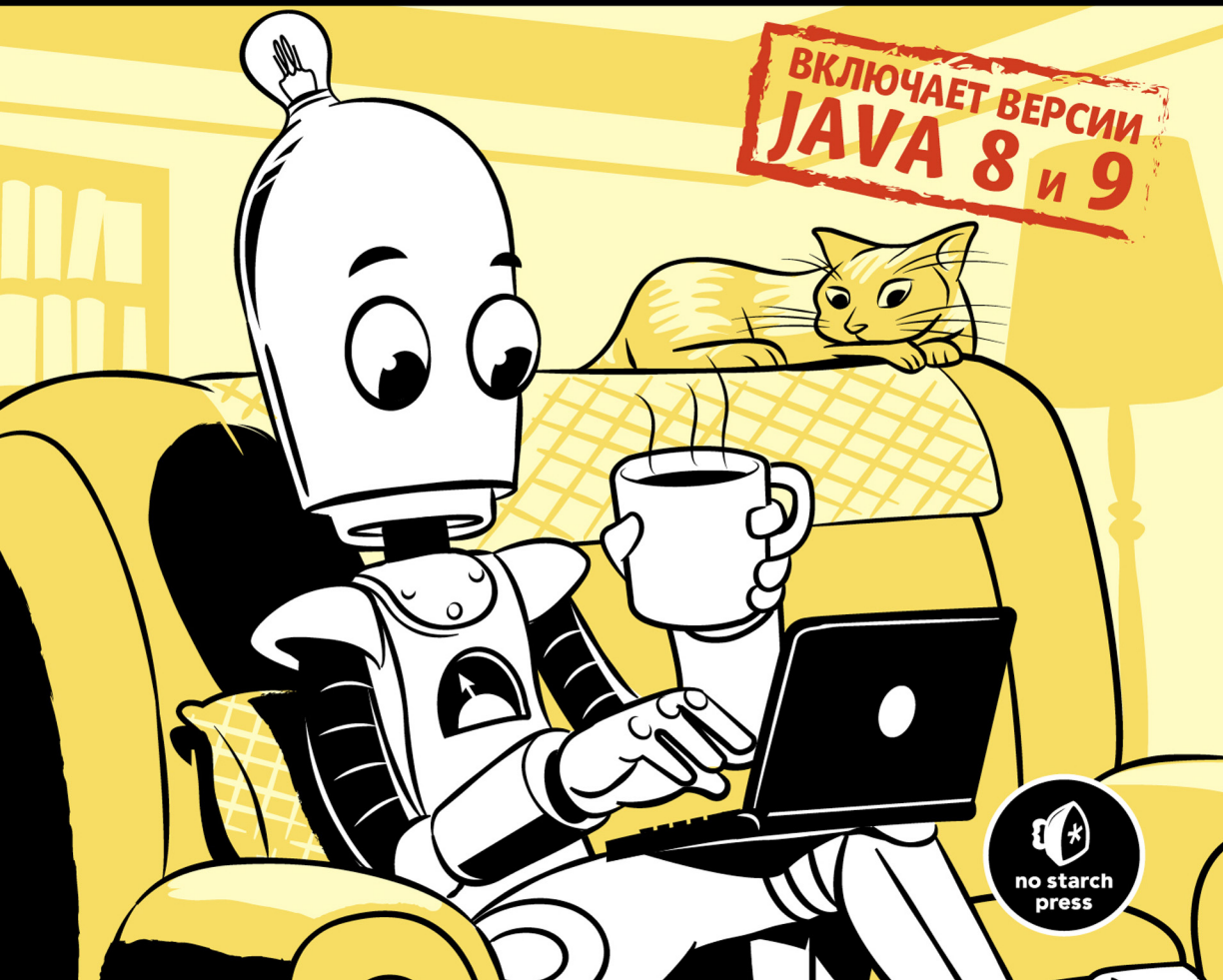
ЛЕГКИЙ СПОСОБ ВЫУЧИТЬ JAVA

БРАЙСОН ПЕЙН

ПРАКТИЧЕСКИЕ ПРИМЕРЫ ДЛЯ БЫСТРОГО СТАРТА




мировой
компьютерный
бестселлер

BRYSON PAYNE

LEARN JAVA THE EASY WAY

A HANDS-ON INTRODUCTION TO PROGRAMMING

БРАЙСОН ПЕЙН

ЛЕГКИЙ СПОСОБ ВЫУЧИТЬ JAVA

ПРАКТИЧЕСКИЕ ПРИМЕРЫ ДЛЯ БЫСТРОГО СТАРТА

БОМБОРА™
Москва 2019

УДК 004.45
ББК 32.973-018.2
П23

LEARN JAVA THE EASY WAY: A HANDS-ON INTRODUCTION TO PROGRAMMING
Bryson Payne

Пейн, Брайсон.

П23 Легкий способ выучить Java / Брайсон Пейн. — Москва : Эксмо, 2019. — 400 с. — (Мировой компьютерный бестселлер).

ISBN 978-5-04-093540-6

Java — один из самых популярных и востребованных языков программирования в мире, но и один из самых сложных для изучения, особенно для новичков. Автор этой книги, Брайсон Пейн, разработал собственный метод обучения, который строится на прохождении материала исключительно на практических примерах. Начните изучать Java, создавая несложные игры для ПК и Android, узнавайте, как работает инструмент JShell, используйте популярные среды разработки Eclipse и Android Studio, учитесь искать и исправлять ошибки в коде и становитесь востребованным программистом с книгой «Легкий способ выучить Java»!

УДК 004.45
ББК 32.973-018.2

Научно-популярное издание

МИРОВОЙ КОМПЬЮТЕРНЫЙ БЕСТСЕЛЛЕР

Брайсон Пейн

ЛЕГКИЙ СПОСОБ ВЫУЧИТЬ JAVA

Главный редактор *Р. Фасхутдинов.* Ответственный редактор *Е. Истомина*
Младший редактор *Е. Минина.* Художественный редактор *А. Гусев*

ООО «Издательство «Эксмо»
123308, Москва, улМ. Зорге, д. 1. Тел.: 8 (495) 411-68-86.
Home page: www.eksmo.ru E-mail: info@eksmo.ru
Өндіруші: «ЭКСМО» АҚБ Баспасы, 123308, Мәскеу, Ресей, Зорге көшесі, 1 үй.
Тел.: 8 (495) 411-68-86.
Home page: www.eksmo.ru E-mail: info@eksmo.ru.
Тауар белгісі: «Эксмо»
Интернет-магазин : www.book24.ru
Интернет-дүкен : www.book24.kz
Импортёр в Республику Казахстан ТОО «РДЦ-Алматы».
Қазақстан Республикасындағы импорттаушы «РДЦ-Алматы» ЖШС.
Дистрибьютор и представитель по приему претензий на продукцию,
в Республике Казахстан: ТОО «РДЦ-Алматы»
Қазақстан Республикасында дистрибьютор және өнім бойынша арыз-талаптарды
қабылдаушының өкілі «РДЦ-Алматы» ЖШС,
Алматы қ., Домбровский көш., 3«а», литер Б, офис 1.
Тел.: 8 (727) 251-59-90/91/92; E-mail: RDC-Almaty@eksmo.kz
Өнімнің жарамдылық мерзімі шектелмеген.
Сертификация туралы ақпарат сайтта: www.eksmo.ru/certification

Сведения о подтверждении соответствия издания согласно законодательству РФ о техническом регулировании можно получить на сайте Издательства «Эксмо» www.eksmo.ru/certification
Өндірген мемлекет: Ресей. Сертификация қарастырылмаған

Подписано в печать 29.11.2018. Формат 70x100 1/16.
Печать офсетная. Усл. печ. л. 32,41.
Тираж экз. Заказ

ISBN 978-5-04-093540-6

ISBN 978-5-04-093540-6

ОГЛАВЛЕНИЕ

БЛАГОДАРНОСТИ

Эта книга никогда не была бы написана без исключительной поддержки команды издательства No Starch Press. Особенная благодарность Биллу Поллоку, Тайлеру Ортману, Райли Хоффману, Яну Кэшу, Серене Ян, Аманде Харири и Джулии Борден за их неустанное редактирование, обзоры и маркетинг, а также за бесчисленные способы, которыми они помогли мне улучшить мою первоначальную рукопись до этой книги.

Благодарю Брайана Фагана за его замечательную работу в качестве технического редактора.

Благодарю всех моих нынешних и бывших учеников, которые вдохновляют меня на продолжение создания интересного, содержательного контента — особенно Шаха и Сьюзан Рахман, Джастина и Диану Тернер, Якова Эллиота, Брайана Мюррея, Аарона Уокера, Роберта Брауна, Трента Диала, Сета Парка, Саймона Сингха, Эндрю Миллера, команду робототехники Ctrl-Alt-Del, Джексона Гранта, Квентина Кернса, Грейс и Джека Галли, Мэтью Харпура и Дэвида Найта. Спасибо также студентам из 150 стран, которые подписаны на мои онлайн-курсы, особенно Хейдену Редду и Бобу Уотсону.

Благодарю моих вдохновляющих коллега и друзей, которые постоянно мотивируют меня делать все наилучшим образом: Маркуса Хитца, Чака Робертсона, Ирен Коккала, Тамирата Абегаза,

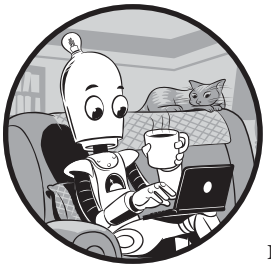
Антонио Санс Монтемайора, Дона Уоткинса, Эдди Мини, Роуз Проктер, Рона Ларсоан, Виктора Паркера, Кейт Антония, Билли Уэллса, Джима Голди, Джеймса Даниэла и Крейга Джентри.

Благодарю моего тестя, Нормана Петти, чью страсть к технологии превосходит только его любовь к семье, и моего отчима Дейла Карвера, у которого всегда есть время на мои 3D-принтеры. Огромная благодарность моей прекрасной жене Бев и моим удивительным сыновьям Алексу и Максу за их нескончаемое терпение в то время, когда я писал две книги за три года.

Особая благодарность Калену Коулу. В одиннадцать лет Кален прочитал и воспринял мою первую книгу «Научите своих детей коду», научившись кодировать в своем собственном темпе. Он даже отправил мне свой первый творческий проект, используя строки кода, которые он написал сам. В добрый путь, Кален! Ты вдохновляешь всех детей, которые только начинают осознавать, кто они и кем они хотят стать.

Посвящается моей матери за то, что всегда в меня верила

ВВЕДЕНИЕ



Язык программирования Java используется в миллиардах устройств по всему миру. Начиная с мобильных приложений и заканчивая программным обеспечением стационарных компьютеров, Java приводит в действие крупнейшие корпоративные компьютерные системы и самые маленькие персональные устройства. Студенты, специалисты по информационным технологиям и все, кто интересуется карьерой в области программирования, в итоге понимают, что им необходимо изучить Java. Будучи профессором информатики, преподающим язык Java уже почти 20 лет, я написал эту книгу, чтобы помочь вам изучить Java так же, как я сам научился программировать: на практических примерах. Я обнаружил, что лучше всего студенты учатся, создавая реальные приложения и игры, которые интересны, увлекательны и достойны внимания. В этой книге вы создадите простую игру-угадайку «Больше-Меньше», приложение для шифрования сообщений для обмена ими с друзьями и интерактивное приложение для рисования под названием «Рисование пузырьков». Читателям книги не требуется опыт программирования, но если вы изучили другие языки, вы также быстрее разберетесь в Java, используя этот практический подход.

Почему стоит изучать программирование?

Первая причина — наличие рабочих мест. По данным Бюро статистики труда США, семь из десяти самых быстрорастущих и высокооплачиваемых вакансий относятся к области компьютерных технологий. Программисты уже востребованы во всем мире, и в ближайшие несколько лет потребуются миллионы новых. Ваше местоположение не имеет значения, пока у вас есть подключение к Интернету, вы можете зарабатывать деньги в качестве программиста из любой точки земного шара.

Но высокий доход и гарантия занятости — это не единственные причины, чтобы научиться кодировать. Программирование — это решение проблем, а мир нуждается в большем количестве специалистов по решению проблем. Вы можете создавать приложения, которые соединяют людей и помогают им работать. Вы можете придумать новые формы торговли и даже создать совершенно новые рынки. Вы можете разрушить барьеры, помочь человеку или сообществу или даже целому континенту и создать возможности, которых раньше не было. Благодаря доступу в Интернет и смартфонам вы можете написать приложение и поделиться им с миллиардами людей.

Дрю Хьюстон (Drew Houston), основатель Dropbox, говорит, что программирование — «ближе всего к сверхспособностям из того, что нам доступно». Гейб Ньюэлл (Gabe Newell), соучредитель компании Valve, занимающейся разработкой компьютерных игр, говорит, что знание того, как написать код, «сродни обладанию магическими способностями в отличие от всех остальных». Компьютеры вокруг нас: в каждом устройстве, каждой системе и каждой сети в нашей повседневной жизни, а код — это то, что заставляет все эти компьютеры работать. Научитесь программировать, и вы обеспечите себе успех в высокотехнологичном будущем.

Почему стоит изучать язык программирования Java?

Существует несколько веских причин, по которым Java многими считается языком программирования № 1 в мире. Во-первых, он работает практически на всех типах устройств, которые можно себе представить: от стационарных компьютеров и ноутбуков до умных телевизоров. Один и тот же код на языке Java будет

одинаково работать в операционных системах Windows, macOS или Linux.

Во-вторых, Java используется компаниями для запуска некоторых крупнейших корпоративных приложений. Java — это объектно-ориентированный язык программирования, который специально спроектирован для того, чтобы разработчики программного обеспечения могли создавать многопользовательские приложения в различных областях, начиная от производства и заканчивая продажами и подбором сотрудников бухгалтерии.

В-третьих, Java — один из самых популярных языков в колледжах и университетах по всему миру, поэтому его знание позволит быстро наладить взаимодействие с коллегами-программистами.

Независимо от того, чем является для вас программирование: хобби, подработкой или основной работой — с языком Java вы можете начать программировать быстро и бесплатно. Java хорош и как первый язык, и как *следующий*, если вы уже изучали программирование.

Структура этой книги

Давайте проведем краткий обзор того, что вы изучите в каждой главе.

В **главе 1** вы найдете руководство по установке и настройке языка Java и сред разработки Eclipse и Android Studio. Вы также напишете свои первые команды на языке Java, используя интерактивную оболочку JShell.

В **главе 2** вы создадите свое первое приложение — игру «Больше-Меньше»: консольную программу с текстовыми управляющими конструкциями, в которой игрок угадывает случайное число от 1 до 100.

В **главе 3** вы улучшите «Больше-Меньше» до оконного, настольного приложения, оснащенного графическим пользовательским интерфейсом (GUI, Graphical User Interface) с метками, текстовым полем и кнопкой, срабатывающей по щелчку мыши.

В **главе 4** вы создадите свое первое мобильное приложение для Android, повторно использовав большую часть кода игры из предыдущих двух глав.

После того как вы закончите мобильное приложение, в **главе 5** вы добавите несколько последних штрихов в игру «Больше-Меньше», включая меню настроек и возможность сохранять рекорды.

В **главе 6** вы начнете создавать новую программу — приложение «Секретные сообщения», программу с текстовыми управляющими конструкциями, которая шифрует сообщения с помощью шифра Цезаря.

В **главе 7** вы улучшите приложение «Секретные сообщения», оснастив его графическим интерфейсом: добавите ползунковый регулятор, который позволит вам быстро взламывать шифр Цезаря.

Затем в **главе 8** вы создадите мобильную версию приложения «Секретные сообщения» с функцией совместного использования, которое позволит одним нажатием кнопки обмениваться друг с другом секретными сообщениями по электронной почте, SMS или в социальных сетях.

В **главе 9** вы начнете создавать самое визуализированное приложение в книге — приложение «Рисование пузырьков», которое позволяет рисовать на экране разноцветные пузырьки.

В **главе 10** вы добавите анимацию, чтобы пузыри начинали двигаться по экрану и отскакивать от краев. В **главе 11** вы добавите мультисенсорную функцию для рисования пузырьков в нескольких местах одновременно. В конце этой главы у вас будет готово приложение профессионального уровня, которым вы наверняка захотите поделиться со всеми своими друзьями!

И, наконец, в приложении к этой книге вы найдете советы по отладке и предотвращению распространенных ошибок при написании программ на языке Java в средах разработки Eclipse и Android Studio.

Какие инструменты вам понадобятся?

С помощью этой книги вы научитесь работать в двух самых популярных в отрасли средах разработки на языке Java — Eclipse и Android Studio. Поэтому, как только вы закончите книгу, вы сразу же будете готовы создавать настоящие рабочие приложения. Очень удобно, что эти среды разработки можно скачать и использовать совершенно бесплатно.

Чтобы начать работу вам понадобится только доступ в Интернет и обычный настольный компьютер или ноутбук с операционной системой Windows, macOS или Linux. Вам даже не нужен смартфон или планшет с операционной системой Android для создания мобильных приложений, потому что Android Studio предоставляет бесплатный эмулятор для Android, который вы можете

использовать для тестирования своих приложений. Конечно, если у вас есть устройство под Android, вы сможете запускать свои мобильные приложения прямо на своем смартфоне или планшете.

Информационные ресурсы в Интернете

Если вы захотите получить дополнительную помощь, вы можете бесплатно загрузить весь исходный код для приложений из Интернета по адресу **https://eksmo.ru/files/learn_java_easy_way.zip**. Если вы захотите получить пошаговые видеоинструкции, онлайн-курс по адресу **www.udemy.com/java-the-easy-way/** проведет вас через каждый пример и каждую строку кода.

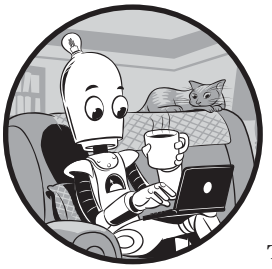
Не откладывайте на завтра!

Вы точно ничего не потеряете и много получите, научившись программировать на Java. Начните сегодня. Программирование — это ключ к новому миру возможностей, и изучение языка программирования Java может стать вашим первым шагом к новой карьере и в новое будущее. Вне зависимости от возраста у моих студентов в университете есть одна общая черта: они сделали первый шаг. Они написали свою первую строку кода и свою первую программу, а затем росли и развивались как программисты. Вы можете сделать то же самое.

Китайская мудрость гласит: «Лучшее время для посадки дерева было 20 лет назад. Сейчас второе лучшее время». Если вы все еще учитесь в школе или задумываетесь о смене профессии, то теперь для вас самое лучшее время, чтобы заняться программированием.

Глава 1

ПРИСТУПАЯ К РАБОТЕ



В этой главе вы начнете с установки на ваш компьютер языка Java и сред разработки Eclipse и Android Studio, а затем познакомитесь с некоторым базовыми основами программирования на языке Java и примените некоторые команды в оболочке интерактивной командной строки JShell.

Java — это мощный многоплатформенный язык программирования, который можно загрузить и установить под операционными системами Windows, macOS и Linux. Eclipse, *интегрированная среда разработки* (*IDE*, Integrated development environment), представляет собой инструментарий для быстрого и легкого создания приложений на языке Java. Android Studio — это IDE для разработки мобильных приложений для операционной системы Android на языке Java. Она дает вам возможность создавать собственные мобильные игры и приложения для смартфонов, планшетов и т. д.

Java в Windows, macOS и Linux

Одной из замечательных особенностей языка Java является то, что программы, написанные на Java, могут запускаться на любом

другом компьютере, на котором установлено программное обеспечение виртуальная машина Java (JVM, Java Virtual Machine), иногда называемое Java Runtime Environment или JRE. Технология JVM позволяет вам написать программу один раз и запускать один и тот же код в любой операционной системе (Windows, macOS, Linux, Android и т. д.).

Такая технология кажется естественной, но большинство других языков программирования требуют либо написания разного кода для Windows, macOS, Linux и смартфонов, либо компиляции отдельной версии для каждой операционной системы. JVM — среда выполнения, которая делает ненужной такую дополнительную работу.

На рис. 1.1 показано одно и то же графическое приложение Java, работающее в операционной системе Windows, macOS и Ubuntu Linux.

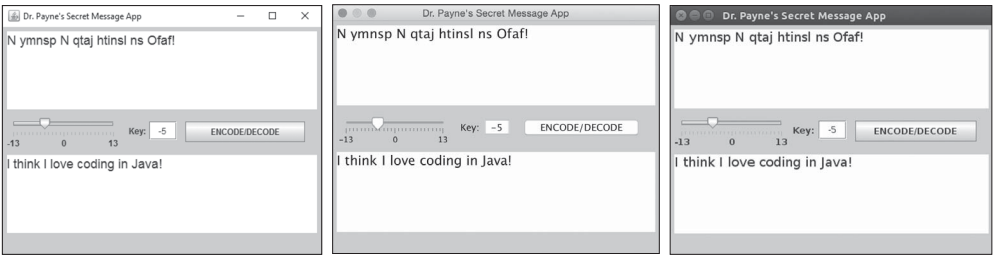


Рис. 1.1. Один и тот же код Java, работающий под тремя разными операционными системами: Windows, macOS и Linux

Эта простая, но мощная идея является одной из причин, по которой язык программирования Java получил признание у энтузиастов и компаний по всему миру.

Установка версий 8 и 9 языка Java для разработчиков

Комплект разработчика приложений на языке Java (JDK, Java Development Kit) — это версия языка Java для разработчиков или программистов, таких как вы. JDK дает вам возможность писать и компилировать ваши собственные приложения на языке Java, которыми вы можете поделиться с друзьями, использовать для работы и запускать в любом месте и практически на любом устройстве.

Мы установим две версии JDK 8 и 9. Поступив таким образом, мы можем воспользоваться преимуществами широкого

распространения версии 8, а также получить доступ к новейшим функциям версии 9.

Последовательность действий для установки JDK 8 следующая:

1. В браузере перейдите на страницу **jdk.java.net/8/**.
2. Нажмите кнопку **Accept License Agreement** (Принять лицензионное соглашение).
3. Найдите ссылку для своей операционной системы в списке загрузок (**Downloads**) в столбце JDK и щелкните по ней мышью. Если ваша операционная система Windows или Linux, выберите 64-разрядную версию.
4. Откройте установочный файл JDK из каталога загрузок вашего компьютера и установите JDK.

Чтобы установить JDK 9, перейдите на страницу **jdk.java.net/9/**, а затем повторите шаги 2–4 из инструкции по установке JDK 8*.

Чтобы получить пошаговые видеоинструкции по установке, вы можете посмотреть бесплатный демонстрационный ролик онлайн-курса, связанного с этой книгой, по адресу **www.udemy.com/java-the-easy-way/**.

Это все, что требуется для подготовки компьютера к компиляции и запуску программ на языке Java из оболочки командной строки. Но мы хотим использовать возможности Java для создания приложений, обладающих графическим интерфейсом пользователя (GUI, graphical user interface), таких как на рис. 1.1. Чтобы иметь возможность использовать широкие возможности графического интерфейса языка Java, следующим шагом мы установим интегрированную среду разработки Eclipse.

Установка интегрированной среды разработки Eclipse для разработчиков на языке Java

Eclipse — одна из самых популярных сред разработки для программирования на языке Java. Она распространяется как программное обеспечение с открытым исходным кодом, а это значит, что она

* На момент написания книги была доступна версия только для операционной системы Linux (*прим. ред.*).

бесплатна для личного и коммерческого использования. Также существует процветающее сообщество, постоянно совершенствующее и поддерживающее Eclipse. Существует ряд альтернативных сред разработки, но именно Eclipse славится простотой в использовании для разработки приложений на Java. Ее также довольно легко установить.

Чтобы установить Eclipse на свой компьютер, перейдите по адресу **www.eclipse.org/downloads/**, загрузите программу установки для вашей операционной системы (как показано на рис. 1.2), а затем запустите ее. На момент написания книги текущей версией была Eclipse Oxygen.

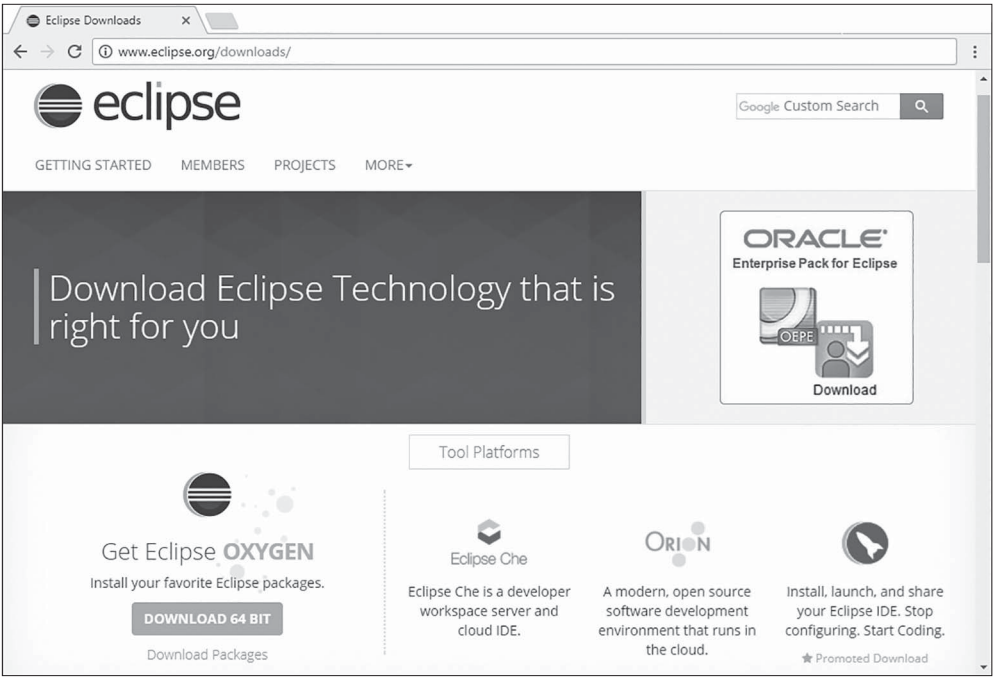


Рис. 1.2. Загрузите установщик среды разработки Eclipse для вашей операционной системы

Вы увидите меню, подобное показанному на рис. 1.3. Выберите пункт **Eclipse IDE for Java Developers** и нажмите кнопку **Install** (Установить). Будьте внимательны, выберите именно вариант **Eclipse IDE for Java Developers**, а не один из других вариантов (модули Java EE или Enterprise Edition не имеют некоторых функций, которые мы будем использовать в этой книге). Установка может занять несколько минут.

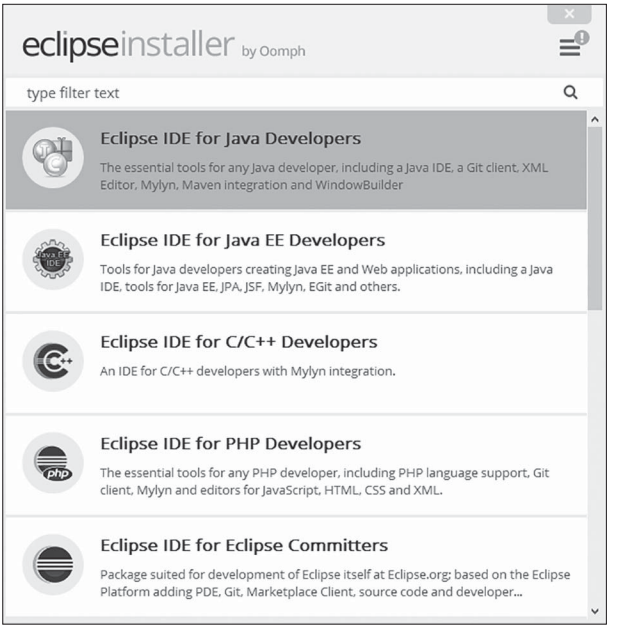


Рис. 1.3. Выберите пункт **Eclipse IDE for Java Developers** из меню установщика Eclipse

Настройка IDE Eclipse

Давайте настроим программу Eclipse так, чтобы она выглядела и работала как профессиональная среда разработки, в комплекте с редактором WindowBuilder, комфортной цветовой схемой и удобными для чтения шрифтами.

Запустите среду разработки Eclipse, щелкнув мышью по ярлыку программы. При запуске программы Eclipse появляется запрос о том, где вы хотите разместить свое рабочее пространство, в котором будут храниться все ваши проекты, как показано на рис. 1.4. Вы можете использовать местоположение по умолчанию (в Windows это *C:\Users\<имяПользователя>\eclipse-workspace*, в macOS — */Users/<имяПользователя>/Documents/eclipse-workspace/*, а в Linux — */home/<имяПользователя>/eclipse-workspace/*) либо выбрать другую папку для размещения рабочего пространства.

Если у вас нет особых предпочтений, используйте местоположение рабочего пространства Eclipse по умолчанию. И в любом случае запомните расположение этой папки, потому что там будут храниться все ваши проекты. Если вы установите флажок **Use this as the default and do not ask again** (Использовать эту папку

по умолчанию, больше не спрашивать), диалоговое окно **Eclipse Launcher** не будет появляться при каждом запуске Eclipse. Если вы планируете использовать несколько рабочих областей, вы можете оставить флажок снятым, чтобы вы могли легко переключаться между рабочими пространствами при запуске Eclipse.

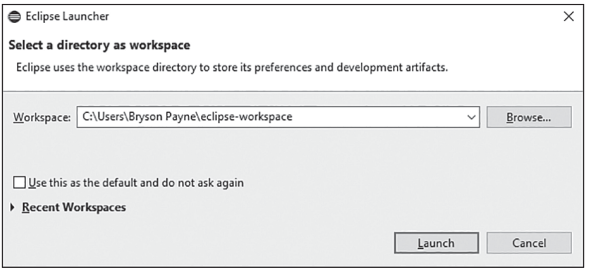


Рис. 1.4. IDE Eclipse стартует с запроса о месте хранения ваших проектов

НАСТРОЙКА ECLIPSE ДЛЯ РАБОТЫ С JAVA 9

Если программа Eclipse не запускается, вам потребуется внести еще одно изменение. На момент написания этой книги Java 9 все еще была новинкой, и в то время как версии Eclipse Oxygen и более поздние поддерживают Java 9, некоторые версии требуют внести изменение в конфигурационный файл *eclipse.ini* для работы с Java 9. Чтобы внести это изменение, выполните следующие действия.

1. Найдите папку установки Eclipse.
   - В Windows нужно щелкнуть правой кнопкой мыши по ярлыку программы Eclipse и выбрать в контекстном меню пункт **Open file location** (Расположение файла). Файл *eclipse.ini* находится в той же папке, что и файл программы *eclipse.exe*.
   - В macOS найдите приложение Eclipse в папке *Applications* с помощью файлового менеджера Finder, затем, нажав и удерживая клавишу **Ctrl**, щелкните мышью по ярлыку приложения и выберите пункт меню **Show Package Contents** (Показать содержимое пакета). Откройте папку *Contents*, затем папку *Eclipse*, и вы увидите *eclipse.ini* в списке файлов. В Linux перейдите в свою домашнюю папку и откройте *eclipse/java-oxygen/eclipse*, чтобы найти *eclipse.ini*.

2. Щелкните правой кнопкой мыши (щелкните мышью, нажав и удерживая клавишу **Ctrl)** по файлу *eclipse.ini*. Выберите пункт

Open with (Открыть с помощью) и затем программу Notepad (Блокнот), TextEdit или другой текстовый редактор на ваш выбор.

3. Добавьте следующую строку в конец файла *eclipse.ini*:

```
--add-modules=ALL-SYSTEM
```

Сохраните файл *eclipse.ini*, а затем снова откройте среду разработки Eclipse. Eclipse должна запускаться правильно с этого момента.

При первом запуске программы Eclipse вы увидите экран приветствия. В зависимости от вашей версии Eclipse этот экран может включать в себя некоторые полезные примеры проектов и учебники, или он может выглядеть проще. Если хотите, вы можете щелкнуть мышью и немного изучить предоставленный материал, а когда вы будете готовы двигаться дальше, закройте вкладку экрана приветствия, щелкнув мышью по маленькому значку ×.

Установка редактора WindowBuilder

Самое важное обновление, которое мы сделаем для Eclipse — это установка редактора WindowBuilder, который позволит нам создавать оконные приложения, перетаскивая элементы графического пользовательского интерфейса, такие как кнопки, метки и текстовые поля, на графический макет приложения.

В некоторые версии Eclipse уже встроен редактор WindowBuilder, но мы пошагово пройдем через этапы его установки, чтобы быть уверенными в готовности создавать графические приложения, которые нам предстоят начиная с главы 3.

Сначала пройдите по ссылке **www.eclipse.org/windowbuilder/** и нажмите кнопку **Download** (Скачать). На странице загрузки найдите версию редактора WindowBuilder, которая соответствует вашей версии IDE Eclipse (для Eclipse Oxygen, это версия 4.7), щелкните правой кнопкой мыши (щелкните мышью, нажав и удерживая клавишу **Ctrl**) по соответствующей ссылке и скопируйте адрес ссылки, как показано на рис. 1.5.

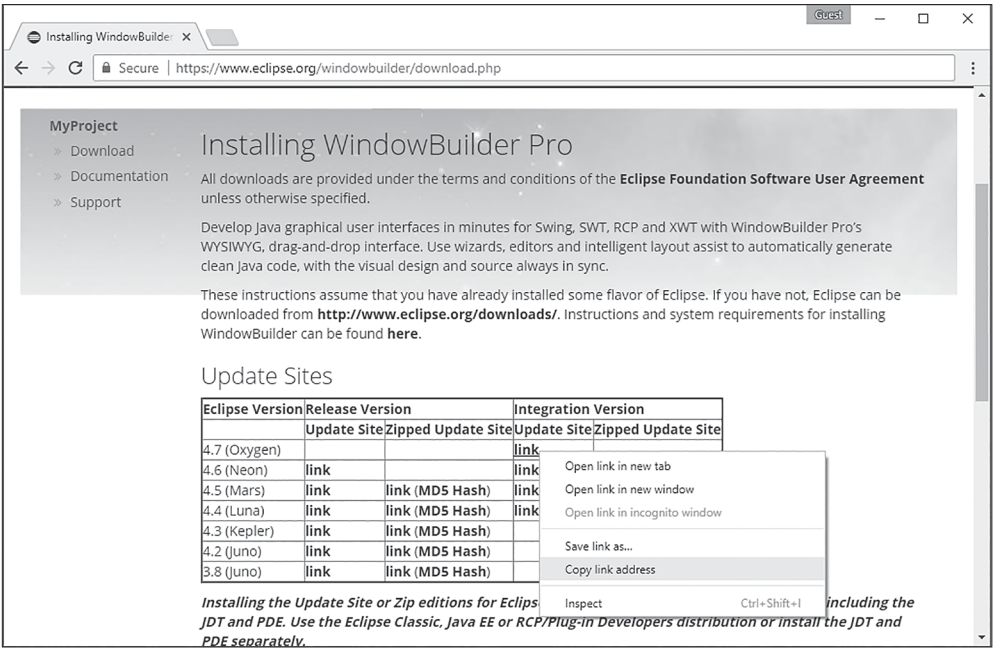


Рис. 1.5. Найдите ссылку для скачивания редактора WindowBuilder, а затем скопируйте ее

Затем вернитесь в Eclipse и выберите команду меню **Help** ⇒ **Install New Software** (Помощь ⇒ Установить новое ПО). В текстовом поле **Work with** (Обработать) вставьте ссылку на дистрибутив редактора WindowBuilder (для Eclipse Oxygen, это **http://download.eclipse.org/windowbuilder/WB/integration/4.7/**), нажмите кнопку **Add...** (Добавить) и введите буквы **WB** в поле **Name** (Имя) в появившемся диалоговом окне, как показано на рис. 1.6.

Нажмите кнопку **OK**, и когда в окне установки появится пункт **WindowBuilder**, нажмите кнопку **Select All** (Выбрать все), чтобы установить все необходимые компоненты WindowBuilder. Нажимайте кнопку **Next** (Далее), пока не будет предложено принять лицензионное соглашение, и, наконец, нажмите кнопку **Finish** (Завершить).

Установка программного обеспечения может занять несколько минут, вы увидите индикатор прогресса в нижнем правом углу окна Eclipse. По завершении установки вам будет предложено перезапустить программу Eclipse. Нажмите кнопу **Restart Now** (Перезапустить сейчас), и установка WindowBuilder завершится.

А теперь давайте выполним несколько дополнительных настроек, чтобы упростить чтение кода в Eclipse путем изменения фона, цветов текста и шрифта.

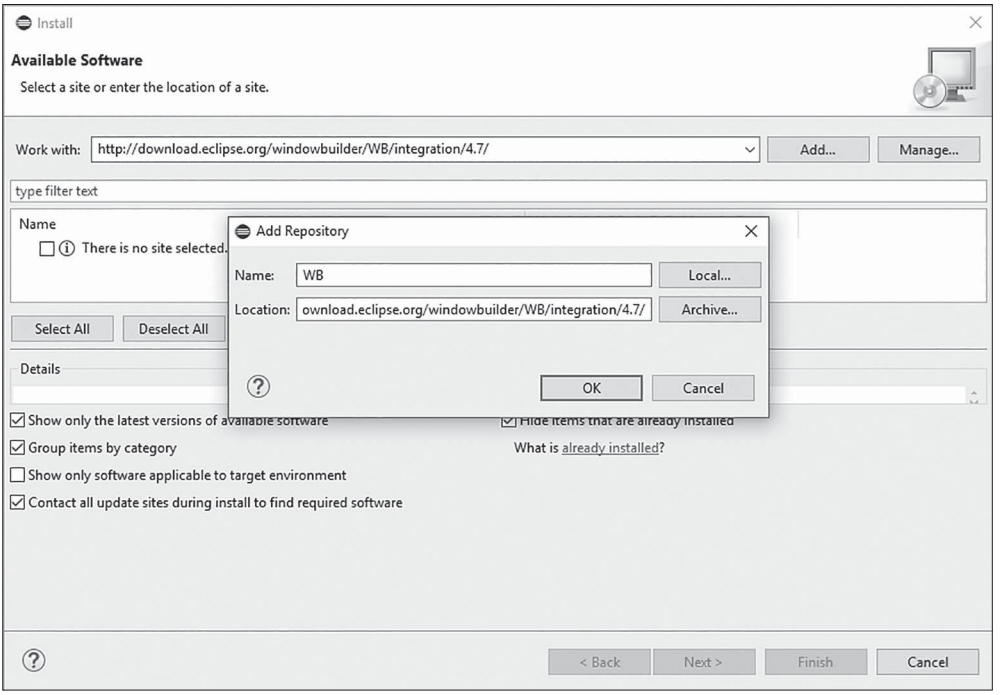


Рис. 1.6. Используйте диалоговое окно **Install New Software** среды разработки Eclipse, чтобы установить редактор WindowBuilder

Настройка внешнего вида и поведения среды разработки Eclipse

Завершив установку всех необходимых элементов, вы можете настроить интерфейс программы Eclipse. В Windows и Linux вы можете получить доступ к окну настроек, выполнив команду меню **Window** ⇒ **Preferences** (Окно ⇒ Настройки). В операционной системе macOS выберите команду меню **Eclipse** ⇒ **Preferences** (Eclipse ⇒ Настройки).

Вы можете изменить, например, *тему* (цветовую палитру) и *размер шрифта*, используемые в текстовом редакторе. В зависимости от диагонали используемого монитора и среды, в которой вы программируете, выбранная тема и размер шрифта могут иметь большое значение для удобочитаемости, комфорта и даже производительности.

В категории **General** ⇒ **Appearance** (Основные ⇒ Внешний вид) диалогового окна **Preferences** (Настройки) вы увидите раскрывающийся список **Theme** (Тема). Вы можете выбирать тему,

например **Classic** (Классическая; светлый, сероватый фон с темными шрифтами) или **Dark** (Темная; черный фон со светлыми, более яркими шрифтами). Я предпочитаю вариант **Dark** (Темная), потому что цвета шрифтов выглядят ярче на темном фоне и текст легче читать на мониторе или через проектор.

В категории **General** ⇒ **Appearance** ⇒ **Colors and Fonts** (Основные ⇒ Внешний вид ⇒ Цвета и шрифты) диалогового окна **Preferences** (Настройки) вы можете изменить размер шрифта. В разделе **Colors and Fonts** (Цвета и шрифты) в правой части диалогового окна выберите пункт **Basic** ⇒ **Text Font** (Основные ⇒ Шрифт текста) и нажмите кнопку **Edit** (Изменить). Появится диалоговое окно **Font** (Шрифт). Выберите шрифт, который вы можете легко прочитать. Я предпочитаю Courier New или Consolas. Я рекомендую использовать размер шрифта от 18 до 20 и стиль шрифта **Bold** (Полужирный). Диалоговое окно **Font** (Шрифт) отобразит образец выбранного шрифта, стиля и размера, как показано на рис. 1.7.

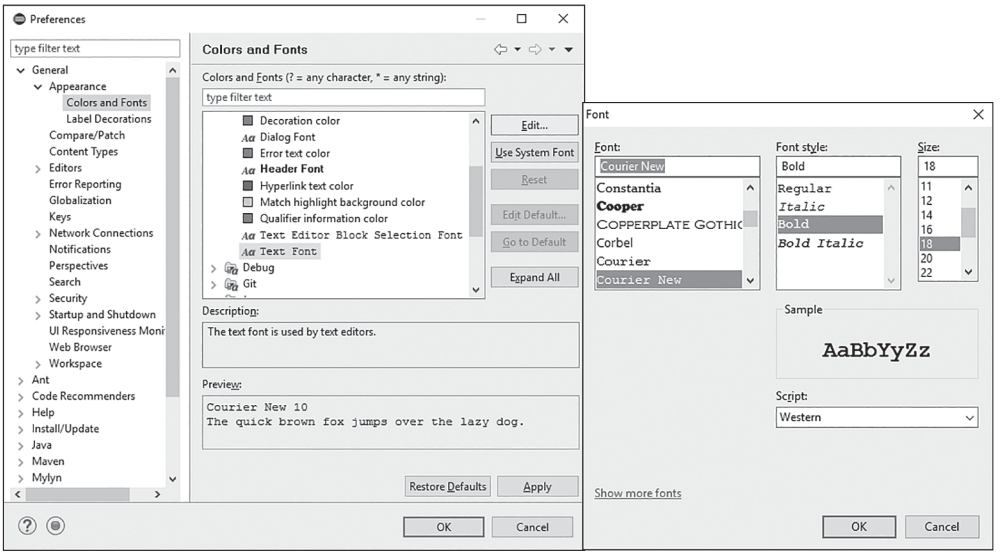


Рис. 1.7. Раздел **Colors and Fonts** (слева) и диалоговое окно **Font** (справа)

Нажмите кнопку **OK**, когда закончите настройку, и вы вернетесь в основное рабочее пространство программы Eclipse. Результат настройки вы увидите, когда начнете писать код в текстовом редакторе языка Java в главе 2.

Установка IDE Android Studio для разработки мобильных приложений

Android Studio — официальная среда разработки мобильных приложений, работающих на устройствах под управлением операционной системы Android. Она предоставляет возможность разрабатывать и кодировать мобильные приложения с использованием языка программирования Java, адаптированного для Android. Подобно Java и Eclipse, Android Studio можно бесплатно скачать, установить и использовать. Из-за большого размера Android Studio ее загрузка и установка могут занять от нескольких минут до нескольких часов, в зависимости от скорости интернет-соединения.

Чтобы загрузить Android Studio, перейдите на страницу **developer.android.com/studio/** и нажмите кнопку **Download Android Studio**, как показано на рис. 1.8. Ознакомьтесь с условиями и примите их, а затем нажмите кнопку **Download Android Studio for** ***ваша операционная система***.

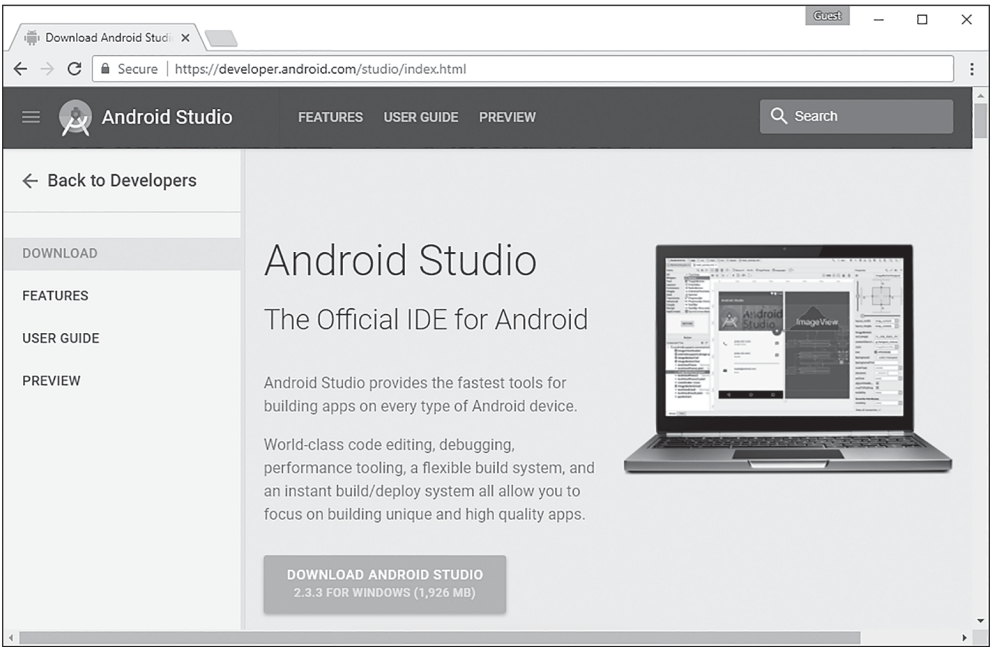


Рис. 1.8. Android Studio — официальная среда разработки мобильных приложений для платформы Android

Следуйте инструкциям для завершения установки. Программа установки может загрузить дополнительные *компоненты для*

разработки программного обеспечения (*SDK*, software development kit), и это может занять еще несколько минут после загрузки всех компонентов.

Знакомство с языком Java при помощи JShell

После подготовки среды программирования для себя давайте протестируем наши настройки с помощью JShell — интерактивного интерпретатора языка Java. JShell — отличный способ начать изучение того, как работает язык Java, так как сразу показывает вам результат работы вашего кода. До появления версии языка Java 9 программисты Java должны были ввести полный код программы, скомпилировать его и запустить, чтобы увидеть результат. Теперь с помощью JShell мы можем ввести одну строку кода на языке Java, такую как `System.out.println("Hello, Java!")`, нажать клавишу **Enter** и сразу увидеть результат на экране, как показано на рис. 1.9.

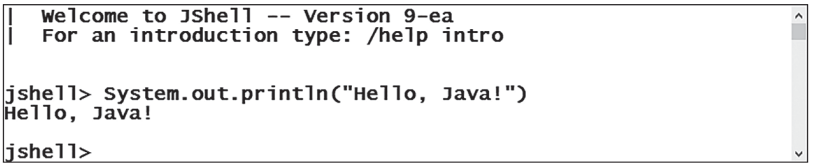


Рис. 1.9. JShell в версии языка Java 9 позволяет нам быстро протестировать работу кода в оболочке интерактивной командной строки

Любая допустимая инструкция языка Java будет работать в оболочке JShell, что делает ее отличным инструментом для изучения основ программирования на Java, поэтому давайте приступим!

Запуск JShell

Вы должны иметь возможность запускать JShell либо из оболочки командной строки, либо с помощью ярлыка. На тот случай, если один из этих вариантов у вас не сработает, мы познакомимся с обоими.

Для начала у вас должен быть установлен JDK 9. Вы можете убедиться в этом, выполнив команду в оболочке командной строки, указанную ниже.

Ниже указаны способы запуска интерфейса командной строки в операционных системах Windows, macOS и Linux.

- В операционной системе Windows запустите оболочку командной строки, нажав кнопку поиска на панели задач и введя значение **cmd** в открывшемся поле. Нажмите клавишу **Enter** или щелкните мышью по значку оболочки командной строки.
- В операционной системе macOS запустите приложение Launchpad и введите **terminal** в поле поиска. Щелкните мышью по значку приложения Terminal (Терминал).
- В операционной системе Linux выполните поиск по слову **terminal** и щелкните мышью по значку приложения Terminal (Терминал).

После этого появится окно оболочки командной строки. Введите команду **`java -version`** в приглашении и вы увидите, какая версия Java JDK установлена.

В операционной системе Windows вы увидите что-то подобное:

```
C:\Users\Payne> java -version
java version "9-ea"
Java(TM) SE Runtime Environment (build 9-ea+153)
Java HotSpot(TM) 64-Bit Server VM (build 9-ea+153, mixed mode)
```

В операционной системе macOS и Linux:

```
Payne:~ payne$ java -version
java version "1.9.0_33"
Java(TM) SE Runtime Environment (build 1.9.0_33)…
```

Если в ответе Java содержится текст с указанием версии 9 или версии 1.9, как показано выше, то вы можете запустить JShell. Если вы хотите запустить его из оболочки командной строки, перейдите к следующему разделу этой главы. Если в ответе вы получили более раннюю версию Java, например 1.8, вернитесь к разделу «Установка версий 8 и 9 языка Java для разработчиков» ранее в этой главе и установите JDK 9. Если версия 9 не появляется в оболочке командной строки после установки или если вы хотите настроить ярлык на рабочем столе, перейдите к разделу «Запуск JShell с помощью ярлыка» далее в этой главе.

Запуск JShell из оболочки командной строки

Чтобы запустить JShell из оболочки командной строки, введите **`jshell`** и нажмите клавишу **Enter**. Java ответит приветственным сообщением и приглашением JShell, которые в операционной системе Windows выглядят следующим образом:

```
C:\Users\Payne> jshell
|  Welcome to JShell — Version 9-ea
|  For an introduction type: /help intro
jshell>
```

В операционной системе macOS или Linux сообщение и приглашение выглядят одинаково:

```
Payne:~ payne$ jshell
|  Welcome to JShell — Version 9-ea
|  For an introduction type: /help intro
jshell>
```

Если вы видите приглашение `jshell>`, можно переходить к разделу «Работа с выражениями языка Java в JShell» далее в этой главе. Если команда `jshell` не работает на вашем компьютере после установки JDK 9, попробуйте воспользоваться инструкцией, приведенной в следующем разделе.

Запуск JShell с помощью ярлыка

Если у вас возникли проблемы с запуском JShell из оболочки командной строки или если вы хотите запустить его с помощью ярлыка на рабочем столе, следуйте этим инструкциям для доступа к JShell непосредственно из папки *bin* JDK 9 и настройте ярлык для запуска. (Имя папки *bin* является сокращением от *binaries* (двоичные файлы), которые представляют собой программы, написанные на компьютерном языке с помощью всего двух символов 1 и 0.)

К сожалению, этот метод не работает в Linux, там вы должны воспользоваться оболочкой командной строки.

Вы можете найти JShell в следующем каталоге:

- В операционной системе Windows: *C:\Program Files\Java\jdk-9\bin\jshell.exe*

- В операционной системе macOS: */Library/Java/JavaVirtualMachines/jdk-9.jdk/Contents/Home/bin/jshell*

Папка с установкой JDK на вашем компьютере может иметь имя вида *jdk-1.9.x* вместо *jdk-9*, но она будет работать в любом случае. Перейдите в папку *bin* JDK 9 и найдите файл JShell, как показано на рис. 1.10.

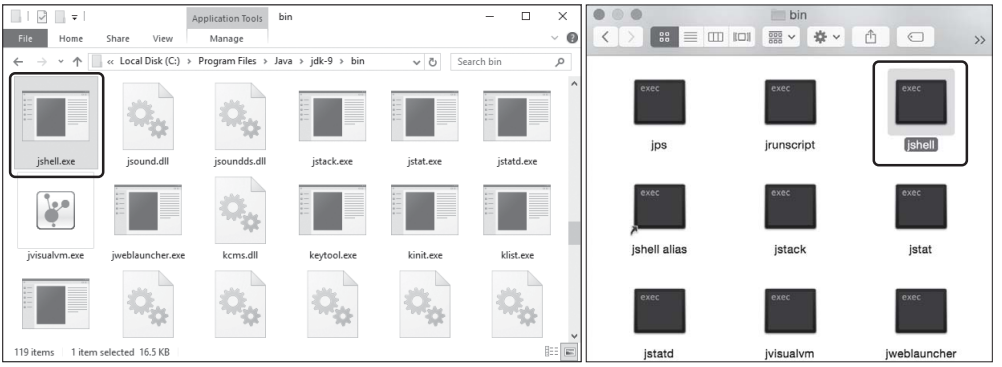


Рис. 1.10. Файл программы JShell в папке bin JDK 9 в Windows (слева) и macOS (справа)

Вы можете дважды щелкнуть мышью по значку для запуска JShell. Для удобства вы также можете создать ярлык для запуска JShell прямо на рабочем столе.

- В операционной системе Windows щелкните правой кнопкой мыши по файлу *jshell.exe* и выберите команду **Create shortcut** (Создать ярлык). Появится уведомление, что «Windows не может создать ярлык здесь. Вы хотите, чтобы ярлык был помещен на рабочий стол?» Нажмите кнопку **Yes** (Да).
- В операционной системе macOS, нажав и удерживая клавишу ∧, щелкните мышью по файлу *jshell* и выберите пункт **Make Alias** (Создать псевдоним) в контекстном меню. Появится файл с именем *jshell alias*. Перетащите его на рабочий стол.

Теперь каждый раз, когда вы хотите запустить JShell, просто дважды щелкните мышью по значку на рабочем столе, и вы будете готовы программировать на Java.

Работа с выражениями языка Java в JShell

Выражение представляет собой любую комбинацию операндов (например, числа или текст) и операторов, которые приводят к другому значению. *Оператор* выполняет операцию, такую как сложение, вычитание, умножение или деление. В языке Java символы, которые мы используем для этих операторов, соответственно, `+`, `-`, `*` и `/`. Давайте попробуем применить один из них с простым математическим выражением. В оболочке командной строки JShell введите **`2+2`** и нажмите клавишу **Enter**:

```
jshell> 2+2
$1 ==> 4
```

JShell сообщает ответ: значение выражения `2+2` равно 4. `$1` — это *временная переменная*. JShell создает эти переменные для хранения значений, в данном примере переменная `$1` временно сохраняет значение `4`, чтобы вы могли использовать его позже, если захотите. Если вы хотите узнать, какое значение находится в переменной, вы можете запросить его в JShell. Например, если вы сейчас введете **`$1`**, JShell скажет вам, что `$1` хранит значение `4`:

```
jshell> $1
$1 ==> 4
```

Давайте попробуем другое выражение. На этот раз давайте проведем операцию конкатенации (соединения) двух строк текста. *Строки* — это символы между кавычками, которые используются для отображения слов, имен и другого текста. Вы можете использовать для конкатенации оператор `+`:

```
jshell> "Your" + "Name"
$3 ==> "YourName"
```

Вы увидите, что в этом случае создается другая временная переменная — `$3`. Число после символа `$` указывает на строку кода, в которой было введено выражение. Это мой третий фрагмент кода, поэтому JShell хранит строку `"YourName"` в переменной `$3`.

Также обратите внимание, что при операции *конкатенации* строки соединяются без пробела между ними. Если нам нужен

пробел между двумя конкатенированными строками, мы должны добавить его, используя двойные кавычки, например:

```
jshell> "Your" + " " + "Name"
$4 ==> "Your Name"
```

В оболочке JShell можно вычислить любое допустимое выражение на языке Java. Попробуйте еще несколько выражений. Чтобы изменить введенную инструкцию, нажмите клавишу ↑. JShell отобразит последнюю введенную команду и позволит отредактировать ее. Затем нажмите клавишу **Enter**, чтобы снова запустить команду. Нажатие клавишу ↑ более одного раза позволит вам возвращаться от команды к команде вплоть до первой введенной вами строки.

Объявление переменных Java в JShell

Конечно, забавно вычислять выражения с простыми значениями, но, как правило, хочется хранить значения в переменных, чтобы впоследствии их использовать. Оболочка JShell автоматически создает временные переменные, например, `$1` и `$3` в предыдущем разделе, но также вы можете создавать или *объявлять* свои собственные переменные.

Числовые переменные

Давайте создадим целочисленную переменную с идентификатором (именем) `x` и сохраним в ней значение `42`. Для хранения целых чисел в языке Java следует использовать тип `int`, поэтому в приглашении JShell введите **`int x = 42`**:

```
jshell> int x = 42
x ==> 42
```

В языке Java знак равенства (=) является *оператором присваивания*. Это означает, что он используется для *присваивания* переменной значения. JShell сообщает нам, что переменная `x` содержит значение `42`. Когда мы захотим использовать это значение, мы можем просто вызвать переменную по имени. Давайте выясним, что **`x * 2`** (значение переменной `x` умножается на два) равно:

```
jshell> x * 2
$6 ==> 84
```

В языке Java символ звездочки (`*`) используется в качестве знака умножения, и JShell сообщает нам, что `x * 2` равно `84`. Изменили ли мы значение переменной `x`, получив ее значение и умножив это значение на два? Давайте узнаем, введя **`x`**:

```
jshell> x
x ==> 42
```

Наша переменная `x` по-прежнему равна `42`. Это означает, что мы можем использовать значение, хранящееся в переменной, без изменения переменной.

Как же нам изменить значение, хранящееся в переменной? Для этого нужно снова использовать оператор присваивания. Введите **`x = x + 7`** в оболочке командной строки JShell:

```
jshell> x = x + 7
x ==> 49
```

Мы *заменили* значение, хранящееся в переменной `x`, взяв старое значение `x` и добавив к нему `7`. С этого момента, каждый раз, когда мы будем запрашивать значение переменной `x`, мы получим в результате `49`, пока не изменим его снова. Значение переменной может меняться (*переменяться*) столько раз, сколько нам нужно, поэтому она и называется переменной.

Давайте рассмотрим несколько переменных разных типов. Мы уже рассмотрели пример с целочисленным значением. Теперь давайте попробуем десятичные числа или числа *с плавающей запятой*. Язык Java хранит десятичные числа в переменных типа `double` (сокращенно от *double-precision floating point* – число с плавающей запятой двойной точности), поэтому создайте переменную `meters` типа `double` и сохраните в ней десятичное число `1,83`:

```
jshell> double meters = 1.83
meters ==> 1.83
```

В качестве десятичного разделителя в программировании всегда используется точка. Язык Java обрабатывает десятичные числа

так же легко, как и целочисленные значения. Давайте займемся арифметикой и переведем метры в сантиметры:

```
jshell> double centimeters = meters * 100
centimeters ==> 183.0
```

Мы перевели метры в сантиметры, умножив значение на `100`.

Язык Java обрабатывает несколько других типов численных значений, но, как правило, чаще всего используются типы `int` и `double`. Каждый раз, когда вы сталкиваетесь с новым типом переменной, вы можете открыть JShell и поэкспериментировать с разными значениями.

Строковые переменные

Тип переменных `String` используется для хранения строк, состоящих из текстовых символов. Давайте определим переменную `myName` типа `String` и сохраним в ней имя автора следующим образом (вы можете использовать свое имя вместо моего):

```
jshell> String myName = "Bryson Payne"
myName ==> "Bryson Payne"
```

Мы используем знак равенства (=) в качестве оператора присваивания, так же как и с числовыми переменными.

Имена переменных, методов и классов в языке Java чувствительны к регистру. Имя `MYNAME` отличается от `myname`, и оба они отличаются от `myName`. В языке Java существует соглашение, то есть способ взаимодействия, об использовании так называемого *горбатого регистра*, когда прописной делают первую букву каждого нового слова в имени, например `myName` или `thisIsASillyNameButShowsCamelCase`, так что слова напоминают горбы верблюда. Имена классов также соответствуют горбатому регистру и начинаются с прописной буквы.

Теперь используем значение, хранящееся в переменной `myName`. Допустим, вы получили сертификат или ученую степень. Добавьте несколько символов после вашего имени:

```
jshell> myName + ", PhD"
$12 ==> "Bryson Payne, PhD"
```

Обратите внимание, что мы не использовали оператор присваивания, поэтому в переменной myName по-прежнему хранится ваше имя без дополнительных символов.

Теперь давайте изменим значение, хранящееся в переменной myName. Добавьте официальное обращение или приветствие к своему имени, как, например, вы бы сделали на конверте или в приглашении:

```
jshell> myName = "Dr. " + myName
myName ==> "Dr. Bryson Payne"
```

JShell отображает обновленное значение, хранящееся в переменной myName. Мы продолжим работу с числовыми и текстовыми переменными в следующем разделе, и вы научитесь выводить значения на экран из программы на языке Java.

Вывод на экран в языке Java

До этого момента мы узнавали значения выражений, просто введя их в JShell, но обычно, когда мы пишем реальные программы, это происходит следующим образом. В программе на языке Java мы ничего не увидим на экране по мере написания кода строка за строкой.

Когда мы хотим что-то отобразить на экране, мы используем функцию System.out.println(), которая выводит строку в системную консоль или на ваш экран. Если у вас еще открыто окно JShell из предыдущего раздела, вы можете вывести значение, хранящееся в переменной x, следующим образом:

```
jshell> System.out.println(x)
49
```

Если вы получите сообщение об ошибке, объявите новую переменную int x = 49 и снова запустите команду печати.

Обратите внимание, что на этот раз JShell не отвечает x ==> 49, потому что вы не просите его оценить выражение. Инструкция println() просит JShell распечатать только содержимое в круглых скобках, которое является значением переменной x, поэтому JShell просто отвечает 49.

Давайте попробуем вывести на экран строковую переменную. Введите следующую команду:

```
jshell> System.out.println("Hello, " + myName)
Hello, Dr. Bryson Payne
```

Пока у вас есть переменная с именем `myName` из предыдущего раздела, Java поприветствует вас по имени.

Каждый раз, когда вам потребуется вывести на экран информацию для пользователя, команда `System.out.println()` позволит вам напечатать именно то, что вы хотите вывести.

Команды JShell

Оболочка JShell настолько проста в использовании, что вы, возможно, никогда не захотите ее покинуть. Но в итоге вам нужно заняться другими делами, например создавать захватывающие настольные и мобильные приложения на языке Java. Давайте рассмотрим команды, которые может предложить JShell, включая команду выхода из JShell.

В оболочке командной строки JShell введите **`/help`**:

```
jshell> /help
```

Оболочка JShell ответит полным списком специальных команд, которые она распознает. Перед каждой командой мы ставим слеш (/), чтобы указать, что мы обращаемся непосредственно к программе JShell, а не к языку Java. Ниже приведена сокращенная версия команд, которые распознает JShell:

```
|  /list [<name or id>|-all|-start]    – выводит список указанных
                                                объектов
|  /edit <name or id>                  – редактирует указанную
                                                исходную запись
|  /save [-all|-history|-start] <file> – сохраняет указанный
                                                фрагмент кода в файл
|  /open <file>                        – открывает указанный файл
|  /vars [<name or id>|-all|-start]    – выводит список
                                          объявленных переменных и их значений
|  /imports                            – выводит список того, что
                                                было импортировано
|  /exit                               – выход из jshell
|  /reset                              – сброс jshell
```

```
|  /history                          — выводит историю того,
                                                что вы вводили
|  /help [<command>|<subject>]       — выводит информацию
                                          об указанном элементе jshell
```

Попробуйте выполнить некоторые команды, например `/list`, чтобы просмотреть весь исходный код, который вы вводили. Вы можете заметить, что JShell добавляет точки с запятой в конце строк, где это необходимо; точка с запятой является стандартным разделителем инструкций в языке Java. Команда `/history` показывает вам то, что вы вводили, включая команды, такие как `/help`, `/list` и даже `/history`.

Когда вы будете редактировать обычные программы на языке Java по мере работы с книгой, вы будете работать с файлами, часто сохраняя и открывая их, чтобы продолжить работу над ними. В JShell, однако, как только вы закрываете окно JShell, все, что вы набрали, теряется навсегда, если, конечно, вы не сохраните информацию. К счастью JShell предоставляет возможность сохранять, открывать и редактировать удачные фрагменты кода, которые вы написали. Чтобы сохранить код, созданный в JShell, используйте команду `/save` и укажите, где вы хотите сохранить файл:

```
jshell> /save ~/Desktop/имяфайла.txt
```

Тильда (~) означает пользовательскую директорию на вашем компьютере, поэтому команда сохранит в файле на рабочем столе весь код, который вы набрали с момента открытия JShell или с момента последнего применения команды `/reset`. Посмотрите на рабочий стол вашего компьютера, и вы увидите новый файл.

Чтобы открыть файл, используйте команду `/open` и укажите JShell путь до него:

```
jshell > /open ~/Desktop/имяфайла.txt
```

JShell откроет файл и запустит код.

Каждый раз, когда вы пишете фрагмент кода, который вы хотите сохранить для последующего использования, используйте команды `/save` и `/open`, чтобы сохранить код между сеансами JShell.

Чтобы начать новый фрагмент кода, используйте команду `/reset`. JShell помнит и сохраняет только то, что вы набираете после

команды `/reset`, но в любое время вы можете снова открыть файл, который сохранили. Попробуйте использовать следующий код для быстрого примера сохранения файла, сброса и открытия файла:

```
❶ jshell> /reset
  |  Resetting state.
❷ jshell> System.out.println("Hello, Java!")
  Hello, Java!
  jshell> System.out.println("My name is Bryson, nice to meet you!")
  My name is Bryson, nice to meet you!
❸ jshell> /save ~/Desktop/myJava.txt
❹ jshell> /reset
  |  Resetting state.
❺ jshell > /open ~/Desktop/myJava.txt
❻ Hello, Java!
  My name is Bryson, nice to meet you!
```

Цифры в окружностях обозначают важные строки и фрагменты, и не являются частью кода!

Во-первых, вам нужно сбросить все набранное в JShell командой ❶ для того, чтобы более ранний код не добавился к коду, который вы хотите сохранить. Как только вы это сделаете, вы можете написать свою программу, которая в этом случае будет состоять из двух команд `print` ❷, а затем сохранить ее на рабочем столе командой ❸. Даже после сброса JShell ❹, команда `/open` ❺ вызывает две строки кода, которые мы ввели перед командой `/save`, и запускает их ❻. Чтобы изменить код вы можете использовать команду `/edit` и затем команду `/save` для сохранения изменений.

JShell может хранить и вызывать фрагменты кода состоящие как из одной строки, так и из любого количества строк по вашему желанию.

Если вы закончили работу, используйте команду `/exit`, чтобы закрыть JShell:

```
jshell> /exit
Goodbye
```

Программа JShell очень вежливая и прощается с вами, когда вы из нее выходите.

Что вы изучили

В этой главе вы не только установили программное обеспечение Java, Eclipse и Android Studio, но также начали изучать язык Java, тестируя различные команды в интерактивной оболочке JShell. JShell — отличное дополнение к широко распространенным отраслевым стандартам языка Java и отличный инструмент для начинающих и опытных разработчиков.

Оболочка JShell устраняет некоторые излишние препятствия для обучения базовым командам языка Java и поощряет самостоятельную работу, что удобно для начинающих. Для опытных программистов программа JShell обеспечивает быстрое тестирование новых фрагментов кода и мгновенное отображение результатов на экране. Как педагог и программист, я очень рад перспективам, которые интерактивная оболочка JShell открывает для развития языка Java и для миллионов программистов, которые смогут воспользоваться преимуществами этого набора инструментов.

Вы настроили свои среды разработки для настольных и мобильных приложений на языке Java и научились проверять код в JShell. Вы также познакомились с рядом концепций программирования, который будете использовать при создании настольных и мобильных приложений на языке Java в следующих главах. Ниже представлен краткий обзор того, что вы сделали на данный момент:

- установили Java JDK версий 8 и 9;
- установили IDE Eclipse для разработчиков на языке Java и редактор WindowBuilder;
- установили IDE Android Studio для разработки мобильных приложений;
- запустили интерактивную оболочку JShell из командной строки и из папки *JDK9/bin*;
- вычислили результаты операций с числами и текстом на языке Java в JShell;
- объявили целочисленные, десятичные и строковые переменные в языке Java;
- вывели информацию на экран с помощью команды `System.out.println()`;
- использовали команды JShell, такие как `/reset`, `/edit`, `/save`, `/open` и `/exit`;
- сохранили и открыли файлы из оболочки JShell.

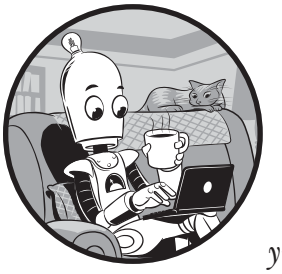
В следующей главе мы создадим наше первое полноценное приложение на языке Java — игру-угадайку «Больше-Меньше». В последующих главах мы создадим консольные, настольные и, наконец, мобильные (для операционной системы Android) версии каждой из наших программ.

Попутно мы будем использовать концепции программирования, которые мы освоили во время работы в JShell, включая выражения, переменные, потоки вывода и многое другое. Независимо от того, пишете ли вы первую или тысячную программу, настольное приложение для работы или мобильную игру для развлечения, вы будете использовать в своей программе эти понятия и концепции.

Вы получили инструменты, поэтому давайте перейдем к главе 2 и начнем!

Глава 2

РАЗРАБОТКА ИГРЫ «БОЛЬШЕ-МЕНЬШЕ»



Давайте приступим к программированию забавной игры «Больше-Меньше».

Мы создадим эту игру в виде *приложения, управляемого из оболочки командной строки**, то есть полностью текстовое (или консольное) приложение (см. рис. 2.1). После запуска прогаммы в командной строке пользователь получит предложение угадать число от 1 до 100. На каждый ответ пользователя программа выдаст, была ли догадка верна, либо больше или меньше загаданного числа.

* Также употребляется выражение «консольное приложение» (*прим. ред.*).

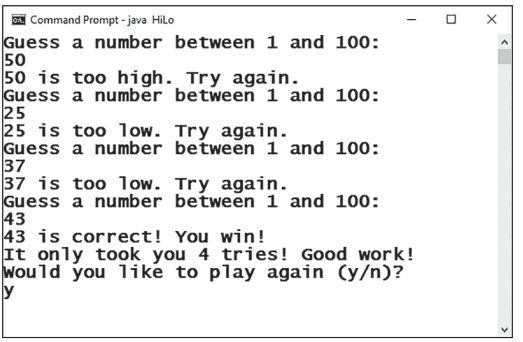


Рис. 2.1. Консольная версия игры «Больше-Меньше»

Поняв принцип игры, все, что вам нужно сделать, это написать код для ее реализации. Мы начнем с планирования приложения на общем уровне, а затем напишем программу для очень простой версии игры. Если вы начнете работать, имея в голове цель и понимая принцип работы приложения, вы сможете легче освоить навыки программирования, а также будете изучать их с конкретной целью. Вы сможете начать играть сразу же после того, как напишете код.

Планирование игры шаг за шагом

Давайте обдумаем все этапы написания кода, необходимые для того, чтобы наша игра «Больше-Меньше» заработала. Базовая версия игры должна делать следующее.

1. Сгенерировать случайное число от 1 до 100, которое пользователь будет угадывать.
2. Вывести на экран строку текста (*запрос*), попросив пользователя угадать число в указанном диапазоне.
3. Принять введенное пользователем число в качестве ответа.
4. Сравнить ответ пользователя с загаданным числом и определить их соотношение: больше, меньше или равно.
5. Отобразить результат сравнения на экране.
6. Просить пользователя дать ответ до тех пор, пока он не угадает число.
7. Спросить пользователя, хочет ли они снова сыграть в игру.

Мы пока ограничимся этой базовой структурой. В задаче № 2 в конце этой главы вы попытаетесь добавить дополнительную функцию, чтобы сообщить пользователю число попыток.

Создание нового проекта на языке Java

Первый шаг в разработке нового приложения на языке Java в среде разработки Eclipse состоит в создании проекта. В Eclipse выберите команду меню **File ⇒ New ⇒ Java Project** (Файл ⇒ Новый ⇒ Проект Java) (или выберите команду меню **File ⇒ New ⇒ Project** (Файл ⇒ Новый ⇒ Проект), а затем пункт **Java ⇒ Java Project** (Java ⇒ Проект Java) в окне создания нового проекта). Появится диалоговое окно **New Java Project** (Новый проект Java), как показано на рис. 2.2.

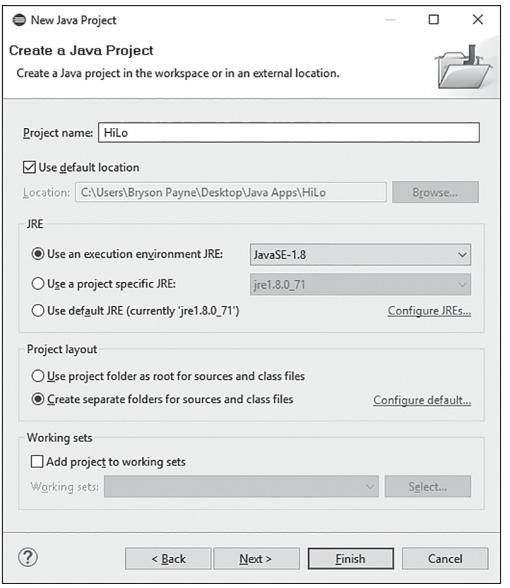


Рис. 2.2. Диалоговое окно **New Java Project** для приложения игры «Больше-Меньше»

Введите **HiLo** в поле **Project name** (Название проекта). Обратите внимание, что в языке Java важен регистр букв, и у вас должно войти в привычку называть с прописной буквы все ваши проекты, файлы и классы. Это является обычной практикой в языке Java. И мы будем использовать уже упомянутый горбатый регистр, так как *Hi* (higher — больше) и *Lo* (lower — меньше) — это два слова: *HiLo*. Оставьте все остальные настройки без изменений и нажмите кнопку **Finish** (Готово). В зависимости от вашей версии программы Eclipse может спросить, хотите ли вы открыть проект с помощью Java Perspective. Perspective в Eclipse — это рабочее пространство, настроенное для программирования на определенном языке.

Нажмите кнопку **Yes** (Да), и рабочая область Eclipse будет настроена для работы на языке Java.

Создание класса HiLo

Java — это *объектно-ориентированный язык программирования*. Объектно-ориентированные языки программирования используют *классы* для разработки фрагментов кода, которые будут использоваться многократно. Классы похожи на шаблоны, которые упрощают создание *объектов* или экземпляров этого класса. Если представить класс в виде формочки для печенья, то объекты как раз и являются печеньем. И, как и формочку, классы можно использовать повторно, поэтому, как только мы создадим класс, мы можем многократно его использовать для создания сколь угодно большого количества объектов.

В игре «Больше-Меньше» используется один файл класса, который будет создавать объект игры со всем необходимым для игрового процесса кодом. Мы назовем наш новый класс `HiLo`. Регистр имеет значение, и имя класса `HiLo` соответствует нескольким соглашениями по поводу имен в языке Java. Обычно все имена классов начинаются с прописной буквы, поэтому мы используем прописную `H` в `HiLo`. Кроме того, в имени класса не должно быть пробелов, дефисов или специальных символов. И последнее: мы используем горбатый регистр для имен классов, состоящих из нескольких слов, начиная каждое новое слово с прописной буквы в именах `HiLo`, `GuessingGame` и `BubbleDrawApp`.

Чтобы создать новый класс `HiLo`, сначала найдите папку проекта *HiLo* на панели **Package Explorer** (Обозреватель пакетов) в левой части окна Eclipse. Разверните папку, щелкнув мышью по маленькой стрелке слева от ее имени. Вы должны увидеть папку под названием *src*, сокращенно от *source code* (исходный код). Эта папка будет хранить всетекстовые файлы, содержащие ваши программы на языке Java.

Щелкните правой кнопкой мыши по папке *src* и выберите пункт **New ⇒ Class** (Создать ⇒ Класс), как показано на рис. 2.3.

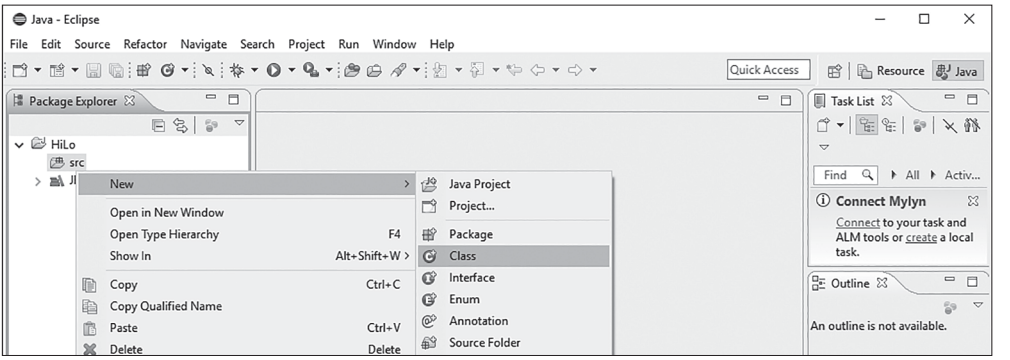


Рис. 2.3. Создание нового файла класса для приложения «Больше-Меньше»

Появится диалоговое окно **New Java Class** (Новый класс Java), как показано на рис. 2.4.

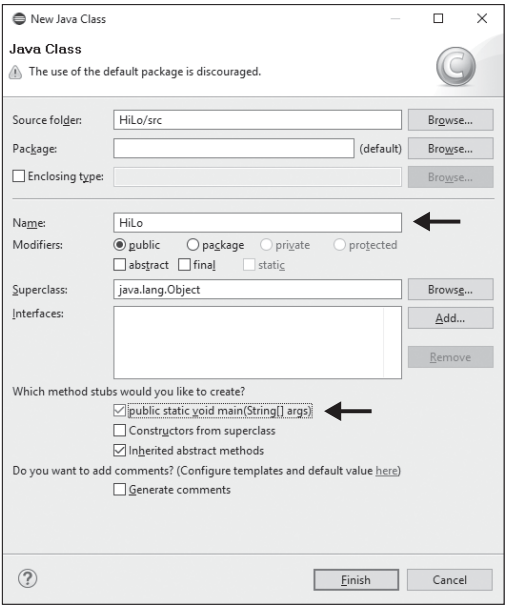


Рис. 2.4. Присвойте новому классу имя **HiLo** и установите выделенный флажок для создания метода `main()`

Введите **HiLo** в поле **Name** (Имя). Затем, под вопросом **Which method stubs would you like to create?** (Заглушку какого метода вы хотели бы создать?) установите флажок **public static void main (String [] args)**. Таким образом вы сообщаете программе Eclipse, что планируете написать метод `main()`, поэтому Eclipse подготовит *заглушку* или каркас для метода `main()`, которую вы можете заполнить вашим собственным кодом. *Методы* — это функции в объекте или классе. Если

вы хотите запускать приложение в качестве самостоятельной программы, то оно обязательно должно содержать метод `main()`.

Нажмите кнопку **Finish** (Готово) в диалоговом окне **New Java Class** (Новый класс Java), и вы увидите новый файл с именем *HiLo.java*, который содержит код, показанный в листинге 2.1. Этот файл на языке Java будет основой игры «Больше-Меньше». Мы напишем программу игры, отредактировав этот файл и добавив в него код.

```
❶ public class HiLo {
❷     public static void main(String[] args) {
          // Автоматически сгенерированная заглушка метода
      }
  }
```

Листинг 2.1. Код-заглушка для класса игры HiLo, сгенерированный программой Eclipse

Eclipse создает этот код самостоятельно. Класс `HiLo` является открытым (на что указывает слово `public`) ❶, то есть мы можем запустить его из оболочки командной строки.

Язык Java группирует инструкции с помощью фигурных скобок `{` и `}`. Открывающая скобка `{` начинает блок инструкций, которые будут формировать тело класса `HiLo`. Закрывающая скобка `}` завершает блок инструкций. Внутри класса находится метод `main()` ❷, который будет запускаться при выполнении класса.

Внутри скобок метода `main()` есть строка комментария, которая начинается с двух слешей, `//`. Комментарии игнорируются компьютером, поэтому мы можем использовать их, чтобы в будущем можно было легко вспомнить, что делает этот раздел кода. Вы можете удалить комментарий в листинге 2.1.

Генерация случайного числа

Первая задача программирования для нашей игры — сгенерировать случайное число. Мы будем использовать класс `Math`, который содержит метод, генерирующий случайные числа с плавающей запятой (десятичная дробь) в диапазоне от 0,0 до 1,0. Затем мы преобразуем эту десятичную дробь в *целое* число в диапазоне от 1 до 100. Класс `Math` является встроенным, и он содержит много полезных математических функций, таких же, как те, которые вы можете найти в хорошем научном калькуляторе.

Внутри метода `main()` добавьте комментарий и строку кода, как в листинге 2.2. (Новый код выделен черным цветом, а исходный — серым.)

```
public class HiLo {
    public static void main(String[] args) {
        // Генерация случайного числа, которое будет угадывать
           пользователь
        int theNumber = (int)(Math.random() * 100 + 1);
    }
}
```

Листинг 2.2. Код для генерации случайного числа в диапазоне от 1 до 100

Во-первых, нам нужно создать переменную для хранения загаданного случайного числа. Поскольку приложение попросит пользователя угадать целое число от 1 до 100, мы будем использовать тип `int` (сокращение от *integer* (целое число)). Назовем нашу переменную `theNumber`. Знак равенства (=) присваивает значение нашей новой переменной `theNumber`. Мы используем встроенную функцию `Math.random()` для генерации случайного числа в диапазоне от 0,0 и до почти 1,0 (0,99999). Поскольку метод `Math.random()` генерирует числа только в этом конкретном диапазоне, нам нужно умножить полученное случайное число на 100, чтобы растянуть диапазон от 0,0 до чуть меньше 100,0 (примерно 99,99999). Затем мы добавим 1 к этому значению, чтобы число попало в диапазон от 1,0 (0,0 + 1) до 100,99999.

Часть выражения `(int)` называется *приведением типа* или просто *приведением*. Приведение изменяет *тип* с десятичной дроби на целое число. В этом случае дробная часть после запятой отбрасывается, в результате получается целое число от 1 до 100. Затем это значение присваивается переменной `theNumber`, и именно его пользователь пытается угадать в игре. В конце строки мы ставим точку с запятой (`;`), чтобы зафиксировать конец команды.

Теперь вы можете добавить команду `System.out.println()`, чтобы вывести на экран сгенерированное число:

```
        int theNumber = (int)(Math.random() * 100 + 1);
        System.out.println(theNumber);
        }
}
```

После добавления этой строки кода мы можем запустить программу, чтобы увидеть, как она генерирует и выводит на экран случайное число. Нажмите зеленую кнопку запуска (**Run**) на панели в верхней части окна, чтобы скомпилировать и запустить программу, как показано на рис. 2.5. Вы также можете перейти в меню **Run** (Выполнить) и выбрать пункт **Run** (Выполнить).

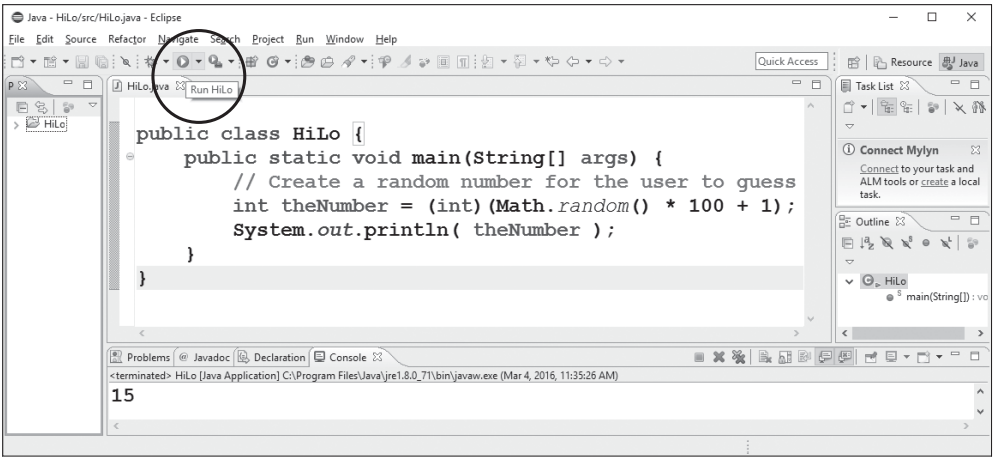


Рис. 2.5. Вывод случайного числа на экран

Ваше случайное число появится в маленьком окне консоли в нижней части экрана, как показано на рис. 2.5. Если вы снова запустите свою программу, то увидите другое число от 1 до 100.

Попробуйте поэкспериментировать с программой. Попытайтесь сгенерировать число от 1 до 10, или 1 и 1000, даже от 1 до 1 000 000. Язык Java будет обрабатывать числа до миллиарда или около того. Вероятно, вы не захотите угадывать число от 1 до 1 000 000 при первом запуске игры, так что не забудьте вернуть первоначальный код перед тем, как двигаться вперед.

Не забывайте периодически сохранять код. Среда разработки Eclipse автоматически сохраняет проект при каждом запуске вашей программы, но рекомендуется сохранять проекты самостоятельно по мере ввода кода. В принципе, нажатие сочетания клавиш **Ctrl+S** (или **⌘+S** в операционной системе macOS) для сохранения после каждой строки кода — замечательная привычка. Я никогда не слышал, чтобы программист сказал, что не хочет сохранять работу так часто, да я и сам несколько раз испытал досаду после

потери несохраненного кода, и это совсем невесело. Сохраняйте работу чаще и помните, что вы можете использовать команду **Edit ⇒ Undo** (Правка ⇒ Отменить), если вы ввели что-то неправильно или случайно удалили часть кода.

Обработка данных, введенных с клавиатуры

Теперь давайте добавим код, который позволит пользователю угадывать число. Для этого нам нужно *импортировать* некоторые дополнительные функциональные возможности языка Java. Java поставляется с большим количеством библиотек и пакетов, которые мы можем использовать в наших собственных проектах. Библиотеки и пакеты представляют собой наборы кода, созданные кем-то другим. Когда мы импортируем их, то получаем новые возможности, которые упрощают создание наших собственных программ. Мы можем обращаться к пакетам и библиотекам, когда нам это нужно, используя команду `import`.

В программе игры на угадывание необходимо иметь возможность принимать данные, введенные пользователем с клавиатуры. Класс `Scanner`, содержащийся в пакете утилит `java.util`, предоставляет несколько полезных функций для обработки ввода с клавиатуры. Мы импортируем класс `Scanner` в нашу программу. Добавьте следующую команду вверху файла `HiLo.java` перед объявлением класса `HiLo`:

```
import java.util.Scanner;

public class HiLo {
```

Эта строка импортирует класс `Scanner` и все его функции из основного пакета утилит Java. Класс Scanner включает такие функции как `nextLine()`, позволяющую принять строку, введенную с клавиатуры, и `nextInt()`, превращающую текст, введенный с клавиатуры, в целое число, которое можно сравнить или использовать в вычислениях. Чтобы использовать класс `Scanner` для обработки ввода с клавиатуры, мы должны указать ему использовать клавиатуру в качестве источника.

Это нужно сделать в самом начале программы, поэтому добавьте следующую строку кода в начало метода `main()`:

```
public class HiLo {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        // Генерация случайного числа, которое будет угадывать
        пользователь
        int theNumber = (int)(Math.random() * 100 + 1);
```

Эта строка создает объект `scan` класса `Scanner`, получающий данные с клавиатуры компьютера с помощью объекта `System.in`.

Хотя эта новая строка кода определяет объект `scan`, она еще не запрашивает ввод данных пользователем. Чтобы пользователь мог ввести свой ответ, нам нужно попросить его ввести число в оболочке командной строки. Затем мы возьмем число, которое он введет с клавиатуры, и сохраним его в переменной, которую мы можем сравнить с загаданным случайным числом `theNumber`. Давайте назовем переменную, которая будет хранить ответы пользователя так, чтобы ее имя легко запоминалось, например `guess`. Добавьте следующую строку кода:

```
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        // Генерация случайного числа, которое будет угадывать
        пользователь
        int theNumber = (int)(Math.random() * 100 + 1);
        System.out.println(theNumber);
        int guess = 0;
```

Эта команда сразу *объявляет* переменную типа `int` (целое число в Java) с именем `guess` и *инициализирует* переменную `guess` начальным значением 0. Некоторые языки программирования требуют, чтобы переменная была объявлена и затем инициализирована в отдельных строках кода, но язык Java позволяет программистам проводить и декларацию, и инициализацию переменных в одной строке. В языке Java каждая переменная должна быть объявлена с определенным *типом*. Поскольку пользователь будет вводить целые числа, мы объявили переменную `guess` типа `int`.

Теперь нам нужно предложить пользователю ввести ответ. Мы можем сообщить пользователю, что программа готова для ввода, напечатав строку текста в окне консоли. Мы получаем доступ к оболочке командной строки через класс `System`, как и для обработки ввода с клавиатуры. Но на этот раз мы хотим *вывести* информацию

для чтения пользователем. Объект, который позволяет нам получить доступ к оболочке командной строки для вывода, называется `System.out`. Подобно объекту `System.in`, который позволяет нам получать текстовый ввод с клавиатуры, `System.out` дает нам возможность выводить текст на экран. Специальной функцией для печати строки текста является команда `println()`:

```
int guess = 0;
System.out.println("Guess a number between 1 and 100:");
```

Здесь мы используем запись через точку (точечную нотацию), в которой перечислены класс или объект, за которым следует точка, а затем метод или атрибут этого класса или объекта. Методы должны вызываться с помощью точечной нотации, чтобы сообщить языку Java, к какому объекту или классу они принадлежат, например `Math.random()`. *Атрибуты* — это значения, хранящиеся в объекте или классе.

Класс `System` представляет вашу компьютерную систему. `System.out` — это объект командной строки экрана, содержащийся в классе `System`, потому что ваш монитор является частью общей компьютерной системы. `System.out.println()` — это метод вывода строки текста с использованием объекта `System.out`. В дальнейшей работе у вас будет возможность попрактиковаться в использовании точечной нотации.

Теперь пользователь знает, какие данные ожидает от него программа. Для получения ответа пользователя нам нужно проверить ввод с клавиатуры. Мы применим созданный ранее объект `scan` класса `Scanner`. Класс `Scanner` содержит метод `nextInt()`, который ожидает следующее значение типа `int`, введенное пользователем с клавиатуры. Мы будем хранить ответ пользователя в переменной `guess`:

```
System.out.println("Guess a number between 1 and 100:");
guess = scan.nextInt();
```

Эта команда ожидает, пока пользователь что-то напечатает в окне консоли (надеюсь, что целое число от 1 до 100 — мы рассмотрим способы проверки корректности введенных пользователем данных в главе 3) и нажмет клавишу **Enter**. Метод `nextInt()` примет строку текстовых символов, введенных пользователем (`"50"`, например), превратит ее в правильное числовое значение (`50`),

а затем сохранит это число в переменной `guess`. Обязательно сохраните изменения, которые вы сделали к этому моменту.

Создание потока вывода программы

Мы также можем провести дополнительную проверку работы программы, добавив еще одну команду `println()`:

```
guess = scan.nextInt();
System.out.println("You entered " + guess + ".");
```

Эта строка снова использует метод `System.out.println()`, но теперь мы комбинируем текстовый и числовой вывод. Если пользователь вводит ответ `50`, мы хотим вывести на экран: `"You entered 50"`. Для этого мы формируем команду `println()`, которая соединяет текст с числом, хранящимся в переменной `guess`.

Как мы уже знаем, язык Java позволяет конкатенировать строки текста с помощью оператора +. Мы используем двойные кавычки для обозначения текста, который мы хотим вывести первым (`"You entered"`). Обратите внимание на пробел перед закрывающими кавычками — он необходим, чтобы пробел отображался в строке вывода после последнего слова. Язык Java игнорирует большинство пробелов, но когда пробел включен в кавычки текстовой строки, он становится частью этого текста.

Мы также хотим вывести число, которое пользователь ввел в качестве ответа. Мы сохранили это значение в переменной, называемой `guess`, поэтому нам просто нужно использовать команду `println()` для вывода этого значения. Язык Java позволяет выводить значения переменных при помощи метода `println()`. Итак, сразу после текста `"You entered"` добавим оператор конкатенации (+), за которым следует имя переменной. И наконец, мы хотим закончить предложение точкой, поэтому мы используем еще один оператор конкатенации, а затем необходимый текст, содержащийся в двойных кавычках. Выглядит это как `" ."`.

Листинг 2.3 содержит все строки кода, написанные к этому моменту.

```
import java.util.Scanner;

public class HiLo {
```

```
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        // Генерация случайного числа, которое будет угадывать
           пользователь
        int theNumber = (int)(Math.random() * 100 + 1);
❶       // System.out.println(theNumber);
        int guess = 0;
        System.out.println("Guess a number between 1 and 100:");
        guess = scan.nextInt();
        System.out.println("You entered " + guess + ".");
    }
}
```

Листинг 2.3. Программа к этому моменту генерирует случайное число и дает пользователю возможность угадать один раз

Обратите внимание, что я добавил строку ❶, содержащую код `System.out.println(theNumber)` и два слеша в начале. Благодаря этим слешам строка *закомментирована*, и это полезный прием для *отладки* — поиска и исправления ошибок в программах. Ранее мы использовали команду `println()`, чтобы вывести значение переменной `theNumber` во время написания и тестирования программы. Теперь, вместо того чтобы удалять строку, мы можем ее закомментировать, и компьютер ее проигнорирует. Если мы захотим снова использовать эту строку, мы можем просто удалить символы `//`, чтобы включить ее в программу.

Теперь мы сохраним нашу программу и запустим ее. Для запуска программы нажмите зеленую кнопку запуска на панели инструментов или выберите команду меню **Run** ⇒ **Run** (Выполнить ⇒ Выполнить). Прямо сейчас пользователь может ввести только один ответ, и программа не проверяет правильность его догадки. Итак, далее мы добавим код, чтобы пользователь мог угадывать более одного раза, а затем узнаем, как сравнивать ответы пользователя с переменной `theNumber`.

Циклы: условие, проверка, повтор

Чтобы предоставить пользователю больше одной попытки угадать число, нам нужно научиться создавать циклы. В программе, реализующей игру «Больше-Меньше», нам необходимо запрашивать ответ

пользователя до тех пор, пока он не угадает случайное число, загаданное компьютером. *Циклы* предоставляют возможность повторять определенную последовательность действий снова и снова. В этом разделе мы напишем цикл, который будет запрашивать у пользователя ответ и обрабатывать данные, введенные с клавиатуры.

Циклы являются очень мощными инструментами и предоставляют огромные возможности в программировании. Они являются одной из причин того, что компьютеры настолько ценны в нашей повседневной жизни и в мире бизнеса — машины могут предсказуемо выполнять многократные одинаковые действия. И если они запрограммированы правильно, то могут делать это без перерыва, каждый день, не допуская ошибок. Вы или я, возможно, устали бы говорить кому-то, что их ответ больше или меньше загаданного числа, но компьютер никогда не устает. Он также никогда не забудет загаданное число и не сообщит игроку ошибочный ответ.

