isReaded() ? 1 : 0);
   return values;
}
private Cursor queryBooks(String whereClause, String[] whereArgs) {
   Cursor cursor = mDatabase.query(
      BookTable.NAME,
      null, // Columns - null выбирает 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(BookTable.Cols.UUID));
String title = cursor.getString(
           cursor.getColumnIndex(BookTable.Cols.TITLE));
long date = cursor.getLong(
           cursor.getColumnIndex(BookTable.Cols.DATE));
int isReaded = cursor.getInt(
           cursor.getColumnIndex(BookTable.Cols.READED));
```

Каждый раз, когда будет происходить извлечение Book из курсора, этот код придется записывать снова. (Не говоря уже о коде создания экземпляра Book с этими значениями!)

Поэтому следует 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.

Создать новый класс в пакете базы данных с именем BookCursorWrapper.

Листинг 13.13 – Создание BookCursorWrapper (BookCursorWrapper.java)

```
public class BookCursorWrapper extends CursorWrapper {
   public BookCursorWrapper(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.

Добавить метод `getBook()` для извлечения данных столбцов. (Использовать для BookTable прием импортирования, состоящий из двух шагов, как это было сделано ранее.)

Листинг 13.14 – Добавление метода getBook() (BookCursorWrapper.java)

```
public class BookCursorWrapper extends CursorWrapper {
   public BookCursorWrapper(Cursor cursor) {
      super(cursor);
   }
```

```
    public Book getBook() {
        String uuidString =
                  getString(getColumnIndex(BookTable.Cols.UUID));
        String title = getString(getColumnIndex(BookTable.Cols.TITLE));
        long date = getLong(getColumnIndex(BookTable.Cols.DATE));
        int isReaded = getInt(getColumnIndex(BookTable.Cols.READED));
        return null;
    }
}
```

Метод должен возвращать объект Book с соответствующим значением UUID. Добавить в Book конструктор для выполнения этой операции.

Листинг 13.15 – Добавление конструктора Book (Book.java)

```
public Book() {
    this(UUID.randomUUID());
    mId = UUID.randomUUID();
    mDate = new Date();
}
public Book(UUID id) {
    mId = id;
    mDate = new Date();
}
```

А затем доработать метод `getBook()`.

Листинг 13.16 – Окончательная версия getBook() (BookCursorWrapper.java)

```
public Book getBook() {
    ...
    int isReaded = getInt(getColumnIndex(BookTable.Cols.READED));
    Book book = new Book(UUID.fromString(uuidString));
    book.setTitle(title);
    book.setDate(new Date(date));
    book.setReaded(isReaded != 0);
    return book;
    return null;
}
```

(Android Studio предлагает выбрать между java.util.Date и java.sql.Date. И хотя работа осуществляется с базой данных, но здесь следует выбрать java.util.Date.)

Преобразование в объекты модели

С BookCursorWrapper процесс получения List<Book> от BookLab достаточно прямолинеен. Курсор, полученный при запросе, упаковывается в BookCursorWrapper, после чего его содержимое перебирается методом `getBook()` для получения объектов Book.

Для первой части перевести queryBooks(…) на использование BookCursorWrapper.

Листинг 13.17 – Использование CursorWrapper (BookLab.java)

```
~~private Cursor queryBooks(String whereClause, String[] whereArgs) {~~
private BookCursorWrapper queryBooks(String whereClause, String[]
          whereArgs) {
   Cursor cursor = mDatabase.query(
      ...
   );
   ~~return cursor;~~
   return new BookCursorWrapper(cursor);
}
```

Метод getBooks() принял нужную форму. Добавить код запроса всех книг, организовать перебор курсора и заполнения списка Book.

Листинг 13.18 – Возвращение списка книг (BookLab.java)

```
public List<Book> getBooks() {
   ~~return new ArrayList<>();~~
   List<Book> books = new ArrayList<>();
   BookCursorWrapper cursor = queryBooks(null, null);
   try {
      cursor.moveToFirst();
      while (!cursor.isAfterLast()) {
         books.add(cursor.getBook());
         cursor.moveToNext();
      }
   } finally {
      cursor.close();
   }
   return books;
}
```

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, вызывается moveToNext(), пока isAfterLast(), наконец, не сообщит, что указатель вышел за пределы набора данных.

Последнее, что осталось сделать, — вызвать close() для объекта Cursor. Не следует забывать об этой служебной операции, это важно. Если забыть закрыть курсор, устройство Android начнет выдавать в журнал сообщения об ошибках. Со временем это приведет к исчерпанию файловых дескрипторов и сбою приложения. Итак, курсоры нужно закрывать.

Метод `BookLab.getBook(UUID)` похож на `getBooks()`, не считая того, что он должен извлечь только первый элемент данных (если тот присутствует).

Листинг 13.19 – Переработка getBook(UUID) (BookLab.java)

```
public Book getBook(UUID id) {
    return null;
    BookCursorWrapper cursor = queryBooks(
        BookTable.Cols.UUID + " = ?",
        new String[] { id.toString() }
    );
    try {
        if (cursor.getCount() == 0) {
            return null;
        }
        cursor.moveToFirst();
        return cursor.getBook();
    } finally {
        cursor.close();
    }
}
```

На данный момент удалось сделать следующее:

– реализована возможность вставлять новые книги, так что код, добавляющий Book в BookLab при нажатии элемента действия New Book, теперь работает;

– приложение обращается с запросами к базе данных, так что BookPagerActivity видит все объекты Book в BookLab;

– метод BookLab.getBook(UUID) тоже работает, так что каждый экземпляр BookFragment, отображаемый в BookPagerActivity, отображает существующий объект Book.

Теперь при нажатии *Навая книга* в BookPagerActivity появляется новый объект Book.

Запустить BookDepository и убедиться в том, что эта функциональность работает. Если что-то пошло не так, перепроверить свои реализации из этого раздела.

Обновление данных модели

Однако работа еще не закончена. Книги сохраняются в базе данных, но данные не читаются из нее. Таким образом, если нажать кнопку Back в процессе редактирования новой книги, оно не появится в BookListActivity.

Дело в том, что BookLab теперь работает немного иначе. Прежде существовал только один список List<Book> и один объект для каждой книги: тот, что содержится в List<Book>. Это объяснялось тем, что

переменная mBooks была единственным авторитетным источником данных о книгах, известный приложению.

Сейчас ситуация изменилась. Переменная mBooks исчезла, так что список List<Book>, возвращаемый getBooks(), представляет собой «моментальный снимок» данных Book на некоторый момент времени. Чтобы обновить BookListActivity, необходимо обновить этот снимок.