Давайте познакомимся с возможностями циклов на примере цикла `while`. Цикл `while` повторяет набор команд, пока выполняется заданное *условие*. Условие — это то, что мы можем проверить. Например, в этой программе мы хотим узнать, угадал ли пользователь загаданное машиной число. Если он не угадал, то мы хотим предоставлять ему возможность для новой попытки до тех пор, пока он не введет правильный ответ.

Чтобы написать цикл `while`, нам нужно знать, какое условие мы будем проверять перед тем, как повторить цикл. В нашей игре мы хотим, чтобы пользователь вносил ответ до тех пор, пока переменная `guess` не будет равна переменной `theNumber`. Если ответ пользователя равен загаданному числу, пользователь выигрывает, и игра заканчивается, поэтому цикл должен быть остановлен.

Чтобы создать цикл `while`, нам нужно вставить инструкцию `while` перед последними тремя строками кода, а затем обернуть три строки, в которых пользователь угадывает число, новой парой фигурных скобок, как показано ниже:

```
        int guess = 0;
        while (guess != theNumber) {
        System.out.println("Guess a number between 1 and 100:");
        guess = scan.nextInt();
        System.out.println("You entered " + guess + ".");
        }
    }
}
```

Мы используем ключевое слово `while`, чтобы создать цикл, а затем помещаем соответствующее условие в круглые скобки. Часть внутри скобок `guess != TheNumber` означает, что, пока значение переменной `guess` не равно (`!=`) значению переменной `TheNumber`, цикл должен повторять любую команду или набор команд, написанных сразу же после этой строки кода. `!=` — *оператор сравнения*, который в нашем случае сравнивает значения переменных `guess` и `TheNumber` и оценивает, являются ли они разными, т. е. *не равными*. Вы изучите другие операторы сравнения в следующем разделе.

Нам нужно сообщить языку Java, какие команды должны повторяться в цикле `while`, поэтому я добавил открывающуюся фигурную скобку `{` после инструкции `while`. Точно так же, как скобки группируют все команды в методе `main()`, эти скобки группируют команды внутри цикла `while`.

Мы хотим заключить в цикл три команды. Сначала нам нужна команда `println()`, которая предложит пользователю ввести ответ. Затем нам понадобится команда, которая получает ввод с клавиатуры и сохраняет ответ с помощью метода `nextInt()`. И наконец, нам нужна команда `println()`, которая сообщает пользователю ответ. Чтобы превратить этот набор команд в блок кода, который будет повторяться в рамках цикла `while`, сначала пишем инструкцию `while` и условие, затем открывающую фигурную скобку, после этого все три инструкции и, наконец, закрывающую фигурную скобку. Не забудьте закрыть скобку! Ваша программа не будет работать, если вы ошибетесь со скобками.

Правильное использование отступов — хороший прием в программировании, который поможет вам сохранить код организованным и читаемым. Выделите три команды внутри фигурных скобок цикла `while`, а затем нажмите клавишу **Tab**, чтобы сместить их.

Результат должен выглядеть следующим образом:

```
        int guess = 0;
        while (guess != theNumber) {
            System.out.println("Guess a number between 1 and 100:");
            guess = scan.nextInt();
            System.out.println("You entered " + guess + ".");
            }
        }
}
```

Правильная организация отступов поможет вам сопоставлять открывающиеся и закрывающие фигурные скобки. Также она поможет вам быстро определить, какие инструкции находятся внутри цикла или другого блока кода, а также какие инструкции находятся вне цикла. Отступы не влияют на работу вашей программы, но если они оформлены верно, то существенно упрощают чтение и обслуживание вашей программы.

Сохраните свою программу и запустите ее, чтобы убедиться в ее работе. К этому моменту игра почти готова, но нам все еще нужно дописать программу, чтобы она проверяла больше, меньше или равен ответ пользователя загаданному числу. Наступило время инструкции `if`.

Инструкции `if`: проверка правильности условий

Сейчас пользователь может вносить ответ до тех пор, пока не угадает. Теперь нам нужно проверять его ответы, чтобы сообщить, были ли они больше или меньше загаданного числа. Для этого мы применим инструкцию `if`.

Инструкция `if` будет решать, следует ли запускать блок команд один или несколько раз на основе условия или *условного выражения*.

Ранее мы использовали условное выражение в цикле: (`guess != TheNumber`). Чтобы проверить, больше или меньше загаданного числа ответ пользователя, нам нужны еще несколько операторов сравнения: меньше (<), больше (>) и равно ли (==).

Для начала, вместо того, чтобы просто сообщить пользователю его ответ, давайте напишем какой-нибудь код, чтобы проверить, был ли ответ игрока меньше загаданного числа. Замените последнюю строку в цикле `while` следующими двумя строками с инструкцией `if`:

```
while (guess != theNumber) {
    System.out.println("Guess a number between 1 and 100:");
    guess = scan.nextInt();
    if (guess < theNumber)
        System.out.println(guess + " is too low. Try again.");
}
```

Инструкция начинается с ключевого слова `if`, за которым следует условное выражение в круглых скобках. В этом случае условием

является выражение `guess < theNumber`, означающее, что ответ пользователя меньше загаданного числа. Обратите внимание, что после круглых скобок нет точки с запятой, потому что следующая команда `println()` входит в состав команды `if`. Вся команда указывает программе, что если условие истинно, то нужно вывести ответ пользователя и сообщить, что он меньше загаданного числа. Мы используем оператор конкатенации (+) между ответом пользователя и строкой текста, сообщающей, что число в ответе было меньше загаданного. Обратите внимание на пробел между открывающейся двойной кавычкой и текстом `is too low`. Таким образом, ответ пользователя будет отделен пробелом от текста `is too low`.

Если вы запустите программу сейчас и введете заведомо меньший ответ, например 1, инструкция `if` должна сообщить программе, что ваша догадка меньше. Это хорошо для начала, но что, если мы введем слишком большое число? В этом случае нам понадобится ключевое слово `else`.

Инструкция `else` предоставляет программе возможность выбора альтернативного пути или набора шагов, если условие в выражении `if` неверно. С помощью конструкции `if-else` мы можем проверить ответ пользователя и установить, больше он или меньше загаданного числа. Давайте добавим инструкцию `else` сразу после инструкции `if`:

```
❶     if (guess < theNumber)
          System.out.println(guess + " is too low. Try again.");
❷     else if (guess > theNumber)
          System.out.println(guess + " is too high. Try again.");
```

Обратите внимание, что код конструкции ❷ похож на код фрагмента ❶. Часто, когда мы используем конструкцию `if-else`, нам нужно проверить не одно, но несколько условий подряд. В данном случае нам нужно проверить, больше ли ответ пользователя, меньше или совпадает с загаданным числом. В таких случаях мы можем выстроить цепочку из условий `if-else`, поместив следующую инструкцию `if` внутри части `else` предыдущей конструкции `if-else`. В фрагменте ❷ мы начали следующую инструкцию `if` сразу после `else` из предыдущего условия. Если ответ больше загаданного числа, программа сообщает пользователю, что его ответ больше. Теперь программа может сказать пользователю, больше или меньше его догадка. Нам осталось только добавить сообщение о том, что пользователь правильно угадал и выиграл!

Если ни одно из предыдущих условий не является истинным, ответ пользователя не больше и не меньше, тогда он должен был угадать число. Итак, добавим еще одно, последнее ключевое слово `else`:

```
❶    if (guess < theNumber)
         System.out.println(guess + " is too low. Try again.");
❷    else if (guess > theNumber)
         System.out.println(guess + " is too high. Try again.");
❸    else
         System.out.println(guess + " is correct. You win!");
```

Обратите внимание, что нам не нужно условное выражение для этого последнего ключевого слова `else`. Правильный ответ — единственный оставшийся вариант, если число не больше и не меньше загаданного. В случае правильного ответа мы выводим на экран сообщение о выигрыше пользователя. Полная программа до этого момента показана в листинге 2.4. Сохраните файл *HiLo.java* и запустите программу, чтобы проверить, что она работает. Она должна предлагать вам вводить ответы, пока вы не угадаете загаданное компьютером число.

```
import java.util.Scanner;

public class HiLo {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        // Генерация случайного числа, которое будет угадывать пользователь
        int theNumber = (int)(Math.random() * 100 + 1);
        // System.out.println(theNumber);
        int guess = 0;
        while (guess != theNumber) {
            System.out.println("Guess a number between 1 and 100:");
            guess = scan.nextInt();
            if (guess < theNumber)
                System.out.println(guess + " is too low. Try again.");
            else if (guess > theNumber)
                System.out.println(guess + " is too high. Try again.");
            else
                System.out.println(guess + " is correct. You win!");
```

```
            } // Конец цикла угадывания while
        }
    }
```

Листинг 2.4. Игра «Больше-Меньше» готова сыграть с вами один раз

Теперь программа представляет собой игру, в которую уже можно играть! После того как пользователь вводит правильный ответ, программа сообщает, что он выиграл, а затем игра заканчивается, как показано на рис. 2.6.

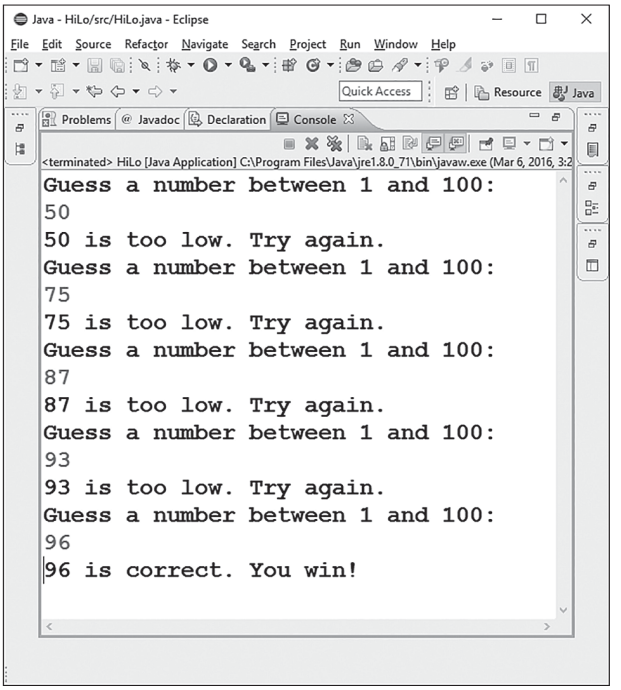


Рис. 2.6. Один полный раунд игры «Больше-Меньше» — программа заканчивается, когда пользователь угадывает число

Пожмите себе руку! Вы создали программу на языке Java с нуля. Если это ваша первая программа на Java, вы заслуживаете вознаграждения!

Поиграйте с машиной несколько раз. Угадываете ли вы число быстрее с каждой новой игрой? Проверьте свою программу, чтобы убедиться, что она работает так, как вы хотите. В следующем разделе мы ее улучшим.

В данный момент единственный способ снова сыграть в игру — это перезапустить программу в Eclipse. К счастью, мы уже знаем, что есть способ заставить нашу программу делать что-то снова и снова — нам нужен еще один цикл!

Программа игры «Больше-Меньше» заканчивается, когда пользователь угадывает число, потому что после цикла `while` ничего не происходит. Цикл `while` заканчивается, когда условие (`guess != TheNumber`) больше не является истинным. Пользователь может захотеть играть снова, как только он завершает игру. Для этого цикла повторного воспроизведения игры нам потребуется новая инструкция и новый тип цикла — `do-while`.

Как и цикл `while` цикл `do-while` повторяет блок инструкций, пока условие истинно. Однако, в отличие от цикла `while`, блок кода внутри цикла `do-while` гарантированно выполняется хотя бы один раз. Бывают случаи, когда условие в начале цикла `while` будет ложным еще до начала цикла, поэтому весь цикл и все строки кода внутри него будут пропущены. Условие цикла `while` можно представить в виде термостата на обогревателе. Если в помещении уже достаточно тепло и условие включения нагревателя не выполняется, он может вообще не включаться.

Для нашей игры, как и для почти любой игровой программы, мы выбираем цикл `do-while` (иногда мы называем его *игровым циклом*), потому что пользователь, вероятно, захочет сыграть хотя бы один раз. Кроме того, обычно мы спрашиваем пользователя, хочет ли он снова играть, а пользователь обычно отвечает да или нет (или `y` или `n` в консольной игре, как эта). Игра будет повторяться в цикле, пока пользователь отвечает да.

Чтобы проверить ответ пользователя, нам понадобится еще один тип переменной: `String` (строка). Строки — это объекты, которые содержат текст в двойных кавычках, например `"y"`, `"yes"` или `"Меня зовут Брайсон! Надеюсь, вам понравилась моя игра!"`. Раньше мы использовали целочисленную переменную или тип `int` для хранения чисел, которые пользователь угадывал. Теперь нам нужно сохранить текст, поэтому мы будем использовать переменную типа `String`. Мы можем добавить переменную `String` в начало программы сразу после настройки `Scanner`:

```
Scanner scan = new Scanner(System.in);
String playAgain = "";
```

Обратите внимание, что слово `String` начинается с прописной буквы `S`. Это связано с тем, что тип `String` на самом деле является классом, который содержит функции для работы со строками текста. Я назвал переменную `playAgain`, используя горбатый регистр, с большой буквой `A` в начале второго слова. Помните, что в именах переменных не допускаются пробелы. И точно так же, как мы задали начальное значение 0 переменной `guess` путем задания `int guess = 0`, в данном случае мы задали начальное значение переменной `playAgain` таким образом: `playAgain = ""`. Пара двойных кавычек без пробелов между ними означает пустую строку, или переменную типа `String` без текста. Позже мы назначим другое текстовое значение этой переменной, после того, как пользователь введет `y` или `n`.

Как и в цикле `while`, нам потребуется проверить, какие команды должны повторяться в цикле `do-while`. Он будет нашим основным циклом, поэтому почти все команды программы попадут внутрь него. Фактически все остальные команды после `Scanner` и `String playAgain` будут внутри цикла `do-while`. Эти шаги описывают один полный раунд игры, поэтому в каждом раунде эти шаги будут повторяться снова, от генерации нового случайного числа до объявления правильного ответа и запроса пользователя о повторной игре.

Таким образом, мы можем добавить ключевое слово `do` и открывающую скобку сразу после первых двух строк и перед кодом, генерирующим случайное число:

```
Scanner scan = new Scanner(System.in);
String playAgain = "";
do {
    // Генерация случайного числа, которое будет угадывать пользователь
    int theNumber = (int)(Math.random() * 100 + 1);
    // System.out.println(theNumber);
    int guess = 0;
    while (guess != theNumber) {
```

Затем, после закрытия скобки для цикла `while`, использующегося для угадывания и скобки после нашей последней инструкции `else`, мы спрашиваем пользователя, хочет ли он играть снова и получаем ответ, введенный с клавиатуры. После этого нам нужно закрыть цикл `do-while` с условием `while`, проверяющим, ответил ли пользователь согласием:

```
            } // Конец цикла угадывания while
❶           System.out.println("Would you like to play again (y/n)?");
❷          playAgain = scan.next();
❸         } while (playAgain.equalsIgnoreCase("y"));
❹     }
❺ }
```

Команда ❶ выдает в консоли запрос, хочет ли пользователь сыграть снова — `"Would you like to play again (y/n)?"`, на что он может ответить одной буквой, `y` (да) или `n` (нет). В строке ❷ функция `scan.next()` сканирует ввод с клавиатуры, но вместо того, чтобы искать следующее целое число как функция `nextInt()`, она ищет следующий символ или группу символов, которые пользователь вводит на клавиатуре. Все, что введет пользователь, будет храниться в переменной `playAgain`.

В строке ❸ скобкой закрывается блок кода, который повторяет игру, а также содержится условие `while`, которое определяет, будет ли код запущен снова. В условии `while` вы можете увидеть пример метода `equals()` объекта `String`. Метод `equals()` сообщает, является ли строковая переменная точно такой же, как и другая строка символов, а метод `equalsIgnoreCase()` сообщает, равны ли строки, вне зависимости от регистра содержащихся в них букв. В нашей игре, если пользователь хочет снова сыграть, ему предлагается ввести `y`. Однако если мы просто проверим маленькую букву `y`, мы можем пропустить ответ, данный в верхнем регистре — `Y`. В данном случае мы хотим проверить ввод буквы `y`, независимо от регистра, поэтому используем строковый метод `equalsIgnoreCase()`.

Последняя команда `while` говорит Java продолжать цикл игры, пока строковая переменная `playAgain` равна прописной или строчной буквой `y`. Две последние закрывающие скобки в строках ❹ и ❺ уже были в программе. Скобка в строке ❹ закрывает метод `main()`, а в строке ❺ закрывает весь класс `HiLo`. Я включил их, чтобы показать, где должны быть вставлены строки ❶–❸.

Полный код игры на данный момент приведен в листинге 2.5.

```
import java.util.Scanner;
    public class HiLo {
        public static void main(String[] args) {
            Scanner scan = new Scanner(System.in);
            String playAgain = "";
            do {
```

```
            // Генерация случайного числа, которое будет угадывать пользователь
            int theNumber = (int)(Math.random() * 100 + 1);
            // System.out.println(theNumber);
            int guess = 0;
            while (guess != theNumber) {
                System.out.println("Guess a number between 1 and 100:");
                guess = scan.nextInt();
                if (guess < theNumber)
                    System.out.println(guess + " is too low. Try again.");
                else if (guess > theNumber)
                    System.out.println(guess + " is too high. Try again.");
                else
                    System.out.println(guess + " is correct. You win!");
            } // Конец цикла угадывания while
            System.out.println("Would you like to play again (y/n)?");
            playAgain = scan.next();
        } while (playAgain.equalsIgnoreCase("y"));
    }
}
```

Листинг 2.5. В игру «Больше-Меньше» можно играть бесконечно

Просмотрите свой код и убедитесь в правильности всех инструкций, проверьте фигурные скобки и точки с запятой и сохраните свой файл. Мы протестируем игру в следующем разделе.

Отступы в начале каждой строки могут выглядеть не так, как в последнем фрагменте кода, поскольку мы добавили фигурные скобки в паре мест. К счастью, добавление новых функций, в том числе циклов и других блоков кода, настолько распространено в Java, что в Eclipse есть пункт меню для автоматического сброса отступов. Сначала выберите (выделите) весь текст на экране в файле *HiLo.java*. Затем выберите команду меню **Source ⇒ Correct Indentation** (Исходный код ⇒ Исправить отступы). Eclipse создаст правильные отступы для каждой строки кода, тем самым показав, какие инструкции должны быть сгруппированы вместе. Как я уже говорил, отступы не имеют значения для компьютера (программа будет работать точно так же совсем без отступов или дополнительных пробелов), но правильные отступы и пробелы помогают облегчить чтение кода программы.

Тестирование игры

Выше мы закончили код игры. Не забудьте сохранить файл *HiLo.java*. Теперь выберите команду меню **Run ⇒ Run** (Выполнить ⇒ Выполнить), чтобы протестировать вашу программу. После того, как вы угадаете первое случайное число, программа должна спросить вас, хотите ли вы сыграть еще раз. Пока вы будете отвечать **y** (или **Y**) и нажимать клавишу **Enter**, программа должна продолжить загадывать новые случайные числа и предлагать вам их отгадать. На снимке экрана на рис. 2.7 обратите внимание, что игра начинается заново, когда я отвечаю **y** в оболочке командной строки.

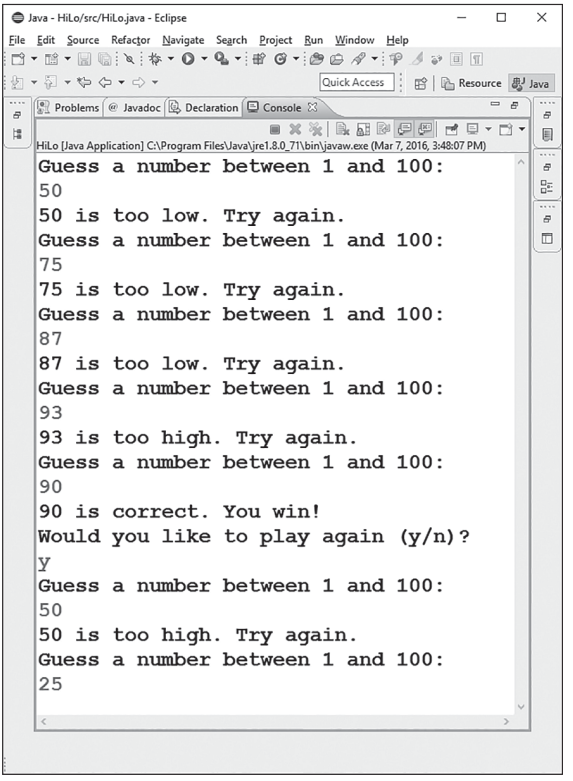


Рис. 2.7. Можно играть несколько раз подряд, пока пользователь отвечает y или Y

Когда пользователь заканчивает игру и на вопрос об игре заново отвечает `n` или что угодно кроме `y` или `Y`, игра заканчивается. Однако мы могли бы поблагодарить его за игру после ее окончания. Добавьте следующую строку после последнего выражения `while`, перед двумя последними закрывающими скобками:

```
        } while (playAgain.equalsIgnoreCase("y"));
        System.out.println("Thank you for playing! Goodbye.");
    }
}
```

Наконец, последняя строка, которую мы добавим в приложение, адресована предупреждению, которое вы, возможно, заметили в Eclipse. Это предупреждение отображается в виде светло-желтой строки под объявлением объекта `scan`, а также желтого треугольника с восклицательным знаком слева от этой строки. Eclipse обращает наше внимание на то, что мы открыли ресурс, но не закрыли его. В программировании это может создать так называемую *утечку ресурсов*. Обычно это не имеет значения, если мы просто открываем один объект `Scanner` для ввода с клавиатуры, но если мы оставим несколько объектов `Scanner` открытыми, не закрыв их, память программы может переполниться, что приведет к замедлению работы или даже сбою всей системы. Мы сообщим программе о том, что можно закрыть соединение с клавиатурой, с помощью метода `close()` класса `Scanner`.

После инструкции `println()`, в которой благодарите пользователя за игру, но перед последними двумя закрывающими скобками добавьте следующую команду:

```
        System.out.println("Thank you for playing! Goodbye.");
        scan.close();
    }
}
```

Вы заметите, что желтое предупреждение исчезнет из окна редактора Eclipse после добавления этой строки. Eclipse помогает избегать распространенных ошибок программирования, таких как опечатки или отсутствующие знаки пунктуации, и даже предупреждает нас о проблемах, которые *могут* возникнуть, как, например, утечка ресурсов или неиспользуемые переменные. По мере того как вы будет разрабатывать на Java более сложные приложения, эти возможности IDE окажутся еще ценными для вас. В приложении можно найти дополнительную информацию об использовании Eclipse для отладки ваших программ.

Законченная программа, показанная в листинге 2.6, представляет собой полностью готовую игру, которая позволяет играть и раз за разом угадывать новое случайное число.

```
import java.util.Scanner;

public class HiLo {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        String playAgain = "";
        do {
            // Генерация случайного числа, которое будет угадывать пользователь
            int theNumber = (int)(Math.random() * 100 + 1);
            // System.out.println(theNumber);
            int guess = 0;
            while (guess != theNumber) {
                System.out.println("Guess a number between 1 and 100:");
                guess = scan.nextInt();
                if (guess < theNumber)
                    System.out.println(guess + " is too low. Try again.");
                else if (guess > theNumber)
                    System.out.println(guess + " is too high. Try again.");
                else
                    System.out.println(guess + " is correct. You win!");
            } // Конец цикла угадывания while
            System.out.println("Would you like to play again (y/n)?");
            playAgain = scan.next();
        } while (playAgain.equalsIgnoreCase("y"));
        System.out.println("Thank you for playing! Goodbye.");
        scan.close();
    }
}
```

Листинг 2.6. Готовая консольная текстовая игра

Отметим еще несколько моментов в законченной программе игры «Больше-Меньше». Во-первых, несмотря на всю проделанную работу по его написанию, код игры получился относительно коротким — менее 30 строк. Тем не менее вы можете играть в нее бесконечно, если захотите. Во-вторых, эта программа не только демонстрирует использование условий и циклов, но даже использует цикл внутри другого цикла. Это называется *вложенным циклом*, поскольку цикл угадывания находится, или вложен, в цикл бесконечной игры. Отступы помогают нам увидеть, где цикл `do-while` начинается и заканчивается, а также мы можем видеть, что меньший цикл

while и его инструкции if вложены с помощью отступов в большой цикл do-while. Наконец, мы аккуратно заканчиваем программу — как с точки зрения пользователя, благодаря его за игру, так и с точки зрения компьютера, закрывая ресурс сканера.

Что вы изучили

В процессе создания простой и забавной игры мы познакомились с несколькими ценными концепциями программирования. Именно так, как ребенок, я бы учился программировать — я бы выбрал забавную игру или графическое приложение, написал бы для него программу, затем изменил бы ее и попробовал бы новые вещи. Игра и исследование — важные части изучения чего-либо нового, и я надеюсь, что вы найдете что-то новое в каждой программе. Дополнительные задачи в конце каждой главы также дадут вам возможность попробовать несколько новых вещей.

Создавая эту игру, мы развили большой спектр навыков программирования на Java:

- создание нового класса, HiLo;
- импорт существующего пакета Java java.util.Scanner;
- использование объекта Scanner для ввода данных с клавиатуры;
- объявление и инициализация целочисленных и строковых переменных;
- генерация случайного числа с помощью метода Math.random() и его приведение к целочисленному типу;
- использование циклов while и do-while для повторения набора шагов до тех пор, пока выполняется определенное условие;
- вывод текстовых строк и значений переменных в оболочке командной строки;
- ввод целых чисел и строк с клавиатуры и их сохранение в переменных;
- тестирование различных условных выражений в инструкциях if и if-else;
- использование методов String для сравнения строковых значений с помощью метода equalsIgnoreCase();
- закрытие входных ресурсов, таких как объекты Scanner с помощью метода close();
- запуск консольной программы из Eclipse.

Помимо практических навыков, вы также получили знания о нескольких важных концепциях программирования на Java.

Переменные. `theNumber` — целочисленная переменная типа `int`. `guess.playAgain` — строковая переменная типа `String`. Мы изменяем значение этих переменных, когда пользователь вводит ответы во время игры.

Методы. Методами мы называем функции, принадлежащие классам в Java. `Math.random()` — метод генерации случайных чисел между 0,0 и 1,0. Метод `scan.nextInt()` принимает число, введенное пользователем. `System.out.println()` — это функция для отображения текста в оболочке командной строки.

Условные выражения. Инструкция `if-else` позволяет нам проверить, истинно ли условие, например, `guess < theNumber`, и запускает нужный блок кода в зависимости от результата проверки. Кроме того, мы используем условные выражения, чтобы определить, следует ли снова выполнять цикл, как в выражении `while (guess != theNumber)`. Эта инструкция будет запускать новый цикл до тех пор, пока значение переменной `guess` не будет равно значению переменной `theNumber`. Помните, что проверка на «равно» — это двойной знак равенства: ==.

Циклы. Цикл `while` позволяет нам повторять блок кода до тех пор, пока условие истинно. Мы использовали цикл `while` в игре для того, чтобы пользователь мог вводить ответы, пока не угадает число. Цикл `do-while` всегда выполняется хотя бы один раз; мы использовали его, чтобы спросить пользователя, хочет ли он играть заново.

Классы. Вся программа HiLo представляет собой открытый класс `HiLo`. Класс — это шаблон. Единожды создав шаблон класса для игры «Больше-меньше», мы можем повторно использовать его и играть на разных компьютерах. Кроме того, мы импортировали в наше приложение классы `Scanner` и `Math`, чтобы пользователь мог вводить данные с клавиатуры, а компьютер — генерировать случайные числа. Мы будем писать наши собственные классы для решения новых задач, а также будем использовать классы, уже включенные в Java, для выполнения повседневных целей, таких как ввод, математические функции и многого другого.

Дополнительные задачи

Попробуйте выполнить следующие задания. Таким образом вы повторите пройденный материал, попрактикуетесь в программировании и получите новые навыки. Если у вас возникнут проблемы, вы можете загрузить примеры решений на сайте по адресу **https://eksmo.ru/files/learn_java_easy_way.zip** либо посмотреть видеокурс по адресу **www.udemy.com/java-the-easy-way/** и ознакомиться с пошаговыми решениями.

Задача № 1: Расширение диапазона

Измените игру «Больше-Меньше» таким образом, чтобы можно было использовать больший диапазон чисел. Вместо числа от 1 до 100 предложите пользователю угадать число в диапазоне от –100 до 100!

§ Умножьте `Math.random()` на `200` и вычтите `100` из результата.

Не забудьте изменить как команду, генерирующую случайное число, так и приглашение, которое сообщает пользователю диапазон, в рамках которого он должен угадывать число.

Если вы хотите облегчить игру, вы можете задать диапазон от 1 до 10 и удивить своих друзей, когда вы сможете угадать загаданное число всего за четыре попытки. Попробуйте задать другие диапазоны, например, от 1 до 1000 или даже от 1 до 1 000 000, или используйте диапазоны отрицательных чисел, если хотите! (Не забывайте, что вы не можете использовать запятые при записи чисел на языке Java.) Вы не только разовьете свой навык программирования, но и улучшите математические навыки. Измените программу по вашему собственному желанию. Это может быть весело!

СТРАТЕГИЯ УГАДЫВАНИЯ

Вы можете обнаружить, что, чем больше вы играете в игру «Больше-Меньше», тем быстрее вам удается угадать число. Вы можете даже столкнуться с тем, что угадываете число быстрее, если в качестве догадок вводите числа из середины диапазона. Этот метод

называется *двоичным поиском*. Вводя числа из середины возможного диапазона, вы с каждой попыткой сужаете поле поиска в два раза.

Вот как это работает. Для числа в диапазоне от 1 до 100 отвечаем 50. Если оно меньше, вы знаете, что загаданное число должно быть в диапазоне от 51 до 100, так что отвечайте 75, прямо посередине этого диапазона. Если это меньше, попробуйте число на полпути между 76 и 100, это будет 87. Одна из причин, по которой двоичный поиск настолько ценен, состоит в том, что нам в каждой игре понадобится всего семь попыток (или меньше), чтобы найти загаданное число в диапазоне от 1 до 100. Попробуйте!

Когда вы научитесь угадывать число от 1 до 100 за семь или меньше попыток, попробуйте угадать число от 1 до 1000 всего за 10 попыток. Если вы действительно храбры (и у вас есть карандаш поблизости), попробуйте угадать число от 1 до 1 000 000. Верьте или нет, вам нужно всего 20 попыток.

Задача № 2: Подсчет попыток

Мы уже создали неплохое приложение для игры «БольшеМеньше», но давайте попробуем добавить в нашу игру еще одну функцию. Ваша задача — подсчитать и сообщить о том, сколько попыток потребовалось пользователю, чтобы угадать число. Это может выглядеть примерно так:

```
62 is correct! You win!
It only took you 7 tries! Good work!
```

Чтобы выполнить эту задачу, вам нужно создать новую переменную (вы можете добавить строку типа `int numberOfTries = 0;`), и вам придется увеличивать количество попыток каждый раз, когда будет выполняться цикл угадывания. Вы можете сделать это, увеличивая переменную `numberOfTries` на 1 для каждого нового цикла, используя команду `numberOfTries = numberOfTries + 1`. Обязательно добавьте вывод на экран, сообщающий пользователю количество сделанных попыток.

Может потребоваться несколько попыток, чтобы весь код работал правильно, но это стоит усилий и поможет вам попрактиковаться в программировании. В главе 3 мы создадим эту функцию в другой версии игры. Тем временем, я надеюсь, вы придумаете

еще больше идей для улучшения и изменения игры. Эксперименты с вашими программами — лучший способ обучения.

Задача № 3: Игра в чепуху

В качестве последнего испытания в этой главе давайте напишем совершенно новую программу. Мы изучили, как попросить пользователя ввести данные с клавиатуры и как сохранить их в переменной. Мы также изучили, как выводить на экран текст и значения переменных. Благодаря этим навыкам вы можете создавать еще более интересные и забавные программы.

Вы когда-нибудь играли в чепуху? Давайте попробуем использовать наши новые навыки для создания программы в том же стиле. Игра «Чепуха» просит игрока ввести несколько слов или частей речи, таких как цвет, глагол в прошедшем времени, или прилагательное, а затем вставляет введенные игроком слова в шаблон, в результате обычно получается забавная история. Например, если игрок ввел «розовый» цвет, глагол прошедшего времени «прыгнул» и прилагательное мужского рода в родительном падеже «глупого», а затем вставил их в шаблон «____ дракон ____ на ____ рыцаря», они получило бы результат «Розовый дракон прыгнул на глупого рыцаря».

Ваша задача состоит в том, чтобы написать новую программу *MadLibs.java* с классом `MadLibs` и методом `main()`, которая запрашивает у пользователя несколько слов. Каждое из этих слов должно храниться в собственной переменной типа `String`, например `color`, `pastTenseVerb`, `adjective` и `noun`, которые вы инициализируете как пустые строки. Затем, после того как пользователь ввел все слова, программа должна вывести на экран завершенное предложение или историю, заменив пустые строки введенными словами. Например:

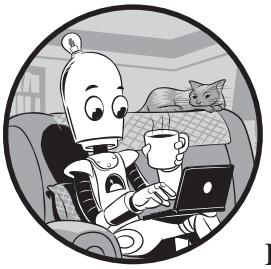
```
System.out.print("The " + color + " dragon " + pastTenseVerb + " at the " +
adjective);
System.out.println(" knight, who rode in on a sturdy, giant " + noun + ".");
```

Обратите внимание, что первая команда представляет собой `print()` вместо `println()`. Команда `print()` продолжает печатать в той же строке, что позволяет нам использовать вывод без разрывов. А вот команда `println()` всегда переходит на новую

строку после окончания печати, как если бы вы нажали клавишу **Enter** в конце строки. Вы можете написать более длинную историю в «Чепухе», используя разные имена переменных, такие как `noun1`, `noun2` и `noun3`. Попробуйте, и приготовьтесь посмеяться над забавными историями, которые вы создадите! Попытайтесь персонализировать каждую созданную вами программу, добавив новые функции и сделав ее индивидуальной.

Глава 3

СОЗДАНИЕ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ ДЛЯ НАШЕЙ ИГРЫ



В этой главе мы создадим версию игры «Больше-Меньше» из главы 2 с графическим интерфейсом пользователя, как показано на рис. 3.1. После запуска этой версии программы, на экране появится графический или визуальный интерфейс, с которым будет взаимодействовать пользователь. Графический интерфейс пользователя позволяет играть в игру в окне рабочего стола, аналогично работе с повседневными программами на ПК. Это вполне профессиональное оконное приложение с текстовым полем для внесения ответа пользователя, кнопкой для его отправки и меткой, которая сообщает пользователю, больше или меньше число, введенное пользователем, загаданного или оно наконец отгадано.

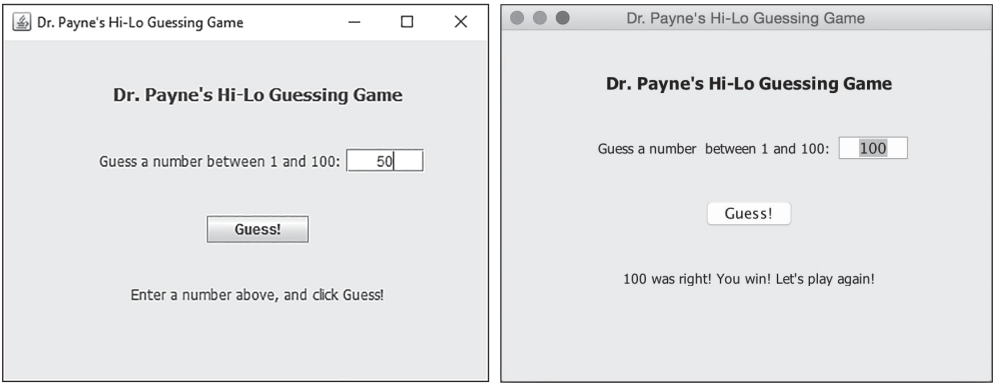


Рис. 3.1. Версия игры «Больше-Меньше» с графическим интерфейсом пользователя для Windows (слева) и macOS (справа)

Самое замечательное, что обе версии приложения на рис. 3.1 получены с использованием одного и того же исходного кода на языке Java. С кодом, который мы напишем в этой главе, наша игра будет работать в операционной системе Windows, macOS и Linux!

Практика в оболочке JShell

Оболочка JShell работает в командной строке, принимает текстовые команды и обычно отвечает выводом текста. Однако JShell также имеет доступ к полному набору библиотек языка Java — предварительно написанным пакетам кода — и не ограничивается только текстом. Прежде чем приступить к написанию программы для нашей игры, давайте создадим простой графический интерфейс в оболочке JShell.

Создание графического интерфейса пользователя всего четырьмя строками кода

Можно создать простое окно графического интерфейса пользователя всего лишь четырьмя строками кода в оболочке JShell. Давайте рассмотрим каждую из них. Сначала напишем инструкцию `import` для импорта класса `javax.swing.JFrame`:

```
jshell> import javax.swing.JFrame
```

Мы испортировали класс `JFrame`, который создает фрейм или окно. Давайте используем его для создания фрейма. Введите следующее объявление для создания объекта `myFrame` класса `JFrame`:

```
jshell> JFrame myFrame = new JFrame("Hello!")
```

Ключевое слово `new` создает новый объект класса `JFrame` — в данном случае окно графического интерфейса пользователя, которое мы запрограммируем таким образом, чтобы получить надпись `Hello!` в заголовке. Оболочка JShell отвечает длинной строкой, сообщая нам о свойства окна по умолчанию:

```
myFrame ==> javax.swing.JFrame[frame1,0,0,0x0, invalid, hidden,
layout=java.awt.BorderLayou …
```

Информация, отображаемая после квадратной скобки, является строкой, представляющей фрейм `myFrame` и значения его свойств, таких как его размер (`0x0`, то есть 0 на 0 пикселей) и скрыт он или отображается. Давайте изменим одно из этих свойств, установив размер больше, чем 0 на 0 пикселей.

Третья строка кода задает размер окна в пикселях, определяя ширину и высоту фрейма:

```
jshell> myFrame.setSize(300,200)
```

Эта строка говорит компилятору Java, что надо сделать окно 300 пикселей в ширину и 200 пикселей в высоту. Давайте наконец покажем окно на экране, вызвав метод `setVisible()`:

```
jshell> myFrame.setVisible(true)
```

Чтобы показать окно, надо написать код `setVisible(true)`, а чтобы скрыть — `setVisible(false)`. Поскольку мы вызвали метод `myFrame.setVisible(true)`, окно должно появиться на экране, как показано на рис. 3.2.

Мы создали окно графического интерфейса пользователя, введя всего четыре строки кода в JShell!

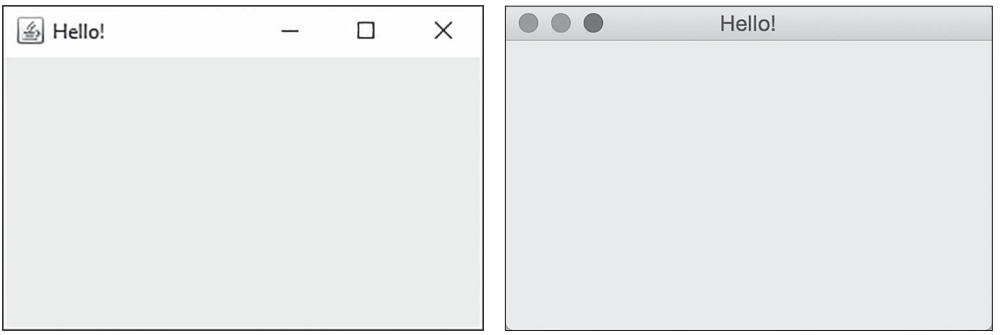


Рис. 3.2. Наше окно с графическим интерфейсом пользователя в операционной системе Windows (слева) и macOS (справа)

Создание интерактивного графического интерфейса пользователя всего десятью строками кода!

Я показал предыдущий пример своим студентам в университете и сыновьям дома, и те и другие ответили что-то вроде: «Это действительно здорово, но можно ли заставить окно что-нибудь *сделать*?»

К счастью, ответ: «Да». Написав еще несколько строк кода, можно добавить в окно кнопку, при каждом нажатии на которую мы будем что-либо выводить на экран.

Начнем с обновления рабочей области JShell. Очистите историю JShell, введя команду `/reset` в оболочке командной строки JShell:

```
jshell> /reset
|  Resetting state.
```

Затем используем встроенный редактор JShell и напишем фрагмент кода для создания интерактивного графического приложения, реагирующего на щелчки мыши. Чтобы открыть редактор JShell, введите в оболочке командной строки JShell команду `/edit`:

```
jshell> /edit
```

Редактор JShell Edit Pad очень удобен, если мы хотим написать сразу несколько строчек кода или вернуться и отредактировать уже набранную строку. Изначально окно редактора пустое, как показано на рис. 3.3.

Рис. 3.3. JShell Edit Pad — удобный редактор для написания длинных фрагментов кода

В данном случае нужно ввести 10 строк кода, которые создадут окно интерактивного интерфейса пользователя с кнопкой, выводящей что-то на экран при нажатии. Будьте внимательны с прописными буквами и не забывайте указывать точку с запятой в конце каждой законченной команды для их разделения. Введите следующий фрагмент:

```
   import javax.swing.*;
   JFrame window = new JFrame("Bryson's Window");
❶  JPanel panel = new JPanel();
   JButton button = new JButton(❷"Click me!");
❸  panel.add(button);
❹  window.add(panel);
   window.setSize(❺300,100);
❻  button.addActionListener(e -> System.out.println("Ouch! You clicked me!"));
❼  window.setVisible(true);
```

Прежде всего, импортируем все классы графического интерфейса пользователя Swing, включая `JFrame`, `JPanel` и `JButton`. Звездочка (`*`) в конце библиотеки на языке Java называется *подстановочным символом* и означает «включать все классы в этот пакет». Затем мы создаем объект `JFrame` так же, как в предыдущем примере. Командой ❶ мы создаем панель внутри окна. Панель будет служить контейнером для других компонентов графического интерфейса, таких как метки и кнопки. Затем добавим кнопку с надписью **Click me!**. Строкой ❸ мы добавим кнопку в панель графического интерфейса, а затем добавим панель в окно командой ❹. После этого, как и в предыдущем примере, установим размер окна в 300 пикселей в ширину и 100 пикселей в высоту.

Фрагмент ❻ — это пункт, где «происходит волшебство»: здесь мы добавляем к кнопке **Click me!** слушатель действий, чтобы реагировать на каждое нажатие пользователем кнопки. Слушатель действия выводит в консоль сообщение `"Ouch! You clicked me!"` каждый раз, когда пользователь нажимает кнопку. Мы будем использовать такие слушатели в графических приложениях, которые будем создавать в этой книге для того, чтобы программы взаимодействовали с пользователем каждый раз, когда пользователь выполняет какое-либо действие.

Наконец, командой ❼ мы делаем окно видимым.

Набрав все 10 строк кода и дважды проверив, все ли верно, нажмите кнопку **Accept** (Принять) в JShell Edit Pad, чтобы принять код и запустить его в JShell, как показано на рис. 3.4. После принятия кода нажмите кнопку **Exit** (Выход), для закрытия JShell Edit Pad. JShell выполнит фрагмент кода, и если вы набрали все правильно, то увидите небольшое окно, подобное тому, которое показано на рис. 3.5.

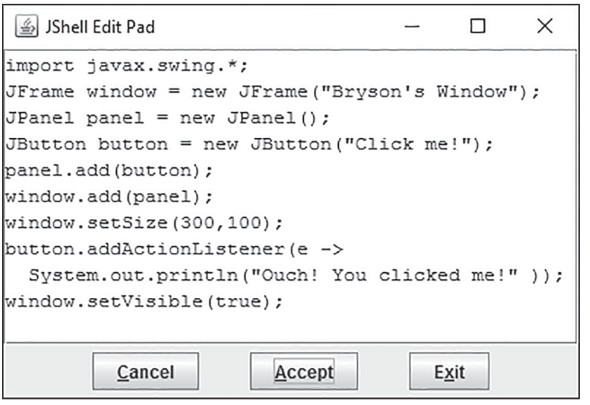


Рис. 3.4. После ввода всех 10 строк кода в JShell Edit Pad нажмите кнопку **Accept**

Рис. 3.5. Интерактивное графическое окно с кнопкой, которую можно нажимать

Щелкните мышью по кнопке с надписью **Click me!**, и вы увидите, что компилятор Java ответит следующим образом в окне оболочки JShell:

```
jshell> Ouch! You clicked me!
Ouch! You clicked me!
Ouch! You clicked me!
Ouch! You clicked me!
```

Если вы закроете небольшое окно графического интерфейса, то, чтобы его вернуть, нужно всего лишь повторить последнюю строку, которая устанавливает его видимость. После возврата в оболочку командной строки JShell несколько раз нажмите клавишу **Enter**, чтобы на экране появилось приглашение и курсор, а затем введите следующее:

```
jshell> window.setVisible(true)
```

Окно снова появляется после присвоения свойству видимости значения `true`. Если вы закроете окно, свойство видимости примет значение `false`.

Это было не так сложно, не правда ли? Теперь вы можете создать игру с графическим интерфейсом пользователя!

Настройка приложения с графическим интерфейсом пользователя в среде разработки Eclipse

Если проект *HiLo* из главы 2 у вас все еще открыт в Eclipse, как на рис. 3.6, то закройте или сверните окно редактора любых других файлов программ на языке Java, с которыми вы, возможно, работали ❶. Кроме того, сверните проект *HiLo*, нажав небольшую стрелку вниз рядом с папкой *HiLo* на панели **Package Explorer** (Обозреватель пакетов) с левой стороны ❷. Это позволит нам объединить проекты в одном рабочем пространстве, не смешивая файлы.

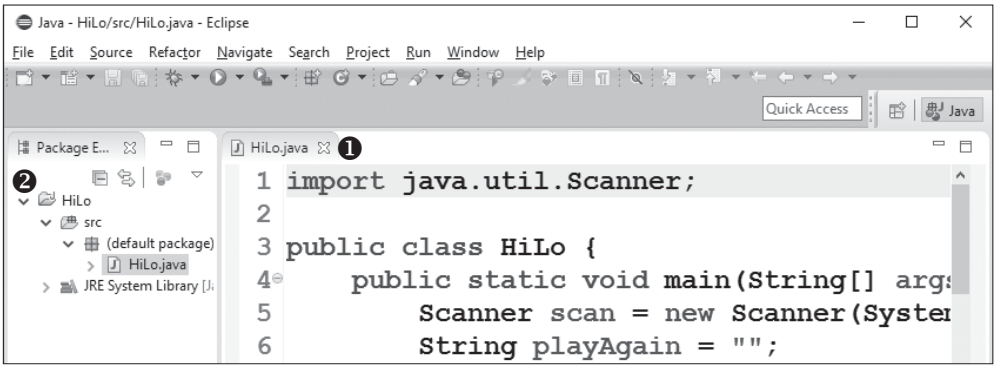


Рис. 3.6. Закройте все открытые файлы и сверните папку проекта HiLo

В строке меню редактора Eclipse выберите команду меню **File** ⇒ **New** ⇒ **Java Project** (Файл ⇒ Новый ⇒ Проект Java) и назовите

проект *GuessingGame*. После ввода имени проекта щелкните мышью по кнопке **Finish** (Завершить) в правом нижнем углу. Тем самым будет создана новая папка проекта, в которой мы разработаем версию игры с графическим интерфейсом пользователя.

Разверните папку *GuessingGame* на панели **Package Explorer** (Обозреватель пакетов) и найдите папку *src*. Щелкните правой кнопкой мыши (щелкните мышью с нажатой клавишей ^ в операционной системе macOS) по папке *src* и выберите пункт **New** ⇒ **Class** (Создать ⇒ Класс). Назовите новый класс **`GuessingGame`**. Обязательно используйте для имени класса горбатый регистр, начиная с прописной буквы G. Установите флажок рядом с заглушкой метода **public static void main (String [] args)**. Тем самым вы добавите каркас метода `main()`, что позволит запускать приложение как отдельную программу.

Прежде чем мы закончим, нам нужно сделать еще один шаг, отличный от того, что мы делали при создании консольного приложения в главе 2. Мы собираемся изменить суперкласс с установленного по умолчанию `java.lang.Object` на класс `javax.swing.JFrame`, который мы использовали в JShell в начале этой главы. *Суперкласс*, или *родительский класс* в языке Java, — это класс, который мы расширяем для повторного использования уже написанного кода, в нашем случае кода, необходимого для построения графического оконного интерфейса. Класс `JFrame` в пакете `javax.swing` — это один из способов включения компонентов графического интерфейса пользователя в наши приложения, поэтому мы используем фрейм `JFrame` для создания окна, а затем создадим собственную версию этого класса и добавим несколько новых возможностей. Поскольку класс `JFrame` является расширением суперкласса `Object`, нам не нужен класс `java.lang.Object`, так как класс `JFrame` *наследует* код суперкласса `Object`. Это означает, что класс `JFrame` будет иметь все возможности класса `Object`, а также некоторые дополнительные. Позже мы рассмотрим работу расширений.

Рис. 3.7 показывает эти настройки для нашей игры в диалоговом окне **New Java Class** (Новый класс Java).

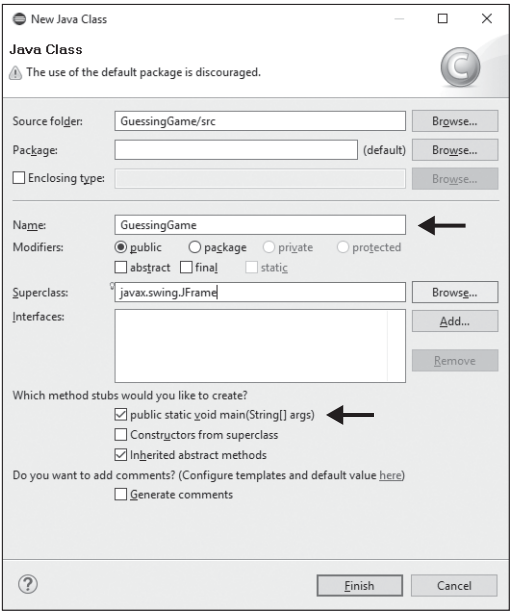


Рис. 3.7. Не забудьте изменить суперкласс на javax.swing.JFrame, чтобы реализовать класс с графическим интерфейсом пользователя

Нажмите кнопку **Finish** (Завершить), и вы увидите следующий исходный код для *GuessingGame.java*:

```
❶ import javax.swing.JFrame;
  public class GuessingGame ❷ extends JFrame {
      public static void main(String[] args) {
          // Автоматически сгенерированная заглушка метода
      }
  }
```

В первой строке ❶ среда разработки Eclipse импортировала класс `javax.swing.JFrame`, чтобы позволить нашей программе использовать графические возможности набора пакетов Swing. Во второй строке мы используем новые возможности программирования — ключевое слово `extends` ❷.

В объектно-ориентированном программировании у родительского класса или суперкласса могут быть *дочерние классы*, которые наследуют все методы и атрибуты, содержащиеся в родительском классе. Когда мы пишем класс для выполнения какой-либо важной функции, которую планируем использовать повторно, мы можем ссылаться на этот оригинальный класс и *расширять* его, добавляя новую функциональность, не меняя при этом код в родительском

классе. В нашем случае родительский класс `JFrame` может отображать текстовые поля, метки, кнопки и другие компоненты графического интерфейса, которые можно настроить под свои задачи и использовать для новой версии нашей игры.

Родительский класс `JFrame`, который мы расширяем в этом приложении, позволит нам отображать окно с элементами графического интерфейса, такими как текстовые поля, метки и кнопки, создавая макет графического интерфейса пользователя с помощью редактора WindowBuilder в среде разработки Eclipse. Давайте посмотрим, как работает редактор WindowBuilder.

Создание графического интерфейса пользователя в Eclipse с помощью редактора WindowBuilder

В самых популярных средах разработки на языке Java реализованы инструменты, помогающие программистам создавать привлекательный графический интерфейс, основываясь на концепции «что видишь, на экране то и получишь» (WYSIWYG). WYSIWYG* позволяет пользователям размещать элементы проекта так, как они будут выглядеть в конечном продукте. Например, редактор Microsoft Word является интерфейсом WYSIWYG, поскольку, редактируя с его помощью, вы получаете текст, выглядящий именно так, как он будет напечатан. Аналогичным образом, инструменты среды разработки Java IDE, такие как редактор WindowBuilder, позволяют программистам размещать компоненты графического интерфейса, такие как текстовые поля, метки и кнопки, именно так, как они будут выглядеть в готовом оконном приложении. Тем самым программистам дается инструмент, позволяющий создавать профессионально выглядящие графические приложения.

Чтобы открыть редактор WindowBuilder, щелкните правой кнопкой мыши по файлу *GuessingGame.java* на панели **Package Explorer** (Обозреватель пакетов) слева, а затем в контекстном меню выберите пункт **Open With ⇒ WindowBuilder Editor** (Открыть в ⇒ Редактор WindowBuilder). Откроется новое окно, похожее на обычный текстовый редактор. Но если вы посмотрите на нижний левый угол окна, вы увидите две новые вкладки — **Source** (Исходный код) и **Design** (Конструктор). На вкладке **Source** (Исходный код) вы обнаружите исходный код вашего приложения.

* От *англ.* What You See Is What You Get (*прим.ред.*).

Если вы выберите вкладку **Design** (Конструктор), вы увидите окно на рис. 3.8.

Редактор WindowBuilder доступен только для классов графического интерфейса, таких как `GuessingGame`, которые расширяют класс `JFrame` или какой-то другой из суперклассов графического интерфейса. Обратите внимание, что в окне предварительного просмотра есть строка заголовка со значком Java слева и тремя кнопками для свертывания, разворачивания и закрытия окна. Окно предварительного просмотра на рис. 3.8 выглядит немного по-разному в операционных системах Windows, macOS и Linux. Это окно будет нашей игровой площадкой для создания игры с графическим интерфейсом пользователя, внешний вид и поведение которой будет соответствовать настоящему приложению.

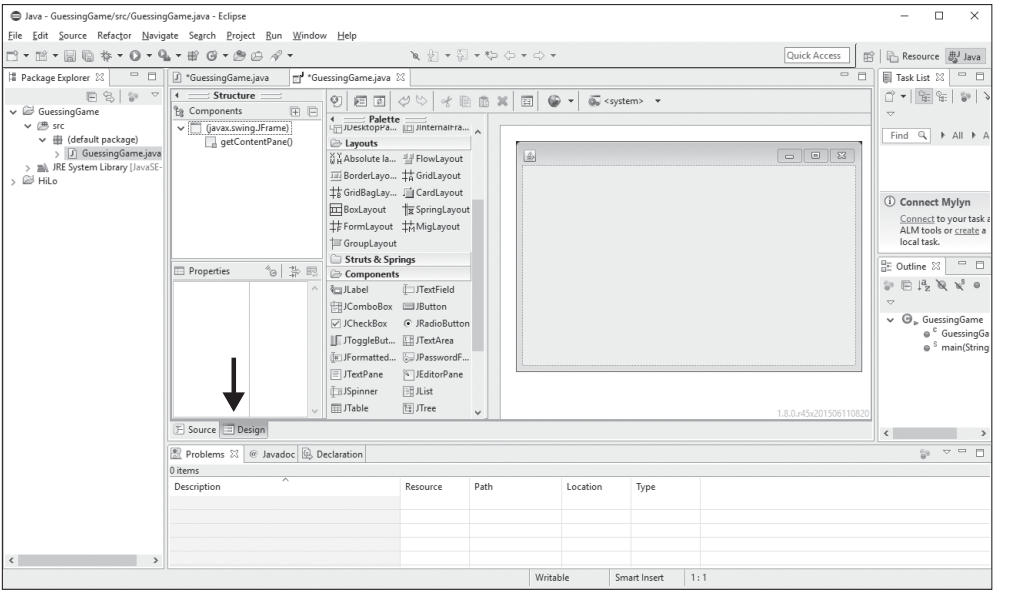


Рис. 3.8. Вкладка **Design** в редакторе WindowBuilder облегчает процесс создания оконных приложений с графическим интерфейсом пользователя

Разработка пользовательского интерфейса

Двойной щелчок по вкладке *GuessingGame.java* в верхней части редактора WindowBuilder разворачивает режим проекта на полный экран. Это дает больше пространства для разработки графического интерфейса. Чтобы восстановить обычный вид редактора, дважды щелкните мышью по вкладке, и вы сможете вернуться к программированию.

Настройка свойств графического интерфейса пользователя на панели Properties

Мы будем использовать панель **Properties** (Свойства) для настройки окна графического интерфейса под задачи нашей игры. Панель **Properties** (Свойства) находится в левом нижнем углу вкладки **Design** (Конструктор) в редакторе WindowBuilder. Как вы можете видеть на рис. 3.9, если вы щелкнете мышью по компоненту на панели **Components** (Компоненты) (расположена слева, ниже строки **Structure** (Структура)), панель **Properties** (Свойства) непосредственно под ней отобразит несколько свойств и значений, которые мы можем просмотреть и отредактировать для данного компонента.

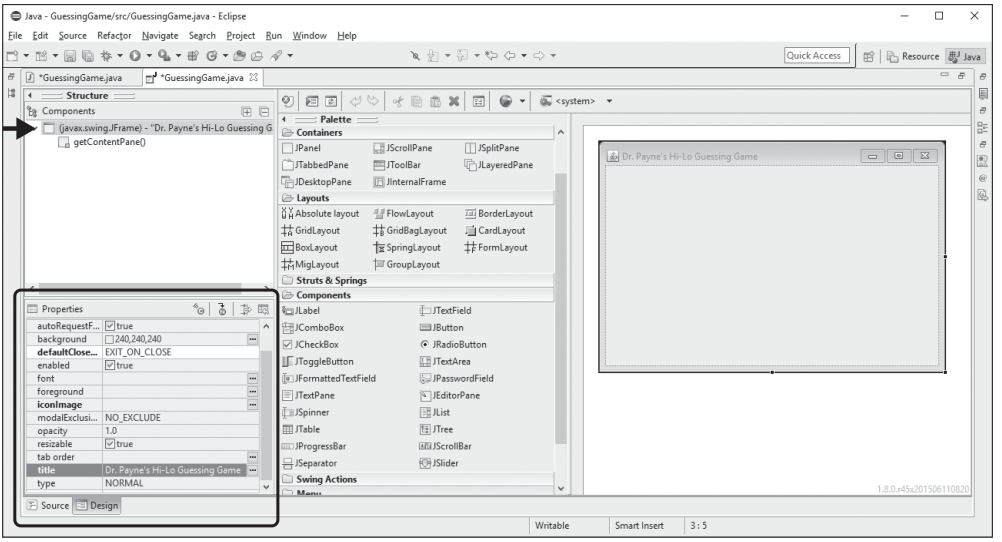


Рис. 3.9. Панель **Properties** помогает настроить свойства всех графических компонентов в вашем приложении

Мы собираемся создать настраиваемое окно графического интерфейса для приложения, угадывающего число. Во-первых, давайте изменим свойство, отвечающее за название для этого фрейма `JFrame`. На панели **Components** (Компоненты) выберите пункт `javax.swing.JFrame`, затем прокрутите вниз содержимое панели **Properties** (Свойства), пока не увидите свойство `title`. Щелкните мышью в пустом поле справа от имени `title` и введите ***Ваше имя*** **Hi-Lo Guessing Game**. (Замените курсивный текст на свое имя, это ваше приложение, в конце концов!) После ввода текста заголовка

нажмите клавишу **Enter**. Окно предварительного просмотра графического интерфейса (справа) автоматически обновится и отобразит в строке заголовка *Ваше имя* Hi-Lo Guessing Game.

Теперь давайте установим еще несколько свойств, чтобы графический интерфейс пользователя выглядел и вел себя так, как мы хотим. Найдите на панели **Properties** (Свойства) значение `defaultCloseOperation` (если панель слишком узкая, на ней может отображаться только часть имени, например `defaultClose...` или что-то похожее). Щелкните мышью по полю справа от `defaultCloseOperation` и выберите пункт `EXIT_ON_CLOSE` в контекстном меню. Это означает, что при нажатии кнопки закрытия окна в строке заголовка этого фрейма `JFrame` вы выйдете из программы. Обычно это значение установлено по умолчанию, но может так случиться, что вы захотите закрыть окно, не закрывая все приложение, например в диалоговом окне сохранения или всплывающем окне. Однако для данной игры с одним окном мы хотим выходить из приложения, когда закрываем главное окно.

Затем изменим расположение (макет) графических компонентов внутри окна приложения. Щелкните мышью по элементу `getContentPane()` на панели **Components** (Компоненты), ниже пункта `javax.swing.JFrame`. *Панель содержимого* — это *внутренность* фрейма `JFrame`, где мы создадим макет графического интерфейса нашей игры. Теперь, найдите пункт **Layout** (Макет) на панели **Properties** (Свойства) и щелкните мышью по стрелке вниз на правом краю поля. Выберите вариант **Absolute Layout** (Абсолютная компоновка) в раскрывающемся списке значений; это позволяет нам размещать элементы графического интерфейса пользователя с точностью до пикселя.

Настройка компонентов графического интерфейса пользователя с помощью панели Palette

Давайте начнем настраивать нашу игру. На панели **Palette** (Панель элементов) можно найти элементы, которые пригодятся почти для любого приложения с графическим интерфейсом пользователя. Панель **Palette** (Панель элементов) находится в центре редактора WindowBuilder на вкладке **Design** (Конструктор). Обратите внимание, что вы можете развернуть или свернуть панель, щелкнув мышью по маленькой стрелке слева от ее заголовка. Щелкните мышью по стрелке один раз, чтобы свернуть панель. Щелкните

еще раз, чтобы развернуть. Это удобно, когда вам необходимо дополнительное место для компоновки большого, сложного графического интерфейса.

Прокрутите содержимое панели **Palette** (Панель элементов) вниз, пока не увидите раздел **Components** (Компоненты), как показано на рис. 3.10.

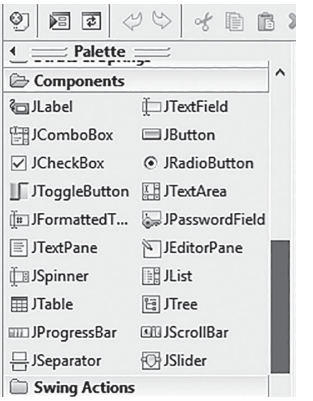


Рис. 3.10. Раздел **Components** на панели **Palette**, где вы можете найти большинство компонентов графического интерфейса пользователя

Панель **Palette** (Панель элементов) содержит все стандартные компоненты, с которыми вы, возможно, уже знакомы, например: метки (`JLabel`), текстовые поля (`JTextField`), кнопки (`JButton`) и флажки (`JCheckBox`). Возможно, вы заметили, что все компоненты графического интерфейса пользователя в пакете `javax.swing` начинаются с прописной буквы J, за которой следует имя компонента в горбатом регистре. Это позволяет легко запомнить имя класса для каждого из элементов графического интерфейса пользователя в приложении, использующем инструментарий Swing.

Поместим в верхнюю часть окна графического интерфейса метку, на которой напишем текст «*Ваше имя* Hi-Lo Guessing Game». Щелкните мышью по пункту `JLabel` в разделе **Components** (Компоненты) панели **Palette** (Панель элементов). Затем установите указатель мыши на окно предварительного просмотра графического интерфейса справа. Вы должны увидеть линии сетки. По умолчанию редактор WindowBuilder показывает эти линии сетки, помогая размещать элементы. Теперь установите указатель мыши на верхнюю часть в центре серой панели содержимого (внутренности фрейма `JFrame`) и щелкните, чтобы поместить метку `JLabel` в этой позиции.

Когда вы размещаете метку `JLabel` в первый раз, вы можете редактировать текст метки непосредственно в окне предварительного просмотра графического интерфейса. Напечатайте текст ***Ваше имя* Hi-Lo Guessing Game** и нажмите клавишу **Enter**. Если позже вы захотите поменять текст, перейдите к свойству `text` объекта `JLabel` на панели **Properties** (Свойства) в левом нижнем углу и введите текст. Сначала вы, вероятно, увидите только часть текста на метке, поскольку метка слишком мала и не может показать

полный текст. Щелкните мышью по метке, и в каждом углу вы увидите маленькие черные квадраты для изменения размера. Нажав и удерживая кнопку мыши на левом нижнем углу метки, перетащите мышь к левой границе области содержимого, чтобы растянуть метку до левого края панели содержимого. Затем, нажав и удерживая кнопку мыши на правом нижнем углу метки, перетащите его к правому краю панели содержимого для изменения размера.

Теперь весь ваш текст должен помещаться внутри метки в верхней части окна предварительного просмотра графического интерфейса. Затем отцентрируем текст метки и выделим его полужирным. Сначала найдите свойство `horizontalAlignment` для метки на панели **Properties** (Свойства). Щелкните мышью по значению и выберите пункт `CENTER` в раскрывающемся списке. Затем найдите для метки свойство `font` и щелкните мышью по троеточию справа от значения свойства `font`. Откроется окно выбора шрифтов, позволяющее выбрать шрифт, его стиль и размер. Я выбрал полужирный шрифт Tahoma размером 15 пунктов, как показано на рис. 3.11. Имейте в виду, что вам, возможно, придется изменить размер метки, чтобы она могла вместить текст, отформатированный более крупным шрифтом.

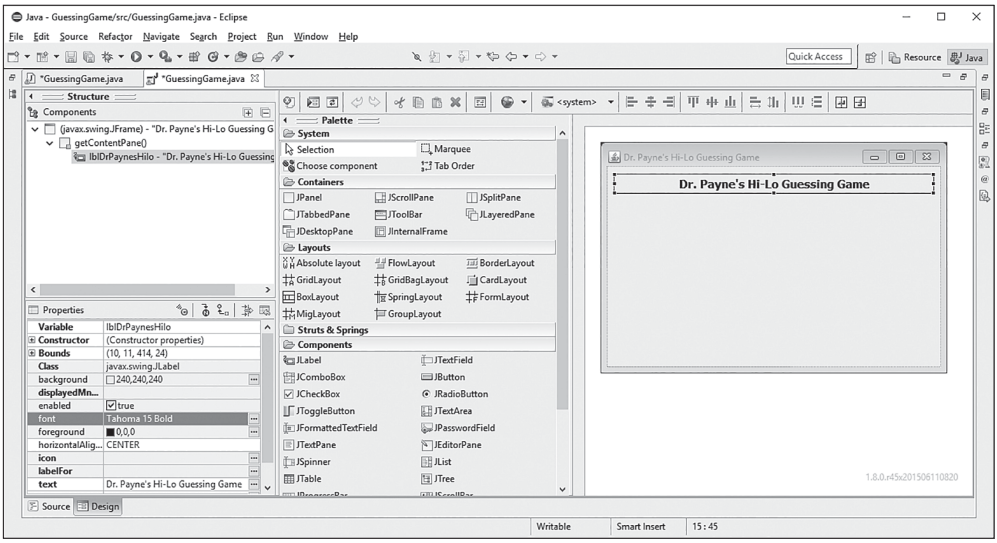


Рис. 3.11. Первая метка настроена и помещена в верхней части панели содержимого

Давайте добавим метку для первого приглашения, которое пользователь увидит в игре. Щелкните мышью по метке `JLabel` на панели **Palette** (Панель элементов), а затем поместите новую

метку чуть выше середины области содержимого. Введите значение **Guess a number between 1 and 100:** в качестве текста метки. Вам нужно будет сделать размер метки чуть больше размера текста из-за внутренних отступов.

Справа от метки поместим текстовое поле для ввода ответа пользователя. Выберите элемент `JTextField` на панели **Palette** (Панель элементов), а затем поместите текстовое поле справа от только что созданной метки.

Измените размер поля, чтобы оно могло вместить трехзначное число. Щелкните мышью по метке еще раз и присвойте ее свойству `horizontalAlignment` значение `RIGHT`, чтобы текст оказался максимально близко с текстовым полем, как показано на рис. 3.12.

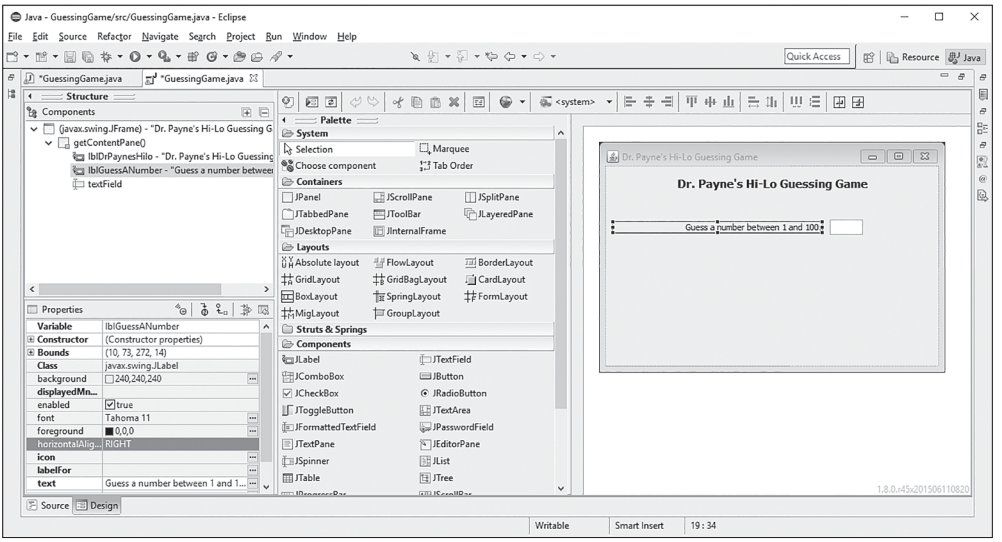


Рис. 3.12. Для игры сделана метка и текстовое поле для хода пользователя

Затем давайте сделаем кнопку, которую пользователь может нажать, чтобы ввести свое число. Найдите на панели **Palette** (Панель элементов) пункт `JButton` и щелкните по нему мышью. Установите указатель мыши в центр окна предварительного просмотра графического интерфейса и щелкните мышью, чтобы поместить объект `JButton`. Измените текст на кнопке на `"Guess!"`.

Наконец, поместите объект `JLabel` под кнопку и задайте ему текст **Enter a number above and click Guess!**. Затем измените размер метки так, чтобы она занимала всю ширину фрейма `JFrame`. Потом присвойте свойству `horizontalAlignment` значение `CENTER`. Позже мы воспользуемся этой меткой, чтобы сообщить

пользователю, что его число больше, меньше или равно загаданному. Все компоненты возможно еще не выровнены идеально, однако ваш графический интерфейс должен выглядеть примерно так, как показано на рис. 3.13.

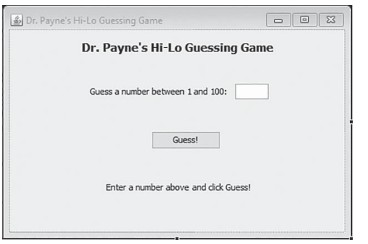


Рис. 3.13. Все компоненты размещены в графическом интерфейсе, но еще не выровнены

Разметив на экране все компоненты графического интерфейса, мы готовы сбалансировать и отцентрировать компоненты макета. Но прежде всего, не забудьте сохранить свой файл.

Выравнивание элементов графического интерфейса

До сих пор мы помещали компоненты графического интерфейса на глазок, пытаясь правильно их разместить и выровнять по центру, насколько это возможно. Однако, используя программу, мы ожидаем, что компоновка будет «идеальной». К счастью, среда разработки Eclipse имеет встроенные инструменты, помогающие нам выравнивать компоненты.

Во-первых, давайте распределим четыре основных элемента равномерно, расположив их на одном и том же расстоянии друг от друга по вертикали. Выберите три метки и кнопку, выделяя компоненты щелчками мышью, удерживая нажатой клавишу **Shift**. Не надо пока щелкать по текстовому полю, в которое пользователь будет вводить свой ответ. Вы должны увидеть небольшой ряд инструментов выравнивания, которые появятся над окном предварительного просмотра графического интерфейса пользователя. Наведя указатель мыши на каждый из этих инструментов вы увидите подсказку о том, что делает та или иная кнопка. Возможно, вам придется изменить размер окна программы Eclipse, если вы изначально не видите все эти инструменты.

Рис. 3.14. Вы можете выровнять и разместить компоненты равномерно с помощью инструментов, расположенных над окном предварительного просмотра графического интерфейса

Выберите инструмент, который выглядит как уложенные в стопку кнопки (крайний правый значок на рис. 3.14). Все четыре выбранных элемента теперь должны расположиться на равном расстоянии друг от друга сверху вниз.

Теперь выберите только текстовое поле. Переместите его так, чтобы оно располагалось рядом с меткой, в которой пользователю

предлагается сделать ход. Мы не стали первоначально выделять текстовое поле, поскольку среда разработки Eclipse равномерно распределила бы все пять компонентов, отделив текстовое поле от его метки и создав пять рядов вместо четырех.

Если вы что-либо испортили, выравнивая компоненты (или делая что-либо еще) в WindowBuilder, вы можете отменить свое предыдущее действие, нажав сочетание клавиш **Ctrl+Z** (или ⌘**+Z** в macOS). Это отличная возможность, предусмотренная в редакторе WindowBuilder для безопасности и позволяющая экспериментировать, не нарушая при этом компоновку макета.

Наконец, щелкните мышью по кнопке с надписью «**Guess!**» на макете, а затем по кнопке выравнивания над окном предварительного просмотра графического интерфейса (по той, что находится рядом со всплывающей подсказкой «Center horizontally in window»). Вы можете центрировать или выравнивать другие компоненты как хотите. Если вы хотите переместить несколько компонентов вместе, выделите их щелчками мыши, удерживая нажатой клавишу **Shift**.

Если вы довольны макетом, то пора подготовиться к программированию, присвоив имена компонентам. Перед этим сохраните изменения.

Именование компонентов графического интерфейса для программирования

Мы подготовили интерфейс нашего приложения. Однако чтобы упростить работу с приложением, необходимо сделать несколько завершающих настроек в файле исходного кода на языке Java, там, где мы будем программировать оставшуюся часть игры. Нам нужно именовать каждый компонент для того, чтобы знать, как обращаться к нему внутри исходного кода.

Этот шаг иногда называют «соединение», поскольку мы будем соединять каждый используемый в программе компонент графического интерфейса с именами переменных, к которым можно обратиться в исходном коде на языке Java. Когда мы добавляли элементы графического интерфейса, среда разработки Eclipse

присваивала имена каждому из них автоматически, однако мы хотели бы изменить эти имена. Выберите текстовое поле, в котором пользователь делает свой ход. На панели **Properties** (Свойства) обратите внимание на самое верхнее свойство — `Variable`. Имя, которое программа Eclipse дала вашему первому текстовому полю по умолчанию `textField`. Щелкните мышью по полю со значением рядом со свойством `Variable` и задайте переменной имя **`txtGuess`**, как показано на рис. 3.15.

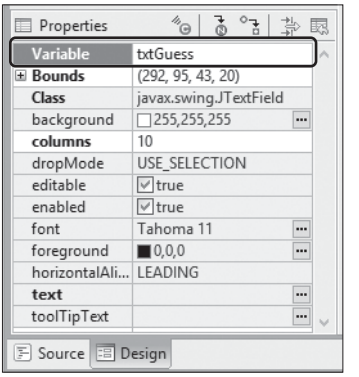


Рис. 3.15. Самостоятельно задайте имена переменным для каждого элемента графического интерфейса, чтобы их можно было легко использовать в коде Java

Имя `txtGuess` сообщает о том, что это текстовое поле хранит ходы пользователя. Позже вы поймете, что логичное и последовательное именование компонентов графического интерфейса помогает программировать быстрее, эффективнее и с меньшим количеством ошибок.

После того как вы переименовали текстовое поле, выберите метку в нижней части окна графического интерфейса, на которой написано «**Enter a number above and click Guess!**». Это метка, в которой будет выводиться информация для пользователя, например, `"Too high"`, `"Too low"` или `"You win!"`, поэтому давайте назовем эту переменную `lblOutput`. Это имя поможет нам вспомнить, что это метка графического интерфейса (`lbl`), и мы хотим использовать ее для вывода информации для пользователя.

На вкладке **Source** (Исходный код) в левом нижнем углу окна редактора WindowBuilder вы можете увидеть, что среда разработки Eclipse добавила исходный код на языке Java для создания разработанной нами компоновки графического интерфейса (рис. 3.16).

Обратите внимание, что новые имена переменных, которые вы дали текстовому полю (`txtGuess`) и метке, выводящей информацию (`lblOutput`), видны в исходном коде. Сейчас самое подходящее время сохранить файл со всеми сделанными на данный момент изменениями.

Однако прежде чем начать писать код для нашей игры, необходимо внести изменения в исходный код.

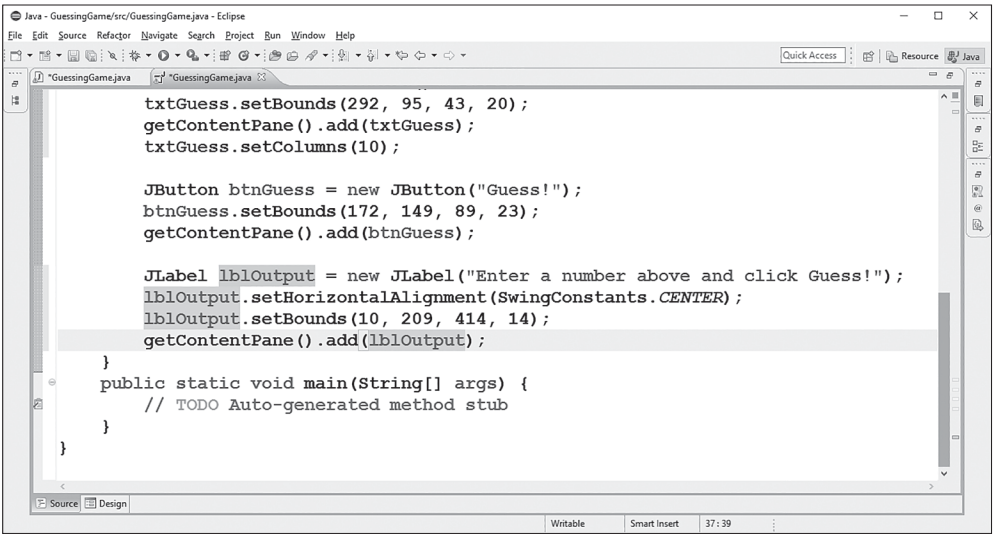


Рис. 3.16. Перейдите на вкладку **Source**, и вы увидите, что среда разработки Eclipse сгенерировала Java-код для компонентов графического интерфейса

Соединение графического интерфейса с кодом на языке Java

Прокрутите окно вверх до начала класса `GuessingGame`, и вы увидите, что среда разработки Eclipse объявила текстовое поле `JTextField` по имени `txtGuess` прямо в верхней части класса `GuessingGame`:

```
public class GuessingGame extends JFrame {
    private JTextField txtGuess;
```

Программа Eclipse делает это по умолчанию для текстовых полей не случайно. Объявление текстовых полей в верхней части класса позволяет получить доступ к текстовому полю (чтобы, например, посмотреть, больше или меньше загаданного число, введенное пользователем) из *любого* места класса. Текстовые поля обычно используются именно таким образом, поэтому среда разработки Eclipse объявляет `private JTextField txtGuess` прямо в верхней части класса `GuessingGame`, а не в методе или функции ниже в классе. Модификатор `private` — *отличный прием* (рекомендуемый способ делать что-то) объектно-ориентированного программирования. Объявление чего-либо с модификатором `private` не позволяет другим классам видеть эту часть кода. Поскольку текстовое поле `txtGuess` объявлено закрытым (`private`), другие

классы, кроме GuessingGame, не смогут получить к нему доступ и, соответственно, изменить значение переменной txtGuess, чего мы и добиваемся.

В нашем приложении мы хотим иметь доступ к значению, введенному пользователем в текстовое поле txtGuess, поэтому здорово, что среда разработки Eclipse объявила его на верхнем уровне класса. Мы также хотели бы иметь доступ к метке lblOutput, чтобы была возможность изменять текст сообщения пользователю (больше ли число пользователя загаданного, меньше или пользователь угадал). Поэтому нам нужно добавить еще одно объявление в начало класса, прямо под объявлением текстового поля txtGuess:

```
public class GuessingGame extends JFrame {
    private JTextField txtGuess;
    private JLabel lblOutput;
```

В этой строке кода объявляется переменная lblOutput, которая ссылается на объект JLabel графического интерфейса приложения. Мы решили сделать ссылку на этот объект private, чтобы скрыть эти данные от внешних программ, однако все равно сможем использовать ее во всем классе GuessingGame.

Наше последнее изменение состоит в исправлении строки (в нижней части класса), где первоначально был объявлен объект lblOutput. Ниже в коде найдите строку, которая выглядит примерно так:

```
JLabel lblOutput = new JLabel("Enter a number above and click Guess!");
```

И измените ее следующим образом:

```
lblOutput = new JLabel("Enter a number above and click Guess!");
```

Обратите внимание, что мы удалили дополнительное объявление метки JLabel. Если вы оставите это второе объявление объекта JLabel в программе, ваше приложение не будет работать, поскольку вы уже объявили переменную lblOutput как тип JLabel в верхней части вашего класса. Если вы оставите второе объявление метки JLabel, компилятор Java подумает, что вы имеете в виду два отдельных объекта JLabel с именем lblOutput. Удалив второе объявление метки JLabel, вы инструктируете компилятор

Java использовать *единственную* переменную `lblOutput` класса `JLabel`, которую вы создали в верхней части программы, и инициализировать ее текстовым значением `"Enter a number above and click Guess!"`.

Пока мы объявляем переменные в верхней части класса `GuessingGame`, добавим еще одну важную переменную: `theNumber`. Это число, которое в консольной версии игры мы называли секретным случайным числом, которое пользователь пытается угадать. Добавьте объявление для переменной `theNumber` после текстового поля `txtGuess` и метки `lblOutput` следующим образом:

```
public class GuessingGame extends JFrame {
    private JTextField txtGuess;
    private JLabel lblOutput;
    private int theNumber;
```

Завершив объявление переменных в верхней части класса, мы готовы начать программирование нашей игры с графическим интерфейсом. В следующем разделе вы узнаете, как получить ход пользователя из текстового поля графического интерфейса и проверить, больше оно или меньше загаданного. Мы также покажем в метке `lblOutput` текст с информацией, помогающей пользователю сделать следующий ход.

Добавление метода проверки хода игрока

После подключения графического интерфейса пользователя к исходному коду на языке Java с добавленными закрытыми переменными `txtGuess` и `lblOutput` мы начнем программировать логику нашей игры. Давайте в верхней части класса `GuessingGame` напишем метод с именем `checkGuess()`. Сигнатура или каркас этого метода будет выглядеть так:

```
public class GuessingGame extends JFrame {
    private JTextField txtGuess;
    private JLabel lblOutput;
    private int theNumber;
    public void checkGuess() {
    }
```

Обратите внимание, что метод объявлен открытым (`public`). Переменные в классе обычно объявляются как закрытые (`private`), но методы или функции, которые работают с этими переменными, обычно объявляются как открытые (`public`), чтобы другие классы могли обращаться к ним. Вы можете думать об этом как о своем текущем счете. Ваш баланс закрыт и может быть доступен только вам или банку, но функция внесения вкладов открыта (другие люди могут внести деньги на ваш счет).

Второе слово, `void`, говорит компилятору Java, что мы не ожидаем *возвращаемого* значения, то есть значения, которое функция или метод возвращает из этой функции. Например, калькулятор преобразования температуры из градусов по Фаренгейту в градусы по Цельсию под названием `convertFtoC()` может принимать значение, представляющее температуру в градусах Фаренгейта, и возвращать числовое значение, представляющее преобразованную температуру в градусах Цельсия. В нашем приложении метод `checkGuess()` не будет возвращать информацию подобную этой остальной части программы. Вместо этого мы собираемся запрограммировать приложение так, чтобы информация для пользователя отображалась в графическом интерфейсе, поэтому мы используем тип возвращаемого значения `void`, то есть такой тип, который не возвращает никакого значения.

Итак, мы назвали метод `checkGuess()` и поставили открывающую и закрывающую фигурные скобки, внутри которых будет содержаться код, сообщающий программе, как именно мы собираемся проверять ходы пользователя.

Получение текста из текстового поля JTextField

Давайте начнем создание метода `checkGuess()`, который принимает строку, введенную пользователем, и сравнивает ее с загаданным числом. Добавьте следующую строку в метод `checkGuess()` между двумя фигурными скобками:

```
public void checkGuess() {
    String guessText = txtGuess.getText();
}
```

В этой строке создается новая переменная типа `String` с именем `guessText`, которая будет содержать число, введенное

пользователем в текстовом поле графического интерфейса. Чтобы получить текст, введенный пользователем, нужно использовать метод `getText()`, вызвав его следующим образом — `txtGuess.getText()` и сохранив результат в новой строковой переменной под именем `guessText`.

После ввода `txtGuess` и оператора (`.`) появится окно, показанное на рис. 3.17.

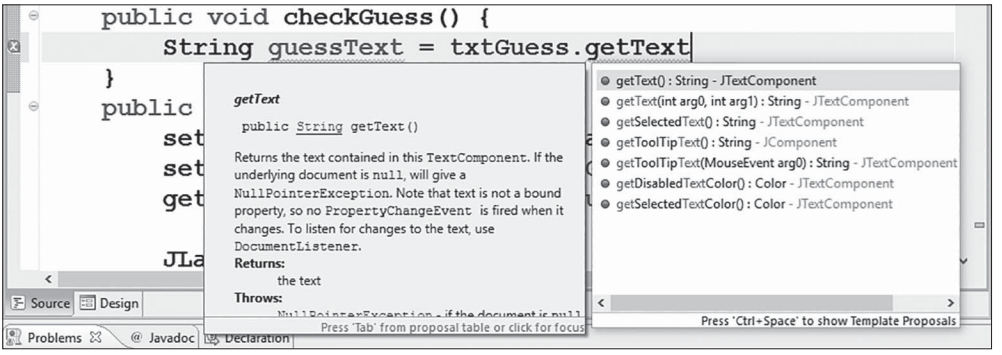


Рис. 3.17. После ввода txtGuess и оператора вы можете увидеть окно с рекомендациями по автозавершению

Это называется *автозавершение* — таким образом среда разработки Eclipse пытается помочь, предлагая строки кода для завершения команды, над которой вы работаете. Этот особый вид автозавершения также называется *советчик кода*, поскольку он рекомендует типичные варианты кода, например методы текущего класса или объекта. Рекомендация кода — это одна из возможностей, которая делает Eclipse мощной интегрированной средой разработки на языке Java для профессионалов. В большинстве случаев рекомендации советчика кода не только позволяют быстрее программировать, но и помогают предотвратить ошибки и делают работу программистов более эффективной, особенно при разработке сложных приложений.

Теперь нам нужно создать строку, в которой мы будем хранить сообщение для пользователя о том, было ли его число больше загаданного, меньше его или пользователь угадал. Давайте объявим еще одну строковую переменную:

```
public void checkGuess() {
    String guessText = txtGuess.getText();
    String message = "";
}
```

Мы еще не знаем, введенное пользователем число больше или меньше загаданного, поэтому мы инициализировали переменную `message` пустой строкой с помощью пары двойных кавычек (`""`).

После того как мы сравним введенное пользователем число с загаданным случайным числом, то есть с `theNumber`, мы обновим ее, прежде чем вывести эту строку в метке графического интерфейса. Чтобы оценить верность хода пользователя, нам необходимо преобразовать текст, введенный им и хранящийся в переменной `guessText`, в числовую форму для сравнения с `theNumber`.

Преобразование строк в числа

В консольной версии этого приложения из главы 2 мы использовали объект класса `Scanner` для сканирования числового значения, введенного пользователем с клавиатуры. Сканер также может работать в графическом интерфейсе. Но мы будем использовать более компактный способ, который будет лучше работать в этой версии.

Так же, как в классе `Math` есть метод `Math.random()`, позволяющий генерировать случайное число, существует также класс для работы с целыми числами по имени `Integer`. В классе `Integer` реализовано несколько функций, полезных для работы с целыми числами, включая метод поиска целых чисел в строках текста `Integer.parseInt()`. Под строкой `String message = "";` введите следующую строку:

```
public void checkGuess() {
    String guessText = txtGuess.getText();
    String message = "";
    int guess = Integer.parseInt(guessText);
}
```

Сначала эта строка объявляет `guess` целочисленной переменной. Затем ищет (*разбирает*) целое число во введенном пользователем тексте. Например, строка `"50"` станет числом `50`. Наконец, сохраняет это число в переменной `guess`. Нам нужна числовая версия хода пользователя, чтобы мы могли сопоставить ее с переменной `theNumber`, используя операторы сравнения < и >.

Сохранив ответ пользователя в переменной `guess`, мы готовы сравнить ее с загаданным компьютером случайным числом

theNumber. На самом деле мы еще не написали код, позволяющий хранить загаданное число в переменной theNumber, но мы сделаем это в самое ближайшее время. Сравнение попытки пользователя со значением переменной theNumber будет выглядеть так же, как и в консольной версии игры, за исключением того, что мы не будем печатать в консоли с помощью System.out.println(). Вместо этого мы сохраним информацию для пользователя в строковой переменной message, которую мы создали только что:

```
        String guessText = txtGuess.getText();
        String message = "";
        int guess = Integer.parseInt(guessText);
        if (guess < theNumber)
            message = guess + " is too low. Try again.";
        else if (guess > theNumber)
            message = guess + " is too high. Try again.";
        else
            message = guess + " is correct. You win!";
    }
```

Обратите внимание, что три инструкции if-else практически идентичны тем, что мы использовали в консольной версии игры, за исключением того, что мы сохраняем сообщение для пользователя (больше, меньше или угадал) в переменной message вместо того, чтобы выводить его непосредственно в консольное окно. Нам все равно нужно показывать переменную message пользователю, что мы и сделаем это, используя метку lblOutput графического интерфейса. Для этого мы используем метод setText() следующим образом:

```
            else
                message = guess + " is correct. You win!";
                lblOutput.setText(message);
        }
```

Эта команда покажет сообщение пользователю в окне графического интерфейса, присвоив свойству text объекта JLabel с именем lblOutput значение переменной типа String с именем message, которое, в свою очередь, было установлено на основе хода пользователя.

Полностью законченный код метода `checkGuess()` должен выглядеть следующим образом:

```
public void checkGuess() {
    String guessText = txtGuess.getText();
    String message = "";
    int guess = Integer.parseInt(guessText);
    if (guess < theNumber)
        message = guess + " is too low. Try again.";
    else if (guess > theNumber)
        message = guess + " is too high. Try again.";
    else
        message = guess + " is correct. You win!";
    lblOutput.setText(message);
}
```

Далее нужно написать код, который присваивает случайное значение переменной `theNumber`. Мы будем использовать метод `newGame()`, потому что хотим получать новое случайное число каждый раз, когда пользователь начинает игру заново.

Запуск новой игры

Мы хотим предложить компьютеру выбрать новое секретное случайное число в момент начала нового раунда игры, поэтому целесообразно сделать это в методе класса `GuessingGame`. Реализуя это таким образом, мы сможем вызывать метод каждый раз, когда пользователь выиграет и захочет начать новую игру.

Определение метода `newGame()` аналогично методу `checkGuess()`. Поместите его сразу после закрывающей скобки метода `checkGuess()`, но перед `public GuessingGame()`:

```
    lblOutput.setText(message);
}
public void newGame() {
}
public GuessingGame() {
```

Мы сделаем этот метод открытым, чтобы его можно было вызывать из внешнего класса. Нам также нужно сообщить компилятору

Java, что подобно методу `checkGuess()`, этот метод не возвращает данные, поэтому тип его возвращаемого значения `void`. Наконец, имя метода — `newGame`, и оно включает открывающую и закрывающую круглую скобку `()`. Тело метода сейчас пусто, между фигурными скобками нет ничего, но этим мы займемся чуть позже.

Новая игра в этом приложении просто означает, что мы просим компьютер выбрать новое случайное число. Код для назначения присвоения этого случайного значения переменной `theNumber` будет выглядеть и функционировать так же, как это было в консольной версии игры, за исключением того, что сейчас мы поместим этот код в свой собственный отдельный метод. Итак, готовый метод `newGame()` должен выглядеть следующим образом:

```
public void newGame() {
    theNumber = (int)(Math.random() * 100 + 1);
}
```

Готово! Теперь у нас есть все необходимые компоненты для создания функционального приложения для игры, поэтому нам просто нужно подключить их к графическому интерфейсу пользователя. Как только приложение сможет прослушивать пользовательские события и, как следствие, сможет отреагировать на нажатие кнопки **Guess!**, мы будем готовы запустить игру.

Прослушивание пользовательских событий: щелкните, чтобы угадать!

Последний шаг, который нам нужно выполнить перед запуском приложения, — это соединить кнопку `btnGuess` (с надписью «**Guess!**») с функцией, которая проверяет ходы пользователя, `checkGuess()`. В языке программирования Java для этих целей можно использовать *слушатели событий*. Слушатели событий — это всего лишь код, который говорит программе ждать или *слушать события*, возникающие из-за взаимодействия пользователем с системой, такие как щелчки по кнопкам, перемещения мыши или нажатия клавиш клавиатуры.

Вернитесь к графическому интерфейсу приложения, перейдя на вкладку **Design** (Конструктор). В окне предварительного просмотра графического интерфейса найдите кнопку **Guess!**, как показано на рис. 3.18, и дважды щелкните по ней мышью.

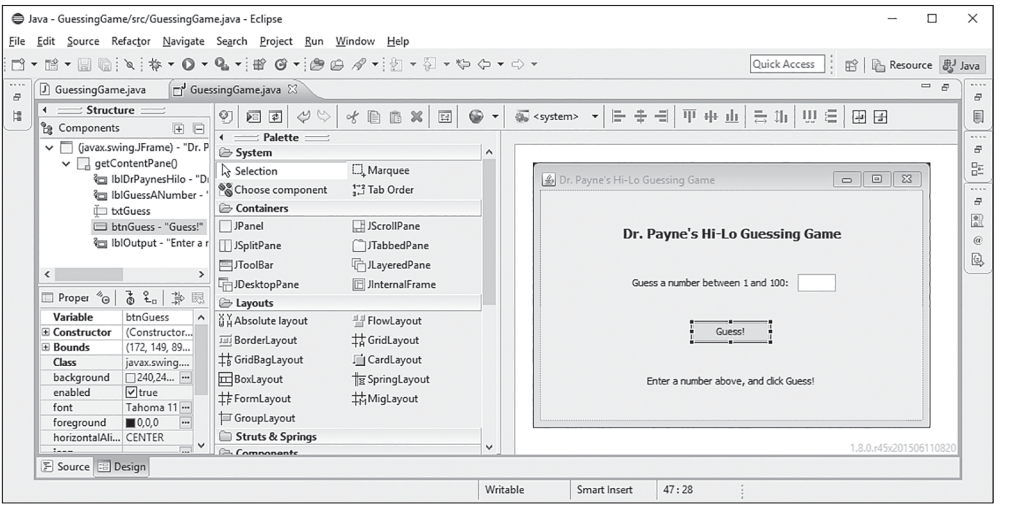


Рис. 3.18. На вкладке **Design** дважды щелкните мышью по кнопке **Guess!**, чтобы добавить слушателя событий в код

Двойной щелчок мышью по кнопке **Guess!** автоматически открывает вкладку **Source** (Исходный код), где вы заметите, что среда разработки Eclipse добавила новый код. Этот новый код является каркасом слушателя событий — в данном случае `ActionListener()` для события, которое происходит, когда пользователь предпринимает действие с объектом `JButton btnGuess`. В данном случае это действие состоит в щелчке мышью по кнопке.