Большинство необходимых составляющих уже готово. BookListActivity уже вызывает `updateUI()` для обновления других частей интерфейса. Остается лишь заставить этот метод обновить его представление BookLab.

Сначала добавить в BookAdapter метод `setBooks(List<Book>)`, чтобы закрепить отображаемые в нем данные.

Листинг 13.20 – Добавление setBooks(List<Book>) (BookListFragment.java)

```
private class BookAdapter extends RecyclerView.Adapter<BookHolder> {
    ...
    @Override
    public int getItemCount() {
        return mBooks.size();
    }
    public void setBooks(List<Book> books) {
        mBooks = books;
    }
}
```

Вызвать setBooks(List<Book>) в updateUI().

Листинг 13.21 – Вызов setBook(List<>) (BookListFragment.java)

```
private void updateUI() {
    ...
    if (mAdapter == null) {
        ...
    } else {
        mAdapter.setBooks(books);
        mAdapter.notifyItemChanged(position);
    }
    updateSubtitle();
}
```

Теперь все должно работать правильно. Запустить BookDepository и убедиться в том, что есть возможность добавить книгу, нажать кнопку Back, и эта книга появится в BookListActivity.

Заодно можно проверить, что вызовы `updateBook(Book)` в BookFragment работают. Нажать на книгу и отредактировать её краткое описание в BookPagerActivity. Нажать кнопку Back и убедиться в том, что новый текст появился в списке.

Самостоятельные задания.

Задание 1. Удаление книги

В ранее добавленный элемент действия *Удалить книгу*, необходимо дополнить возможностью удаления информации из базы данных. Для этого следует вызывать метод `deleteBook(Book)` для BookLab, который вызывает метод `mDatabase.delete(…)` для завершения работы.

14. 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 описывается операция, которая должна быть выполнена, а ОС запускает активность соответствующего приложения.

В приложении BookDepository будет использован неявный интент для отправки текстовых отчетов о книге. Пользователь получит позможность выбрать приложение их список для отправки отчета (рисунок 14.1).

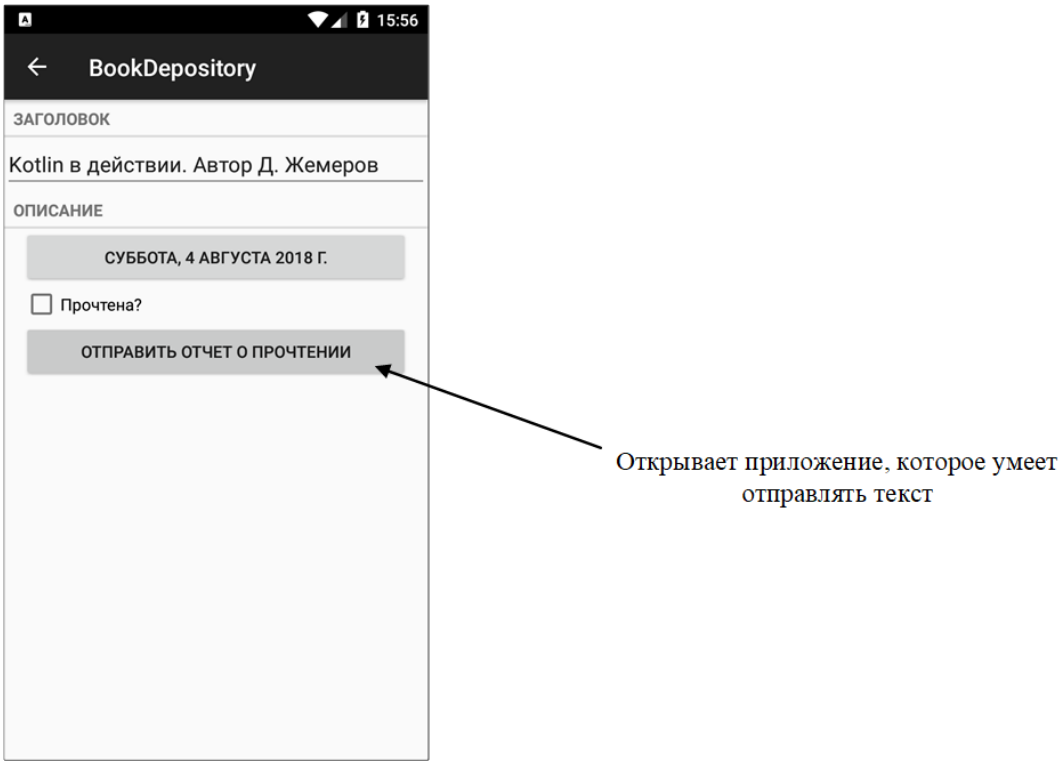

Рисунок 14.1 – Открытие приложений для отправки отчетов

Использовать функциональность других приложений при помощи неявных интентов намного проще, чем писать собственные реализации стандартных задач. Пользователям также нравится работать с приложениями, которые им уже хорошо знакомы, в сочетании с используемым приложением.

Прежде чем создавать неявные интенты, необходимо выполнить с BookDepository ряд подготовительных действий:

- добавить в макеты BookFragment кнопу отправки отчета;
- создать отчет о прочтении книги с использованием форматных строк ресурсов.

Добавление кнопок

На первом этапе необходимо включить в макеты BookFragment новую кнопку и добавить строку в ресурсы, которая будет отображаться на кнопке.

Листинг 14.1 – Добавление строки для надписи на кнопке (strings.xml)

```
...
    <string name="subtitle_format">%1$s books</string>
    <string name="book_report_text">Отправить отчет о прочтении</string>
</resources>
```

Добавить в файл layout/fragment_book.xml виджет Button, представленный на рисунке 14.2.

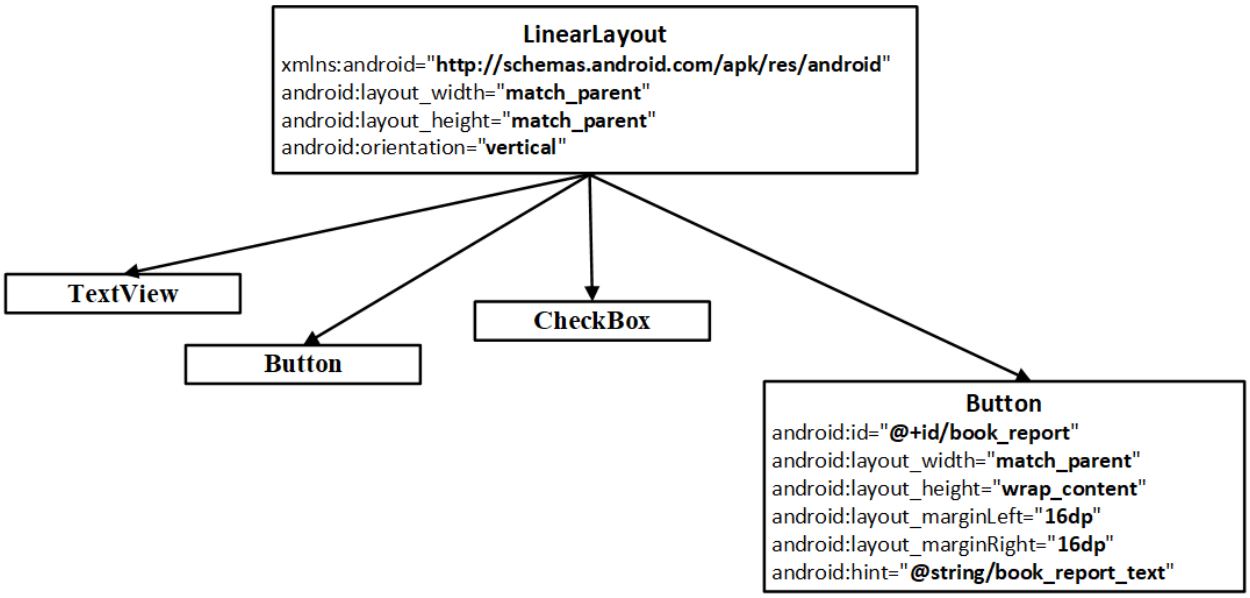

Рисунок 14.2 – Добавление кнопки для отправки отчетов (layout/fragment_book.xml)

В альбомном макете изменения будут аналогичными. Измененный макет изображен на рисунке 14.3.

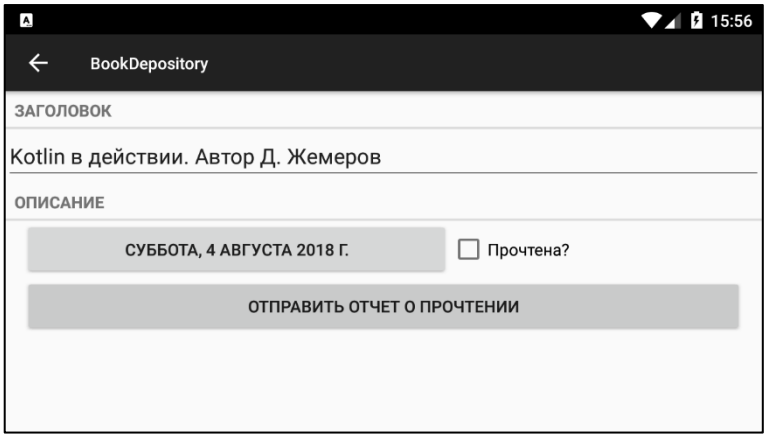

Рисунок 14.3 – Измененный альбомный макет

На этой стадии можно проверить макеты в области предварительного просмотра или запустить приложение BookDepository, чтобы убедиться в правильности расположения новой кнопки.

Форматные строки

Вторым подготовительным шагом станет создание шаблона отчета о прочтении книги, который заполняется информацией о конкретной книге, т.к. 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="book_report">%1$s Книга была запланирована к прочтению %2$s и %3$s </string>
```

Поля %1$s, %2$s и т. д. — заполнители для строковых аргументов. В коде вызывается `getString(…)`, куда передается форматная строка и необходимое количество строк в том порядке, в каком они должны заменять заполнители.

Сначала следует добавить в strings.xml строки из листинга 14.2.

Листинг 14.2 – Добавление строковых ресурсов (strings.xml)

```
...
   <string name="book_report_text">Отправить отчет о прочтении</string>
   <string name="book_report">%1$s
       Книга была запланирована к прочтению %2$s и %3$s
   </string>
   <string name="book_report_readed">Книга прочитана</string>
   <string name="book_report_unreaded">Книга не прочитана</string>
   <string name="book_report_subject">BookDepository отчет о прочтении
             </string>
   <string name="send_report">Отправить отчет о прочтении через</string>
 </resources>
```

В файле BookFragment.java необходимо добавить метод, который создает три строки, соединяет их и возвращает полный отчет.

Листинг 14.3 – Добавление метода getBookReport() (BookFragment.java)

```
...
private void updateDate() {
    mDateButton.setText(...);
}
private String getBookReport() {
    String readedString = null;
    if (mBook.isReaded()){
        readedString = getString(R.string.book_report_readed);
    }else {
        readedString = getString(R.string.book_report_unreaded);
    }
```

```
    String dateFormat = "EEE, MMM dd";
    String dateString = DateFormat
              .getDateInstance(DateFormat.MEDIUM).format(mBook.getDate());
    String report = getString(R.string.book_report,
        mBook.getTitle(), dateString, readedString);
    return report;
}
```

Обратить внимание: класс `DateFormat` существует в двух версиях, `android.text.format.DateFormat` и `java.text.DateFormat`. Используйте `android.text.format.DateFormat`.

Использование неявных интентов

Объект Intent описывает для ОС некую операцию, которую необходимо выполнить. Для *явных* интентов, использовавшихся до настоящего момента, разработчик явно указывает активность, которую должна запустить ОС.

```
Intent intent = new Intent(getActivity(), BookPagerActivity.class);
intent.putExtra(EXTRA_BOOK_ID, bookId);
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, которые находятся за пределами устройства (например, URL веб-страницы), так и URI файла или URI контента, который ссылается 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 следует 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 создается приложение-браузер, то включается 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="developer.android.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.

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.

Отправка отчета

Чтобы закрепить на практике описанную схему, будет создан неявный интент для отправки отчета о прочтении книг в приложении BookDepository. 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 `BookFragment.onCreateView(…)` получить ссылку на кнопку Send Book Report и назначить для нее слушателя. В реализации слушателя создать неявный интент и передать его `startActivity(Intent)`.

Листинг 14.4 – Отправка отчета о книге (BookFragment.java)

```
private Book mBook;
private EditText mTitleField;
private Button mDateButton;
private Button mReportButton;
...
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    ...
    mReportButton = (Button)v.findViewById(R.id.book_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, getBookReport());
            i.putExtra(Intent.EXTRA_SUBJECT,
                    getString(R.string.book_report_subject));
            startActivity(i);
        }
    });
    return v;
}
```

Здесь используется 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.