Программа Eclipse добавила все, кроме действия, которое мы хотим выполнить, когда пользователь щелкает по кнопке. Давайте посмотрим на код слушателя событий, созданный для нас средой разработки Eclipse.

```
        JButton btnGuess = new JButton("Guess!");
❶       btnGuess.addActionListener(new ❷ ActionListener() {
❸           public void actionPerformed(ActionEvent e) {
            }
        });
        btnGuess.setBounds(172, 149, 89, 23);
        getContentPane().add(btnGuess);
```

Слушатель событий для `ActionListener()` — специфический слушатель, который обрабатывает щелчки по кнопкам, соединен с кнопкой `btnGuess` с помощью метода `addActionListener()`

в строке ❶. Следующие несколько строк кода созданы внутри скобок функции `addActionListener()`.

Первое, что мы видим внутри метода `addActionListener()`, — это ключевое слово `new`, дающее нам знать, что программа Eclipse создает новый объект класса `ActionListener` во фрагменте ❷. Однако этот новый объект отличается от уже знакомых нам. Дело в том, что это *анонимный внутренний класс*. Это означает, что не существует переменной, которая ссылается на него, и он создается полностью *внутри* класса `GuessingGame`. Анонимные внутренние классы могут помочь нам быстрее программировать, поскольку нет необходимости создавать отдельный класс для обработки коротких быстрых событий, таких как щелчок кнопки. Вместо этого мы можем вставить код для обработчика событий туда, где мы создаем кнопку. В нашем случае `ActionListener` будет прослушивать только щелчок по кнопке, так что это идеальный кандидат на роль анонимного внутреннего класса.

Среда разработки Eclipse создает каркас для анонимного внутреннего класса, но нам все равно придется запрограммировать поведение кнопки. Этот следующий раздел кода будет находиться внутри фигурных скобок метода `actionPerformed()` в строке ❸. В методе `actionPerformed()` мы опишем действия, которые должно производить приложение, когда пользователь нажал кнопку `btnGuess`. Метод принимает `ActionEvent` как аргумент, который является событием или действием пользователя, которое прослушивает слушатель. В данном случае мы назвали переменную `ActionEvent e`, но вы можете использовать любое имя переменной, какое хотите. Когда пользователь вводит свой ход и нажимает кнопку **Guess!**, это действие назначается переменной `e`, а нам необходимо проверить его ход с помощью ранее созданного метода `checkGuess()`:

```
btnGuess.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        checkGuess();
    }
});
```

Благодаря среде Eclipse нам нужно было лишь добавить команду `checkGuess()`. Возможность автоматического завершения кода сэкономила нам время и помогла избежать опечаток.

После того, как мы закончили этот раздел кода, нам осталось лишь сообщить приложению, что делать при запуске. Сохраните сделанную на текущий момент работу.

Настройка окна графического интерфейса

Прежде чем запустить игру в первый раз, нам нужно сообщить программе, как настроить окно графического интерфейса. Это включает в себя создание графического интерфейса пользователя, запуск новой игры с помощью вызова метода `newGame()` и установки желаемых размеров ширины и высоты окна.

Внутри фигурных скобок метода `main()` недалеко от конца файла добавьте следующие строки кода:

```
        public static void main(String[] args) {
❶           GuessingGame theGame = new GuessingGame();
❷           theGame.newGame();
        }
    }
```

Строка кода ❶ создает новый объект класса `GuessingGame` с именем `theGame`. Строка ❷ запускает игру с новым случайным числом.

Теперь давайте дадим компилятору Java представление о нужном нам размере окна. Для настольного приложения с графическим интерфейсом пользователя мы можем указать размер фрейма приложения с помощью метода `setSize()`, введя ширину и высоту окна, которое мы хотим создать, следующим образом:

```
        public static void main(String[] args) {
            GuessingGame theGame = new GuessingGame();
            theGame.newGame();
            theGame.setSize(new Dimension(450,300));
```

Для метода `setSize()` требуется параметр типа `Dimension`, встроенный в библиотеку `java.awt` (`.awt` — это аббревиатура *абстрактного оконного инструментария* (abstract window toolkit)). Ключевые слова `new Dimension()` в круглых скобках метода `setSize()` сообщают компилятору Java о создании объекта типа

Dimension с шириной 450 пикселей и высотой 300 пикселей. Но мы еще не импортировали библиотеку java.awt, поэтому среда разработки Eclipse не знает, к чему относится Dimension. Поэтому редактор Eclipse сообщит вам, что в этой строке есть ошибка, подчеркнув имя класса Dimension красным цветом. Если вы в редакторе щелкнете мышью по слову Dimension, программа Eclipse отобразит контекстное меню действий, как показано на рис. 3.19. (Если контекстное меню действий не отображается, нажмите сочетание клавиш **Ctrl+1** (или **⌘+1** в macOS), установив указатель мыши на слово Dimension.)

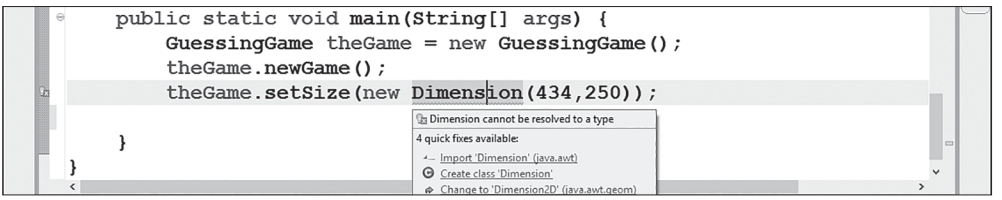


Рис. 3.19. Eclipse отображает контекстное меню действий для введенного кода; выберите верхнее действие, чтобы импортировать нужный класс

Мы хотим использовать класс Dimension, который включен в пакет java.awt, поэтому нам в данном случае нужно выбрать верхнее действие, **Import 'Dimension' (java.awt)** (Импортировать 'Dimension' (java.awt)). Щелкните мышью по данному пункту, и среда разработки Eclipse добавит java.awt.Dimension к списку импортированных классов в верхней части файла.

```
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.SwingConstants;
import java.awt.Dimension;
import java.awt.Font;
import javax.swing.JTextField;
```

Наконец, добавьте в метод main() следующую строку, чтобы приложение появилось на экране:

```
        theGame.setSize(new Dimension(450,300));
        theGame.setVisible(true);
    }
```

Эта команда вызывает метод `setVisible()` для объекта `theGame` класса `GuessingGame`. Помните, что `GuessingGame` расширил класс `JFrame`, поэтому `theGame` также является потомком родительского класса `JFrame`. Нужно присвоить логическое значение `true` свойству видимости основного фрейма `JFrame`, содержащего графическую версию игры.

Существует два возможных *логических (булевых) значения*: `true` (истина) и `false` (ложь) (все буквы строчные), и они используются в логических выражениях. *Логические выражения* — это любой набор аргументов, значение которого может быть только `true` или `false`. Инструкции `if`, которые мы использовали в методе `checkGuess()`, например в выражении (`guess < theNumber`), являются лишь одним из примеров, с которым мы уже встречались. Когда мы вычисляем это выражение и проверяем, меньше ли значение переменной `guess`, чем значение `theNumber`, выражение принимает значение либо `true` (означающее, что `guess` меньше, чем `theNumber`), либо `false` (`guess` не меньше, чем `theNumber`).

Результат этого логического выражения определяет, будет ли выполняться инструкция, следующая за инструкцией `if`.

Готовый метод `main()` для класса `GuessingGame` выглядит следующим образом:

```
public static void main(String[] args) {
    GuessingGame theGame = new GuessingGame();
    theGame.newGame();
    theGame.setSize(new Dimension(450,300));
    theGame.setVisible(true);
}
```

Пора играть!

После того как мы закончили писать программу, сохраните файл *GuessingGame.java*. После этого щелкните мышью по кнопке запуска или выберите команду меню **Run ⇒ Run** (Выполнить ⇒ Выполнить). Появится окно игры.

Попробуйте сделать несколько ходов и посмотрите, работает ли приложение. Пока вы играете, проверьте каждый компонент и убедитесь, что он работает. Не забудьте проверить кнопку

Guess!, метку в нижней части окна и текстовое поле. Если вы обнаружите проблемы, остановитесь и проверьте свой код. Обратите внимание на то, когда или где возникла ошибка. Тем самым вы сузите источник возникновения ошибки. Опыт в обнаружении ошибок появляется вместе с навыком программирования. На рис. 3.20 показано, как должно выглядеть корректно работающее приложение.

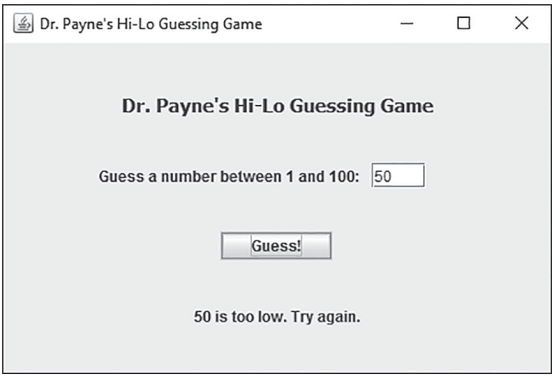


Рис. 3.20. Кнопка **Guess!** работает

Если ваше приложение работает без ошибок, можно перейти к следующему шагу.

В настоящий момент в вашу игру можно играть, но только до победы. В текущей версии приложения нет кода, который отвечает за начало перезапуск игры при выигрыше пользователя.

Давайте настроим приложение так, чтобы мы могли продолжать играть даже после того, как выиграем в первый раз, а затем сделаем несколько настроек в пользовательском интерфейсе, улучшающих процесс игры.

Добавление возможности повторной игры

Точно так же, как мы это делали в первой консольной версии игры, давайте добавим возможность повторной игры. Мы уже создали метод `newGame()`, но вызвали его только раз из метода `main()` во время настройки игры для ее первоначального отображения.

Нам нужно либо изменить либо графический интерфейс, либо поведение приложения. Мы можем добавить кнопку **Play Again**, которую пользователь сможет нажать, если захочет сыграть еще

раз, или можем просто заставить приложение автоматически запускать новую игру после выигрыша пользователя.

Преимущество добавления кнопки **Play Again** заключается в том, что она понятна пользователю. Однако мы не хотим, чтобы эта кнопка постоянно отображалась в окне графического интерфейса, поскольку она нужна, только когда пользователь угадал загаданное число. Один из способов решить эту проблему состоит в добавлении скрытой во время игры кнопки. Эта кнопка становится видимой только после того, как пользователь угадал загаданное компьютером число. Чтобы не усложнять игру, сейчас мы не будем этого делать, но если вам интересно попробовать, обратитесь к задаче № 2 в конце этой главы.

Вместо добавления кнопки мы изменим поведение приложения так, чтобы машина загадывала новое число автоматически после победы пользователя в предыдущем раунде. Это потребует изменения только одного раздела программы, а именно команды `else`, которая обрабатывает ход игрока в случае выигрыша. Если пользователь угадал число, новая игра должна начинаться автоматически. Такой подход кажется интуитивно правильным — никаких дополнительных кнопок, никаких дополнительных решений. Если пользователь считает, что игра окончена, он может просто закрыть окно. Добавим такое поведение приложения, изменив следующие строки:

```
        else
            message = guess + " is correct. You win!“;
```

Чтобы начать новую игру вызовем метод `newGame()`. Чтобы добавить вызов метода `newGame()` внутри инструкции `else`, мы сгруппируем две инструкции (присвоение значения переменной `message` и вызов метода `newGame()`) в один блок:

```
    else {
        message = guess + " is correct. You win! Let's play again!";
        newGame();
    }
```

Скобки группируют эти две инструкции в единый блок кода, который будет выполняться вместе, если программа пойдет по пути инструкции `else`. Без фигурных скобок будет выполняться только первая строка после ключевого слова `else`, а метод `newGame()` будет выполняться после выполнения всей конструкции `if-else`.

Это приведет к тому, что компьютер будет загадывать новое число после каждого хода игрока.

Обратите также внимание, что мы добавили текст `"Let's play again!"` в конец строки сообщения. Таким образом, пользователь поймет, что компьютер загадал новое число и, чтобы сыграть снова, нужно просто сделать новый ход. Внесите эти изменения, сохраните программу и запустите ее снова, чтобы увидеть новую версию игры.

Улучшение UX-дизайна

Пока вы играли в игру, вам могли прийти в голову идеи, как можно улучшить интерфейс пользователя. Например, было бы удобно, если бы приложение понимало, что пользователь сделал свой ход, либо по нажатию кнопки **Guess!**, либо по нажатию на клавиатуре клавиши **Enter**. Такое изменение обеспечило бы более естественный и интуитивно понятный *опыт взаимодействия с пользователем* (user experience — *UX*).

Другая проблема заключается в том, что, когда мы собираемся сделать новый ход, мы должны выделить уже введенное число (и нажать клавишу **Del** или **Backspace**), чтобы удалить предыдущий ход. Кроме того, надо щелкнуть мышью в текстовом поле, потому что фокус переключается на кнопку «**Guess!**». Чтобы улучшить пользовательский интерфейс, можно возвращать курсор в текстовое поле после каждого хода и автоматически выделять или удалять содержимое поля, чтобы новый ход заменял старый.

Давайте разберемся с этими двумя моментами.

Ввод хода нажатием клавиши Enter

Первое улучшение, которое мы сделаем — это предоставление возможности пользователю вводить свой ход с помощью нажатия клавиши **Enter**. Сначала перейдите на вкладку **Design** (Конструктор) и щелкните правой кнопкой мыши по кнопке **Guess!**. В контекстном меню выберите пункт **Add event handler ⇒ action ⇒ actionPerformed** (Добавить обработчик события ⇒ action ⇒ actionPerformed).

Так же, как и при создании обработчика событий для кнопки `btnGuess`, среда разработки Eclipse автоматически добавит код для создания слушателя действий для текстового поля `txtGuess`, как

показано в следующем фрагменте кода. Как и раньше, мы добавим метод `checkGuess()` внутрь скобок метода `actionPerformed()`. Единственное событие, которое мы хотим обработать внутри текстового поля, — это нажатие пользователем клавиши **Enter**.

```
txtGuess.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        checkGuess();
    }
});
```

После того как вы добавите эту строку, сохраните файл и запустите его снова. Теперь нажатие клавиши **Enter** активирует метод `checkGuess()`. Кнопка «**Guess!**» также работает, поэтому пользователь может выбрать, взаимодействовать ли ему только с клавиатурой или и с мышью тоже.

Автоматическое удаление предыдущих ходов

Теперь давайте решим вопрос с необходимостью повторного щелчка мышью в текстовом поле для удаления предыдущих ходов. Чтобы решить эту проблему, нам нужно изучить несколько новых методов работы с объектами `JTextField`, к которым относится текстовое поле `txtGuess`.

Сначала мы должны решить, когда мы хотим, чтобы текст был выбран. В нашем случае, позволить пользователю снова угадывать целесообразно после проверки его текущего хода. В коде это будет реализовано в конце метода `checkGuess()`.

Итак, мы решили, что код, помещающий курсор назад в текстовое поле `txtGuess` и выделяющий предыдущий ход, должен быть добавлен в конец метода `checkGuess()`. Но как мы узнаем, какие команды помещают курсор в текстовое поле и выделяют весь текст? Пора воспользоваться возможностью автозавершения в программе Eclipse. Введите имя объекта, `txtGuess`, затем оператор (`.`), а среда разработки Eclipse отобразит список доступных методов, как показано на рис. 3.21.

Как видите, существует множество методов для работы с объектами класса `JTextFields`, такими как `txtGuess`, но сейчас нам нужен метод `requestFocus()`. В документации к этой функции говорится: «Требует, чтобы этот компонент получил фокус ввода». В данном

случае под *фокусом* понимается курсор. Этот метод будет делать именно то, что мы хотим, а именно требовать, чтобы курсор был помещен назад в текстовое поле перед каждым ходом пользователя.

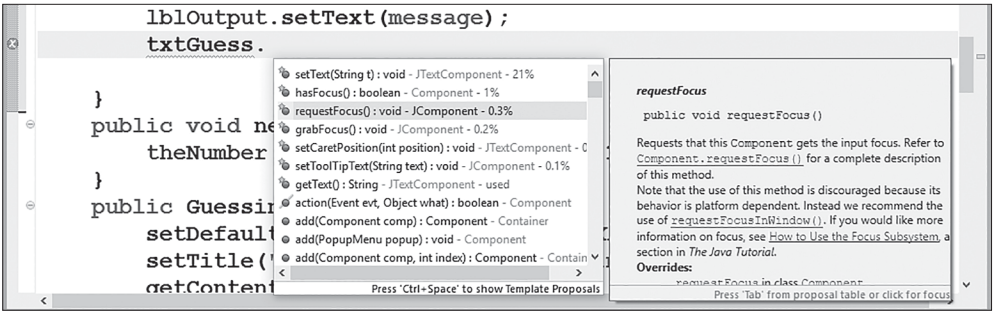


Рис. 3.21. Функция автозавершения отображает все возможные методы для объекта `txtGuess` класса `JTextField`

Наконец, мы хотим выделять весь текст предыдущего хода пользователя, чтобы его новый ход автоматически заменял старый. Чтобы выделить весь текст в текстовом поле, используйте метод `selectAll()`, который также указан в списке автозаполнения среды Eclipse. Добавьте эти две строки в конец метода `checkGuess()` сразу после строки `lblOutput.setText(message)` и прямо перед закрывающей скобкой:

```
        lblOutput.setText(message);
        txtGuess.requestFocus();
        txtGuess.selectAll();
    }
```

Теперь самое время сохранить ваше приложение. При новом запуске игры вы должны обнаружить гораздо более логичное и интуитивно понятное взаимодействие с компьютером. Наконец у нашей игры элегантный и простой в использовании графический интерфейс, обеспечивающий простое взаимодействие с пользователем. В нашу игру легко играть, но сделать ее было непросто. Она похожа на игры, в которые мы обычно играем, однако эту мы создали с нуля, используя язык программирования Java и инструментарий Swing.

Мы создали отличное приложение, но пока не учли одной важной вещи: безопасности. Возможно, вы спросите: «Что? Безопасность? Я думал, что мы просто создаем забавную игру с графическим

интерфейсом». Все так — программа работает красиво, но только до тех пор, пока пользователь вводит корректные числа.

Что произойдет, если пальцы пользователя соскользнут и нажмут клавишу **Enter** или кнопку **Guess!**, до того, как будет введен ход? Что, если игрок введет в качестве своего хода какой-то текст, не являющийся целым числом, например `"blah"`? На рис. 3.22 показано, что произойдет в программе Eclipse, если пользователь делает некорректный ввод.

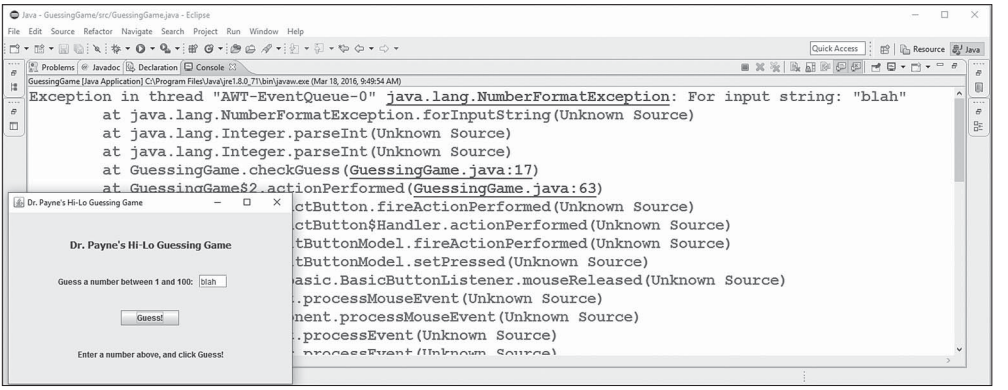


Рис. 3.22. Если пользователь ввел некорректные данные в поле ввода, например «`blah`», произойдут неприятности

Как видите, наше консольное окно заполнилось ошибками. Некорректный ввод привел к *исключению* — непредвиденной или ошибочной программной ситуации. Первая строка вывода на рис. 3.22 говорит о том, что было выброшено исключение `NumberFormatException`, поскольку строка `"blah"` не распознается как число. Нужно научиться обрабатывать исключения в создаваемых вами программах.

Обработка некорректного ввода пользователя

Язык программирования Java обеспечивает безопасный способ ожидания и работы с исключениями, неприятными ситуациями, которые могут возникнуть при запуске программы. Помимо некорректного ввода пользователя, который мы только что обсудили, исключения могут включать в себя отсутствующие файлы (файл был удален, или вы случайно отключили USB-накопитель) или сетевые проблемы (например, отсутствующее соединение или ошибки сервера).

Мы называем код, созданный для работы с исключениями, *обработчиком исключений*. Обработчик исключений имеет дело с исключениями. Но как мы узнаем, когда может произойти исключение? Должны ли мы предсказывать все плохое, что может случиться во время работы программы?

Большинство ситуаций, при которых могут возникнуть ошибки, — это конкретные типы событий, такие как ввод данных пользователем, открытие либо сохранение файла или сетевой доступ. Например, подумайте о перечисленных выше исключениях. Некорректный ввод может произойти в любой момент, когда мы просим пользователя ввести данные определенного типа. Отсутствие файла может стать проблемой, когда мы пытаемся его прочитать или записать в него что-то; например при просмотре и сохранении фотографий из специального приложения. Если мы сможем предсказать возможные исключения и написать обработчики исключений для устранения этих неполадок, наш код будет более надежным и безопасным, более защищенным от злоумышленников, пожелающим вызвать сбой вашей программы, введя некорректные данные.

В нашем случае исключение произошло, когда мы попросили пользователя ввести число, но вместо этого он ввел текст. Строго говоря, исключение было выброшено, когда мы попытались извлечь целое число из строки, в которой такого числа не было. Ввод пользователя является типичным источником исключений, как случайных, так и преднамеренных. В качестве примера последних можно привести усилия хакера, пытающегося сломать вашу программу.

Язык программирования Java предлагает нам простой способ обработки исключений с помощью команд `try-catch`. Идея блока `try-catch` состоит в том, что мы хотим, чтобы программа попыталась (`try`) сделать что-то, что, как мы знаем, может вызвать исключение. Если исключение все-таки возникнет, мы хотим его поймать (`catch`) и обработать, не позволяя ему сломать нашу программу. Блок `catch` выполняется только тогда, когда происходит исключение. Последней частью `try-catch` является дополнительный блок `finally`. Если блок `finally` существует в команде `try-catch-finally`, он будет выполняться всегда, вне зависимости от того, было ли выброшено исключение. Если исключений не произошло, блок `finally` все равно будет выполняться после завершения работы блока `try`. Если исключение возникает, блок `finally` будет выполняться после завершения блока `catch`.

Формат try-catch обычно выглядит примерно так (не вводите его в свой код, это просто пример):

```
try {
    // Что-то, что может вызвать исключение. Например, некорректный ввод
} catch (Exception e) {
    // Обработка или восстановление после исключения.
    // Например, просьба ввести корректные данные
} finally {
    // Любой завершающий код; он всегда будет выполнен в конце,
    // независимо от наличия или отсутствия исключения
}
```

Обратите внимание, что мы группируем инструкции для каждой части команды try-catch-finally с помощью фигурных скобок, как и для инструкций if-else и цикла. Для элемента catch требуется параметр Exception (исключение). Это может быть особое исключение, например NumberFormatException, или мы можем использовать базовый класс Exception (исключение). Этот параметр получит информацию об Exception, например номер строки, в которой произошло исключение.

Чтобы правильно поместить блоки try-catch-finally в код нашей игры, нам нужно продумать несколько моментов. Во-первых, где может произойти исключение (и обернуть этот код блоком try). Во-вторых, что мы хотим сделать, если исключение будет выброшено (и поместить эти действия в блок catch). Наконец, что мы хотим сделать после того, как блоки try или catch закончатся (этот код войдет в блок finally).

Поскольку проблема с вводом возникла тогда, когда мы попытались извлечь целое число из введенного пользователем текста, мы обернем этот код блоком try. Напишем ключевое слово try и открывающую скобку прямо перед строкой, которая извлекает целое число из текста:

```
public void checkGuess() {
    String guessText = txtGuess.getText();
    String message = "";
    try {
        int guess = Integer.parseInt(guessText);
        if (guess < theNumber)
```

Если пользователь не ввел число, мы не можем проверить, больше оно или меньше загаданного, поэтому раздел `try` будет заканчиваться после пары инструкций `if-else`. Поместите закрывающую скобку блока `try` после закрывающей скобки инструкции `else`, как показано в следующем листинге. Дальше должно идти ключевое слово `catch`. В этой игре, если ввод пользователя некорректен, мы просто хотим попросить его ввести целое число от 1 до 100. Мы можем сделать это с помощью строковой переменной `message` (сообщение), которая будет показана позже в метке `lblOutput`. Таким образом, блок `catch` должен выглядеть следующим образом:

```
            else {
                message = guess + " is correct. You win! Let's play again!";
                newGame();
            }
        } catch (Exception e) {
            message = "Enter a whole number between 1 and 100.";
        }
```

И последнее, что мы хотим сделать, вне зависимости от того, было выброшено исключение или нет, — показать итоговое сообщение, потребовать фокус для текстового поля и выделить предыдущий ход для подготовки к вводу нового. Эти три команды войдут в блок `finally`, который будет помещен после закрывающей скобки только что добавленной команды `catch`:

```
        } finally {
            lblOutput.setText(message);
            txtGuess.requestFocus();
            txtGuess.selectAll();
        }
    }
    public void newGame() {
```

Вы могли заметить, что инструкции внутри каждого из блоков `try-catch-finally` начинаются с отступа, тем самым код становится более удобным для чтения. Помните, что в редакторе Eclipse есть способ, который автоматически все аккуратно выровняет после внесения каких-либо изменений. Просто выделите весь код, а затем выберите команду меню **Source ⇒ Correct Indentation** (Исходный код ⇒ Исправить отступы).

Сохраните ваше приложение и запустите его несколько раз, чтобы протестировать пользовательский интерфейс. Попробуйте ввести что-то отличное от целого числа и убедитесь, что обработка исключений работает так, как надо. Если вы столкнулись с ошибками, которые мешают компиляции, проверьте расстановку фигурных скобок. Каждый раз, когда мы добавляем блоки вокруг существующего кода, скобки — это наиболее распространенное место для появления ошибок. Добавив команды `try-catch-finally`, вы получаете более надежную, предсказуемую и безопасную версию приложения с графическим интерфейсом.

Безопасное программирование — это не просто прекрасно, часто без этого нельзя написать хорошее программное обеспечение. Для нашей игры последствия ошибок в коде не кажутся критичными. Но подумайте о программе, запущенной на медицинском оборудовании, спутнике, автомобиле, самолете или электростанции, поставляющей электричество в ваш дом. Защищенный код жизненно важен для нашей защиты и безопасности. И даже небольшая игра или приложение на вашем смартфоне, в которой используется незащищенный код, могут открыть ваши данные хакерам. Безопасное программирование — это всегда правильно.

Что вы изучили

В версии игры с графическим интерфейсом пользователя мы углубили наши навыки программирования на языке Java, которые включают в себя следующее:

- создание окна графического интерфейса пользователя путем импорта пакета `javax.swing`;
- настройка приложения с графическим интерфейсом пользователя в среде разработки Eclipse;
- проектирование графического интерфейса с помощью редактора WindowBuilder;
- использование панели **Properties** (Свойства) для изменения свойств компонентов графического интерфейса;
- единообразное именование компонентов графического интерфейса для удобства использования в коде на языке Java;
- соединение элементов графического интерфейса с переменными на языке Java и исходным кодом;

- получение текста из текстового поля `JTextField` и его использование в программе;
- прослушивание пользовательских событий, таких как щелчки по кнопкам;
- улучшение взаимодействия с пользователем или UX, путем тестирования приложение с точки зрения пользователя;
- обработка исключений с помощью `try-catch-finally`.

Дополнительные задачи

Попробуйте выполнить следующие задания. Таким образом, вы повторите пройденный материал, попрактикуетесь в программировании и получите новые навыки. Если у вас возникнут проблемы, вы можете загрузить примеры решений на сайте по адресу **https://eksmo.ru/files/learn_java_easy_way.zip**.

Задача № 1: Вывод пользователю числа сделанных им ходов

Измените сообщение о победе пользователя так, чтобы оно сообщало ему, с какого раза он смог угадать загаданное число. Например, следующим образом:

```
62 is correct! You win after 7 tries!
```

Для выполнения этой задачи вам нужно создать новую переменную (например, `numberOfTries`), в которой будет храниться число попыток пользователя. После успешного завершения метода `checkGuess()` число попыток пользователя должно увеличиваться на единицу, а после того, как число угадано, отображаться на экране.

Как вариант, можно начать игру с семью или восемью попытками, а затем вычитать попытку после каждого хода пользователя. Когда последняя попытка была использована, можно сообщить пользователю, что он проиграл, показать загаданное число и начать новую игру. Попробуйте сделать это.

Задача № 2: Отображение и скрытие кнопки запуска повторной игры

Создавая графический интерфейс для нашей игры в этой главы, мы решили не делать кнопку «**Play Again**», поскольку большую часть времени она не использовалась бы и только загромождала бы интерфейс. Эта кнопка нужна только в конце игры. Однако оказывается, мы можем скрыть компонент графического интерфейса с помощью метода `setVisible(false)`. Затем мы можем показать его снова с помощью метода `setVisible(true)`, как мы делаем с нашим фреймом `JFrame`.

Прежде всего, надо добавить кнопку **Play Again**, назвав ее `btnPlayAgain`, в макет графического интерфейса. В коде после ее создания можно вызвать метод `btnPlayAgain.setVisible(false)` и скрыть кнопку. В команде `else`, которая выполняется в случае победы игрока, в методе `checkGuess()` мы должны вызвать метод `btnPlayAgain.setVisible(true)`, чтобы показать кнопку вместо запуска новой игры. После того, как вы напишите этот код, не забудьте выполнить двойной щелчок по этой кнопке в режиме предварительного просмотра графического интерфейса, чтобы добавить слушатель событий, вызвать `newGame()` в этом слушателе и снова скрыть кнопку.

Перед началом работы сделаем одно важное замечание: в отличие от другой кнопки (`btnGuess`), вам нужно изменить свойство `visible` для скрытия и отображения кнопки в нескольких местах кода. Как и в случае с текстовым полем `txtGuess` и меткой `lblOutput`, вам нужно объявить `private JButton btnPlayAgain` в верхней части класса. При этом убедитесь, что вы не объявляете эту кнопку еще раз ниже.

Когда у вас все заработает, игра должна начинаться со скрытой кнопкой **Play Again**. Как только вы выиграете игру, кнопка должна появиться. Когда вы нажмете кнопку, начнется новая игра, и кнопка снова должна исчезнуть. Если вы придумаете еще какие-нибудь улучшения, смело реализуйте их!

Задача № 3: Создание приложения с графическим интерфейсом для игры MadLib

Вспомните программу игры в чепуху *MadLibs.java*, которую вы создали в главе 2 (задача № 3), и создайте новое приложение *MadLibGUI.java*, в котором пользователю предлагается ввести

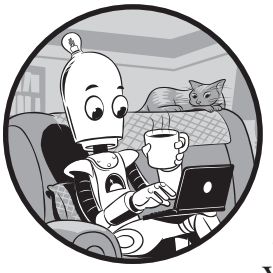
несколько слов в графическом интерфейсе с метками и текстовыми полями, например `txtColor`, `txtPastTenseVerb`, `txtAdjective` и `txtNoun`. Добавьте кнопку, которую пользователь должен нажать, чтобы создать историю в стиле чепухи.

Изучите и попробуйте использовать еще несколько свойств элементов графического интерфейса в редакторе WindowBuilder, включая цвета фона, шрифты, начальные значения текста и многое другое. Когда пользователь нажимает кнопку, программа должна отображать готовую чепуховую историю в метке `JLabel` или текстовом поле `JTextArea`, как показано на рис. 3.23.

Рис. 3.23. Приложение MadLib с графическим интерфейсом пользователя

Глава 4

СОЗДАНИЕ ВАШЕГО ПЕРВОГО ПРИЛОЖЕНИЯ ДЛЯ ANDROID



Чтобы создать мобильную версию игры «Больше-Меньше», мы будем использовать среду разработки Android Studio. Если вы не установили ее во время чтения главы 1, вернитесь к разделу «Установка IDE Android Studio для разработки мобильных приложений» и установите его. Как и в случае с графической версией игры, мы создадим пользовательский интерфейс, показанный на рис. 4.1.

Приложения, которые вы создавали до сих пор, могли быть запущены только на компьютере, а приложения для операционной системы Android могут работать на любом устройстве Android, включая смартфоны, планшеты, умные часы, телевизоры и так далее. Если у вас нет устройства, работающего под управлением операционной системы Android, ничего страшного — вы все равно сможете программировать для этих устройств, так как эмулятор, который поставляется вместе с программой Android Studio, позволяет имитировать запуск мобильного устройства.

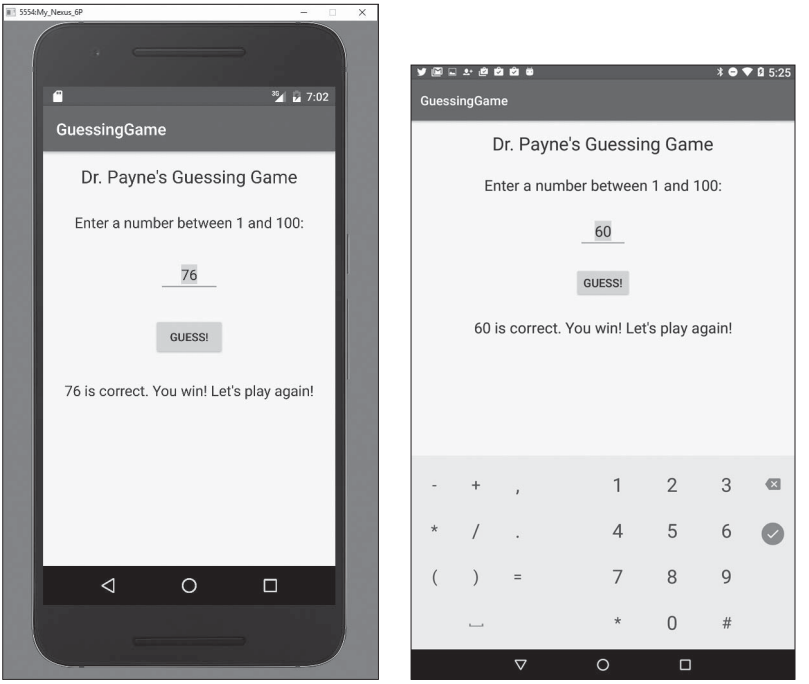


Рис. 4.1. В этой главе наша игра станет мобильным приложением. Приложение работает на эмулированном смартфоне Android (слева) и на реальном планшете Android (справа)

На рис. 4.1 показано наше приложение, работающее на эмуляторе устройства Android, имитирующем смартфон Nexus 6P, и на планшете с ОС Android. В обеих версиях используется один и тот же код, и, что еще удобнее, мы можем повторно использовать много фрагментов кода на языке Java, который мы создали для версий программы для компьютеров, поскольку операционная система Android создана на языке программирования Java!

Обратите внимание, что эмулятор и настоящее устройство выглядят очень похожими, с некоторыми незначительными различиями, учитывающими различия в размерах экрана между смартфоном и 7-дюймовым планшетом. Точно так же вы можете написать приложение на языке Java для операционной системы Windows и запустить его под операционными системами macOS или Linux, написать мобильное приложение для устройства Android на языке Java и запустить его на любом из миллионов устройств на Android. Давайте создадим наше первое мобильное приложение для устройства Android!

Создание нового проекта в среде разработки Android Studio.

Когда вы запускаете программу Android Studio в первый раз, может потребоваться несколько минут для ее обновления и запуска. После успешного запуска вы увидите экран, подобный показанному на рис. 4.2. Выберите пункт **Start a new Android Studio project** (Запустить новый проект в программе Android Studio).

Назовите свой новый проект **GuessingGame** (без пробела). Если у вас есть веб-сайт, вы можете указать его адрес в поле **Company Domain** (Домен компании), как показано на рис. 4.3. В противном случае оставьте в качестве домена значение по умолчанию. Затем выберите местоположение проекта. Я создал папку с именем *AndroidProjects*, чтобы все было упорядочено. Сделайте то же самое.

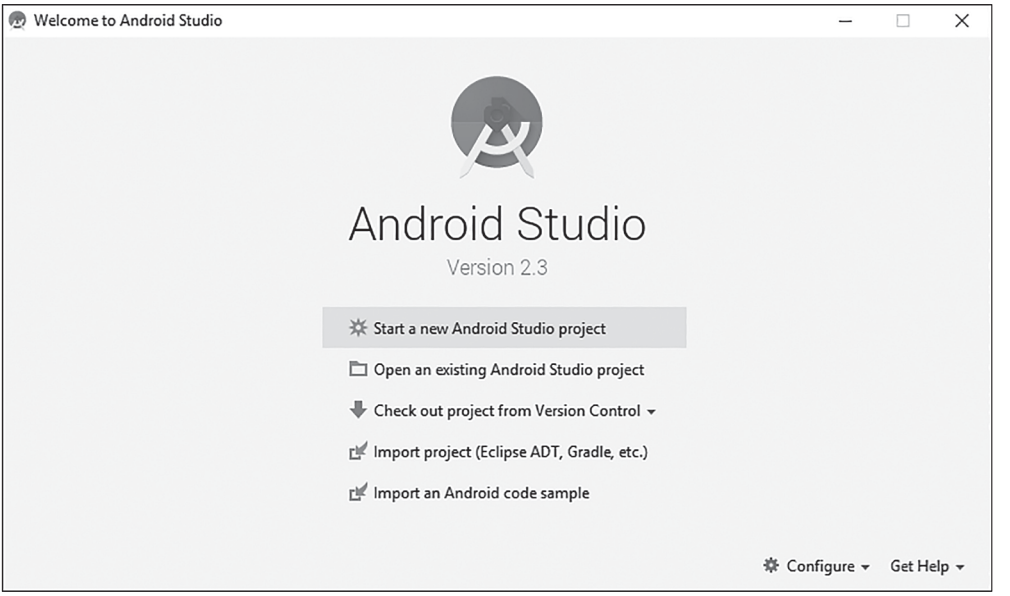


Рис. 4.2. Когда Android Studio закончит обновление, выберите пункт **Start a new Android Studio project**

Нажмите кнопку **Next** (Далее) после того, как ввели название проекта. Теперь у вас есть возможность выбрать номер версии Android, на которой должно работать приложение. Платформа Android быстро развивается по мере появления новых устройств и возможностей, поэтому нужно выбрать нужную нам версию. К счастью, среда разработки Android Studio регулярно обновляется

как последними версиями операционной системы Android, так и информацией о том, сколько устройств по-прежнему используют старые версии. Кроме того, выбирая, какие версии Android будут поддерживаться, вам придется выбрать уровень *набор средств разработки* (software development kit — *SDK*) или *программный интерфейс приложения* (application program interface — *API*). Уровни SDK и API включают инструменты, которые вы будете использовать для разработки своего приложения, но они привязаны к определенным версиям операционной системы Android. Окно **Target Android Devices** (Целевые Android-устройства), показанное на рис. 4.4, позволяет вам выбрать минимальный уровень SDK или API для каждого разрабатываемого вами приложения.

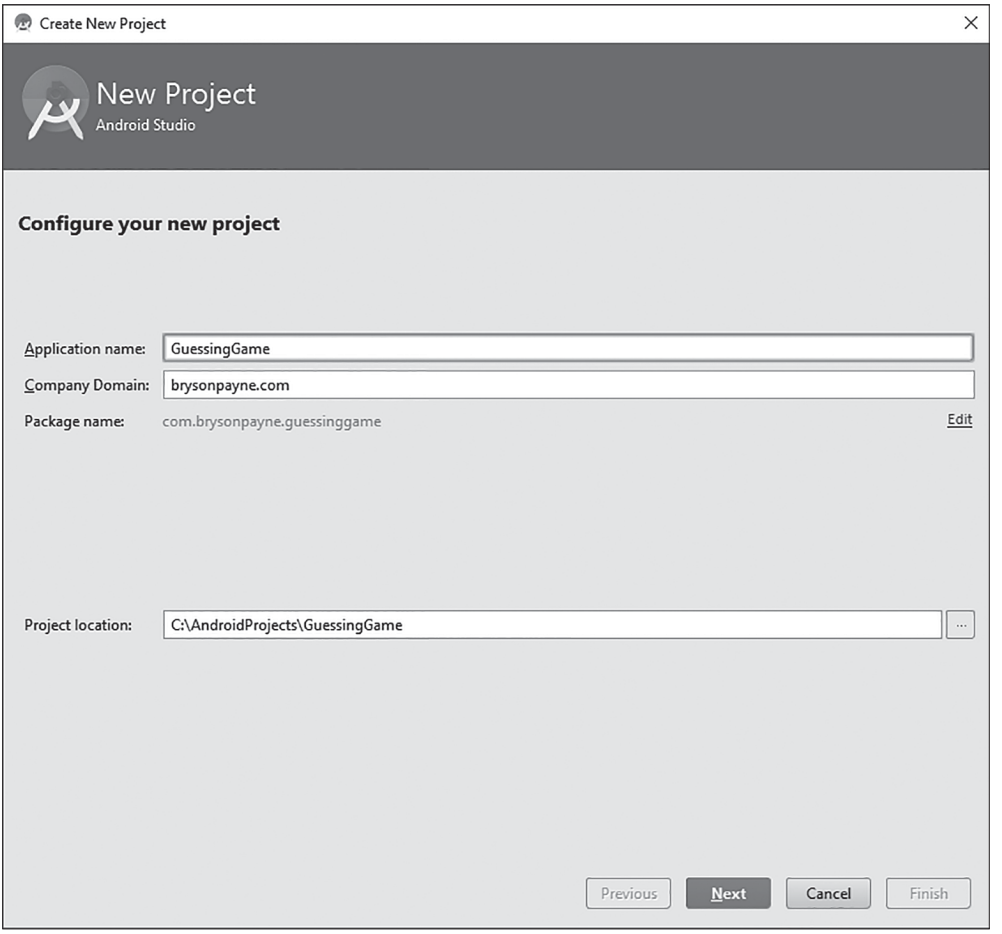


Рис. 4.3. Назовите новый проект **GuessingGame** (без пробела)

Выберите вариант **API 16 (Android 4.1, Jelly Bean)** в качестве минимальной версии SDK для нашей игры. Android Studio сообщит вам, что он работает на более чем 99% активных Android-устройств. Нажмите кнопку **Next** (Далее).

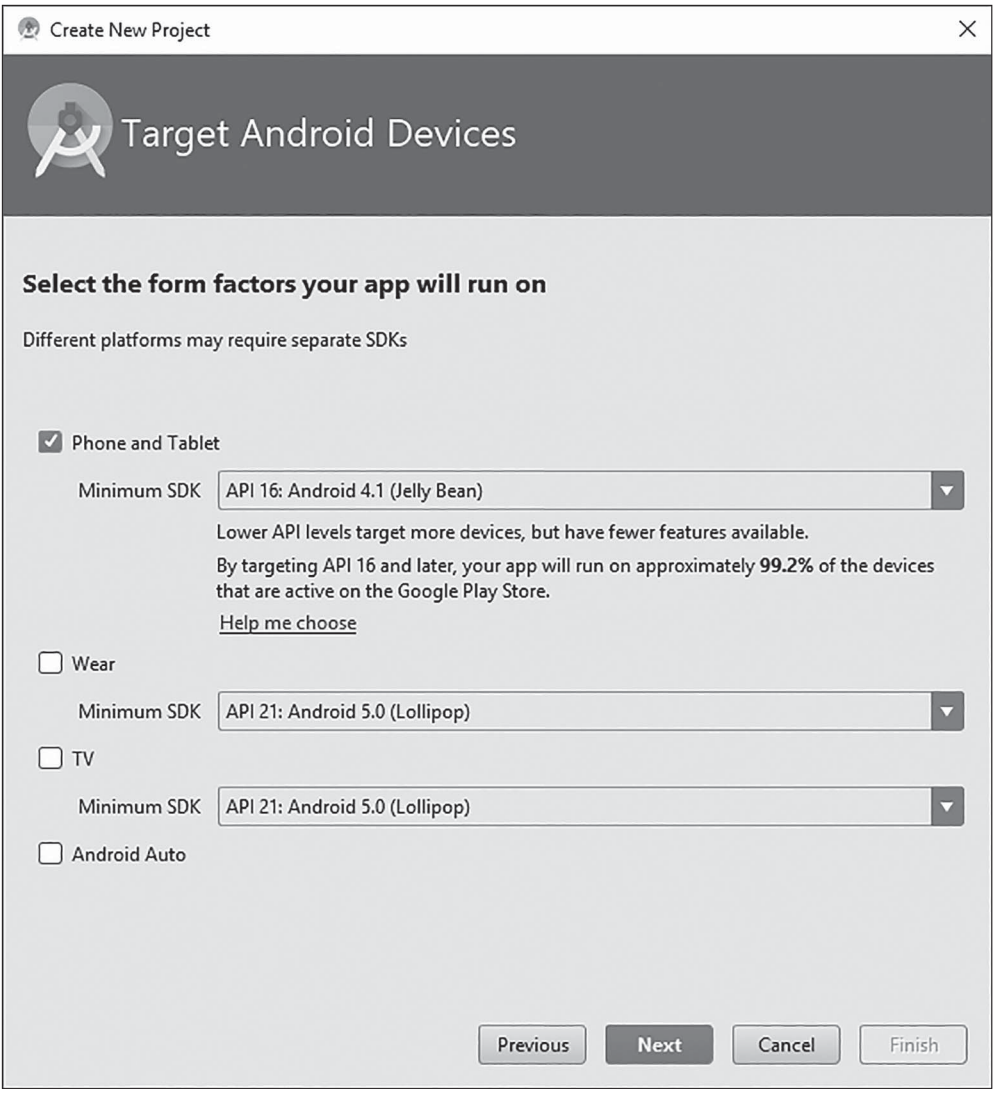


Рис. 4.4. Выберите минимальный уровень SDK для устройств Android, для которых разрабатывается приложение

Появится запрос, какие *операции* (*activity*) вы хотите добавить в свое приложение. *Операция* — это то, что пользователь может сделать. Каждая операция обычно имеет собственный экран с пользовательским интерфейсом, как показано на рис. 4.5.

Существует множество возможностей выбора основных операций для приложения, такого как Карты Google, Авторизация, Настройки, Работа с вкладками и многое другое. Набор операций Basic Activity станет хорошей основой для создания нашего игрового приложения, поэтому выберите этот вариант и нажмите кнопку **Next** (Далее).

Мы будем использовать имена, предложенные по умолчанию программой Android Studio на следующем экране (показано на рис. 4.6): *MainActivity* для названия операции и *activity_main* для названия макета. Файл операции содержит исходный код на языке Java, который запускает приложение. Макет — это отдельный файл, в котором хранится информация об интерфейсе приложения. Нажмите кнопку **Finish** (Готово), чтобы завершить настройку проекта приложения.

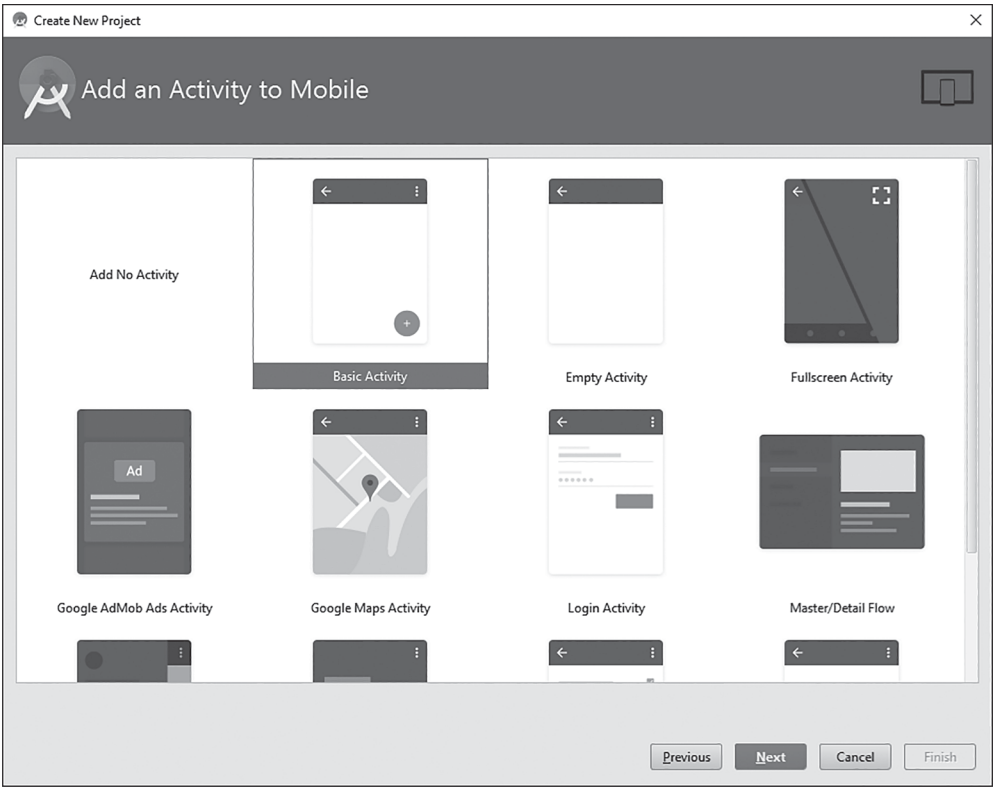


Рис. 4.5. Выбор набора операций **Basic Activity** в качестве основы приложения

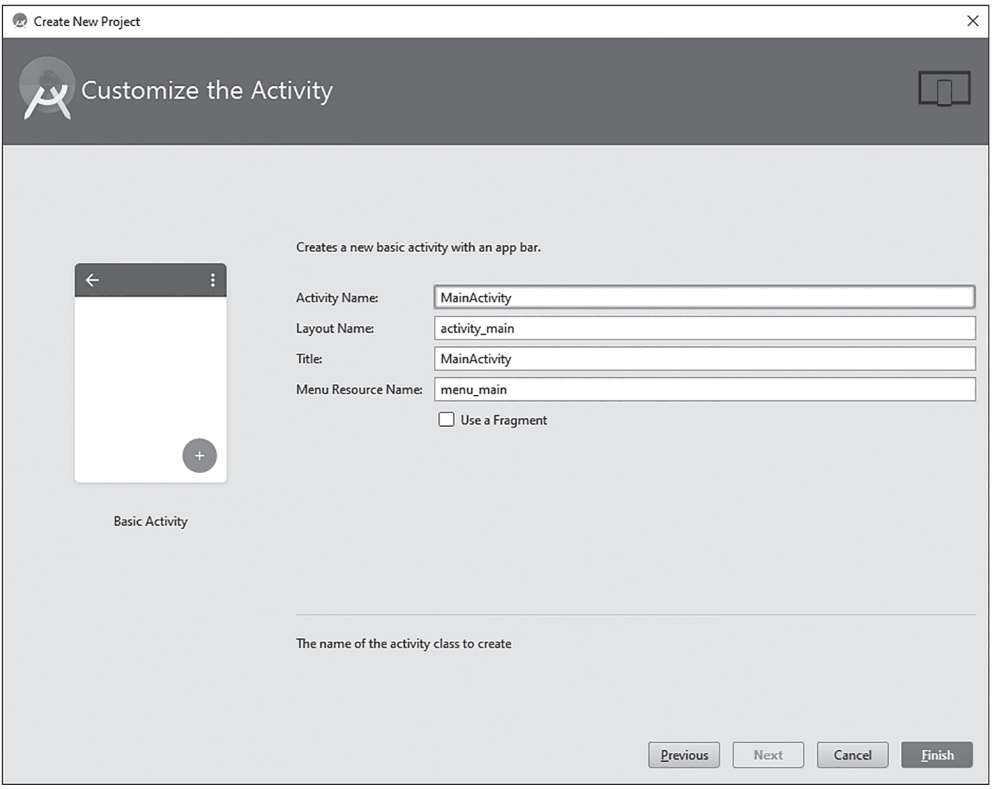


Рис. 4.6. Оставьте значения по умолчанию для названия операций и макета

Создание проекта может занять несколько минут, так как среда разработки Android Studio создает файлы проекта и настраивает приложение *GuessingGame*. После окончания загрузки программа Android Studio откроет проект в режиме по умолчанию, как показано на рис. 4.7.

Если при открытии вашего проекта появился другой экран, перейдите на вкладку **Project** (Проект) в левом верхнем углу, разверните папку *app* на панели **Project Explorer** (Обозреватель проекта), разверните папку *res* (сокращение от *resources* (ресурсы)), раскройте *layout* (макет) и дважды щелкните мышью по файлу *content_main.xml*. Вы увидите экран, схожий с изображением на рис. 4.7.

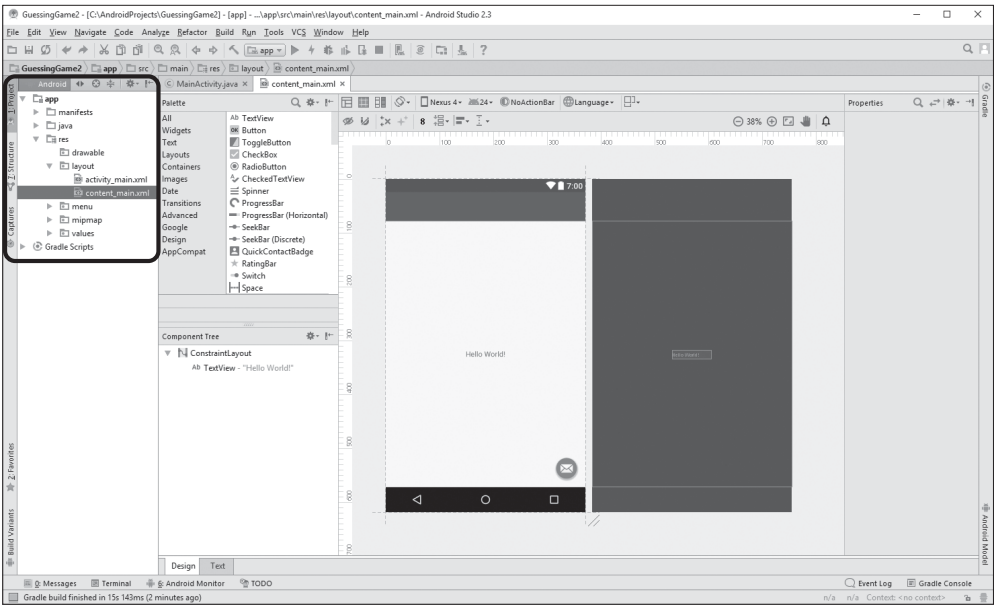


Рис. 4.7. Вид проекта в среде разработки Android Studio по умолчанию вместе с макетом мобильного графического интерфейса

Создание макета графического интерфейса в конструкторе

В приложениях для устройств Android данные макета и код приложения разделены, поэтому это приложение будет немного отличаться от приложения для компьютера, которое мы написали в главе 3. Макет описан не на Java, а на *расширяемом языке разметки* (eXtensible Markup Language, XML). К счастью, несмотря на эти отличия, для разработки графического интерфейса мы можем использовать палитру в духе drag-and-drop, похожую на используемую в редакторе WindowBuilder в программе Eclipse. Основное отличие среды разработки Android Studio заключается в том, что компоненты были названы с учетом мобильных приложений.

Как и в редакторе WindowBuilder, в программе Android Studio имеются две вкладки: одна для конструктора, другая — для исходного кода. Перейдите на вкладку **Design** (Конструктор) в нижнем левом углу панели **content_main.xml**. Вы увидите режим конструктора, показанный на рис. 4.8.

На рис. 4.8 показаны четыре панели конструктора, которые мы будем чаще всего использовать: **Project Explorer** (Обозреватель

проекта) ❶, **Palette** (Панель элементов) ❷, **Preview** (Предварительный просмотр) ❸ и **Properties** (Свойства) ❹.

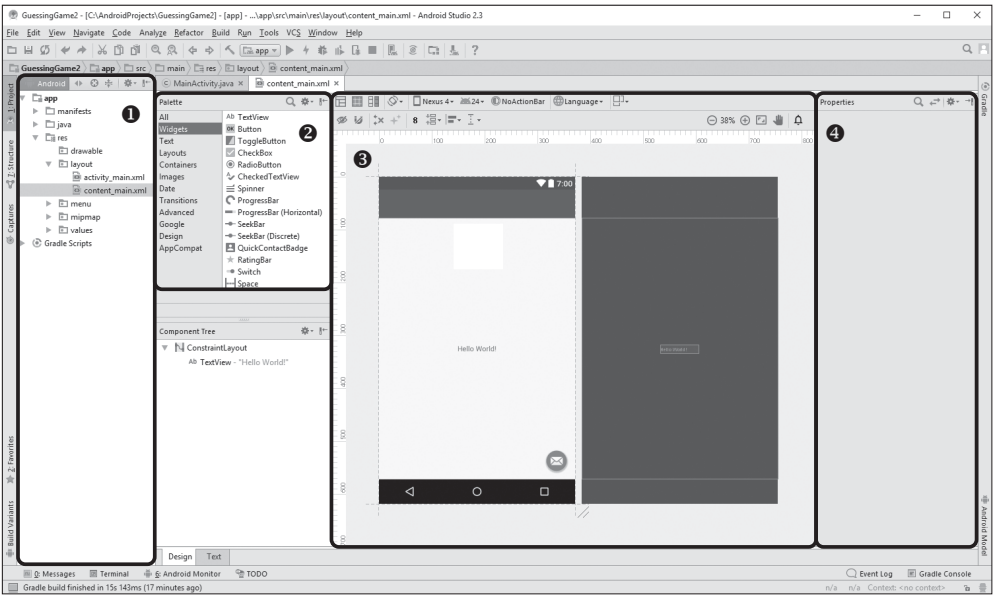


Рис. 4.8. Режим конструктора в среде разработки Android Studio очень похож на аналогичный режим в редакторе WindowBuilder в программе Eclipse

В центре панели **Preview** (Предварительный просмотр) вы увидите метку, известную как `TextView` в терминологии устройств Android, на которой написано *Hello World!*. Щелкните мышью по метке и нажмите клавишу **Del** (или щелкните правой кнопкой мыши и выберите **Delete** (Удалить) в контекстном меню), чтобы удалить ее из окна панели **Preview** (Предварительный просмотр). Затем щелкните мышью по параметру **Layouts** (Макеты) на панели **Palette** (Панель элементов), выберите макет **RelativeLayout** и перетащите его либо на панель **Preview** (Предварительный просмотр), либо на пункт **ConstraintLayout** (на рис. 4.8) на панели **Component Tree** (Дерево компонентов) чуть ниже панели **Palette** (Панель элементов). Разрабатывать графический интерфейс нашей игры мы начнем с пустого макета RelativeLayout.

На панели **Palette** (Панель элементов) выберите пункт **Widgets** (Виджеты) и далее **TextView**. Объект `TextView` в интерфейсе для устройств Android похож на метку `JLabel` в наборе инструментов Swing, а *виджетами* мы называем большинство компонентов графического интерфейса в устройстве Android. Перетащим

компоненты графического интерфейса в окне предварительного просмотра конструктора.

Щелкните мышью по виджету `TextView` и перетащите его на белый фон приложения в окне имитации устройства Android, поместив виджет в верхнюю часть. Этот виджет будет заголовком вашего приложения.

После размещения заголовка `TextView` вы увидите, что на панели **Properties** (Свойства) справа отобразились элементы управления настройками. Если панель не отображается, щелкните мышью по пункту **Properties** (Свойства) в правом верхнем углу вкладки **Design** (Конструктор), чтобы ее развернуть. Найдите свойство `text` только что добавленного виджета `TextView`. Измените его значение на текст ***Ваше имя*** **Guessing Game**, а свойству `textAppearance` присвойте значение `Large`. Затем перетащите `TextView` в центр верхней части экрана, как показано на рис. 4.9. Правильно разместить виджет вам поможет серая пунктирная направляющая.

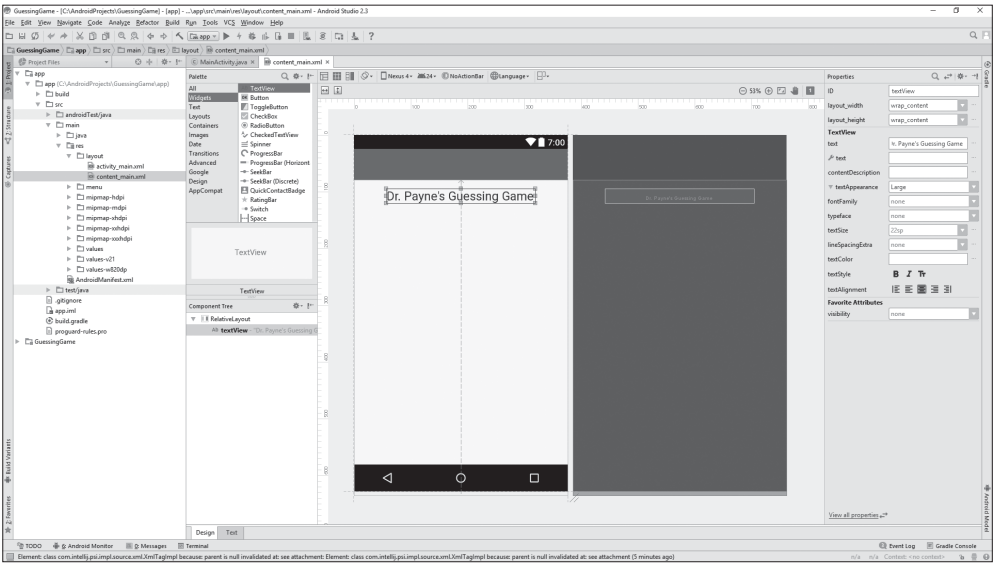


Рис. 4.9. Размещение виджетов/компонентов графического интерфейса пользователя в программе Android Studio очень похоже на то, как это происходит в среде разработки Eclipse

Чтобы добавить метку, в которой пользователь вводит число от 1 до 100, поместите другой виджет `TextView` в окне предварительного просмотра конструктора чуть ниже названия. Присвойте

свойству `textAppearance` значение `Medium` на панели **Properties** (Свойства). Затем введите текст **Enter a number between 1 and 100:** в качестве значения свойства `text` виджета `TextView`.

Затем поместим в программу для устройства Android текстовое поле для ввода хода пользователя. Текстовые поля (в устройстве Android их еще называют `EditText`) расположены прямо под пунктом **Widgets** (Виджеты) на панели **Palette** (Панель элементов). В устройстве Android существует множество типов текстовых полей, и каждый из них ведет себя в зависимости от потребностей приложения. Например, **Plain text** (Обычный текст) является простым текстовым полем, **Password** (Пароль) скрывает символы по мере их ввода, **Phone** (Телефон) показывает цифровую клавиатуру и форматирует ввод в виде номера телефона, а **E-mail** показывает модифицированную клавиатуру с символом @.

Выберите текстовое поле **Number** (Число), чтобы пользователь мог ввести свой ход с помощью цифровой клавиатуры на своем экране. Поместите это поле немного ниже предыдущей метки. Затем поместите виджет **Button** (Кнопка) на такое же расстояние ниже текстового поля и присвойте его свойству `text` значение **`Guess!`**. Наконец, разместите еще один виджет `TextView` под кнопкой, присвоив его свойству `text` значение **Enter a number, then click Guess!**. У вас могут возникнуть проблемы с размещением виджета на таком же расстоянии от кнопки, из-за того, что вы находитесь слишком близко к вертикальному центру экрана. Разместить текст ниже или выше можно с помощью свойства `Layout_Margin` на панели **Properties** (Свойства) (возможно, вам придется щелкнуть мышью по ссылке **View all properties** (Показать все свойства), а затем щелкнуть по серой стрелке рядом с надписью `Layout_Margin` для отображения свойств поля). Найдите пункт `layout_marginTop` и измените его значение на `30dp`. Готовый графический интерфейс должен выглядеть так, как показано на рис. 4.10.

Прежде чем присваивать имена компонентам графического интерфейса пользователя, сделаем одно последнее изменение. Присвойте свойству `width` (ширина) числового текстового поля, куда пользователь вводит свой ход, значение `75dp`. Это можно сделать, щелкнув мышью по ссылке **View all properties** (Показать все свойства) или нажав кнопку со стрелками влево и вправо на панели **Properties** (Свойства). Исходное текстовое поле было слишком широким для игры, в которой используются числа только от 1 до 100, поэтому мы сделали его более узким.

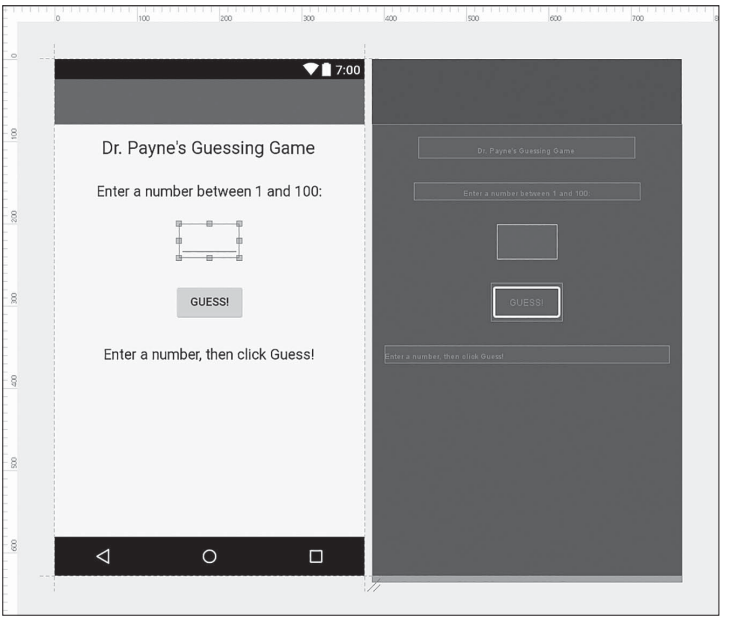


Рис. 4.10. Готовый макет графического интерфейса для нашей игры для Android в режиме конструктора

После этого небольшого изменения настала пора начать присваивать имена нашим компонентам графического интерфейса, чтобы их можно было легко найти в коде на языке Java.

Присваивание имен компонентам графического интерфейса пользователя в среде разработки Android Studio

Точно так же, как мы присваивали имена компонентам графического интерфейса в программе Eclipse, нам нужно будет присвоить имена элементам графического интерфейса в среде разработки Android Studio. Тем самым мы сможем получить доступ к ним из кода. Однако в программе Android Studio имена самих элементов интерфейса не отображаются в исходном коде на языке Java по умолчанию. Вместо этого, прежде чем мы сможем использовать их в коде, нам придется вручную связать эти компоненты графического интерфейса на языке XML с кодом на языке Java.

Однако на данный момент нам нужно просто присвоить свойствам `id` числового текстового поля, кнопки и самой нижней метки значения `txtGuess`, `btnGuess` и `lblOutput` соответственно.

Если появится запрос, следует ли обновить использование (update usages), нажмите кнопку **Yes** (Да). Мы будем использовать эти имена для согласованности и удобства. На рис. 4.11 показаны переименованное текстовое поле, кнопка и метка.

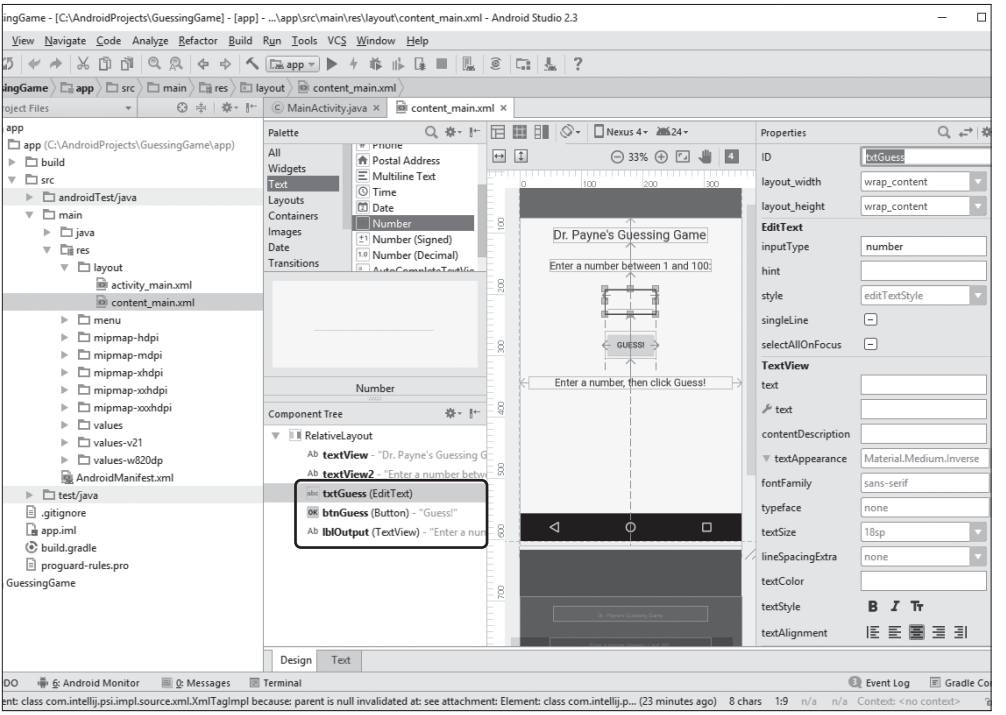


Рис. 4.11. Переименуйте компоненты `EditText`, `Button` и `TextView` (обведены), присвоив их свойству `id` значения `txtGuess`, `btnGuess` и `lblOutput`, соответственно

Последнее, что мы должны сделать, прежде чем начнем программирование на языке Java, — это скрыть значок маленькой *всплывающей кнопки действия* (*floating action button*, *fab*), показанной в правом нижнем углу окна предварительного просмотра конструктора на рис. 4.12. В вашем проекте значок всплывающей кнопки действия может не отображаться, но если это не так, дважды щелкните мышью по файлу *activity_main.xml* на панели **Project Explorer** (Обозреватель проекта) ❶. Перейдите на вкладку *activity_main.xml* над экраном просмотра ❷, а затем выберите значок всплывающей кнопки действия ❸. На панели **Properties** (Свойства) присвойте свойству `visibility` (видимость) значение `invisible` (скрыта) ❹.

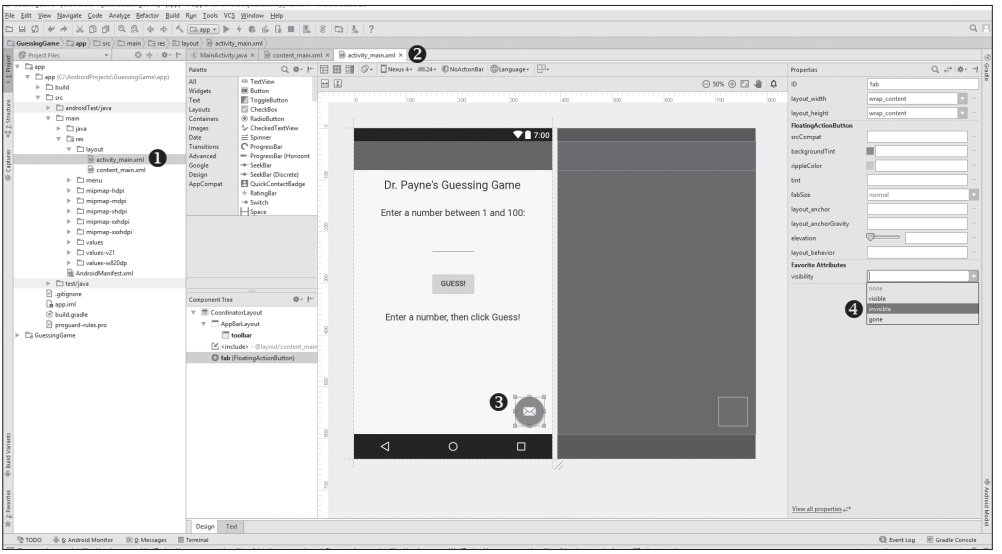


Рис. 4.12. Спрячьте значок всплывающей кнопки действия следующим способом: откройте файл *activity_main.xml*, щелкните мышью по значку в виде маленького конверта и присвойте свойству `visibility` значение `invisible`

Я решил скрыть значок всплывающей кнопки действия, а не удалять ее полностью, поскольку впоследствии, возможно, вы захотите использовать эту кнопку для добавления новых возможностей в приложение. Всплывающая кнопка действия позволяет публиковать информацию из приложения в Facebook и Twitter, отправлять электронные письма друзьям и многое другое.

Затем мы подключим графический интерфейс, чтобы его можно было использовать для программирования оставшейся части приложения.

Соединение графического интерфейса пользователя с программой на языке Java в среде разработки Android Studio

Пришло время подключить графический интерфейс пользователя к программе на языке Java, чтобы мы могли написать код для нашего приложения. Сначала давайте откроем файл исходного кода на языке Java. Для этого на панели **Project Explorer** (Обозреватель проекта) раскройте иерархию ***app*** ⇒ ***src*** ⇒ ***main*** ⇒ ***java*** ⇒ ***com.example.guessinggame*** (или имя вашего пакета) ⇒ ***MainActivity***.

Дважды щелкните мышью по пункту *MainActivity*, чтобы открыть файл *MainActivity.java* с исходным кодом.

Возможно, вы обратили внимание, что в папке *java* на панели **Project Explorer** (Обозреватель проекта) находится еще один пакет или даже два, возможно, названные так же, но со словами (*test*) или (*androidTest*) после имени пакета. Эти пакеты используются для тестирования в более крупных приложениях, где инженеры оценивают качество, безопасность и функциональность приложений при тестировании программного обеспечения. Мы не будем заниматься тестированием, однако это отличный способ проникнуть в отрасль разработки программного обеспечения.

Исходный код на языке Java в файле *MainActivity.java* будет выглядеть примерно так, как показано на рис. 4 .13.

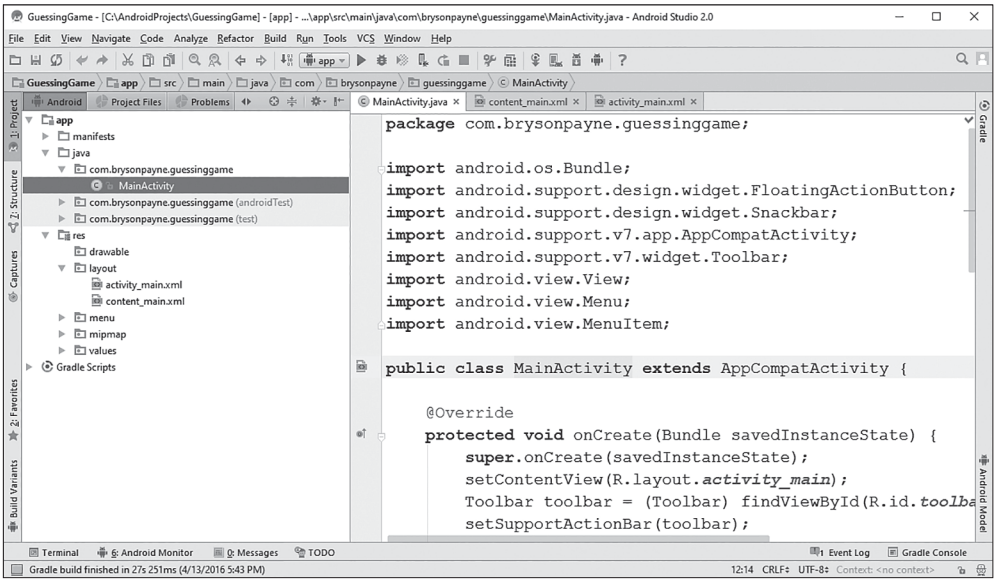


Рис. 4.13. Исходный код на языке Java для приложения *MainActivity.java*, названного так по умолчанию

Обратите внимание, что в среде разработки Android Studio по умолчанию объявление пакета появляется в самой верхней части файла. Пакеты помогают нам организовать все файлы, необходимые для более сложных программ, например мобильных приложений. В данном случае именем пакета является доменное имя компании, которое мы вводили на этапе создания нового проекта, как показано на рис. 4.3, но в обратном порядке — обратите внимание, что *com* идет первым.

За объявлением пакета следует несколько команд `import`. Они работают так же, как и в языке Java для компьютера, импортируя существующие функции и функциональные возможности в код.

Следующий фрагмент кода — `public class MainActivity` — у вас может выглядеть немного иначе, в зависимости от минимального уровня API, который вы выбрали для своего приложения. Однако в целом код будет аналогичным, а приложение, которое мы напишем, будет работать на нескольких уровнях API. Для начала объявим переменные для подключения графического интерфейса к программе. Щелкните мышью внутри скобки, открывающей содержимое класса, и добавьте строки, в которых объявляются переменные для текстового поля, кнопки и метки вывода:

```
public class MainActivity extends AppCompatActivity {
    private EditText txtGuess;
    private Button btnGuess;
    private TextView lblOutput;
```

Когда вы вводите новый тип переменной, может появиться контекстное меню с предложением его импортировать (например, `android.widget.EditText`). Если вы выберете нужный тип виджета для импорта, среда разработки Android Studio автоматически добавит соответствующую команду импорта. Если вы ввели три строки, ни разу не приняв автоматический импорт, щелкните мышью по каждому типу виджетов, а затем нажмите сочетание клавиш **Alt+Enter** (⌥ + ⏎ в операционной системе macOS), чтобы импортировать все отсутствующие классы, как показано на рис. 4.14.

В верхней части вашего файла должны появиться еще три команды `import`:

```
import android.widget.EditText;
import android.widget.Button;
import android.widget.TextView;
```

После объявления трех переменных для виджетов графического интерфейса нам необходимо связать каждую из этих переменных с компонентами языка XML. Мы сделаем это в методе `onCreate()`, показанном в нижней части рис. 4.14. Это функция, которая запускается при загрузке приложения. Среда разработки Android Studio автоматически генерирует код, запускающий этот метод.

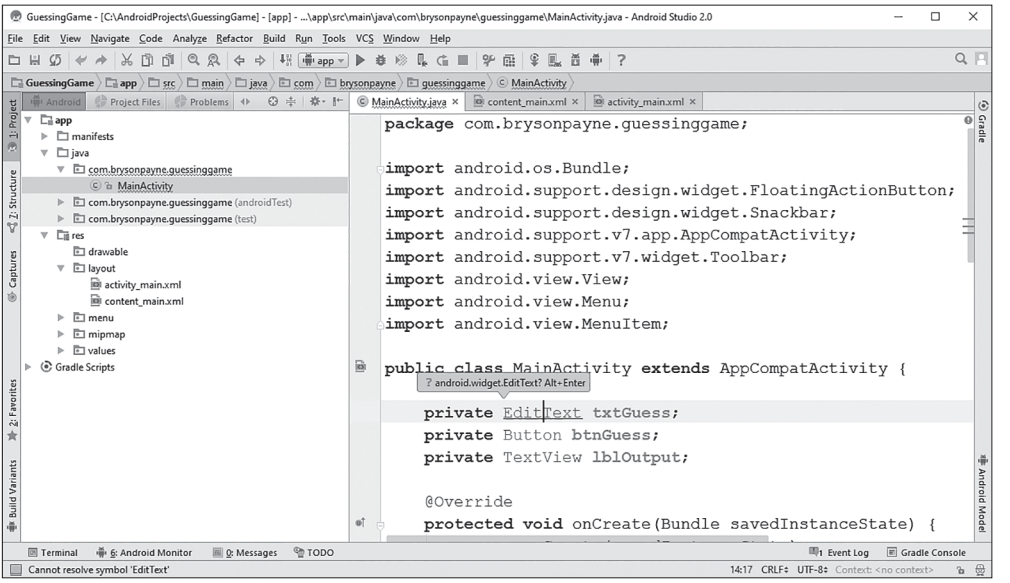


Рис. 4.14. Android Studio позволяет автоматически импортировать классы так же, как и в Eclipse, либо по мере ввода, либо после него с помощью комбинации клавиш **Alt+Enter** (⌥ + ⏎ в macOS)

Просмотрите в начало метода `onCreate()` и найдите следующую строку:

```
setContentView(R.layout.activity_main);
```

Сразу после этой строки нажмите несколько раз клавишу **Enter** и введите следующую неполную строку кода. В этой строке мы соединим переменную `txtGuess` с виджетом `EditText` в файле макета на языке XML:

```
txtGuess = (EditText) findViewById(R.id.
```

Метод `findViewById()` определяет как мы соединим виджеты графического интерфейса в макета на языке XML с переменными, которые будут представлять их в исходном коде. `R` внутри этой функции относится к специальному файлу *R.java*, который генерирует среда разработки Android Studio для подключения ресурсов. `R` — это сокращение от *resources* (ресурсы), обычно хранящихся в папке *res* в проекте. Когда вы начнете вводить эту строку, вы увидите контекстное меню, подобное тому, что показано на рис. 4.15. Найдите в контекстном меню пункт `txtGuess` и дважды щелкните

по нему мышью. Если вы не видите параметр `txtGuess`, вернитесь в режим конструктора файла *content_main.xml* и убедитесь, что вы присвоили свойству `id` текстового поля значение `txtGuess`.

Закончите строку закрывающей скобкой и поставьте точку с запятой, как показано в следующем листинге, а затем настройте кнопку и метку вывода. Полные три строки кода внутри метода `onCreate()` должны выглядеть так:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    txtGuess = (EditText) findViewById(R.id.txtGuess);
    btnGuess = (Button) findViewById(R.id.btnGuess);
    lblOutput = (TextView) findViewById(R.id.lblOutput);
```

Если в макете все указано правильно, эти строки кода должны соединить переменные `txtGuess`, `btnGuess` и `lblOutput` с компонентами графического интерфейса `EditText`, `Button` и `TextView`. Пора бы остановиться и сохранить ваш проект, чтобы зафиксировать достигнутый результат.

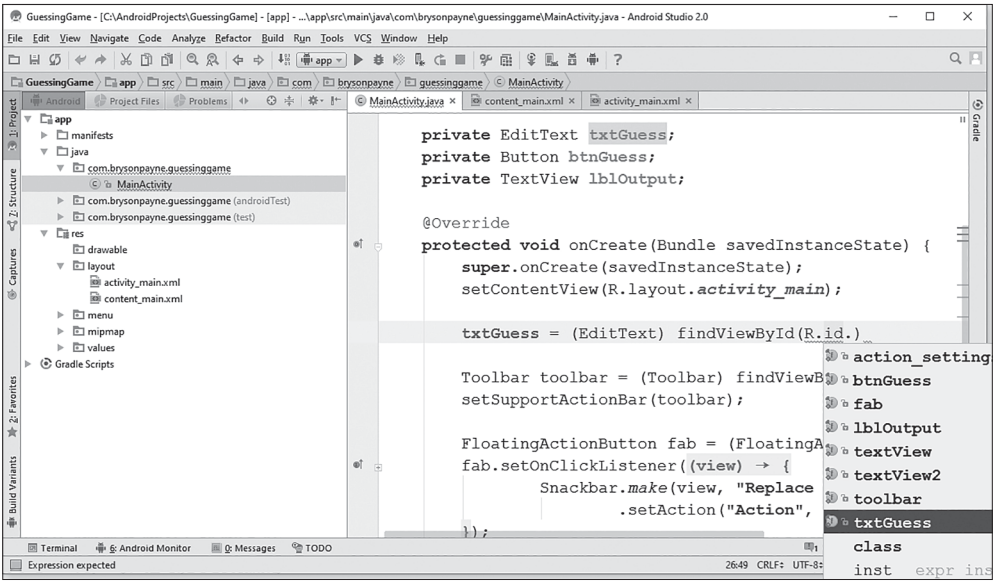


Рис. 4.15. Среда разработки Android Studio помогает нам «подключить» код на языке Java к ресурсам графического интерфейса в макете с помощью контекстного меню

Добавление методов для проверки хода и начала новой игры

Теперь давайте приступим к написанию кода функции `checkGuess()`. Мы начнем писать метод `checkGuess()` чуть ниже объявления переменных `txtGuess`, `btnGuess` и `lblOutput` и чуть выше метода `onCreate()`, как показано на рис. 4.16.

Прежде всего, нам нужно получить ход пользователя из текстового поля графического интерфейса и сохранить его в переменной типа `String`:

```
public class MainActivity extends AppCompatActivity {
    private EditText txtGuess;
    private Button btnGuess;
    private TextView lblOutput;
    public void checkGuess() {
        String guessText = txtGuess.getText().toString();
```

Код для получения хода пользователя из текстового поля выглядит почти так же, как и в приложении для компьютера, за исключением дополнительного метода.`toString()` в конце. В Android есть отдельный класс `Text`, и текст, который вводится в текстовое поле, является объектом именно этого класса, поэтому мы должны преобразовать этот объект в объект типа `String`.

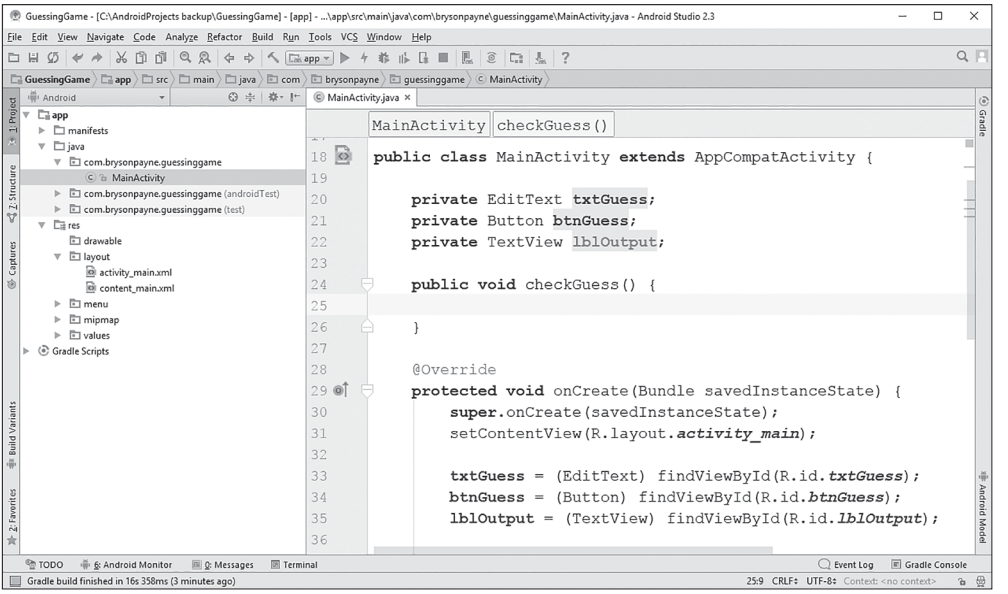


Рис. 4.16. Начинаем писать метод `checkGuess()` на языке Java

К счастью, остальную часть метода `checkGuess()` можно просто скопировать и вставить без изменений из версии графического интерфейса для компьютера, сделанной в программе Eclipse в главе 3. Полный метод `checkGuess()` должен выглядеть следующим образом:

```
    public void checkGuess() {
❶       String guessText = txtGuess.getText().toString();
        String message = "";
❷       try {
❸           int guess = Integer.parseInt(guessText);
            if (guess < theNumber)
                message = guess + " is too low. Try again.";
            else if (guess > theNumber)
                message = guess + " is too high. Try again.";
            else {
                message = guess + " is correct. You win! Let's play again!";
                newGame();
            }
❹       } catch (Exception e) {
            message = "Enter a whole number between 1 and 100.";
❺       } finally {
            lblOutput.setText(message);
            txtGuess.requestFocus();
            txtGuess.selectAll();
        }
    }
```

Приятной особенностью языка Java является то, что мы можем использовать один и тот же код на разных платформах. Так же, как мы использовали код из консольной игры для создания игры с графическим интерфейсом, мы можем использовать код из версии для компьютера в версии для устройства Android. Давайте рассмотрим этот код и обсудим, как он будет работать в приложении для устройства Android.

В пункте ❶ мы получаем ответ пользователя из текстового поля и готовим строку для вывода сообщения о результатах хода. В пункте ❷ мы начинаем блок команд `try-catch-finally` для обработки ошибок ввода пользователя или исключений. Внутри блока `try`, в пункте ❸, мы извлекаем целочисленное значение хода пользователя из введенной им строки символов. Остальная часть блока `try`

содержит команды `if-else`, которые проверяют, была ли попытка пользователя больше или меньше загаданного числа, формируют соответствующее сообщение и запускают новую игру. Далее, инструкция `catch` ❹ предлагает пользователю ввести целое число от 1 до 100. Затем блок `finally` ❺ передает в виджет `lblOutput` соответствующее сообщение, возвращает курсор в текстовое поле и выделяет текст, тем самым подготавливая поле к вводу следующего хода пользователя.

Возможно вы заметили, что пара элементов подчеркнута красным: `theNumber` и `newGame()`. Дело в том, что мы пока не определили их в этой версии нашего приложения. В верхней части класса `MainActivity` под объявлением трех виджетов графического интерфейса добавьте объявление загаданного числа, `theNumber`:

```
public class MainActivity extends AppCompatActivity {
    private EditText txtGuess;
    private Button btnGuess;
    private TextView lblOutput;
    private int theNumber;
```

Это тот же самый код, что мы использовали в версии для компьютера, поскольку код Java для создания целого числа одинаков для консольной, настольной и мобильной версии приложения. Код метода `newGame()` также совпадает с кодом для ПК. Добавьте метод `newGame()` непосредственно перед методом `onCreate()`:

```
    public void newGame() {
        theNumber = (int)(Math.random() * 100 + 1);
    }
@Override
protected void onCreate(Bundle savedInstanceState) {
```

Создание новой игры в приложении для устройства Android происходит точно так же, как и в приложении для компьютера. Мы просто присваиваем переменной `theNumber` случайное целое число от 1 до 100, используя метод `Math.random()`.

Далее необходимо добавить вызов метода `newGame()` внутри метода `onCreate()` после кода, который соединяет три компонента графического интерфейса:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    txtGuess = (EditText) findViewById(R.id.txtGuess);
    btnGuess = (Button) findViewById(R.id.btnGuess);
    lblOutput = (TextView) findViewById(R.id.lblOutput);
    newGame();
```

После запуска игры программа вызовет метод `newGame()` и выберет случайное число которое будет отгадывать игрок. Теперь нам просто нужно обработать событие щелчка по кнопки и узнать, как запустить приложение на нашем собственном устройстве или на эмуляторе Android.

Обработка событий в Android

В среде разработки Eclipse мы смогли добавить слушатель событий к кнопке хода, дважды щелкнув по ней мышью в режиме конструктора. К сожалению, в редакторе Android Studio сделать это не так просто, поскольку в нем макет графического интерфейса отделен от исходного кода. К счастью, среда разработки Android Studio помогает нам добавлять слушателей событий, предоставляя возможности *завершения кода*, аналогичные функции автозавершения в Eclipse.

Чтобы добавить слушатель нажатия кнопки, начните вводить указанную ниже неполную строку кода внутри метода `onCreate()` на следующей строке после вызова метода `newGame()`:

```
    txtGuess = (EditText) findViewById(R.id.txtGuess);
    btnGuess = (Button) findViewById(R.id.btnGuess);
    lblOutput = (TextView) findViewById(R.id.lblOutput);
    newGame();
    btnGuess.setOn
```

Появится всплывающее окно завершения кода среды разработки Android Studio со списком рекомендаций по вводу, как показано на рис. 4.17. Выберите метод `setOnClickListener()` из списка и двойным щелчком добавьте его в свою программу.

Внутри круглых скобок в методе `btnGuess.setOnClickListener()` введите ключевое слово **`new`** и начните вводить **`OnClickListener`**.

Функция завершения кода Android Studio отобразит список параметров, как показано на рис. 4.18.

Выберите `OnClickListener` из списка параметров, и вы увидите, что среда разработки Android Studio добавит несколько дополнительных строчек кода. Ваш код слушателя событий для кнопки `btnGuess` теперь будет выглядеть следующим образом:

```
btnGuess.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
    }
});
```

Выглядит знакомо? Это еще один пример анонимного внутреннего класса. В версии для компьютера также был анонимный внутренний класс, но он имел другое имя. Они оба будут работать практически одинаково. Возможно, вы заметили, что редактор для устройства Android вставил `@Override` в несколько мест. Это так называемая *директива компилятору*. Она сообщает компилятору, что вы реализуете свою собственную версию метода в родительском классе.

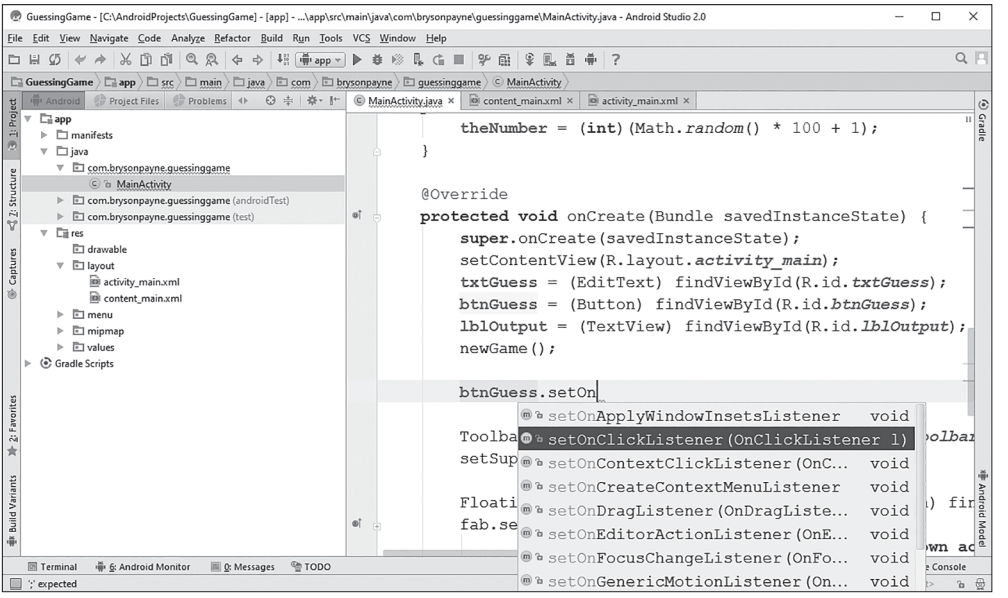


Рис. 4.17. Функция завершения кода в среде разработки Android Studio рекомендует код по мере ввода. Эта функция аналогична автозавершению в программе Eclipse

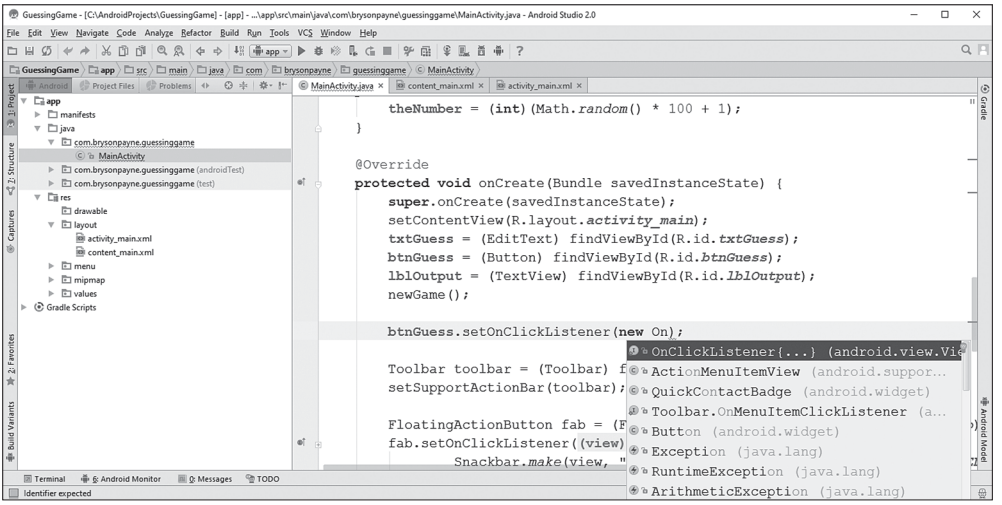


Рис. 4.18. Используйте функцию завершения кода для создания нового компонента `OnClickListener` и его добавления к кнопке `btnGuess`

После того как пользователь нажал кнопку «**Guess!**», программа должна сравнить введенное им число с загаданным. Добавьте строку `checkGuess();` в код внутри фигурных скобок метода `onClick()`.

```
public void onClick(View v) {
    checkGuess();
}
```

Мобильное приложение нашей игры готово для первого запуска. Ниже приведен полный рабочий код первой версии нашей игры. Ваш вариант может немного отличаться. В нем, возможно, будут присутствовать несколько дополнительных методов обработки элементов меню, но, поскольку мы не используем меню, мы удалили их для экономии места (см. главу 5, чтобы узнать, как создавать команды меню и настройки параметров):

```
package com.brysonpayne.guessinggame;
import android.os.Bundle;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.Snackbar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.View;
import android.view.Menu;
```

```
import android.view.MenuItem;
import android.widget.EditText;
import android.widget.Button;
import android.widget.TextView;
import org.w3c.dom.Text;
public class MainActivity extends AppCompatActivity {
    private EditText txtGuess;
    private Button btnGuess;
    private TextView lblOutput;
    private int theNumber;
    public void checkGuess() {
        String guessText = txtGuess.getText().toString();
        String message = "";
        try {
            int guess = Integer.parseInt(guessText);
            if (guess < theNumber)
                message = guess + " is too low. Try again.";
            else if (guess > theNumber)
                message = guess + " is too high. Try again.";
            else {
                message = guess + " is correct. You win! Let's play again!";
                newGame();
            }
        } catch (Exception e) {
            message = "Enter a whole number between 1 and 100.";
            } finally {
                lblOutput.setText(message);
                txtGuess.requestFocus();
                txtGuess.selectAll();
            }
        }
    public void newGame() {
        theNumber = (int)(Math.random() * 100 + 1);
    }
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        txtGuess = (EditText) findViewById(R.id.txtGuess);
        btnGuess = (Button) findViewById(R.id.btnGuess);
        lblOutput = (TextView) findViewById(R.id.lblOutput);
```

```
            newGame();
            btnGuess.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    checkGuess();
                }
            });
            Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
            setSupportActionBar(toolbar);
        }
    }
```

В двух следующих разделах вы узнаете, как запускать приложение на эмуляторе Android и на реальных Android-устройствах.

Запуск приложения в эмуляторе Android

Теперь у нас есть рабочая версия программы на языке Java для запуска приложения, но нам все равно нужно ее протестировать. В отличие от консольной и настольной версий, мобильное приложение не может работать на вашем компьютере самостоятельно, поскольку на ПК нет операционной системы Android. Чтобы запустить приложение для тестирования, вам понадобится устройство с ОС Android или эмулятор, который будет имитировать такое устройство на вашем ПК. В этом разделе мы создадим виртуальное устройство Android для тестирования ваших приложений.

В режиме просмотре файла *MainActivity.java* нажмите кнопку запуска или выберите пункт меню **Run** ⇒ **Run *'app'*** (Выполнить ⇒ Выполнить *'приложение'*). В открывшемся диалоговом окне появится запрос выбора *цели развертывания*, то есть устройства, на котором вы хотите запустить приложение, как показано на рис. 4.19.

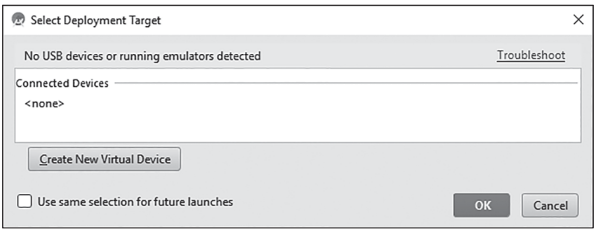


Рис. 4.19. Для запуска приложение для Android вы должны выбрать целевое устройство — им может быть как эмулятор, так и реальное устройство

Нажмите кнопку **Create New Virtual Device** (Создать новое виртуальное устройство) чтобы начать настройку нового *виртуального устройства Android (Android virtual device – AVD)*. Начнем с выбора типа устройства, как показано на рис. 4.20. Затем нажмите кнопку **Next** (Далее).

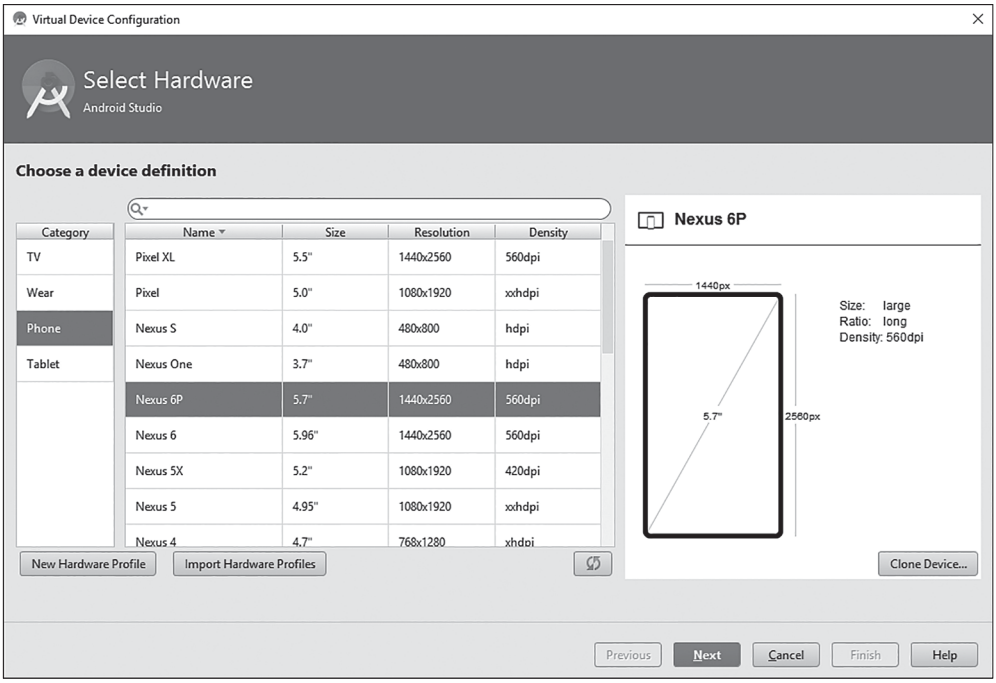


Рис. 4.20. Выбор устройства для эмуляции

Для начала мы выберем смартфон Nexus 6P, но вы можете эмулировать и другие устройства. Если вы решили разработать собственное приложение для Google Play Store, вам нужно протестировать его на разных типах устройств с разными размерами экрана. В этом случае вам придется настроить несколько разных эмуляторов. Но давайте сейчас начнем с малого и выберем только одно устройство.

Затем нам нужно выбрать системный образ, обычно x86 или ARM. Мы будем использовать образ ARM, поэтому перейдите на вкладку **Other Images** (Другие образы), как показано на рис. 4.21.

Если нужная вам версия устройства Android, уровень API или эмулятор отображается серым шрифтом, щелкните мышью по соответствующей ссылке **Download** (Загрузить).

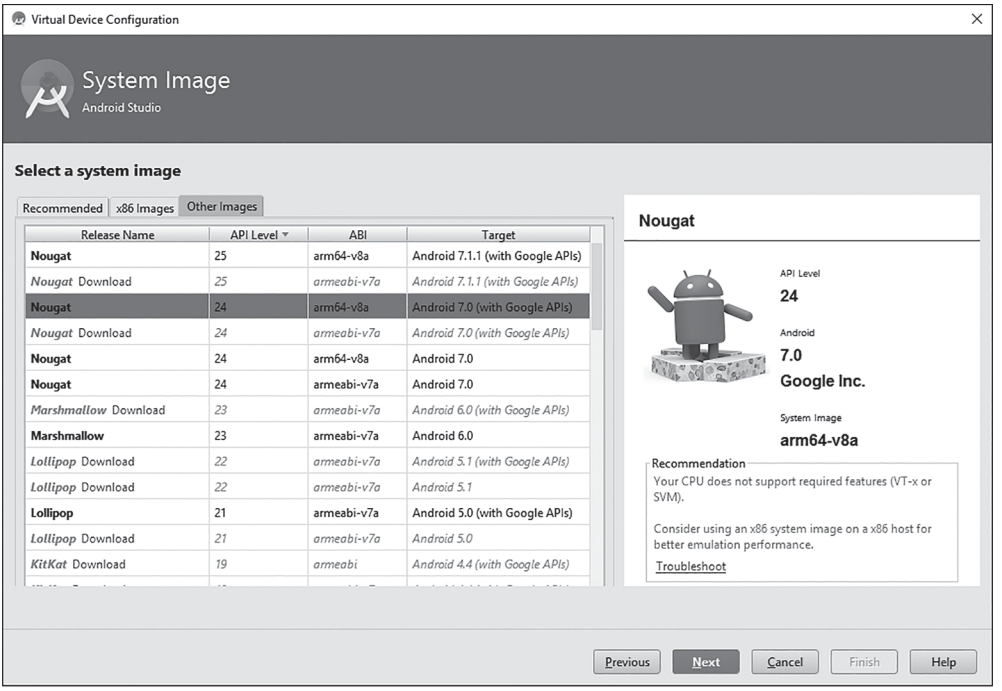


Рис. 4.21. Вы можете выбрать эмулятор x86 или ARM. ARM медленнее, но одинаково работает на разных типах процессоров

Выбрав нужный эмулятор устройства Android, нажмите кнопку **Next** (Далее). Сейчас мы будем использовать версию Nougat API 24 ARM Android 7.0 с API Google. Вы увидите заключительный экран с просьбой подтвердить свою конфигурацию, включая дополнительные изменения настроек, которые вы хотите сделать. Далее мы изменим имя AVD, устройства, которое вы эмулируете, на *My Nexus 6P*, как показано на рис. 4.22.

Если у вас возникли проблемы с запуском вашего эмулятора на старом компьютере или на компьютере с объемом оперативной памяти менее 8 ГБ, нажмите кнопку **Show Advanced Settings** (Показать дополнительные настройки), прокрутите окно вниз до раздела **Memory and Storage** (Память и накопители) и измените значение в строке **RAM** (ОЗУ) на **768** Мб.

Нажмите кнопку **Finish** (Готово). Ваше новое устройство будет создано и сохранено на диске. Теперь пришло время запустить эмулятор и поэкспериментировать с вашим устройством. Вы снова увидите окно **Select Deployment Target** (Выбор цели развертывания), но на этот раз в нем отображается пункт *My Nexus 6P* в качестве доступного эмулятора, как показано на рис. 4.23. Нажмите кнопку **OK**.

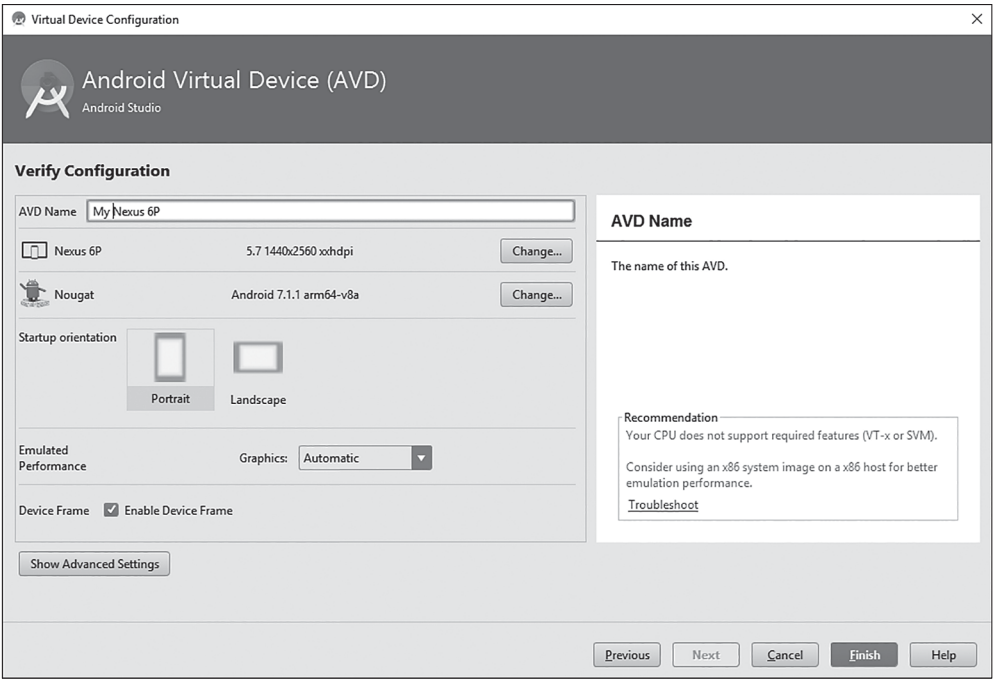


Рис. 4.22. Присвоение имени виртуальному устройству Android

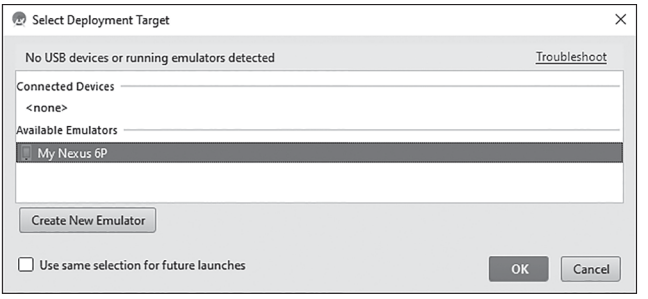


Рис. 4.23. Выберите эмулятор, который вы только что создали, в качестве цели развертывания

Через несколько секунд вы увидите сообщение **Starting AVD** (Запуск AVD), а затем появится окно эмулятора, похожее на экран запуска устройства Android, как показано на рис. 4.24. Первый запуск эмулятора может занять *несколько* минут. Также вы можете увидеть экран блокировки.

Если ваш эмулятор показывает экран блокировки, нажмите значок блокировки в нижней части экрана и перетащите его вверх, чтобы разблокировать виртуальное устройство. Это действие похоже

на смахивание экрана блокировки вверх на реальном смартфоне под управлением операционной системы Android. Возможно, вы увидите один или два приветственных экрана; щелкайте мышью по ним, пока не доберетесь до главного экрана, который показан на рис. 4.24.

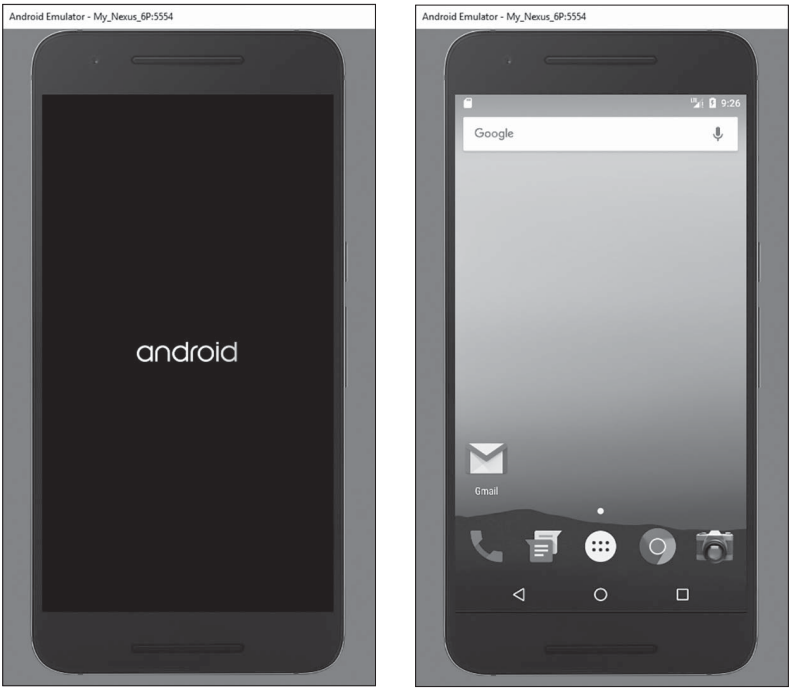


Рис. 4.24. Загрузка эмулятора устройства Android (слева); главный экран виртуального устройства Android (справа)

Теперь вернитесь в среду разработки Android Studio и снова нажмите кнопку запуска. На этот раз вы увидите свой AVD в диалоговом окне **Select Deployment Target** (Выбор цели развертывания) (см. рис. 4.25). Выберите свое устройство и нажмите кнопку **OK**.

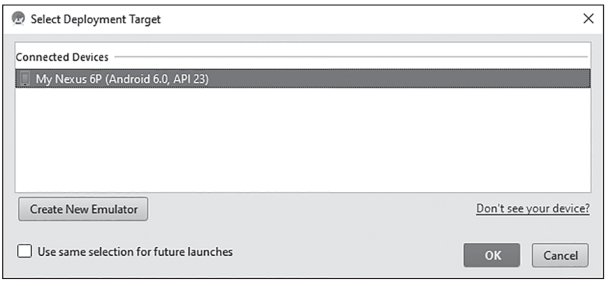


Рис. 4.25. Во время своей работы эмулятор появится в списке подключенных устройств

Проект будет собран еще раз, а затем система перенесет исполняемую версию приложения в эмулятор. (При первом запуске эмулятора может появиться запрос о необходимости обновления Android SDK Tools.) Через несколько минут вы увидите, что ваше приложение работает на эмуляторе.

Мы написали код всего приложения, так что вы можете сразу сыграть в игру на эмуляторе. Посмотрите на рис. 4.26, как может выглядеть раунд игры в приложении.

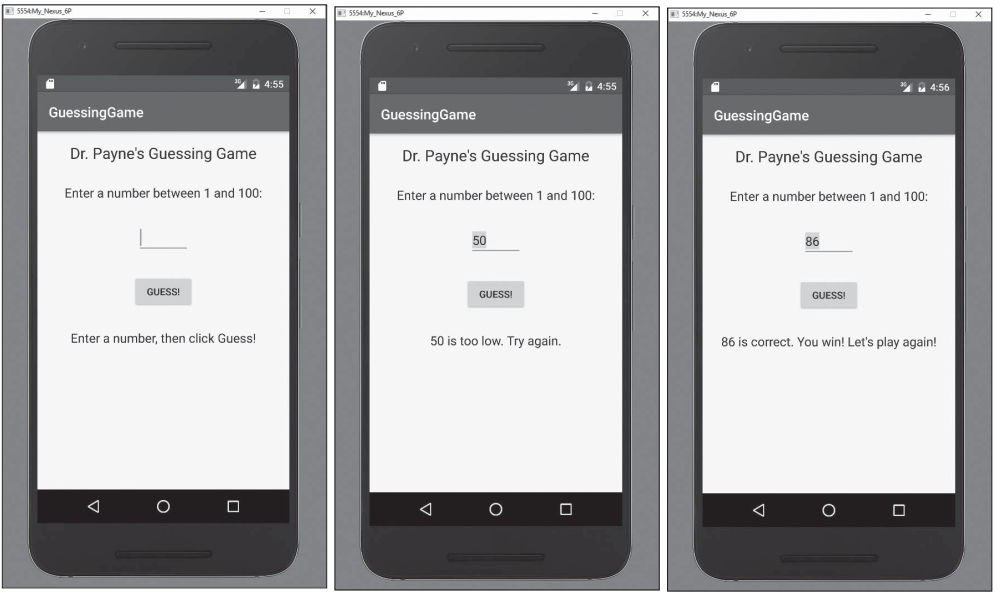


Рис. 4.26. Приложение, работающее на экране эмулятора (слева), после одной попытки (в центре) и после выигрыша (справа)

Все работает точно так же, как и в приложении для компьютера, но на эмуляторе устройства Android. Вводите свой ход с помощью клавиатуры и щелкайте мышью по кнопке «**Guess!**». Мы можем улучшить интерфейс взаимодействия с пользователем, но наше приложение для устройства Android полностью функционально.

Перед тем как мы улучшим UX, рассмотрим процесс запуска приложения на реальном устройстве Android. Но пока оставьте ваш эмулятор устройства Android открытым. Его можно свернуть, если вы его не используете, но оставьте его в фоновом режиме во время программирования, чтобы избежать длительного повторного запуска.

Если у вас нет устройства под управлением операционной системы Android, вы можете перейти к разделу «Улучшение UX-дизайна» далее в этой главе.

Запуск приложения на реальном устройстве Android

Запуск приложения на реальном устройстве Android может потребовать усилий при подготовке и занять некоторое время, однако, при наличии USB-кабеля для подключения устройства к компьютеру, много времени это не займет.

Подготовка устройства

Прежде чем вы сможете развернуть свои собственные приложения на устройстве Android, вам необходимо включить на вашем устройстве режим разработчика. Также потребуется изменить несколько настроек, чтобы можно было разрабатывать и отлаживать приложения.

На устройстве Android коснитесь значка **Settings** (Настройки), а затем прокрутите страницу вниз до конца экрана настроек и коснитесь пункта **About tablet** (О планшете), **About phone** (О телефоне) или **About device** (Об устройстве). В нижней части этого раздела вы увидите строку **Build number** (Номер сборки). Коснитесь ее семь раз подряд, чтобы включить секретный режим разработчика. Режим разработчика позволяет тестировать на устройстве ваши приложения. На рис. 4.27 показаны экраны **Settings** (Настройки) (слева) и **About device** (Об устройстве) (в центре) с разблокированным режимом разработчика.

Включив режим разработчика, коснитесь стрелки назад в левом верхнем углу экрана **About device** (Об устройстве) и выберите пункт **Settings** ⇒ **Developer options** (Настройки ⇒ Параметры разработчика). Убедитесь, что параметры разработчика включены (**On**), как показано на рис. 4.27 (справа). Кроме того, необходимо включить отладку через USB.

Теперь вы готовы подключить смартфон, планшет или другое устройство Android к компьютеру и запустить приложение.

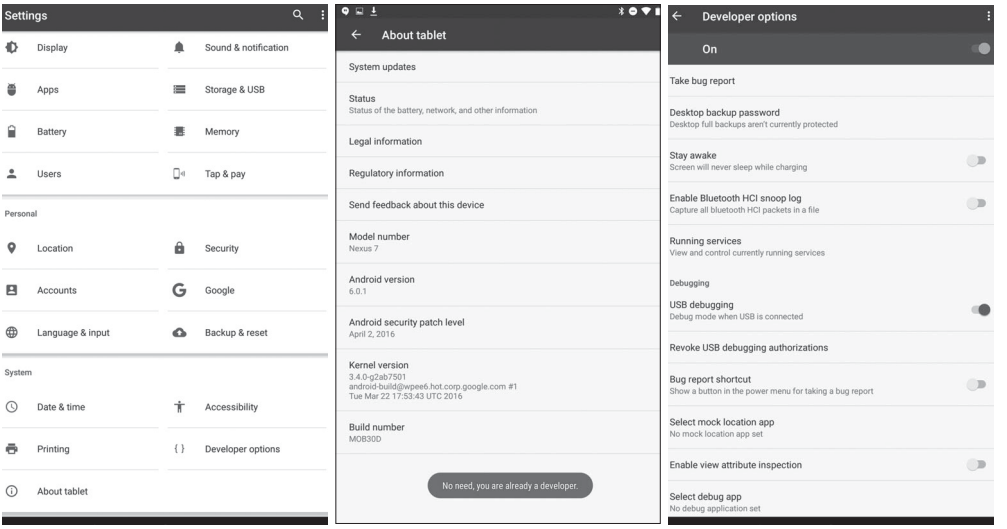


Рис. 4.27. Экраны **Settings** *на планшете Android Nexus 7 (слева),* **About device** *(в центре) и* **Developer options** *(справа)*

Подключение устройства

Чтобы подключить устройство Android к компьютеру, вам понадобится USB-кабель, желательно тот, который прилагался в комплекте со смартфоном или планшетом. Как правило, это кабель с разъемом micro-USB, который часто поставляется в составе зарядного устройства. Обратите внимание, что не все кабели зарядного устройства являются полностью функциональными USB-кабелями — если ваш кабель не будет работать после выполнения следующих шагов, попробуйте другой.

Подключите устройство к компьютеру с помощью USB-кабеля. После подключения на экране смартфона или планшета появится окно, похожее на изображенное на рис. 4.28, с вопросом, хотите ли вы разрешить отладку через интерфейс USB с компьютера, к которому вы только что подключились. Коснитесь кнопки **ОК**. Вы можете установить флажок **Always allow from this computer** (Всегда разрешать с этого компьютера), чтобы при следующем подключении через USB к этому же компьютеру не появлялись уведомления.

Разрешение отладки через интерфейс USB позволяет сделать следующее. Во-первых, мы сможем перенести приложения, которые мы программируем, прямо на устройство Android, передав *файл пакетов Android* (*Android package file – APK*). Во-вторых, это работает намного быстрее, чем запуск приложений на эмуляторе. Кроме того,

это лучший способ проверить, как приложения будут вести себя на реальном устройстве. В-третьих, мы сможем отлаживать приложения с помощью USB-соединения, то есть получать с устройства Android на компьютер информацию, которая поможет нам отладить приложение. Эта информация будет содержать ошибки и другие записи в журнале, которые мы сможем прочитать в консольном приложении (называемом *logcat* в среде разработки Android Studio). Это похоже на вывод консоли, который мы использовали для поиска ошибок в наших приложениях для командной строки и для ПК.

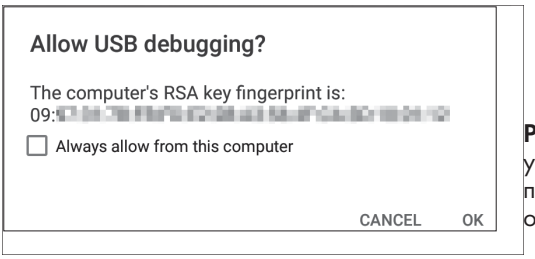


Рис. 4.28. Когда вы впервые подключаете устройство Android к компьютеру, появится запрос, хотите ли вы разрешить отладку через интерфейс USB

Мы подключили наше устройство Android к компьютеру. Давайте посмотрим, как запустить на нем нашу игру.

Запуск приложения на устройстве

Мы готовы попробовать запустить приложение на устройстве Android. В среде разработки Android Studio нажмите кнопку запуска или выберите команду меню **Run** ⇒ **Run *'app'*** (Выполнить ⇒ Выполнить *'приложение'*). На этот раз в окне **Select Deployment Target** (Выбор цели развертывания) будут показаны *два* подключенных устройства на выбор, как показано на рис. 4.29: ваш эмулятор (My Nexus 6P) и ваше реальное устройство Android.

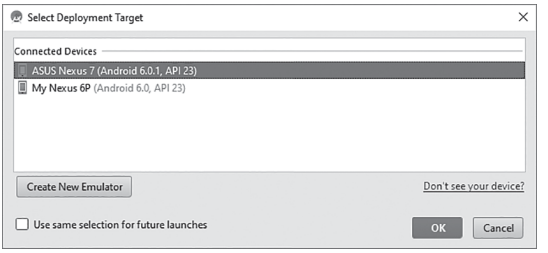


Рис. 4.29. Если вы разблокировали режим разработчика, включили отладку через USB и успешно подключили свое устройство Android, вы можете выбрать его для запуска своего приложения

В этот раз выберите реальное устройство Android и нажмите кнопку **ОК**. Возможно, программе Android Studio потребуется некоторое время, чтобы создать приложение, но как только APK-файл будет перенесен на ваше устройство, вы увидите, как приложение открывается прямо на планшете или на смартфоне. Сыграйте пару раундов и убедитесь, что оно работает так же, как и на эмуляторе (возможно, даже лучше или, по крайней мере, быстрее). Здорово!

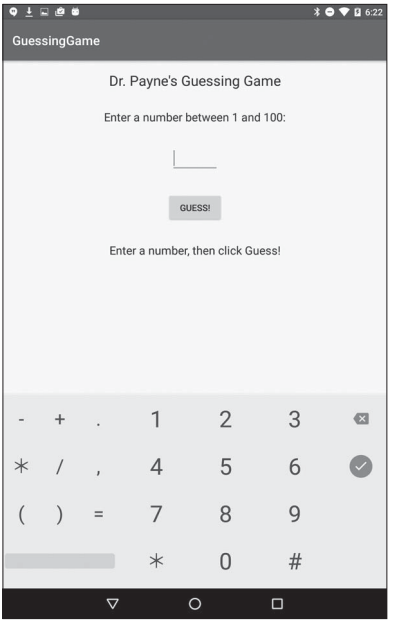


Рис. 4.30. Приложение работает на вашем устройстве Android и по умолчанию отображает цифровую клавиатуру

Возможно, вы сразу обратите внимание на одно существенное отличие: в нижней части экрана отображена цифровая клавиатура, как показано на рис. 4.30.

Когда мы выбирали текстовое поле (`EditText`) для хода пользователя, мы выбрали поле **Number** (Число) для ввода. Именно поэтому и появилась цифровая клавиатура. Но поскольку эмулятор работает на компьютере, у которого есть собственная клавиатура, вы, скорее всего, не видели ее во время предыдущих запусков приложения.

Мы научились запускать приложения на смартфоне, планшете или другом устройстве под управлением операционной системы Android. Теперь давайте сделаем нашу игру удобнее.

Улучшение UX-дизайна

Наше приложение прекрасно работает как на эмуляторе устройства Android, так и на реальных устройствах, однако есть несколько элементов UX-дизайна, которые можно улучшить. Во-первых, отцентрируем ход пользователя в поле ввода. Во-вторых, узнаем, как сделать так, чтобы вместо касания кнопки «**Guess!**» можно было нажимать клавишу **Enter**. Это сделает приложение интуитивно более понятным для пользователя.

Выравнивание ответа пользователя в текстовом поле

В среде разработки Android Studio снова откройте файл макета *content_main.xml*. Выберите текстовое поле `txtGuess`, щелкнув мышью по нему в окне предварительного просмотра в режиме конструктора. На панели **Properties** (Свойства) найдите пункт `textAlignment`. Свойство `textAlignment` в устройстве Android аналогично свойству `horizontalAlignment` в графическом интерфейсе компьютерного приложения — оно позволяет изменять выравнивание текста, который вводит пользователь.

Щелкните мышью по значку в виде выровненного по центру текста. Если вы хотите увидеть результат во время игры, не запуская приложение, прокрутите вниз до свойства `text` и введите значение **50**. Число должно отображаться в центре текстового поля в области предварительного просмотра. Не забудьте удалить тестовое значение, если вы не хотите, чтобы оно отображалось в запущенном приложении.

Добавление слушателя для клавиши Enter

Давайте настроим слушатель событий для обработки нажатия клавиши **Enter** так же, как мы обработали щелчок мышью по кнопке «**Guess!**», то есть добавим вызов метода `checkGuess()`.

Вернитесь к файлу *MainActivity.java*, прокрутите его содержимое вниз до метода `onCreate()` и найдите позицию, где мы добавили метод `onClickListener` к кнопке `btnGuess`. (Возможно, вам нужно будет *развернуть* код слушателя событий, щелкнув мышью по символу + слева от пункта `btnGuess.setOnClickListener()`, чтобы код выглядел так же, как здесь.) Ниже добавим слушатель событий к текстовому полю `txtGuess` для обработки таких событий, как нажатие клавиши **Enter**. Следующий код добавляет слушатель к текстовому полю `txtGuess`:

```
btnGuess.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        checkGuess();
    }
});
txtGuess.setOnEditorActionListener(new TextView.OnEditorActionListener() {
    @Override
```

```
        public boolean onEditorAction(TextView v, int actionId, KeyEvent event) {
            checkGuess();
❶           return true;
        }
    });
```

Мы слушаем *действие поля ввода*, состоящее в нажатии пользователем клавиши **Enter** во время ввода данных в текстовом поле. После того как это событие произойдет, мы будем проверять ход пользователя.

Мы добавили инструкцию `return`, чтобы возвращать значение `true` (строка ❶), поскольку мы бы хотели, чтобы на экране отображалась клавиатура, с помощью которой пользователь мог бы ввести свой следующий ход. Инструкция `return` сообщает текстовому полю, завершил ли код обработчика событий свою работу. Возвращая значение `true`, мы сообщаем устройству Android, что сделали все необходимое для проверки хода пользователя. Если вместо этого мы вернем здесь значение `false`, Android-устройство завершит обработку событий клавиши **Enter** и удалит цифровую клавиатуру с экрана — как это происходит по умолчанию, когда вы вводите текст в форме на веб-странице. Мы не хотим, чтобы клавиатура исчезала после каждого хода, поэтому мы возвращаем значение `true`.

Завершающий штрих

Запустите приложение несколько раз, и вы увидите, что клавиша **Enter** позволяет быстро и эффективно вводить и проверять ходы. Вы также увидите, что введенные вами числа хорошо отцентрированы в текстовом поле. Однако когда вы выиграете, вы можете столкнуться с ошибками выравнивания, как показано на рис. 4.31. Эти ошибки зависят от версии API, размера и плотности пикселей экрана вашего устройства.

Проблема в том, что объект `TextView` с именем `lblOutput` изменяет свой размер, чтобы показать более длинную строку «You win! Let's play again!», а некоторые старые API-интерфейсы могут не справиться с выравниванием ее по центру. Этого можно избежать, растянув метку `lblOutput` в режиме конструктора на всю ширину экрана и присвоив свойству `textAlignment` значение `center`. Теперь у вас есть полностью готовое игровое приложение, в которое можно играть на вашем устройстве и даже поделиться

с друзьями! На рис. 4.32 показана финальная версия 1.0 приложения, запущенная на смартфоне Nexus 7 (с крупным шрифтом).

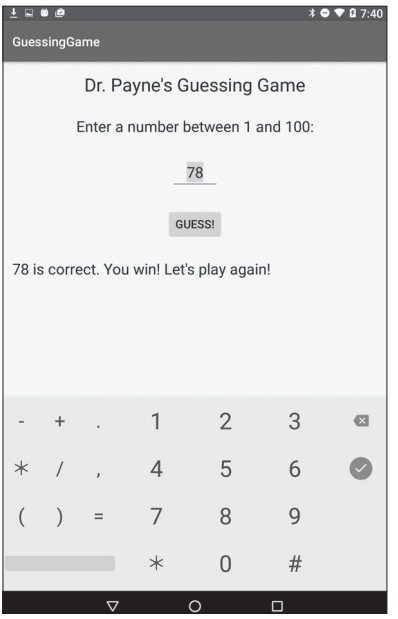


Рис. 4.31. Если возникла проблема выравнивания, нужно растянуть метку `lblOutput` и присвоить свойству `textAlignment` значение center

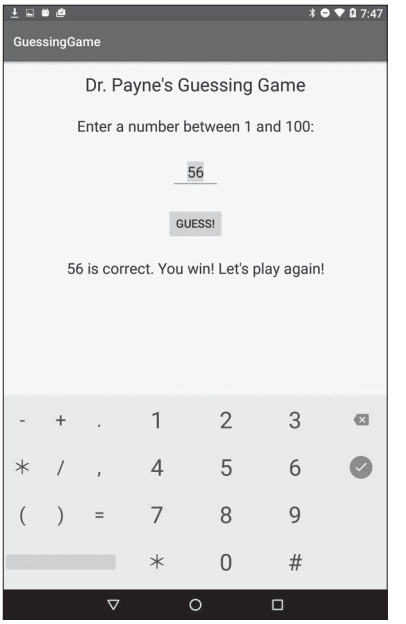


Рис. 4.32. Финальная версия 1.0 мобильного приложения со всеми улучшениями интерфейса

Ура! Вы прошли путь от простой текстовой программы, запускаемой из оболочки командной строки, к игре с графическим интерфейсом для ПК и, наконец, к полнофункциональному мобильному приложению, работающему на устройствах Android. Вы начинаете чувствовать мощь и гибкость языка программирования Java? В главе 5 мы сделаем приложение еще более профессиональным, добавив меню настроек и возможность сохранять настройки пользователя!

Что вы изучили

Вы увидели, что значительная часть кода на языке Java для запуска из оболочки командной строки или с использованием графического интерфейса на компьютере может создать основу современного мобильного приложения. Это происходит благодаря возможности языка Java работать на нескольких платформах. Кроме того, в этой главе вы получили навыки работы с мобильными устройствами, а именно научились следующему:

- создавать проекты новых приложений в среде разработки Android Studio;
- создавать графический интерфейс в режиме конструктора в среде разработки Android Studio, в том числе изменять свойства различных элементов с помощью панели **Properties** (Свойства);
- присваивать имена компонентам графического интерфейса в макете для удобства дальнейшего использования в программе на языке Java;
- соединять элементы макета графического интерфейса в устройстве Android с кодом на языке Java. Добавлять собственные методы, такие как `checkGuess()` и `newGame()`, в приложение для устройства Android;
- использовать в приложениях для устройств под управлением ОС Android код на языке Java, написанный для компьютера;
- обрабатывать события в устройстве Android, включая щелчки по кнопкам и работу с клавиатурой и полем ввода, например, нажатие клавиши **Enter**;
- тестировать приложения, запуская их на виртуальном устройстве Android с использованием эмулятора устройства Android;
- запускать приложения на смартфоне, планшете или другом устройстве под управлением операционной системы Android, включив режим разработчика и выполняя отладку через интерфейс USB;
- настраивать интерфейс путем изменения свойств виджета и добавления удобных для пользователя завершающих штрихов.

Дополнительные задачи

Попробуйте выполнить следующие задания. Таким образом, вы повторите пройденный материал, попрактикуетесь в программировании и получите новые навыки. Если у вас возникнут проблемы, вы можете загрузить примеры решений на сайте по адресу **https://eksmo.ru/files/learn_java_easy_way.zip**.

Задача № 1. Всплывающее уведомление о количестве ходов

В дополнительных заданиях в главе 3 вы меняли сообщение о выигрыше так, чтобы сообщать пользователю, сколько попыток он сделал:

```
62 is correct! You win after 7 tries!
```

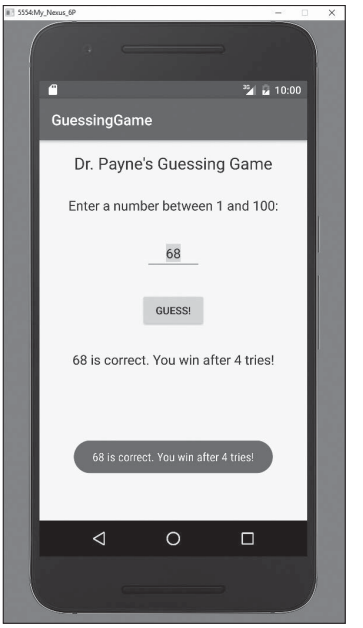


Рис. 4.33. Всплывающие уведомления — это удобный способ показать быстрое сообщение

Давайте реализуем нечто подобное в мобильной версии с помощью, так *всплывающих уведомлений* (Toast). Toast — это Android-виджет для создания всплывающих окон сообщений. Всплывающие уведомления — это удобный способ показать быстрое сообщение, например, об ошибке или о том, что пользователь выиграл игру. См. всплывающее уведомление в нижней части экрана на рис. 4.33.

Вы можете создать всплывающее уведомление с помощью метода `Toast.makeText()`. Приведем пример, в котором добавим всплывающее уведомление в блок кода с инструкцией `else`, который выполняется при выигрыше:

```
else {
    message = guess + " is correct. You win after " + numberOfTries + " tries!";
    Toast.makeText(MainActivity.this, message, Toast.LENGTH_LONG).show();
    newGame();
}
```

Благодаря методу `Toast.LENGTH_LONG` всплывающие уведомления в этом коде будут появляться в `MainActivity` (наша игра), отображая `String message` (строковое сообщение) в течение нескольких секунд. Есть также метод `Toast.LENGTH_SHORT`, но при его использовании всплывающие сообщения появляются и исчезают так быстро, что их трудно прочитать. Метод.`show()` в конце строки показывает всплывающее уведомление на экране.

Как и в версии с графическим интерфейсом для компьютера, для выполнения этой задачи требуется создать новую переменную в верхней части вашего класса (например, `int numberOfTries = 0;`), увеличивать количество использованных попыток при каждом успешном запуске метода `checkGuess()` и изменить `message` (сообщение), показываемое при выигрыше игрока, в котором надо будет отобразить количество попыток, как во всплывающем окне, так и в поле `lblOutput`. После подсчета количества попыток добавьте функцию (которую вы уже делали в дополнительных заданиях в главе 3), сообщающую игроку об его поражении в случае превышения заданного лимита попыток. Предоставьте пользователю семь попыток и сообщайте ему, сколько их еще осталось после каждого хода.

Задача № 2. Делаем красиво

Исследуйте некоторые эстетические характеристики компонентов графического интерфейса пользователя в среде разработки Android Studio, включая цвета фона и шрифта, различные шрифты, `textSize` и много другое. Переместите компоненты в вашем приложении, чтобы сделать интерфейс более красивым. Например, можно настроить игру, как показано на рис. 4.34.

Можно даже добавить пользовательское фоновое изображение (добавьте файл с изображением в папку *app\res\drawable*, а затем выберите его в качестве значения свойства `background` (фон) вашего макета, кнопки или другого компонента). Поиграйте с различными

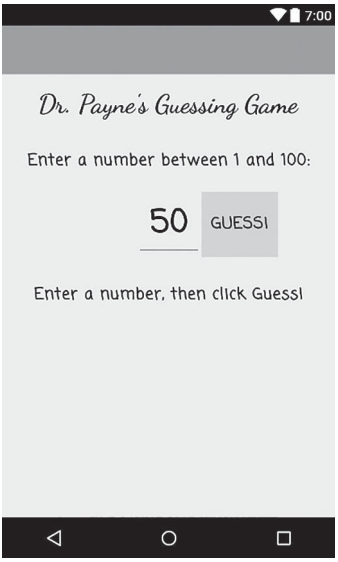


Рис. 4.34. Изменение цветов, шрифтов, размера текста и расположения компонентов макета в приложении может сделать его необычным

настройками — вы всегда можете их отменить, нажав сочетание клавиш **Ctrl+Z** (или **⌘+Z** в macOS).

Задача № 3 Создание мобильного приложения игры MadLib

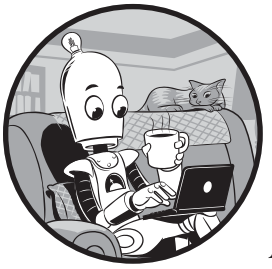
Вспомните программу игры в чепуху *MadLibGUI.java*, которую вы создали в главе 3 (Дополнительное задание № 3). Теперь создайте мобильную версию вашего приложения «Чепуха», в котором пользователю предлагается ввести несколько слов в графическом интерфейсе с метками и текстовыми полями, такими как `txtBigAnimal`, `txtPastTenseVerb`, `txtSmallContainer` и `txtFood`. Добавьте кнопку, которую пользователь сможет нажать, чтобы создать собственную историю в стиле чепухи. Красивым штрихом может быть включение какого-либо стандартного или начального текста в каждом текстовом поле, чтобы дать пользователю представление о том, как работает программа.

Когда пользователь нажимает кнопку, программа должна отображать законченную историю в виджете `TextView` или `EditText` — `EditText` позволит скопировать забавную историю пользователя и вставить куда-нибудь, если захочется поделиться ею. Поэкспериментируйте с цветами, шрифтами и компоновкой макета, пока не удовлетворитесь результатом, а затем поделитесь своим творением с друзьями!

§ Чтобы поля `TextView` или `EditText` показывали несколько строк, используйте символы `\n`, чтобы вставить новую строку в сообщение типа `String`, которое вы выводите с помощью метода `setText()`, например: `"Когда-то… \n принцесса-буйвол, \n которая жила в банке с супом."` Эта цитата будет занимать три строки, каждая из которых будет начинаться после символов `\n`.

Глава 5

УЛУЧШЕНИЕ ПРИЛОЖЕНИЯ ПУТЕМ ДОБАВЛЕНИЯ МЕНЮ И ВОЗМОЖНОСТЕЙ НАСТРОЙКИ



Вы написали забавное приложение для устройства Android, но в нем все еще не хватает нескольких вещей. Вы не узнали, как создать меню для Android-приложения или как сохранять лучшие результаты, статистику игры и другую информацию. В этой главе мы добавим в нашу игру «Больше-Меньше» меню настроек и возможность сохранять информацию.

Добавление меню настроек в Android-приложение

В большинстве приложений и игр доступны настройки, до которых пользователь может добраться с помощью меню. В игре «Больше-меньше» можно позволить пользователю изменить уровень сложности игры, начать заново, посмотреть статистику или

вывести экран «О программе». Для этих целей мы создадим меню, с помощью которого можно будет выполнять все эти действия.

Для добавления меню настроек в Android-приложение необходимо сделать следующие четыре шага.

1. Отредактировать созданный по умолчанию XML-файл меню и создать элементы, которые сможет выбрать пользователь.
2. Изменить файл исходного кода приложения и отобразить меню и действия, которые были созданы на предыдущем шаге.
3. Создать обработчик событий, реагирующий на выбор пользователя.
4. Написать код, который будет выполняться при выборе той или иной команды меню.

Добавление меню настроек не только сделает наше приложение более профессиональным, но и позволит пользователю лучше контролировать игровой процесс. Мои сыновья любили изменять диапазон загаданного числа от 1 до 10, затем от 1 до 100, затем от 1 до 1000. Однако когда мы добавили пункт «Статистика игры», которая отображала количество выигранных игр, я не мог отобрать у них мое устройство — они хотели, чтобы это значение становилось все больше и больше! Надеюсь, вы найдете эти дополнительные возможности такими же приятными (а, возможно, и такими же захватывающими), как и они.

Добавление элементов меню в XML-файл

Откройте ваш проект игры «Больше-Меньше» в среде разработки Android Studio и на панели **Project Explorer** (Обозреватель проекта) измените представление в левом верхнем углу на **Android**. Затем откройте файл меню, созданный по умолчанию, раскрыв пункт **app** ⇒ **res** ⇒ **menu** и дважды щелкнув мышью по файлу *menu_main.xml*.

Измените код на языке XML в файле *menu_main.xml*, чтобы он соответствовал следующему листингу:

```
<menu xmlns: android=»http://schemas.android. com/apk/res/android»>
    <item
        android: id=»@+id/action_settings»
        android: title="Settings" />
```

```
    <item
        android: id="@+id/action_newgame"
        android: title="New Game" />
    <item
        android: id="@+id/action_gamestats"
        android: title="Game Stats" />
    <item
        android: id="@+id/action_about"
        android: title="About" />
</menu>
```

Тег <menu> создает ресурс меню, используя пространство имен языка XML для документов Android XML, идентифицируемый унифицированным идентификатором ресурса (uniform resource identifier — URI)*. Мы можем использовать язык XML для хранения и отображения чего угодно, от веб-страниц до баз данных, добавляя XML-теги к этим элементам. Атрибут xmlns (пространство имен языка XML) в этом коде выбирает основное пространство имен устройства Android, так что теги в этом файле XML будут ссылаться на общие элементы в приложении для устройства Android. Таким образом, тег <menu> относится к меню устройства Android, а каждый тег <item> описывает элемент или запись в этом меню с атрибутами. В этом меню будет четыре пункта: **Settings** (Настройки), **New Game** (Новая игра), **Game Stats** (Статистика игры) и **About** (О программе), поэтому мы добавляем четыре тега <item>. Мы присваиваем эти имена атрибуту title каждого элемента. Этот атрибут определяет текст, который видит пользователь для каждого пункта, когда открывает меню. Мы будем использовать атрибут id позже в нашем коде, чтобы определить, какой пункт выбрал пользователь.

Сохраните файл *menu_main.xml*. Теперь давайте отобразим наше меню настроек в игре «Больше-Меньше».

Отображение меню настроек

Мы настроили меню, но для его отображения нужно добавить код на языке Java в файл *MainActivity.java* нашего приложения. На панели **Project Explorer** (Обозреватель проекта) найдите

* **schemas.android.com/apk/res/android**

и откройте файл *MainActivity.java*, расположенный в папке **app ⇒ java ⇒ com.*вашдомен*.GuessingGame**.

В нижней части класса `MainActivity` найдите метод с именем `onCreateOptionsMenu()`. Измените его, как показано ниже. (Если в вашей программе нет метода `onCreateOptionsMenu()`, добавьте следующий код после закрывающей фигурной скобки метода `onCreate()`, но перед последней закрывающей скобкой файла `MainActivity`.)

```
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.menu_main, menu);
        return true;
    }
} // Последняя закрывающая скобка файла MainActivity.java
```

Метод `onCreateOptionsMenu()` делает ровно то, что подразумевает его название — он сообщает устройству Android, что делать при создании меню настроек для нашего приложения. В данном случае мы говорим устройству Android, чтобы файл *menu_main.xml* служил в качестве меню настроек. Файл *menu_main.xml* еще не является меню, поэтому его нужно преобразовать в меню с помощью класса, называемого `MenuInflator`. Мы создадим экземпляр класса `MenuInflator`, используя метод `getMenuInflator()`, и назовем его `inflater`. Как только у нас появился объект `inflater`, мы можем вызвать для него метод `inflate()` и передать ему файл формата XML (`R.menu.menu_main`) и меню, в которое должны войти элементы файла XML (меню). После того как вы добавите этот код в свой файл, вам, возможно, потребуется нажать сочетание клавиш **Alt+Enter** (⌥ + ⏎ в операционной системе macOS), чтобы добавить отсутствующие инструкции импорта.

После внесения этих изменений сохраните и запустите приложение. Устройство Android сообщит о существовании меню настроек, отобразив три точки в панели действий вашего приложения (см. рис. 5.1, вверху). Щелкнув мышью по этим точкам, вы увидите меню настроек (см. рис. 5.1, внизу).

Вы, естественно, заметите, что выбор пунктов меню не приводит к каким-либо последствиям. Дело в том, что мы не добавили код для реакции на выбор пользователя. Мы сделаем это позже.

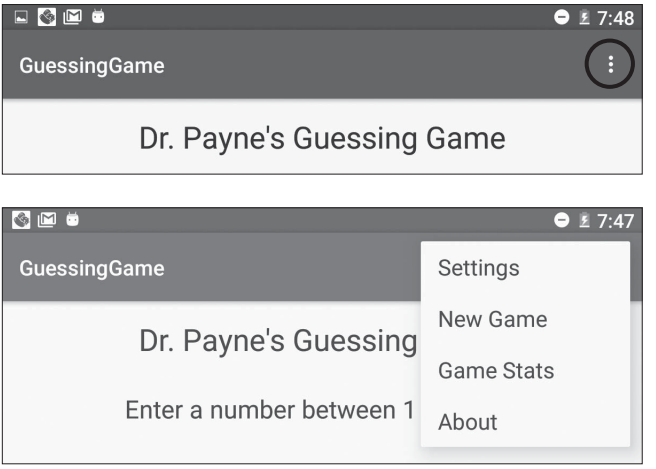


Рис. 5.1. Меню настроек отображается в виде трех точек в панели действий приложения (вверху). Нажатие развернет меню настроек (внизу)

Реакция на выбор пользователя

Когда пользователь выбирает команду в меню, мы хотим, чтобы наше приложение выполняло запрошенное действие. Чтобы оно могло это сделать, нам нужно добавить обработчик событий, который отследит, какая именно команда была выбрана. Мы будем использовать атрибут `id`, связанный с каждым элементом.

В файле *MainActivity.java* найдите и измените метод обработчика событий `onOptionsItemSelected()`. В качестве альтернативы вы можете добавить его прямо под методом `onCreateOptionsMenu()`, который мы модифицировали в предыдущем разделе, но до закрывающей фигурной скобки в последней строке файла.

```
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.menu_main, menu);
    return true;
}
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_settings:
            return true;
        case R.id.action_newgame:
```

```
            newGame();
            return true;
        case R.id.action_gamestats:
            return true;
        case R.id.action_about:
            return true;
        default:
            return super.onOptionsItemSelected(item);
        }
    }
}
```

В этом коде мы используем инструкцию `switch`, чтобы определить, какую из команд в меню выбрал пользователь. Инструкция `switch` — это еще один способ проверки нескольких условий, аналогичный длинной цепочке инструкций `if-else`. Вместо того, чтобы связывать вместе четыре инструкции `if-else` для проверки каждого возможного выбора пункта меню, мы можем использовать одну инструкцию `switch`. Переменную, которую мы проверяем, следует заключить в круглые скобки после ключевого слова `switch`. В данном примере мы проверяем атрибут `id` выбранного пункта меню, поэтому мы напишем код следующим образом: `switch (item.getItemId())`. Затем внутри фигурных скобок блока `switch` мы перечислим значения, которые хотим проверить, каждое как аргумент инструкции `case` (например, `case R.id.action_settings`). После каждого аргумента инструкции `case` будет указано двоеточие (`:`), а затем код, который должен выполняться в случае именно этого выбора, а также инструкция `break` или `return`. Этот обработчик событий возвращает логическое значение, поэтому в каждом из блоков `case` мы использовали инструкции `return`, а не `break`. Если бы здесь не было инструкций `return`, нам нужно было бы использовать команду `break` в конце каждого блока.

В данном фрагменте каждая инструкция `case` проверяет одно из значений атрибута `id` элементов, которые мы ввели в файле *menu_main.xml*. В зависимости от выбора пользователя будет выполняться тот или иной код. В настоящий момент у нас есть код только для случая `action_newgame`, который запускает новую игру с помощью метода `newGame()`. Другие случаи требуют написания чуть большего количества кода, поэтому мы разберем их по очереди.

Создание всплывающего окна «О программе»

В случае выбора в меню команды **About** появится окно «О программе». Вы, скорее всего, видели такие окна в других приложениях. Мы будем использовать *окно оповещений* — всплывающее окно, используемое для информирования пользователя или взаимодействия с ним. Оно настраивается лучше, чем всплывающие уведомления, которые мы использовали в главе 4 (Задача программирования № 1), потому что класс `AlertDialog` позволяет нам настраивать свойства нашего окна с помощью подкласса `Builder`. В данном случае мы будем использовать окно оповещения, чтобы отреагировать на выбор пользователем в меню пункта **About**, сообщив ему, кто создал потрясающую игру «Больше-Меньше», в которую он играет. Добавьте следующий код в инструкцию `case` элемента `action_about`:

```
    case R.id.action_about:
❶       AlertDialog aboutDialog = new AlertDialog.Builder(MainActivity.this).create();
❷       aboutDialog.setTitle("About Guessing Game");
❸       aboutDialog.setMessage("(c)2018 Ваше имя.");
❹       aboutDialog.setButton(AlertDialog.BUTTON_NEUTRAL, "OK",
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int which) {
❺                       dialog.dismiss();
                    }
                });
❻       aboutDialog.show();
        return true;
```

Мы использовали класс `AlertDialog.Builder` в строке ❶ для создания настраиваемого всплывающего окна. Код в пункте ❷ определяет текст заголовка всплывающего окна как **About Guessing Game**, а строка ❸ выводит простое сообщение с информацией об авторских правах и вашим именем (вы, естественно, можете написать здесь любой текст, какой захотите). Метод `setButton()` ❹ добавляет во всплывающее окно кнопку с текстом **OK**, а последующий слушатель событий `onClick()` закрывает всплывающее окно после того, как пользователь нажмет кнопку **OK**, при помощи вызова метода `dismiss()` ❺. Наконец, всплывающее окно будет показано с помощью команды `show()` ❻.

Нажмите сочетание клавиш **Alt+Enter** (⌥ + ⏎ в операционной системе macOS) для импорта класса `AlertDialog`. Затем сохраните обновленный код и запустите новую версию приложения. Если вы откроете меню настроек и выберете пункт **About**, вы должны увидеть всплывающее окно, подобное показанному на рис. 5.2.

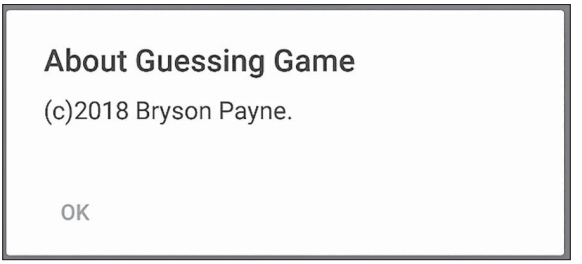


Рис. 5.2. Настроенное пользователем всплывающее окно оповещений

Игра «Больше-Меньше» становится все больше похожа на профессиональное приложение для устройства под управлением операционной системы Android. Давайте сделаем ее еще лучше, позволим пользователю выбирать уровень сложности игры и сосчитаем количество выигранных им игр.

Изменение диапазона загаданного числа

Существенным улучшением игры будет добавление возможности выбора диапазона, в котором находится загаданное компьютером число Загаданное число может лежать, например, в диапазоне от 1 до 10, от 1 до 100 или от 1 до 1000. Разобравшись с меню настроек и окнами оповещений, давайте составим план, что мы должны сделать, чтобы пользователь мог изменить величину диапазона.

Во-первых, нам нужно добавить переменную для хранения значения верхней границы диапазона, которое будет использоваться вместо постоянного значения `100`. Во-вторых, нам придется немного изменить поведение приложения. Нам надо изменить метод `newGame()`, чтобы в нем использовалась новая переменная верхней границы диапазона. Кроме того, нам надо внести изменения в объект `TextView`, в котором в настоящее время выдается фраза: «`Enter a number between 1 and 100:`». Здесь должно отображаться другое приглашение в зависимости от выбранного диапазона. Наконец, нам нужно предоставить пользователю возможность

выбора диапазона. Мы сделаем это, создав еще одно настраиваемое окно оповещений с тремя вариантами выбора: от 1 до 10, от 1 до 100 и от 1 до 1000.

Добавление переменной для верхней границы диапазона

Для начала давайте использовать переменную вместо фиксированного значения 100, которое мы используем при вычислении случайного загаданного числа. В верхней части класса `MainActivity` добавьте переменную для верхней границы диапазона и установите для нее значение по умолчанию `100`:

```
public class MainActivity extends AppCompatActivity {
    private EditText txtGuess;
    private Button btnGuess;
    private TextView lblOutput;
    private int theNumber;
    private int range = 100;
```

При добавлении переменных давайте добавим второй объект `TextView` для метки, на которой написано: `"Enter a number between 1 and 100:"`. Если пользователь выберет верхнее значение диапазона, отличное от 100, эта метка перестанет быть корректной. Поэтому нам нужна переменная, в которой будет храниться соответствующий текст. Для этих целей мы создадим переменную `lblRange`:

```
    private int range = 100;
    private TextView lblRange;
```

Чтобы соединить переменную `lblRange` с графическим интерфейсом добавьте следующую строку кода в метод `onCreate()`:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    txtGuess = (EditText) findViewById(R.id.txtGuess);
    btnGuess = (Button) findViewById(R.id.btnGuess);
    lblOutput = (TextView) findViewById(R.id.lblOutput);
    lblRange = (TextView) findViewById(R.id.textView2);
```

Если вы получите сообщение об ошибке, проверьте имя элемента `TextView` в режиме конструктора: откройте файл **app ⇒ res ⇒ layout ⇒ *content_main.xml*** и щелкните мышью по метке с надписью «Enter a number between 1 and 100:». Присвойте свойству `id` этой метки значение `textView2`.

Теперь у нас есть переменные `range` и `lblRange`. Давайте изменим поведение приложения так, чтобы вместо фиксированных значений в нем использовались эти переменные.

Использование переменной для задания диапазона загаданных значений

Сначала давайте изменим метод `newGame()`, чтобы в нем использовалась переменная верхней границы диапазона. Давайте также добавим код, изменяющий надпись, сообщающую пользователю, в каком диапазоне находится число, которое нужно угадать:

```
public void newGame() {
    theNumber = (int)(Math.random() * range + 1);
    lblRange.setText("Enter a number between 1 and " + range + ".");
    txtGuess.setText("" + range/2);
    txtGuess.requestFocus();
    txtGuess.selectAll();
}
```

Помимо использования переменной `range` для правильного задания случайного загаданного числа мы изменили приглашение `lblRange`, в котором теперь также используется переменная `range`. Последние три строки — это завершающий штрих — я взял на себя смелость установить значение по умолчанию для первого хода пользователя в текстовое поле `txtGuess`. В качестве этого значения я выбрал половину диапазона. Таким образом, если пользователь угадывает числа от 1 до 10, в текстовом поле ввода будет стоять число 5 в качестве первого хода по умолчанию; если загаданное число будет варьироваться от 1 до 1000, текстовое поле будет рекомендовать для первого хода число 500.

Последнее изменение, связанное с диапазоном, надо внести в метод `checkGuess()`. В нем уже реализована инструкция `try-catch` для обработки неправильного ввода пользователя, а в выводе блока `catch` мы говорили пользователю ввести корректное

целое число в диапазоне от 1 до 100. Давайте изменим только инструкцию catch и скорректируем выбранный диапазон пользователя:

```
public void checkGuess() {
— пропуск —
        }
    } catch (Exception e) {
        message = "Enter a whole number between 1 and " + range + ".";
    } finally {
        — пропуск —
    }
}
```

Теперь обе метки `TextView` будут правильно отображать выбранный пользователем диапазон чисел. Пришло время создать окно оповещения, в котором пользователь сможет выбрать уровень сложности игры.

Создание диалогового окна выбора диапазона

Если пользователь выберет в меню команду **Settings**, ему должно быть показано окно, в котором будут отражены все варианты диапазонов (от 1 до 10, от 1 до 100 и от 1 до 1000). Для отображения вариантов мы создадим еще один настраиваемый диалог оповещений, в котором будет отображаться три возможных варианта диапазона.

Найдите метод `onOptionsItemSelected()` и добавьте следующий код в инструкцию `case` для опции `action_settings`:

```
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_settings:
            final CharSequence[] items = {"1 to 10", "1 to 100", "1 to 1000"};
            AlertDialog.Builder builder = new AlertDialog.Builder(this);
            builder.setTitle("Select the Range:");
            builder.setItems(items, null);
            AlertDialog alert = builder.create();
            alert.show();
            return true;
```

Эти шесть строк кода отобразят окно оповещения со списком из трех вариантов диапазона загаданного числа. Однако нам нужно еще добавить код для обработки выбора пользователя. Метод `builder.setItems()` принимает список элементов и слушатель событий для обработки выбора пользователя из списка.

Если пользователь выбирает первый вариант, нам нужно присвоить переменной `range` значение `10`, а для второго и третьего вариантов — `100` и `1000` соответственно. Код для слушателя событий включен внутрь блока команды `builder.setItems()`:

```
case R.id.action_settings:
    final CharSequence[] items = {"1 to 10", "1 to 100", "1 to 1000"};
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setTitle("Select the Range:");
    builder.setItems(items, new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int item) {
            switch(item) {
                case 0:
                    range = 10;
                    newGame();
                    break;
                case 1:
                    range = 100;
                    newGame();
                    break;
                case 2:
                    range = 1000;
                    newGame();
                    break;
            }
            dialog.dismiss();
        }
    });
    AlertDialog alert = builder.create();
    alert.show();
    return true;
```

Обратите внимание, что после задания верхней границы диапазона в каждом из случаев мы вызываем метод `newGame()` для создания нового случайного числа в этом диапазоне и для

соответствующего изменения сообщения пользователю, отражающего выбранный диапазон.

Сохраните файл после внесения изменений и запустите игру, чтобы протестировать новые возможности. Установите диапазон от 1 до 10 и поиграйте несколько раундов. Затем вернитесь к диапазону от 1 до 100, а если вы достаточно храбры, перейдите к диапазону от 1 до 1000.

Когда вы закроете приложение и откроете его снова, вы заметите, что при повторном запуске игра не запоминает предпочитаемый вами диапазон. Приложение также не помнит, насколько вы умелы в угадывании загаданного числа. Если бы существовал способ, чтобы приложение помнило предпочитаемый вами диапазон и количество выигранных игр...

Хранение пользовательских настроек и игровой статистики

Ключом к запоминанию пользовательских настроек и статистики игр является возможность сохранения *постоянной информации* на вашем устройстве Android. Постоянная информация — это любые данные, которые остаются на устройстве после закрытия приложения. В нашей игре «Больше-Меньше» в качестве постоянной информации мы хотим сохранить предпочитаемый уровень сложности игры и количество выигранных пользователем игр.

Существует три способа сохранения постоянных данных на вашем устройстве Android: сохранение общих настроек, сохранение файлов и хранение в базе данных. *Общие настройки* — это тип объекта, хранящий сравнительно короткий список настроек, которые необходимо сохранить до следующего использования приложения. Они называются *общими* настройками, потому что они могут использоваться разными функциональными возможностями или экранами в вашем приложении, например меню настроек и главным экраном игры. Сохранение файла на устройстве полезно, когда вам нужно хранить большой объем данных, таких как текстовый документ, а базы данных необходимы для таких приложений, как адресная книга или список контактов. Для нашей игры «Больше-Меньше» нам просто нужно сохранить несколько чисел, поэтому мы воспользуемся общими настройками.

Хранение и получение предпочитаемого диапазона пользователя

Общие настройки сохраняются в виде наборов *пар ключ/значение*, где каждое значение имеет связанный с ним ключ, необходимый для его получения. Например, у вас может быть пара `"range"` и `"100"`, где `"range"` — это *ключ*, а `"100"` — это *значение*, которое мы храним под этим ключом. Давайте напишем метод хранения предпочитаемого диапазона пользователя в общих настройках. Недалеко от конца файла *MainActivity.java* добавьте следующий метод после метода `onOptionsItemSelected()` и непосредственно перед закрывающей фигурной скобкой:

```
            default:
                return super.onOptionsItemSelected(item);
        }
    }
    public void storeRange(int newRange) {
        SharedPreferences preferences =
        PreferenceManager.getDefaultSharedPreferences(this);
        SharedPreferences.Editor editor = preferences.edit();
        editor.putInt("range", newRange);
        editor.apply();
    }
}
```

Для каждого создаваемого вами приложения уже существует объект общих настроек по умолчанию, к которому вы можете получить доступ, создав объект `SharedPreferences`. Для этого из объекта `PreferenceManager`, который создает и поддерживает списки общих настроек, необходимо вызвать метод `getDefaultSharedPreferences()`. При этом не забудьте импортировать соответствующие классы с помощью сочетания клавиш **Alt+Enter** (⌥ + ⏎ в операционной системе macOS).

Для записи в общие настройки необходимо использовать объект `Editor`, который позволяет редактировать значения отдельных общих настроек. Чтобы сохранить конкретную пару ключ/значение, мы используем один из методов `put`, например `putString` для записи строкового значения, `putInt` для записи целого числа, `putFloat` для записи десятичного значения с плавающей точкой, `putBoolean` для записи значения `true/false` и так далее.

Каждый раз, когда пользователь выбирает новый диапазон, мы передаем переменную `range` методу `storeRange()` под именем `newRange`. Чтобы сохранить значение `newRange` для ключа range, мы используем команду `editor.putInt("range", newRange);`. Таким образом, мы записываем новое значение верхней границы диапазона (10, 100 или 1000) в общие настройки с именем ключа `"range"`. Метод `apply()` сообщает устройству Android, что вы закончили обновление значений общих настроек и изменения можно применить.

Таким образом мы можем сохранить диапазон загаданного числа в общих настройках.

Теперь нам нужно добавить метод `storeRange()` во все варианты диапазона, который пользователь может выбрать, то есть к каждой команде `case` в слушателе событий внутри метода `onOptionsItemSelected()`:

```
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_settings:
        final CharSequence[] items = {"1 to 10", "1 to 100", "1 to 1000"};
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setTitle("Select the Range:");
        builder.setItems(items, new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int item) {
                switch(item) {
                    case 0:
                        range = 10;
                        storeRange(10);
                        newGame();
                        break;
                    case 1:
                        range = 100;
                        storeRange(100);
                        newGame();
                        break;
                    case 2:
                        range = 1000;
                        storeRange(1000);
                        newGame();
                        break;
                }
```

```
            dialog.dismiss();
        }
    });
    AlertDialog alert = builder.create();
    alert.show();
    return true;
```

Чтобы последний выбранный пользователем диапазон загаданных чисел стал диапазоном, который игра будет использовать при следующем запуске, нам необходимо получить значение верхней границы в момент загрузки игры. Найдите метод `onCreate()` и добавьте в него следующие две строки для получения значения верхней границы диапазона из общих настроек:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    txtGuess = (EditText) findViewById(R.id.txtGuess);
    btnGuess = (Button) findViewById(R.id.btnGuess);
    lblOutput = (TextView) findViewById(R.id.lblOutput);
    lblRange = (TextView) findViewById(R.id.textView2);
    SharedPreferences preferences =
        PreferenceManager.getDefaultSharedPreferences(this);
    range = preferences.getInt("range", 100);
    newGame();
```

Обратите внимание, что мы получили общие настройки *перед тем*, как вызвать метод `newGame()`, для гарантии того, что загаданное число будет в том же диапазоне, в котором оно было при последнем запуске приложения. Метод `getInt()` ищет значение, хранящееся в ключе `"range"`, но если он не сможет найти значение, из-за наличия второго параметра он вернет значение по умолчанию `100`. Мы сделали это для того, чтобы переменная `range` имела значение уже при первом запуске приложения.

Сохраните файл, соберите и запустите его. На этот раз выберите другой диапазон, а затем полностью закройте приложение. При новом запуске игры она будет помнить выбранный вами диапазон.

Сохранение количества побед

Высокие результаты, таблица лучших игроков, количество выигрышей подряд — все, что фиксирует наши достижения, как правило, заставляет нас играть лучше, играть дольше и бить рекорды. Последний штрих, который мы хотим добавить в игру, — это способность отслеживать количество выигранных игр. Мы можем легко сохранить это число среди общих настроек.

Когда пользователь отгадывает число и выигрывает, мы можем использовать общие настройки и получить количество выигранных им на данный момент игр. Затем надо добавить 1 к этому числу и сохранить новое значение. Давайте добавим этот код в метод `checkGuess()`, а именно в блок инструкции `else` для верного хода пользователя:

```
    public void checkGuess() {
        String guessText = txtGuess.getText().toString();
        String message = "";
        try {
            int guess = Integer.parseInt(guessText);
            if (guess < theNumber)
                message = guess + " is too low. Try again.";
            else if (guess > theNumber)
                message = guess + " is too high. Try again.";
            else {
                message = guess + " is correct. You win! Let's play again!";
❶           SharedPreferences preferences =
                PreferenceManager.getDefaultSharedPreferences(this);
❷           int gamesWon = preferences.getInt("gamesWon", 0) + 1;
❸           SharedPreferences.Editor editor = preferences.edit();
❹           editor.putInt("gamesWon", gamesWon);
❺           editor.apply();
            newGame();
        }
```

В пункте ❶ мы получили доступ к объекту `SharedPreferences` по умолчанию, а в строке ❷ мы получили значение, хранящееся под ключевым именем `gamesWon` (со значением по умолчанию `0`, если это первый выигрыш пользователя) и добавили `1`, чтобы учесть эту победу. В пункте ❸ мы создаем редактор, чтобы написать новое значение в общие настройки. В строке ❹ мы помещаем

целочисленное значение gamesWon в общие настройки под ключом с соответствующим именем (для последующего использования), а в пункте ❺ указываем устройству Android применить изменения.

Написанный нами код будет хранить количество выигранных игр. Как же мы отобразим эту информацию пользователю? Для этого нам нужно добавить код для блока case, соответствующего выбору пользователем опции action_gamestats в методе onOptionsItemSelected(), как показано ниже:

```
    case R.id.action_gamestats:
❶       SharedPreferences preferences =
            PreferenceManager.getDefaultSharedPreferences(this);
❷       int gamesWon = preferences.getInt("gamesWon", 0);
❸       AlertDialog statDialog = new AlertDialog.Builder(MainActivity.this).create();
        statDialog.setTitle("Guessing Game Stats");
❹       statDialog.setMessage("You have won "+gamesWon+" games. Way to go!");
        statDialog.setButton(AlertDialog.BUTTON_NEUTRAL, "OK",
            new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int which) {
                    dialog.dismiss();
                }
            });
        statDialog.show();
        return true;
```

В пункте ❶ мы подключаемся к общим настройкам приложения по умолчанию. В строке ❷ мы получаем количество выигранных игр (со значением по умолчанию 0, если это первый выигрыш пользователя). В пункте ❸ мы создаем диалоговое окно оповещения, а в пункте ❹ мы показываем количество выигранных пользователем игр вместе с ободряющим сообщением.

Сохраните все изменения, а затем постройте и запустите приложение. Возможно, вашей самой сложной задачей будет прекратить играть! На рис. 5.3 показано, как может выглядеть экран со статистикой, если в вашу игру играет пара математических гениев.

Добавление меню настроек, сохранение статистики игр и предпочтений пользователя, отображение диалоговых окон оповещения — это те завершающие штрихи, которые могут сделать вашу

игру или любое другое приложение действительно профессионально выглядящим, полностью функциональным мобильным приложением. Продолжайте улучшать свое приложение, придумывайте новые функции, и у вас получится приложение, которым будет не стыдно поделиться с друзьями или со всем миром. Счастливого программирования!

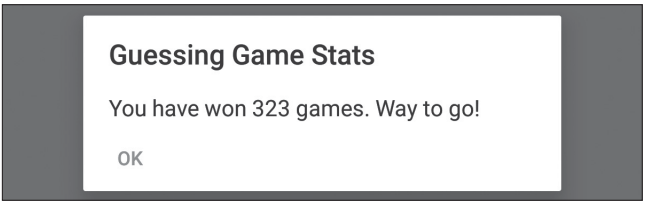


Рис. 5.3. Экран со статистикой игры показывает количество раз, когда вы (или ваши друзья) отгадали загаданное компьютером число

Что вы изучили

Вы создали вполне профессиональную мобильную игру для Android, добавив несколько завершающих штрихов в игру «Больше-Меньше», включая:

- добавление меню настроек в приложение для ОС Android;
- разработка меню настроек с помощью редактирования файла меню на языке XML;
- отображение меню настроек с помощью объекта `MenuInflater`;
- реакция на выбор пользователя в меню;
- использование инструкции `switch` с несколькими инструкциями `case` для замены длинных цепочек `if-else`;
- создание настраиваемых всплывающих окон на Android с помощью класса `AlertDialog`;
- сохранение общих настроек и статистики приложения с помощью класса `SharedPreferences`;
- получение общих настроек пользователя при запуске приложения.

Дополнительные задачи

Попробуйте выполнить следующие задания. Таким образом, вы повторите пройденный материал, попрактикуетесь в программировании и получите новые навыки. Если у вас возникнут проблемы, вы можете загрузить примеры решений на сайте по адресу **https://eksmo.ru/files/learn_java_easy_way.zip**.

Задача № 1: Иногда вы выигрываете, иногда проигрываете

В дополнительной задаче № 1 в главе 4 (с. 106) требовалось дать пользователю семь попыток на угадывание числа от 1 до 100. После добавления возможности изменять диапазон загаданных чисел вам также нужно соответствующим образом изменить количество попыток.

В главе 4 вы узнали, что число от 1 до 100 можно угадывать с помощью стратегии двоичного поиска (когда каждый ход должен быть в центре диапазона оставшихся возможных значений). Поскольку 2^7 (или два в седьмой степени) равно 128, это гарантирует, что, используя метод двоичного поиска, мы можем отгадать число от 1 до 128 за семь попыток. Но сколько же ходов нам нужно, чтобы гарантированно отгадать число в диапазоне от 1 до 10 или от 1 до 1000?

Чтобы определить количество необходимых попыток, нам нужно знать наименьшую степень, в которую можно возвести число 2, чтобы получить число, превышающее верхнюю границу диапазона. Например, для чисел от 1 до 10 берем $2^4 = 16$, и $16 > 10$, поэтому для этого диапазона нам нужно не более четырех попыток; для диапазона от 1 до 1000 берем $2^{10} = 1024$, следовательно, нам нужно 10 ходов.

Чтобы найти степень, в которую нужно возвести число для получения заданного числа, можно использовать *логарифмы*. Логарифм числа по заданному основанию равен степени, в которую надо возвести основание, чтобы получить означенное число. В языке программирования Java реализован метод `Math.log()`, который принимает число и находит логарифм по основанию 10. Если вы разделите логарифм одного числа по основанию 10 на логарифм другого числа по тому же основанию, вы получите результат, соответствующий логарифму первого числа по основанию второго. Это означает, что деление `Math.log(range)` на `Math.log(2)`

подскажет вам, в какую степень нужно возвести число 2, чтобы получить величину range. Степени могут оказаться дробными числами, например, `7,25`, и тогда вам придется округлить и привести результат к типу `int`, потому что вы не можете дать пользователю нецелое число попыток. Чтобы определить количество попыток, необходимых для гарантированного отгадывания числа в каждом из диапазонов, можно использовать выражение `(int)(Math.log(range)/Math.log(2)+1)`.

Измените игру «Больше-Меньше», чтобы максимальное количество попыток соответствовало выбранному диапазону загадываемых значений, как при запуске игры, так и при каждой смене диапазона в меню настроек. Например, вы можете создать переменную с именем `maxTries`, которая будет использоваться в программе вместо фиксированного значения `7` при проверке, остались ли у пользователя еще попытки.

Задача № 2: Соотношение между победами и поражениями

Выполнив задачу 1, измените приложение так, чтобы сохранялось количество не только выигранных игр, но *и* проигранных тоже. Измените код в опции игровой статистики так, чтобы получить оба числа и показать количество выигранных игр, общее количество игр и процент выигранных игр (число выигранных игр делится на обще число игр и умножается на 100). На рис. 5.4 показан пример.

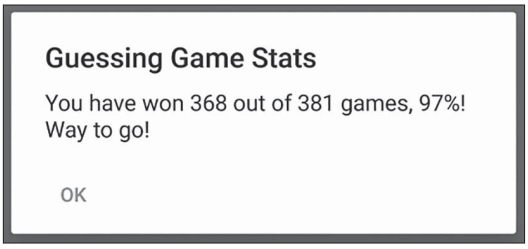
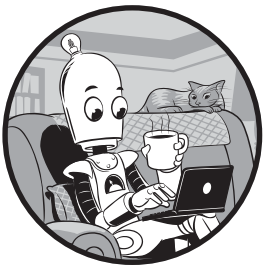


Рис. 5.4. На экране «Статистика игры» выводится процент выигранных игр

Глава 6

РАСШИФРОВКА СЕКРЕТНЫХ СООБЩЕНИЙ



Вы наверняка писали секретные зашифрованные записки своим друзьям, чтобы смысл написанного не могли понять ваши родители или учителя. В этой главе мы будем делать что-то подобное и создадим приложение для секретных сообщений.

Наше предыдущее приложение (игра «Больше-Меньше») работало с числами — больше, меньше, угадал. Приложение «Секретные сообщения», напротив, будет сосредоточено на тексте. Вы научитесь работать с текстовыми строками и манипулировать символьными значениями в языке программирования Java для шифровки сообщений.

Шифр Цезаря

Наше приложение будет зашифровывать и расшифровывать секретные сообщения с использованием *шифра Цезаря*, алгоритма, разработанного более 2000 лет назад, и использующего буквенные подстановки для зашифровки сообщений.

Шифр Цезаря назван в честь римского императора Юлия Цезаря (100–44 до н. э.). Историки утверждают, что Цезарь любил шифровать свои личные сообщения, например записки своим генералам, путем «сдвига» букв алфавита. Например, шифровальный диск, показанный на рис. 6.1, сдвигает алфавит на 13 букв.

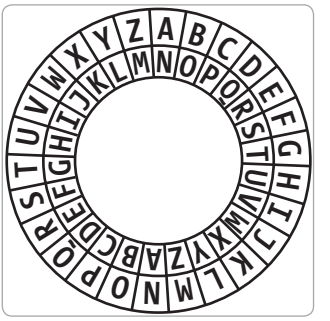


Рис. 6.1. Шифровальный диск шифра Цезаря с ключевым значением 13

Буквы на внешнем круге диска соответствуют буквам на его внутреннем круге. При замене оригинальной литеры на соответствующую ей буква A становится буквой N, буква B становится буквой O и так далее. С помощью этого шифра мы зашифровали следующую фразу:

```
Secret messages are so cool!
Frperg zrffntrf ner fb pbby!
```

Первая строка — это *открытый текст*: оригинальная, читаемая версия сообщения. Вторая строка — это *зашифрованный текст*, закодированная версия этого сообщения.

Попробуйте расшифровать это сообщение с помощью обратной замены: найдите каждую букву на внутреннем круге и замените ее соответствующей буквой на внешнем.

Не все шифры Цезаря симметричны, как этот. *Симметричность* означает, что один и тот же процесс может использоваться как для шифрования, так и для расшифровки сообщения. Например, F становится S при шифровании путем добавления 13, а S становится F при расшифровке путем вычитания 13. В данном случае можно использовать один и тот же диск и *ключ* — число позиций, на которое сдвигаются буквы — для обоих процессов. Число, на которое сдвигаются буквы (в данном случае 13) называется ключом, поскольку зная его, можно «открыть» шифр.

Мы создадим приложение «Секретные сообщения», которое позволит использовать любое значение ключа.

Настройка приложения «Секретные сообщения»

Мы создадим приложение «Секретные сообщения» так же, как мы делали игру «Больше-Меньше»: сначала мы создадим консольную версию, затем графический интерфейс для компьютера и, наконец, мобильное приложение для устройств под управлением операционной системы Android. Консольная версия будет выглядеть довольно просто, как показано на рис. 6.2. Однако она позволит нам легко и быстро протестировать алгоритм секретных сообщений.

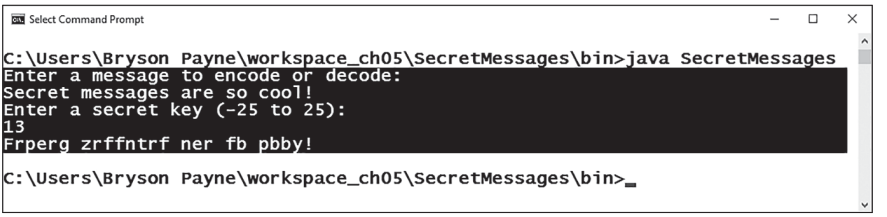


Рис. 6.2. Консольная версия приложения «Секретные сообщения»

Как показано на рис. 6.2, программа просит пользователя ввести сообщение для шифрования или расшифровки, а затем запрашивает значение секретного ключа. Затем программа выдает зашифрованное сообщение. Обратите внимание, что изначально наше приложение будет шифровать все сообщение, включая пробелы и пунктуацию. Окончательную версию мы сделаем более логичной: там зашифрованными будут только буквы, как показано на рис. 6.2.

Создание проекта «Секретные сообщения» в среде разработки Eclipse

Давайте начнем с открытия программы Eclipse. Если у вас открыты файлы других проектов, закройте их. Чтобы создать новый проект на языке Java, выберите команду меню **File** ⇒ **New** ⇒ **Java Project** (Файл ⇒ Новый ⇒ Проект Java) и присвойте проекту имя *SecretMessages*. Нажмите кнопку **Finish** (Готово).

На панели **Package Explorer** (Обозреватель пакетов) разверните папку проекта *SecretMessages*, чтобы была видна папка *src*. Щелкните правой кнопкой мыши по папке *src* и выберите пункт **New** ⇒

Class (Создать ⇒ Класс), чтобы создать новый файл с исходным кодом на языке Java, который назовите также *SecretMessages*. Установите флажок, чтобы создать метод `main()`, как показано на рис. 6.3.

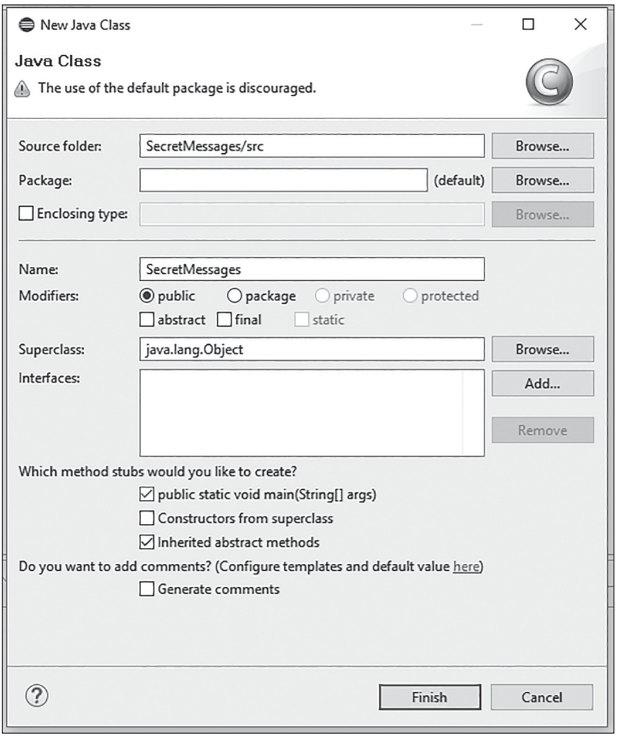


Рис. 6.3. Создайте новый проект на языке Java с новым файлом класса `SecretMessages` и установите флажок, чтобы создать заглушку метода `main()`

Нажмите кнопку **Finish** (Готово), и вы увидите файл *SecretMessages.java* в главном окне в среде разработки Eclipse. Приступим к написанию кода.

Начало работы с кодом в файле *SecretMessages.java*

В верхней части файла *SecretMessages.java* над объявлением `public class SecretMessages` добавьте инструкцию импорта для класса `java.util.Scanner`, чтобы мы могли запросить у пользователя ввод:

```
import java.util.Scanner;
public class SecretMessages {
```

```
    public static void main(String[] args) {
❶       Scanner scan = new Scanner(System.in);
❷       System.out.println("Enter a message to encode or decode:");
    }
}
```

Внутри метода `main()` в строке ❶ мы создали объект класса `Scanner` с именем `scan`. В строке ❷, мы предлагаем пользователю ввести сообщение для шифрования или расшифровки.

Затем мы создаем переменную типа `String`, с именем `message`, которая примет введенную пользователем строку:

```
        System.out.println("Enter a message to encode or decode:");
        String message = scan.nextLine();
```

С помощью метода `nextLine()` объекта `scan` мы получаем строку ввода пользователя, вплоть до символа `ENTER` (ввода) или `RETUTN` (возврата строки), и сохраняем ее в строковую переменную `message`.

Итак, приложение может запросить у пользователя ввод строки и сохранить ее в переменной. Теперь нам нужно научиться манипулировать символами внутри строки, чтобы создать зашифрованную версию сообщения. Сохраните файл, прежде чем перейти к следующему разделу.

Работа со строками

До этого момента приложение «Секретные сообщения» выглядело очень похоже на игру «Больше-Меньше». Мы настроили сканер ввода, предложили пользователю что-то ввести, сосканировали с консоли ответ пользователя, а затем зафиксировали введенное значение в переменной. Что будет отличать «Секретные сообщения», так это работа программы с символами внутри строки.

Чтобы сделать консольное приложение, работающее с шифром Цезаря, потребуется выполнить много шагов, поэтому мы будем создавать его *итерациями*. Это означает, что вместо того, чтобы писать сразу все приложение и только в конце проверить, как работает код, мы будем создавать наше приложение по частям, или по одной итерации за раз, и тестировать его по мере продвижения. Таким образом, после каждой итерации у нас будет работающий

код. Безусловно, не на всех итерациях у нас будут реализованы все функции, которые нам будут нужны, но в итоге мы получим законченное полнофункциональное приложение.

Нам понадобится использовать текстовую обработку для превращения введенной строки в ту, что мы будем выводить. Давайте начнем работу со строками текста на языке программирования Java с создания простого перевертыша сообщений. Другими словами, мы берем сообщение, введенное пользователем, а затем возвращаем его, переставив буквы в обратном порядке. Так, например, сообщение `Meet me at the arcade at 5pm` превратится в `mp5 ta edacra eht ta em teeM`.

Для начала создадим переменную `output` для перевернутой или реверсированной строки и присвоим ей значение, равное пустой строке:

```
String message = scan.nextLine();
String output = "";
```

Нам понадобится цикл для прохода по всем символам в сообщении. Для удобства выберем цикл `for`, поскольку мы знаем количество символов в сообщении. (Позже для определения количества символов в сообщении мы будем использовать метод `message.length()`.) Объявление цикла `for` на языке программирования Java требует выполнения трех операций: *инициализации* переменной цикла, проверки *условия* продолжения цикла и *обновления* переменной цикла для следующей итерации. Каждую из операций друг от друга мы отделяем точкой с запятой. Синтаксис выглядит следующим образом:

```
for (инициализация; условие; обновление) {тело_цикла}
```

В качестве примера введите указанный ниже код для создания цикла `for`:

```
jshell> for (int x = 0; x < 10; x++) {System.out.println(x);}
```

Этот цикл выведет на экран числа от `0` до `9`. Инициализация установит значение первой переменной цикла `x` равным `0`. Далее, прежде чем продолжить, будет проверено условие, что `x` меньше `10`. Наконец, обновление увеличивает `x` на 1 после каждой итерации цикла. Краткая запись `x++` называется *инкрементом* или *инструкцией приращения*, поскольку она приращивает или добавляет

1 к x после каждого прохода через цикл. Запись x++ эквивалентна присвоению x = x + 1.

В этом примере цикл for выводит на экран значение x после каждого прохода через цикл. Цикл повторяется 10 раз и заканчивается, когда x становится не меньше 10.

Таким образом, чтобы расположить символы в строке сообщения в обратном порядке, мы должны инициализировать переменную номером позиции или *индексом* последнего символа в строке, а затем пройти по символам строки от последнего до первого (в обратном порядке). В нашем примере это будет выглядеть следующим образом:

```
for (int x = message.length()-1; x >= 0; x--) {}
```

Позиции символов в строке в Java нумеруются от 0 (индекс первого символа) до длины строки, уменьшенной на 1. Индекс первого символа 0, а *n*-й символ в строке имеет индекс (*n*-1).

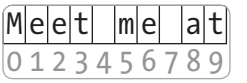


Рис. 6.4. Первые десять символов сообщения помечены соответствующими индексами

На рис. 6.4 показаны первые 10 символов сообщения из нашего примера. Под каждым символом показан его индекс в строке. У первой буквы (M) индекс 0, у второй буквы (e) индекс 1 и так далее. Обратите внимание, что пробелы считаются символами — на рисунке показаны два пробела, с индексами 4 и 7. Индекс десятого символа сообщения (буквы t) равен 9. Такая логика продолжается до конца строки, то есть до символа с индексом, равным message.length()-1.

Мы начнем писать обратное сообщение с конца строки, поэтому инициализируем x значением, равным message.length()-1, то есть индексом последнего символа в сообщении. В качестве условия используем x >= 0, поскольку хотим продолжать идти по строке вплоть до первого ее символа, то есть символа с индексом 0. Наконец, обновление выглядит так: x— поскольку мы проходим по строке назад. В противоположность x++, x— использует *декремент или оператор уменьшения*, который каждый раз уменьшает значение x на 1. Вернитесь в среду разработки Eclipse и начните писать цикл for чуть ниже предыдущей строки кода:

```
String output = "";
for (int x = message.length()-1; x >= 0; x--) {
}
```

Чтобы получить символ, находящийся в заданной позиции в строке, мы используем метод `charAt()` (*character at* — символ в) и передаем ему индекс символа, который хотим получить. Чтобы добавить символ в конец строки, можно использовать оператор `+`. Объединив эти приемы, можно построить перевернутую версию строки `message` и сохранить ее в строке `output`:

```
for (int x = message.length()-1; x >= 0; x—) {
    output += message.charAt(x);
}
```

Тело цикла `for` представлено единственной строкой. Мы получаем из строки `message` символ с индексом `x` и добавляем его к строке `output`. Не забудьте поставить фигурные скобки вокруг тела цикла `for`, поскольку мы собираемся по мере роста приложения добавить внутрь цикла дополнительные строки.

Нам осталось просто вывести результат на экран. Это можно сделать с помощью метода `System.out.println(output)`. Полностью эта итерация нашего приложения показана в листинге 6.1.

```
import java.util.Scanner;
public class SecretMessages {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter a message to encode or decode:");
        String message = scan.nextLine();
        String output = "";
        for (int x = message.length()-1; x >= 0; x—) {
            output += message.charAt(x);
        }
        System.out.println(output);
    }
}
```

Листинг 6.1. Первая итерация приложения «Секретные сообщения» переставляет в обратном порядке символы в сообщении пользователя

Вы можете запустить программу в среде разработки Eclipse и протестировать ее с помощью собственного сообщения, как показано на рис. 6.5. Это не окончательная версия приложения

«Секретные сообщения», однако уже в этой версии чтение исходного сообщения затруднено, при этом его легко расшифровать.