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.

Запустить приложение BookDepository и нажать кнопку Send Book Report. Так как этот интент с большой вероятностью совпадет со многими активностями на устройстве, скорее всего, на экране появится список активностей (рисунок 14.4).

Если список не появился, это может означать одно из двух: либо приложение по умолчанию уже было назначено для идентичного неявного интента, либо на устройстве имеется всего одна активность, способная реагировать на этот интент.

Также имеется возможность создать список, который будет отображаться каждый раз при использовании неявного интента для запуска активности. После создания неявного интента способом, показанным ранее, вызывается следующий метод Intent и передается ему неявный интент и строку с заголовком:

```
public static Intent createChooser(Intent target, String title).
```

Затем интент, возвращенный `createChooser(…)`, передается `startActivity(…)`.

В файле BookFragment.java создать список выбора для отображения активностей, реагирующих на неявный интент.

Листинг 14.5 – Использование списка выбора (BookFragment.java)

```
public void onClick(View v) {
    ...
    i.putExtra(Intent.EXTRA_SUBJECT,
            getString(R.string.book_report_subject));
    i = Intent.createChooser(i, getString(R.string.send_report));
    startActivity(i);
}
```

Запустить приложение BookDepository и нажать кнопку Send Book Report. Если в системе имеется несколько активностей, способных обработать интент, на экране появляется список для выбора (рисунок 14.5).

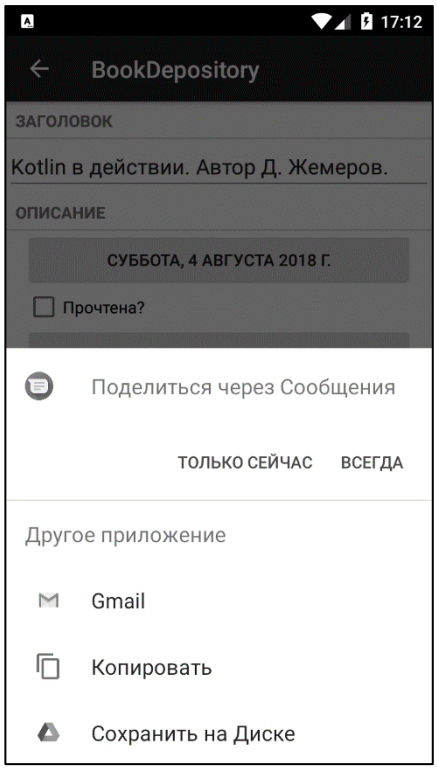

Рисунок 14.4 – Приложения, готовые отправить отчет

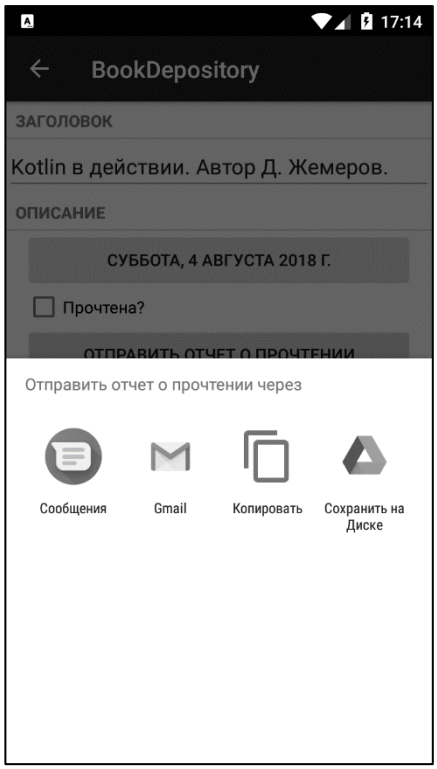

Рисунок 14.5 – Отправка текста с выбором активности

Самостоятельные задания.

Задание. ShareCompat

В библиотеку поддержки Android входит класс ShareCompat с внутренним классом IntentBuilder. ShareCompat.IntentBuilder упрощает построение интентов точно такого вида, какой был использован для кнопки отчета. Необходимо в слушателе OnClickListener кнопки mReportButton использовать для построения интента класс ShareCompat.IntentBuilder (вместо того, чтобы строить его вручную).

15. ИНТЕНТЫ ПРИ РАБОТЕ С КАМЕРОЙ

В данном разделе будет реализовано взаимодействие приложения BookDepository с камерой. Располагая снимком обложки книги, записи становятся более интересными, при этом ими можно поделиться со всеми желающими.

Для создания снимков потребуется пара новых инструментов, которые используются в сочетании с уже знакомыми неявными интентами. Неявный интент используется при запуске приложения для работы с камерой и получения от него нового снимка.

Место для хранения фотографий

Прежде всего следует обеспечить место, в котором будет отображаться фотография. Для этого понадобятся два новых объекта View: ImageView для отображения фотографии на экране и Button для создания снимка (рисунок 15.1).

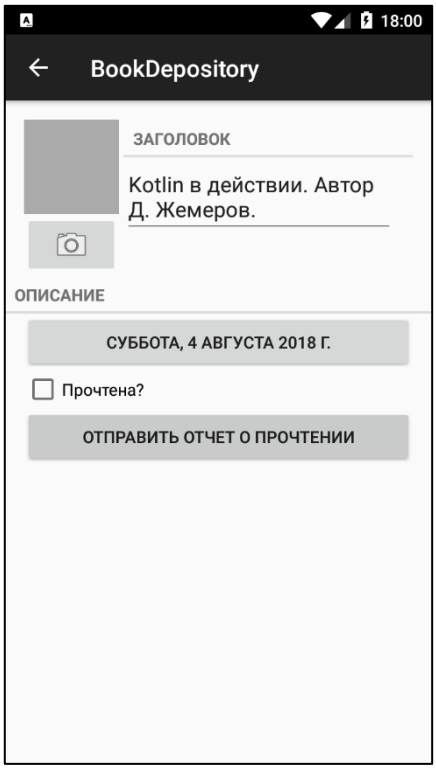

Рисунок 15.1 – Новые элементы интерфейса детализации

Включение файлов макетов

Новый макет будет включать большой раздел, который будет выглядеть одинаково как в книжной, так и в альбомной версии fragment_book.xml. Конечно, можно просто продублировать этот раздел res/layout/fragment_book.xml и res/layout-land/fragment_book.xml. Более эфективное решение —воспользоваться механизмом включения.