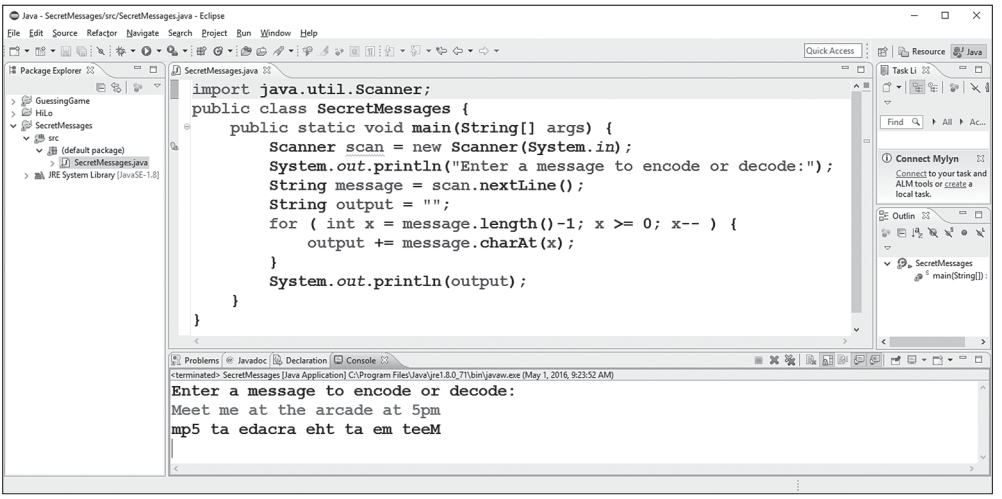


Рис. 6.5. Запустите программу и введите свое собственное сообщение. Сообщение с переставленными в обратном порядке буквами появится в консоли в нижней части экрана

Вы даже можете расшифровать сообщение, скопировав зашифрованное перевернутое сообщение и введя его в программу, как показано на рис. 6.6. Таким образом, вы можете ввести сообщение, зашифровать его и вставить в сообщение другу, а затем он его расшифрует, вставив закодированное сообщение в ту же программу. Вы создали свое первое шифровальное приложение!

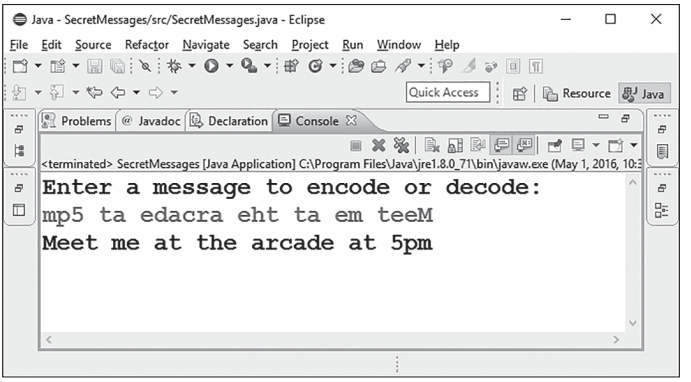


Рис. 6.6. Копирование и вставка зашифрованного сообщения в работающую программу позволяет его расшифровать

Эта версия приложения настолько проста, что сообщения можно читать и без помощи программы. В общем, это не самый безопасный способ отправки сообщений вашим друзьям. Но вы научились проходить в цикле `for` строку текста, определять длину строки методом `length()`, получать символ по заданному индексу или позиции в строке с помощью метода `charAt()` и добавлять символы в конец строки, используя оператор `+`. В следующем разделе вы узнаете, как изменить значения отдельных символов в сообщении, чтобы сделать его более трудным для чтения, но по-прежнему простым для расшифровки нашей программой.

Символы и значения в Java

Для разработки хорошей шифровальной программы требуется работа с символьными значениями в строках текста. Для шифра Цезаря мы должны уметь сдвигать эти значения, например, изменить *A* на *N*, а *N* на *A*. Для этого вам необходимо понять, как символы хранятся в памяти компьютера.

В языке программирования Java отдельные символы, такие как `'A'`, могут храниться в собственном типе данных: `char`. Обратите внимание, что мы используем одиночные кавычки для обозначения значений символов. Метод `charAt()`, примененный к строке, возвращает значение `char`, представляющее собой один символ. Java использует 16-разрядные символы в кодировке Юникод. *Юникод* — это международный набор символов, представленных в виде числовых значений. Юникод содержит тысячи всевозможных символов из алфавитов всего мира. Тип `char` — способ хранения символов Юникода, таких как `'A'`, `'ñ'` и `'ç'`, с помощью их числовых значений.

Мы можем прибавить к переменной типа `char` значение так же, как мы можем прибавить его к переменной типа `int`. В приложении «Секретные сообщения» нам нужно прибавить к значению char для `'A'` (`65`) значение ключа (`13`), чтобы получить новое значение `char` (`78`), которое представляет зашифрованную букву (`'N'`).

Для нашей второй итерации приложения, давайте начнем кодировать шифр Цезаря, с создания переменной типа `char` с именем `key` и присвоения ей значения `13`. Мы напишем эту строку под строкой `String output`:

```
String output = "";
char key = 13;
```

Кроме того, нам надо изменить цикл `for` таким образом, чтобы теперь проходить строку с начала до конца. На этот раз мы начнем с индекса `0` и будем перемещаться по строке до тех пор, пока `x` меньше, чем `message.length()`. Каждый раз, начиная цикл заново, мы будем увеличивать `x` на единицу, изменяя позицию символа. Новый цикл `for` будет выглядеть следующим образом:

```
        char key = 13;
        for (int x = 0; x < message.length(); x++) {
```

Кроме того, вместо добавления исходных символов строки `message` к строке `output` нам нужно прибавить значение ключа к каждому символу, сначала убедившись, что мы получаем значение типа `char`, которое можно добавить к строке результата:

```
            output += (char)(message.charAt(x) + key);
```

Мы прибавили значение ключа к каждому символу в строке `message`, а затем взяли эту сумму и привели ее к типу `char`. Ключевое слово char в круглых скобках перед выражением означает, что значение справа от него будет преобразовано в тип данных `char` или значение будет *приведено* к этому типу. Мы должны в данном случае привести значение к типу `char`, поскольку в Java правая часть этого выражения может по умолчанию получить значение типа `int`. В листинге 6.2 показана программа после внесения этих изменений.

```
import java.util.Scanner;
public class SecretMessages {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter a message to encode or decode:");
        String message = scan.nextLine();
        String output = "";
        char key = 13;
    vfor (int x = 0; x < message.length(); x++) {
            output += (char)(message.charAt(x) + key);
```

```
            }
            System.out.println(output);
        }
    }
```

Листинг 6.2. Приложение «Секретные сообщения» хотя и ограничено по-прежнему только лишь 14 строками, но теперь оно зашифровывает строки текста

Если сейчас вы запустите программу и введете сообщение, вы увидите следующий результат:

```
Enter a message to encode or decode:
Secret messages are so cool!
`rp□r?-zr??ntr?-n□r-?|-p||y.
```

Сообщение зашифровано, но мы ожидали несколько другого результата. Во-первых, мы зашифровываем все символы, включая пробелы и знаки препинания. Во-вторых, мы не всегда получаем буквы, поскольку мы *в любом случае* добавляем 13 к каждому символу, не учитывая того факта, что буквы, расположенные в конце алфавита, должны закольцовываться в его начало. Из-за этого в зашифрованном сообщении мы получили странные знаки и непечатаемые символы. Мы рассмотрим оба эти вопроса в следующем разделе. Прежде чем двигаться дальше, сохраните свою программу.

Шифрование только букв

Во второй итерации приложения «Секретные сообщения» мы получили зашифрованное сообщение, но в нем отсутствуют некоторые моменты, которые мы бы хотели получить в финальном приложении. Для создания финальной версии нам потребуется добавить инструкции `if` и условия для кодирования только букв (а не пробелов или знаков препинания) и для закольцовывания алфавита во время шифрования. Эти улучшения мы реализуем в рамках нашей третьей итерации.

Сначала выполним проверку каждого символа в исходном сообщении, а не просто будем добавлять символ непосредственно к конечному сообщению. Давайте изменим тело цикла `for` следующим образом:

```
for (int x = 0; x < message.length(); x++) {
    char input = message.charAt(x);
}
```

Переменная `input` сначала будет хранить первый символ строки `message`, затем второй и так далее. Нам нужно проверить один за другим каждый символ и выяснить, нужно ли его зашифровывать. Мы собираемся кодировать буквы, добавляя к ним значение ключа и, при необходимости, закольцовывая к началу алфавита. В противном случае, если символ является пробелом или знаком препинания, мы оставляем его без изменений и добавляем в конечное сообщение.

Давайте добавим следующую инструкцию `if` в тело нашего цикла `for`:

```
char input = message.charAt(x);
if (input >= 'A' && input <= 'Z')
```

В условии в инструкции `if` можно использовать либо `65` (значение `'A'`), либо сам символьный литерал `'A'`. Условие проверяет, находится ли символ, хранящийся в переменной `input`, в интервале от `'A'` до `'Z'` включительно. Другими словами, содержит ли переменная `input` прописную букву латинского алфавита. Если это условие истинно, мы прибавим значение ключа к значению переменной `input`, чтобы сдвинуть его, тем самым зашифровав эту букву шифром Цезаря. Обратите внимание, что в данном случае мы проверяем только прописные буквы — строчные буквы потребуют отдельной проверки.

Теперь, в теле инструкции `if`, мы прибавляем значение ключа к символу, хранящемуся в переменной `input`, и зашифровываем букву. Кроме того, здесь мы будем обрабатывать закольцовывание к началу алфавита, если зашифрованное значение сдвигается дальше буквы `'Z'`. Таким образом, код внутри цикла `for` станет следующим:

```
for (int x = 0; x < message.length(); x++) {
    char input = message.charAt(x);
    if (input >= 'A' && input <= 'Z')
{
❶       input += key;
❷       if (input > 'Z')
```

```
❸             input -= 26;
          }
❹         output += input;
      }
```

Получив очередной символ из сообщения и убедившись, что это прописная буква, мы ее зашифровываем, добавляя к ней значение ключа (строка ❶). Затем мы проверяем, выходит ли результат, полученный после добавления значения ключа, за букву `'Z'` (строка ❷). Если это так, то в строке ❸ мы вычитаем 26 (количество букв в английском алфавите) из зашифрованного исходного значения, тем самым возвращая его назад к началу алфавита. Наконец, мы можем добавить полученный в переменной `input` символ к конечной строке `output` (строка ❹).

Если вы сейчас запустите программу, то сможете зашифровать сообщение из прописных букв следующим образом:

```
Enter a message to encode or decode:
SECRET MESSAGES ARE SO COOL!
FRPERG ZRFFNTRF NER FB PBBY!
```

Обратите внимание, что пунктуация и пробелы остаются неизменными, а все прописные буквы сдвигаются на 13 символов. При этом зашифрованный символ «закольцовывается», если он должен выйти за границы алфавита. Таким образом, буква `S` стала буквой `F` и так далее.

Шифрование строчных букв логически идентично обработке прописных. Можно скопировать и вставить тот же код инструкции `if`, но заменить прописные буквы `A` и `Z` на строчные. Не забудьте добавить ключевое слово `else` перед второй инструкцией `if`. Попробуйте сделать это самостоятельно. Если у вас возникнут проблемы, можете посмотреть полный код, приведенный в листинге 6.3 в следующем разделе.

ШИФРОВАНИЕ СООБЩЕНИЙ НА ДРУГИХ ЯЗЫКАХ

Версия программы, которую мы пишем, будет работать только для основных латинских букв от A до Z и от a до z. Однако если вы предпочитаете использовать не английский язык, вы можете

исправить программу, если в Юникоде есть алфавит или символы вашего языка, сгруппированные вместе в непрерывном наборе, один за другим. Так же, как и в приложении для английского алфавита, вы можете проверить, попадает ли символ в интервал между первой и последней буквами вашего алфавита. Если же буквы вашего алфавита либо представлены в Юникоде не подряд, либо используются некоторые базовые латинские буквы, а также некоторые другие символы (например, в испанском языке используется ñ, а во французском — ç и также другие символы с диакритическими знаками), вы можете прибавлять значение ключа к любому символу, как мы делали в листинге 6.2. По желанию в сообщении можно оставить пробелы, проверив символы с помощью метода `Character.isSpace()` и зашифровать все, кроме них. Поиграйтесь с различными методами для выбранного вами алфавита и выберите наиболее подходящую схему шифрования. Изменение программы и добавление в нее новых возможностей — это самый лучший способ развить навыки программирования!

Закрытие сканера

Здесь нужно добавить еще один штрих, который вы можете помнить по главе 2. Переменная `scan` класса Scanner, скорее всего, подчеркнута желтой линией, что в среде разработки Eclipse означает предупреждение, указывающее на возможность утечки ресурсов. Если вы наведете указатель мыши на это предупреждение, появится сообщением `'scan' is never closed`. Не забывайте, что нам нужно закрыть все ресурсы ввода/вывода, такие как объекты класса `Scanner`, после окончания работы с ними. Для этого добавим команду `scan.close()` сразу после последнего метода `System.out.println` в методе `main()` программы:

```
        System.out.println(output);
        scan.close();
```

После добавления этой строки предупреждение об утечке ресурсов исчезнет. В листинге 6.3 представлена полнофункциональная версия приложения для шифрования и расшифровки сообщений с помощью шифра Цезаря с ключом 13.

```
import java.util.Scanner;
public class SecretMessages {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter a message to encode or decode:");
        String message = scan.nextLine();
        String output = "";
        char key = 13;
        for (int x = 0; x < message.length(); x++) {
            char input = message.charAt(x);
            if (input >= 'A' && input <= 'Z')
            {
                input += key;
                if (input > 'Z')
                input -= 26;
            }
            else if (input >= 'a' && input <= 'z')
            {
                input += key;
                if (input > 'z')
                input -= 26;
            }
            output += input;
        }
        System.out.println(output);
        scan.close();
    }
}
```

Листинг 6.3. Полнофункциональная версия приложения для шифрования и расшифровки сообщений с помощью шифра Цезаря со значением ключа 13

Запустите программу и зашифруйте сообщение. Затем скопируйте зашифрованное сообщение и снова запустите программу; вставьте зашифрованное сообщение в качестве исходного текста и нажмите клавишу **Enter**. Программа ответит исходным, расшифрованным сообщением. Ниже представлены два примера прогона программы, в которых я скопировал зашифрованное сообщение из первого запуска и вставил его как открытый текст во втором:

```
Enter a message to encode or decode:
Secret messages are so cool!
Frperg zrffntrf ner fb pbby!
Enter a message to encode or decode:
Frperg zrffntrf ner fb pbby!
Secret messages are so cool!
```

Шифр Цезаря со значением ключа 13 симметричен, поэтому если ввести в программу зашифрованное сообщение, она расшифрует его в исходный открытый текст. Это означает, что вы можете запустить программу, представленную в листинге 6.3, и зашифровать сообщение. Затем зашифрованный текст можно отправить другу, у которого есть такая же шифровальная программа, которая мгновенно расшифрует ваше сообщение.

К сожалению, это означает, что *любой*, кто запустит программу (или любой, кто знает этот шифр), может так же быстро расшифровать ваши сообщения. В нашей следующей итерации давайте сделаем код интереснее, позволяя пользователю устанавливать собственное значение ключа. Это позволит вам выбирать ключ для отправки сообщений разным людям.

Добавление ключа пользователя

Программа шифрования секретных сообщений хорошо работает с ключевым значением `13`. Что делать, если мы хотим зашифровывать и расшифровывать сообщения с использованием другого значения ключа, например `3` (классический сдвиг шифр Цезаря) или `5`, или `25`?

Нам нужно будет запросить у пользователя, помимо сообщения, еще и значение ключа. Мы можем сделать это, добавив запрос перед строкой, которая создает переменную ключа `char`:

```
        String output = "";
        System.out.println("Enter a secret key (-25 to 25):");
    vchar key = 13;
```

Мы разрешаем отрицательные значения ключа для удобства расшифровки сообщений. Если вы используете значение ключа `5`, чтобы зашифровать сообщение и отправить его другу, то он может использовать значение ключа `-5` для расшифровки.

После того как пользователь ввел значение ключа, программа должна отсканировать строку ввода, извлечь из нее целочисленное значение и сохранить его в переменной типа `int`:

```
String output = "";
System.out.println("Enter a secret key (-25 to 25):");
int keyVal = Integer.parseInt(scan.nextLine());
```

Наконец, вместо того, чтобы использовать `13` в качестве значения переменной `key`, мы установим этой переменной следующее значение:

```
int keyVal = Integer.parseInt(scan.nextLine());
char key = (char) keyVal;
for (int x = 0; x < message.length(); x++) {
```

Здесь мы приводим переменную `keyVal` к типу `char` и сохраняем это значение в переменной `key`, поскольку не можем хранить целочисленное значение непосредственно в переменной типа `char`.

Чтобы обрабатывать отрицательные значения ключа для процесса расшифровки, нам нужно внести изменения в логику кода внутри блока `if`. Мы должны проверить, не будет ли *вычитание* значения (или добавление отрицательного ключа) сдвигать букву за начало алфавита (или меньше, чем `'A'`). Если это произойдет, мы должны сдвинуть букву назад в диапазон от `A` до `Z`, прибавив `26`. Добавьте следующий код в инструкцию `if` для прописных букв:

```
if (input >= 'A' && input <= 'Z')
{
    input += key;
    if (input > 'Z')
    input -= 26;
    if (input < 'A')
    input += 26;
}
```

При вводе отрицательного ключа мы проверяем, сдвигаемся ли мы к символам, идущим до `A`, и прибавляем `26`, чтобы вернуться назад к концу алфавита.

Не забудьте сделать то же самое для строчных букв внутри следующей инструкции `else-if`:

```
else if (input >= 'a' && input <= 'z')
{
    input += key;
    if (input > 'z')
    input -= 26;
    if (input < 'a')
    input += 26;
}
```

После этих изменений вы сможете запустить программу для шифрования и расшифровки сообщений с помощью собственного настраиваемого ключа. Ваша программа должна работать следующим образом:

```
Enter a message to encode or decode:
You've written a really cool app in Java!
Enter a secret key (-25 to 25):
7
Fvb'cl dypaalu h ylhssf jvvs hww pu Qhch!
Enter a message to encode or decode:
Fvb'cl dypaalu h ylhssf jvvs hww pu Qhch!
Enter a secret key (-25 to 25):
-7
You've written a really cool app in Java!
```

Запустив программу в первый раз, мы использовали ключ, равный 7. Чтобы расшифровать секретное сообщение, мы запускаем программу с ключевым значением –7. Таким образом, мы получаем исходное сообщение.

Попробуйте сами!

Шифрование цифр

Итак, у нас есть приложение, шифрующее буквы. Однако если нам понадобится зашифровать сообщение, подобное следующему, все цифры остаются незашифрованными:

```
Enter a message to encode or decode:
Meet me at the arcade at 5pm.
Enter a secret key (-25 to 25):
```

```
8
Ummb um ib bpm izkilm ib 5xu.
```

Обратите внимание, что число `5` в `5pm` остается `5` и после зашифровки. Если мы хотим шифровать числа, как и буквы, нам нужно добавить еще один раздел в цикл `for`. Нам нужно проверить, попадает ли символ в интервал между `0` и `9`, а затем зашифровать его как другую цифру. Мы также должны помнить, что необходимо возвращаться назад к началу или концу набора из 10 цифр. Нам придется обрабатывать это иначе, чем закольцовывание в наборе из 26 букв, так как теперь мы имеем дело с числами.

Во-первых, давайте добавим еще одну инструкцию `else-if` сразу после блока `else-if`, который обрабатывает строчные буквы:

```
else if (input >= 'a' && input <= 'z')
{
    input += key;
    if (input > 'z')
    input -= 26;
    if (input < 'a')
    input += 26;
}
else if (input >= '0' && input <= '9')
output += input;
```

Эта часть похожа на предыдущие два условия `if`, за исключением того, что мы используем в качестве диапазона цифры от `'0'` до `'9'` вместо букв от `'A'` до `'Z'`.

Следующая строка внутри фигурной скобки инструкции `if` выглядит несколько иначе:

```
else if (input >= '0' && input <= '9')
{
    input += (keyVal % 10);
```

Во-первых, мы используем *целочисленную* версию секретного ключа `keyVal`, который пользователь ввел ранее. Во-вторых, мы использовали оператор деления по модулю (`%`), чтобы определить значение сдвига для цифр в диапазоне от `−10` до `+10`.

Кроме того, нам нужно проверить, не вышли ли мы после прибавления значения `keyVal` за пределы цифрового ряда от `0` до `9`, точно так же,

как мы проверяли, попадает ли буква в интервал от A до Z. Но вместо того, чтобы вычитать 26 для возвращения назад к началу алфавита, нам нужно вычесть 10, чтобы остаться в интервале от 0 до 9:

```
else if (input >= '0' && input <= '9')
            {
                input += (keyVal % 10);
                if (input > '9')
                input -= 10;
                if (input < '0')
                input += 10;
            }
            output += input;
```

Если при шифровании цифры мы получаем символ больше 9, мы вычитаем 10. А если при расшифровке получается символ меньше 0, прибавляем 10.

Теперь мы можем шифровать в наших сообщениях как буквы, так и числа:

```
Enter a message to encode or decode:
Meet me at the arcade at 5pm and bring $2 to play Pac-Man:)
Enter a secret key (-25 to 25):
7
Tlla tl ha aol hyjhkl ha 2wt huk iypun $9 av wshf Whj-Thu:)
```

Цифра 5 в 5pm правильно сдвинута на семь и закольцовывается до 2wt в зашифрованном сообщении. 2 в $2 сдвинута на семь до $9. Вы можете протестировать предыдущее секретное сообщение, расшифровав его с ключом −7. В результате вы должны получить исходное открытое сообщение, включая цифры.

Полная консольная версия приложения «Секретные сообщения» представлена в листинге 6.4.

```
import java.util.Scanner;
public class SecretMessages {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter a message to encode or decode:");
        String message = scan.nextLine();
        String output = "";
```

```
        System.out.println("Enter a secret key (-25 to 25):");
        int keyVal = Integer.parseInt(scan.nextLine());
        char key = (char) keyVal;
        for (int x = 0; x < message.length(); x++) {
            char input = message.charAt(x);
            if (input >= 'A' && input <= 'Z')
            {
                input += key;
                if (input > 'Z')
                    input -= 26;
                if (input < 'A')
                    input += 26;
            }
            else if (input >= 'a' && input <= 'z')
            {
                input += key;
                if (input > 'z')
                    input -= 26;
                if (input < 'a')
                    input += 26;
            }
            else if (input >= '0' && input <= '9')
            {
                input += (keyVal% 10);
                if (input > '9')
                    input -= 10;
                if (input < '0')
                    input += 10;
            }
            output += input;
        }
        System.out.println(output);
        scan.close();
    }
}
```

Листинг 6.4. Готовая консольная версия приложения «Секретные сообщения»

Приложение «Секретные сообщения» может стать интересным способом обмена зашифрованными сообщениями с друзьями. Но у некоторых ваших друзей может не быть установленных на компьютере

среды разработки Eclipse или комплекта разработчика приложений на языке Java (JDK). Разве не хотелось бы иметь возможность переписываться секретными сообщениями и в этом случае? В следующем разделе мы увидим, как запустить приложение на языке Java из оболочки командной строки, не открывая программу Eclipse.

Запуск консольных приложений без среды разработки Eclipse

Мы уже создали два консольных приложения, однако мы всегда запускали их из программы Eclipse. Среда разработки Eclipse предоставляет удобный эмулятор консоли так, что мы можем видеть, как будет выглядеть консольное приложение. Но что, если мы хотим запустить приложение из реальной оболочки командной строки, например, Command Prompt в операционной системе Windows или Terminal (Терминал) в macOS? Или что, если мы хотим отправить одно из наших консольных приложений другу, у которого на компьютере не установлена программа Eclipse?

На нашу удачу, у многих людей часто установлена среда выполнения для языка Java (Java Runtime Environment, JRE).

Поиск папки рабочего пространства

Чтобы запустить приложение, которое вы написали и скомпилировали в среде разработки Eclipse, прежде всего, необходимо найти папку *рабочего пространства* программы Eclipse. Откройте ее в Проводнике файлов (или программе Finder), как показано на рис. 6.7.

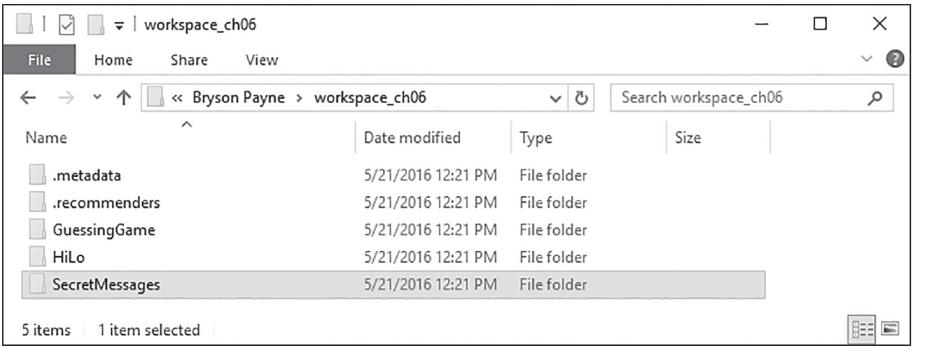


Рис. 6.7. Внутри моего каталога рабочего пространства содержатся папки для каждого из созданных проектов, а также некоторые стандартные папки программы Eclipse

Зайдите в папку рабочего пространства и откройте папку проекта *SecretMessages*. В ней вы увидите несколько файлов и папок, как показано на рис. 6.8. В папке *src* находятся файлы вашего исходного кода (с расширением.j*ava*), а в папке *bin* — скомпилированная версия вашего приложения (с расширением.c*lass*).

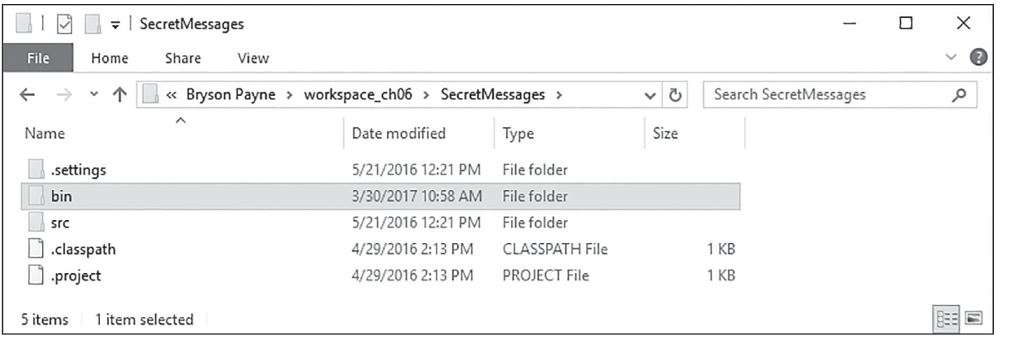


Рис. 6.8. Найдите папку *bin* внутри папки проекта *SecretMessages*

Запуск оболочки командной строки в операционной системе Windows

Затем откройте окно оболочки командной строки.

В операционной системе Windows вы можете сделать это, нажав на панели задач кнопку поиска в виде лупы, затем перейдя к панели поиска, набрав **cmd** (сокращение от «*command*») и нажав клавишу **Enter**. Также вы можете найти ярлык «Командная строка» в меню Пуск.

В операционной системе macOS используйте функцию Spotlight для поиска приложения Terminal (Терминал) или в окне программы Finder перейдите к каталогу Applications/Utilities (Приложения/Утилиты) и щелкните мышью по значку **Terminal** (Терминал). В операционной системе Linux запустите приложение Terminal (Терминал) или используйте средства поиска для нахождения оболочки командной строки.

Нам нужно перейти в описанную ранее папку *bin* с помощью оболочки командной строки. Напечатайте `cd` и пробел (от англ. change directory — изменить каталог. Далее, чтобы сменить каталог на папку *bin* для вашего приложения, перетащите папку *bin* из окна Проводника или программы Finder в окно оболочки командной строки, как показано на рис. 6.9.

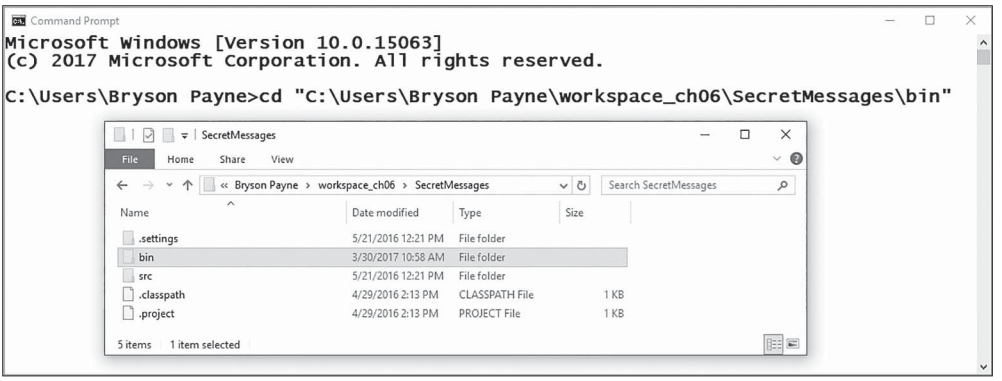


Рис. 6.9. Вы можете перетащить папку *bin* из окна файлового менеджера в окно оболочки командной строки, чтобы вставить путь к каталогу

Обратите внимание, что после команды `cd` будет указан полный путь к папке, который выглядит примерно следующим образом (ваш каталог *рабочее_пространство/SecretMessages/bin* будет расположен, разумеется, по другому пути):

```
cd "C:\Users\Bryson Payne\workspace_ch06\SecretMessages\bin"
```

В операционной системе macOS команда будет выглядеть так:

```
cd /Users/BrysonPayne/Desktop/workspace_ch06/SecretMessages/bin
```

Нажмите клавишу **Enter** (↵) после ввода команды, и оболочка командной строки изменит приглашение, показывающее, что вы теперь находитесь в папке *bin*. В операционной системе Windows это выглядит следующим образом:

```
C:\Users\Bryson Payne\workspace_ch06\SecretMessages\bin>_
```

Теперь мы находимся в папке *bin*, где находится скомпилированный файл *SecretMessages.class*. Чтобы запустить программу, содержащуюся в байткодовом файле *SecretMessages.class*, введите следующую команду:

```
C:\Users\Bryson Payne\workspace_ch06\SecretMessages\bin>java SecretMessages
```

В оболочке командной строки имеют значение орфография, прописные буквы и пробелы, поэтому вам нужно будет точно указать имя, которое вы дали вашему классу на языке Java. Если вы

правильно напишете команду, приложение должно запуститься, и вы сможете протестировать его с помощью сообщения и секретного ключа, например, так:

```
C:\Users\Bryson Payne\workspace_ch06\SecretMessages\bin>java SecretMessages
Enter a message to encode or decode:
I'm running my app directly from the command line terminal!
Enter a secret key (-25 to 25):
12
U'y dgzzuzs yk mbb pudqofxk rday ftq oayymzp xuzq fqdyuzmx!
```

Вы можете снова запустить приложение, заново набрав строку **`java SecretMessages`** или нажав на клавиатуре клавишу ↑ и **Enter**. На рис. 6.10 показано несколько запусков программы, шифрующих и расшифровывающих сообщение, в оболочке командной строки операционной системы Windows.

Рис. 6.10. Научившись запускать из командной строки приложение на языке Java, вы сможете зашифровывать и расшифровывать сообщения на любом компьютере с установленной средой выполнения Java вне зависимости от наличия установленной программы Eclipse

Научившись запускать приложение на своем компьютере, вы сможете поделиться файлом *SecretMessages.class* с друзьями и переписываться с ними зашифрованными сообщениями. Если на их компьютере установлена Java (JDK или JRE), все, что им нужно сделать, это запустить программу оболочки командной строки, перейти в папку, содержащую файл *SecretMessages.class*, и запустить команду `java SecretMessages`. Чтобы обмениваться сообщениями, просто согласуйте значение ключа. Можно каждый раз использовать

разные значения ключа! Однако не забывайте, что это просто забавное приложение — любой человек, владеющий этой программой или затративший некоторое время, сможет взломать простой шифр Цезаря. В главе 7 мы рассмотрим, как легко это делается.

Что вы изучили

Приложение «Секретные сообщения» — это интересный способ погрузиться в мир манипулирования символами и текстовыми строками на языке программирования Java. В этой главе вы узнали следующее:

- использование шифрования Цезаря для шифрования и расшифровки простых сообщений;
- использование типа данных `char` для хранения одиночных символов в кодировке Юникод;
- получение отдельного символа в строке с помощью метода `charAt()`;
- доступ к определенному местоположению в строке с использованием индекса или номера позиции символа в строке;
- соединение строк и символов с помощью оператора +;
- прохождение по строке с помощью цикла `for`, используя функцию `length()` для получения количества символов в строке;
- представление символов и других данных в памяти компьютера числовыми значениями;
- запуск консольных приложений непосредственно из оболочки командной строки без среды разработки Eclipse.

Дополнительные задачи

Попробуйте выполнить следующие задания. Таким образом, вы повторите пройденный материал, попрактикуетесь в программировании и получите новые навыки. Если у вас возникнут проблемы, вы можете загрузить примеры решений на сайте по адресу **https://eksmo.ru/files/learn_java_easy_way.zip**.

Задача № 1: Вложенный цикл

Мы создали забавное приложение для шифрования-расшифровки сообщений, которое позволит нам отправлять и получать секретные тексты, электронные письма, твиты и так далее. В качестве первой дополнительной задачи добавьте в приложение «Секретные сообщения» цикл, который позволит вам зашифровывать и расшифровывать сообщения любое количество раз.

Вы можете постоянно использовать один и тот же ключ либо каждый раз запрашивать у пользователя новый. Если вы просите пользователя вводить секретный ключ каждый раз в начале цикла, можно разрешить ему шифровать и расшифровывать сообщения одно за другим (например, вводить `8`, для шифрования сообщения другу и `-8` для расшифровывания сообщение от него).

Один из способов сделать это — предложить пользователю ввести новое сообщение для шифрования-расшифровки или нажать клавишу **Enter** для выхода. Затем используйте выражение `if` для проверки, ввел ли пользователь сообщение или просто нажал клавишу **Enter**. Пробуйте сделать это самостоятельно.

Значение `message.length()` будет больше нуля, если пользователь что-то напечатал.

Задача № 2: Переворачивание и шифрование

Давайте сделаем так, чтобы наши сообщения было еще сложнее расшифровать. Для этого сначала перевернем сообщение, расставив все символы в обратном порядке, а затем зашифруем его с помощью шифра Цезаря. Эта версия приложения с двойным кодированием объединит подход переворачивания сообщений первой версии (листинг 6.1) с возможностями шифрования последней (листинг 6.4).

Переворачивание сообщения — еще один пример симметричного шифрования. Сделав его первый раз, мы получим закодированное сообщение; выполнив его снова, получим исходное. Объединение переворачивания строк со сдвигом при шифровании методом Цезаря не сделает программу сложнее, но ваши сообщения станут еще более запутанными для перехватившего их злоумышленника.

Убедитесь, что ваше приложение правильно зашифровывает и расшифровывает сообщения, и не забудьте, что вашим друзьям

понадобится новая версия программы, чтобы расшифровать ваши новые, дважды закодированные сообщения. Наслаждайтесь!

Задача № 3: Безопасная обработка ключа с помощью конструкции try-catch

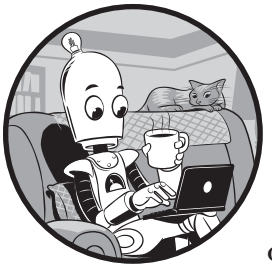
Последнее усовершенствование, которое стоило бы внести в приложение «Секретные сообщения», — это безопасная обработка некорректного ввода значения ключа. Помните конструкции `try-catch`?

Добавление блоков `try-catch` в это приложение может оказаться непростым. Помимо добавления кода обработки исключений, чтобы пользователь не мог вызвать остановку программы, введя некорректный ключ (например, введя текст вместо числа при вводе значения ключа), вам нужно подумать о том, что приложение должно делать, когда вводится некорректный ключ. Должно ли оно сообщить пользователю, что введенный ключ некорректен, и вместо него использовать предварительно определенный ключ (допустим, 13)? Или следует повторить запрос в цикле, пока пользователь не введет корректный ключ?

Попробуйте запрограммировать оба варианта и выберите тот, который вам больше нравится!

Глава 7

СОЗДАНИЕ РАСШИРЕННОГО ГРАФИЧЕСКОГО ИНТЕРФЕЙСА И ОБМЕН ДАННЫМИ С ДРУГИМИ ПРИЛОЖЕНИЯМИ



Для версии приложения «Секретные сообщения» с графическим интерфейсом для компьютера мы сделаем две большие текстовые области, которые позволят пользователю копировать и вставлять длинные сообщения в графическом интерфейсе. В конце главы мы добавим ползунковый регулятор для выбора ключа, как показано на рис. 7.1. Это очень упростит взлом сообщений, зашифрованных шифром Цезаря, даже если ключ неизвестен.

Эта версия приложения будет комфортнее для пользователя по сравнению с консольным приложением. При этом алгоритм для шифрования и расшифровки сообщений останется неизменным, что позволит нам повторно использовать код из главы 6.

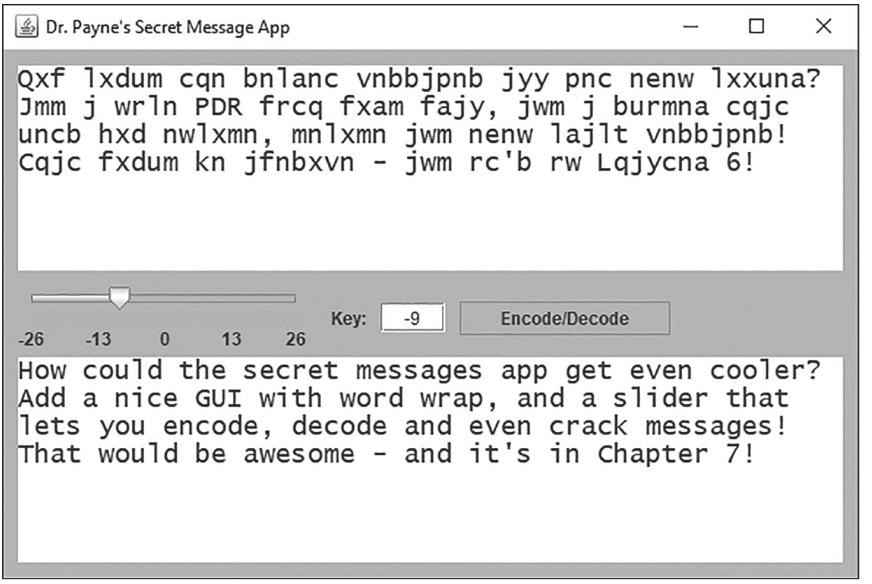


Рис. 7.1. Приложение шифрования-расшифровки секретных сообщений, которое мы сделаем в этой главе

Настройка проекта приложения «Секретные сообщения» с графическим интерфейсом

Откройте среду разработки Eclipse и создайте новый проект на языке Java, выбрав команду меню **File** ⇒ **New** ⇒ **Java Project** (Файл ⇒ Новый ⇒ Проект Java). Назовите проект **`SecretMessagesGUI`** и нажмите кнопку **Finish** (Готово). Закройте все открытые файлы, разверните папку проекта *SecretMessagesGUI* на панели **Package Explorer** (Обозреватель пакетов) и откройте папку *src*. Щелкните правой кнопкой мыши по папке *src* и выберите пункт **New** ⇒ **Class** (Создать ⇒ Класс), чтобы создать новый файл исходного кода на языке Java. Назовите этот файл тоже **`SecretMessagesGUI`**.

Мы снова будем использовать инструментарий Swing, поэтому в диалоговом окне **New Java Class** (Новый класс Java) измените суперкласс на `javax.swing.JFrame` и установите флажок, чтобы был создан метод `main()`.

Нажмите кнопку **Finish** (Готово), и вы увидите знакомый каркас программы для файла *SecretMessagesGUI.java*. Щелкните правой кнопкой мыши по файлу *SecretMessagesGUI.java* на панели **Package Explorer** (Обозреватель пакетов) и выберите пункт **Open With** ⇒ **WindowBuilder Editor** (Открыть в ⇒ Редактор WindowBuilder)

в контекстном меню, чтобы начать создание графического интерфейса для приложения «Секретные сообщения».

Проектирование компонентов графического интерфейса и присвоение им имен

Откройте вкладку **Design** (Конструктор) редактора WindowBuilder. Разверните компонент `javax.swing.JFrame` на панели **Components** (Компоненты) и выберите пункт `getContentPane()`. Теперь перейдите на панель **Properties** (Свойства) и присвойте свойству **Layout** (Макет) значение **Absolute layout** (Абсолютный макет), как показано на рис. 7.2. Это позволит нам позиционировать компоненты с точностью до пикселя.

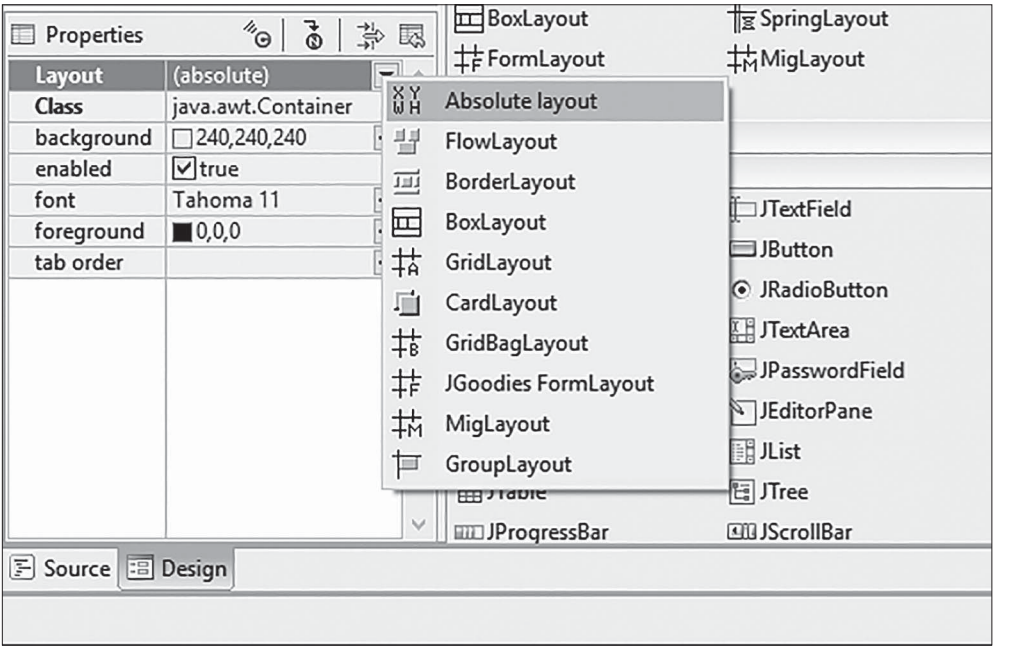


Рис. 7.2. Присвойте свойству **Layout** значение **Absolute layout** перед тем, как размещать компоненты графического интерфейса

На панели **Components** (Компоненты) выберите пункт `javax.swing.JFrame`. Затем на панели **Properties** (Свойства) присвойте свойству `defaultCloseOperation` значение `EXIT_ON_CLOSE`. Кроме того, присвойте свойству `title` значение ***Ваше имя*** **Secret Message App**, как показано на рис. 7.3.

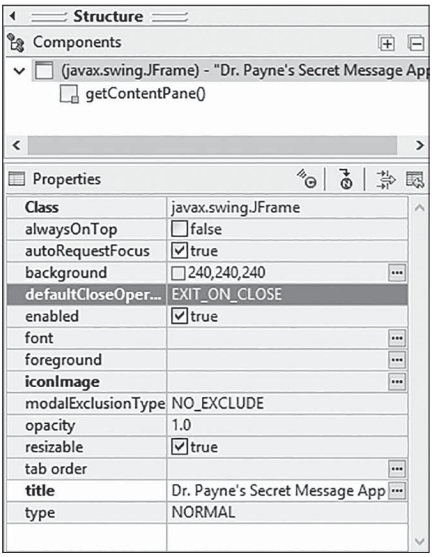


Рис. 7.3. Измените значения свойств `defaultCloseOperation` и `title`

Давайте сделаем графический интерфейс крупнее, чтобы с удобством зашифровывать и расшифровывать длинные сообщения. Выбрав на панели **Components** (Компоненты) компонент `javax.swing.JFrame`, щелкните мышью по внешней границе окна в предварительном просмотре проекта. Щелкните мышью по маленькому черному квадрату изменения размера в правом нижнем углу окна и, удерживая нажатой кнопку мыши, измените размер фрейма, чтобы он составлял 600 × 400 пикселей, как показано на рис. 7.4.

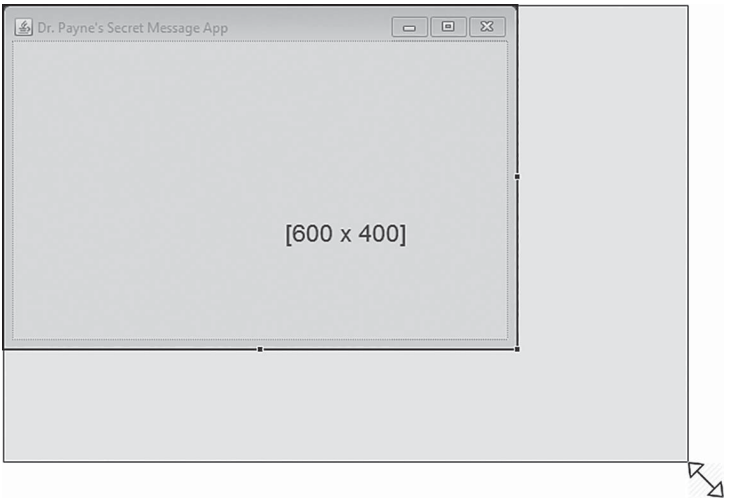


Рис. 7.4. Изменение размера объекта `JFrame`, чтобы можно было шифровать большие сообщения

Теперь мы готовы приступить к размещению компонентов графического интерфейса. В версии с графическим интерфейсом пользователя мы хотим иметь возможность шифровать и расшифровывать длинные сообщения, поэтому первый компонент, который мы разместим, — это текстовая область `JTextArea`. Как и поле `JTextField`, текстовые области позволяют вводить текст, но области `JTextArea` могут отображать несколько строк текста и даже поддерживают перенос по словам.

Чтобы вставить первый объект `JTextArea` для ввода сообщения, на панели **Palette** (Панель элементов) во вкладке **Components** (Компоненты) выберите пункт `JTextArea`. Затем поместите его в верхней части фрейма `JFrame` в окне предварительного просмотра. Сделайте так, чтобы объект `JTextArea` занял примерно одну треть высоты области содержимого, с небольшим промежутком между текстовой областью `JTextArea` и краем рамки. Текстовая область `JTextArea` должна составлять примерно 564 пикселей в ширину и 140 пикселей в высоту. Затем на панели **Properties** (Свойства) присвойте свойству `Variable` значение `txtIn`.

Чтобы создать вторую текстовую область `JTextArea` для выводимого сообщения, щелкните правой кнопкой мыши по элементу `txtIn` в окне панели **Preview** (Предварительный просмотр), выберите **Copy (Копировать)**, щелкните правой кнопкой мыши в области содержимого и выберите команду **Paste** (Вставить) в контекстном меню. Поместите новую текстовую область `JTextArea` рядом с нижним краем фрейма. Задайте ей имя, присвоив ее свойству `Variable` значение `txtOut`. У вас должно получиться два компонента `JTextArea`, как показано на рис. 7.5.

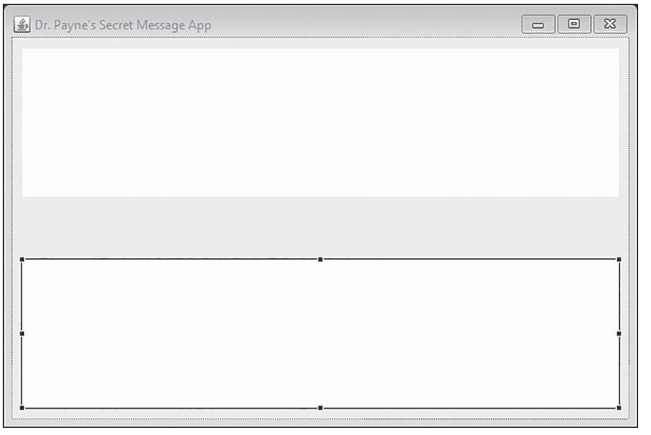


Рис. 7.5. Для создания второго объекта `JTextArea` скопируйте и вставьте объект `txtIn`. Второй объект назовите `txtOut`

Теперь добавим текстовое поле для ввода ключа шифрования. Выберите пункт `JTextField` на вкладке **Components** (Компоненты) панели **Palette** (Панель элементов), а затем щелкните мышью возле центра области содержимого и поместите текстовое поле `JTextField`. Присвойте свойству `Variable` текстового поля `JTextField` значение `txtKey` и уменьшите размер поля примерно вдвое.

Добавьте слева от `txtKey` метку `JLabel`, присвойте ее свойству `text` значение **Key:**. Присвойте свойству `horizontalAlignment` значение `RIGHT`. Наконец, добавьте кнопку `JButton` справа от `txtKey` и присвойте ее свойству `text` значение **Encode/Decode**. Увеличьте ширину кнопки, чтобы на ней поместился весь текст. Интерфейс должен выглядеть так, как показано на рис. 7.6. Чтобы просмотреть интерфейс без запуска приложения, нажмите маленькую кнопку **Test GUI** (Проверить графический интерфейс) над панелью **Palette** (Панель элементов). На рис. 7.6 эта кнопка обведена кружком.

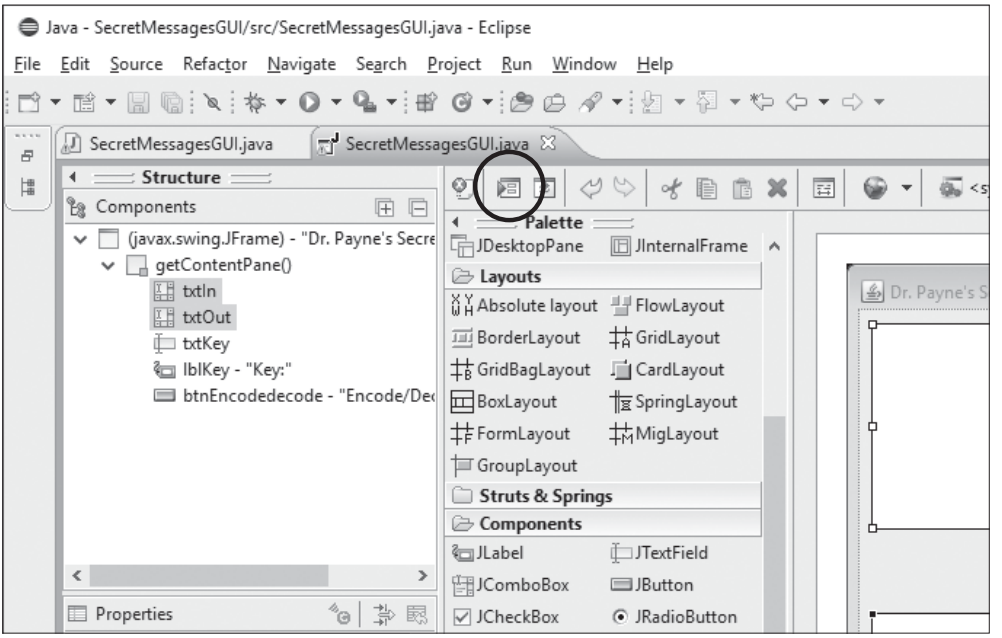


Рис. 7.6. Протестируйте внешний вид графического интерфейса, нажав кнопку **Test GUI** над панелью **Palette**

Мы подготовили графический интерфейс пользователя. Пора заняться программированием.

Написание кода приложения «Секретные сообщения» с графическим интерфейсом

Вернитесь к коду, перейдя на вкладку **Source** (Исходный код) в левом нижнем углу окна конструктора. Вы увидите, что среда разработки Eclipse добавила весь код графического интерфейса в файл *SecretMessagesGUI.java*. Чтобы подключить графический интерфейс к приложению, нам нужно добавить объявления для двух переменных `JTextArea` в верхней части класса `SecretMessagesGUI`. Программа Eclipse знает, что компонентам `JTextField` обычно нужны обработчики событий для работы с вводом пользователя, поэтому она помещает текстовое поле `txtKey` в начало класса. Теперь мы должны сделать то же самое с двумя текстовыми областями `JTextArea` — `txtIn` и `txtOut`. Добавьте две нижние строки, как показано в листинге ниже:

```
public class SecretMessagesGUI extends JFrame {
    private JTextField txtKey;
    private JTextArea txtIn;
    private JTextArea txtOut;
```

После объявления переменных JTextArea в верхней части файла нам необходимо изменить код внутри конструктора `SecretMessagesGUI()`. Удалите тип переменной `JTextArea` в начале каждой из этих двух строк:

```
    public SecretMessagesGUI() {
        setTitle("Dr. Payne's Secret Message App");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        getContentPane().setLayout(null);
        txtIn = new JTextArea(); // Удаление JTextArea из начала строки
        txtIn.setBounds(10, 11, 564, 140);
        getContentPane().add(txtIn);
        txtOut = new JTextArea(); // Удаление JTextArea из начала строки
        txtOut.setBounds(10, 210, 564, 140);
        getContentPane().add(txtOut);
```

Теперь мы готовы написать метод, шифрующий сообщения после нажатия пользователем кнопки **Encode/Decode**.

Создание метода `encode()`

Метод `encode()`, который мы будем писать, похож на `checkGuess()` из игры «Больше-Меньше» (см. главу 3). Метод `checkGuess()` принимал число, введенное пользователем, но ничего возвращать не требовалось. Поэтому типом возвращаемого значения был `void`. В отличие от `checkGuess()`, метод `encode()` будет принимать сообщение и ключ шифрования в качестве значений, а возвращать будет зашифрованное сообщение.

Методы возвращают значения, чтобы программа могла эти значения в дальнейшем использовать. Мы хотим, чтобы метод `encode()` принимал сообщение и ключ, а затем запускал некоторый код, зашифровывающий сообщение, которое наша программа будет выводить пользователю. При объявлении метода, который возвращает какое-либо значение, мы должны указать тип данных возвращаемого значения перед именем метода. Информация, которую мы хотим вернуть из метода `encode()`, является строковой переменной. Таким образом, метод `encode()` будет объявлен как `public String encode()`.

Нам нужно сообщить компилятору Java, какую информацию мы хотим передать методу `encode()`; эту информацию называют *параметрами* метода. Чтобы объявить параметры метода, мы помещаем их в круглые скобки, следующие за именем метода, сначала тип данных, а затем имя параметра. Несколько параметров разделяются запятыми. Объявление метода `encode()` будет выглядеть следующим образом:

```
    public String encode(String message, int keyVal)
```

Не забудьте добавить фигурные скобки для тела метода и поместите метод `encode()` прямо под объявлениями двух переменных `JTextArea` и выше конструктора `SecretMessagesGUI()`:

```
public class SecretMessagesGUI extends JFrame {
    private JTextField txtKey;
    private JTextArea txtIn;
    private JTextArea txtOut;
    public String encode(String message, int keyVal) {
    }
    public SecretMessagesGUI() {
```

Среда разработки Eclipse подчеркнет метод `encode()` красной линией, поскольку он должен возвращать значение, но фактически этого не делает, потому что мы еще не успели написать код внутри тела метода. При этом мы будем повторно использовать некоторые фрагменты кода из главы 6.

Объявите переменную типа `String` с именем `output` и присвойте ей значение, равное пустой строке. После этого добавьте строку кода с инструкцией `return`, которая будет возвращать переменную `output` как результат метода `encode()`:

```
public String encode(String message, int keyVal) {
    String output = "";
    return output;
}
```

Поскольку при объявлении метода `encode()` мы указали в качестве типа возвращаемого значения `String`, а в теле метода мы возвращаем переменную `output`, типом которой *является* `String`, программа Eclipse уберет красное подчеркивание, тем самым сообщая нам, что мы решили проблему отсутствия возвращаемого значения.

Давайте повторно используем фрагмент кода из консольной версии приложения. Откройте проект *SecretMessages*, сделанный в главе 6, на панели **Project Explorer** (Обозреватель проекта). Нам надо скопировать код, начинающийся с `char key` и заканчивающийся закрывающейся скобкой цикла `for`, прямо перед `System.out.println(output);` ближе к концу программы. Это часть приложения, в котором сообщение зашифровывается с использованием алгоритма шифра Цезаря.

Вставьте код, скопированный из файла *SecretMessages.java*, в файл *SecretMessagesGUI.java*, внутрь метода `encode()`, между двумя только что добавленными строками. Готовый метод `encode()` выглядит следующим образом:

```
public String encode(String message, int keyVal) {
    String output = "";
    char key = (char) keyVal;
    for (int x = 0; x < message.length(); x++) {
        char input = message.charAt(x);
        if (input >= 'A' && input <= 'Z')
        {
```

```
                input += key;
                if (input > 'Z')
                    input -= 26;
                if (input < 'A')
                    input += 26;
            }
            else if (input >= 'a' && input <= 'z')
            {
                input += key;
                if (input > 'z')
                    input -= 26;
                if (input < 'a')
                    input += 26;
            }
            else if (input >= '0' && input <= '9')
            {
                input += (keyVal% 10);
                if (input > '9')
                    input -= 10;
                if (input < '0')
                    input += 10;
            }
            output += input;
        }
        return output;
    }
```

Поскольку мы называли наши переменные последовательно, мы можем повторно использовать код шифрования алгоритма шифра Цезаря. Последовательное присвоение имен является хорошей привычкой, и хорошо написанный код на языке программирования Java можно повторно использовать на разных платформах.

Написание обработчика событий для кнопки Encode/Decode

Мы хотим, чтобы пользователь вводил в графическом интерфейсе исходное сообщение и значение ключа, которые мы передадим в метод `encode()`. Кроме того, мы хотим, чтобы получившееся сообщение отображалось после нажатия кнопки **Encode/Decode**. Давайте напишем код для обработки события щелчка по кнопке.

Перейдите в режим конструктора в *SecretMessagesGUI.java* и дважды щелкните мышью по кнопке **Encode/Decode**. Среда разработки Eclipse переключится в режим кодирования и вставит код обработчика события `actionPerformed()` для кнопки **Encode/Decode**, например приведенный здесь фрагмент кода:

```
JButton btnEncodedecode = new JButton("Encode/Decode");
btnEncodedecode.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
    }
});
```

Это анонимный внутренний класс, подобный тому, с которым мы сталкивались в главе 3. Теперь нам нужно добавить код внутри метода `actionPerformed()`, который сообщит компилятору языка Java, что нужно делать после щелчка по кнопке.

Мы хотим, чтобы по щелчку по кнопке **Encode/Decode** выполнялись следующие шаги.

1. Получить сообщение из текстовой области `txtIn`.
2. Получить ключ из поля `txtKey`.
3. Зашифровать сообщения с помощью ключа.
4. Вывести зашифрованного сообщения в текстовую область `txtOut`.

Подумайте, как можно выполнить каждый из этих шагов. Попробуйте реализовать свое решение самостоятельно, прежде чем двигаться дальше, а затем сравните его с моим вариантом решения.

Для реализации первого шага необходимо определить переменную `message` типа `String`, которая получает текст из области `txtIn`. Подобно текстовому полю `JTextField`, объекты класса `JTextArea` имеют метод `getText()`:

```
public void actionPerformed(ActionEvent arg0) {
    String message = txtIn.getText();
```

На втором шаге мы можем получить ключ из поля `txtKey` с помощью метода `Integer.parseInt()`, который извлекает целочисленное значение из текстовой строки. Затем сохраним полученное целое число в переменной `key`:

```
String message = txtIn.getText();
int key = Integer.parseInt(txtKey.getText());
```

Чтобы зашифровать сообщение на третьем шаге, нам просто нужно вызвать метод `encode()` и передать ему два аргумента. *Аргументы* — это значения, которые мы передаем методу в качестве его параметров. Помните, что мы определили метод `encode()` с двумя параметрами, поэтому нам нужно передать в метод `encode()` два аргумента, которыми будут переменные `message` и `key`.

Метод `encode()` вызывается с помощью команды `encode(message, key)`. По завершении работы он вернет зашифрованное сообщение, то есть `output`. Обратите внимание, что метод возвращает само значение, а не переменную, в которой это значение хранится. Это означает, что зашифрованное сообщение хранится в переменной `output`, но когда метод `encode()` возвращает эту переменную, мы фактически получаем только зашифрованную строку. Помимо этого, обратите внимание, что код и все переменные, созданные внутри метода `encode()`, не переносятся к остальной части программы, поэтому переменная `output` не существует после того, как метод `encode()` вернул ее значение. Нам нужно сохранить возвращаемое значение в новой переменной, если, конечно, мы хотим его сохранить. Чтобы наше именование оставалось согласованным, давайте назовем новую переменную также `output`:

```
String message = txtIn.getText();
int key = Integer.parseInt(txtKey.getText());
String output = encode(message, key);
```

И наконец, используем метод `setText()` для отображения переменной `output` в поле `txtOut`:

```
String message = txtIn.getText();
int key = Integer.parseInt(txtKey.getText());
String output = encode(message, key);
txtOut.setText(output);
```

Код метода `actionPerformed()` почти завершен, за исключением обработки ошибок. Мы использовали метод `Integer.`

parseInt(), который может выбрасывать исключения, если пользователь вводит некорректные данные, поэтому нам нужно добавить инструкции try-catch.

Обработка некорректного ввода и ошибок пользователя

При щелчке по кнопке вызывается код, содержащий метод Integer.parseInt(). В главе 3 мы узнали, что этот метод может выбрасывать исключения в случае ошибочного ввода пользователя. В частности, если пользователь оставит поле txtKey пустым или введет что-либо кроме целого числа в это текстовое поле, метод Integer.parseInt() завершится с ошибкой.

Нам нужно защитить программу от некорректного ввода, используя блоки try-catch для обработки исключений. Поскольку мы хотим зашифровать или расшифровывать сообщение только тогда, когда пользователь правильно ввел ключ шифрования, мы можем включить все четыре строки метода actionPerformed() внутрь блока try, добавив ключевое слово try и открывающую фигурную скобку перед первой строкой, а также закрывающую скобку после четвертой, как показано ниже:

```
public void actionPerformed(ActionEvent arg0) {
    try {
        String message = txtIn.getText();
        int key = Integer.parseInt(txtKey.getText());
        String output = encode(message, key);
        txtOut.setText(output);
    } catch (Exception ex) {
    }
}
```

Среда разработки Eclipse выделит закрывающую фигурную скобку красным цветом, так что давайте продолжим и добавим блок catch. Нам нужно сообщить компилятору языка Java, что делать, когда был осуществлен некорректный ввод. Эти шаги мы добавим в фигурные скобки после инструкции catch немного позже. Но прежде всего давайте закончим метод main(), чтобы можно было протестировать наше приложение.

Создание метода `main()` и запуск приложения

Подобно программированию игры в главе 3, нам нужно добавить в метод `main()` код, который создает экземпляр приложения «Секретные сообщения», устанавливает правильный размер окна графического интерфейса и делает интерфейс видимым для пользователя. В нижней части файла *SecretMessagesGUI.java*, найдите заглушку метода `main()`, которую сделала среда разработки Eclipse, и добавьте следующие три строки.

```
    public static void main(String[] args) {
❶       SecretMessagesGUI theApp = new SecretMessagesGUI();
❷       theApp.setSize(new java.awt.Dimension(600,400));
❸       theApp.setVisible(true);
    }
```

Строка ❶ создает объект `SecretMessagesGUI` с именем `theApp`. Ключевое слово `new`, за которым следует имя класса, запускает метод конструктора `SecretMessagesGUI()`, который настраивает все компоненты графического интерфейса пользователя.

В строке ❷ мы устанавливаем размер фрейма `JFrame` в соответствии с шириной и высотой, которые мы задали при компоновке макета, то есть размером 600 × 400 пикселей. И наконец, в строке ❸ мы устанавливаем свойству `visible` (видимость) объекта `JFrame` значение `true` (истина). После этого графический интерфейс станет виден пользователю. Сохраните эти изменения, а затем запустите приложение, чтобы проверить его работу. Результат должен быть похож на рис. 7.7.

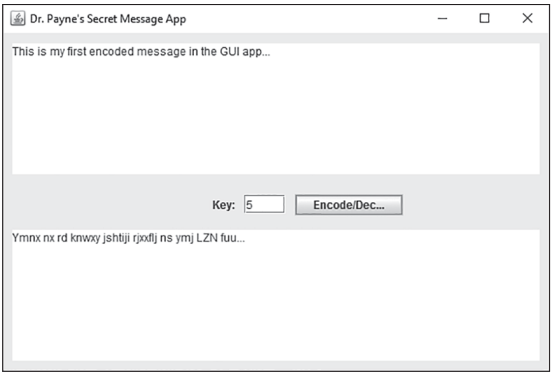


Рис. 7.7. Приложение «Секретные сообщения» пока еще не очень сложное, но оно уже может зашифровывать и расшифровывать сообщения

На данный момент полный исходный код файла *SecretMessagesGUI.java* выглядит следующим образом:

```
import javax.swing.JFrame;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.JLabel;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
public class SecretMessagesGUI extends JFrame {
    private JTextField txtKey;
    private JTextArea txtIn;
    private JTextArea txtOut;
    public String encode(String message, int keyVal) {
        String output = "";
        char key = (char) keyVal;
        for (int x = 0; x < message.length(); x++) {
            char input = message.charAt(x);
            if (input >= 'A' && input <= 'Z')
            {
                input += key;
                if (input > 'Z')
                    input -= 26;
                if (input < 'A')
                    input += 26;
            }
            else if (input >= 'a' && input <= 'z')
            {
                input += key;
                if (input > 'z')
                    input -= 26;
                if (input < 'a')
                    input += 26;
            }
            else if (input >= '0' && input <= '9')
            {
                input += (keyVal% 10);
                if (input > '9')
                    input -= 10;
```

```
            if (input < '0')
                input += 10;
        }
        output += input;
    }
    return output;
}
public SecretMessagesGUI() {
    setTitle("Dr. Payne's Secret Message App");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    getContentPane().setLayout(null);
    txtIn = new JTextArea();
    txtIn.setBounds(10, 11, 564, 140);
    getContentPane().add(txtIn);
    txtOut = new JTextArea();
    txtOut.setBounds(10, 210, 564, 140);
    getContentPane().add(txtOut);
    txtKey = new JTextField();
    txtKey.setBounds(258, 173, 44, 20);
    getContentPane().add(txtKey);
    JLabel lblKey = new JLabel("Key:");
    lblKey.setBounds(202, 176, 46, 14);
    getContentPane().add(lblKey);
    JButton btnEncodedecode = new JButton("Encode/Decode");
    btnEncodedecode.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {
            try {
                String message = txtIn.getText();
                int key = Integer.parseInt(txtKey.getText());
                String output = encode(message, key);
                txtOut.setText(output);
            } catch (Exception ex) {
        }
    }
    });
    btnEncodedecode.setBounds(312, 172, 144, 23);
    getContentPane().add(btnEncodedecode);
}
public static void main(String[] args) {
    SecretMessagesGUI theApp = new SecretMessagesGUI();
    theApp.setSize(new java.awt.Dimension(600,400));
```

```
            theApp.setVisible(true);
        }
    }
```

Эта первая версия приложения работает вполне успешно, но визуально она оставляет желать лучшего. Мы можем сделать графический интерфейс пользователя более профессиональным с помощью лишь нескольких настроек.

Улучшение графического интерфейса пользователя

Вернемся в режим конструктора и исправим проблемы, отмеченные нами в последнем разделе. Сначала щелкните мышью по кнопке **Encode/Decode** и сделайте ее немного шире. На разных компьютерах могут быть установлены по умолчанию другие шрифты и их размеры, поэтому сделайте кнопку немного шире, чтобы она всегда корректно отображалась.

Теперь давайте сделаем размер шрифта в двух текстовых областях `JTextArea` больше и удобнее для чтения. Мы будем изменять свойства шрифта в обеих текстовых областях, поэтому стоит одновременно выбрать оба компонента `txtIn` и `txtOut`, сначала щелкнув мышью по области `txtIn`, а затем, удерживая клавишу **Ctrl** (или ⌘ в операционной системе macOS), щелкнув мышью по области `txtOut`.

Выбрав обе текстовые области, найдите свойство `font` на панели **Properties** (Свойства) и щелкните мышью по трем точкам рядом с размером шрифта. Откроется диалоговое окно **Font Chooser** (Выбор шрифта), как показано на рис. 7.8.

Выберите шрифт, который вам нравится, и установите его размер 18 или больше. Придерживайтесь знакомого вам шрифта, чтобы при совместном использовании приложения с друзьями на разных компьютерах его внешний вид был примерно одинаков. Я выбрал шрифт Lucida Console, размер — 18 пунктов. Нажмите кнопку **OK**, чтобы сохранить настройки шрифта.

Теперь, давайте изменим цвет фона области содержимого, чтобы сделать его индивидуальным. Выберите область содержимого, щелкнув мышью по пункту `getContentPane()` на панели **Components** (Компоненты) в левом верхнем углу. Затем щелкните мышью по трем точкам рядом со свойством `background` на панели **Properties** (Свойства). В появившемся диалоговом окне **Color**

Chooser (Выбор цвета) выберите вкладку **Named colors** (Именованные цвета), как показано на рис. 7.9. Вы увидите широкий выбор цветов. Выберите нужный цвет и нажмите кнопку **OK**.

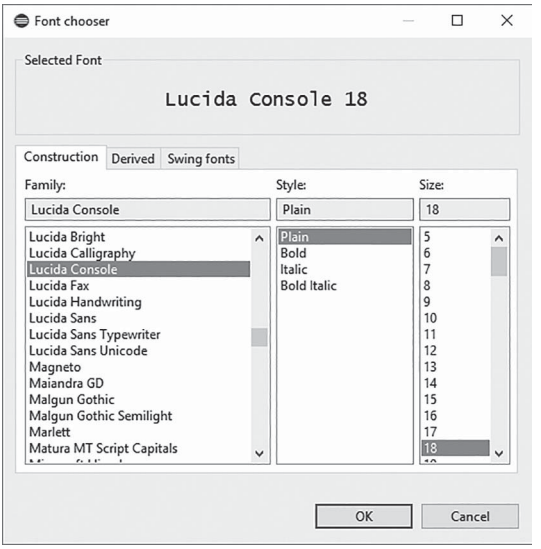


Рис. 7.8. Мы можем изменять свойства шрифта для обеих текстовых областей `JTextArea` одновременно

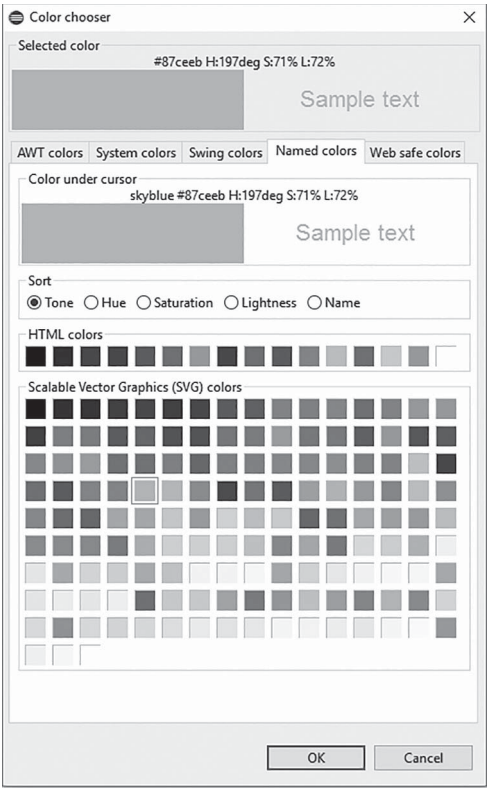


Рис. 7.9. Измените цвет фона панели содержимого и кнопку, чтобы сделать приложение более стильным

Кроме того, вы можете изменить цвет фона кнопки **Encode/Decode**. Возможно, все изменения не будут видны в предварительном просмотре проекта, но после запуска приложения вы увидите все улучшения, как показано на рис. 7.10, в том числе и новый цвет фона.

Рис. 7.10. Графический интерфейс после изменения размеров кнопки и шрифта

Существует множество других свойств, которые вы можете настроить, основываясь на ваших предпочтениях. Исследуйте каждый компонент на панели **Properties** (Свойства) и настраивайте параметры, пока приложение не будет выглядеть согласно вашему вкусу. Не забудьте остановиться и сохранить приложение, чтобы ваши изменения не пропали!

Настройка переноса строк по словам

До сих пор все сообщения, использованные в примерах, были короче одной строки. Что же произойдет, если мы наберем несколько строк текста?

Давайте попробуем ввести или вставить длинное предложение в текстовую область. Я воспользуюсь первым предложением Геттисбергской речи президента США Авраама Линкольна:

«Минуло восемьдесят семь лет, как отцы наши основали на этом континенте новую нацию, своим рождением обязанную свободе и убежденную, что все люди рождены равными».

Вы также можете воспользоваться этим предложением или составить свое собственное длинное предложение. В любом случае,

как только вы введете его, вы заметите проблему, показанную на рис. 7.11.

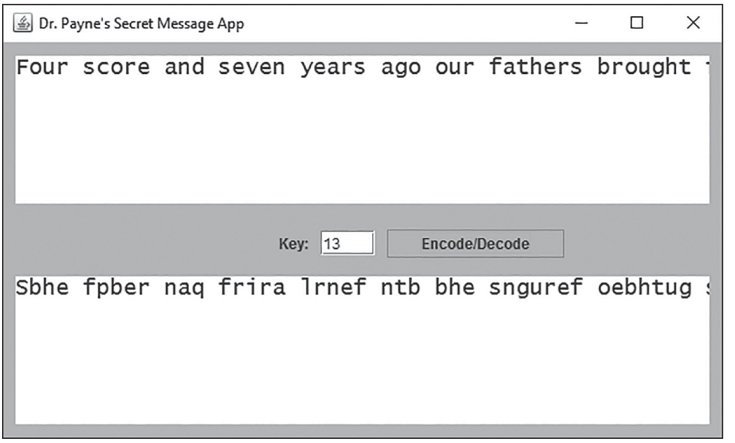


Рис. 7.11. В нашем приложении длинные сообщения еще не переносятся автоматически в конце строки

Чтобы правильно переносить длинные строки текста по словам, нужно изменить *два* свойства для каждой текстовой области на панели **Properties** (Свойства). Выделите в предварительном просмотре проекта оба объекта `JTextArea` еще раз, а затем на панели **Properties** (Свойства) установите флажки `lineWrap` и `wrapStyleWord`. Первое свойство, `lineWrap`, сообщает компилятору языка Java о необходимости переноса текста (длиной более одной строки) на следующие строки. Установка стиля переноса в `wrapStyleWord` говорит языку Java о том, что переносить строку следует в конце целого слова (а не, например, в его середине), как это делается в текстовом редакторе.

Теперь сохраните приложение и запустите его еще раз. Вставьте свое длинное предложение в первое текстовое поле, и вы должны сразу заметить улучшение. Теперь текст корректно перенесен на несколько строк, с разрывами строки после целого слова. Введите ключ и нажмите кнопку **Encode/Decode**, и в текстовой области вывода будет отображаться зашифрованный текст, так же перенесенный по словам, как показано на рис. 7.12.

Наша программа выглядит все более профессиональной. Что же произойдет, если мы осуществим неверный ввод в текстовое поле для ключа шифрования? В настоящий момент — ничего. Ранее мы добавили инструкцию `try`, но мы не реализовали инструкцию `catch`. Давайте сделаем это сейчас. Кроме того, мы

узнаем, как создавать всплывающие окна, предупреждающие пользователя об ошибках.

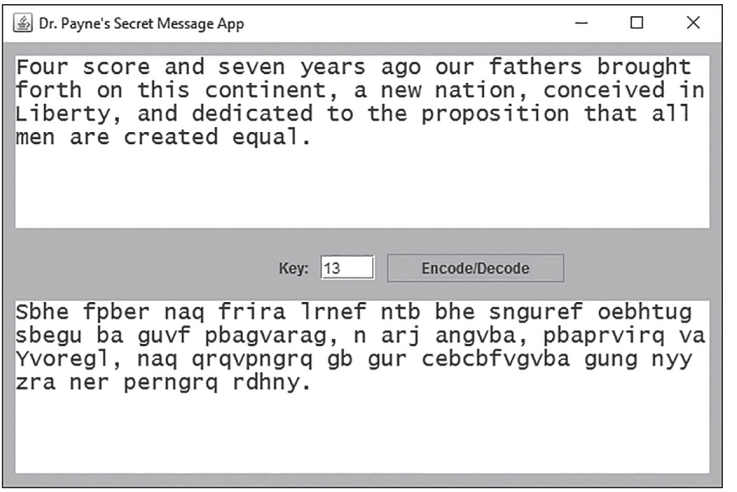


Рис. 7.12. Присвоение свойствам `lineWrap` и `wrapStyleWord` значения `true` улучшает интерфейс, позволяя переносить длинные предложения по словам

Обработка некорректного ввода и ошибок пользователя: часть 2

Ранее в этой главе мы создали блок `try` вокруг обработчика событий кнопки, но мы оставили тело блока `catch` пустым. Пришла пора наполнить его кодом.

Наша команда `catch` будет обрабатывать некорректный ввод пользователя или отсутствие ввода в текстовом поле `txtKey`. Если пользователь забыл ввести значение ключа шифрования или если введенное значение не является числом, одним из способов обработки этого является вывод сообщения, описывающего проблему, как показано на рис. 7.13.

После того как пользователь закрыл всплывающее окно, было бы удобно переместить курсор в текстовое поле `txtKey` и выделить находящийся там текст, чтобы пользователю не требовалось вручную перемещать курсор для ввода нового значения.

В инструментарии javax.`swing`, который мы использовали для создания компонентов графического интерфейса пользователя, таких как фрейм `JFrame` и текстовая область `JTextField`, есть также классы, обрабатывающие различные виды всплывающих окон. Примерами таких окон являются: *диалоговые окна ввода*,

которые позволяют пользователю вводить текст; *диалоговые окна подтверждения*, которые позволяют пользователю выбирать Yes/No/OK или Cancel, и *диалоговые окна сообщений*, которые показывают пользователю информацию во всплывающем окне.

Рис. 7.13. Появление всплывающего окна, предупреждающего пользователя об ошибке

Сообщение об ошибке, которое мы хотели бы предоставить пользователю, лучше всего отображать в диалоговом окне сообщений. В классе `javax.swing.JOptionPane` реализован метод `showMessageDialog()`, который принимает два параметра: родительское окно и отображаемое сообщение. Если наше приложение было бы больше, мы использовали бы параметр родительского окна, чтобы расположить диалоговое окно, например, в центре главного окна обработки текста. Но в этом приложении мы можем использовать ключевое слово `null`, чтобы поместить диалоговое окно по центру рабочего стола, как показано ниже:

```
JOptionPane.showMessageDialog(null,
        "Please enter a whole number value for the encryption key.");
```

Когда вызывается эта строка кода, появляется диалоговое окно с сообщением «Please enter a whole number value for the encryption key».

Чтобы добавить это всплывающее сообщение об ошибке в тело созданной ранее инструкции `catch`, перейдите в режим конструктора и дважды щелкните мышью по кнопке **Encode/Decode**. Это

быстрый способ заставить программу Eclipse напрямую перейти к исходному коду для обработчика событий нажатия кнопки, содержащего инструкцию `catch`.

После появления сообщения об ошибке мы собираемся поместить курсор в поле `txtKey` и выделить находящийся там текст, как это было сделано в приложении с графическим интерфейсом для игры «Больше-Меньше». Мы добавим всплывающее окно и те же команды `requestFocus()` и `selectAll()`, которые мы использовали в игре «Больше-Меньше» в фигурных скобках блока `catch`. Готовый код обработчика для кнопки **Encode/Decode** выглядит следующим образом:

```
btnEncodedecode.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        try {
            String message = txtIn.getText();
            int key = Integer.parseInt(txtKey.getText());
            String output = encode(message, key);
            txtOut.setText(output);
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(null,
            "Please enter a whole number value for the encryption key.");
            txtKey.requestFocus();
            txtKey.selectAll();
        }
    }
});
```

Когда вы добавляете панель `JOptionPane`, если вы не используете автозаполнение или если вы видите ошибку в панели `JOptionPane`, возможно вам стоит нажать сочетание клавиш **Ctrl+Shift+O**, чтобы правильно импортировать класс `javax.swing.JOptionPane` в верхнюю часть кода.

Запустите приложение и оставьте поле ключа пустым, либо введите в него нечисловой текст. Откроется окно с просьбой ввести корректный ключ. После этого текст в поле ключа будет выделен, так что после закрытия окна будет легко ввести целое число в это поле.

Пользователь, использующий приложение в первый раз, может не знать, что вводить в поле ключа, поэтому добавление значения

по умолчанию также будет полезной настройкой пользовательского интерфейса. Перейдите на вкладку **Design** (Конструктор), выберите текстовое поле `txtKey` и присвойте значение ключа по умолчанию свойству `text` на панели **Properties** (Свойства). Я выберу 3 в качестве ключа шифрования по умолчанию, но вы можете выбрать любое другое число. Давайте также присвоим свойству `horizontalAlignment` поля `txtKey` значение `CENTER`. Вы можете изменять другие свойства, чтобы поле `txtKey` выглядело так, как вам хотелось бы, включая цвет шрифта и стиль.

Добавление ползункового регулятора в интерфейс приложения «Секретные сообщения»

Мы добавим еще одно усовершенствование пользовательского интерфейса в приложение «Секретные сообщения» — числовой ползунковый регулятор, который позволит пользователю быстро перемещаться по значениям ключа и видеть, как меняется сообщение с каждым новым ключом.

В режиме конструктора в разделе **Components** (Компоненты) на панели **Palette** (Панель элементов) выберите компонент `JSlider`. Наведите указатель мыши на центральную часть нашего фрейма левее метки с надписью **key:** и щелкните, чтобы установить объект `JSlider` на это место, как показано на рис. 7.14. При необходимости точное расположение можно будет отрегулировать позже.

Теперь давайте настроим некоторые свойства ползункового регулятора `JSlider`. Во-первых, давайте изменим цвет его фона, чтобы он сочетался с остальной частью приложения. Выбрав новый ползунковый регулятор `JSlider` в окне предварительного просмотра, щелкните мышью по трем точкам рядом со свойством `background` на панели **Properties** (Свойства). Откроется диалоговое окно **Color Chooser** (Выбор цвета), в котором вам надо перейти на вкладку **Named colors** (Именованные цвета) и выбрать тот же цвет, который использовался для фона остальной части графического интерфейса. Нажмите кнопку **OK**, чтобы сохранить новый цвет фона для ползункового регулятора.

Затем добавим пользовательские метки, отметки на оси и значение по умолчанию для ползункового регулятора. Во-первых, включите свойство `paintLabels`, установив флажок `true`. Во-вторых,

введите максимальное (maximum) значение 26 и минимальное (minimum) значение –26, чтобы значение ключа можно было легко выбрать в корректном диапазоне.

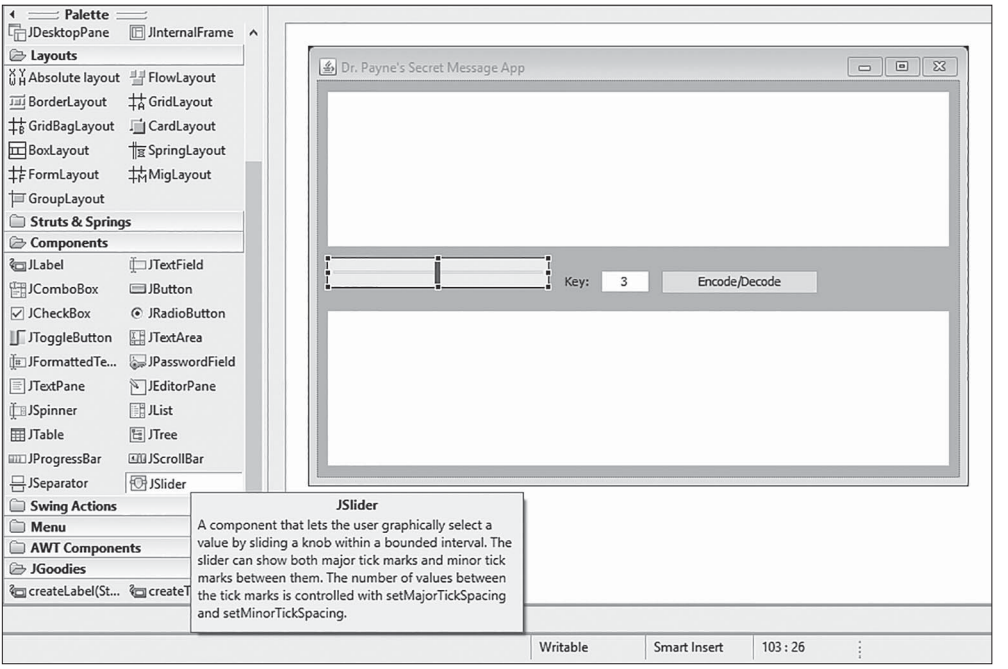


Рис. 7.14. Добавление ползункового регулятора JSlider в графический интерфейс пользователя позволяет быстро и просто пробовать разные значения ключа

Затем свойству minorTickSpacing присвойте значение 1 — это действие добавит небольшие отсечки на дорожке ползункового регулятора, тем самым помогая пользователю видеть диапазон возможных значений. Затем свойству majorTickSpacing присвойте значение 13 — это действие добавит на ползунковый регулятор числовые метки через каждые 13 значений между –26 и 26. Теперь задействуйте свойство paintTicks, установив флажок true рядом с ним. Панель **Properties** (Свойства) со всеми внесенными нами изменениями показана на рис. 7.15.

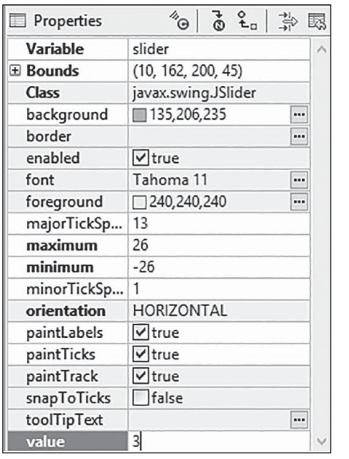


Рис. 7.15. Настройка свойств ползункового регулятора

Мы почти закончили! Измените начальное значение ползункового регулятора так, чтобы оно соответствовало установленному в поле ключа значению по умолчанию. Для этого измените свойство `value`. Я выбрал 3 в качестве значения ключа по умолчанию, поэтому я ввел 3 в свойстве `value` для ползункового регулятора. Наконец, вам нужно сместить ползунковый регулятор немного выше, чтобы были видны метки. Готовый ползунковый регулятор показан на рис. 7.16.

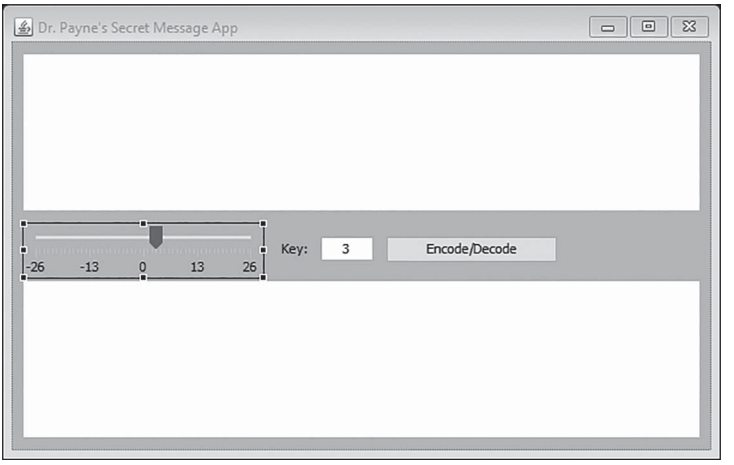


Рис. 7.16. Приложение «Секретные сообщения» с готовым ползунковым регулятором

Теперь нам нужно добавить код, заставляющий ползунковый регулятор работать, позволяя пользователю легко перемещаться между различными значениями ключа. Таким образом, скорость, как шифрования, так и расшифровки увеличится.

Взлом шифра с помощью ползункового регулятора

Мы хотим, чтобы пользователь мог изменять значение ключа шифрования, передвигая мышью ползунковый регулятор, который мы добавили в предыдущем разделе. Давайте создадим обработчик событий для прослушивания изменений состояния ползункового регулятора.

Щелкните правой кнопкой мыши по ползунковому регулятору (щелкните мышью с нажатой клавишей ^ в операционной системе macOS) в окне предварительного просмотра и выберите пункт **Add event handler ⇒ change ⇒ stateChanged** (Добавить обработчик события ⇒ change ⇒ stateChanged) в контекстном меню, как

показано на рис. 7.17. Обработчик события `stateChanged` будет работать аналогично обработчику событий `ActionPerformed` для кнопки **Encode/Decode**, но с одним исключением: он будет срабатывать каждый раз, когда пользователь, изменив положение ползункового регулятора, обновит значение ключа шифрования.

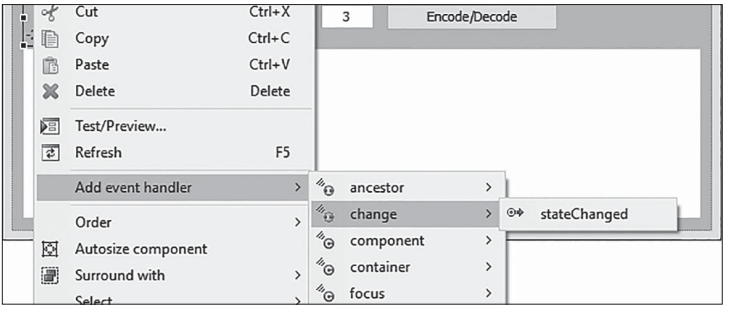


Рис. 7.17. Добавление к ползунковому регулятору обработчика событий, который будет реагировать на изменение его положения (значения)

Когда вы добавите к ползунковому регулятору обработчик события `stateChanged`, среда разработки Eclipse автоматически сгенерирует еще один анонимный внутренний класс, подобно созданному для кнопки **Encode/Decode**. Но прежде чем мы рассмотрим анонимный внутренний класс, нам нужно добавить объявление ползункового регулятора `JSlider` в начало класса рядом с объявлениями текстовых областей `JTextArea`. Чтобы сделать это, мы должны объявить экземпляр класса `JSlider`., К объявлениям в верхней части класса `SecretMessagesGUI` добавьте последнюю строку из представленных ниже:

```
public class SecretMessagesGUI extends JFrame {
    private JTextField txtKey;
    private JTextArea txtIn;
    private JTextArea txtOut;
    private JSlider slider;
```

Затем пролистайте код программы вниз до блока ползункового регулятора. Напомню, чтобы быстро найти в коде место, куда среда разработки Eclipse добавила обработчик события, можно вернуться в режим конструктора, щелкнуть правой кнопкой мыши по ползунковому регулятору и снова добавить обработчик. Теперь измените первую строку, удалив объявление типа `JSlider`, как показано ниже:

```
slider = new JSlider();
slider.addChangeListener(new ChangeListener() {
    public void stateChanged(ChangeEvent arg0) {
```

Первым действием ползункового регулятора должно быть обновление текстового поля `txtKey`, в котором отображается текущая позиция ползункового регулятора. Ее значение можно получить с помощью метода `getValue()`, а установить текст в поле `txtKey` — с помощью метода `setText()`. Объединив их вместе, мы можем написать первую строку метода `stateChanged()`:

```
    public void stateChanged(ChangeEvent arg0) {
        txtKey.setText("" + slider.getValue());
```

Метод `slider.getValue()` возвращает целочисленное значение текущей позиции ползункового регулятора. Мы прибавляем пустую строку этому к числу, чтобы превратить его в строку или текстовое значение. Тем самым значение текстового поля `txtKey` будет меняться, отображая текущее значение ползункового регулятора. Но сообщение не будет заново зашифровываться с использованием нового значение ключа. Если вы сейчас запустите приложение, вам все равно придется нажимать кнопку **Encode/Decode** для изменения сообщений.

Давайте добавим код в метод `stateChanged()`, чтобы ползунковый регулятор обновлял сообщение, то есть действовал подобно кнопке **Encode/Decode**. Мы можем скопировать код из блока `try` кнопки **Encode/Decode** и вставить его после первой строки метода `stateChanged()` для ползункового регулятора, сделав только одно изменение:

```
slider = new JSlider();
slider.addChangeListener(new ChangeListener() {
    public void stateChanged(ChangeEvent arg0) {
        txtKey.setText("" + slider.getValue());
❶           String message = txtIn.getText();
❷           int key = slider.getValue();
❸           String output = encode(message, key);
❹           txtOut.setText(output);
    }
});
```

Строки ❶, ❸ и ❹ скопированы непосредственно из обработчика событий кнопки **Encode/Decode**, но в строке ❷ мы сделали небольшое изменение. Вместо извлечения значения из текстового поля мы можем получить доступ напрямую к значению ползункового регулятора, которое является целым числом, с помощью метода `getValue()`. Это означает, что нам не нужны команды `try-catch`, поскольку мы больше не ориентируемся на значение, введенное пользователем вручную. Это удобнее для пользователя и безопаснее для программиста, поскольку использование ползункового регулятора в графическом интерфейсе устраняет потенциальную ошибку, а также упрощает использование интерфейса.

Сохраните изменения и запустите приложение. Вы увидите, что можете ввести сообщение, а затем изменять положение ползункового регулятора, тем самым пробуя различные значения ключа шифрования. Чтобы проверить возможность расшифровки, скопируйте закодированное сообщение из нижней текстовой области с помощью комбинации клавиш **Ctrl+C** (или ⌘ + C в операционной системе macOS) и вставьте его в верхнюю текстовую область с помощью **Ctrl+V** (или ⌘ + **V**). Затем перемещайте ползунковый регулятор влево или вправо, пока не увидите исходное сообщение.

Вы даже сможете «взломать» шифр Цезаря и прочитать сообщения, к которым у вас нет ключа. Медленно сдвигайте ползунковый регулятор, пока вы не сможете прочитать расшифрованное сообщение в нижней текстовой области. Попробуйте! Введите следующее сообщение в область ввода:

```
Epdetyr esp lmtwtej ez mcplv esp Nlpdlc ntaspc htes xj Dpncpe Xpddlrp laa…
```

Затем установите ползунковый регулятор в крайнее левое положение и медленно сдвигайте его вправо, пока не сможете прочитать незашифрованное секретное сообщение. Можете ли вы сказать, какой секретный ключ использовался для кодирования сообщения? См. рис. 7.18 для подсказки.

Перемещение ползункового регулятора в положение `-11` приводит к сообщению на английском языке с открытым текстом, поэтому секретный ключ должен быть равен `11`. Также вы можете заметить, что ключ `15` также сломает сообщение (потому что 15 = 26–11). Всегда можно найти пару значений, которые взламывают сообщения, написанные с использованием базового латинского алфавита. Конечно, вы можете писать сообщения и на других языках, как показано на рис. 7.19.

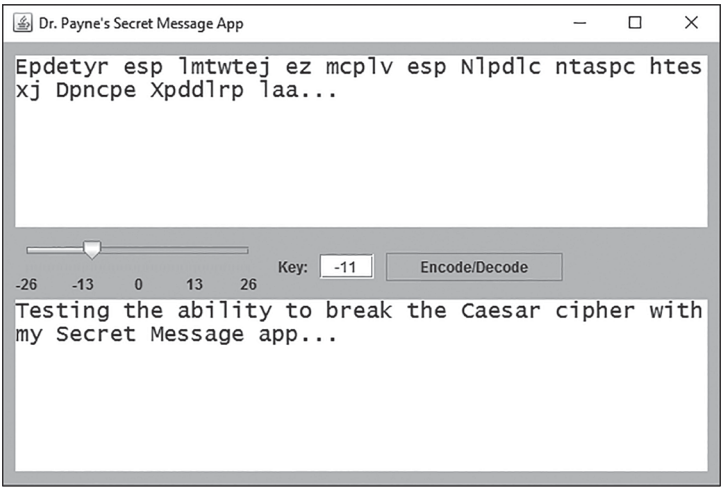


Рис. 7.18. Приложение «Секретные сообщения» можно использовать для взлома сообщений, зашифрованных шифром Цезаря. Сдвигайте ползунковый регулятор влево и вправо, пока не появится сообщение с открытым текстом

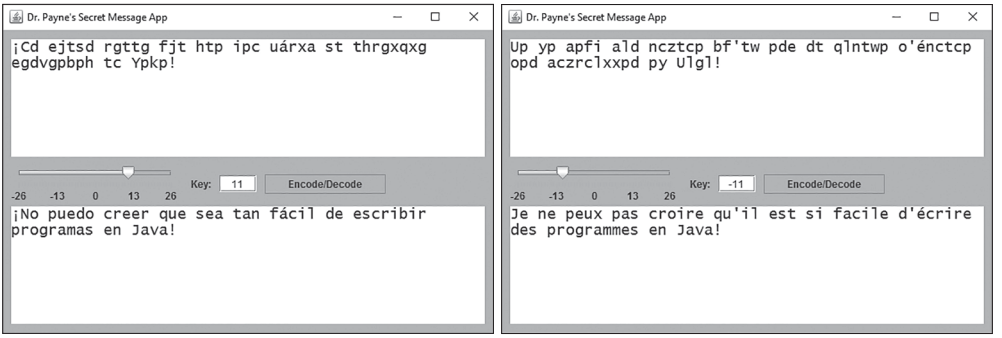


Рис. 7.19. Любые языки, которые используют символы базового латинского алфавита, могут использоваться в этой версии приложения «Секретные сообщения» для отправки и получения зашифрованных сообщений; Здесь показаны испанский (слева) и французский (справа) языки

Вы можете копировать и вставлять зашифрованные сообщения, используя их для переписки по электронной почте, Twitter и Facebook — вы даже можете отправлять закодированные текстовые сообщения, хотя это будет проще сделать с вашего мобильного устройства. Мы рассмотрим, как превратить приложение «Секретные сообщения» для компьютера в мобильное приложение, в главе 8.

Чтобы с легкостью переписываться секретными сообщениями с друзьями, необходимо поделиться с ними нашим приложением «Секретные сообщения». Мы научимся это делать в следующем разделе.

Бонус: совместное использование вашего приложения как запускаемого JAR-файла

Чтобы приложение «Секретные сообщения» было действительно полезным, должна быть возможность делиться им со своими друзьями — возможно, даже с такими друзьями, которые не знают, как программировать на языке Java и не имеют на компьютере установленной среды разработки Eclipse.

Однако это не проблема — дело в том, что программа Eclipse и язык Java позволяют легко экспортировать приложение в виде запускаемого файла и делиться им с любым из миллионов пользователей компьютеров по всему миру. Вашим друзьям даже не нужно загружать JDK для разработчиков; все, что им нужно, это JRE, которая установлена на большинстве компьютеров. Среда разработки Eclipse может экспортировать приложение в исполняемый Java-архив (JAR), который можно отправить по электронной почте, перенести на USB-накопителе или другим способом. Вашим друзьям нужно лишь сделать двойной щелчок по JAR-файлу для запуска приложения!

Чтобы экспортировать исполняемый JAR-файл из среды разработки Eclipse, выберите команду меню **File ⇒ Export** (Файл ⇒ Экспорт). Затем разверните папку *Java* и выберите пункт **Runnable JAR file** (Исполняемый JAR-файл), как показано на рис. 7.20.

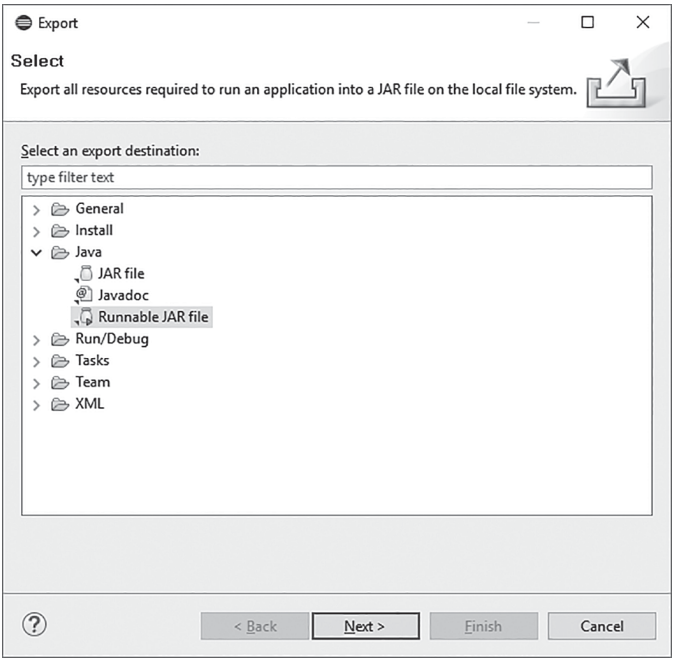


Рис. 7.20. Поделитесь своим приложением с другими пользователями, экспортировав исполняемый JAR-файл

Нажмите кнопку **Next** (Далее), и программа Eclipse спросит о конфигурации запуска и месте назначения экспорта. Конфигурация запуска означает, какой класс или приложение вы хотите запустить и из какого проекта. В разделе **Launch confguration** (Конфигурация запуска) откройте раскрывающийся список и выберите пункт **SecretMessagesGUI — SecretMessagesGUI**. Это означает, что вы хотите запустить файл *SecretMessagesGUI.class* в проекте *SecretMessagesGUI*.

Вы должны скомпилировать и запустить приложение хотя бы один раз, чтобы получить конфигурацию запуска для этого приложения.

Расположение экспорта — это местоположение и имя файла, которые вы хотите использовать для исполняемого приложения. В разделе **Export destination** (Расположение экспорта) нажмите кнопку **Browse** (Обзор), а затем выберите папку, в которую хотите сохранить свое готовое приложение, например каталог *Рабочий стол*. Присвойте программе название, например *Secret Messages.jar*. Вы можете использовать пробелы (или другие специальные символы) в имени файла. По завершении (рис. 7.21) нажмите кнопку **Save** (Сохранить), а затем **Finish** (Готово).

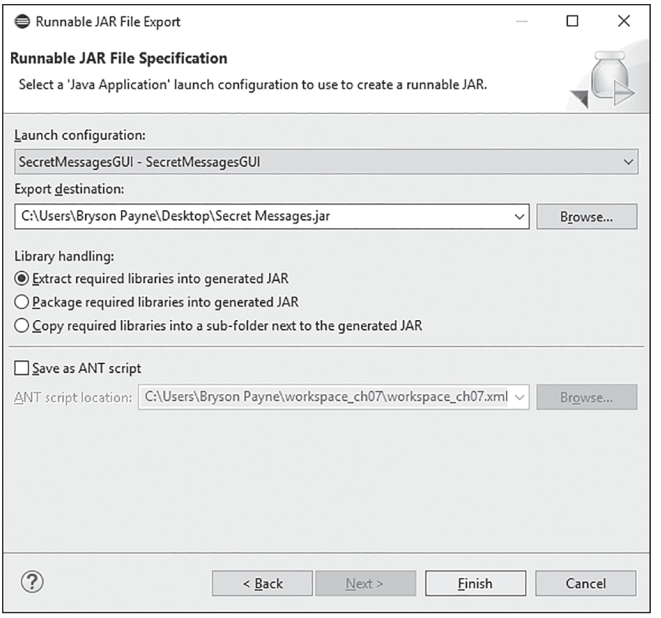


Рис. 7.21. Выберите местоположение и имя для запускаемого JAR-файла

Возможно, после того, как вы нажмете кнопку **Finish** (Готово), появится предупреждение, однако его можно проигнорировать и нажать кнопку **OK**. Перейдите в папку, в которой сохранили JAR-файл, и вы увидите значок приложения на языке Java с указанным вами именем файла (рис. 7.22). Вы можете запустить его и отправлять и получать зашифрованные сообщения.

Рис. 7.22. Значок исполняемого JAR-файла на рабочем столе, в операционной системе Windows (слева) и macOS (справа)

В новых версиях операционной системы macOS вам нужно будет нажать и удерживать клавишу ^, когда вы щелкнете мышью по значку приложения в первый раз, а затем выбрать пункт **Open** (Открыть). Затем нужно нажать кнопку **Open** (Открыть) в появившемся окне. После этого приложение запустится.

Из-за соображений безопасности некоторые почтовые программы могут блокировать пересылку файлов с расширением jar, однако вы можете загрузить файл в Облако или в свою любимую программу совместного доступа (например, Dropbox, Google Drive или OneDrive) и поделиться им таким образом. Получив файл, ваши друзья могут просто запустить его. Через это вы сможете делиться зашифрованными сообщениями друг с другом.

Точно также можно экспортировать исполняемый JAR-файл для приложения с графическим интерфейсом «Больше-Меньше» из главы 3. Повторите описанные выше действия для файла *GuessingGame.java* в проекте *GuessingGame*.

Что вы изучили

При создании этого приложения вы повторно использовали код из консольного приложения «Секретные сообщения». При этом вы существенно расширили свое понимание проектирование и программирования графического интерфейса пользователя. Вот некоторые из навыков, полученных вам в этой главе:

- последовательное именование компонентов и переменных графического интерфейса для удобства чтения и повторного использования кода;
- повторное использование алгоритмов, таких как метод `encode()`, который мы скопировали из консольной версии нашего приложения;
- объявление методов, принимающих параметры и возвращающих данные;
- написание более продвинутых обработчиков событий для различных компонентов графического интерфейса, включая кнопки, ползунковые регуляторы и текстовые поля;
- работа со свойствами компонентов графического интерфейса в режиме конструктора для дальнейшей настройки пользовательского интерфейса;
- настройка переноса текстовых строк по словам для объектов `JTextArea`;
- создание всплывающих окон `javax.swing.JOptionPane`, в том числе диалоговых окон сообщений;
- добавление графического ползункового регулятора `JSlider` и работа с ним;
- изменение текстового поля с помощью обработчика событий ползункового регулятора `stateChanged()`;
- экспорт исполняемого JAR-файла, который ваши друзья смогут запускать у себя на компьютере.

Дополнительные задачи

Попробуйте выполнить следующие задания. Таким образом, вы повторите пройденный материал, попрактикуетесь в программировании и получите новые навыки. Если у вас возникнут проблемы, вы можете загрузить примеры решений на сайте по адресу **https://eksmo.ru/files/learn_java_easy_way.zip**.

Задача № 1: Перемещение вверх!

Приложение «Секретные сообщения» отлично подходит для отправки простых, зашифрованных сообщений друзьям, но не менее забавно использовать его самостоятельно. Однако довольно часто приходится копировать зашифрованные сообщения нижней

(результирующей) текстовой области и вставлять их в верхнюю. Ваша первая задача состоит в создании кнопки **Move Up** ∧, которая перемещает зашифрованное сообщение в верхнюю текстовую область и автоматически расшифровывает его!

Поместите кнопку **Move Up** ∧ рядом с кнопкой **Encode/Decode**. Затем добавьте обработчик событий, который получит текст из текстовой области `txtOut` и установит его как содержимое текстовой области `txtIn`. В качестве бонуса обработчик события для кнопки **Move Up** ∧ должен изменить значение ползункового регулятора на противоположное. Кроме того, полагаю, стоит изменить фон новой кнопки, чтобы он соответствовал фону остальной части приложения. Пример решения этой задачи показан на рис. 7.23.

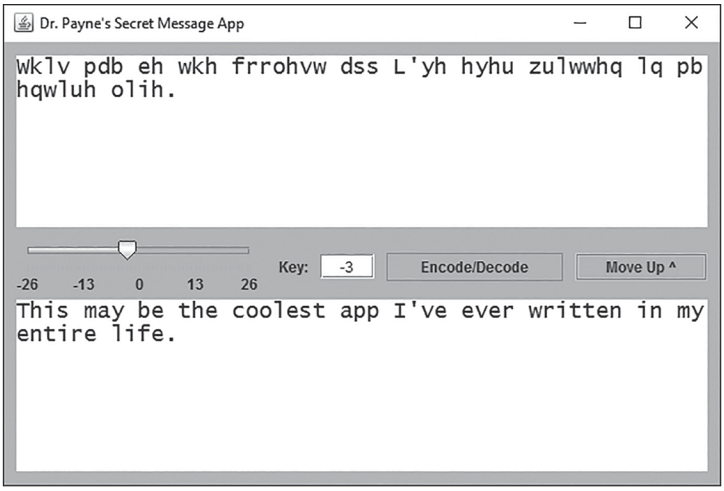


Рис. 7.23. Добавление кнопки **Move Up** ∧ для перемещения сообщения из нижней в верхнюю текстовую область и автоматической его расшифровки. Кроме того, соответствующим образом изменяется значение ползункового регулятора

Задача № 2: Прокрутка!

Еще одно улучшение, которое можно было бы внести в приложение, — это обработка очень длинных сообщений. Для этого нужно добавить вертикальную полосу прокрутки и автоматически промотывать ее, когда пользователь вводит слишком длинное для текстовой области сообщение. Текстовая область `JTextArea` не добавляет полосу прокрутки сама по себе, но среда разработки Eclipse

позволяет быстро и просто добавить полосу прокрутки в любую текстовую область.

В режиме конструктора щелкните правой кнопкой мыши по любой из текстовых областей `JTextArea` и выберите пункт **Surround with ⇒ javax.swing.JScrollPane** (Заключить в ⇒ javax.swing.JScrollPane) в контекстном меню, как показано на рис. 7.24.

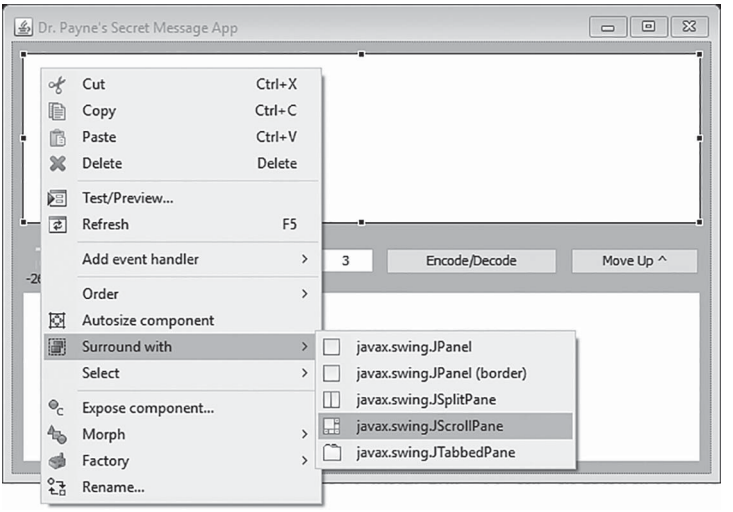


Рис. 7.24. Добавление автоматически генерируемой полосы прокрутки в текстовую область `JTextArea`

Разумеется, вам придется сделать такие же действия и для другой текстовой области `JTextArea`. Когда вы все сделаете, запустите приложение и введите длинный текст в текстовое поле ввода, чтобы убедиться, что полоса прокрутки появляется автоматически. На рис. 7.25 я ввел всю Конституцию США в верхнюю текстовую область и зашифровал ее.

На рис. 7.25 видна только часть текста, однако теперь с правой стороны обеих текстовых областей есть полоса прокрутки. Полоса прокрутки показывает нам, что текст, с которым мы работаем, полностью не помещается в текстовую область и, если мы хотим посмотреть на оставшуюся часть, нам надо промотать полосу прокрутки вниз.

Чтобы еще лучше настроить приложение, изучите панель **Properties** (Свойства) для всех компонентов графического интерфейса пользователя, которые мы добавили, и попробуйте изменить различные значения свойств, такие как цвета фона, шрифты и так далее. Добавьте нашему приложению индивидуальности и поделитесь им с друзьями.Похвастайтесь своим творением!

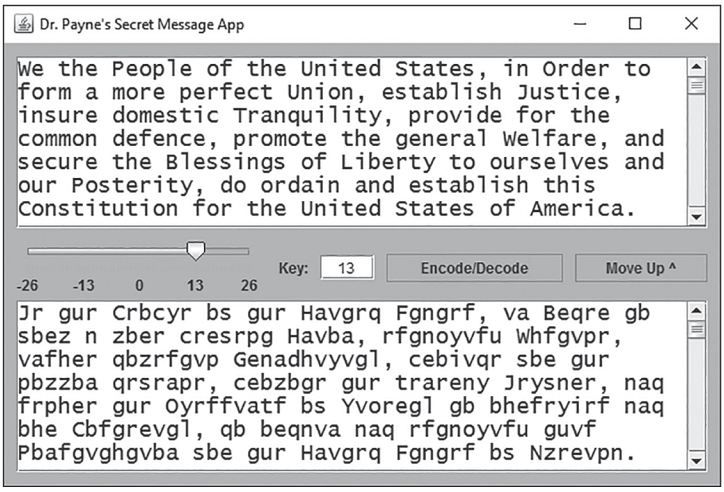


Рис. 7.25. Добавив полосу прокрутки `JScrollPane` к каждой текстовой области `JTextArea`, вы сможете зашифровать текст любого размера

Задача № 3: Изменение значения ползункового регулятора после изменения текста в текстовом поле

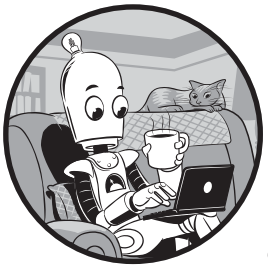
Давайте сделаем последнюю настройку в пользовательском интерфейсе. Когда пользователь перемещает ползунковый регулятор, значение ключа, отображаемое в текстовом поле, изменяется. Но что, если мы хотим, чтобы ползунковый регулятор двигался каждый раз, когда изменяется значение в текстовом поле?

§ Вы собираетесь добавить обработчик события для текстового поля `txtKey`. Щелкните правой кнопкой мыши по полю `txtKey`, а затем выберите пункт **Add event handler** ⇒ **key** ⇒ **keyReleased** (Добавить обработчик события ⇒ key ⇒ keyReleased) в контекстном меню. Эта операция создаст обработчик событий, который будет отслеживать нажатия клавиш в текстовом поле `txtKey`.

Чтобы выполнить это задание, вам нужно написать код для обработчика событий, который получит целочисленное значение из текстового поля и установит ползунковый регулятор в это значение. Не забудьте использовать блок `try-catch` при работе с введенным пользователем текстом. Удачи!

Глава 8

МОБИЛЬНАЯ ВЕРСИЯ ПРИЛОЖЕНИЯ «СЕКРЕТНЫЕ СООБЩЕНИЯ» ДЛЯ ОБЩЕНИЯ С ДРУЗЬЯМИ



В этой главе мы завершим разработку приложения «Секретные сообщения», создав мобильную версию, с помощью которой можно будет отправлять тайные сообщения по электронной почте, с помощью программы отправки текстовых сообщения и даже размещать их в социальных сетях.

Приложение будет похоже на версию с графическим интерфейсом из главы 7 с метками, текстовыми полями и с ползунковым регулятором, позволяющим быстро менять значения ключа для шифрования и расшифровки сообщений. Кроме того, вы сможете копировать и вставлять сообщения из приложения в электронные письма и текстовые сообщения, а в конце главы вы сможете отправлять электронную почту или текст непосредственно из самого приложения «Секретные сообщения»! На рис. 8.1 показано работающее приложение на устройстве Android.

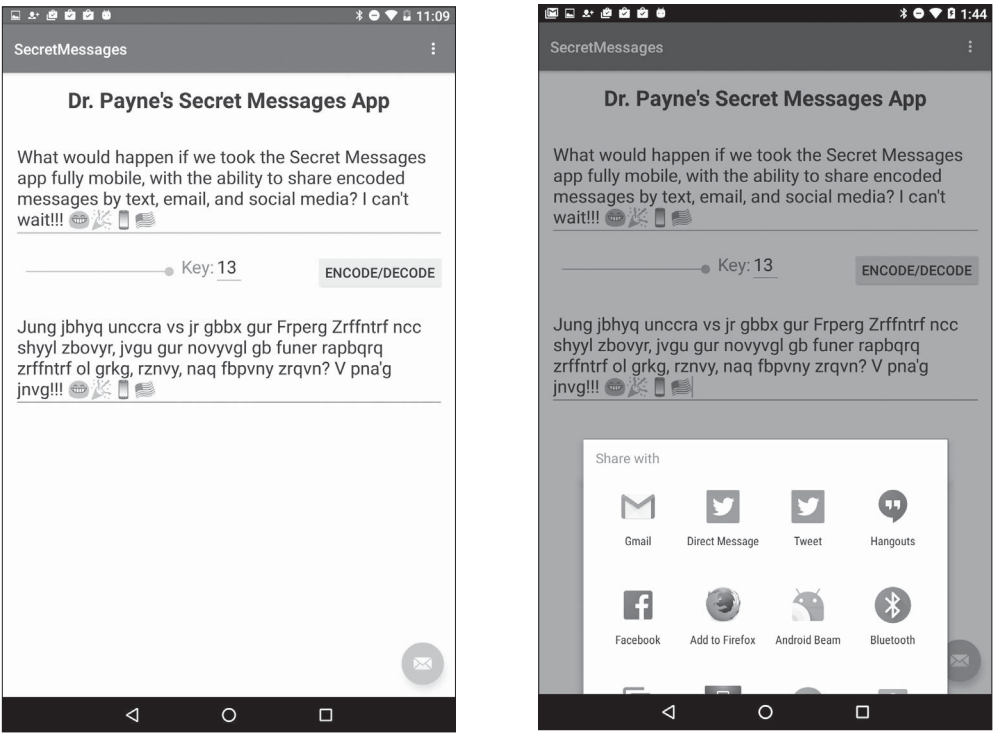


Рис. 8.1. Приложение «Секретные сообщения» расшифровывает секретные сообщения (слева) и позволяет делиться ими и передавать их с помощью электронной почты непосредственно из приложения, одним нажатием кнопки (справа)

Мы будем повторно использовать код из приложения для компьютера, сохранив имена компонентов графического интерфейса в версии для устройства Android, используя преимущества согласованности структуры и языка Java на разных платформах.

Настройка графического интерфейса мобильного приложения

Давайте приступим к разработке приложения «Секретные сообщения» с помощью среды разработки Android Studio. Эта программа, возможно, при запуске запросит установку обновлений, что рекомендуется сделать. Обновление программ несет в себе некоторую долю риска, так как оно может изменить внешний вид и интерфейс, переместив кнопки или элементы меню, но обновления также включают в себя новые настройки безопасности и последние возможности ОС Android.

Когда вы откроете среду разработки Android Studio, вы можете увидеть последний проект, над которым вы работали, — в нашем случае — игру «Больше-Меньше». Можно закрыть его, выбрав команду меню **File ⇒ Close Project** (Файл ⇒ Закрыть проект).

Чтобы начать создание нового приложения «Секретные сообщения», выберите **Start a new Android Studio project** (Начать новый проект Android Studio) на экране приветствия Android Studio или запустите команду меню **File ⇒ New Project** (Файл ⇒ Новый проект) в уже открытой среде разработки.

Присвойте новому проекту имя **SecretMessages**, сохраните его в удобной для вас папке и нажмите кнопку **Next** (Далее).

На экране **Target Android Devices** (Целевые устройства Android) выберите пункт **API 16: Android 4.1 (Jelly Bean)** в качестве минимальной версии SDK, как мы делали для игры «Больше-Меньше», а затем нажмите кнопку **Next** (Далее). Выберите пункт **Basic Activity** (Основнные операции) на экране **Add an Activity** (Добавить операцию) и нажмите кнопку **Next** (Далее). В окне **Customize the Activity** (Настроить операции) сохраните все имена по умолчанию и нажмите кнопку **Finish** (Готово).

Настройка и открытие нового проекта может потребовать некоторое время. После успешного старта дважды щелкните мышью по файлу *content_main.xml* в разделе *app ⇒ res ⇒ layout* на панели **Project Explorer** (Обозреватель проекта). Если появится необходимость, закройте уведомление о всплывающей кнопке действия (мы займемся ею позже).

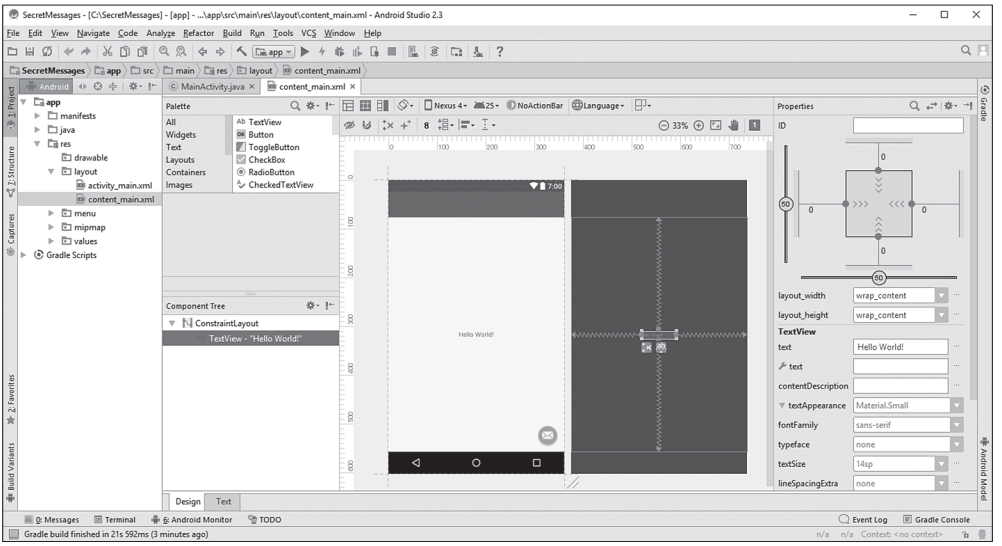


Рис. 8.2. Начальный вид приложения в среде разработки Android Studio

Ваш экран должен выглядеть так, как показано на рис. 8.2. (Возможно, вам придется перейти на вкладку **Design** (Конструктор), чтобы увидеть окно предварительного просмотра графического интерфейса пользователя.)

Разработка графического интерфейса мобильного приложения

Мы создали новый проект для Android. Теперь давайте начнем разработку макета нашего приложения. Сначала удалите текст «Hello World!», отображаемый в окне предварительного просмотра. Для этого выделите его мышью, а затем нажмите клавишу **Del**. Затем, как и в предыдущем приложении, добавим макет RelativeLayout для хранения элементов графического интерфейса. Для этого на панели **Palette** (Панель элементов) в разделе **Layouts** (Макеты) выберите макет **RelativeLayout** и перетащите его в окно панели **Preview** (Предварительный просмотр) или на элемент `ConstraintLayout` на панели **Component Tree** (Дерево компонентов).

Затем добавим заголовок в верхнюю часть приложения. Для этого выберите виджет `TextView` в разделе виджетов в палитре. Поместите виджет `TextView` в верхнюю часть экрана, измените текст на **Ваше имя Secret Messages App** и на панели **Properties** (Свойства) присвойте свойству `textAppearance` значение `Material.Large`, как это показано на рис. 8.3.

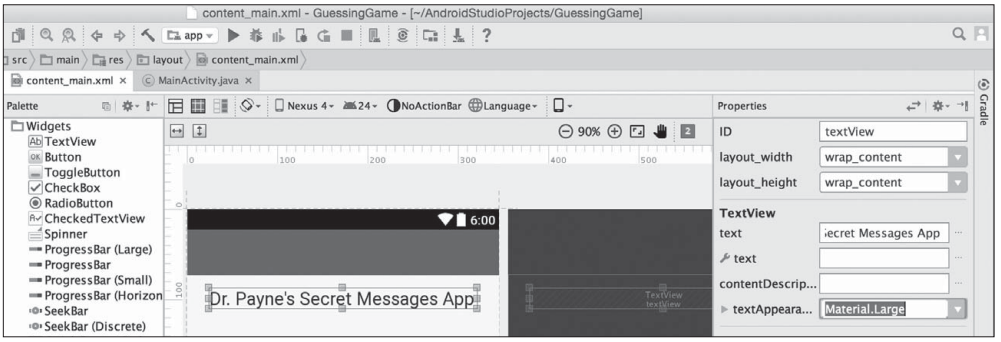


Рис. 8.3. В верхнюю центральную часть приложения добавьте крупный заголовок с вашим именем

На панели **Properties** (Свойства) разверните группу свойств `textAppearance`, найдите свойство `textStyle` и установите

флажок `bold`, чтобы отформатировать текст заголовка полужирным начертанием.

Теперь давайте разместим текстовое поле `EditText`, в которое пользователь будет вводить текст. На панели **Palette** (Панель элементов) в разделе **Text** (Текст) выберите пункт **Multiline Text** (Многострочный текст). Увеличьте окно предварительного просмотра и поместите текстовое поле примерно на 30 пунктов ниже заголовка. Измените текст на **Secret messages \n are so cool, \n aren't they?** как показано на рис. 8.4. В позициях символов **\n** будут добавлены символы *новой строки*; эти символы не отображаются на экране, но они начинают новую строку, которую вы получаете, нажимая на клавиатуре клавишу **Enter**. Помещение текста в несколько строк показывает пользователю, что он тоже может ввести сообщение, состоящее из нескольких строк.

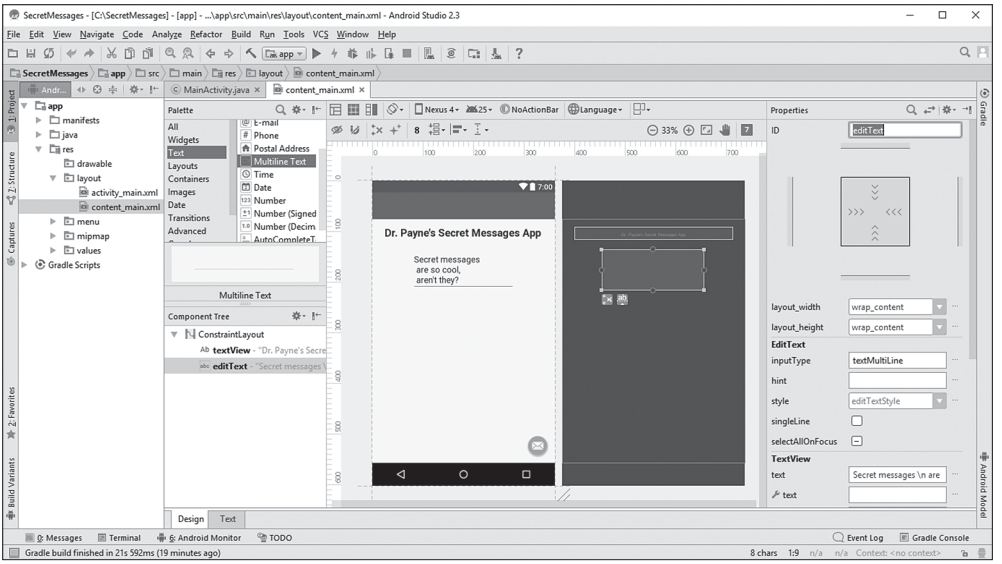


Рис. 8.4. Добавьте текстовое поле, состоящее из нескольких строк, для ввода сообщения пользователя. Для этой цели используется escape-последовательность `\n`, означающая переход на новую строку

Кроме того, можно изменить количество строк текста, которое пользователь может видеть одновременно на экране. Для элемента **Multiline Text** (Многострочный текст) можно, например, установить значение свойства `lines` (линии) равное `4`. Чтобы увидеть многострочное сообщение на всех устройствах, выберите пункт `textMultiLine` в раскрывающемся списке `inputType`.

Наконец, установим свойству `id` текстового поля значение `txtIn`. Так достигаются две цели: во-первых, нам будет проще использовать это текстовое поле дальше в коде, а во-вторых, нам необходимо поддерживать согласованность имен с версией приложения для компьютера. Теперь растяните текстовое поле на всю ширину приложения, перетащив его левую и правую границы, как показано на рис. 8.5.

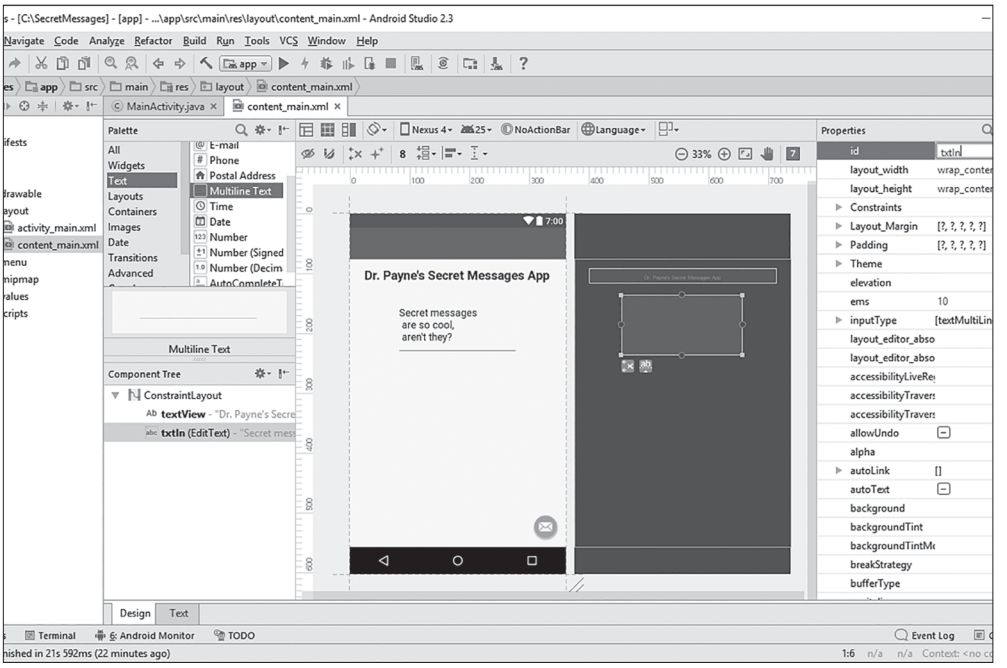


Рис. 8.5. Присвойте свойству `id` текстового поля значение `txtIn`, чтобы позже мы могли это использовать при программировании

Мы добавим четыре компонента в средний ряд приложения «Секретные сообщения»: компонент `SeekBar`, чтобы пользователь мог прокрутить панель и установить значение ключа, метку, текстовое поле для отображения значения ключа и кнопку **Encode/Decode**. Компонент `SeekBar` будет работать аналогично ползунковому регулятору `JSlider` в приложении для настольного компьютера. Не переживайте, если ваши компоненты будут расположены неровно, когда вы их разместите — позже мы сможем это исправить.

Во-первых, найдите компонент `SeekBar` в разделе **Widgets** (Виджеты) панели **Palette** (Панель элементов). Поместите его примерно на 30 пунктов ниже текстового поля для ввода сообщения, вдоль

левого поля, как показано на рис. 8.6. Вы можете установить расстояние от текстового поля вручную на панели **Properties** (Свойства), щелкнув мышью по ссылке **View all properties** (Показать все свойства) или нажав кнопку со стрелками влево и вправо в верхней части панели **Properties** (Свойства). В разделе `Layout_Margin` укажите значение **30dp** для параметра `layout_marginTop`.

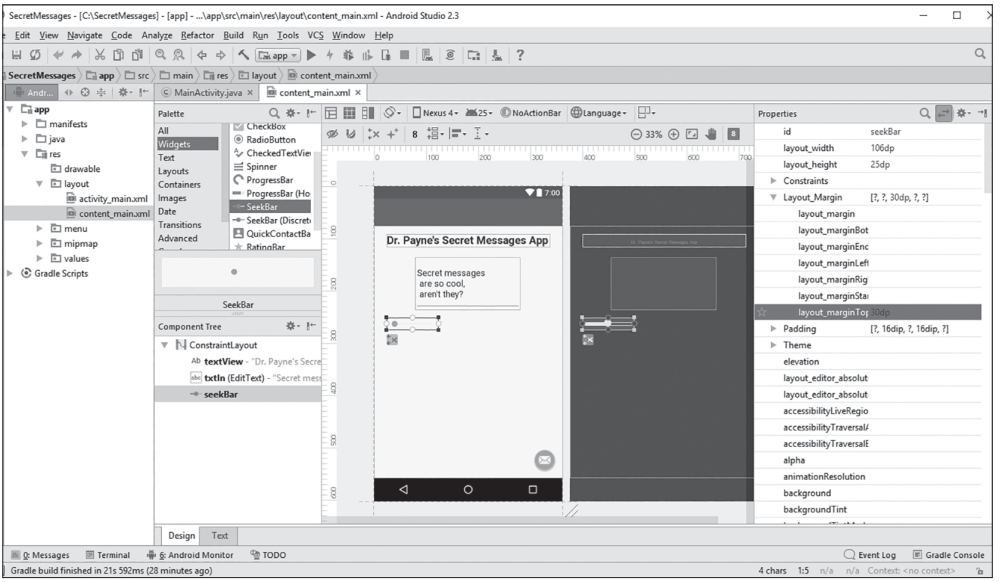


Рис. 8.6. Компонент `SeekBar` будет использоваться, подобно ползунковому регулятору в версии для компьютера, позволяя пользователю выбирать различные значениями ключа

Затем рядом с компонентом `SeekBar` добавьте виджет `TextView`, чтобы он служил меткой для текстового поля секретного ключа пользователя. На панели **Properties** (Свойства) присвойте свойству `textAppearance` значение `Material.Medium`, а свойству `text` — значение **Key:**.

Затем выберите текстовое поле **Number** (Число) для значения секретного ключа пользователя и отцентрируйте его по горизонтали. Установите `13` (или другое число, по вашему выбору) в качестве значения по умолчанию. Для этого изменение значение свойства `text`. Измените ширину текстового поля, чтобы в нем помещались числовые значения, изменив значение свойства `width` (щелкните мышью по ссылке **View all properties** (Просмотреть все свойства)) примерно до `40dp`, а затем присвойте свойству `id` значение `txtKey`.

После того, как вы разместите текстовое поле `txtKey`, вы сможете перетащить метку **Key:**, чтобы она расположилась непосредственно рядом с ним. Кроме того, вы можете увеличить ширину компонента `SeekBar`, чтобы заполнить промежуток, как показано на рис. 8.7.

Завершите средний ряд макета добавлением кнопки **Encode/Decode**. Выберите кнопку **Button** в разделе **Widgets** (Виджеты) на панели **Palette** (Панель элементов) и вставьте кнопку около правого края рядом с текстовым полем. Присвойте свойству `text` значение **Encode/Decode**.

Наконец, создайте текстовое поле для выводимого сообщения, скопировав текстовое поле `txtIn` и вставив его под средний ряд виджетов. Удалите текст в текстовом поле, растяните его до ширины окна приложения и присвойте свойству `id` значение `txtOut`.

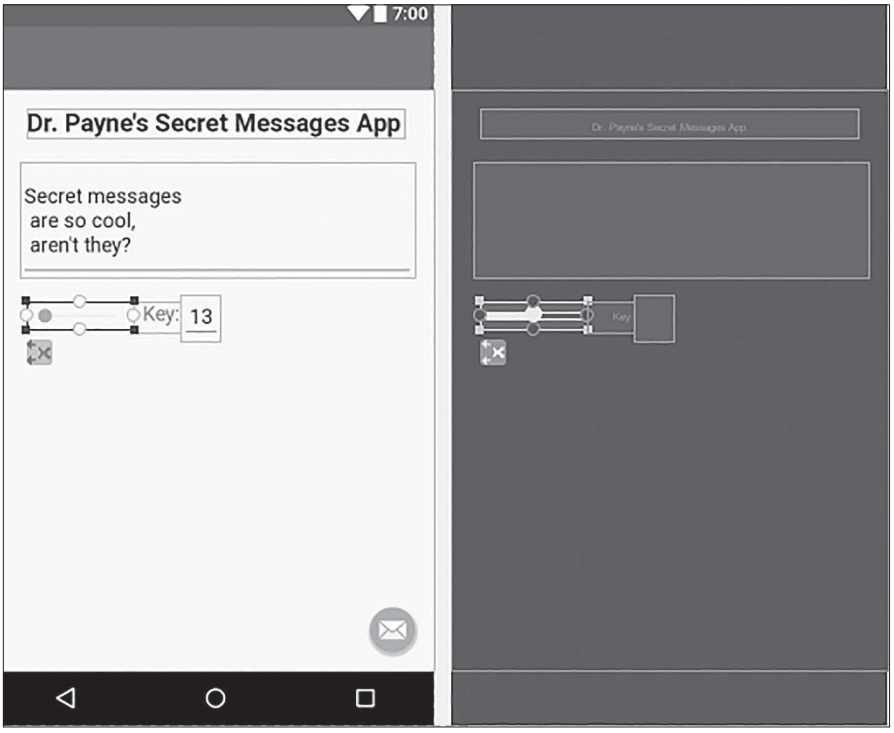


Рис. 8.7. После размещения текстового поля `txtKey` отрегулируйте расположение метки **Key:** и растяните компонент `SeekBar`

Вот и все. Готовый макет графического интерфейса показан на рис. 8.8.

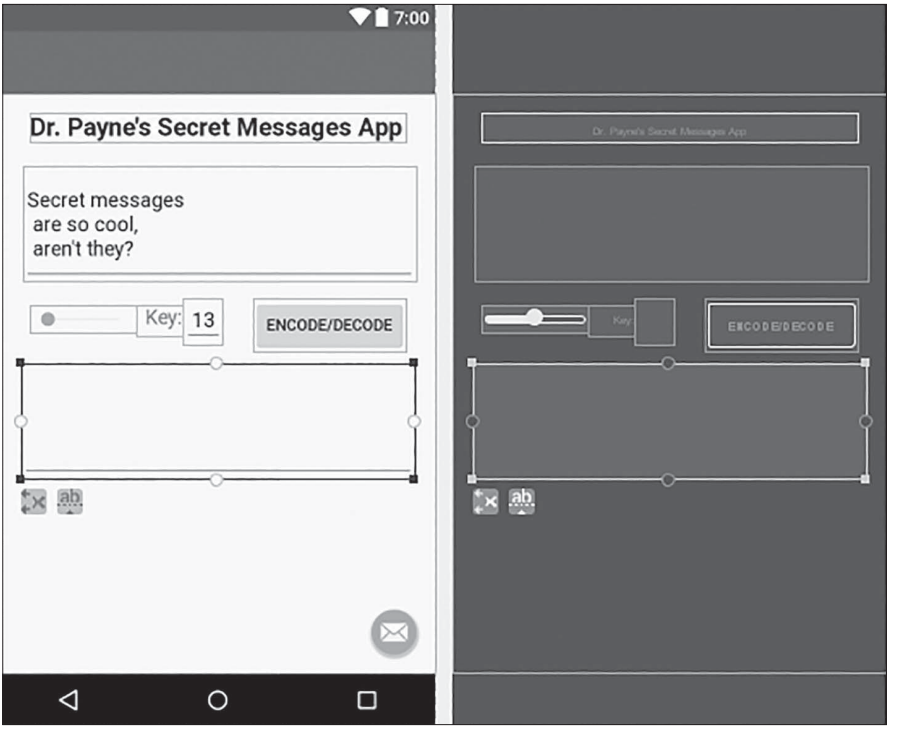


Рис. 8.8. Готовый макет мобильного приложения «Секретные сообщения»

Перед тем, как двигаться дальше, убедитесь, что свойство `id` текстового поля ввода пользователя равно значению `txtIn`, выводимого текста — `txtOut`, а центрального текстового поля для ключа — `txtKey`. Нам необходимо придерживаться строгого именования для корректной работы кода в следующем разделе.

Подключение графического интерфейса к коду на языке Java

Пришло время подключить макет графического интерфейса к исходному коду на языке Java, чтобы затем запрограммировать работу мобильного приложения «Секретные сообщения». Перейдите к файлу *MainActivity.java*, щелкнув мышью по соответствующей вкладке в левом верхнем углу окна программы.

Добавьте следующие пять строк непосредственно в объявление `public class MainActivity`:

```
public class MainActivity extends AppCompatActivity {
```

```
    EditText txtIn;
    EditText txtKey;
    EditText txtOut;
    SeekBar sb;
    Button btn;
```

Возможно, вам придется нажимать сочетание клавиш **Alt+Enter** (⌥ + ↩ в операционной системе macOS) после каждой строки, чтобы импортировать виджеты в код, написанный на языке Java.

Эти пять переменных указывают на компоненты графического интерфейса в макете. Мы можем связать эти имена переменных с реальными виджетами, добавив пять строк в метод `onCreate()` ниже инструкции `setSupportActionBar(toolbar);`:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
    txtIn = (EditText)findViewById(R.id.txtIn);
    txtKey = (EditText)findViewById(R.id.txtKey);
    txtOut = (EditText)findViewById(R.id.txtOut);
    sb = (SeekBar)findViewById(R.id.seekBar);
    btn = (Button)findViewById(R.id.button);
```

При наборе вам может помочь функция автозавершения среды разработки Android Studio, которая сама сумет закончить большинство команд. Все, что вам нужно сделать, это набрать несколько символов, затем выбрать нужный вариант из открывшегося списка или нажать клавишу **Enter**, чтобы принять рекомендованный. Этот инструмент позволит вам работать быстрее и избежать типичных ошибок.

Подключение кнопки шифрования к методу `encode()`

Мы соединили код с компонентами макета графического интерфейса. Теперь мы можем скопировать метод `encode()` из версии приложения для компьютера из главы 7. Мы вставим метод в объявлении `public class MainActivity`, прямо под пять строк,

которые мы только что добавили, объявляя переменные, указывающие на компоненты графического интерфейса в макете.

Откройте свой проект *SecretMessagesGUI* для компьютера в программе Eclipse и выделите весь метод `encode()`. Скопируйте этот код и вставьте его в объявление `public class MainActivity` в среде разработки Android Studio:

```
public class MainActivity extends AppCompatActivity {
    EditText txtIn;
    EditText txtKey;
    EditText txtOut;
    SeekBar sb;
    Button btn;
    public String encode(String message, int keyVal) {
        String output = "";
        char key = (char) keyVal;
        for (int x = 0; x < message.length(); x++) {
            char input = message.charAt(x);
            if (input >= 'A' && input <= 'Z')
            {
                input += key;
                if (input > 'Z')
                    input -= 26;
                if (input < 'A')
                    input += 26;
            }
            else if (input >= 'a' && input <= 'z')
            {
                input += key;
                if (input > 'z')
                    input -= 26;
                if (input < 'a')
                    input += 26;
            }
            else if (input >= '0' && input <= '9')
            {
                input += (keyVal% 10);
                if (input > '9')
                    input -= 10;
                if (input < '0')
                    input += 10;
```

```
            }
            output += input;
        }
        return output;
    }
```

Теперь, нам просто нужно вызывать метод `encode()` при каждом нажатии кнопки **Encode/Decode**. Мы создадим метод `OnClickListener` для кнопки **Encode/Decode** и вызовем из него метод `encode()`.

Внутри метода `onCreate()` в файле *MainActivity.java* начните вводить код **`btn.setOnCl`**, пока не появится список рекомендаций автозавершения кода, как показано на рис. 8.9. В списке выберите **`setOnClickListener()`**.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
    txtIn = (EditText)findViewById(R.id.txtIn);
    txtKey = (EditText)findViewById(R.id.txtKey);
    txtOut = (EditText)findViewById(R.id.txtOut);
    sb = (SeekBar)findViewById(R.id.seekBar);
    btn = (Button)findViewById(R.id.button);

    btn.setOnCl
        setOnClickListener(OnClickListener l)
        setOnContextClickListener(OnC...  void
        setOnLongClickListener(OnLong...  void
```

Рис. 8.9. Вы можете полагаться на рекомендации по автозавершению кода от программы Android Studio, которые помогают ускорить ввод. В данном случае рекомендуется команда `setOnClickListener()` для кнопки **Encode/Decode**

После выбора команды `setOnClickListener()` щелкните мышью внутри круглых скобок этого метода и начните вводить new **`OnClickListener`**, как показано на рис. 8.10. Затем дважды щелкните мышью по первой из появившихся подсказок кода:

Рис. 8.10. Еще раз используйте автозавершение, чтобы закончить команду `OnClickListener`. На этот раз среда разработки Android Studio добавит несколько строк кода

Вы заметите, что программа Android Studio добавила несколько строк кода для обработчика событий `OnClickListener()`:

```
btn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
    }
});
```

Эти несколько строчек создают слушатель событий, используя анонимный внутренний класс типа `View.OnClickListener()`, с обработчиком события `onClick()` для ответа на нажатие пользователем кнопки.

Затем мы настроим метод `encode()`. Для метода `encode()` неважно, что происходит в остальной части нашего кода, если мы передадим ему два аргумента правильного типа данных. Это означает, что нам не нужно в новом приложении менять метод `encode()`, однако нам все-таки придется настроить значения, которые мы передадим в этот метод в качестве аргументов.

Чтобы зашифровать сообщение, введенное пользователем в поле `txtIn`, нам нужно будет получить значение ключа из поля `txtKey`. Затем мы получим строку текста, введенную пользователем в поле `txtIn`. Мы передадим оба эти значения методу `encode()` в качестве аргументов и сохраним зашифрованный результат в переменной с именем `output`. Наконец, мы выведем зашифрованный текст, хранящийся в переменной `output`, в текстовом поле `txtOut`.

Добавьте следующие четыре строки внутрь фигурных скобок метода `onClick()`:

```
btn.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
```

```
❶          int key = Integer.parseInt(txtKey.getText().toString());
❷          String message = txtIn.getText().toString();
❸          String output = encode(message, key);
❹          txtOut.setText(output);
       }
   });
```

В строке ❶ мы используем метод `Integer.parseInt()` для преобразования текста, введенного пользователем, в целочисленное значение ключа. Единственное отличие этого метода `parseInt()` от тех, что мы использовали ранее, заключается в том, что для поля txtKey используются два метода: `getText()` и `toString()`. Для устройства Android метод `getText()` для текстового поля `EditText` не возвращает строку напрямую — он возвращает гибкий тип под названием `Editable` (Редактируемый). Все текстовые типы от `Plain` (Простой текст) и `Password` (Пароль) до `Number` (Число), возвращают объект типа `Editable`, потому что текст внутри типа `EditText` предназначен для изменения и редактирования пользователем. Мы используем метод `toString()`, чтобы превратить текст из типа `EditText` в строку, из которой мы можем извлечь введенное пользователем число.

То же самое можно сказать и про строку ❷: нам нужны оба метода: `getText()` и `toString()`, чтобы получить исходное сообщение из текстового поля `txtIn` и превратить его в строку, которую нужно зашифровать. В строке ❸ мы вызываем метод `encode()` с параметрами `message` и `key` и сохраняем результат в переменной `output`. Наконец, в строке ❹ мы передали значение переменной `output` в текстовое поле `txtOut` для отображения зашифрованного сообщения.

Теперь наше приложение готово для первого теста. Давайте запустим приложение в эмуляторе Android Emulator и проверим, как оно работает.

Тестирование приложения

Сохраните свою программу. Затем выберите команду меню **Run** ⇒ **Run *'app'*** (Выполнить ⇒ Выполнить *'приложение'*). Выберите эмулятор, который вы установили в главе 4 (например, My Nexus 6P), как показано на рис. 8.11.

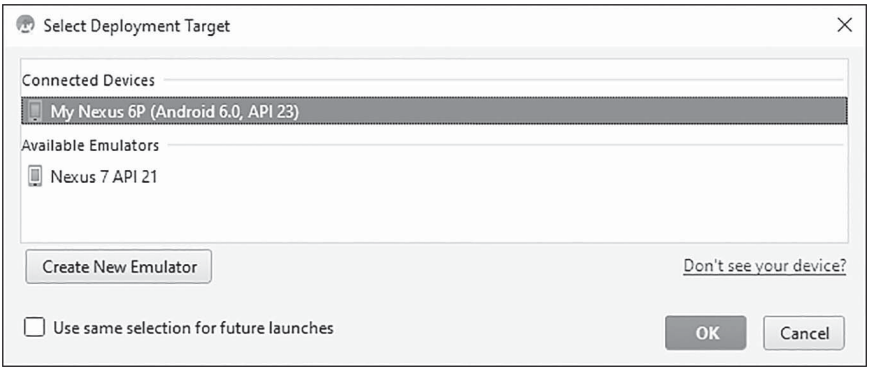


Рис. 8.11. Выбор эмулятора

Эмулятору может понадобиться несколько минут для первого запуска. Не забудьте, что вы можете не останавливать его во время работы в программе Android Studio. Тем самым вы избежите длительного запуска эмулятора при каждом запуске приложения. При желании вы можете запустить приложение непосредственно на своем устройстве Android, как мы делали в главе 4, и пропустить работу с эмулятором.

Когда эмулятор запустится и приложение «Секретные сообщения» загрузится, вы увидите начальный макет, как показано на рис. 8.12 (слева). Введите значение ключа, затем нажмите кнопку **Encode/Decode**, и вы увидите, что устройство Android преобразует сообщение в зашифрованный текст, как показано на рис. 8.12 (справа).

Вы можете копировать сообщение в одном текстовом поле и вставлять текст в другое, но ползунковый регулятор `SeekBar` пока не работает. Всплывающая кнопка действия — с символом конверта в правом нижнем углу — тоже пока не задействована. Не переживайте, мы скоро поработаем над ними.

Обратите внимание на еще одну внешнюю проблему: зашифрованное сообщение (`Frperg zrffntrf`...) подчеркнуто, потому что оно похоже на текст с ошибками. Чтобы отключить проверку орфографии, вернитесь к файлу макета *content_main.xml* и выберите текстовое поле `txtOut`. На панели **Properties** (Свойства) разверните категорию `inputType` и выберите свойство `textNoSuggestions`. Кроме того, при желании можете отключить проверку орфографии для текстового поля `txtIn`. Это можно сделать как для каждого поля по отдельности, так и одновременно, выделив в макете оба текстовых поля при нажатой клавише **Shift**, а затем выбрав свойство `textNoSuggestions`. После этого снова запустите приложение на эмуляторе. Больше проверка орфографии не производиться не будет.

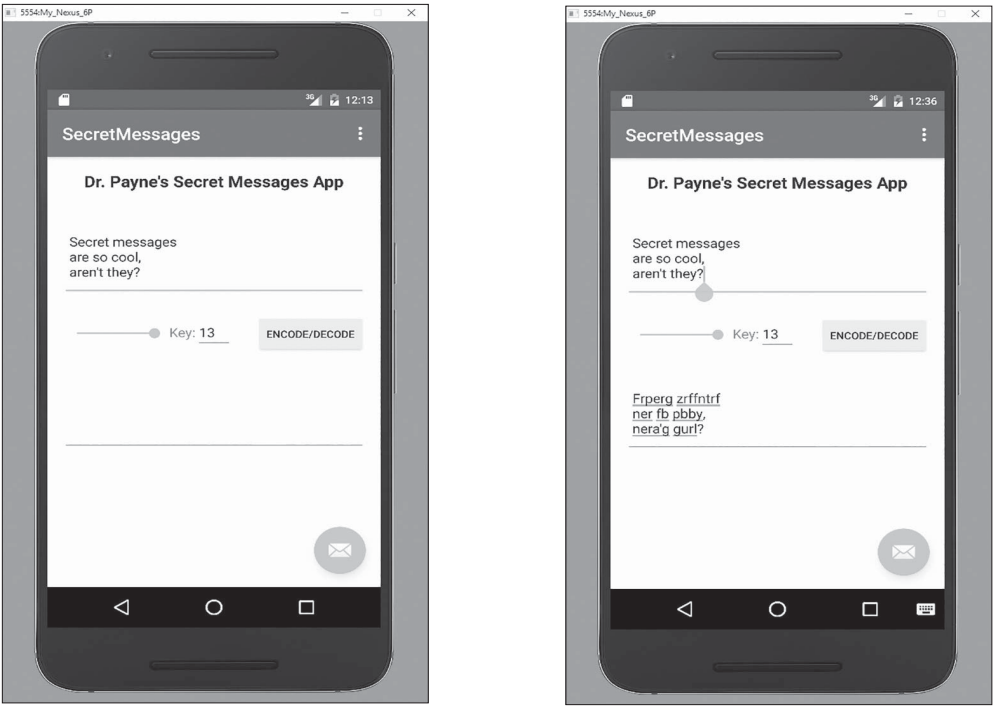


Рис. 8.12. Приложение «Секретные сообщения», работающее на эмуляторе устройства Android Nexus 6P (слева); нажатие кнопки **Encode/Decode** зашифровывает текст (справа)

После этой небольшой настройки мы готовы подключить компонент `SeekBar` для легкого и быстрого переключения между разными значениями ключа.

Работа с компонентом `SeekBar`

Мы собираемся подключить компонент `SeekBar`, при перемещении которого влево или вправо будет меняться выводимое сообщение. Нам потребуется настроить некоторые его свойства.

Во-первых, давайте немного узнаем о виджете `SeekBar`. Вероятно, вы видели компонент `SeekBar` при воспроизведении видео на своем мобильном устройстве. Например, при проигрывании видео на YouTube компонент `SeekBar` показывает, на каком месте видео вы находитесь, и позволяет вам перемотать вперед или назад — иногда этот процесс называют *поиском*.

Компонент `SeekBar` отличается от ползункового регулятора `JSlider` несколькими характеристиками. Во-первых, компонент

SeekBar может принимать только неотрицательные значения. Минимальным значением является ноль, а в качестве максимального можно задать любое положительное значение, например 26. Кроме того, компонент SeekBar не показывает метки чисел на самой панели. Это означает, что если мы хотим, чтобы пользователи нашего приложения могли использовать отрицательные значения ключа, нам придется выполнить ряд расчетов для преобразования положительных значений в отрицательные. Чтобы пользователь мог использовать значения ключа в интервале от −13 до +13, мы сделаем диапазон значений компонента SeekBar от 0 до 26. Кроме того, нам потребуется обновлять значение поля txtKey, чтобы в нем отражалось текущее значение ключа, определяемое положением компонента SeekBar. Начнем с настройки некоторых свойств компонента SeekBar в макете графического интерфейса.

Вернитесь в файл макета *content_main.xml* и выберите компонент SeekBar в окне предварительного просмотра. На панели **Properties** (Свойства) для компонента SeekBar, найдите свойства max и progress и измените значения обоих, сделав их равными **26**.

Свойство max определяет максимальное значение, которое отобразит компонент SeekBar. Для приложения «Секретные сообщения» нам нужно как минимум значение 26 для нумерования всего латинского алфавита. Мы хотим, чтобы значения варьировались от −13 до +13, чтобы пользователю было удобнее. Таким образом, всего будет 27 значений, поэтому в качестве максимального значения мы установим 26 (что и составит 27 значений вместе с нулем). Свойство progress отражает текущее значение компонента SeekBar, аналогичное свойству value ползункового регулятора JSlider. Для получения значения, на которое указывает компонент SeekBar, мы будем использовать метод getProgress().

Сохраните изменения, а затем вернитесь в файл *MainActivity.java*. Внутри метода onCreate(), прямо под строкой});, завершающей код btn.setOnClickListener(), начните вводить код **sb.set** и, используя автозавершение, найдите пункт setOnSeekBarChangeListener(). Это действие сохранит компонент SeekBar в переменной sb.

Код слушателя компонента SeekBar отличается от кода кнопки **Encode/Decode**. Мы хотим прослушивать не только события щелчков на компоненте SeekBar. Мы хотим прослушивать все события, в том числе ситуацию, когда пользователь нажимает

на компонент SeekBar и сдвигает его влево или вправо. Для этого нам нужен слушатель OnSeekBarChangeListener.

Как и ранее, мы будем использовать автозавершение для автоматического создания нового слушателя OnSeekBarChangeListener. Внутри круглых скобок после команды sb.setOnSeekBarChangeListener() начните вводить код new OnSeekBar и используйте рекомендацию ввода кода.

Среда разработки Android Studio дополнит код слушателя OnSeekBarChangeListener тремя методами: onProgressChanged(), onStartTrackingTouch() и onStopTrackingTouch(). Мы хотим использовать onProgressChanged(), поскольку тем самым, мы узнаем, когда пользователь изменил значение компонента SeekBar.

Действия, на которые мы хотим запрограммировать слушатель событий SeekBar, похожи на код для кнопки **Encode/Decode**, за исключением того, что нам нужно вычислить ключ на основе значения компонента SeekBar. Кроме того, нам нужно будет отобразить значение ключа в поле txtKey. Добавьте следующие строки кода в метод onProgressChanged():

```
    sb.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
        @Override
        public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
❶           int key = sb.getProgress() – 13;
            String message = txtIn.getText().toString();
            String output = encode(message, key);
            txtOut.setText(output);
❷           txtKey.setText("" + key);
    }
```

Во фрагменте ❶ мы создаем переменную с именем key. Мы получаем текущее значение компонента SeekBar из переменной sb и вычитаем из этого значения 13. Значения компонента SeekBar могут варьироваться только от 0 до 26, поэтому вычитание 13 даст диапазон от –13 до +13 для зашифровки и расшифровки сообщений. Следующие три строки идентичны тем, которые содержатся в коде обработчика событий для кнопки **Encode/Decode**. В этих строках принимается входящее сообщение, затем оно шифруется с помощью метода encode(), и наконец получившийся зашифрованный текст отображается в поле txtOut.

Строка ❷ является новой. Когда пользователь перемещает ползунковый регулятор `SeekBar`, мы хотим, чтобы значение поля `txtKey` изменилось. Поскольку переменная `key` является целым числом, мы добавляем к ней пустую строку (`""`), чтобы преобразовать результат в текстовую строку, которая будет показана пользователю. Для преобразования значения переменной `key` в строку можно было бы использовать команду `Integer.toString(key)`, то есть использовать метод `toString()` класса `Integer`, но добавление пустой строки к переменной типа `int` — это простой способ.

Запуск приложения на эмуляторе и на устройстве Android

Сохраните изменения, а затем повторите попытку запуска приложения в эмуляторе. Теперь вы можете двигать компонент `SeekBar` влево и вправо и сразу видеть изменение зашифрованного текста для каждого значения ключа, как показано на рис. 8.13.

Минуло уже несколько глав с тех пор, как мы подключали физическое устройство Android к нашему компьютеру, поэтому давайте кратко повторим эту процедуру. Сначала подключите устройство Android к компьютеру при помощи USB-кабеля. Когда появится всплывающее окно с вопросом, хотите ли вы разрешить USB-отладку, нажмите кнопку **Yes** (Да).

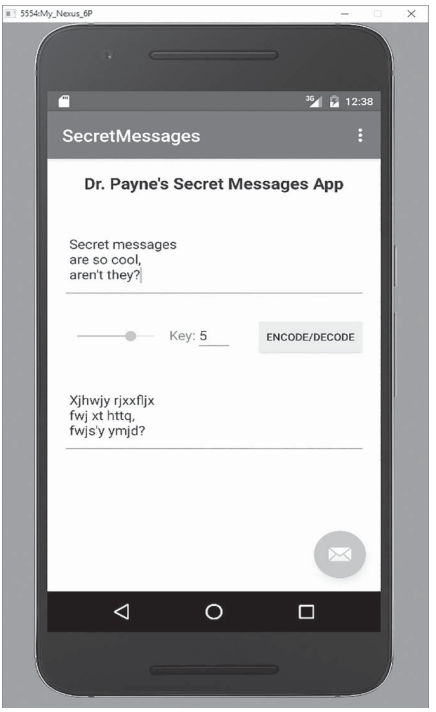


Рис. 8.13. Компонент SeekBar теперь полностью работоспособен в эмуляторе устройства Android. Значение ключа равно 5

В среде разработки Android Studio нажмите кнопку **Run** (Запустить), и вы увидите диалоговое окно **Select Deploy Target** (Выбор цели развертывания). (Возможно, вам придется закрыть приложение в окне эмулятора, чтобы открыть это диалоговое окно.) Найдите запись для подключенного устройства Android и нажмите кнопку **OK**.

Когда вы запустите приложение на своем реальном устройстве Android, вы заметите, что компонент `SeekBar` более отзывчив, и у вас есть доступ к нескольким параметрам при нажатии определенных частей приложения. Если вы щелкнете по зашифрованному тексту и выберете пункт **Select All** (Выбрать все), появится контекстное меню, показанное на рис. 8.14, где вы можете выбрать команды **Cut** (Вырезать), **Copy** (Копировать), **Share** (Поделиться) и **Assist** (Помочь).

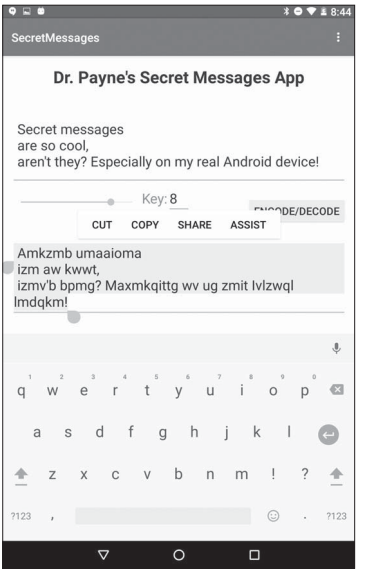


Рис. 8.14. Контекстное меню позволяет вам вырезать, копировать и делиться вашими зашифрованными сообщениями

Если выбрать пункт **Share** (Поделиться), выделенный текст можно будет отправить по электронной почте, с помощью службы текстовых сообщений, опубликовать в чате, в Twitter, в Facebook или передать через Bluetooth, либо использовать любое другое настроенное для подобного применения на вашем устройстве приложение, как показано на рис. 8.15 (слева). С помощью команды **Share** (Поделиться) вы сможете отправлять зашифрованные сообщения кому угодно, не выходя из приложения! Вы даже сможете отправить секретное сообщение другу, у которого есть компьютерная версия приложения «Секретные сообщения» с графическим интерфейсом, как показано на рис. 8.15 (справа). Все, что ему понадобится сделать, это скопировать зашифрованный текст из электронного письма, которое вы ему отправите, в запущенное приложение «Секретные сообщения», а затем использовать обратное значение секретного ключа для расшифровки

Рис. 8.15. Команда **Share** позволяет делиться сообщениями по электронной почте, с помощью службы текстовых сообщений, опубликовать его в Twitter, Facebook или с помощью другого приложения, настроенного для этих целей (слева). Выбор электронной почты автоматически создаст письмо с вашим зашифрованным текстом (справа)

Итак, теперь у нас есть полнофункциональное мобильное приложение «Секретные сообщения», которое позволяет нам делиться тайными сообщениями по электронной почте, с помощью службы текстовых сообщений или социальных сетей.

Чтобы завершить наше приложение, осталось лишь одна маленькая деталь: добавить пользовательское действие к всплывающей кнопке действия. Это позволит пользователям делиться своими сообщениями одним касанием пальца!

Бонус: настройка всплывающей кнопки действия

До сих пор мы игнорировали всплывающую кнопку действия (также известную как *значок fab*), символ скругленного конверта электронной почты в нижнем правом углу экрана приложения (рис. 8.16).

Значок всплывающей кнопки действия предоставляет пользователю возможность быстрого вызова определенной команды, например для обмена сообщениями по электронной почте или социальным сетям.

Среда разработки Android Studio автоматически добавляет в метод `onCreate()` код для всплывающей кнопки действия, который отображает всплывающее сообщение нового типа, называемое на сленге устройств Android «снэкбаром»:

```
FloatingActionButton fab = (FloatingActionButton) findViewById(R.
id.fab);
fab.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Snackbar.make(view, "Replace with your own action",
Snackbar.LENGTH_LONG)
```

Рис. 8.16. Всплывающая кнопка действия

```
                .setAction("Action», null).show();
        }
    });
```

Все, что нам нужно сделать, это заменить код в методе `onClick()` для пиктограммы `fab.setOnClickListener()` на код, который будет выполнять то, что нам надо.

Во-первых, удалите код между фигурными скобками метода `onClick()`. Затем введите следующий код в тело метода `onClick()`:

```
    FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
    fab.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
❶           Intent shareIntent = new Intent(Intent.ACTION_SEND);
❷           shareIntent.setType("text/plain");
❸           shareIntent.putExtra(Intent.EXTRA_SUBJECT, "Secret Message "+
                    DateFormat.getDateTimeInstance().format(new Date()));
❹           shareIntent.putExtra(Intent.EXTRA_TEXT, txtOut.getText().toString());
❺           try {
                startActivity(Intent.createChooser(shareIntent, "Share message…"));
                finish();
            }
❻           catch (android.content.ActivityNotFoundException ex) {
                Toast.makeText(MainActivity.this, "Error: Couldn't share.",
                        Toast.LENGTH_SHORT).show();
            }
        }
    });
```

Рассмотрим код построчно. В строке ❶ мы создаем объект `Intent` с именем `shareIntent`. Объект `Intent` в устройстве Android — это операция, которую мы хотим запустить, например электронная почта, Twitter или камера. В экземпляре `shareIntent` используется ключевое слово `ACTION_SEND`, означающее, что мы хотим передать данные в другое приложение на устройстве Android.

Мы устанавливаем для объекта `Intent` тип `text/plain` в строке ❷, который требуется для сообщений электронной почты, твитов или постов в социальных сетях, а затем добавляем строку темы к операции в блоке ❸. Если пользователь хочет поделиться своим сообщением по электронной почте или через другое приложение,

использующее строку темы, в ней будет написано «Секретное сообщение» и текущие дата и время. Класс `DateFormat` сформирует дату в текстовой форме на основе данных вашего региона, например `7 декабря 2017 года 4:33:12 PM`.

Код в строке ❹ отправляет закодированное сообщение из поля `txtOut` как тело сообщения электронной почты, твита или поста, в зависимости от того, какое приложение используется пользователем.

Далее следует команда `try` в блоке ❺. Все что угодно может пойти не так, когда мы пытаемся вызвать одно приложение из другого. Что делать, если у пользователя нет установленного приложения электронной почты? Что делать, если приложение занято или сеть отсутствует? Команда `try` нужна как раз для таких случаев. Первая строка кода внутри блока `try` пытается запустить выбранную пользователем операцию (например, электронную почту, программу отправки текстовых сообщений или Twitter), затем передает информацию выбранному приложению и закрывает эту операцию в следующей строке.

Если возникает ошибка или исключение, команда `catch` в блоке ❻ вызовет всплывающее сообщение с текстом: «Error: Couldn't share.»

Это все, что нам нужно для работы всплывающей кнопки действия. При наборе кода не забудьте импортировать новые классы, нажимая сочетание клавиш **Alt+Enter** (⌥ + ⏎ в операционной системе macOS). Помните, что, если среда разработки Android Studio подчеркивает новый класс красным цветом, часто это происходит из-за того, что класс не был импортирован, поэтому нажатие сочетания клавиш **Alt+Enter** (⌥ + ⏎ в операционной системе macOS) автоматически добавит корректную инструкцию импорта (как в главе 4).

После того как вы добавили код и импортировали классы, запустите приложение еще раз. После зашифровки сообщения скройте клавиатуру, нажав направленный вниз треугольник в нижней части экрана. Затем нажмите всплывающую кнопку действия в правом нижнем углу экрана, как показано на рис. 8.17 (слева). Выберите электронную почту из списка приложений в разделе **Share message...** (Поделиться сообщением...), и приложение должно запуститься, при этом в теме письма должен быть указан текст «Секретные сообщения» и текущие дата и время, а зашифрованное сообщение должно оказаться в теле письма, как показано на рис. 8.17 (справа).

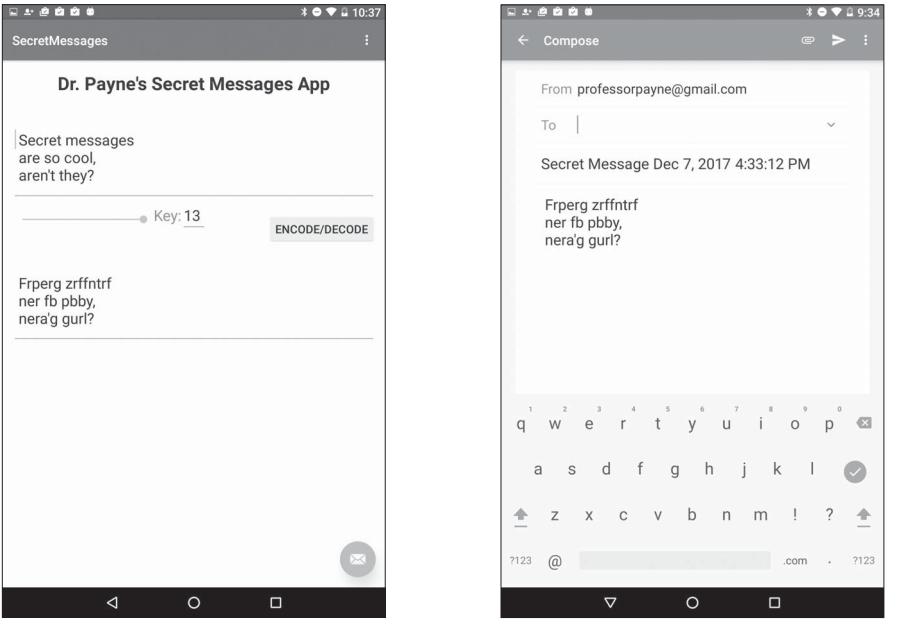


Рис. 8.17. При нажатии всплывающей кнопки действия в правом нижнем углу появляется меню выбора способа публикации (слева). Если будет выбрана электронная почта, то автоматически создастся письмо с темой «Секретные сообщения» и текущими датой и временем и зашифрованным сообщением в теле письма (справа)

Теперь мы можем отправлять секретные сообщения из нашей программы лишь одним касанием кнопки! Но чтобы сделать это приложение действительно достойным магазина Google Play Store, мы должны запрограммировать его *получать* секретные сообщения извне.

Получение секретных сообщений из других приложений

Давайте добавим последнее улучшение мобильной версии приложения «Секретные сообщения». Мы можем прямо из нашего приложения посылать сообщения по электронной почте, размещать их в Twitter и так далее. Но что делать, если мы хотим *получать* сообщения *из* писем, твитов и постов без копирования и вставки в приложение? Для решения этой задачи мы должны добавить наше приложение в список **Share** (Поделиться) на устройстве Android, чтобы другие приложения могли направлять текст

непосредственно в приложение «Секретные сообщения». Нам понадобится всего лишь несколько строк кода!

Это непростая тема, поэтому давайте разберем ее шаг за шагом. Во-первых, откройте файл *AndroidManifest.xml* вашего приложения, расположенный в папке *manifests* на панели **Project Explorer** (Обозреватель проекта). В файле манифеста хранится «сопутствующая» информация о вашем приложении, также как в судовом манифесте перечисляются все грузы, перевозимые судном. Вы можете объявить свойства для всего приложения и включить информацию о том, как устройство Android должно его запускать, так же, как вы декларируете содержимое посылки, отправляя ее за границу.

Для этого вам нужно будет отредактировать XML-код. Файл XML содержит *теги* для маркировки информации, хранящейся в файле. Обычно тег выглядит как его имя, заключенное в угловые скобки, например `<tag>`. В этом случае нам нужно изменить запись тега `intent-filter` манифеста, в которой есть информация о том, какие данные приложение будет принимать. Добавьте следующие пять строк ближе к концу вашего файла *AndroidManifest.xml*, перед тегом `</activity>`:

```
            <intent-filter>
                <action android: name=»android.intent.action.MAIN» />
                <category android: name="android.intent.category.LAUNCHER" />
            </intent-filter>
❶           <intent-filter>
❷               <action android: name="android.intent.action.SEND" />
❸               <category android: name="android.intent.category.DEFAULT" />
❹               <data android: mimeType="text/*" />
❺           </intent-filter>
        </activity>
    </application>
</manifest>
```

В строке ❶ мы добавляем новый элемент `intent-filter` в приложение. Раньше мы использовали объект `Intent` для отправки текста в другое приложение, такое как электронная почта, Twitter или Facebook. В этом случае мы создаем фильтр для *получения* объектов `Intent`, отправленных из других приложений на нашем смартфоне (строка ❷). (Мы использовали `Intent.ACTION_SEND` в коде для

всплывающей кнопки действия при отправке сообщений Intent в другие приложения, здесь мы делаем обратное действие.)

В строке ❸ применяется категория совместного использования DEFAULT (по умолчанию), чтобы добавить приложение «Секретные сообщения» в список приложений на вашем телефоне, которые могут обращаться ко внешней среде. С категорией DEFAULT ваше приложение может получать данные из любого приложения. (Существуют другие категории, позволяющие отфильтровать или ограничить типы приложений, которые могут отправлять данные в ваше приложение. Например, категория BROWSABLE используется приложениями, которые должны получать данные только из веб-браузера.) В строке ❹ мы сообщаем устройству Android, что приложение «Секретные сообщения» должно получать любые текстовые данные из других приложений. (Если вы пишете приложение, которое принимает изображения в качестве входных данных, для параметра mimeType может быть указано "image/*" или "image/png" для изображений только в формате PNG. Для видео следует указать "video/mp4" и так далее.)

Наконец, строкой ❺ мы закрываем элемент intent-filter в манифесте вашего приложения. Необходимо *закрывать* все XML-элементы, что обычно выполняется с помощью слеша и имени закрывающего XML-тега, например </intent-filter>. Тем самым компьютер понимает, что вы закончили код элемента intent-filter. Сохраните файл *AndroidManifest.xml*, а затем откройте файл *MainActivity.java* для редактирования исходного кода программы на языке Java.

В файле *MainActivity.java* найдите метод onCreate(). Сразу же после кода, который соединяет компоненты графического интерфейса с переменными от txtIn до btn, добавьте четыре приведенные ниже команды:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
    txtIn = (EditText)findViewById(R.id.txtIn);
    txtKey = (EditText)findViewById(R.id.txtKey);
    txtOut = (EditText)findViewById(R.id.txtOut);
    sb = (SeekBar)findViewById(R.id.seekBar);
```

```
        btn = (Button)findViewById(R.id.button);
❶       Intent receivedIntent = getIntent();
❷       String receivedText = receivedIntent.getStringExtra(Intent.EXTRA_TEXT);
❸       if (receivedText!= null)
❹           txtIn.setText(receivedText);
            btn.setOnClickListener(new View.OnClickListener() {
```

В строке ❶ мы создаем переменную с именем `receivedIntent`, чтобы принять входящее сообщение `Intent`, полученное из другого приложения. Затем мы получаем текст из сообщения `Intent` с помощью метода `getStringExtra()` в строке ❷. Части сообщения, отправленные `Intent`, называются *дополнительными* (extra) данными. Инструкция `if` в строке ❸ проверяет, чтобы значение переменной `receivedText` не оказалось `null` и, если это не так, меняет текст в текстовом поле `txtIn` в верхней части приложения секретных сообщений на полученный текст (строка ❹).

Это все, что необходимо сделать для приема сообщений из других приложений нашим приложением «Секретные сообщения»! Теперь вы можете не только отправлять зашифрованные сообщения из созданного приложения, но также напрямую расшифровывать сообщения из электронной почты, твитов и постов. На рис. 8.18 показано, как выглядит передача сообщения из электронной почты в приложение «Секретные сообщения» для быстрой расшифровки.

Выделите текст в электронной почте, Twitter или Facebook и нажмите кнопку **Share** (Поделиться). Приложение «Секретные сообщения» появится в списке приложений, которые могут принять сообщение. Выберите «Секретные сообщения» как приложение для приема сообщения, и выделенный текст отобразится во входном текстовом поле приложения «Секретные сообщения». Вы можете легко зашифровать или расшифровать его; просто помните, что, возможно, вам потребуется изменить значение ключа в зависимости от того, как было зашифровано сообщение.

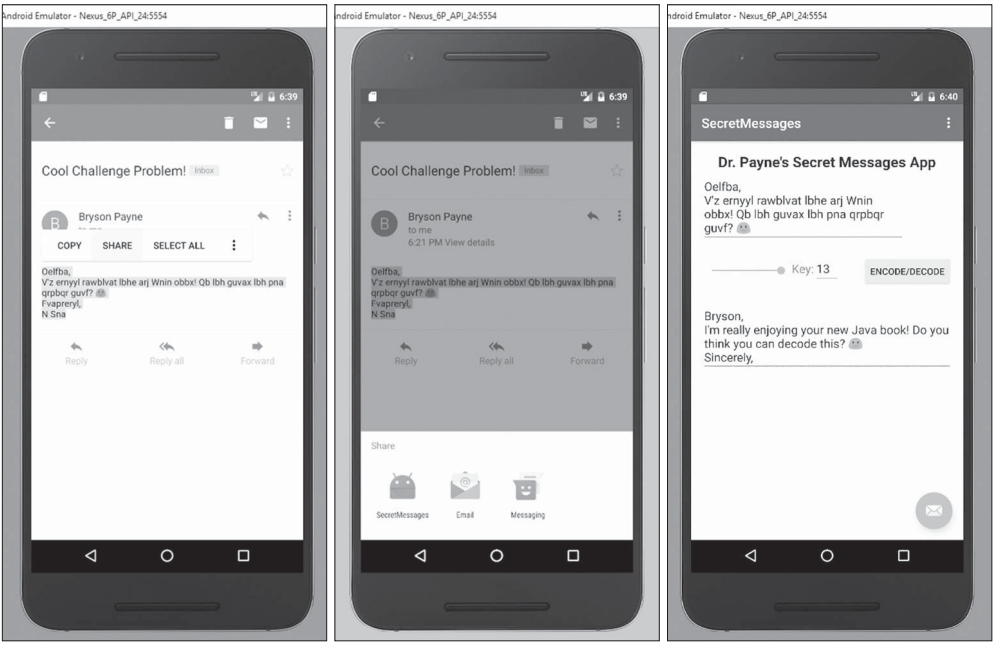


Рис. 8.18. Теперь приложение «Секретные сообщения» может обмениваться сообщениями с другими приложениями на вашем телефоне, включая электронную почту

Поздравляю! Вы создали мобильное приложение «Секретные сообщения». Играйте с ним и делитесь с друзьями!

Давайте обсудим, какие навыки, вы получили в этой главе. Кроме того, попробуйте выполнить пару дополнительных заданий.

Что вы изучили

В этом приложении «Секретные сообщения» мы повторно использовали код из версии с графическим интерфейсом для настольного компьютера. Однако мы доработали его так, что наше приложение стало по-настоящему законченным мобильным приложением. Вот некоторые из новых знаний и навыков, которые вы получили и укрепили в этой главе:

- написание обработчиков событий для компонентов устройства Android, таких как кнопки, `SeekBars` и всплывающая кнопка действия;
- настройка компонентов графического интерфейса устройства Android;

- использование полей многострочного текстового ввода в приложении для устройства Android;
- добавление компонента `SeekBar` и использование его подобно ползунковому регулятору;
- запуск приложений Android на эмуляторе и реальных устройствах;
- настройка всплывающей кнопки действия;
- создание пользовательских объектов `Intent` для вызова специальных функциональных возможностей, включая отправку данных в другие приложения;
- создание пользовательских фильтров `Intent` для получения текстовых (или графических/видео) данных, отправленных из других приложений.

Дополнительные задачи

Попробуйте выполнить следующие задания. Таким образом, вы повторите пройденный материал, попрактикуетесь в программировании и получите новые навыки. Если у вас возникнут проблемы, вы можете загрузить примеры решений на сайте по адресу **https://eksmo.ru/files/learn_java_easy_way.zip**.

Задача № 1: Создание кнопки для перемещения вверх

Эта задача состоит в том, чтобы создать кнопку **Move Up** ∧, аналогичную описанной в главе 7, которая перемещает текст из поля вывода зашифрованного сообщения в поле ввода исходного текста.

Поместите кнопку **Move Up** ∧ под текстовое поле `txtOut`. Затем добавьте обработчик событий, который получает текст из текстового поля `txtOut` и устанавливает его как содержимое текстового поля `txtIn`. В качестве бонуса в обработчик событий для кнопки **Move Up** ∧ можно добавить функцию установки противоположного значения компоненту `SeekBar`. Ключ шифрования `7` станет ключом дешифрования `-7`. В результате, нажатие кнопки будет перемещать зашифрованное сообщение наверх и автоматически расшифровывать его!

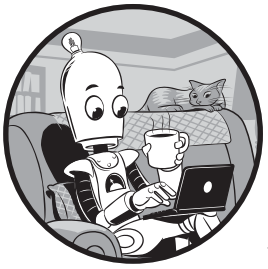
Задача № 2: Изменение прогресса компонента SeekBar.

Второе улучшение интерфейса, которое можно было бы сделать, состоит в изменении свойства прогресса (то есть значения) компонента `SeekBar` после того, как пользователь ввел значение ключа вручную в поле `txtKey`. Попробуйте!

Вместо создания обработчика событий для поля `txtKey` измените код кнопки **Encode/Decode**, чтобы значение прогресса компонента `SeekBar` соответствовало значению в поле `txtKey + 13`. Не забудьте, что значение компонента `SeekBar` варьируется от 0 до 26, а не от –13 до +13. Решение должно составить пару строк кода.

Глава 9

РАЗНОЦВЕТНЫЕ ПУЗЫРЬКИ С ПОМОЩЬЮ МЫШИ



В следующих трех главах мы создадим интерактивное анимированное приложение «Рисование пузырьков», которое позволит пользователю рисовать разноцветные пузырьки при помощи мыши на настольном компьютере и пальцами в мобильной версии. Пузырьки будут парить по экрану, отталкиваясь от его границ!

Первая версия приложения «Рисование пузырьков» будет выглядеть так, как показано на рис. 9.1, причем каждый пузырек будет окрашен в случайный цвет. Каждый раз, когда пользователь будет щелкать мышью и двигать ее в окне приложения, будут появляться разноцветные пузырьки. Также пользователь сможет изменять размер пузырьков, прокручивая колесико мыши вверх и вниз или используя жесты прокрутки на сенсорном экране.

Для создания приложения «Рисование пузырьков» мы будем использовать подход объектно-ориентированного программирования. Переменные, функции, циклы и условные конструкции, которые вы изучали до сих пор, являются элементами *процедурного программирования*. Процедурное программирование представляет

собой написание программ линейным способом, шаг за шагом, подобно следованию рецепту. Объектно-ориентированное программирование использует все эти концепции, но позволяет нам программировать гораздо более крупные и более сложные приложения, разбивая большой программный проект на более мелкие кусочки, называемые *объектами*.

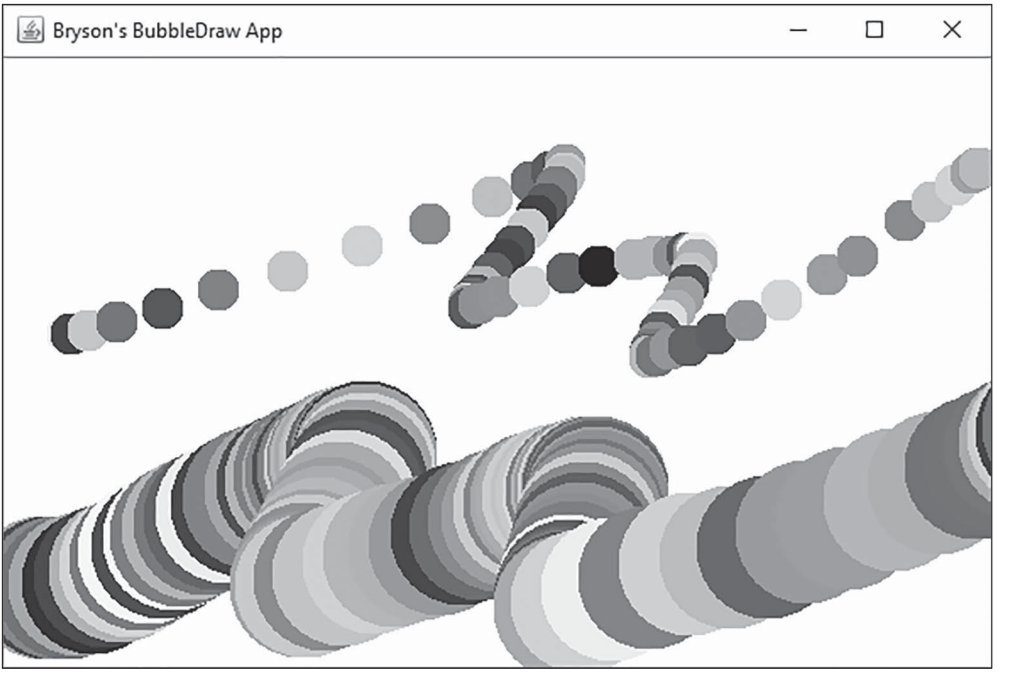


Рис. 9.1. Приложение «Рисование пузырьков» позволяет пользователю с помощью мыши рисовать разноцветные пузырьки

Например, пузырек является важной сущностью в приложении «Рисование пузырьков», поэтому пузырьки могут стать объектами, с которых мы начнем программировать это приложение. Во-первых, мы займемся проблемой определения того, что такое пузырек. Во-вторых, мы выберем способ хранения информации о множестве пузырьков и о том, как нарисовать пузырьки на экране. Наконец, мы добавим возможность создавать пузырьки, щелкая и передвигая мышью. Вместо написания отдельных фрагментов кода для управления каждым пузырьком индивидуально, мы напишем один блок кода, который будет применяться ко всем пузырькам-объектам.

Для начала разместим исходный код в двух файлах: один, *BubbleDraw*, будет использоваться для окна приложения под названием, а другой, *BubblePanel*, — для холста. Окно приложения расширит

уже знакомый нам класс `JFrame`, а холст внутри оконного фрейма будет использовать новый вид контейнера графического интерфейса пользователя под названием `JPanel`. Создав два отдельных файла, мы сможем повторно использовать наш холст в графическом приложении в главе 10. Давайте начнем программировать!

Создание файлов проекта «Рисование пузырьков»

В программе Eclipse выберите команду меню **File ⇒ New ⇒ Java Project** (Файл ⇒ Новый ⇒ Проект Java) и создайте новую папку проекта для приложения «Рисование пузырьков». Присвойте проекту имя **BubbleDraw** и нажмите кнопку **Finish** (Готово).

Разверните папку проекта *BubbleDraw* на панели **Project Explorer** (Обозреватель проекта), щелкните правой кнопкой мыши по папке *src* и выберите пункт **New ⇒ Class** (Создать ⇒ Класс). Создайте класс с именем `BubbleDraw` для окна приложения, с суперклассом `javax.swing.JFrame` и в разделе **Which method stubs would you like to create?** (Заглушку какого метода вы хотели бы создать?) выберите пункт **public static void main (String [] args)**. Затем нажмите кнопку **Finish** (Готово).

Теперь создадим холст `BubblePanel`. Щелкните правой кнопкой мыши по папке *src* и выберите пункт **New ⇒ Class** (Создать ⇒ Класс). Присвойте классу имя **`BubblePanel`** с суперклассом `javax.swing.JPanel`. Нажмите кнопку **Finish** (Готово).

Используя эти классы, мы создадим холст `BubblePanel`, который мы сможем расширить и использовать в других приложениях. Фрейм `BubbleDraw` будет контейнером, отображающим холст `BubblePanel`.

Создание фрейма `BubbleDraw`

Давайте начнем с настройки основного окна приложения в исходном коде *BubbleDraw.java*. Перейдите на вкладку *BubbleDraw.java* в верхней части области содержимого в среде разработки Eclipse. Вы должны увидеть следующий блок кода:

```
import javax.swing.JFrame;
public class BubbleDraw extends JFrame {
    public static void main(String[] args) {
    }
}
```

Окно приложения работает во фрейме `JFrame`, как и предыдущие приложения с графическим интерфейсом. В данном приложении нам нужно окно для отображения нашего холста `BubblePanel`. Мы не будем добавлять никакие другие компоненты графического интерфейса в эту первую версию приложения.

Создадим окно JFrame и добавим код установки для фрейма, как в предыдущих главах, но мы также добавим во фрейм холст `BubblePanel`, на котором будем рисовать пузырьки. Полный класс `BubbleDraw` будет выглядеть так, как показано в листинге 9.1:

```
import javax.swing.JFrame;
public class BubbleDraw extends JFrame {
    public static void main(String[] args) {
❶       JFrame frame = new JFrame("Ваше имя BubbleDraw App");
❷       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
❸       frame.getContentPane().add(new BubblePanel());
❹       frame.setSize(new java.awt.Dimension(600,400));
❺       frame.setVisible(true);
    }
}
```

Листинг 9.1. Класс BubbleDraw

Строка ❶ создает новый фрейм `JFrame` с заголовком. Вместо слов «Ваше имя» укажите свое имя. В строке ❷ мы устанавливаем операцию закрытия по умолчанию, для корректного выхода из приложения при закрытии окна пользователем. Строка ❸ создает холст `BubblePanel` как содержимое этого фрейма. Последние две строки задают размер окна ❹ и делают его видимым ❺.

Сохраните файл *BubbleDraw.java* и запустите его. Вы увидите серый фрейм Java с заголовком «*Ваше имя* BubbleDraw App». Теперь давайте создадим логику для холста `BubblePanel`.

Создание класса для пузырьков

Перейдите на вкладку файла *BubblePanel.java*. Этот файл будет содержать всю логику рисования пузырьков на экране. Наша первая задача будет состоять в создании класса `Bubble` для хранения цвета, размера и местоположения каждого пузырька на экране.

Определение пузырька

Причина построения класса весьма практична: в предшествующем процедурном подходе к программированию нам понадобилось бы завести отдельные переменные для каждой координаты (`x` и `y`), размера и так далее — для каждого пузырька. Например, нам понадобилась бы переменная `bubble1x` для координаты `x` первого пузырька, а также переменные `bubble1y`, `bubble1size` и `bubble1color` для этого пузыря. Это было бы не так плохо, но что, если мы двигаем мышью несколько секунд, и все это заканчивается тысячей пузырьков? Нам потребовалось бы 4000 переменных или даже больше! Вместо этого мы можем использовать свойства внутри класса для хранения этих значений для каждого пузырька.

Прежде чем мы начнем писать код, давайте разберемся, что такое пузырек. Как было показано на рис. 9.1, пузырек представляет собой цветной круг, при этом пузырьки могут быть разных размеров и располагаться в разных местах.

Все эти возможности являются атрибутами пузырьков в этом приложении. В объектно-ориентированном программировании мы используем существительные и прилагательные, которые описывают объект, для создания списка атрибутов при создании нового класса. Атрибуты хранятся как переменные класса.

Помимо атрибутов, класс также может содержать методы. Метод — это функция, связанная с конкретным классом. Методы позволяют классу производить какие-либо действия. Подумайте о том, что мы хотим делать с пузырьками в нашем приложении. Нам нужно создать пузырек, когда пользователь щелкает мышью или двигает ею. Кроме того, нам придется рисовать пузырек каждый раз, когда экран обновляется. Глаголы, такие как *создать* (*create*) и *рисовать* (*draw*), помогут нам определить методы, которые понадобятся в нашем классе пузырьков.

Когда мы зафиксируем все эти атрибуты и методы в одном классе, мы сможем описать любой пузырек, который захотим отобразить на экране. Именно так программист решает проблемы при объектно-ориентированном подходе к программированию: разбивает большие приложения на более мелкие части; создает классы, задавая вопросы о том, что содержит программа, а затем, определяет методы и атрибуты, исходя из того, что каждый объект нового класс должен делать и в какой информации он нуждается.

Давайте начнем писать класс `Bubble`. Мы сделаем это в классе `BubblePanel`, так как `BubblePanel` — это единственное место,

где нам нужно рисовать пузырьки. Начните программировать свой класс `Bubble` перед закрывающей скобкой в конце класса `BubblePanel` следующим образом:

```
import javax.swing.JPanel;
public class BubblePanel extends JPanel {
    private class Bubble {
    }
}
```

Обычно мы делаем внутренние и вспомогательные классы закрытыми, чтобы другие программы не могли обращаться к ним напрямую. Чтобы сделать `Bubble` внутренним классом, доступным только из `BubblePanel`, мы объявляем его закрытым (`private`). Эта методика называется *инкапсуляцией* и означает, что класс `BubblePanel` скрывает свою внутреннюю работу от других классов. Инкапсуляция является одним из основных принципов объектно-ориентированного программирования. Этот прием заслуживает внимания, поскольку подразумевает, что мы можем изменять внутреннее устройство класса `Bubble`, будучи уверенными, что не сломаем при этом другие части кода. Поскольку класс `Bubble` объявлен закрытым, мы знаем, что кода, зависящего от него вне класса `BubblePanel`, не существует. Это особенно важно для больших приложений с множеством классов, над которыми работают разные команды программистов.

Холст `BubblePanel` будет полагаться на класс `Bubble` для хранения информации об отдельных нарисованных пузырьках. У пузырька есть (`x`, `y`) координаты расположения на экране, размер и цвет, и мы можем реализовать эти атрибуты как переменные внутри класса `Bubble`. Координаты положения пузырька и его размер могут храниться как целочисленные значения:

```
import javax.swing.JPanel;
public class BubblePanel extends JPanel {
    private class Bubble {
        private int x;
        private int y;
        private int size;
    }
}
```

Мы создали две переменные x и y для хранения координат пузырька, а также переменную size. Эти атрибуты определены закрытыми, поэтому только сам класс Bubble может напрямую изменять их значения. Мы инкапсулировали всю информацию о пузырьках внутри класса Bubble и будем взаимодействовать с ними, используя только методы этого класса.

Как упоминалось ранее, каждый пузырек в нашем приложении может быть раскрашен в свой собственный цвет. В библиотеке java.awt есть класс Color, который на основе комбинации цветов *RGB* (*red-green-blue* — красный-зеленый-синий) воспроизводит любой цвет, который можно отобразить на нашем мониторе. Мы подробнее поговорим о модели RGB чуть позже в этой главе, когда напишем код для управления цветами пузырьков, а сейчас просто импортируем класс java.awt.Color в верхней части файла и добавим атрибут color в класс Bubble, например так:

```
import java.awt.Color;
import javax.swing.JPanel;
public class BubblePanel extends JPanel {
    private class Bubble {
        private int x;
        private int y;
        private int size;
        private Color color;
    }
}
```

После того как мы добавили необходимые атрибуты каждому пузырьку (x, y, size и color), мы можем начать программировать функции, позволяющие устанавливать значения этих атрибутов. Мы сделаем это, добавив методы внутри класса Bubble.

Проектирование методов для пузырьков

Нам нужно реализовать в классе Bubble два действия, которые мы превратим в методы: создание пузырька и рисование пузырька на нашем экране.

У методов создания объектов есть специальное имя: *конструкторы*. Конструктор создает объект, назначая значения его атрибутам. Когда мы присваиваем значения атрибутам внутри конструктора, мы их *инициализируем*. В классе Bubble мы собираемся

инициализировать атрибуты x, y, size и color для каждого нового пузырька.

Создание конструктора

Метод-конструктор начинается с ключевого слова public, за которым следует имя класса и пары круглых скобок. Параметры, которые нужно передать в создаваемый объект, помещаются в круглые скобки. Мы присвоим каждому пузырьку свои собственные значения координат x и y, а также size при создании, поэтому конструктор класса Bubble выглядит как в листинге 9.2.