Включение (include) позволяет встроить один файл макета в другой. В данном случае встраивается раздел, содержащий общие элементы. Работа начинается с создания файла макета для части представления, изображенной на рисунке 15.2.

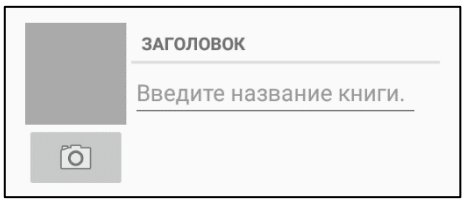

Рисунок 15.2 – Камера и текст

Присвоить файлу макета имя view_camera_and_title.xml. Начать с построения левой стороны (листинг 15.1).

Листинг 15.1 – Определение виджетов ImageView и ImageButton (view_camera_and_title.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="16dp"
    android:layout_marginRight="16dp"
    android:layout_marginTop="16dp">
    <LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:layout_marginRight="4dp">
    <ImageView
        android:id="@+id/book_photo"
        android:layout_width="80dp"
        android:layout_height="80dp"
        android:scaleType="centerInside"
        android:background="@android:color/darker_gray"
        android:cropToPadding="true"/>
    <ImageButton
        android:id="@+id/book_camera"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:src="@android:drawable/ic_menu_camera"/>
    </LinearLayout>
</LinearLayout>
```

Затем создать правую сторону (листинг 15.2).

Листинг 15.2 – Определение виджетов TextView и EditText (view_camera_and_title.xml)

```
<LinearLayout
    ...
    <LinearLayout
        ...
    </LinearLayout>
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:layout_weight="1">
        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/book_title_label"
            style="?android:listSeparatorTextViewStyle"/>
        <EditText
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:id="@+id/book_title"
            android:layout_marginRight="16dp"
            android:hint="@string/book_title_hint"/>
    </LinearLayout>
</LinearLayout>
```

Перейти в режим графического конструктора и убедиться в том, что файл макета выглядит так, как показано на рисунке 15.2.

Для включения этого макета в другие файлы макетов используются теги include. В теге include атрибут layout не использует обычный префикс android.

Начать следует с изменения главного файла макета (рисунок 15.3).

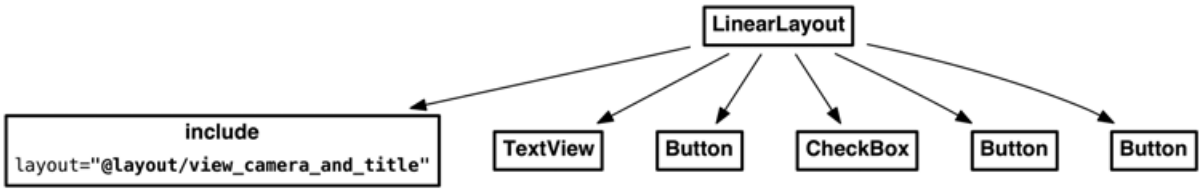

Рисунок 15.3 – Включение макета камеры для книжной ориентации (res/layout/fragment_book.xml)

Затем проделать то же для альбомного макета (рисунок 15.4).

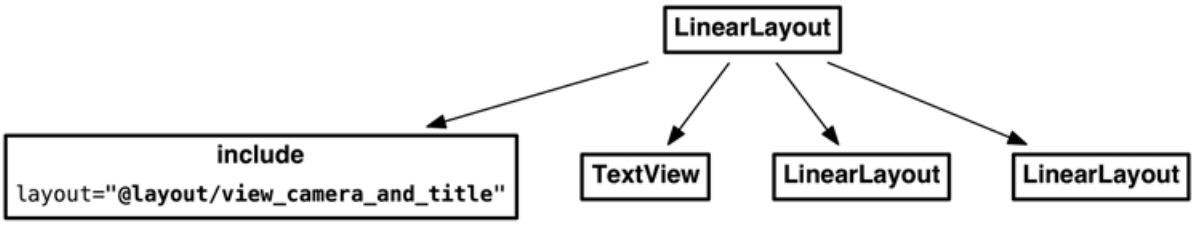

Рисунок 15.4 – Включение макета камеры для альбомной ориентации (res/layout-land/fragment_book.xml)

Запустить приложение BookDepository; новый пользовательский интерфейс должен выглядеть так, как показано на рисунке 15.1.

Внешнее хранилище

Фотографиям недостаточно одного лишь места на экране. Полноразмерная фотография слишком велика для хранения в базе данных SQLite. Ей необходимо место для хранения в файловой системе устройства.

Обычно такие данные размещаются в закрытом (приватном) хранилище. Такие методы, как: `Context.getFileStreamPath(String)` и `Context.getFilesDir()`, позволяют хранить в папке по соседству с папкой databases, в которой размещается база данных SQLite, и обычные файлы. В таблице 15.1 перечислены методы для работы с внешними файлами и каталогами в Context.

Таблица 15.1 – Методы для работы с внешними файлами и каталогами в Context

Метод	Назначение
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 сохраняются файлы, которые впоследствии будут использованы другим приложением, или возвращаются файлы от другого приложения (как, например, сохраненные фотографии), хранение должно осуществляться во внешнем хранилище.

Память внешнего хранилища делится на два вида: первичная область и все остальное. На всех устройствах Android присутствует по крайней мере одна область для внешнего хранения: *первичная* (primary) область, которая находится в папке, возвращаемой `Environment.getExternalStorageDirectory()`. Это может быть и SD-карта, но в наши дни она чаще интегрируется в самом устройстве. На некоторых устройствах имеется дополнительная внешняя память; она относится к категории «все остальное».

Класс Context тоже предоставляет методы для получения доступа к внешнему хранилищу (таблица 15.2).

Таблица 15.2 – Основные методы для работы с файлами и каталогами в Context

Метод	**Назначение**
File getExternalCacheDir()	Возвращает дескриптор папки кэша в первичной внешней области.
File[] getExternalCacheDirs()	Возвращает папки кэша для нескольких разделов внешнего хранилища.
File getExternalFilesDir(String)	Возвращает дескриптор папки в первичном внешнем хранилище, предназначенной для хранения обычных файлов. При передаче типа String можно обратиться к папке, предназначенной для конкретного типа содержимого. Константы типов определяются в Environment с префиксом DIRECTORY_. Например, изображения хранятся в Environment.DIRECTORY_PICTURES.
File[] getExternalFilesDirs(String)	То же, что `getExternalFilesDir(String)`, но возвращает все возможные папки для заданного типа.

Метод	Назначение
File[] getExternalMediaDirs()	Возвращает дескрипторы для всех внешних папок, предоставляемых Android для хранения изображений, видео и музыки. От вызова `getExternalFilesDir(Environment.DIRECTORY_PICTURES)` этот метод отличается тем, что папка автоматически обрабатывается медиасканером, соответственно файлы становятся доступными для приложений, которые воспроизводят музыку, отображают графику или видеоролики. Соответственно все, что размещается в папке, возвращаемой `getExternalMediaDirs()`, автоматически появляется в этих приложениях.

Эти методы предоставляют простые средства для обращения к первичной области, а также *относительно* простые средства для обращения ко всему остальному. Все эти методы сохраняют файлы в общедоступных местах, так что будьте осторожны.

Выбор места для хранения фотографии

Пора выделить фотографиям место, где они будут существовать. Сначала добавить в Book метод для получения имени файла.

Листинг 15.1 – Добавление свойства для получения имени файла (Book.java)