```
    private class Bubble {
        private int x;
        private int y;
        private int size;
        private Color color;
❶       public Bubble(int newX, int newY, int newSize) {
❷           x = newX;
❸           y = newY;
❹           size = newSize;
        }
    }
```

Листинг 9.2. Конструктор класса Bubble

Координаты и размер каждого пузырька определяются пользователем, поэтому мы передаем координаты x и y, а также размер как целочисленные значения в конструктор Bubble() при создании каждого нового пузырька. Этими значениями являются три параметра, которые мы передаем в метод Bubble(). Они называются newX, newY и newSize (строка ❶). Позже в коде, когда мы будем обрабатывать созданные пользователем пузырьки, мы будем находиться за пределами закрытого класса Bubble. Это означает, что у нас не будет прямого доступа к атрибутам пузырька, поэтому мы не сможем просто присваивать значения x, y и size атрибутам каждого пузырька. Чтобы обойти эту проблему, мы задаем значения атрибутов через конструктор Bubble(), который принимает входные значения от пользователя, присваивает входные значения переменным newX, newY и newSize, а потом присваивает

введенные пользователем значения атрибутам пузырька в строках ❷, ❸ и ❹.

Также мы хотим, чтобы каждый пузырек был окрашен в случайный цвет. Поскольку цвет будет случайным, нам не нужно передавать его конструктору с помощью параметра. Вместо этого мы можем задавать цвет внутри конструктора.

Нам нужно создать случайные значения цветов RGB, чтобы получить случайный цвет для каждого пузырька. Цветовая модель RGB работает, комбинируя различные количества красного, зеленого и синего света, тем самым получая разные цвета на мониторе вашего компьютера. При программировании каждый из этих трех цветов представлен целым числом от 0 (цвет отсутствует) до 255 (максимальное количество возможного цвета). Чтобы получить конкретный цвет, нам нужно получить три целых числа от 0 до 255, разделить их запятыми и сгруппировать в круглых скобках. Например, чистый красный цвет имеет значение RGB (255, 0, 0), что означает максимальный красный, нет зеленого и нет синего. Желтый получается значениями в RGB (255, 255, 0), что означает максимальное количество красного и зеленого, но отсутствует синий. В общей сложности существует более 16 миллионов значений цветов RGB, которые могут быть созданы таким образом.

Ранее, когда нам требовалось одно число, мы генерировали случайные числа с помощью метода `Math.random()` класса `Math`, но в приложении «Рисование пузырьков» нам понадобятся три случайных числа (по одному для каждой части RGB), так что давайте рассмотрим новый способ генерации случайных значений.

Класс `java.util.Random` содержит несколько полезных методов, включая `nextInt()`, который позволит мгновенно генерировать случайное целое число меньше заданного без необходимости выполнять дополнительные расчеты или округление. Чтобы использовать класс `Random`, нам придется сначала импортировать его, в верхней части файла *BubblePanel.java*:

```
import java.util.Random;
import java.awt.Color;
import javax.swing.JPanel;
```

Поскольку `Random` является классом, для получения доступа к его функциям нам требуется создать новый объект или

переменную типа Random. Добавьте следующую строку в верхнюю часть класса BubblePanel:

```
import java.util.Random;
import java.awt.Color;
import javax.swing.JPanel;
public class BubblePanel extends JPanel {
    Random rand = new Random();
    private class Bubble {
```

Эта строка создает генератор случайных чисел rand, который позволит нам быстро генерировать случайные целые числа или числа с плавающей запятой. Мы инкапсулируем rand, помещая его в начало класса BubblePanel так, чтобы мы могли генерировать случайные числа как в окне для рисования, так и во внутреннем классе Bubble, но не извне BubblePanel.

Чтобы создать случайный цвет в языке программирования Java, мы можем использовать конструктор для класса java.awt.Color, который мы импортировали ранее. Конструктор принимает три целочисленных аргумента, значения которых лежат в диапазоне между 0 и 255 (значение красного, зеленого и синего цвета, соответственно). Добавьте следующий код в метод конструктора Bubble():

```
public Bubble(int newX, int newY, int newSize) {
    x = newX;
    y = newY;
    size = newSize;
    color = new Color(rand.nextInt(256),
            rand.nextInt(256),
            rand.nextInt(256));
}
```

Мы используем ключевое слово new с конструктором для класса Color для создания нового цвета, и каждое из значений цвета RGB будет случайным целым числом от 0 до 255. Каждый раз, когда вызывается метод nextInt(), он генерирует новое случайное целое число от 0 до переданного ему целого числа. В данном случае мы хотим, чтобы случайные значения цвета лежали в диапазоне от 0 до 255, поэтому мы передаем в него число 256, поскольку nextInt() генерирует целое число, не включая переданную ему верхнюю границу.

Программирование метода рисования пузырька

Сейчас у каждого пузырька есть свои координаты `x` и `y`, размер и случайный цвет. Давайте добавим возможность рисовать пузырьки на экране. Чтобы нарисовать яркую графику на экране, мы импортируем класс `java.awt.Graphics`. Этот класс `Graphics` содержит такие методы, как `setColor()` для выбора цвета краски, `drawRect()` для рисования прямоугольника и `fillOval()` для рисования залитого краской овала. Мы будем использовать метод `fillOval()` для рисования пузырьков, поэтому добавьте следующую инструкцию импорта в начало файла *BubblePanel.java*:

```
import java.awt.Graphics;
import java.util.Random;
import java.awt.Color;
import javax.swing.JPanel;
```

Затем добавим метод `draw()` во внутренний класс `Bubble` под конструктором:

```
    private class Bubble {
        private int x;
        private int y;
        private int size;
        private Color color;
        public Bubble(int newX, int newY, int newSize) {
            — пропуск—
        }
        public void draw(Graphics canvas) {
❶           canvas.setColor(color);
❷           canvas.fillOval(x - size/2, y — size/2, size, size);
        }
    }
```

Метод `draw()` принимает один параметр — переменную `canvas` (холст) класса `Graphics`. Внутри метода `draw()` строкой ❶ мы вызываем метод `setColor()` для переменной `canvas`. Тем самым мы задаем рисунку цвет, сохраненный в переменной color пузырька, который мы будем рисовать. В строке ❷ мы вызываем метод `fillOval()`, чтобы нарисовать закрашенный круг на экране.

Метод `fillOval()` принимает четыре параметра: координаты (`x` и `y`) верхнего левого угла прямоугольника, описывающего овал, а также ширину и высоту этого прямоугольника. Наши овалы являются кругами, поскольку ширина и высота описывающих их прямоугольников имеют одинаковую величину — размер пузырька в пикселях, которые мы сохранили в атрибуте `size`. В строке ❷ мы хотим, чтобы центр пузырька находился в точке (`x`, `y`), где пользователь щелкнул мышью, поэтому мы должны скорректировать верхний левый угол каждого пузырька, вычитая половину величины размера (`size/2`) из значения обеих координат.

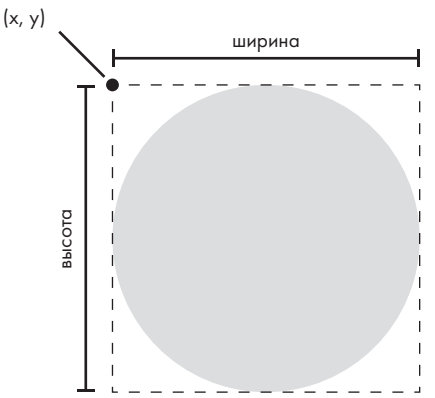


Рис. 9.2. Метод `fillOval()` принимает четыре параметра, координаты `x` и `y` верхнего левого угла, за которыми следуют ширина и высота овала

После добавления метода `draw()` у нас получился готовый класс `Bubble`, который может помнить местоположение, размер и цвет пузырька. Созданные нами методы класса можно использовать для создания новых пузырьков и рисования их на экране. Пришло время добавить логику в класс `BubblePanel`, чтобы эти методы можно было использовать для создания и рисования пузырьков каждый раз, когда пользователь щелкает мышью и двигает ее по экрану.

Хранение пузырьков в динамическом массиве `ArrayList`

Нам нужен способ хранения всех пузырьков, которые создает пользователь, щелкая мышью и двигая ее по экрану. Библиотеки языка программирования Java содержат несколько полезных *структур данных*, то есть классов, созданных для хранения групп объектов.

`Java.util.ArrayList` — это *динамическая* структура данных. Это означает, что она не только хранит коллекцию объектов, но также может увеличиваться или сокращаться в зависимости от потребностей программы. В приложении «Рисование пузырьков» мы не можем предсказать, сколько пузырьков нарисует пользователь, поэтому динамическая структура данных, такая как динамический массив `ArrayList`, является идеальным вариантом

для хранения всех созданных пользователем пузырьков. Динамический массив ArrayList — это гибкий способ хранения элементов при условии, что вы не знаете заранее, сколько элементов вам потребуется. Размер обычного массива в языке программирования Java фиксирован, а в динамический массив ArrayList можно добавлять новые пузырьки по каждому щелчку пользователя.

Для начала давайте получим доступ к типам данных ArrayList, импортировав класс java.util.ArrayList в верхней части файла *BubblePanel.java*:

```
import java.util.ArrayList;
import java.awt.Graphics;
import java.util.Random;
import java.awt.Color;
import javax.swing.JPanel;
```

Затем нам нужно объявить динамический массив ArrayList, в котором будут храниться объекты класса Bubble. Внутри класса BubblePanel добавьте следующее объявление:

```
public class BubblePanel extends JPanel {
    Random rand = new Random();
    ArrayList<Bubble> bubbleList;
```

Структура данных ArrayList принимает внутри угловых скобок < и > спецификатор типа, таким образом, сообщая компилятору языка Java, какой тип объектов будет храниться в динамическом массиве ArrayList. В ArrayList может храниться любой тип объектов, но в объявленном нами списке bubbleList будут храниться только объекты класса Bubble.

Затем давайте зададим переменную size для размера пузырька по умолчанию:

```
public class BubblePanel extends JPanel {
    Random rand = new Random();
    ArrayList<Bubble> bubbleList;
    int size = 25;
```

Переменная size определяет первоначальный диаметр пузырьков в пикселях. Я указал значение 25 пикселей, но вы можете выбрать больший или меньший диаметр по вашему желанию.

Добавление конструктора в класс BubblePanel

Мы объявили список bubbleList динамическим массивом ArrayList объектов Bubble, поэтому давайте добавим конструктор в класс BubblePanel для инициализации списка bubbleList и установки цвета фона холста.

Так же, как это было во внутреннем классе Bubble, конструктор класса BubblePanel использует модификатор доступа public и затем имя класса, за которым следует пара круглых скобок:

```
public class BubblePanel extends JPanel {
    Random rand = new Random();
    ArrayList<Bubble> bubbleList;
    int size = 25;
    public BubblePanel() {
    }
```

Обратите внимание, что после имени конструктора идут открывающая и закрывающая круглые скобки, поскольку конструктор — это метод. Внутри конструктора мы инициализируем список bubbleList, чтобы он был готов хранить гибкий список ArrayList объектов типа Bubble. Кроме того, мы окрашиваем фон холста в черный цвет:

```
    public BubblePanel() {
❶       bubbleList = new ArrayList<Bubble>();
❷       setBackground(Color.BLACK);
    }
```

Хотя я рекомендую черный фон для приложения, изображения в этой книге выполнены на белом фоне для удобства.

Как и при объявлении списка bubbleList, создавая новый список ArrayList в строке ❶, мы можем в угловых скобках (<Bubble>) указать тип объекта, который будем хранить в нем.

В строке ❷ мы можем использовать метод setBackground() напрямую, поскольку класс BubblePanel расширяет класс JPanel, а, как мы видели, у объектов класса JPanel есть фоновый цвет. В качестве значения цвета фона мы устанавливаем константу черного цвета Color.BLACK из ранее выбранного класса Color. Эта константа имеет RGB-значение (0, 0, 0).

Теперь, когда мы запустим приложение BubbleDraw и создадим холст BubblePanel, мы начнем с пустого списка пузырьков и черного экрана.

Сохраните текущую версию приложения. В следующем разделе мы наполним это окно яркими пузырьками!

Добавление метода рисования на экране

Далее нам нужно добавить метод рисования на экране всех пузырьков, хранящихся в динамическом массиве bubbleList. Во всех компонентах графического интерфейса пользователя в наборе инструментов javax.swing, включая класс JPanel, который мы расширили для холста BubblePanel, реализован метод paintComponent(), который рисует этот компонент на экране. Мы собираемся изменить или *переопределить* стандартный метод paintComponent() таким образом, чтобы класс BubblePanel рисовал все пузырьки, хранящиеся в списке bubbleList. Сначала нам нужно объявить метод paintComponent(). Поскольку мы переопределяем метод paintComponent(), который уже существует в панели JPanel, родительском классе класса BubblePanel, мы должны использовать одну и ту же *сигнатуру метода,* или первую строку кода, определяющего его. Это означает, что мы должны определить метод paintComponent() точно так же, как он объявляется в родительском классе, а именно как открытый метод void с одним параметром типа Graphics:

```
public class BubblePanel extends JPanel {
    – пропуск–
    }
    public void paintComponent(Graphics canvas) {
    }
```

Мы поместим метод paintComponent() чуть ниже конструктора BubblePanel(). Обратите внимание, что paintComponent() требует объекта Graphics, который мы назовем canvas точно так же, как и в методе draw() класса Bubble. Любой объект, который рисует на экране, может использовать объект Graphics для изменения цвета пикселей на экране компьютера.

Внутри этого метода мы хотим, чтобы родительский класс JPanel очистил холст и выполнил все другие стандартные

настройки, необходимые для рисования. Мы делаем это, вызывая `paintComponent()` из родительского класса:

```
public void paintComponent(Graphics canvas) {
    super.paintComponent(canvas);
}
```

Ключевое слово `super` просит компилятор языка Java вызвать исходный метод `paintComponent()` из класса `JPanel`. Это означает, что весь код в исходном методе `paintComponent()` будет перенесен в наш новый метод. Это полезная возможность объектно-ориентированного программирования: поскольку мы расширили класс `JPanel` для создания нового класса `BubblePanel`, мы можем воспользоваться всеми функциями, уже встроенными в класс `JPanel`, такими как очистка пикселей внутри окна при открытии приложения и подготовка холста `Graphics` для рисования. Мы просим класс `BubblePanel` *унаследовать* эти функции от класса `JPanel`.

После того как мы подготовили холст, наступает время пройти по списку пузырьков и нарисовать каждый из них на холсте `canvas`. Для этого мы будем использовать цикл `for` по-новому.

Первый раз вы видели цикл `for` в главе 6, когда мы перебирали все символы в строке для шифрования секретных сообщений. На этот раз мы используем специальную версию цикла `for`, называемую инструкцией `for each`. Эта версия предназначена специально для циклического перебора списка или коллекции объектов.

Давайте сначала посмотрим на рабочий код, а затем я разберу инструкцию `for each` по частям:

```
public void paintComponent(Graphics canvas) {
    super.paintComponent(canvas);
❶   for(Bubble b: bubbleList) {
❷       b.draw(canvas);
    }
}
```

Вы можете прочитать инструкцию `for each` в строке ❶ следующим образом «Для каждого объекта `b` типа `Bubble` в списке `bubbleList`». Можно сказать, что это инструкция `for each`, а не обычный цикл `for`, во-первых, из-за двоеточия в середине и, во-вторых, потому что у него нет трех частей, которые мы видели в циклах `for` в главе 6: *инициализация*, *условие* и *обновление*. Java

использует одно и то же ключевое слово for и круглые скобки для обоих циклов, но инструкция `for each` предназначена только для коллекций объектов, таких как массивы, динамические массивы `ArrayList` и так далее.

Для каждого объекта `Bubble b` в динамическом массиве `ArrayList bubbleList` цикл вызовет метод `b.draw(canvas)` (строка ❷) для рисования этого конкретного пузырька на экране. Когда цикл выполняется в первый раз, `b` указывает на первый объект `Bubble` в списке `bubbleList`, и при каждой итерации цикла, `b` указывает на следующий пузырек в списке. Вызов метода `b.draw(canvas)` говорит о том, что пузырек должен нарисовать сам себя на холсте `canvas`.

Этот цикл `for each` нарисует на экране каждый пузырек из списка `bubbleList`. Единственная проблема состоит в том, что у нас пока нет пузырьков для тестирования. Давайте создадим несколько случайных пузырьков, чтобы посмотреть, как работает приложение, прежде чем мы перейти к созданию пузырьков с помощью мыши.

Проверка класса `BubblePanel`

Чтобы протестировать нашу работу, прежде чем мы добавим взаимодействие с мышью, напишем тестовый метод, который рисует 100 пузырьков случайных размеров по всему окну приложения. Тем самым мы сможем на промежуточном этапе проверить работоспособность приложения. Мы сможем раньше отладить ошибки в коде и увидеть, как приложение будет выглядеть, когда мы его доделаем.

Вызовите новый метод `testBubbles()`. Поместите его после метода `paintComponent()` и перед строкой `private class Bubble`, как показано в листинге 9.3.

```
        public void paintComponent(Graphics canvas) {
            — пропуск—
            }
        }
❶       public void testBubbles() {
❷           for(int n = 0; n < 100; n++) {
❸               int x = rand.nextInt(600);
❹               int y = rand.nextInt(400);
```

```
❺              int size = rand.nextInt(50);
❻              bubbleList.add(new Bubble(x, y, size));
           }
❼          repaint();
       }
       private class Bubble {
```

Листинг 9.3. Код метода `testBubbles()`

В строке ❶ мы объявляем метод `testBubbles()` с модификатором доступа `public` и типом возвращаемого значения `void`, то есть никакой информации назад в программу он не возвращает. Затем мы используем обычный цикл `for` (строка ❷) для прохода от `n = 0` до `99`, то есть всего 100 итераций. Это означает, что мы создадим 100 пузырьков, и для каждого пузырька нам понадобится его местоположение (`x`, `y`) и его размер (ширина и высота в пикселях).

В начале этой главы мы задали для фрейма приложения «Рисование пузырьков» размер `600` пикселей в ширину и `400` пикселей в высоту, поэтому для определения положения пузырька нам требуется значение `x` от `0` до `600` и значение `y` от `0` до `400`. Внутри цикла `for` мы используем генератор случайных чисел `rand` и получаем случайное целое число от `0` до `600` и сохраняем его в переменной `x` (строка ❸). Таким образом, мы получаем координату `x` центра пузырька на экране. В строке ❹ мы генерируем значение координаты `y` от `0` до `400` и сохраняем его в `y`. Затем, генерируя случайное число от `0` до `50` в строке ❺, получаем размер пузырька в пикселях. На последнем шаге внутри цикла создается новый объект `Bubble` с использованием только что сгенерированных случайных значений `x`, `y` и `size`. В строке ❻ создается новый объект `Bubble` и добавляется в динамический массив `ArrayList bubbleList`.

В строке ❼ мы вызываем метод `repaint()`. Обычно, прежде чем нарисовать новый элемент компьютерной графики, мы должны очистить экран, рисуя пустой черный фон, но в нашем случае метод `repaint()` выполняет это автоматически. Обратите внимание, что нам также не нужно было очищать фон экрана внутри метода `paintComponent()`; все, что нам нужно было сделать, это нарисовать пузырьки из списка `bubbleList`. Метод `repaint()` выполняет перерисовку фона, а также вызывает метод `paintComponent()`, поэтому мы будем вызывать его при каждой необходимости обновить или перерисовать экран.

Перед тем как мы наконец-то сможем протестировать приложение, нужно выполнить еще один шаг: нам надо вызвать метод `testBubbles()` из конструктора `BubblePanel()`. Добавьте следующую строку кода в конструктор `BubblePanel`:

```
public BubblePanel() {
    bubbleList = new ArrayList<Bubble>();
    setBackground(Color.BLACK);
    testBubbles();
}
```

Сохраните файл *BubblePanel.java*, а затем перейдите на вкладку *BubbleDraw.java*. Сохраните этот файл и нажмите кнопку запуска в программе Eclipse. Первый раз во время компиляции и запуска вам нужно запускать файл с вкладки *BubbleDraw.java*, поскольку `BubbleDraw` содержит метод `main()`, который запускает программу. Вы должны увидеть окно, полное разноцветных пузырьков, похожее на рис. 9.3.

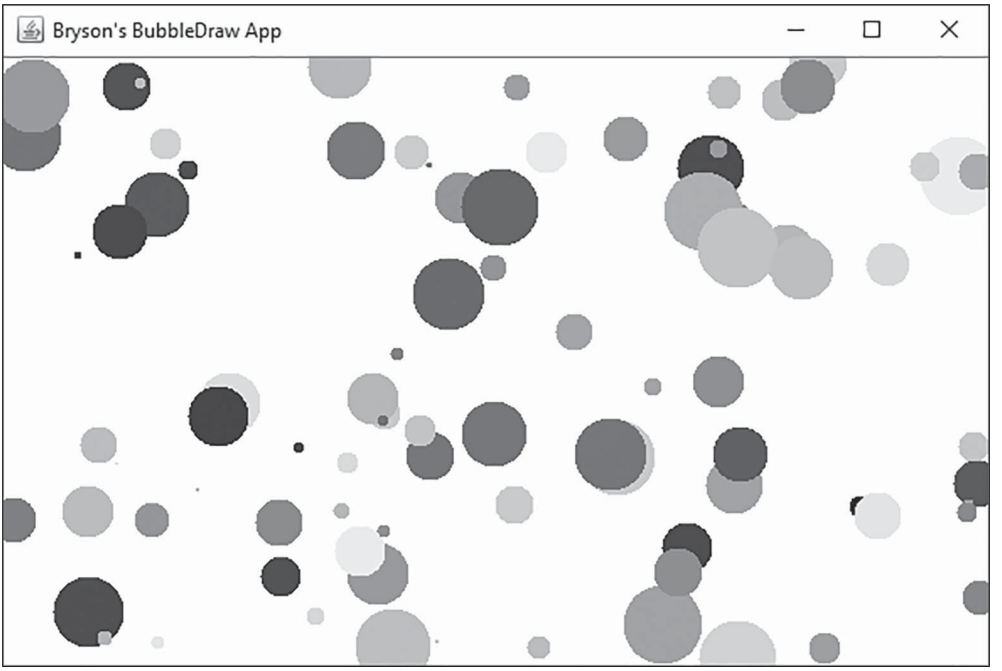


Рис. 9.3. Каждый раз, когда вы запускаете эту версию приложения «Рисование пузырьков», метод `testBubbles()` будет рисовать на экране 100 пузырьков

Если вам нравится внешний вид случайных пузырьков, потратьте немного времени и поиграйте с числами в методе

`testBubbles()`. Посмотрите, можно ли нарисовать 200, 500 или даже 1000 пузырьков вместо 100. Сделайте пузырьки больше, либо генерируя большее случайное число для атрибута `size`, либо добавив некоторое значение к атрибуту `size` уже после генерации случайного числа. Поиграйте со значениями `x` и `y` так, чтобы все пузырьки целиком помещались на экране и не обрезались по краям.

Изменяйте метод `testBubbles()` так, как хотите; это отличная возможность попробовать новые вещи и сразу увидеть визуальный эффект от каждого изменения. Мы закомментируем метод `testBubbles()` в следующем разделе, после того как добавим взаимодействие с мышью, поэтому вы можете свободно экспериментировать с методом `testBubbles()`, не боясь испортить остальную часть программы.

Обработка событий мыши

Цель приложения «Рисование пузырьков» — разрешить пользователю рисовать пузырьки с помощью мыши. В последнем разделе мы увидели, что часть программы, отвечающая за рисование пузырьков, работает. Теперь все, что нам нужно сделать, это добавить взаимодействие с мышью.

Мы будем использовать слушатели событий, чтобы приложение могло обрабатывать щелчки мыши, движение мыши и даже прокрутку колесика мыши. Мы добавляли слушатели событий в наши графические приложения в главах 3 и 7, используя анонимные внутренние классы для обработки щелчков по кнопкам, изменений положения ползунковых регуляторов и значений в текстовых полях. Однако если бы мы стали использовать анонимные внутренние классы в этом приложении, нам пришлось бы делать три отдельных слушателя для каждого типа событий, и их было бы сложно отслеживать. Кроме того, в двух из этих событий, нажатии и перемещении мыши, мы хотим, чтобы результат был один и тот же: приложение должно добавлять пузырьки каждый раз, когда пользователь щелкает мышью на экране *и* когда пользователь перетаскивает мышь по экрану. Было бы намного удобнее использовать один блок кода, который мы можно было бы присоединить к обоим событиям, чем писать слушатель для одного события, а затем копировать и вставлять код в другой. Таким образом, чтобы сделать приложение более управляемым, мы собираемся создать один именованный слушатель событий, который будет реагировать на все три типа событий.

Создание слушателя событий многоразового использования

Прежде чем мы сможем создать наш слушатель событий, нам нужно импортировать еще одну дополнительную библиотеку `java.awt.event.*`. Добавьте в верхней части файла *BubblePanel.java*:

```
import java.awt.event.*;
import java.util.ArrayList;
import java.awt.Graphics;
import java.util.Random;
import java.awt.Color;
import javax.swing.JPanel;
```

Наши предыдущие инструкции `import` импортировали за раз только один класс, а эта инструкция добавит все классы библиотеки `java.awt.event`. Символ подстановки, звездочка (`*`) означает, что мы хотим задействовать все классы верхнего уровня библиотеки `java.awt.event`, включая все события мыши, слушатели и многое другое. Мы могли бы импортировать каждый класс отдельно, по мере необходимостииспользования, но импорт всей библиотеки `java.awt.event.*` облегчит написание программы, позволив нам сосредоточиться на программировании вместо того, чтобы переключаться вверх и вниз в голову файла и обратно каждый раз, когда нам необходимо использовать новый класс событий.

Давайте начнем программирование закрытого, неанонимного внутреннего класса, для прослушивания событий мыши. Мы назовем его `BubbleListener`, поскольку он будет обрабатывать все события, связанные с пузырьками в классе `BubblePanel`. Добавьте этот класс под методом `testBubbles()`, но выше закрытого класса `Bubble`. Программисты, работающие на языке Java, обычно добавляют классы-слушатели в нижней части файла класса, вместе со всеми остальными вспомогательными классами — эта конвенция (соглашение) просто помогает быстро найти код слушателя для отладки или модификации файла.

```
        repaint();
    }
    private class BubbleListener extends MouseAdapter {
    }
    private class Bubble {
```

Класс `BubbleListener` расширяет класс `MouseAdapter`, который обрабатывает события мыши. Мы уже использовали ключевое слово `extends` для создания нового типа `JFrame`, который наследовал все возможности и функции его родительского класса. Точно так же класс `BubbleListener` наследует все функции прослушивания событий мыши класса `MouseAdapter`. Этот класс адаптер включает в себя возможность обработки событий `MouseListener` для щелчков, событий `MouseMotionListener` для перемещения мыши и событий `MouseWheelListener` для прокрутки колеса мыши или сенсорной панели.

Мы будем добавлять эти обработчики событий в программу постепенно, шаг за шагом и после каждого добавления проверять приложение, чтобы понять, как работает именно эта функция.

Обработка щелчков и перемещения мыши

Реализация обработки событий мыши состоит из двух шагов. Во-первых, мы должны добавить код в класс `BubbleListener` для обработки отдельного события, например, нажатия кнопки мыши — `mousePressed()`. Затем мы должны добавить слушатель в конструктор `BubblePanel()`, чтобы экран знал, что надо прослушивать этот тип события, и, когда оно произойдет, вызвать класс `BubbleListener` для его обработки. Давайте сделаем это для первого случая, когда пользователь нажимает кнопку мыши, чтобы нарисовать пузырек.

Прослушивание событий кнопки мыши

Существует три события, которые мы можем прослушивать для кнопок мыши:

`mousePressed()` Происходит, когда пользователь нажимает любую кнопку мыши;
`mouseReleased()` Происходит, когда пользователь отпускает кнопку;
`mouseClicked()` Происходит, когда пользователь нажимает и отпускает кнопку.

Для приложения «Рисование пузырьков» мы будем использовать обработчик `mousePressed()`, чтобы приложение рисовало пузырек, как только пользователь нажал кнопку мыши. Сигнатура

для обработчика события `mousePressed()` выглядит следующим образом:

```
private class BubbleListener extends MouseAdapter {
    public void mousePressed(MouseEvent e) {
    }
}
```

Правописание и регистр символов особенно важны, когда вы пишете обработчики событий, поскольку у них уже есть встроенные имена с определенными правилами регистра букв. Убедитесь, что название метода `mousePressed()` написано точно.

Обработчик событий `mousePressed()` входит в класс `BubbleListener` и должен быть объявлен открытым (`public`) с типом возвращаемого значения `void` для соответствия методу `mousePressed()` в классе `MouseAdapter`. Обратите внимание, что он также принимает параметр типа `MouseEvent`. Все события мыши получают информацию о местоположении указателя мыши на экране, когда произошло некоторое событие. Координаты `x` и `y` события мыши хранятся в объекте `MouseEvent`, и мы можем их получить с помощью методов `getX()` и `getY()`.

Следующий код добавляет пузырек в список `bubbleList` с координатами позиции, в которой пользователь щелкнул мышью, а затем перерисовывает экран, чтобы пузырек появился.

```
private class BubbleListener extends MouseAdapter {
    public void mousePressed(MouseEvent e) {
❶       bubbleList.add(new Bubble(e.getX(), e.getY(), size));
❷       repaint();
    }
}
```

В строке ❶ мы создаем новый объект `Bubble` с координатами позиции (`x`, `y`), в которой пользователь щелкнул мышью, `e.getX()` и `e.getY()`. Класс `MouseEvent` имеет несколько свойств и методов для работы с событиями мыши, например, выяснение того, какая кнопка была нажата и в каком месте экрана находится указатель мыши. Как я только что отметил,

методы `getX()` и `getY()` сообщают нам координаты `x` и `y` события мыши, такого как щелчок или перетаскивание. Посмотрите в класс `Bubble`, и вы увидите, что метод конструктора, который мы там написали, требовал трех параметров: `int newX`, `int newY` и `int newSize`. Таким образом, для создания нового объекта `Bubble` мы должны передать в конструктор параметры `e.getX()`, `e.getY()` и `size`. После создания пузырька мы добавляем его в динамический массив ArrayList с пузырьками с помощью метода `bubbleList.add()`.

В строке ❷ мы вызываем метод `repaint()` для обновления экрана и рисования обновленного списка `bubbleList` на холсте.

Наш обработчик события `mousePressed()` готов, но еще нужно сделать так, чтобы приложение прослушивало события `mousePressed()` и отправляло их в класс `BubbleListener`. Мы должны инструктировать класс `BubblePanel` добавить `BubbleListener` в качестве слушателя событий мыши.

Внесите два изменения в конструктор `BubblePanel()`. Во-первых, закомментируйте строку `testBubbles()`, поместив в ее начало два слеша. Во-вторых, добавьте вызов `addMouseListener()`, чтобы использовать класс `BubbleListener` для обработки событий мыши. Обновленный конструктор должен выглядеть так:

```
    public BubblePanel() {
        bubbleList = new ArrayList<Bubble>();
        setBackground(Color.BLACK);
❶       // testBubbles();
❷       addMouseListener(new BubbleListener());
    }
```

Закомментировав вызов метода `testBubbles()` в строке ❶, мы, с одной стороны, сохранили его на тот случай, если мы захотим снова нарисовать случайные тестовые пузырьки, а с другой, предотвратили его выполнение для того, чтобы была возможность протестировать интерактивное приложение «Рисование пузырьков», используя мышь. Строка ❷ передает указание панели `BubblePanel` прослушивать события мыши и, при возникновении, отправлять их в класс `BubbleListener`.

Внеся эти изменения, вы можете запустить приложение «Рисование пузырьков» и использовать кнопку мыши для размещения пузырьков в любой позиции, где вы щелкнете, как показано на рис. 9.4.

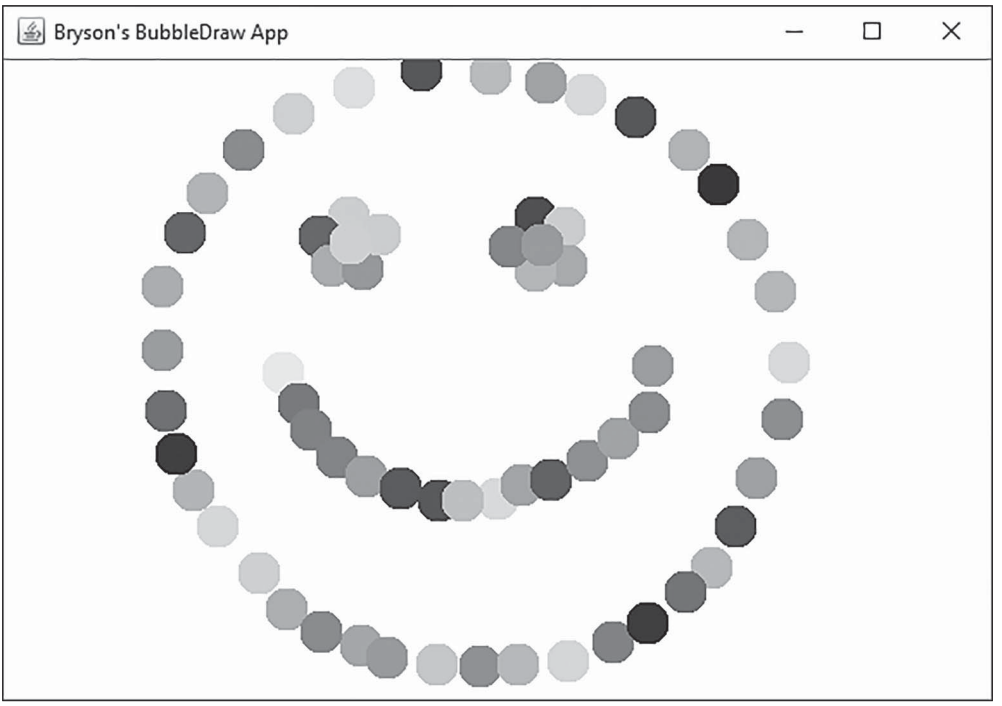


Рис. 9.4. После добавления обработчика **mousePressed()** и установки **BubbleListener** в качестве слушателя событий мыши в приложении вы сможете рисовать, щелкнув в любом месте экрана

Отлично! Вы можете многократно щелкать мышью, чтобы нарисовать на экране фигуры и узоры, однако было бы легче рисовать пузырьки, просто перетаскивая мышь с зажатой клавишей. Давайте это реализуем.

Прослушивание событий перемещения мыши

События перемещения мыши — это тип событий, который отличается от нажатия кнопок мыши, но они все равно могут обрабатываться в классе, который расширяет класс `MouseAdapter`. Например, в классе `BubbleListener`. Два типа событий движения мыши — `mouseMoved()` и `mouseDragged()`.

Событие `mouseMoved()` происходит тогда, когда мышь перемещается в окне. Событие `mouseDragged()` происходит тогда, когда мышь перемещается при нажатой кнопке. Поскольку мы хотим рисовать пузырьки только тогда, когда пользователь зажимает кнопку и перетаскивает мышь, мы сначала реализуем обработчик события `mouseDragged()` в классе `BubbleListener`, а затем добавим

слушатель движения мыши в конструктор класса `BubblePanel`, чтобы его активировать.

Добавьте обработчик события `mouseDragged()` в класс `BubbleListener`, как показано ниже:

```
private class BubbleListener extends MouseAdapter {
    public void mousePressed(MouseEvent e) {
        — пропуск—
    }
    public void mouseDragged(MouseEvent e) {
        bubbleList.add(new Bubble(e.getX(), e.getY(), size));
        repaint();
    }
}
```

Вы наверняка заметили, что код выглядит почти так же, как и в обработчике события `mousePressed()`, за исключением имени `mouseDragged()`. Это потому, что они оба обрабатывают события `MouseEvents`. Метод `mousePressed()` обрабатывает события нажатия кнопки мыши, а `mouseDragged()` вызывается каждый раз, когда пользователь перетаскивает мышь. В этом приложении оба события должны вести себя одинаково, добавляя пузырьки в список `bubbleList` и вызывая функцию `repaint()`.

Теперь добавьте класс `BubbleListener` в качестве слушателя движения мыши в конструктор `BubblePanel()` следующим образом:

```
public BubblePanel() {
    bubbleList = new ArrayList<Bubble>();
    setBackground(Color.BLACK);
    // testBubbles();
    addMouseListener(new BubbleListener());
    addMouseMotionListener(new BubbleListener());
}
```

Еще раз попробуем запустить приложение для тестирования этого нового слушателя событий. Сохраните код, нажмите кнопку запуска, затем нажмите кнопку мыши и, удерживая ее, перетаскивайте мышь. У вас должно получиться нечто похожее на изображение на рис. 9.5.

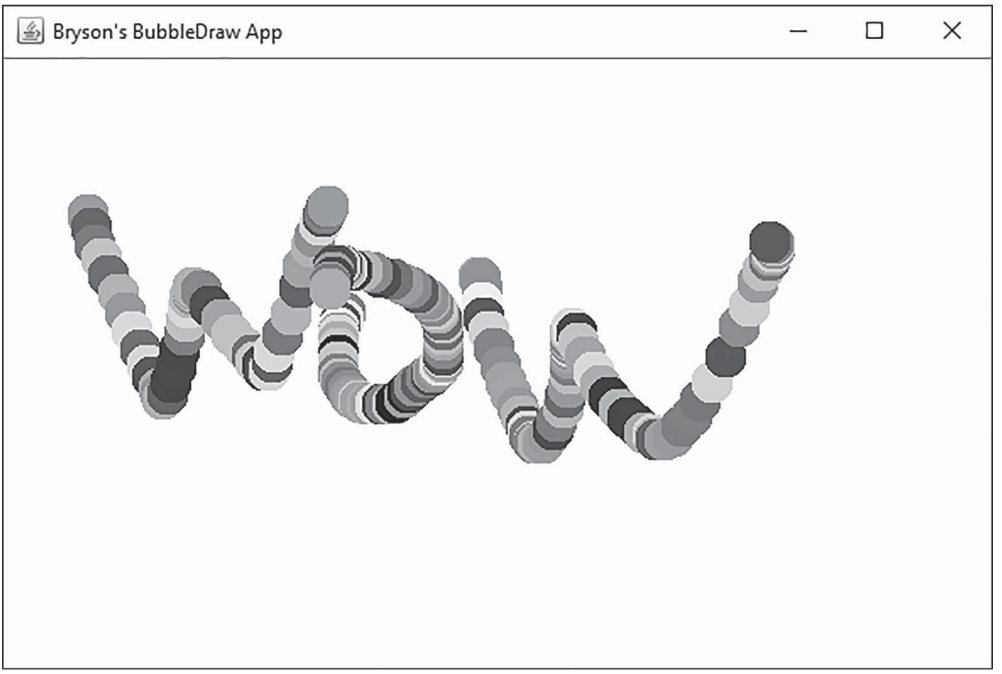


Рис. 9.5. Теперь вы можете нажать кнопку мыши и перетащить ее, тем самым рисуя непрерывные потоки пузырьков

Впечатляюще, не правда ли? Просто добавив в графическое приложение `BubblePanel` два обработчика событий для кнопки мыши и движения мыши, мы создали на языке программирования Java интерактивное пестрое приложение для рисования.

Бонус: обработка событий `MouseWheel`

На данный момент мы рассмотрели, как работают в нашем приложении два небольших, но мощных обработчика событий мыши: нажатия кнопки и перетаскивания мыши с зажатой кнопкой. В `MouseAdapter` существует еще один класс событий

: `MouseWheelEvent`. В зависимости от вашей системы, у вас может быть колесико на вашей мыши или сенсорная панель, которые можно использовать для прокрутки вверх и вниз в документах и на веб-страницах. Или у вас может быть ноутбук с сенсорной панелью, которая позволяет вам прокручивать с помощью жестов, например, смахивание двумя пальцами на MacBook или вертикальное смахивание вдоль крайнего правого края сенсорной панели на большинстве ноутбуков под Windows.

Компилятор языка Java интерпретирует любое событие прокрутки как `MouseWheelEvent`. Мы можем добавить слушатель `mouseWheelMoved()` в класс `BubbleListener` для обработки событий колесика мыши или жестов прокрутки сенсорной панели следующим образом:

```
    private class BubbleListener extends MouseAdapter {
        public void mousePressed(MouseEvent e) {
            — пропуск—
        }
        public void mouseDragged(MouseEvent e) {
            — пропуск—
        }
❶       public void mouseWheelMoved(MouseWheelEvent e) {
❷           size += e.getUnitsToScroll();
        }
    }
```

В строке ❶ метод `mouseWheelMoved()` принимает событие `MouseWheelEvent`. В строке ❷ мы получаем количество единиц, на которое колесико мыши прокручивалось от положения `MouseWheelEvent e` с помощью метода `e.getUnitsToScroll()`, и мы добавляем это число к атрибуту `size`. Число, возвращаемое методом `getUnitsToScroll()`, может быть положительным или отрицательным, в зависимости от операционной системы и от того, прокручивалось ли колесико вверх или вниз.

Отличия в прокручивании в разных операционных системах

Когда дело доходит до прокрутки, операционные системы Windows, macOS и Linux ведут себя по-разному. В операционной системе macOS прокрутка или жестикуляция вверх возвращает положительное значение, поэтому код, который мы только что написали, увеличит размер пузырька, когда пользователь прокручивает или смахивает вверх, и уменьшает размер, когда пользователь прокручивает вниз. В операционной системе Windows и большинстве приложений под Linux прокрутка с помощью колеса мыши вверх перемещает веб-страницу или документ вниз, возвращая

отрицательное число из метода `getUnitsToScroll()`. Это означает, что прокрутка вверх приведет к уменьшению пузырьков, а прокрутка вниз сделает их более крупными. Если вы хотите, чтобы ваше приложение работало одинаково во всех трех операционных системах, измените код на следующий:

```
public void mouseWheelMoved(MouseWheelEvent e) {
    if(System.getProperty("os.name").startsWith("Mac"))
        size += e.getUnitsToScroll();
    else
        size -= e.getUnitsToScroll();
}
```

Инструкция `if` получает имя операционной системы с помощью метода `System.getProperty("os.name")` и проверяет, начинается ли имя со строки `"Mac"`. Если это так, мы прибавим число, полученное с помощью метода `getUnitsToScroll()` к величине атрибута `size`. В противном случае мы его вычтем, тем самым изменив для операционных систем Windows и Linux значение прокрутки: прокрутка вверх будет соответствовать положительному значению, вниз — отрицательному.

Наконец, добавьте слушатель событий колеса мыши в конструктор `BubblePanel()` под двумя предыдущими слушателями:

```
public BubblePanel () {
    bubbleList = new ArrayList<Bubble>();
    setBackground(Color.BLACK);
    // testBubbles();
    addMouseListener(new BubbleListener());
    addMouseMotionListener(new BubbleListener());
    addMouseWheelListener(new BubbleListener());
}
```

Сохраните и запустите приложение, и вы сможете изменить размер пузырька, прокручивая колесико мыши или смахивая вверх или вниз на вашей сенсорной панели. У вас должно получиться похоже на изображение на рис. 9.6.

Первая версия приложения «Рисование пузырьков» завершена! Вы можете щелкать мышью и перетаскивать ее, рисуя на экране

разноцветные пузырьки, прокручивать вверх и вниз, чтобы изменять размер пузырьков, развернуть окно для полноэкранного просмотра и многое другое.

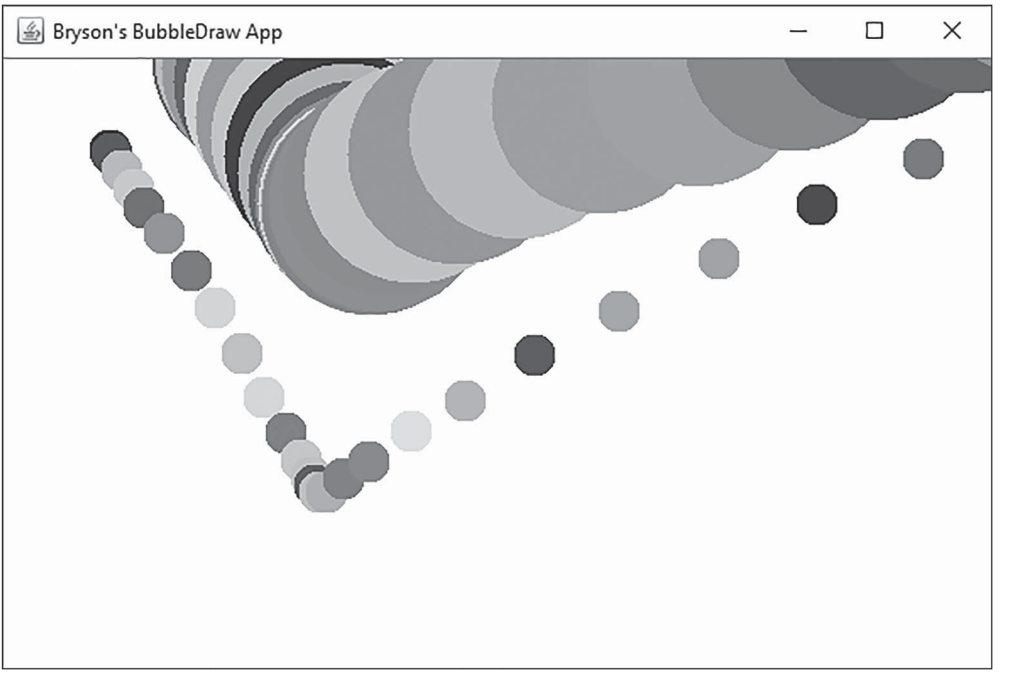


Рис. 9.6. Используйте колесико мыши или сенсорную панель для изменения размера пузырьков

Что вы изучили

Вы разработали яркое интерактивное приложение для рисования. При этом вы использовали объектно-ориентированный подход к программированию, создав класс `Bubble` для фиксации свойств и поведения пузырьков.

Вот некоторые новые знания, полученные вами в этой главе:

- создание единого приложения с несколькими классами и несколькими файлами исходного кода на языке Java;
- использование объектно-ориентированного подхода к программированию для разбиения проблемы на более мелкие фрагменты;
- создание с чистого листа класса `Bubble`, включающего в себя атрибуты, методы и конструктор;

- создание случайных цветов с помощью классов `java.util.Random` и `java.awt.Color`;
- генерирование случайных чисел в заданном диапазоне с помощью вызова `Random.nextInt (range)`;
- рисование с помощью класса `java.awt.Graphics`;
- использование метода `paintComponent()` для описания того, что ваше приложение должно рисовать, и использование метода `repaint()` для очистки экрана и перерисовки;
- написание обработчиков событий для событий мыши, включая события щелчков, перемещения и прокрутки колесика;
- использование класса `java.util.ArrayList` для хранения динамического списка объектов, таких как пузырьки, в приложении «Рисование пузырьков»;
- добавление нескольких слушателей событий в приложение с использованием пользовательского класса слушателя, `BubbleListener`.

Дополнительные задачи

Попробуйте выполнить следующие задания. Таким образом вы повторите пройденный материал, попрактикуетесь в программировании и получите новые навыки. Если у вас возникнут проблемы, вы можете загрузить примеры решений на сайте по адресу **https://eksmo.ru/files/learn_java_easy_way.zip**.

Задача № 1: Ограничение минимального размера пузырьков

Если вы сильно прокрутите колесико мыши вниз, пузырьки просто исчезнут, потому что значение атрибута `size` опустится ниже 0, таким образом, получится круг, который программа на языке Java не может нарисовать на экране. Фактически, уже при значении атрибута `size`, равном `2` или меньше, пузырьки будет создаваться размером всего в один пиксель. Поэтому давайте ограничим минимальное значение размера пузырька тремя пикселями по ширине и высоте, чтобы пузырьки оставались круглыми.

Добавьте в метод `mouseWheelMoved()` в классе `BubbleListener` в файле *BubblePanel.java* проверку, не стало ли значение переменной `size` меньше `3`. В таком случае установите размер равный `3`, чтобы пузырек оставался видимым на экране. Добавьте

новую инструкцию `if` в конец метода `mouseWheelMoved()`, чтобы проверить размер после его изменения, а не до этого.

Запустите приложение с этим изменением, и вы сможете прокрутить колесико мыши до тех пор, пока пузырьки не станут всего пару пикселей в ширину, но, тем не менее, они больше не исчезнут. Отличная работа!

Задача № 2: Рисование пикселями

Добавив немного расчетов в конструктор `Bubble()`, вы сможете создать забавный эффект пикселей, как показано на рис. 9.7.

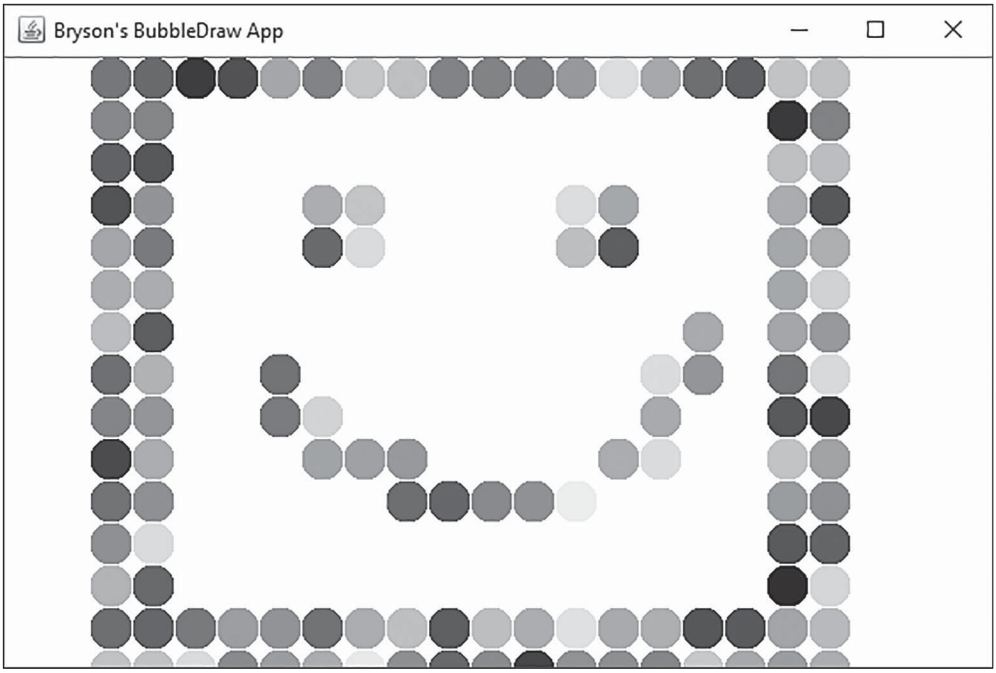


Рис. 9.7. Небольшое изменения местоположения новых объектов `Bubbles` может создать эффект забавного пиксельного рисования

Скопируем папку проекта *BubbleDraw*, чтобы изменять новый проект, не портя оригинальное приложение. Дважды щелкните мышью на вкладке *BubblePanel.java*, чтобы восстановить представление по умолчанию, а затем на панели **Package Explorer** (Обозреватель пакетов) нажмите сочетание клавиш **Ctrl+C** (или ⌘ + **C** в операционной системе macOS), чтобы скопировать папку проекта *BubbleDraw*. Нажмите сочетание клавиш **Ctrl+V** (или ⌘ + **V**),

чтобы вставить копию папки в рабочую область. Измените имя папки на **PixelDraw**, как показано на рис. 9.8.

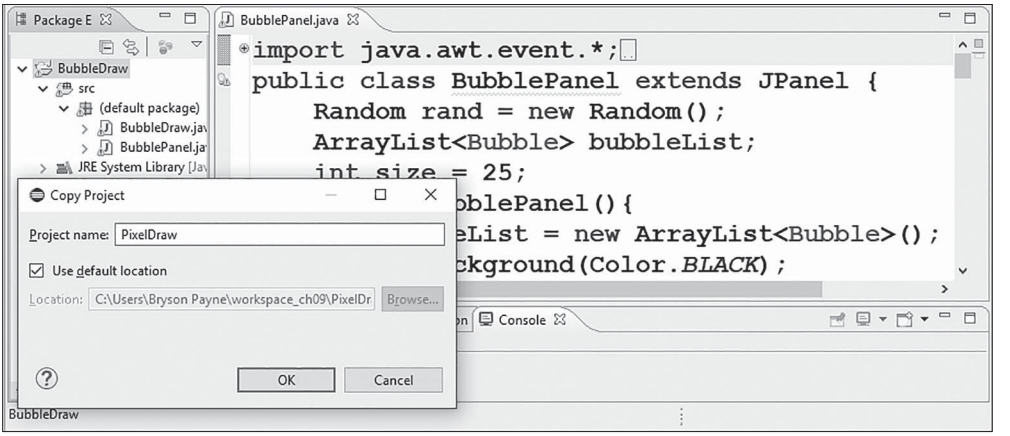


Рис. 9.8. Вы можете скопировать папку проекта на панели **Package Explorer** и вставить ее. Таким образом, будет создана копия проекта

Измените код конструктора `Bubble()` внутри закрытого класса `Bubble` следующим образом:

```
public Bubble(int newX, int newY, int newSize) {
    x = (newX / newSize) * newSize + newSize/2;
    y = (newY / newSize) * newSize + newSize/2;
    size = newSize;
    color = new Color(rand.nextInt(256),
    rand.nextInt(256),
    rand.nextInt(256));
}
```

Добавленные расчеты изменяют координаты (`x`, `y`) каждого нового пузырька. Это происходит из-за способа обработки деления двух целых чисел. Дело в том, что частное от деления двух целых чисел в языке Java является целым числом, поэтому деление на `newSize` и умножение на `newSize` заставят ваши пузырьки располагаться по сетке, как показано на рис. 9.7. Например, если переменная `newSize` равна `10`, а `x` — `25`, то `x`, деленный на `newSize`, будет равен `25 / 10` с отброшенной дробной частью, то есть `2` в программе на языке Java. Умножая на `10`, мы получим `20`, местоположение пузыря на сетке 10 ×10 пикселей. К каждой координате мы добавляем `newSize/2`, чтобы выровнять точки в воображаемой

сетке по центру. Помните, что прокрутка колесика мыши по-прежнему изменяет размер пузырьков, поэтому вы можете нарисовать толстые, блочные изображения или аккуратные чертежи с точностью до пикселя.

Если вы хотите сделать пиксельный эффект в духе игры Minecraft, можете превратить пузырьки в квадраты. Для этого измените метод `draw()` так, чтобы он рисовал закрашенные прямоугольники вместо овалов, закомментировав команду `fillOval()` и добавив вызов `fillRect()`:

```
public void draw(Graphics canvas) {
    canvas.setColor(color);
    // canvas.fillOval(x - size/2, y – size/2, size, size);
    canvas.fillRect(x - size/2, y – size/2, size, size);
}
```

Теперь вы можете рисовать в пиксельном стиле, как показано на рис. 9.9.

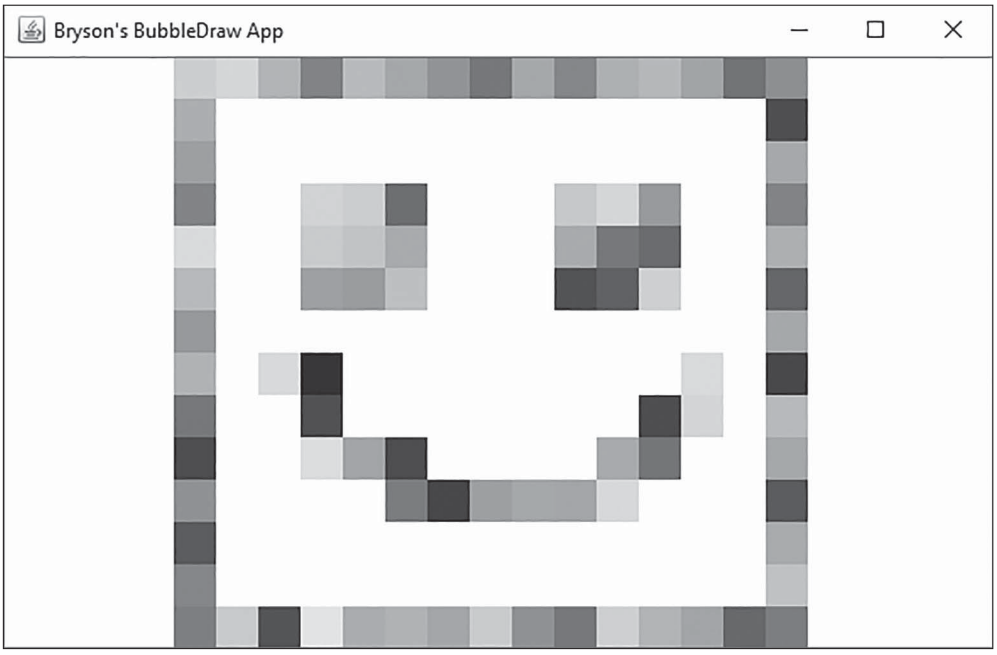
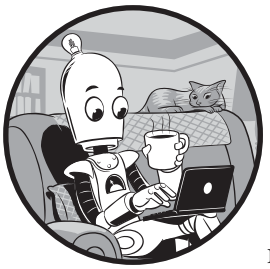


Рис. 9.9. Рисование прямоугольниками вместо овалов создает эффект пиксельного рисования

Глава 10

ДОБАВЛЕНИЕ АНИМАЦИИ И ВЫЯВЛЕНИЕ СТОЛКНОВЕНИЙ С ПОМОЩЬЮ ТАЙМЕРОВ



В этой главе мы добавим в наше приложение «Рисование пузырьков» анимацию на основе таймера, чтобы создать плавающие, отражающиеся от стен пузырьки. Кроме того, мы улучшим приложение с помощью удобного графического интерфейса. В улучшенное приложение под названием BubbleDrawGUI мы добавим панель `JPanel`, содержащую компоненты графического интерфейса, показанные на рис. 10.1, что даст пользователю возможность заставлять пузырьки двигаться или останавливаться, изменять скорость анимации и очищать экран.

Эта версия приложения более интерактивна и удобна для пользователя, чем раньше. Она позволяет пользователю рисовать плавающие, отражающихся от стен пузырьки, щелкая и передвигая мышью.

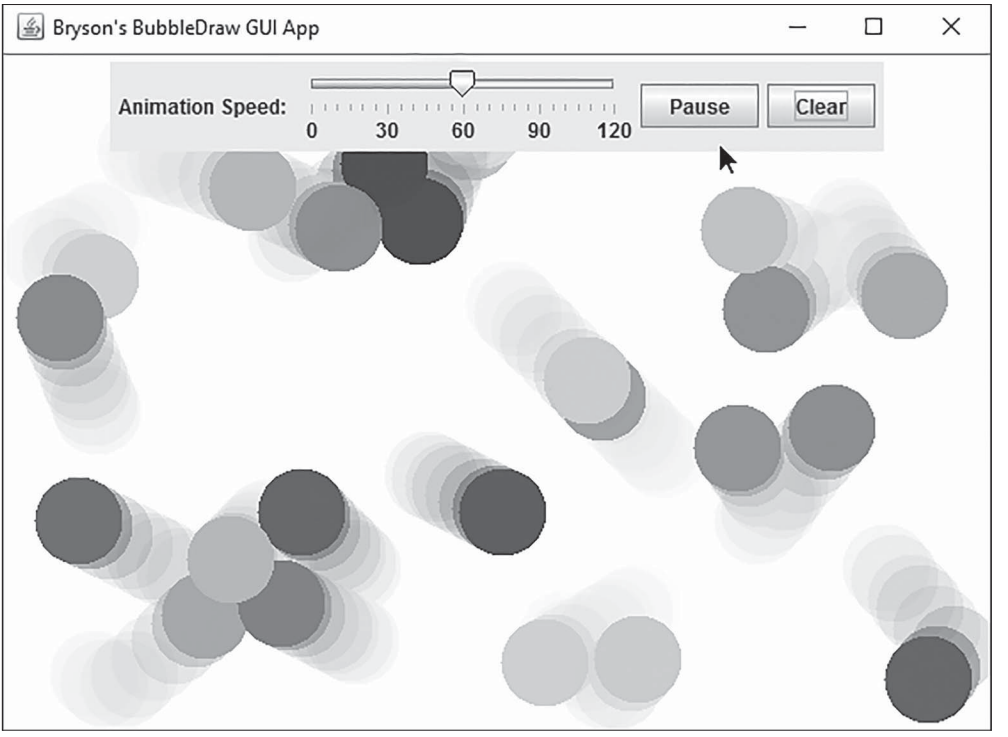


Рис. 10.1. В улучшенном приложении BubbleDrawGUI с графическим интерфейсом пользователя для управления анимацией появились анимированные, отталкивающиеся от стен, полупрозрачные пузырьки

Копирование проекта BubbleDraw для создания BubbleDrawGUI

Новое приложение с графическим интерфейсом будет создаваться на основе проекта BubbleDraw из главы 9. Поэтому вместо создания нового проекта на языке Java с нуля мы скопируем проект BubbleDraw и вставим его в ту же рабочую область с новым именем. Это полезный подход, когда вы хотите расширить существующую версию программы и работать над более новой версией, сохраняя предыдущую версию неприкосновенной.

В среде разработки Eclipse щелкните правой кнопкой мыши по папке проекта *BubbleDraw* на панели **Package Explorer** (Обозреватель пакетов) и выберите пункт **Copy** (Копировать) в контекстном меню. Затем щелкните правой кнопкой мыши на панели **Package Explorer** (Обозреватель пакетов) и выберите команду **Paste** (Вставить) в контекстном меню. Диалоговое окно **Copy Project**

(Копировать проект) позволяет присвоить копии новое имя. Укажите имя **BubbleDrawGUI** и нажмите кнопку **OK**. Программа Eclipse создаст копию проекта BubbleDraw на панели **Package Explorer** (Обозреватель пакетов) под названием BubbleDrawGUI.

Переименование основного класса и файла на языке Java

Теперь давайте переименуем файл *BubbleDraw.java*. В этом файле содержится метод `public static void main()`, который запускает приложение, и переименование файла поможет нам отличить новое приложение от старой версии. Внутри новой папки проекта *BubbleDrawGUI* щелкните правой кнопкой мыши по файлу *BubbleDraw.java* и выберите команду **Refactor** ⇒ **Rename** (Рефакторинг ⇒ Переименовать) в контекстном меню.

Рефакторинг означает реструктуризацию вашего кода, но не его функциональности. Обычно программисты проводят рефакторинг, когда думают о лучшем, более эффективном способе достижения того же результата. Когда появится окно **Rename Compilation Unit** (Переименовать компилируемый модуль), введите новое имя, **BubbleDrawGUI**, и нажмите кнопку **Finish** (Готово). В данный момент может появиться второе окно, предупреждающее, что класс содержит метод `main()`. Вы можете игнорировать это предупреждение, просто нажмите кнопку **Finish** (Готово) второй раз. Процесс рефакторинга переименует класс и файл на языке Java в BubbleDrawGUI. Пока мы оставим класс `BubblePanel` без изменений.

Наконец, давайте изменим заголовок окна `JFrame` в соответствии с новой версией графического интерфейса приложения. Откройте файл *BubbleDrawGUI.java*, найдите строку, в которой создается фрейм `JFrame`, и измените ее следующим образом:

```
import javax.swing.JFrame;
public class BubbleDrawGUI extends JFrame {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Your Name's BubbleDraw GUI App");
```

При первом запуске приложения, содержащего несколько файлов, такого как BubblePanel или BubbleDrawGUI, вам нужно запустить файл, содержащий метод `main()`. Запуск основного файла создаст конфигурации, которые в дальнейшем

позволят выполнять программу простым нажатием кнопки запуска. Класс `BubblePanel` не содержит метод `main()`, поэтому мы должны либо запустить файл *BubbleDrawGUI.java*, либо щелкнуть правой кнопкой мыши по папке проекта *BubbleDrawGUI* и выбрать команду **Run As ⇒ Java Application** (Выполнить как ⇒ Приложение Java) в контекстном меню.

Сохраните файл и запустите его, чтобы увидеть новый заголовок в верхней части окна, как показано на рис. 10.2.

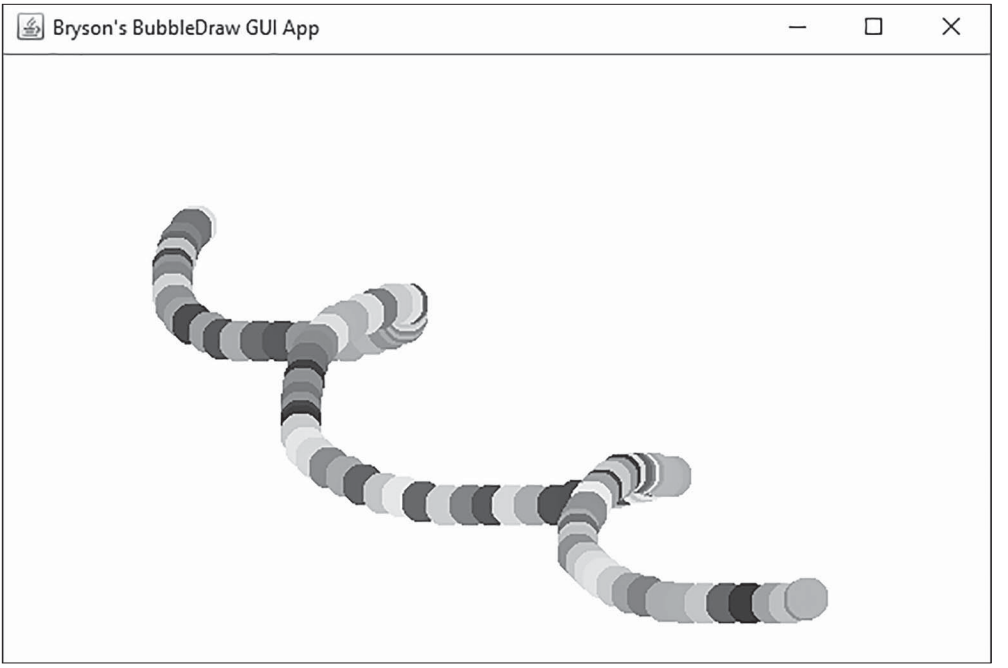


Рис. 10.2. Новый файл *BubbleDrawGUI.java* открывает окно с новым названием приложения в строке заголовка

Теперь давайте сделаем еще одно изменение, чтобы пузырьки выглядели более реалистично.

Добавление прозрачности

Реальные пузырьки часто выглядят полупрозрачными. Подумайте о том, как выдувают пузырь из жевательной резинки: как только он становится достаточно большим, вы сможете глядеть сквозь тонкую поверхность пузыря. Мы можем добавить прозрачность

к пузырькам в приложении BubbleDrawGUI, чтобы придать им более реалистичный вид.

В дополнение к компонентам цвета RGB, которые вы изучили в главе 9, программа на языке Java может хранить четвертый компонент в классе `java.awt.Color`. Он называется *альфа*-компонентом и характеризует то, насколько прозрачным или непрозрачным будет объект, нарисованный на экране перед другими объектами. Подобно значениям цвета RGB, альфа-компонент может иметь значения от `0` до `255`. При значении альфа-компонента 0 цвет (и объект) становится невидимым, `128` сделает его полупрозрачным, как акварельная краска, а при `255` объект, выкрашенный этим цветом, полностью закроет любой объект, находящийся за ним.

Поскольку конструктор класса `Color` может принимать альфа-значение в качестве четвертого аргумента (сразу после значений цвета RGB), то для добавления прозрачности нам потребуется изменить только одну строку в файле *BubblePanel.java*. Откройте файл *BubblePanel.java* в папке *src* проекта BubbleDrawGUI и промотайте до нижней части файла, где определен класс `Bubble`:

```
private class Bubble {
    private int x;
    – пропуск–
        color = new Color(rand.nextInt(256),
            rand.nextInt(256),
            rand.nextInt(256),
            rand.nextInt(256));
    }
```

Здесь мы поменяли конструктор переменной цвета, добавив четвертое случайное значение, которое может варьироваться в диапазоне от 0 до 255. Мы сделали это, поставив запятую после третьего `rand.nextInt(256)` в инструкции `color` и добавив четвертый атрибут `rand.nextInt(256)` перед закрывающей скобкой конструктора `Color()`. Внимательно проверьте, расставили ли вы запятые и скобки так же, как и в приведенном здесь коде. В противном случае приложение не будет работать.

Сохраните файл, а затем запустите его. Щелкайте по экрану и рисуйте пузырьки, которые слегка перекрывают друг друга, как показано на рис. 10.3.

Теперь вы видите, что пузырьки имеют не только случайные цвета, но и различные уровни прозрачности. Некоторые пузырьки

непрозрачны и полностью закрывают экран позади них, в то время как другие настолько прозрачны, что едва заметны. Наши пузырьки стали еще пузыристее! Теперь давайте для большей реалистичности сделаем их двигающимися.

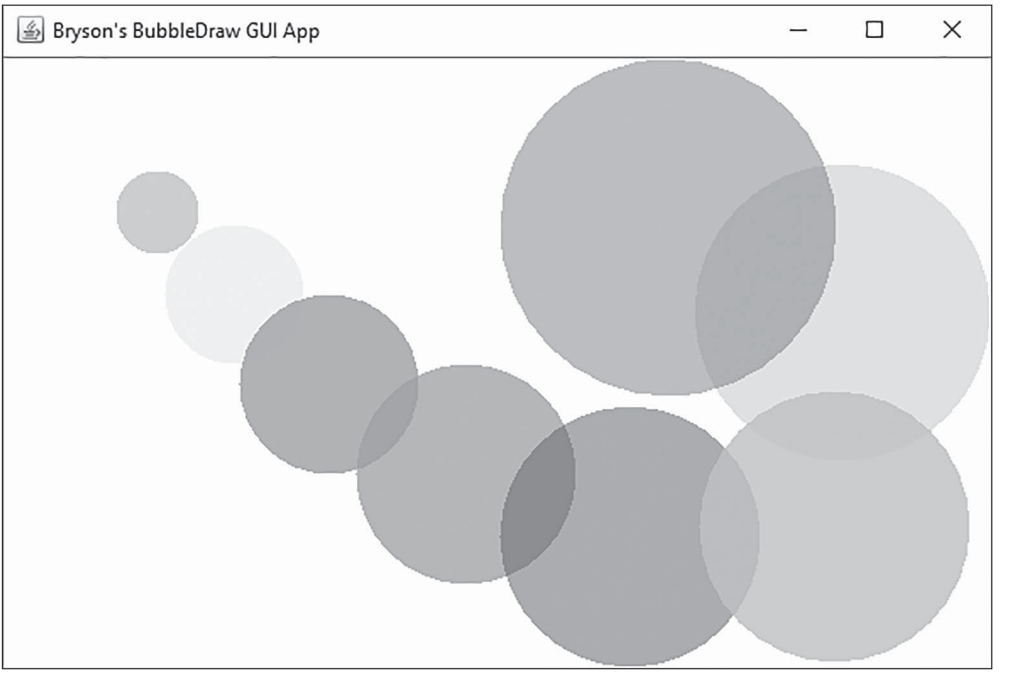


Рис. 10.3. Добавление альфа-компонента к цвету каждого пузырька придает им приятный прозрачный вид

Добавление анимации: пузырьки растут!

Анимация — это иллюзия движения, созданная путем отображения последовательности изображений на экране. Возможно, вам доводилось создавать анимацию в стиле кинеографа (флипбука) в тетради: каждый рисунок немного отличается от предыдущего, и, когда вы пролистывали страницы, картинка оживала. Мы добавим схожий эффект, чтобы выглядело, будто пузырьки двигаются в приложении BubbleDrawGUI.

Чтобы анимировать пузырьки, нам нужно нарисовать все пузырьки на экране, немного изменить их расположение, а затем снова нарисовать экран. И так несколько раз в секунду. Каждый экран, который мы нарисуем, называется *кадром*. Если мы перерисовываем объекты достаточно быстро, наши глаза и мозг заполняют

промежутки между кадрами, заставляя нас считать, что это один и тот же объект, плавно двигающийся по экрану. *Частота кадров* анимации, или скорость перерисовки экрана, обычно составляет около 30 кадров в секунду. Мы будем использовать новый класс `javax.swing.Timer`, который создает таймеры, сообщающие нашей программе, когда нужно перерисовывать пузырьки. Кроме того, мы будем использовать обработчик событий для обновления местоположения пузырьков и перерисовки экрана при каждом срабатывании таймера.

Для создания анимированных пузырьков необходимо сделать четыре шага: добавить таймер, установить таймер, подготовить анимацию и запустить таймер. Это те же самые шаги, которые вы использовали бы для добавления анимации в игру или любое другое приложение, использующее таймер на языке программирования Java.

Добавление таймера

Чтобы добавить таймер в наше приложение, нам нужно импортировать класс `javax.swing.Timer`. В верхней части файла *BubblePanel.java* добавьте следующую инструкцию `import`:

```
import javax.swing.Timer;
import java.awt.event.*;
import java.util.ArrayList;
import java.awt.Graphics;
import java.util.Random;
import java.awt.Color;
import javax.swing.JPanel;
```

Импорт класса `Timer` из `javax.swing` позволяет нам создать объект `timer`, который запускает событие с выбранной нами частотой. Обратите внимание на вторую строку в представленном фрагменте кода: мы уже импортировали классы `java.awt.event.*` ранее. Эта строка импортирует все обработчики событий `java.awt`, в том числе класс `ActionListener`, который мы будем использовать для обработки событий таймера.

Далее, в классе `BubblePanel` добавьте две переменные: одну с именем `timer` для самого таймера и переменную для задержки

с именем `delay` и типом `int`, содержащую количество миллисекунд, которые таймер должен ждать, прежде чем перерисовывать экран:

```
public class BubblePanel extends JPanel {
    Random rand = new Random();
    ArrayList<Bubble> bubbleList;
    int size = 25;
    Timer timer;
    int delay = 33;
```

Таймеры на языке программирования Java должны знать, сколько миллисекунд, или тысячных долей секунды, нужно ждать, прежде чем они должны вызвать событие таймера. Миллисекунда проходит очень быстро, поэтому я выбрал задержку в 33 миллисекунды. Таким образом, экран будет перерисовываться примерно 30 раз в секунду, так как 1 секунда = 1000 миллисекунд, а 1000/33 = 30 рисунков в секунду. Это примерно такая же скорость, как у мультфильма, показанного по телевидению.

Установка таймера

Теперь мы готовы установить таймер. Внутри конструктора `BubblePanel()` добавьте следующую строку для инициализации таймера и его установки с заданной задержкой:

```
public BubblePanel() {
    timer = new Timer(delay, new BubbleListener());
    bubbleList = new ArrayList<Bubble>();
    setBackground(Color.BLACK);
    // testBubbles();
    addMouseListener(new BubbleListener());
    addMouseMotionListener(new BubbleListener());
    addMouseWheelListener(new BubbleListener());
}
```

Конструктор для нового таймера `Timer()` требует двух параметров: первый — это задержка в миллисекундах, а второй — обработчик события, который будет слушать события таймера. При каждом отключении Таймер запускает событие `actionPerformed()`,

похожее на то событие `actionPerformed()`, которое мы обрабатываем для щелчков по кнопкам в графическом интерфейсе пользователя. Таймер похож на автоматическую кнопку, которая «нажимает» себя каждые несколько миллисекунд. Мы поместили таймер в начало конструктора, чтобы позже можно было изменить его в ответ на события графического интерфейса.

Чтобы слушать события таймера, мы изменим класс `BubbleListener`. Прокрутите файл *BubblePanel.java* вниз и найдите закрытый класс `BubbleListener`, созданный нами ранее. Затем добавьте код `implements ActionListener` перед открывающей фигурной скобкой класса:

```
private class BubbleListener extends MouseAdapter implements ActionListener {
```

Это изменение позволит классу `BubbleListener` прослушивать события `actionPerformed()`, реализуя класс `ActionListener` из `java.awt.event.*`. Реализация класса слушателя событий — это еще один способ обработки пользовательских событий. Чтобы обрабатывать эти события таймера, нам нужно добавить обработчик события `actionPerformed()`. Добавьте следующий метод в конец класса `BubbleListener`:

```
    private class BubbleListener extends MouseAdapter implements ActionListener {
        public void mousePressed(MouseEvent e) {
            — пропуск—
        }
        public void mouseDragged(MouseEvent e) {
            — пропуск—
        }
        public void mouseWheelMoved(MouseWheelEvent e) {
            — пропуск—
        }
        public void actionPerformed(ActionEvent e) {
        }
    }
```

В этом новом методе `actionPerformed()` мы добавим код, который перемещает пузырьки и перерисовывает экран при каждом срабатывании таймера. Эти команды мы добавим чуть позже.

Мы добавили таймер и настроили `BubbleListener` для прослушивания событий таймера. Теперь нам нужно сообщить компилятору языка Java, что делать, когда таймер запускает событие.

При каждом запуске события таймером необходимо нарисовать следующее изображение в анимированной последовательности пузырьков. Сначала мы попросим обработчик события `actionPerformed()` обновить пузырьки и перерисовать экран. Затем мы попросим класс `Bubble` обновить пузырек, перемещая его вверх по экрану. В нашей программе эти шаги будут выполняться примерно 30 раз в секунду.

Внутри метода `actionPerformed()`, добавленного в класс `BubbleListener`, добавьте три следующие строки кода для обновления пузырьков и перерисовки экрана:

```
        public void actionPerformed(ActionEvent e) {
❶           for (Bubble b: bubbleList)
❷               b.update();
❸           repaint();
        }
```

В строке ❶ мы используем конструкцию `for each` для перебора каждого объекта `Bubble b` в списке `bubbleList`. Помните, что список `bubbleList` — это динамический массив `ArrayList`, содержащий все пузырьки, которые мы создали, щелкая мышью и перетаскивая мышь по экрану.

Перебирая пузырьки в списке `bubbleList`, мы вызываем новую функцию с именем `update()` для каждого из них в строке ❷. Среда разработки Eclipse подчеркнет эту команду красным, поскольку мы еще не определили метод `update()` в классе `Bubble`, но мы сделаем это буквально через минуту. В методе `update()` мы будем менять местоположение пузырьков, чтобы выглядело так, как будто они всплывают по экрану вверх.

Строкой ❸ мы вызываем метод `repaint()` для обновления экрана, очистки холста и рисования пузырьков в новых, измененных позициях. Делая это 30 раз в секунду, мы достигнем желаемого эффекта анимации.

Теперь давайте создадим метод `update()`, в котором сообщим классу `Bubble`, как перемещать пузырьки после каждого запуска таймера анимации. В системе координат языка Java (`x`, `y`) нам

нужно уменьшать значения y (в верхней часть экрана значение y равно 0). Таким образом, для создания иллюзии движения пузырьков вверх, мы должны, при каждом обновлении, вычитать небольшое число из координаты y.

Прокрутите окно вниз до нижней части файла *BubblePanel.java*, где мы определили класс Bubble и добавьте метод update() под методом draw(), как показано ниже:

```
        public void draw(Graphics canvas) {
            canvas.setColor(color);
            canvas.fillOval(x - size/2, y – size/2, size, size);
        }
        public void update() {
            y -=5;
        }
    }
}
```

Функция обновления позиции пузырька вычитает пять пикселей из значения координаты y этого пузырька. Каждый раз, когда пузырь перерисовывается на экране, он будет располагаться на пять пикселей выше, чем раньше.

Сохраните свой файл. До запуска приложения нам остался всего один шаг!

Пуск таймера

На последнем этапе анимации приложения BubbleDrawGUI необходимо запустить таймер. Промотайте код вниз до конструктора BubblePanel() и добавьте следующую строку:

```
    public BubblePanel() {
        timer = new Timer(delay, new BubbleListener());
        – пропуск–
        addMouseWheelListener(new BubbleListener());
        timer.start();
    }
```

Метод timer.start() запустит таймер так, чтобы он вызывал события с заданной частотой (в миллисекундах), до тех пор,

пока не будет вызван метод `timer.stop()` или пока вы не выйдете из программы.

Сохраните и запустите программу. Нарисованные пузырьки должны плавно всплывать вверх.

Колесо прокрутки мыши и все остальные возможности, которые мы сделали в главе 9, по-прежнему работают одновременно с завораживающими эффектами анимации. Пока пузырьки плывут только в одном направлении, но мы получили иллюзию движения, к которой стремились. В следующем разделе вы узнаете, как заставить пузырьки двигаться во всех направлениях.

Постоянно сдуваемые пузырьки: добавление случайной скорости и направления

Функция `update()`, которую мы создали в предыдущем разделе, изменяет только координату `y` каждого пузырька, в результате чего пузырьки перемещаются вверх после каждой перерисовки экрана. В этом разделе мы сделаем пузырьки движущимися как по вертикали, так и по горизонтали со случайной скоростью, чтобы казалось, что ветер дует во всех направлениях, как показано на рис. 10.4.

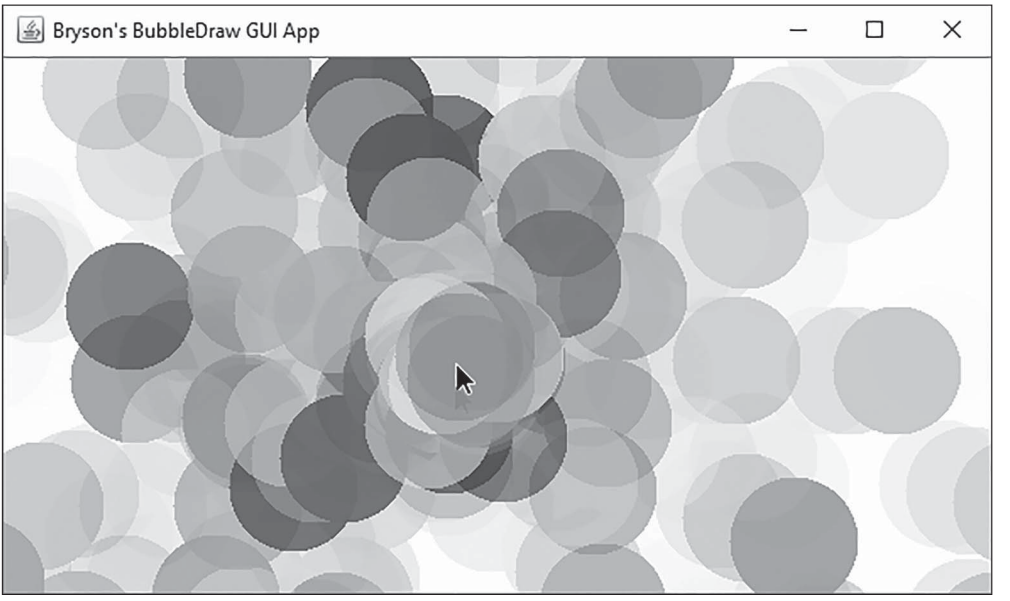


Рис. 10.4. При изменении обеих координат (*x* и *y*) каждого пузырька, кажется, что пузырьки плывут в случайных направлениях

Горизонтальная скорость пузырька — это количество пикселей влево или вправо, на которое он перемещается при изменении кадра. Именно она определяет новую координату `x` пузырька. Точно так же вертикальная скорость пузырька определяет его новую координату `y`. Просто перемещая пузырек в горизонтальном и вертикальном направлении, мы фактически можем заставить его двигаться в любом направлении. На рис. 10.5 показано, как горизонтальная и вертикальная скорость, объединяясь, создают иллюзию движения пузырька по диагонали.

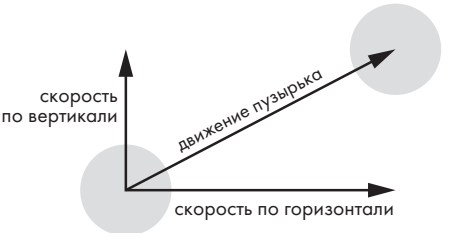


Рис. 10.5. Быстрое изменение обеих координат пузырька (*x* и *y*) приводит к иллюзии, что он перемещается на экране по диагонали

Во-первых, давайте добавим переменные для хранения числа пикселей, на которое должен перемещаться каждый пузырек по горизонтальному (`x`) и вертикальному (`y`) направлениям после каждой перерисовки экрана. Добавьте следующие две строки в начало класса `Bubble`:

```
    private class Bubble {
        — пропуск—
        private Color color;
❶       private int xspeed, yspeed;
❷       private final int MAX_SPEED = 5;
```

В строке ❶ мы объявили две целочисленные переменные: `xspeed` для количества пикселей, на которое пузырек будет перемещаться по горизонтали после каждого обновления экрана, и `yspeed` для количества пикселей, на которые пузырек будет перемещаться по вертикали. В строке ❷ мы добавляем *константу* `MAX_SPEED`, для определения максимального количества пикселей, на которые пузырек может переместиться за один раз. Константы похожи на переменные, но в отличие от них, константы не изменяются внутри программы, поэтому мы объявляем их как `final`, тем самым говоря компилятору языка Java, что значение константы является постоянным. Кроме того, согласно конвенции о наименованиях, мы даем константам имена, состоящие только из прописных букв, что помогает отличать их от обычных переменных.

Мы зададим каждому пузырьку случайную скорость по вертикали и горизонтали, используя, как и для определения его цвета, метод `rand.nextInt()`. Добавьте следующие две строки в конструктор `Bubble()`:

```
public Bubble(int newX, int newY, int newSize) {
    x = newX;
    y = newY;
    size = newSize;
    color = new Color(rand.nextInt(256),
        rand.nextInt(256),
        rand.nextInt(256),
        rand.nextInt(256));
    xspeed = rand.nextInt(MAX_SPEED * 2 + 1) - MAX_SPEED;
    yspeed = rand.nextInt(MAX_SPEED * 2 + 1) - MAX_SPEED;
}
```

Нам нужны значения скорости по горизонтали и вертикали, чтобы пузырек мог двигаться в любом направлении, но у нас есть только две переменные для четырех направлений (влево или вправо, вверх или вниз). Эта проблема легко решается при использовании как отрицательных, так и положительных значений. Когда значение `xspeed` отрицательно, пузырек будет двигаться влево, а когда положительно — вправо. Атрибут `yspeed` позволит пузырьку двигаться вверх (при отрицательном значении) и вниз (при положительном). Чтобы диапазоны значений `xspeed` и `yspeed` охватывали как отрицательные, так и положительные величины, я умножил `MAX_SPEED` на 2 и добавил 1, то есть получил число `11 (5 * 2 + 1 = 11)`. Таким образом, команда `rand.nextInt(MAX_SPEED * 2 + 1)` эквивалентна команде `rand.nextInt(11)`, которая вернет число от `0` до `10`. Вычитая из этого значения константу `MAX_SPEED`, получим результат между `-5` и +5, поскольку `0-5 = -5`, а `10-5 = 5`.

Кроме того, нам нужно изменить функцию `update()` так, чтобы пузырек перемещался в свое новое местоположение после каждой перерисовки экрана. Замените команду `y = 5;` следующими двумя инструкциями:

```
public void update() {
    x += xspeed;
    y += yspeed;
}
```

Вместо того, чтобы перемещаться вверх на пять пикселей после каждой перерисовки экрана, каждый пузырек будет перемещаться горизонтально на `xspeed` пикселей и вертикально на `yspeed` пикселей. Оба эти значения мы сгенерировали случайным образом для этого конкретного пузырька. В результате получается красочный взрыв пузырей от каждого перетаскивания мыши!

Сохраните и запустите программу с этими изменениями, и вы увидите эффект, подобный тому, который мы видели на рис. 10.4. Эффект перемещения пузырьков одновременно по вертикали и горизонтали дает каждому пузырьку иллюзию случайной скорости и направления.

При этом вы могли обратить внимание на любопытное поведение некоторых пузырьков: они просто стоят на месте, как те, что находятся в центре на рис. 10.6. Это происходит из-за того, что мы произвольно генерируем числа от `-5` до `+5` для вертикальной и горизонтальной скорости. Иногда значения обоих атрибутов и `xspeed` и `yspeed` пузырька будут равны `0`. Когда это случится, пузырек вообще не сдвинется с места.

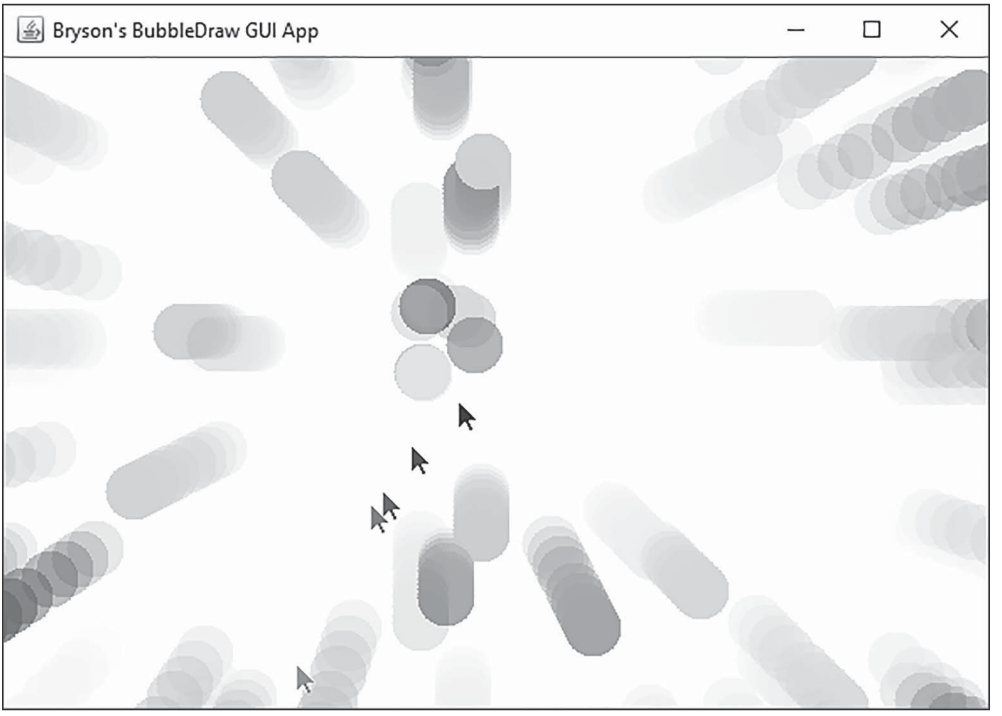


Рис. 10.6. Поскольку значения случайной скорости могут быть равны 0, пузырьки, подобные тем, которые расположены вблизи центра, остаются на месте

«Залипания» пузырьков можно избежать, проверяя, равны ли значения атрибутов `xspeed` и `yspeed` нулю, и заменяя в таких случаях один из них или оба — это задача программирования № 1 далее в этой главе.

Теперь нарисованные нами пузырьки улетают прочь. Давайте добавим несколько функций в приложение. Например, возможность приостановки анимации и очистки экрана. Пришло время создать в нашем анимированном приложении графический интерфейс пользователя.

Создание графического интерфейса для нашего анимированного приложения для рисования

Приложение BubbleDrawGUI является графическим, но у него нет интерфейса, такого как у наших других графических приложений. Кнопки **Pause/Start** и **Clear**, показанные на рис. 10.7, облегчат пользователю понимание и взаимодействие с приложением, поэтому давайте их добавим.

Рис. 10.7. Кнопки **Pause/Start** и **Clear**

Настройка панели и кнопок графического интерфейса пользователя

На панели **Package Explorer** (Обозреватель пакетов) щелкните правой кнопкой мыши по файлу *BubblePanel.java* и выберите пункт **Open With ⇒ WindowBuilder Editor** (Открыть в ⇒ Редактор WindowBuilder) в контекстном меню. Перейдите на вкладку **Design** (Конструктор), и вы попадете в режим конструктора графического интерфейса.

Во-первых, давайте добавим панель `JPanel`, которая будет использоваться в качестве контейнера для кнопок **Pause/Start** и **Clear**, а также любых других компонентов графического интерфейса, которые вы захотите добавить позже. На панели **Palette** (Панель элементов) в разделе **Containers** (Контейнеры) выберите объект `JPanel`. Затем установите указатель мыши в окне предварительного просмотра справа и щелкните мышью нв черном фоне конструктора `BubblePanel`, чтобы разместить новую панель `JPanel`.

Существует и другой способ добавления новой панели `JPanel`: выберите объект `JPanel` в разделе **Containers** (Контейнеры)

и щелкните мышью по строке `javax.swing.JPanel` в разделе **Structure** (Структура) панели **Components** (Компоненты). Вы увидите очень маленькую серую панель `JPanel`, появившуюся в верхней части черной панели `BubblePanel`.

Добавим кнопки **Pause/Start** и **Clear**. Прокрутите панель **Palette** (Панель элементов) вниз до раздела **Components** (Компоненты), выберите компонент `JButton`, а затем щелкните мышью внутри маленькой панели `JPanel`, которую мы только что добавили. Тем самым вы разместите первую кнопку `JButton`. Введите слово **Pause** в качестве текста кнопки, либо непосредственно в предварительном просмотре графического интерфейса пользователя, либо на панели **Properties** (Свойства). (Скоро вы поймете, почему мы называем эту кнопку **Pause/Start**.)

Выполните те же действия для создания кнопки очистки экрана: выберите `JButton` на панели **Palette** (Панель элементов), щелкните мышью внутри панели `JPanel`, чтобы поместить кнопку, и введите текст **Clear** для отображения на ней.

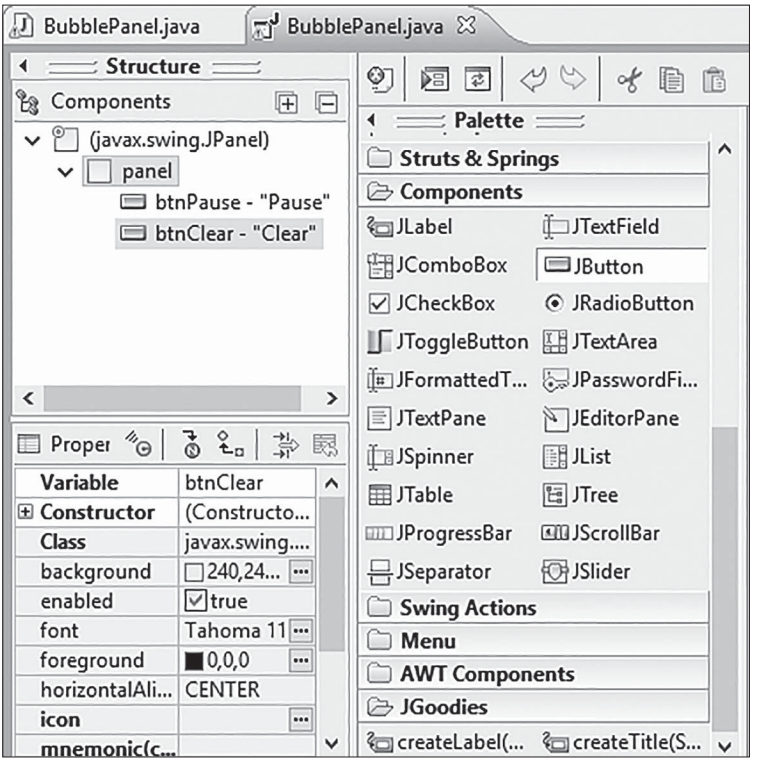


Рис. 10.8. При создании графического интерфейса вы можете выбрать компоненты на панели **Palette** и добавить их непосредственно на панель **Structure**. Сейчас мы добавили кнопки `btnPause` и `btnClear` к только что созданной панели

Если не удается поместить кнопки в панели `JPanel` из-за того, что она слишком мала, выберите объект `JButton` в панели инструментов, а затем щелкните мышью по панели в окне **Structure** (Структура) слева, чтобы разместить каждую кнопку внутри панели, как показано на рис. 10.8. Присвойте кнопкам имена `btnPause` и `btnClear` либо по мере их размещения, либо изменив значение свойства `Variable` на панели **Properties** (Свойства).

Окно **Structure** (Структура) — это удобный способ добавления в графический интерфейс компонентов в ситуациях, подобных этой, когда панель `JPanel` плохо видна в окне предварительного просмотра или когда мы хотим изменить порядок или группировку компонентов в графическом интерфейсе. На рис. 10.9 показаны обе кнопки в готовом графическом интерфейсе.

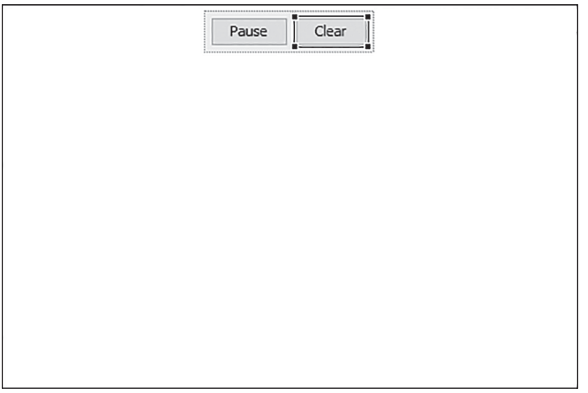


Рис. 10.9. Кнопки **Pause/Start** и **Clear** в готовом графическом интерфейсе

После добавления кнопок **Pause/Start** и **Clear** в верхнюю часть холста, пора запрограммировать обработчики событий кнопок.

Кодирование кнопок Clear и Pause/Start

Начнем с кнопки **Clear**. Одним из способов очистки экрана от всех пузырьков является присвоение переменной `bubbleList` нового пустого списка. Таким образом, пузырьки не будут отрисовываться, и пользователь может начать рисовать заново. Чтобы реализовать такое поведение, дважды щелкните мышью по кнопке **Clear** (помните, что двойной щелчок по кнопке на вкладке **Design** (Конструктор) создает слушатель событий для кнопки и переключает на вкладку **Source** (Исходный код)), а затем добавьте следующие

две строки внутри фигурных скобок для слушателя действий списка `bubbleList`:

```
        JButton btnClear = new JButton("Clear");
        btnClear.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
❶               bubbleList = new ArrayList<Bubble>();
❷               repaint();
            }
        });
        panel.add(btnClear);
```

В строке ❶ мы очищаем переменную `bubbleList`, присваивая ей новый динамический массив `ArrayList` объектов `Bubble`. Этот новый список будет пустым, так что все, что нам нужно сделать, это перерисовать экран. Таким образом, у нас будет чистый черный фон, такой же, как и при открытии приложения. Строкой ❷ мы вызываем функцию `repaint()`, чтобы нарисовать новый пустой экран.

Сохраните и запустите приложение. Затем нарисуйте несколько пузырьков и нажмите кнопку **Clear** для очистки экрана.

Вернитесь на вкладку **Design** (Конструктор) и дважды щелкните мышью по кнопке **Pause/Start**, чтобы создать еще один слушатель событий. После щелчка пользователя по кнопке **Pause/Start** мы хотим не только прекратить анимацию, остановив таймер, но и изменить текст на кнопке на **Start**. Затем, когда пользователь нажимает кнопку снова, мы перезапустим таймер для возобновления анимации и вернем кнопке надпись **Pause**. Добавьте следующий код в метод `actionPerformed()`, который создаст среда разработки Eclipse после двойного щелчка мышью по кнопке **Pause/Start**.

Убедитесь, что переменная `ActionEvent` в методе `actionPerformed()` имеет имя `e` (в нижеприведенном коде выделено полужирным шрифтом).