```
...
    public String getPhotoFilename() {
        return "IMG_" + getId().toString() + ".jpg";
    }
}
```

Метод `Book.getPhotoFilename()` не знает, в какой папке будет храниться фотография. Однако имя файла будет уникальным, поскольку оно строится на основании идентификатора Book.

Затем следует найти место, в котором будут располагаться фотографии. Класс BookLab отвечает за все, что относится к долгосрочному хранению данных в BookDepository, поэтому он становится наиболее естественным кандидатом. Добавить в BookLab метод `getPhotoFile(Book)`, который будет возвращать эту информацию.

Листинг 15.2 – Определение местонахождения файла фотографии (BookLab.java)

```
public class BookLab {
    ...
    public Book getBook(UUID id) {
        ...
    }
    public File getPhotoFile(Book book) {
        File externalFilesDir = mContext
            .getExternalFilesDir(Environment.DIRECTORY_PICTURES);
        if (externalFilesDir == null) {
            return null;
        }
        return new File(externalFilesDir, book.getPhotoFilename());
    }
...
```

Этот код не создает никакие файлы в файловой системе. Он только возвращает объекты File, представляющие нужные места. При этом он проверяет наличие внешнего хранилища для сохранения данных. Если внешнее хранилище недоступно, `getExternalFilesDir(String)` возвращает null — как и весь метод.

Использование интента камеры

Следующий шаг — непосредственное создание снимка. Здесь все просто: необходимо снова воспользоваться неявным интентом.

Надо начать с сохранения местонахождения файла фотографии. (Эта информация будет использоваться еще в нескольких местах, поэтому сохранение избавит от лишней работы.)

Листинг 15.3 – Сохранение местонахождения файла фотографии (BookFragment.java)

```
...
private Book mBook;
private File mPhotoFile;
private EditText mTitleField;
...
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    UUID bookId = (UUID) getArguments().getSerializable(ARG_BOOK_ID);
    mBook = BookLab.get(getActivity()).getBook(bookId);
    mPhotoFile = BookLab.get(getActivity()).getPhotoFile(mBook);
}
...
```

На следующем шаге будет подключена кнопка, которая непосредственно создает снимок.

Отправка интента

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`.

Если целевая версия платформы 24 и выше, то для того чтобы предоставить доступ к определенному файлу или папке и сделать их доступными для других приложений, необходимо использовать класс `FileProvider`. Прежде всего необходимо объявить `FileProvider` путем включения тега `<provider>` в файл AndroidManifest.xml, указав уникальный параметр атрибута `android:authority`, чтобы избежать конфликтов, например, можно использовать `${applicationId}.provider` и другие широко используемые подходы.

Листинг 15.4 – Объявление FileProvider в манифесте (AndroidManifest.xml)

```
<application
...
    <provider
        android:name="android.support.v4.content.FileProvider"
        android:authorities="${applicationId}.provider"
        android:exported="false"
        android:grantUriPermissions="true">
        <meta-data
            android:name="android.support.FILE_PROVIDER_PATHS"
            android:resource="@xml/provider_paths"/>
    </provider>
</application>
```

Затем создать файл provider_paths.xml в папке res/xml. Возможно предварительно потребуется создать папку, если она не существует. Содержимое файла показано в листинге 15.5 – В нем формируется предоставление доступа к внешнему хранилищу в корневой папке (path = ".") с именем external_files.

Листинг 15.5 – Предоставление доступа к внешнему хранилищу (provider_paths.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<paths xmlns:android="http://schemas.android.com/apk/res/android">
    <external-path name="external_files" path="."/>
</paths>
```

На последнем эпапе следует определить используемую версию платформы и исползовать соответствующий подход для формирования URI, как показано ниже в листинге 15.6.

Листинг 15.6 – Формирование URI и отправка интента камеры (BookFragment.java)

```
...
private ImageButton mPhotoButton;
private ImageView mPhotoView;

private static final int REQUEST_DATE = 0;
private static final int REQUEST_PHOTO = 1;
...
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                Bundle savedInstanceState) {
    ...
    mPhotoButton = (ImageButton) v.findViewById(R.id.book_camera);
    final Intent captureImage =
                new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    PackageManager packageManager = getActivity().getPackageManager();
    boolean canTakePhoto = mPhotoFile != null &&
            captureImage.resolveActivity(packageManager) != null;
    if (canTakePhoto) {
        Uri uri;
        if (Build.VERSION.SDK_INT<24)
            uri = Uri.fromFile(mPhotoFile);
        else
            uri = FileProvider.getUriForFile(getActivity(),
                    BuildConfig.APPLICATION_ID + ".provider", mPhotoFile);
        captureImage.putExtra(MediaStore.EXTRA_OUTPUT, uri);
        mPhotoButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
```

```
            startActivityForResult(captureImage, REQUEST_PHOTO);
        }
    });
    mPhotoView = (ImageView) v.findViewById(R.id.book_photo);
    return v;
}
```

Приведенный выше код такде формирует неявный интент для сохранения фотографии в месте, определяемом `mPhotoFile`. Добавляет код, который блокирует кнопку при отсутствии приложения камеры или недоступности места для сохранения фотографии. Также направляется запрос к `PackageManager` активности, реагирующие на неявный интент камеры, с целью проверки доступности приложения камеры.

Запустить BookDepository и нажать кнопку запуска приложения камеры (рисунок 15.5).

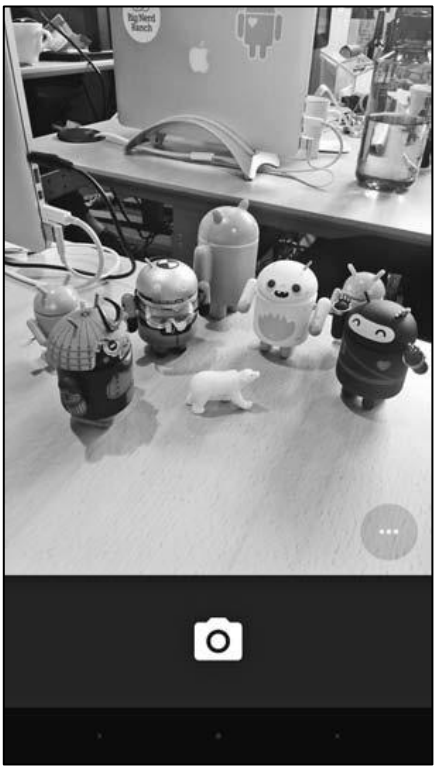

Рисунок 15.5 – Приложение BookDepository, работающее с камерой

Масштабирование и отображение растровых изображений

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());
```

Однако Bitmap — простой объект для хранения необработанных данных пикселов. Таким образом, даже если исходный файл был сжат, в объекте Bitmap никакого сжатия не будет. Поэтому 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 это 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.

Создать для этого метода новый класс с именем PictureUtils.java и добавить в него статический метод с именем `getScaledBitmap(String, int, int)`.

Листинг 15.7 – 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) {
      if (srcWidth > srcHeight) {
         inSampleSize = Math.round(srcHeight / destHeight);
      } else {
         inSampleSize = Math.round(srcWidth / destWidth);
      }
   }
   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.

Также при запуске фрагмента неизвестна величина `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.

Для данной задачи есть два решения: либо подождать, пока будет сделан первый проход, либо воспользоваться консервативной оценкой. Второй способ менее эффективен, но более прямолинеен.

Написать еще один статический метод с именем `getScaledBitmap(String, Activity)` для масштабирования Bitmap под размер конкретной активности.

Листинг 15.8 – 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`, добавить в BookFragment метод для обновления `mPhotoView`.

Листинг 15.9 – Обновление mPhotoView (BookFragment.java)

```
...
    private String getBookReport() {
        ...
    }
    private void updatePhotoView() {
        if (mPhotoFile == null || !mPhotoFile.exists()) {
            mPhotoView.setImageDrawable(null);
        } else {
```

```
            Bitmap bitmap = PictureUtils.getScaledBitmap(
            mPhotoFile.getPath(), getActivity());
            mPhotoView.setImageBitmap(bitmap);
        }
    }
}
```

Затем вызвать этот метод из `onCreateView(…)` и `onActivityResult(…)`.

Листинг 15.10 – Вызов updatePhotoView() (BookFragment.java)

```
mPhotoButton.setOnClickListener(new View.OnClickListener() {
    ...
});
mPhotoView = (ImageView) v.findViewById(R.id.book_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_PHOTO) {
        updatePhotoView();
    }
}
```

Запустить приложение снова. Изображение выводится в уменьшенном виде.

Объявление функциональности

Остается решить еще одну задачу: сообщить об использовании камеры 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 данным устройством.

Чтобы объявить, что в приложении используется камера, включите тег `<uses-feature>` в AndroidManifest.xml.

Листинг 15.11 – Добавление тега uses-feature (AndroidManifest.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
...
   <uses-feature android:name="android.hardware.camera"
       android:required="false"
   />
...
```

В этом примере в тег добавляется необязательный атрибут `android:required`. По умолчанию объявление об использовании некоторой возможности означает, что без нее приложение может работать некорректно. К BookDepository это не относится. Использованый метод `resolveActivity(…)` проверит наличие приложения камеры, после чего корректно заблокирует кнопку, если приложение не найдено.

Атрибут `android:required="false"` корректно обрабатывает эту ситуацию. При этом Android сообщается, что приложение может нормально работать без камеры, но некоторые части приложения окажутся недоступными.

Самостоятельные задания.

Задание 1. Вывод увеличенного изображения

Пользователь видит уменьшенное изображение, но вряд ли ему удастся рассмотреть его во всех подробностях. Создайте новый фрагмент `DialogFragment`, в котором отображается увеличенная версия фотографии обложки книги. Когда пользователь нажимает в какой-то точке миниатюры, на экране должен появиться DialogFragment с увеличенным изображением.

Задание 2. Эффективная загрузка миниатюры

В этом разделе был использован довольно грубый подход для оценки размера изображение, до которого оно должно быть уменьшено. Такое решение не идеально, но оно работает и быстро реализуется. В существующих API можно использовать `ViewTreeObserver` — объект, который можно получить от любого представления в иерархии Activity:

```
ViewTreeObserver observer = mImageView.getViewTreeObserver();
```

Для `ViewTreeObserver` можно зарегистрировать разнообразных слушателей, включая `OnGlobalLayoutListener`. Этот слушатель инициирует событие каждый раз, когда происходит проход обработки макета.

Изменить свой код так, чтобы он использовал размеры `mPhotoView`, когда они действительны, и ожидал прохода обработки макета перед первым вызовом `updatePhotoView()`.

Задание 3. Добавление изображения из Галереи

Пользователи чаше всего читают электронные книги и сфотографировать обложку не представляется возможным. Создать новую кнопку, которая будет создавать неявный интент для открытия приложения Галерея и загрузки выбранной фотографии (рисунок 15.6).

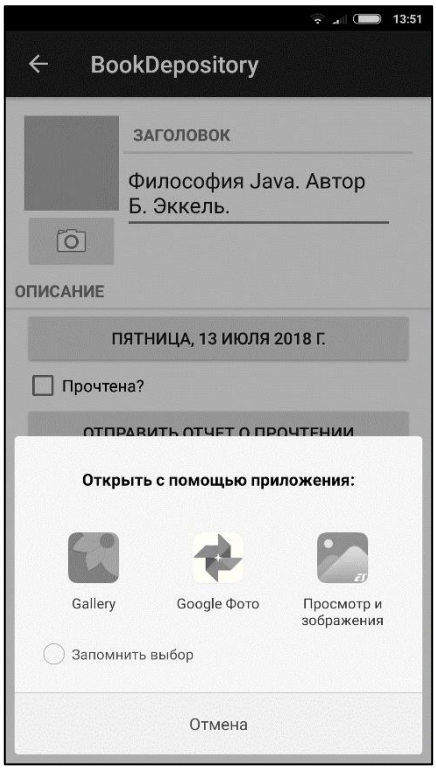

Рисунок 15.6 – Приложения, готовые открыть изображения для загрузки обложки

ЗАКЛЮЧЕНИЕ

При написании данного пособия ставилась цель представить основные программные компоненты и решения для разработки мобильных приложений под операционную систему Android. Многие вопросы не вошли в данное пособие. Это прежде всего работа со звуком и видео, анимация, деловая графика, вопросы интеграции приложений с другими системами, например, Cloud Services.

По мнению авторов, данное пособие может сформировать представление о возможностиях технологий Google для разработки мобильных приложений под операционную систему Android, а детальное изучение данных вопросов и получение практических навыков является прерогативой читателей.

ЗАДАНИЯ ДЛЯ ИНДИВИДУАЛЬНОГО ВЫПОЛНЕНИЯ

А) Содержание задания

Создать Android-приложение в соответствии с вариантом (приложение А). В приложении реализовать функционал, позволяющий:

- отображать элементы списка в соответствии с заданием;
- отображать элементы, удовлетворяющие условию;
- добавлять новые элементы в список;
- редактировать элементы списка;
- удалять элементы списка;
- сортировать список по одному из критериев;
- осуществлять поиск элемента по заданному критерию.

Подготовить отчет о проделанной работе (**обязательно**).

Важно! В названии пакета приложения должна присутствовать фамилия студента, например, ru.rsue.glushenko.

Б) Оформление отчета должно быть осуществлено в строгом соответствии с действующими Гостами и предусматривает обязательное наличие следующего:

Титульный лист

Содержание

Введение

Глава 1. Постановка задачи для приложения

Глава 2. Использование приложения

Заключение

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Голощапов А. Google Android: программирование для мобильных устройств. – СПб.: БХВ-Петербург, 2010. – 448 с.
2. Коматинэни С., Маклин Д., Хэшими С. Google Android: программирование для мобильных устройств. Pro Android 2. – 1-е изд. – СПб.: Питер, 2011. – 736 с.
3. Сатия Коматинени, Дэйв Маклин. Android 4 для профессионалов. Создание приложений для планшетных компьютеров и смартфонов. Pro Android 4. – М.: Вильямс. 2013. – 880 с.
4. Роджерс Р., Ломбардо Д. Android. Разработка приложений. – М.: ЭКОМ Паблишерз, 2010. – 400 с.
5. Донн Фелкер. Android: разработка приложений для чайников. Android Application Development For Dummies. – М.: Диалектика, 2011. – 336 с.
6. Официальный сайт Android // http://www.android.com/
7. Google Groups. Форум разработчиков Android // http://groups.google.com/group/android-developers
8. Сайт разработчиков Android-приложений // https://developers.google.com/android/

Приложение А

Варианты индивидуального задания по разработке Android–приложения

1. В магазине сформирован список постоянных клиентов, который включает ФИО, домашний адрес покупателя и размер предоставляемой скидки. Вывести всех покупателей, имеющих *5 %* - ную скидку.

2. Список товаров, имеющихся на складе, включает в себя наименование товара, количество единиц товара, цену единицы и дату поступления товара на склад. Вывести список товаров, стоимость которых превышает *100 000* рублей.

3. Для получения места в общежитии формируется список студентов, который включает ФИО студента, номер группы, средний балл, доход на члена семьи. Вывести фамилии студентов, у которых доход на члена семьи *меньше двух* минимальных зарплат.

4. В справочной автовокзала имеется расписание движения автобусов. Для каждого рейса указаны его номер, тип автобуса, пункт назначения, время отправления и прибытия. Вывести информацию о рейсах, которыми можно воспользоваться для прибытия в пункт назначения раньше заданного времени.

5. На междугородной АТС информация о разговорах содержит дату разговора, код и название города, время разговора, тариф, номер телефона абонента. Вывести для заданного города общее время разговоров с ним и сумму.

6. Информация о сотрудниках фирмы включает ФИО, табельный номер, количество отработанных часов за месяц, почасовой тариф. Вывести размер заработной платы каждого сотрудника.

7. Информация об участниках спортивных соревнований содержит название страны, название команды, ФИО игрока, игровой номер, возраст, рост и вес. Вывести фамилии спортсменов, возраст которых *больше 20* лет.

8. Для книг, хранящихся в библиотеке, задаются регистрационный номер книги, автор, название, год издания, издательство, количество страниц. Вывести список книг с фамилиями авторов, изданных *после заданного года*.

9. Различные цеха завода выпускают продукцию нескольких наименований. Сведения о выпущенной продукции включают наименование, количество, номер цеха. Для заданного цеха вывести количество выпущенных изделий.

10. Информация о сотрудниках содержит ФИО, номер отдела, должность, стаж работы на предприятии. Вывести список сотрудников заданного отдела, имеющих стаж работы на предприятии *более 20* лет.

11. Ведомость абитуриентов содержит ФИО, адрес, оценки по *трем* предметам. Определить средний балл абитуриентов, проживающих в городе *Минске*.

12. В справочной аэропорта имеется расписание вылета самолетов. Для каждого рейса указаны его номер, тип самолета, пункт назначения, время вылета. Вывести все номера рейсов, вылетающих в заданный пункт назначения.

13. У администратора железнодорожных касс имеется информация о свободных местах в поездах на текущие сутки в следующем виде: пункт назначения, время отправления, число свободных мест. Вывести информацию о числе свободных мест в поездах, следующих до заданного пункта назначения.

14. Ведомость абитуриентов, сдавших вступительные экзамены в университет, содержит ФИО абитуриента и его оценки. Определить средний балл по университету и вывести список абитуриентов, средний балл которых выше среднего балла по университету.

15. В радиоателье хранятся квитанции о сданной в ремонт радиоаппаратуре. Каждая квитанция содержит наименование изделия, дату приемки в ремонт, состояние готовности заказа (выполнен, не выполнен). Вывести информацию об изделиях, ремонт которых еще не выполнен.

Учебное издание

Разработка мобильных приложений

Учебное пособие

Сергей Андреевич Глушенко
Алексей Иванович Долженко

Редактор Грузинская Т. А.
Корректор Петросян И. В.
Выпускающий редактор Акимова Л. И.
Директор издательства

Изд. № . Подписано к печати .
Объем уч.-изд. л. Гарнитура «Times New Roman»
Заказ . Тираж 100 экз.

344002, г. Ростов-на-Дону, ул. Б. Садовая, 69, РГЭУ (РИНХ).
Отпечатано в типографии РИЦ РГЭУ (РИНХ)