```
        JButton btnPause = new JButton("Pause");
        btnPause.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
```

```
❶                JButton btn = (JButton)e.getSource();
❷                if (btn.getText().equals("Pause")) {
❸                    timer.stop();
❹                    btn.setText("Start");
                 }
                 else {
❺                    timer.start();
❻                    btn.setText("Pause");
                 }
             }
         });
         panel.add(btnPause);
```

В строке ❶ (справа налево) мы используем метод `e.getSource()`, чтобы выяснить, по какой кнопке был щелчок, привести ее к типу переменной `JButton` и сохранить ссылку на кнопку как `btn`. Метод `getSource()` полезен для определения того, был ли нажат или изменен элемент графического интерфейса, особенно когда вы пишете обработчики событий для нескольких элементов сразу. В этом примере мы можем использовать `getSource()` для доступа к таким свойствам кнопки, как `text`.

В строке ❷ мы проверяем, соответствует ли текст на кнопке строке `"Pause"`. Если соответствует, мы останавливаем таймер строкой ❸ для приостановки анимации, а затем меняем текст на кнопке на `"Start"` (строка ❹).

Если на кнопке было написано не **Pause** — иными словами, если анимация уже была приостановлена, а текст на кнопке был изменен на **Start**, обработчик события перейдет к инструкции `else` и запустит таймер (строка ❺) для возобновления анимации. В свою очередь, текст кнопки **Pause/Start** изменится на **Pause** (строка ❻).

Сохраните файл и запустите его еще раз. Теперь вы можете приостановить анимацию, нарисовать что-то, а затем перезапустить анимацию, чтобы добиться восхитительного эффекта взрывания пузырьков, показанного на рис. 10.10.

Приложение BubbleDrawGUI — это визуально ошеломляющее анимированное приложение, и кнопки дают пользователю больше контроля над экраном. Но как только анимация запускается, пузырьки уплывают за края экрана и никогда не возвращаются. Что, если мы сможем заставить пузырьки отскакивать от границ окна, оставаясь на экране дольше?

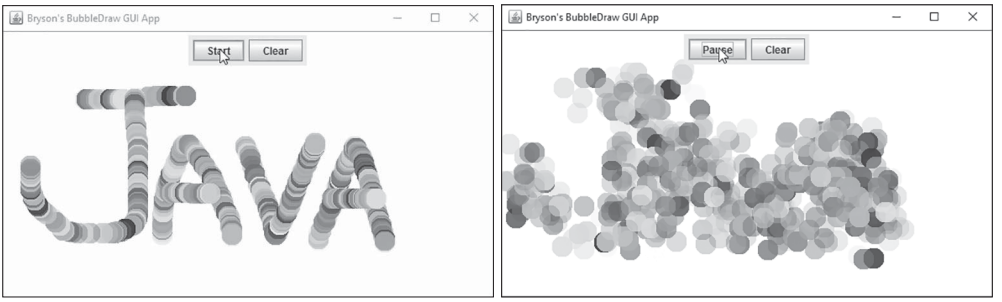


Рис. 10.10. Приостановите анимацию, нарисуйте фигуру, а затем снова запустите анимацию, и вы увидите, как ваш рисунок взрывается яркими пузырьками

Отскок от стенок и обнаружение столкновений

Анимация используется не только в мультфильмах и заставках. Есть еще одна сфера, в которой вы постоянно сталкиваетесь с анимацией: компьютерные игры. Будь то мобильное приложение, самая современная онлайн-игра или консольная программа, анимация — это то, с помощью чего разработчики игр дают пользователю ощущения движения и действия.

Одна из полезных концепций игрового программирования, которую мы можем добавить к последней версии приложения BubbleDrawGUI, — это *обнаружение столкновения*, которое позволяет нам проверять, перекрываются или *сталкиваются* ли два объекта на экране. Вы можете использовать обнаружение столкновений для сообщения программе, что делать, когда игрок стреляет по вражескому космическому кораблю или бьет по мячу в футболе. В этом приложении мы хотим узнать, достиг ли пузырек края холста, чтобы в таком случае изменить его направление, создавая эффект отражения от края экрана назад к центру, как пузырек на рис. 10.11.

Обнаружение столкновений может давать виртуальным объектам, такие как наши пузырьки в приложении BubbleDrawGUI, более реалистичное поведение. В вашей любимой компьютерной игре обнаружение столкновений — это то, что мешает вашему игроку проваливаться сквозь пол или ходить сквозь стены. Эти объекты являются мнимыми — они всего лишь компьютерная графика, поэтому они не могут *действительно* столкнуться друг с другом, но обнаружение столкновений создает иллюзию, что они твердые. Таким образом, если мы сделаем так, что пузырьки будут мягко отскакивать от краев экрана, они будут выглядеть реалистичнее. Посмотрим, как это работает.

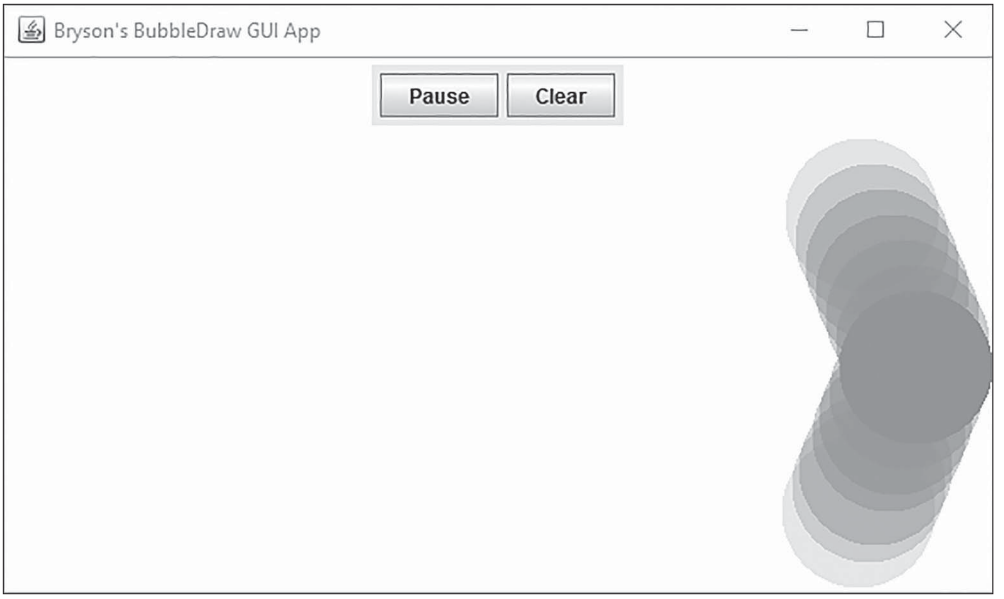


Рис. 10.11. Пузырек отскакивает от правого края окна благодаря обнаружению столкновения

Мягкий отскок

Во-первых, давайте разберемся, как обнаружение столкновения работает для пузырьков, отскакивающих от края окна. Мы уже знаем, что у каждого пузырька есть пара координат (`x` и `y`), определяющих его местоположение, и атрибуты `xspeed` и `yspeed`, определяющие количество пикселей, на которые пузырьки перемещаются горизонтально и вертикально от одного обновления экрана к другому.

Чтобы выяснить, столкнулся ли пузырек с краем окна, нам нужно знать координаты краев экрана, чтобы их можно было сравнить их с координатами пузырька. Самое маленькое значение координаты `x` у левого края экрана, то есть `x == 0`. А наименьшее значение координаты `y`, или `y == 0` у верхнего края экрана. Но как насчет правого и нижнего края экрана?

В программе на языке Java каждый компонент графического интерфейса пользователя наследует пару методов для возврата ширины и высоты компонента: `getWidth()` и `getHeight()`. Наш холст для приложения BubbleDrawGUI представляет собой панель `JPanel BubblePanel`. Поэтому, если мы вызовем функции `getWidth()` и `getHeight()` внутри `BubblePanel`, мы получим максимальное значение `x` — `getWidth()` и `y` — `getHeight()`.

Мы будем проверять, столкнулся ли пузырек с краем экрана в методе `update()` класса `Bubble`. Возможно, вы помните, что в методе `update()` координаты каждого пузырька меняются с использованием атрибутов `xspeed` и `yspeed` для создания иллюзии движения.

Давайте уточним, что мы подразумеваем под «отскоком». На рис. 10.11 пузырек движется вправо к краю экрана, где `x == getWidth()`. Это означает, что значение координаты `x` пузырька достигает своего максимального значения, то есть ширины экрана в пикселях. Чтобы все выглядело так, будто пузырек отскакивает, мы меняем направление его движения, изменяя значение `xspeed` на противоположное. Пузырек перемещается на некоторое положительное количество пикселей при каждом обновлении; после того, как он коснется края экрана, он должен начать двигаться в противоположном направлении, для чего надо изменить знак `xspeed`. Другими словами, значение атрибута `xspeed` должно быть *отрицательным*, чтобы после отскока пузырек двигался влево от правого края экрана.

Мы можем изменить горизонтальную скорость пузырька на противоположную, установив `xspeed = -xspeed`. Таким образом, скорость `xspeed` в 3 пикселя за кадр изменится на `xspeed -3` пикселя за кадр после того, как пузырек столкнется с правым краем экрана и направление его движения изменится после отскока.

То же самое нужно сделать и с левым краем экрана, где `x == 0`. Если значение координаты `x` пузырька показывает, что пузырек касается левого края, команда `xspeed = -xspeed` снова изменит горизонтальное движение: скорость `xspeed -3` станет `-(-3)` или `+3`. Это заставит пузырек двигаться вправо, от края экрана в центр.

Доберитесь до самой нижней части файла *BubblePanel.java*, где мы определяем класс `Bubble`. Найдите метод `update()` и добавьте следующий код обнаружения столкновения для левого и правого края экрана:

```
public void update() {
    x += xspeed;
    y += yspeed;
    if (x <= 0 || x >= getWidth())
        xspeed = -xspeed;
}
```

Если значение координаты `x` пузырька становится меньше `0` или больше ширины экрана, пузырек должен коснуться либо левого, либо правого края экрана, а потом отскочить в противоположном

направлении. Для этого мы меняем атрибут `xspeed`. Для верхнего и нижнего краев мы выполним аналогичные действия, но на этот раз мы изменим атрибут `yspeed`:

```
public void update() {
    x += xspeed;
    y += yspeed;
    if (x <= 0 || x >= getWidth())
        xspeed = -xspeed;
    if (y <= 0 || y >= getHeight())
        yspeed = -yspeed;
}
```

Если значение координаты `y` пузырька становится меньше `0` или больше высоты экрана в пикселях, мы меняем `yspeed` на `-yspeed`, чтобы пузырек начал двигаться в противоположном направлении.

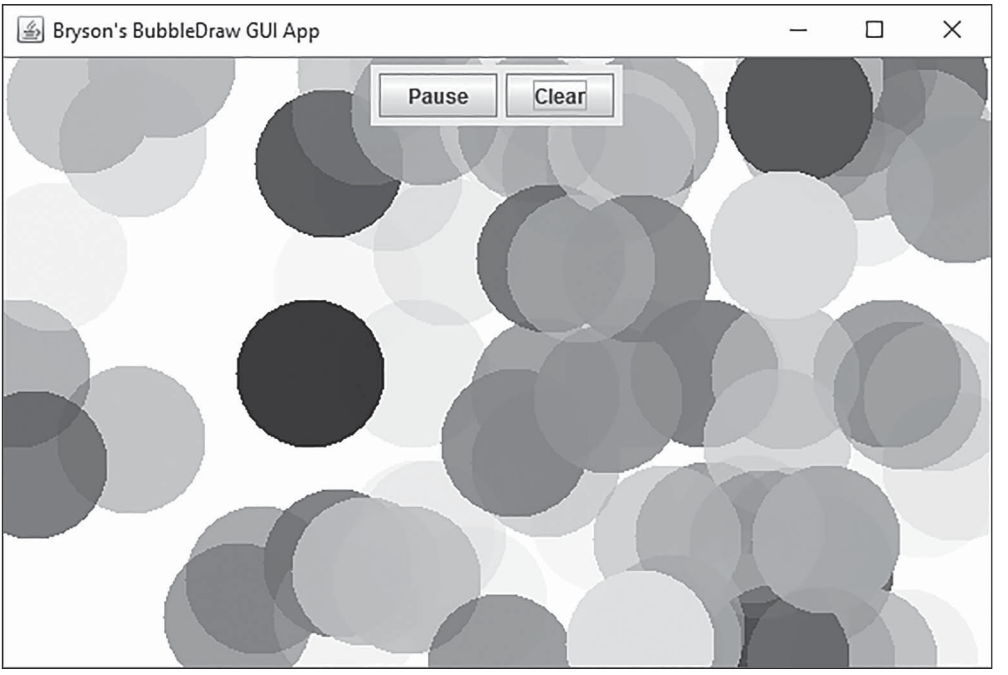


Рис. 10.12. Вы можете заметить, что множество пузырьков, прежде чем отскочить от края экрана, выходят за его границы почти на половину своего размера

После того, как вы внесли эти дополнения в метод `update()`, сохраните и запустите файл. На этот раз вы увидите, как созданные вами пузырьки, отскакивают от всех четырех краев. Однако

вы можете заметить небольшую проблему: прежде чем отскочить, пузырьки выходят за границы экрана почти на половину своего размера, как показано на рис. 10.12.

Этот «мягкий» отскок происходит, потому что мы проверяем столкновение между *центром* каждого пузырька и краями экрана. Напомним, что в главе 9 мы рисовали каждый пузырек вокруг (`x`, `y`) координат щелчка пользователя, поэтому значения каждого `x` и `y` каждого пузыря представляют собой местоположение центра этого пузыря. Чтобы пузырьки оставались на экране полностью, нам нужно проверить наличие столкновений между внешним краем каждого пузыря и краями холста.

Жесткий отскок

Чтобы проверить наличие столкновений с краем каждого пузыря, нам придется учитывать расстояние от центра пузыря до его края, то есть величину его радиуса (так как каждый пузырек — идеальный круг). В данном случае радиус — это половина размера каждого пузырька, или `size / 2`. Чтобы учесть размер каждого пузырька, измените две инструкции `if` в методе `update()`, как показано ниже:

```
        public void update() {
            x += xspeed;
            y += yspeed;
❶           if (x - size/2 <= 0 || x + size/2 >= getWidth())
                xspeed = -xspeed;
❷           if (y - size/2 <= 0 || y + size/2 >= getHeight())
                yspeed = -yspeed;
        }
```

Во фрагменте ❶ мы вычитаем величину `size/2` из `x`, чтобы увидеть, коснулся ли левый край пузырька левого края экрана. Если `x - size/2` меньше или равно `0`, значит касание произошло. Вы же помните, что деление выполняется перед вычитанием? Следовательно, сначала будет подсчитано отношение `size/2`, а затем оно будет вычитаться из `x`, поэтому нам не нужно добавлять круглые скобки вокруг деления. В другом выражении мы прибавляем отношение `size/2` к `x`, чтобы узнать, касается ли правый край пузырька правого края экрана, что произойдет в случае, если

`x + size/2` больше или равно результату, возвращенному методом `getWidth()`. Во фрагменте ❷ мы проделали аналогичные изменения, чтобы получить верхний `(y − size/2)` и нижний `(y + size/2)` край каждого пузырька для проверки, касаются ли они верхнего или нижнего края холста соответственно.

Сохраните свою программу и запустите ее снова. Теперь все созданные вами пузырьки, большие и маленькие, хорошо отскакивают от краев окна, как показано на рис. 10.13.

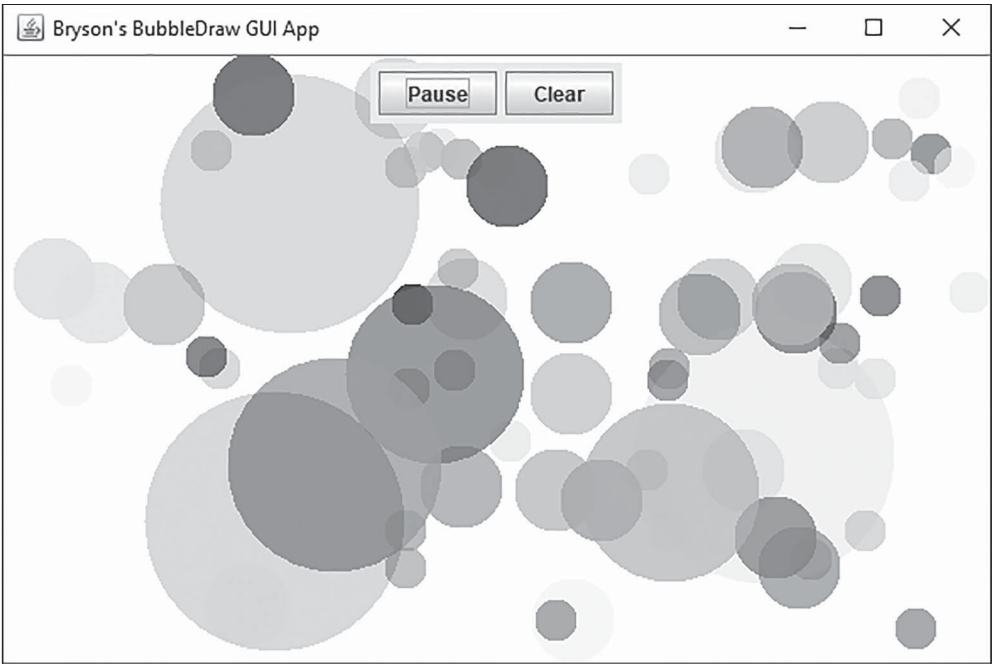


Рис. 10.13. Теперь наши пузырьки «жестко» отскакивают от краев окна, что кажется более реалистичным

Разверните окно, щелкнув мышью по кнопке разворачивания в строке заголовка приложения или дважды щелкнув мышью по строке заголовка. Вы увидите, что пузырьки отскакивают от краев холста даже в полноэкранном режиме. Мы использовали методы `getWidth()` и `getHeight()` для определения правого и нижнего краев, а эти методы всегда возвращают текущую ширину и высоту, поэтому вы можете изменять размер окна приложения как угодно.

Теперь давайте внесем последнее дополнение и предоставим пользователю еще больше возможностей с помощью графического интерфейса пользователя.

Добавление ползункового регулятора для управления скоростью анимации

На данный момент пользователь может останавливать движение пузырьков и запускать его снова, очищать экран, а также создавать пузырьки любого размера. Давайте также предоставим ему контроль над скоростью анимации, добавив ползунковый регулятор, с помощью которого будет изменяться задержка таймера, как показано на рис. 10.14.

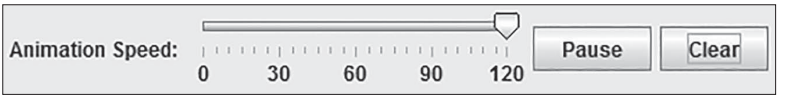


Рис. 10.14. Добавление ползункового регулятора позволит пользователю ускорить или замедлить анимацию

Сначала вернитесь на вкладку **Design** (Конструктор) и добавьте метку `JLabel` и ползунковый регулятор `JSlider` в панель управления графическим интерфейсом. На панели **Palette** (Панель элементов) найдите раздел **Components** (Компоненты) и выберите компонент `JLabel`. Щелкните мышью по маленькой панели перед кнопкой **Pause/Start** в окне предварительного просмотра графического интерфейса, чтобы поместить метку. Измените текст метки на «**Animation Speed:**».

Затем выберите компонент `JSlider` на панели **Palette** (Панель элементов) и щелкните мышью между меткой «**Animation Speed:**» и кнопкой **Pause/Start**, чтобы поместить ползунковый регулятор, как показано на рис. 10.15.

Возможно, вы обратите внимание, что панель `JPanel` для всех компонентов графического интерфейса растет по мере добавления элементов. Если вы выберете панель в окне **Structure** (Структура), вы увидите, что по умолчанию свойству `Layout` присвоено значение `java.awt.FlowLayout`. Это расширяемый макет, позволяющий вместить столько элементов графического интерфейса, сколько необходимо. Для игры «Больше-Меньше» и приложения «Секретные сообщения» мы использовали макет `AbsoluteLayout`, поскольку хотели разместить элементы в определенных позициях. В этом приложении для рисования мы можем быть более гибкими, а `FlowLayout` идеально подходит для добавления компонентов графического интерфейса пользователя на лету.

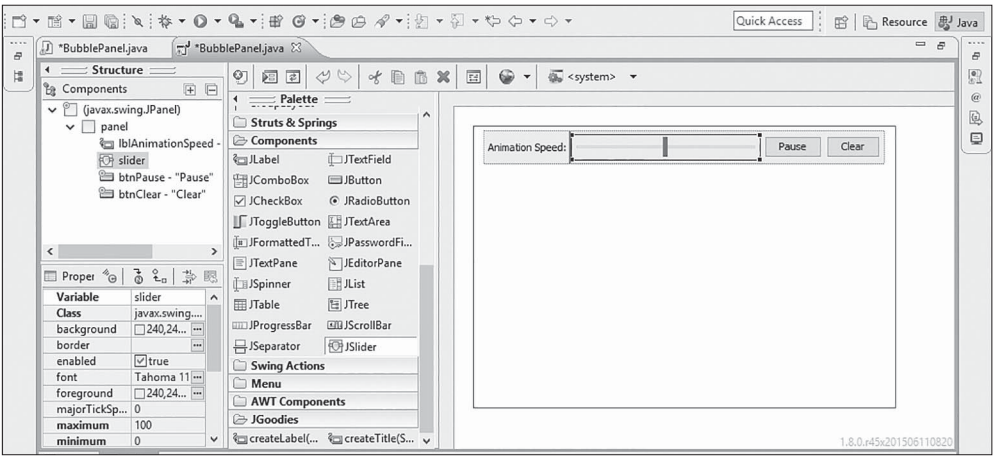


Рис. 10.15. Добавьте метку и ползунковый регулятор на панель управления графическим интерфейсом приложения в режиме конструктора

Настройка ползункового регулятора

Чуть позже мы изменим несколько свойств, чтобы настроить ползунковый регулятор, поэтому давайте для начала решим, как мы хотим, чтобы он выглядел. Ползунковый регулятор должен позволять пользователю легко и интуитивно изменять скорость анимации. Другими словами, если пользователь уменьшает скорость анимации до 0, анимация должна замедляться почти до остановки. Если пользователь передвигает ползунковый регулятор до упора вправо, анимация должна идти очень быстро.

Мониторы обычно обновляют экран в диапазоне от 30 до 120 раз в секунду, а частота 60 Гц (сокращение от *Герц*, мера частоты в секунду) является наиболее типичной частотой обновления. Если мы анимируем пузырьки быстрее, чем 120 раз в секунду, ваш монитор, вероятно, не сможет отображать все кадры анимации. Поэтому имеет смысл установить максимальное значение скорости ползункового регулятора до 120 кадров в секунду.

Количество кадров в секунду, сокращенно *fps* (*frames per second* — кадров в секунду), часто является мерилом плавности анимации. Движения экрана в игре, которая работает на скорости 60 fps на вашем компьютере, будут выглядеть более плавно, чем в игре с 30 fps.

Выберите ползунковый регулятор в окне предварительного просмотра. На панели **Properties** (Свойства) в левом нижнем углу установите диапазон, присвоив свойству `maximum` значение `120`, а `minimum` — значение `0` (по умолчанию). Чтобы подготовить

метки и отметки, присвойте свойству `mainTickSpacing` значение **30**, свойству `minorTickSpacing` значение `5` и установите флажки `paintLabels`, `paintTicks` и `paintTrack`, чтобы значение всех трех параметров было `true`. Наконец, присвойте свойству `value` значение `30`, это количество кадров в секунду по умолчанию. На рис. 10.16 показана панель **Properties** (Свойства) со всеми нашими настройками, а также предварительный просмотр ползункового регулятора `JSlider` после изменений.

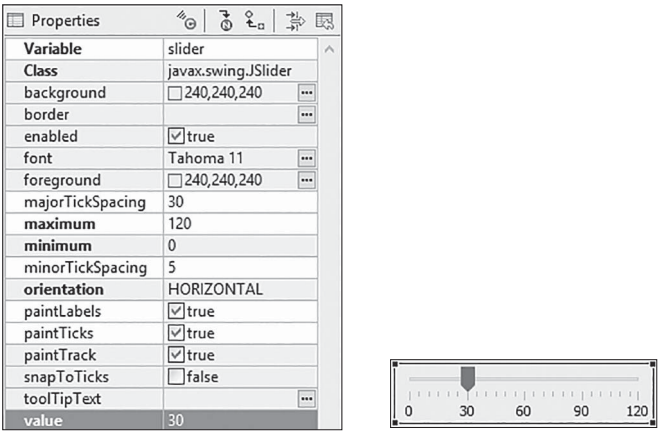


Рис. 10.16. Панель **Properties** для ползункового регулятора **Animation Speed:** с нашими настройками (слева); предварительный просмотр ползункового регулятора после настройки (справа)

Реализация обработчика событий ползункового регулятора

В окне предварительного просмотра конструктора щелкните правой кнопкой мыши по ползунковому регулятору и выберите пункт **Add event handler ⇒ change ⇒ stateChanged** (Добавить обработчик события ⇒ change ⇒ stateChanged) в контекстном меню. Среда разработки Eclipse добавит слушатель `ChangeListener` с методом `stateChanged()`, подобный тому, что мы использовали в реализации ползункового регулятора в приложении «Секретные сообщения». Код будет выглядеть следующим образом:

```
JSlider slider = new JSlider();
slider.addChangeListener(new ChangeListener() {
    public void stateChanged(ChangeEvent e) {
    }
});
```

Во-первых, нам нужно объявить ползунковый регулятор `JSlider` в верхней части класса, чтобы была возможность доступа к значению ползункового регулятора внутри кода обработчика события `stateChanged()`. Найдите класс `BubblePanel` и добавьте следующую строку чуть ниже переменных `timer` и `delay`:

```
public class BubblePanel extends JPanel {
    Random rand = new Random();
    ArrayList<Bubble> bubbleList;
    int size = 25;
    Timer timer;
    int delay = 33;
    JSlider slider;
```

Затем вернитесь к ползунковому регулятору и удалите объявление типа `JSlider` в начале первой строки:

```
        slider = new JSlider(); // Удалите "JSlider" из начала строки
        slider.addChangeListener(new ChangeListener() {
            public void stateChanged(ChangeEvent e) {
            }
        });
```

Когда пользователь меняет положение ползункового регулятора, мы будем изменять скорость анимации, меняя длину задержки между каждым событием таймера. Для этого нам нужно получить значение скорости от ползункового регулятора, преобразовать скорость в число миллисекунд, а затем установить новое значение задержки таймера. Добавьте следующие три строки внутри фигурных скобок метода `stateChanged()`:

```
        slider = new JSlider();
        slider.addChangeListener(new ChangeListener() {
            public void stateChanged(ChangeEvent arg0) {
❶               int speed = slider.getValue() + 1;
❷               int delay = 1000 / speed;
❸               timer.setDelay(delay);
            }
        });
```

В строке ❶ мы используем метод `getValue()`, чтобы получить значение скорости из ползункового регулятора и сохранить его в целочисленной переменной с именем `speed`. Обратите внимание, что здесь мы прибавили `1` к значению ползункового регулятора. Мы делаем это для того, чтобы предотвратить ошибку деления на нуль в строке ❷, где мы делим `1000` на `speed` для определения количества миллисекунд задержки между кадрами. Ползунковый регулятор может дойти до `0`, но, добавив `1` к его значению, мы предотвращаем возникновение ошибки: 1000/0 выбрасывает исключение деления на ноль, но 1000/1 дает нам очень медленную задержку между кадрами в размере 1000 миллисекунд. На экране это выглядит, будто анимация почти остановлена. Это означает, что, когда пользователь перемещает ползунковый регулятор в `0`, анимация фактически не останавливается. Чтобы полностью остановить анимацию, нужно нажать кнопку **Pause/Start**.

Наконец, мы обновляем таймер, устанавливая новое значение задержки в миллисекундах строкой ❸. Изменение времени, проходящего между событиями таймера, замедлит или ускорит анимацию.

Сохраните файл и запустите его. Перемещайте ползунковый регулятор вперед и назад, и вы увидите, что теперь вы контролируете скорость анимации.

На данный момент приложение BubbleDrawGUI — наше самое интерактивное, интересное, визуально привлекательное приложение — анимированное приложение с графическим интерфейсом, которое дает пользователю полный контроль над анимацией. Поиграйте с ним некоторое время — и поздравьте себя с хорошо проделанной работой!

Поделитесь с друзьями

Этим приложением можно поделиться с друзьями. Чтобы экспортировать исполняемый файл с расширением JAR из среды разработки Eclipse, выберите команду меню **File ⇒ Export** (Файл ⇒ Экспорт). Затем разверните папку *Java* и выберите пункт **Runnable JAR file** (Исполняемый JAR-файл). Нажмите кнопку **Next** (Далее) и в разделе **Launch confguration** (Конфигурация запуска) в раскрывающемся списке выберите пункт **BubbleDrawGUI — BubbleDrawGUI**.

В разделе **Export destination** (Расположение экспорта) нажмите кнопку **Browse** (Обзор), а затем выберите папку, в которую хотите сохранить свое готовое приложение, например *Рабочий стол*. Присвойте программе название, например *Bryson's BubbleDraw.jar*. Нажмите кнопку **Save** (Сохранить), а затем **Finish** (Готово).

Откройте каталог, в котором вы сохранили свой файл с расширением JAR, запустите его и поделитесь им с друзьями. Даже если у ваших друзей нет среды разработки Eclipse, они смогут запустить приложение BubbleDrawGUI. Наслаждайтесь!

Что вы изучили

Мы улучшили приложение «Рисование пузырьков» из главы 9, добавив в него анимацию с отскакивающими от краев экрана пузырьками и графический интерфейс. Ниже представлены некоторые из навыков, которые вы получили в этой главе:

- объединение графического приложения и графического интерфейса;
- копирование проекта и вставка новой версии на панели **Package Explorer** (Обозреватель пакетов) в среде разработки Eclipse;
- переименование класса или объекта с помощью рефакторинга;
- использование прозрачности с помощью установки альфа-компонента в цветах RGBA;
- создание, настройка и запуск объекта `Timer`;
- обработка событий объекта `Timer`;
- создание анимации путем перемещения графических объектов с помощью таймера;
- создание отскакивающих виртуальных объектов с помощью обнаружения столкновений;
- использование методов `getWidth()` и `getHeight()` для обнаружения краев окна;
- использование ползункового регулятора для изменения задержки таймера;
- управление скоростью анимации с помощью изменения свойства таймера `delay`.

Дополнительные задачи

Попробуйте выполнить следующие задания. Таким образом, вы повторите пройденный материал, попрактикуетесь в программировании и получите новые навыки. Если у вас возникнут проблемы, вы можете загрузить примеры решений на сайте по адресу **https://eksmo.ru/files/learn_java_easy_way.zip**.

Задача № 1: Ни один пузырек не остается на месте

Одна проблема, которую мы заметили в этой главе, заключалась в том, что некоторые пузырьки получают случайную скорость `0`, заставляя их казаться застрявшими на месте, в то время, как другие пузырьки уплывают за экран. Это происходит в том случае, когда и горизонтальная (`xspeed`) и вертикальная (`yspeed`) скорость пузырька равны `0`. Чтобы решить эту проблему, добавьте код, который гарантирует, что пузырьков со случайной скоростью `0` не будет. Для этого проверьте значения `xspeed` и `yspeed`, не равны ли они оба `0`. Все, что вам нужно сделать для таких случаев, это установить переменным другое значение, например `1`.

§ Добавьте инструкцию `if` внутри конструктора `Bubble ()` после указания значений `xspeed` и `yspeed`.

```
private class Bubble {
    – пропуск–
        public Bubble(int newX, int newY, int newSize) {
        – пропуск–
            xspeed = rand.nextInt(MAX_SPEED * 2) – MAX_SPEED;
            yspeed = rand.nextInt(MAX_SPEED * 2) – MAX_SPEED;
            if // Здесь добавьте код
        }
```

Внесите изменения, сохраните и запустите файл. Вуаля — больше нет пузырьков, сидящих на мели!

Задача № 2: Гибкое рисование

Из-за случайных значений скорости пузырьки движутся в разных направлениях, и это тот эффект, которого мы изначально добивались. Но что, если мы будем рисовать из пузырьков фигуры и захотим, чтобы они держались все вместе? Установка скорости каждого пузырька на одно и то же фиксированное значение создает интересный эффект кручения, когда объекты поворачиваются при отскоке от краев экрана, как показано на рис. 10.17.

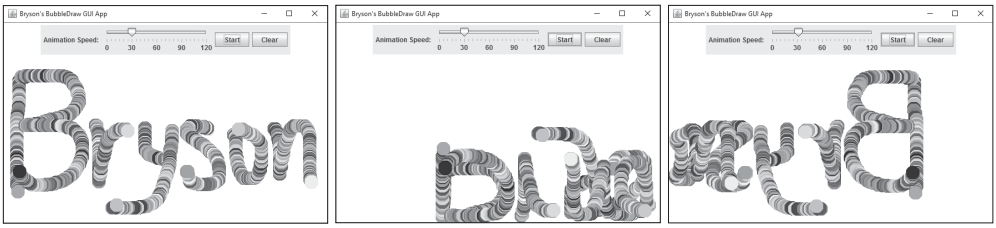


Рис. 10.17. Если всем пузырькам задать одинаковую скорость xspeed и yspeed, то получится причудливый эффект кручения, при котором пузырьки поворачиваются все вместе, отскакивая от края экрана

Для этой задачи создайте новую копию приложения, чтобы не портить исходную версию. На панели **Package Explorer** (Обозреватель пакетов) скопируйте и вставьте проект BubbleDrawGUI, назовите его **FlexiDraw** или по вашему выбору. В файле *BubblePanel* измените код конструктора `Bubble()`, чтобы вместо присвоения случайных значений переменным `xspeed` и `yspeed`, им устанавливалось одно и то же* значение, например, следующим образом:

```
xspeed = yspeed = 2;
```

Эта строка кода использует интересную возможность оператора присваивания (=). Она называется *присваиванием по цепочке*, поскольку обеим переменным `xspeed` и `yspeed` присваивается значение `2`, а знак равенства позволяет нам присваивать одно и то же значение нескольким переменным в цепочке.

Вы можете выбрать большее или меньшее число для вашего значения скорости. Важно отметить, что мы поменяли случайные скорости на фиксированную начальную скорость и направление для каждого пузырька. Каждый пузырек, будучи нарисован, будет

* Чтобы такого эффекта достичь, необходимо присвоить обеим переменным фиксированное, но не обязательно одинаковое, значение (*прим. пер.*).

двигаться вправо на два пикселя и вниз на два пикселя, поэтому пузырьки будут формировать группы.

Сохраните файл, щелкните правой кнопкой мыши по папке *FlexiDraw* на панели **Package Explorer** (Обозреватель пакетов) и выберите команду **Run As ⇒ Java Application** (Выполнить как ⇒ Приложение Java) в контекстном меню. Приостановите анимацию, нарисуйте пузырьки, а затем нажмите кнопку **Start**. Вы увидите, как ваши фигуры изгибаются, скручиваются и отскакивают, перемещаясь по экрану! Если вы будете рисовать во время работы анимации, вы получите забавный спиральный эффект.

Задача № 3: Рисование пикселей 2.0

Для этой задачи вы будете повторно использовать код «Рисование пикселями» из главы 9 (Задача программирования № 2). Добавление пиксельного эффекта в анимированную программу рисования позволит вам рисовать квадратные пиксельные фигуры и анимировать их. Объедините этот эффект с приемом «Гибкое рисование» из предыдущего задания, и вы получите изгибающееся, отскакивающее изображение в стиле Minecraft, как показано на рис. 10.18.

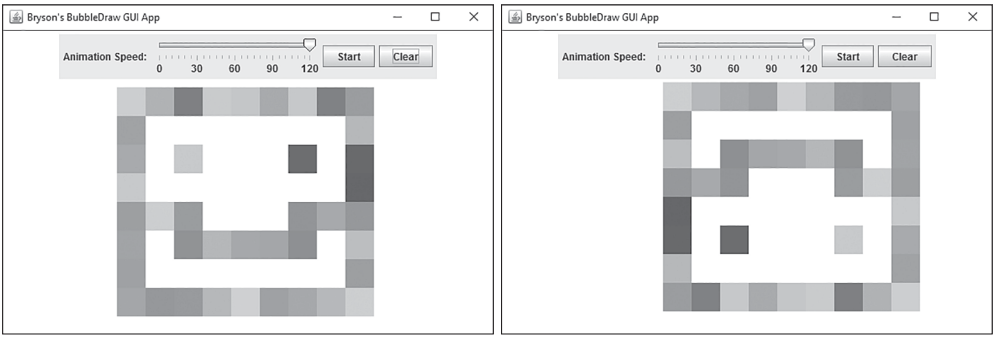


Рис. 10.18. Эффект «Рисование пикселями» 2.0 (слева); такая же форма после отскока от края, перевернутая вверх тормашками и в зеркальном отображении (справа)

Остановите анимацию, чтобы блоки получились ровные, как в примере на рис. 10.18. Рисуя при включенной анимации, можно достичь многоярусного 3D-эффекта, как показано на рис. 10.19.

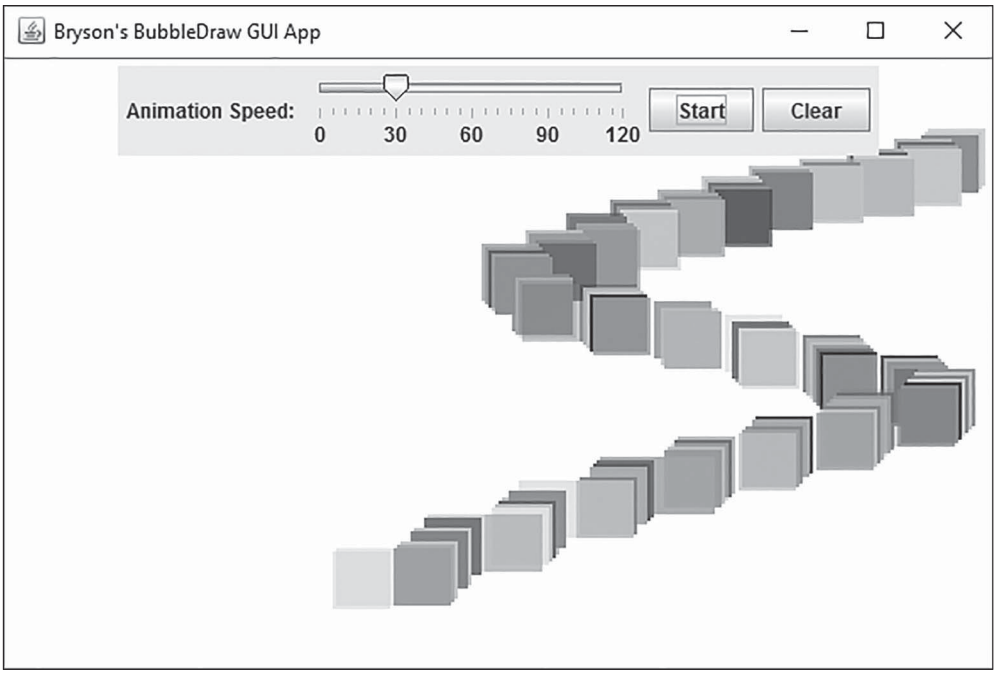


Рис. 10.19. Программа «Рисование пикселями» 2.0 создает многоярусный трехмерный эффект, если вы перетаскиваете мышь во время анимации

Немного подскажем с расчетами: для достижения ячеистого стиля вам нужно «разделить» экран на сетку, с ячейками, размер которых совпадает с размерами пузырьков, а затем нарисовать пузырек в этом месте сетки. В этот раз попробуйте изменить свои переменные `x` и `y` в верхней части конструктора `Bubble()` следующим образом:

```
x = (newX / newSize) * newSize + newSize/2;
y = (newY / newSize) * newSize + newSize/2;
```

Первая часть формулы (`newX / newSize`) * `newSize` делит экран на сетки с размером ячейки — `newSize`, то есть размером текущего пузырька. Чтобы выровнять координаты `x` и `y` по сетке с ячейками размером `newSize` на `newSize`, нам нужно, чтобы они были кратными величине `newSize`. Для координаты `x` мы делаем это путем деления `newX` на `newSize`, в результате чего получается целое число без десятичной части, которое мы затем умножаем на `newSize`, чтобы получить целое число, кратное `newSize`. Таким образом, координата `x` пузырька соответствует краю ячейки сетки, внутри которой пользователь щелкает мышью. Например,

если значение переменной `newSize` равно `10`, деление и умножение «привязывают» координату `x` к числам, кратным `10` (`10`, `20` и так далее). Если мы остановимся здесь, то получим пузырьки с центром в углу сетки. Однако мы бы хотели, чтобы пузырек содержался внутри ячейки сетки. Это достигается при помощи второй части формулы (`+ newSize / 2`).

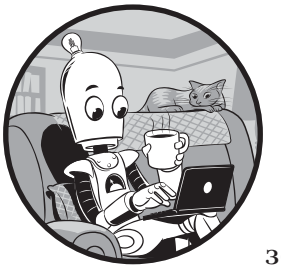
После этого нарисуйте блоки вместо пузырьков, изменив метод `draw()` так, чтобы в нем рисовались прямоугольники вместо овалов. Это делается с помощью метода `fillRect()`.

Вот оно! Но вы можете — и должны! — внести дополнительные изменения, чтобы сделать ваше приложение еще лучше. Если хотите, измените заголовок фрейма `JFrame` в файле *BubbleDrawGUI.java* или даже выполните рефакторинг файла. Нет предела совершенству!

Сделав эти изменения, вы сможете рисовать красивые, блочные, пиксельные и анимированные творения. Сделайте снимок экрана и твитните его своим друзьям. Используйте хэштэг #JavaTheEasyWay или упомяните меня, @brysonpayne, и я ретвитну его нескольким тысячам своих друзей!

Глава 11

СОЗДАНИЕ ПРИЛОЖЕНИЯ «РИСОВАНИЕ ПУЗЫРЬКОВ» С ПОДДЕРЖКОЙ МНОЖЕСТВЕННЫХ ПРИКОСНОВЕНИЙ ДЛЯ УСТРОЙСТВА ANDROID



Наше последнее приложение будет версией приложения «Рисование пузырьков» для устройства Android, которая позволит пользователю рисовать пузырьки прикосновением одного или даже всех десяти пальцев!

Процессоры на устройствах Android обычно намного меньше и медленнее, чем процессоры настольных компьютеров. Если вы когда-либо получали сообщение об ошибке App Not Responding (Приложение не отвечает), значит, вы сталкивались с тем, что происходит, когда приложение потребляет слишком много вычислительной мощности устройства. Поэтому вместо таймера эта

версия приложения будет использовать новый подход к анимации, называемый *многопоточность*, что позволит приложению задействовать меньше ресурсов процессора. Многопоточность позволяет нам запускать одновременно несколько приложений или *работать в режиме многозадачности*.

Кроме того, новое приложение «Рисование пузырьков» будет поддерживать *множественные прикосновения*. Взгляните на рис. 11.1, где показаны пузырьки, появляющиеся из нескольких разных мест, в которых мой младший сын Макс дотронулся пальцами до экрана.

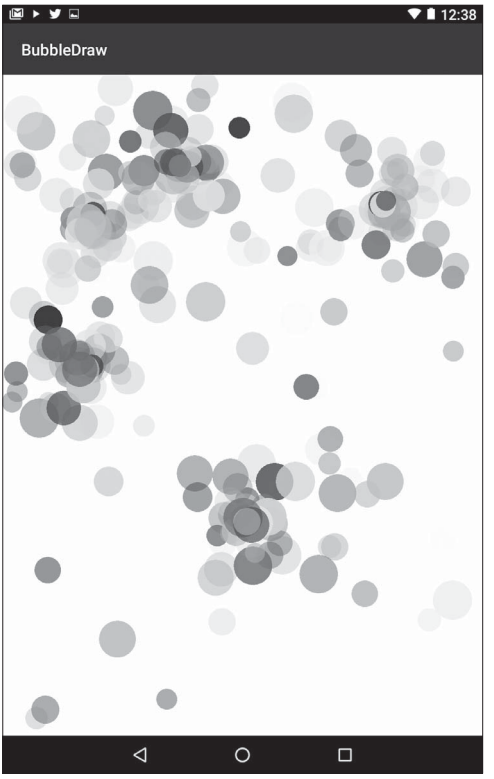


Рис. 11.1. Версия приложения «Рисование пузырьков» для устройства Android будет поддерживать множественные прикосновения, чтобы пользователь мог рисовать пузырьки в нескольких точках экрана одновременно

Мобильное приложение «Рисование пузырьков» будет повторно использовать множество функций из версии для компьютера с графическим интерфейсом, такие как исходный код класса `Bubble`, который мы создали в главе 10. Однако есть несколько отличий в том, как мы рисуем графику на устройстве Android. Поскольку мы также добавляем многопоточность и множественные

прикосновения, вам придется изучить некоторые новые методики создания приложений. Вы разовьете эти новые навыки, используя приложение «Рисование пузырьков» за основу.

Настройка проекта «Рисование пузырьков»

Откройте среду разработки Android Studio, закройте все открытые проекты и выберите пункт **Start a new Android Studio project** (Начать новый проект Android Studio). В окне **Create New Project** (Создать новый проект) введите текст **BubbleDraw** в поле **Application Name** (Имя приложения), оставьте значение в поле **Company Domain** (Домен компании) без изменения (**example.com** или имя вашего сайта), а затем нажмите кнопку **Next** (Далее).

На этот раз мы хотим выбрать другой уровень API вместо того, который мы использовали для наших предыдущих приложений. В игре «Больше-Меньше» и приложении «Секретные сообщения» использовались графические интерфейсы, которые будут работать на более старых устройствах Android. Однако в этом приложении нам для рисования пузырьков понадобится метод `drawOval()`, для чего требуется уровень API 21 или выше. Выберите пункт **API 21: Android 5.0 (Lollipop)** в качестве минимальной версии SDK.

Дополнительное различие заключается в том, что вместо операций, которые мы использовали в игре «Больше-Меньше» и приложении «Секретные сообщения», сейчас мы будем использовать пустую операцию, поскольку нам не нужен базовый графический интерфейс. Вместо обычного графического интерфейса приложения мы собираемся создать интерактивный холст для рисования с сенсорным интерфейсом. На экране **Add an Activity to Mobile** (Добавить функционал на мобильное устройство) выберите пункт **Empty Activity** (Пустая операция) и нажмите кнопку **Next** (Далее).

Мы по-прежнему, как и в предыдущих приложениях, будем использовать значение `MainActivity` в качестве **Activity Name** («Имя операции»). Однако сейчас сбросьте флажок **Backwards Compatibility** (Обратная совместимость) и нажмите кнопку **Finish**(Готово). Отключение обратной совместимости упростит наш код.

Как и в двух предыдущих версиях приложения «Рисование пузырьков», мы будем использовать два файла на языке Java, чтобы сохранить код, отвечающий за пузырьки, отдельно от основного кода приложения. Как только проект откроется, перейдите

на вкладку **Project** (Проект) в левой части экрана, чтобы отобразить панель **Project Explorer** (Обозреватель проекта), если она не открыта. Затем выберите вкладку **Android** в верхней части панели **Project Explorer** (Обозреватель проекта), а потом перейдите в раздел **app ⇒ java** и найдите основной пакет *BubbleDraw* (не выбирайте пакеты *androidTest* или *test*). Щелкните правой кнопкой мыши по пакету *BubbleDraw* на панели **Project Explorer** (Обозреватель проекта) и выберите пункт **New ⇒ Java Class** (Создать ⇒ Класс Java) в контекстном меню, как показано на рис. 11.2.

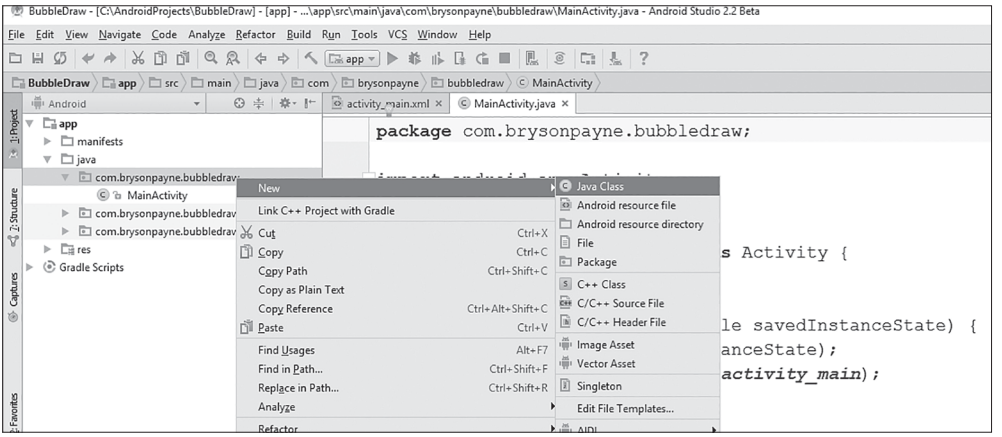


Рис. 11.2. Добавьте новый класс в проект BubbleDraw

В появившемся окне **Create New Class** (Создать новый класс) присвойте создаваемому классу имя `BubbleView`. Для устройства Android, `View` — это любой компонент графического интерфейса пользователя. Класс `BubbleView` будет выполнять ту же функцию, что и класс `BubblePanel` в приложении для компьютера. Весь код рисования пузырьков будет находиться в этом классе.

Окно **Create New Class** (Создать новый класс) среды разработки Android Studio позволяет легко определять суперклассы и интерфейсы. Во-первых, сделаем так, чтобы наш новый класс `BubbleView` унаследовал способность простого вывода графики. В текстовом поле **Superclass** начните вводить значение **`ImageView`**, а затем выберите пункт **ImageView (android.widget)** в списке автозаполнения. Таким образом, наш новый класс будет наследником класса `ImageView`. После щелчка он будет отображаться как `android.widget.ImageView`.

Затем реализуем интерфейс `OnTouchListener`, чтобы наше приложение могло обрабатывать события касания, похожие

на события мыши, которые мы использовали в предыдущих версиях приложения. В текстовом поле **Interface(s)** (Интерфейс(ы)) начните вводить значение **`OnTouchListener`**, а затем выберите пункт **OnTouchListener (android.view.View)** в списке автозаполнения. Как только вы щелкнете мышью по нему, он будет отображаться как `android.view.View.OnTouchListener`.

Нажмите кнопку **ОК**. Теперь вы должны увидеть класс `BubbleView` внутри вашего пакета *com.<вашдомен>.bubbledraw*. Дважды щелкните мышью по классу `BubbleView` на панели **Project Explorer** (Обозреватель проекта), чтобы начать редактирование файла. Возможно, класс `BubbleView` будет подчеркнут красным цветом, поскольку в нем не хватает кода. Однако мы добавим эти недостающие фрагменты в следующих нескольких разделах, по мере дальнейшего программирования приложения.

Создание конструктора BubbleView

Дважды щелкните мышью по вкладке *BubbleView.java*, чтобы развернуть ее на полный экран для удобства редактирования. Давайте начнем создавать класс `BubbleView`, добавляя переменные, похожие на переменные класса `BubblePanel`. Как и в предыдущих версиях приложения, нам понадобится генератор случайных чисел и динамический массив `ArrayList` для рисованных пузырьков, а также несколько целочисленных переменные для хранения начального размера пузырьков и задержки анимации в миллисекундах.

Добавление переменных анимации

Мы реализуем в нашем проекте цвета и скорость пузырьков случайным образом, поэтому внутри открытой фигурной скобки класса `BubbleView` начните вводить значение `private Random`, а затем выберите пункт **Random (java.util)** в списке автозаполнения.

Каждый раз, добавляя в код новый тип объекта, мы будем выбирать элементы из списка автоматического завершения кода. Помните, что функция автозавершения кода программы Android Studio не только помогает быстрее программировать, но и уменьшает количество опечаток и ошибок в именах классов и инструкций импорта.

Завершите объявление новой переменной **`private Random rand = new Random()`**. Затем добавьте каждую из переменных, указанных в приведенном ниже коде. После этого проверьте команды `import` и убедитесь, что они соответствуют указанным ниже:

```
package com.вашдомен.bubbledraw; // Note: your package name may differ
import android.widget.ImageView;
import android.view.View;
import java.util.ArrayList;
import java.util.Random;
public class BubbleView extends ImageView implements View.OnTouchListener {
❶    private Random rand = new Random();
❷    private ArrayList<Bubble> bubbleList;
❸    private int size = 50;
❹    private int delay = 33;
}
```

Эти четыре строки похожи на объявления переменных в верхней части класса `BubblePanel` в главах 9 и 10, с некоторыми изменениями в версии приложения для устройства Android. Генератор случайных чисел в строке ❶ объявляется точно так же, как и в нашей старой версии, поскольку они оба используют `java.util.Random`. То же самое касается и строки ❷, где мы объявляем динамический массив `ArrayList` объектов типа `Bubble`, с именем `bubbleList`. Как и ранее, в этом массиве будут храниться пузырьки, созданные пользователем. Спецификатор типа `Bubble` должен отображаться красным цветом, напоминая, что мы еще не определили класс `Bubble`.

В строке ❸ мы объявляем целочисленную переменную для размера пузырьков. Однако, в отличие от предыдущей версии, мы сделали значение по умолчанию больше для приложения для устройства Android из-за меньшего размера пикселей на мобильных устройствах. Ваш смартфон или планшет часто имеет намного меньший размер экрана и при этом более плотное (пикселей на см) разрешение, чем экран вашего монитора или ноутбука, поэтому мы установили начальный размер пузырьков 50, чтобы сделать их более видимыми. Позже вы сможете отредактировать эту строку и сделать ваши пузырьки больше или меньше, в зависимости от того, как вы хотите, чтобы они отображались на вашем устройстве.

В строке ❹ мы сохранили анимацию со скоростью 30 кадров в секунду, установив величину задержки между кадрами в размере 33 миллисекунд. Вспомните, что для получения скорости анимации

мы делим 1000 миллисекунд на количество кадров в секунду (30 кадров в секунду), чтобы получить количество миллисекунд на кадр, 1000 ÷ 30 = 33.

Графика и анимация на устройстве Android несколько отличаются от ПК-реализации, поэтому нам нужно добавить две новые переменные:

```
public class BubbleView extends ImageView implements View.OnTouchListener {
    private Random rand = new Random();
    private ArrayList<Bubble> bubbleList;
    private int size = 50;
    private int delay = 33;
❶   private Paint myPaint = new Paint();
❷   private Handler h = new Handler();
}
```

В строке ❶ объявляется объект `android.graphics.Paint`, называемый `myPaint`. Рассматривайте его как кисть для рисования пузырьков на экране устройства Android. У вас должен быть объект `Paint`, чтобы рисовать фигуры на объекте `canvas` устройства Android. Нажмите клавишу **Enter** после набора `Paint`, чтобы принять предложение о завершении кода, или щелкните `Paint` после его набора и нажмите сочетание клавиш **Alt+Enter** (⌥ + ↩ в операционной системе macOS) для автоматического импорта класса `android.graphics.Paint`.

Строка ❷ объявляет переменную нового типа — `android.os.Handler` с именем `h`. Убедитесь, что вы импортируете версию `android.os` класса `Handler`, поскольку существуют другие классы с похожими именами. Этот объект `Handler` позволит нам работать с многопоточностью для выполнения анимации. Вы можете рассматривать его как объект, аналогичный объекту `Timer` из версии для компьютера. Однако, в отличие от таймера, обработчик `Handler` позволит нам взаимодействовать с *потоком*, который является отдельным процессом в многозадачной среде, позволяющей запускать несколько приложений одновременно. В отличие от таймера, обработчик `Handler` не нагружает процессор, заставляя его считать время между событиями; вместо этого он освободит ЦП и разрешит выполнение других задач до тех пор, пока не придет время перерисовать другой кадр анимации.

Теперь давайте добавим в приложение конструктор и начнем рисовать пузырьки.

Создание конструктора `BubbleView()`

Следующим шагом будет написание конструктора для класса `BubbleView`. Под последним добавленным объявлением переменных введите следующий код конструктора:

```
public class BubbleView extends ImageView implements View.OnTouchListener {
    private Random rand = new Random();
    private ArrayList<Bubble> bubbleList;
    private int size = 50;
    private int delay = 33;
    private Paint myPaint = new Paint();
    private Handler h = new Handler();
    public BubbleView(Context context, AttributeSet attributeSet) {
        super(context, attributeSet);
    }
}
```

Используйте автоматическое завершение кода для импорта классов `android.content.Context` и `android.util.AttributeSet`. Устройство Android использует эти классы для хранения информации о текущем приложении, и нам нужно импортировать их, чтобы была возможность вызова метода `super()`. Метод `super()` настраивает приложение и холст, вызывая конструктор родительского класса `ImageView`.

Пока что единственная команда, которую мы добавим в конструктор, — это строка с инициализацией динамического массива `bubbleList`:

```
public BubbleView(Context context, AttributeSet attributeSet) {
    super(context, attributeSet);
    bubbleList = new ArrayList<Bubble>();
}
```

Инициализация динамического массива `bubbleList` и установление его равным новому, пустому динамическому массиву `ArrayList` объектов `Bubble` работает так же, как и в предыдущих версиях приложения. Когда пользователь будет касаться экрана, мы будем сохранять новые пузырьки в массиве `bubbleList`.

Подготовка макета к использованию класса BubbleView

Теперь, когда мы начали создавать класс BubbleView, пришло время сообщить нашему файлу макета графического интерфейса, чтобы он отображал BubbleView при запуске приложения. На панели **Project Explorer** (Обозреватель проекта) откройте раздел *app* ⇒ *res* ⇒ *layout* ⇒ *activity_main.xml* и перейдите на вкладку **Text** (Текст) в нижней части окна.

Замените содержимое файла *activity_main.xml* следующим, не забыв поменять имя моего пакета на свое:

```
    <?xml version="1.0" encoding="utf-8"?>
❶   <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"
❷       android: background="#000000"
❸       tools: context="com.вашдомен.bubbledraw.BubbleView">
❹       <com.вашдомен.bubbledraw.BubbleView
❺       android: layout_width="match_parent"
❻       android: layout_height="match_parent"
        />
    </RelativeLayout>
```

В этом приложении используется макет RelativeLayout (блок ❶) по умолчанию, что позволит чуть позже легко разместить другие компоненты графического интерфейса. Значительная часть свойств макета RelativeLayout совпадает с начальными значениями, однако мы добавили цвет фона с помощью ключевого слова background, установив значение #000000 (строка ❷), то есть черный цвет.

Когда вы замените имя пакета com.*вашдомен*.bubbledraw в строках ❸ и ❹ на имя своего пакета, устройство Android дает рекомендации о завершении кода в строке ❹.

Строка ❹ помещает BubbleView в макет, а строки ❺ и ❻ сообщают программе, что его размеры соответствует ширине и высоте окна *activity_main.xml*, которое является родительским объектом для объекта BubbleView.

Файл *action_main.xml* — это представление макета графического интерфейса по умолчанию, загружаемое приложением при запуске.

Поскольку файл *activity_main.xml* загружает объект `BubbleView` в качестве единственного элемента в макете, а его ширина и высота соответствуют размерам макета, то приложение `BubbleView` займет весь экран. Таким образом, после этих изменений файл *action_main.xml* знает, что для показа холста, на котором будут рисоваться пузырьки, требуется вызвать `BubbleView`.

Кстати о пузырьках. Давайте снова используем класс `Bubble` из версии для компьютера.

Изменение класса Bubble

Откройте проект *BubbleDrawGUI* из главы 10 в программе Eclipse, чтобы добраться до старого класса `Bubble`. Откройте класс *BubblePanel.java* и промотайте до нижней части файла, где мы определили класс `Bubble`. Скопируйте весь исходный код класса, от ключевых слов `private class Bubble` до предпоследней закрывающей фигурной скобки. Последняя фигурная скобка в файле является закрывающей для класса `BubblePanel`, поэтому убедитесь, что вы скопировали только закрывающие скобки для метода `update()` и класса `Bubble`.

Скопировав класс `Bubble`, вернитесь в программу Android Studio и поместите курсор непосредственно после закрывающей фигурной скобки конструктора `BubbleView()`. Нажмите клавишу **Enter**, чтобы вставить пустую строку перед последней закрывающей скобкой в нижней части файла.

Значительная часть кода версии класса `Bubble` для компьютера будет работать, но нам нужно внести несколько изменений, чтобы отразить различия в том, как устройство Android рисует графику. Давайте начнем с начала класса `Bubble`. В графике устройств Android цвета хранятся как целочисленные значения, а не как объекты класса `Color`, поэтому измените ключевые слова `private Color color` на `private int color` следующим образом:

```
private class Bubble {
    private int x;
    private int y;
    private int size;
    private int color;
    private int xspeed, yspeed;
    private final int MAX_SPEED = 5;
```

Все остальные переменные остануся без изменений.

Нам также необходимо изменить установку цвета в конструкторе `Bubble()`. Удалите ключевые слова `new Color` и напишите `Color.argb` следующим образом:

```
public Bubble(int newX, int newY, int newSize) {
    x = newX;
    y = newY;
    size = newSize;
    color = Color.argb(rand.nextInt(256),
            rand.nextInt(256),
            rand.nextInt(256),
            rand.nextInt(256));
    xspeed = rand.nextInt(MAX_SPEED * 2) - MAX_SPEED;
    yspeed = rand.nextInt(MAX_SPEED * 2) - MAX_SPEED;
}
```

Обязательно удалите ключевое слово `new`, так как метод `Color.argb()` не создает новый объект. Вместо этого метод `Color.argb()` преобразует четыре значения *ARGB* (*альфа, красный, зеленый* и *синий*) в единое целое число, которое можно использовать для изменения цвета рисования на устройстве Android.

В этом приложении мы использовали класс `Color` в первый раз, поэтому в текстовом редакторе Android Studio он будет выделен красным цветом. Вы можете вручную добавить команду **`import android.graphics.Color;`** в инструкции импорта в верхней части файла, или вы можете щелкнуть мышью по слову **Color** и нажать сочетание клавиш **Alt+Enter** (⌥ + ⏎ в операционной системе macOS), чтобы программа Android Studio выполнила импорт автоматически. Нажатие сочетания клавиш **Alt+Enter** аналогично принятию предложения редактора о завершении кода, за исключением того, что вы можете использовать сочетание клавиш **Alt+Enter** для импорта класса даже *после* ввода кода.

Затем нам нужно полностью изменить метод `draw()` в классе `Bubble`. Замените скопированный метод `draw()` следующим:

```
            xspeed = rand.nextInt(MAX_SPEED * 2) - MAX_SPEED;
            yspeed = rand.nextInt(MAX_SPEED * 2) - MAX_SPEED;
        }
❶       public void draw(Canvas canvas) {
❷           myPaint.setColor(color);
```

```
❸                canvas.drawOval(x - size/2, y – size/2,
                        x + size/2, y + size/2, myPaint);
            }
            public void update() {
```

В строке ❶ метод `draw()` принимает параметр типа `android.graphics.Canvas` вместо `java.awt.Graphics`. Обязательно импортируйте класс `Canvas` либо во время ввода, либо после него, щелкнув мышью по введенной команде, и после этого нажав сочетание клавиш **Alt+Enter** (⌥ + ⏎ в операционной системе macOS).

В строке ❷ мы устанавливаем цвет объекта `myPaint` для использования этого значения как значения `color` пузырька.

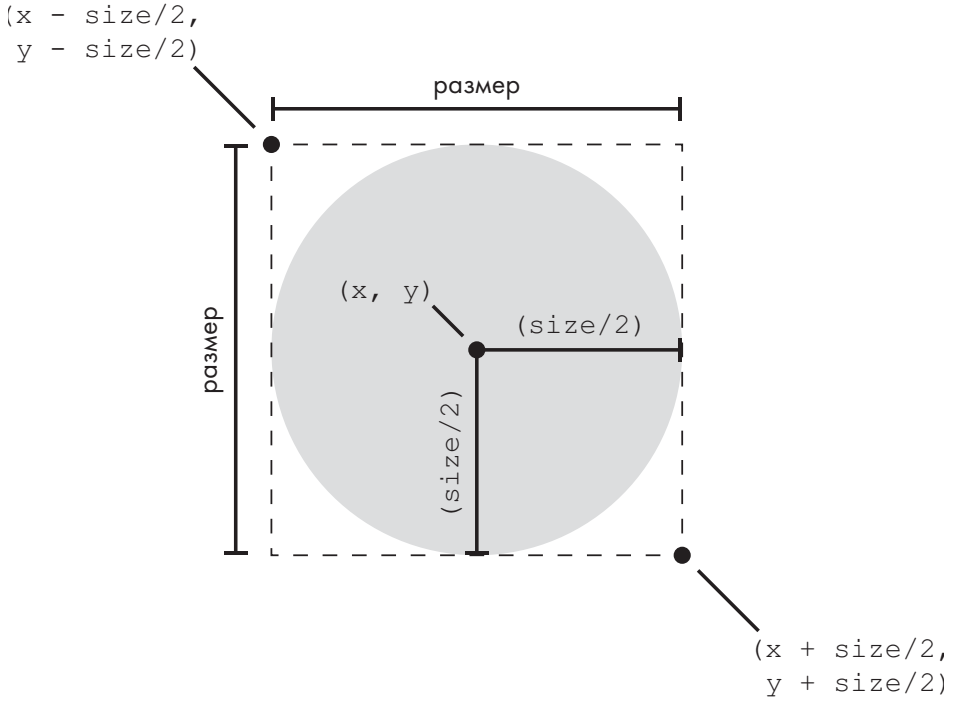


Рис. 11.3. Метод `drawOval()` на Android использует координаты верхнего левого и правого нижнего угла воображаемого прямоугольника, в который вписан овал, вместо координат левого верхнего угла, ширины и высоты, как в пакете Swing для ПК

Строка ❸ в нескольких местах отличается от аналогичной строки в версии для компьютера. Во-первых, команда нарисовать овал в устройстве Android — это `drawOval()`, а не `fillOval()`. Во-вторых, мы указываем ограничивающий прямоугольник, в который будет вписан овал, используя значения его *левой*, *верхней*,

правой и *нижней* границы вместо значений *левой* и *верхней* границы, а также *ширины* и *высоты*. Левая и верхняя координата остаются теми же, что и ранее, а именно `x − size/2` и `y − size/2`. (Помните, что мы вычитаем половину ширины и высоты пузырька, чтобы его центр располагался в том месте (с координатами `x` и `y`), где пользователь коснулся экрана.) Координата правой стороны ограничивающего пузырька прямоугольника равна `x + size/2`, а нижняя — `y + size/2`, как показано на рис. 11.3. Вместо использования ширины и высоты для вычисления координат нижнего правого угла прямоугольника, в который вписан овал, как это было в версии для ПК, устройство Android требует, чтобы мы указали координаты (`x` и `y`) нижнего правого угла, или `x + size/2`, `y + size/2`. Наконец, метод `drawOval()` требует объект класса `Paint`, который мы передаем как `myPaint`.

Это все изменения, необходимые для переноса класса `Bubble` из версии для компьютера в код для устройства Android. Сохраните свой файл после внесения этих изменений. Далее мы нарисуем все пузырьки на экране.

Рисование на устройстве Android с помощью метода `onDraw()`

Мы хотим проверить способность приложения рисовать пузырьки на экране, поэтому сейчас добавим в наш класс `BubbleView` метод `onDraw()`. Метод `onDraw()` в классе `View` аналогичен методу `paintComponent()` для панели `JPanel`: он сообщает программе, что рисовать после каждого обновления экрана.

Мы хотим нарисовать список пузырьков, поэтому добавьте следующий код в файл *BubbleView.java* ниже конструктора `BubbleView()` и выше класса `Bubble`:

```
    public BubbleView(Context context, AttributeSet attributeSet) {
        super(context, attributeSet);
        bubbleList = new ArrayList<Bubble>();
    }
❶   protected void onDraw(Canvas canvas) {
❷       for (Bubble b: bubbleList)
            b.draw(canvas);
    }
    private class Bubble {
```

В строке ❶ мы определяем метод `onDraw()` как защищенный (`protected`) с одним параметром — объектом класса `Canvas` — поскольку он должен точно соответствовать методу `View.onDraw()` по умолчанию, который мы переопределяем. Нам нужен этот метод, поскольку он требуется во всех подклассах класса `View`, а `BubbleView` — это дочерний класс класса `ImageView`, который является подклассом класса `View`. Метод `onDraw()` будет вызываться каждый раз, когда экран, содержащий наш `BubbleView`, нуждается в обновлении.

Внутри метода `onDraw()` в блоке ❷ повторно используется цикл `for-each`, который вызывает функцию `draw()` для каждого пузырька. Строку ❷ можно прочитать следующим образом: «Для каждого объекта `b` класса `Bubble` в динамическом массиве `bubbleList` нарисуйте `b` на объекте `Canvas` устройства Android».

Осталось всего пара шагов, прежде чем мы сможем проверить способность нашего приложения рисовать разноцветные пузырьки. Давайте пока оставим их в покое и проведем предварительное бета-тестирование приложения «Рисование пузырьков» для устройства Android.

Тестирование приложения сотней пузырьков

В первой версии приложения «Рисование пузырьков» для компьютера мы написали короткий метод под названием `testBubbles()`. Он выглядел следующим образом:

```
public void testBubbles() {
    for(int n = 0; n < 100; n++) {
        int x = rand.nextInt(600);
        int y = rand.nextInt(400);
        int size = rand.nextInt(50);
        bubbleList.add(new Bubble(x, y, size));
    }
    repaint();
}
```

Метод `testBubbles()` мы написали для того, чтобы понять до использования обработчиков событий мыши и таймера, можем ли мы рисовать пузырьки на экране. Давайте сделаем то же самое для Android-версии приложения.

Добавление метода `testBubbles()`

Прежде всего, давайте добавим немного модифицированную версию функции `testBubbles()` под методом `onDraw()` в файле *BubbleView.java*:

```
    protected void onDraw(Canvas canvas) {
        for (Bubble b: bubbleList)
            b.draw(canvas);
    }
    public void testBubbles() {
        for(int n = 0; n < 100; n++) {
❶           int x = rand.nextInt(600);
❷           int y = rand.nextInt(600);
❸           int s = rand.nextInt(size) + size;
❹           bubbleList.add(new Bubble(x, y, s));
        }
❺       invalidate();
    }
    private class Bubble {
```

Первые две строки идентичны версии метода `testBubbles()`, реализованной в среде разработки Eclipse. Эти строки объявляют функцию и настраивают цикл `for` для запуска 100 раз. В строке ❶ мы оставляем тот же диапазон значений `x` и устанавливаем его верхнюю границу в размере `600`, однако вы можете сделать его больше, если знаете разрешение вашего устройства. В строке ❷ мы изменяем диапазон для случайного значения `y` для вертикальной координаты пузырька до `600`.

В строке ❸ мы генерируем пузырьки большего размера, прибавляя к начальной величине значения `size` случайное число в диапазоне от `0` до значения переменной `size`. В данном случае у нас будут получаться пузырьки диаметром от 50 до 100 пикселей.

В строке ❹ мы создаем новый объект класса `Bubble`, используя три только что сгенерированных случайных значения, и добавляем его в список `bubbleList`.

Наконец, в строке ❺ мы используем новую функцию `invalidate()`, которая работает аналогично функции `repaint()` в версии игры «Рисование пузырьков» для компьютера. Она сообщает программе, что экран необходимо обновить. Функция `invalidate()` очищает экран и вызывает

метод `onDraw()`, который отрисует все пузырьки из списка `bubbleList`.

Теперь давайте добавим вызов метода `testBubbles()` в конструктор `BubbleView()`:

```
public BubbleView(Context context, AttributeSet attributeSet) {
    super(context, attributeSet);
    bubbleList = new ArrayList<Bubble>();
    testBubbles();
}
```

Теперь при загрузке приложения будет вызван метод `testBubbles()`, который заполнит динамический массив `bubbleList` 100 случайными пузырьками.

Исправление ошибки `OnTouchListener` интерфейса

Осталась лишь одна проблема, которая мешает нам протестировать наше приложение: класс `BubbleView` по-прежнему подчеркнут красным цветом, таким образом, среда разработки сообщает о возможной ошибке компиляции. Установив указатель мыши на строку `public class BubbleView`, вы увидите сообщение об ошибке, в котором говорится, что в классе `BubbleView` не реализован метод `onTouch()` интерфейса `OnTouchListener`. Другими словами, эта ошибка напоминает нам о том, что мы включили интерфейс `OnTouchListener` в определение класса `BubbleView`, но пока еще не добавили метод `onTouch()` для обработки событий касания.

Чтобы исправить эту ошибку, установите указатель мыши на подчеркнутую строку и щелкните мышью по значку предупреждения в виде красной лампочки. В появившемся под красной лампочкой контекстном меню выберите пункт **Implement methods** (Реализовать методы). Появится окно с запросом выбора, какие методы должны быть реализованы, как показано на рис. 11.4.

Нажмите кнопку **ОК**, редактор Android Studio добавит в ваш код метод `onTouch()`, и ошибка будет исправлена:

```
@Override
public boolean onTouch(View view, MotionEvent motionEvent) {
return false;
}
```

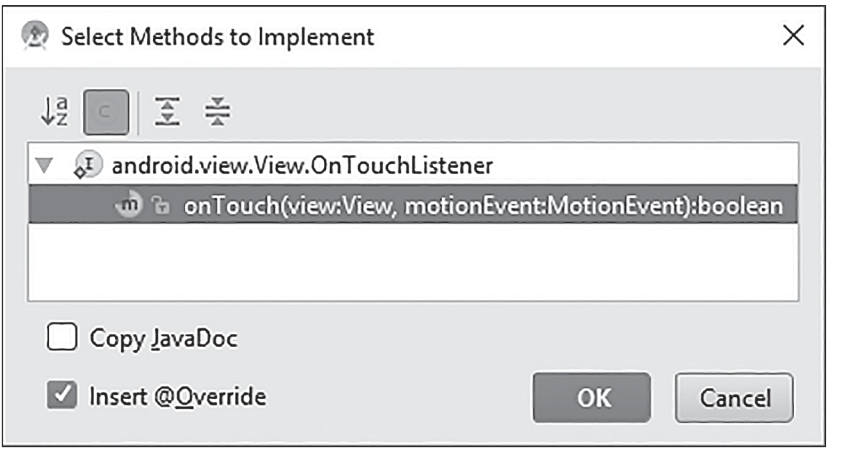


Рис. 11.4. Среда разработки Android Studio реализует метод `onTouch()` для завершения реализации интерфейса `OnTouchListener`

Если добавленный редактором Android Studio код не соответствует представленному здесь, исправьте имена параметров в вашем коде, чтобы они назывались `view` и `motionEvent`, как приведено в листинге. Позже мы модифицируем этот метод для обработки событий касания, но пока давайте запустим наше приложение в тестовом режиме, чтобы увидеть, как оно нарисует 100 пузырьков.

Запуск приложения «Рисование пузырьков»

Нажмите зеленую кнопку запуска, чтобы протестировать приложение. В окне **Select Deployment Target** (Выбор цели развертывания) выберите свой эмулятор или устройство, как мы делали в разделе «Запуск приложения в эмуляторе Android» главы 4. Я выбрал эмулятор **Nexus 6P**, показанный на рис. 11.5.

Нажмите кнопку **ОК**, чтобы запустить ваше устройство и развернуть текущую версию приложения «Рисование пузырьков». Вы должны увидеть пузырьки в левой верхней части экрана, как показано на рис. 11.6. Пузырьки появляются в левом верхнем углу, потому что мы генерировали значения координат `x` и `y` только в диапазоне между `0` и `600`, а большинство устройств Android могут отображать 1000 или более пикселей по ширине и высоте. Помните, что координата (0,0) при программировании находится в левом верхнем углу.

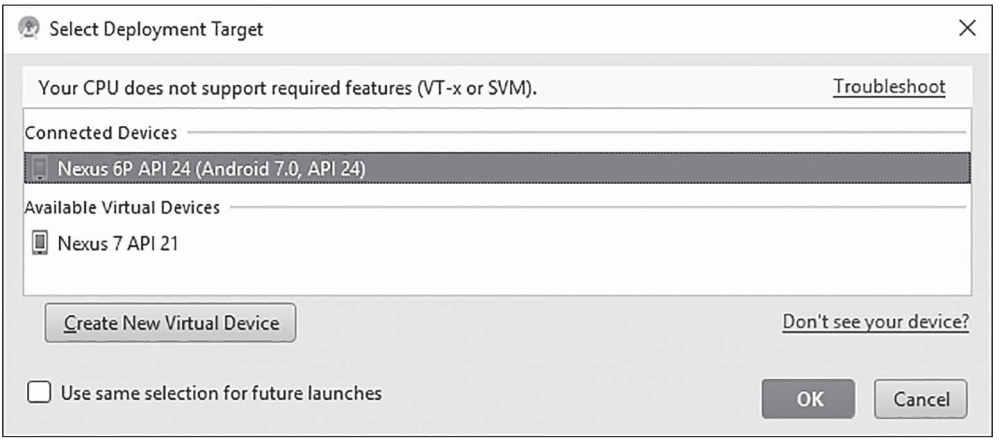


Рис. 11.5. Нажмите кнопку запуска, чтобы скомпилировать и запустить приложение. Затем выберите свой эмулятор или устройство и нажмите кнопку **OK**

Рис. 11.6. Успешный тестовый запуск текущей версии приложения «Рисование пузырьков»; в левом верхнем углу экрана появились 100 пузырьков

Подобно первому тестовому запуску версии приложения для компьютера, анимация и реакция на прикосновения отсутствуют,

поскольку мы их еще не сделали. Но наше приложение работает на эмуляторе и рисует пузырьки на экране.

Сейчас сделаем анимацию, чтобы пузыри перемещались. А затем добавим последний штрих — реакцию на касание!

Использование поточной анимации и многозадачности в программе на языке Java

Многопоточность, которую мы будем использовать для достижения плавной анимации в приложении «Рисование пузырьков» для устройства Android, ничем не отличается от многопоточности, которая используется в любом другом приложении с многозадачностью, написанном на языке Java. Мы уже говорили, что одним из преимуществ использования многопоточности для анимации является то, что она не занимает процессор между выводом кадров. Но многопоточность также может использоваться в любых других разрабатываемых вами приложениях, в которых запускается сразу несколько процессов, например, приложений, которые в фоновом режиме обращаются к базе данных или загружают файл. Использование многопоточности позволяет вашему приложению выполнять фоновые задачи без зависания графического интерфейса, ожидающего завершения процесса.

Многопоточность особенно важна для таких устройств, как смартфоны и планшеты с ограниченной вычислительной мощностью. Зависающие приложения особенно сильно раздражают, а использование многопоточности поможет приложению «Рисование пузырьков» не стать одним из них.

Объект `Handler h`, созданный ранее в этой главе, позволяет нам общаться с потоком. В этом приложении мы создадим поток, отвечающий за анимацию, который будет обновлять координаты всех пузырьков. Затем мы будем использовать наш обработчик `h`, для запуска этого потока. Это позволит обработчику и потоку реализовать анимацию так, как это было сделано с помощью таймера `Timer` в предыдущих главах, но не занимать процессор между кадрами.

Добавить потоки в приложение на языке Java можно двумя способами: путем расширения класса `Thread` или реализацией интерфейса `Runnable`. Мы выберем второй вариант. Класс, реализующий интерфейс `Runnable()`, нуждается в методе `run()`, который сообщает потоку, что делать, когда поток запущен.

В нашем приложении мы хотим, чтобы метод `run()` выполнял анимацию, перемещая пузырьки и перерисовывая экран. Давайте создадим объект `Runnable` с именем `r` и поместим его прямо под конструктором `BubbleView()`:

```
    public BubbleView(Context context, AttributeSet attributeSet) {
        super(context, attributeSet);
        bubbleList = new ArrayList<Bubble>();
        testBubbles();
    }
❶   private Runnable r = new Runnable() {
        @Override
❷       public void run() {
        }
❸   };
    protected void onDraw(Canvas canvas) {
```

Когда вы начнете вводить вторую половину объявления в строке ❶, появится всплывающее окно с предложениями по завершению кода нового объекта `Runnable()`. Выберите вариант `java.lang.Runnable` для завершения кода, и редактор Android Studio автоматически добавит каркас метода `public void run()` для вас (блок ❷). Обратите внимание на точку с запятой после завершающей фигурной скобки объекта `Runnable` в строке ❸ — она необходима, поскольку мы определяем переменную r и одновременно присваиваем ей новый объект `Runnable`. Точка с запятой в строке ❸ фактически завершает команду, которую мы начали в строке ❶. Автоматическое завершение кода не добавляет точку с запятой после закрывающей фигурной скобки в строке ❸, поэтому, чтобы избежать ошибки компилятора, убедитесь, что вы поставили ее самостоятельно.

Теперь нам необходимо добавить код внутри метода `run()`, чтобы сообщить компилятору языка Java, какие действия нам нужны, когда вызывается наш поток `Runnable r`:

```
    private Runnable r = new Runnable() {
        @Override
        public void run() {
❶           for(Bubble b: bubbleList)
❷               b.update();
```

```
❸           invalidate();
        }
    };
    protected void onDraw(Canvas canvas) {
        for (Bubble b: bubbleList)
            b.draw(canvas);
    }
```

В строке ❶ мы снова используем цикл `for-each` для обхода всех пузырьков `b` класса `Bubble` в динамическом массиве `bubbleList`. На каждой итерации цикла мы вызываем метод `b.update()` (строка ❷) для обновления координат конкретного пузырька для следующего кадра анимации.

В строке ❸, после завершения работы цикла мы вызываем функцию `invalidate()` для очистки экрана и перерисовки экрана методом `onDraw()`.

Последним шагом для добавления анимации на основе потоков является подключение обработчика `h` к потоку `Runnable r`. Мы сделаем это в конце метода `onDraw()`:

```
    protected void onDraw(Canvas canvas) {
        for (Bubble b: bubbleList)
            b.draw(canvas);
        h.postDelayed(r, delay);
    }
```

Метод `postDelayed()` отправляет сообщение от обработчика в наш поток `r`, указывая ему снова запустится после задержки в 33 миллисекунды (значение переменной `delay`). Сохраните эти изменения и запустите приложение еще раз. Вы увидите, как 100 тестовых пузырьков медленно расползаются по экрану, как показано на рис. 11.7.

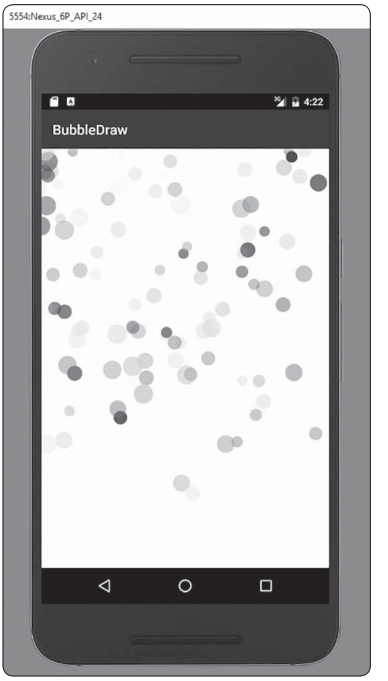


Рис. 11.7. Идем к успеху! Наши пузыри двигаются с помощью многопоточной анимации

Даже при скорости 30 кадров в секунду ваши самые быстрые пузырьки, вероятно, кажутся довольно медленными. Это связано с большим количеством пикселей на вашем устройстве Android или эмуляторе. Вы можете вспомнить,

что мы установили величину максимальной скорости MAX_SPEED в классе Bubble всего 5 пикселей за кадр. У смартфона Nexus 6P, который мы эмулируем, разрешение экрана составляет 1440 × 2560 пикселей, то есть даже самым быстрым пузырькам потребуется более 500 кадров или более 15 секунд для перемещения по всей длине экрана.

Давайте немного ускорим процесс, установив более высокое значение константы MAX_SPEED, например 15 пикселей за кадр:

```
private class Bubble {
    private int x;
    private int y;
    private int size;
    private int color;
    private int xspeed, yspeed;
    private final int MAX_SPEED = 15;
```

Сохраните файл и запустите его снова. Теперь ваши пузырьки будут двигаться по экрану быстрее. Попробуйте увеличить или уменьшить значение константы, чтобы получить лучшую, с вашей точки зрения, картину.

Таким образом мы реализовали анимацию. Давайте добавим обработку касаний для взаимодействия с пользователем.

Рисование касаниями

Приложение «Рисование пузырьков» для компьютеров было настолько интересным, во многом из-за того, что мы могли щелкнуть мышью и нарисовать пузырьки в любом месте. Мы собираемся добиться того же от приложения на устройстве Android, а затем сделаем его еще круче, когда добавим множественные касания!

Вы уже видели, куда нам нужно добавить код обработчика событий касания — в недавно добавленную функцию onTouch().

Чтобы обрабатывать события касания, нам нужно знать координаты точки касания. Затем мы добавим пузырек в это место.

Чтобы определить координаты касания пользователя, мы можем использовать методы motionEvent.getX() и motionEvent.getY(). Давайте напишем метод onTouch() полностью, а затем обсудим его:

```
    public boolean onTouch(View view, MotionEvent motionEvent) {
❶       int x = (int) motionEvent.getX();
❷       int y = (int) motionEvent.getY();
❸       int s = rand.nextInt(size) + size;
❹       bubbleList.add(new Bubble(x, y, s));
❺       return true;
    }
```

В строке ❶ мы получаем координату x места касания пользователя с помощью метода `motionEvent.getX()`. Однако обратите внимание, что мы должны привести полученное значение к целому числу — метод `motionEvent.getX()` в устройстве Android возвращает значение с плавающей точкой, поэтому мы приводим его с помощью ключевого слова (`int`). Координата `y` в строке ❷ получается аналогичным образом. В строке ❸ мы генерируем случайный размер, как и в методе `testBubbles()`, и сохраняем его в переменной `s`.

В строке ❹ мы создаем объект `Bubble` с заданными значениями `x`, `y` и `s` и добавляем его в динамический массив `bubbleList`.

Последнюю строку ❺ нужно пояснить. Обратите внимание, что метод `onTouch()` возвращает значение типа `boolean`. Это означает, что метод `onTouch()` должен возвращать значение `true` или `false`. В устройстве Android метод `onTouch()` должен возвращать значение `true`, если вы полностью обработали событие касания. Если вы хотите, чтобы устройство Android обрабатывало события касания, такие как прокрутку или масштабирование, то необходимо вернуть значение `false` после того, как закончил работу обработчик события `onTouch()`.

В нашем приложении не нужно, чтобы устройство Android прокручивало экран при касаниях пользователя — мы полностью обработали событие касания, добавив пузырек в место, которого коснулся пользователь, поэтому мы возвращаем значение `true`.

Последний шаг аналогичен шагам для слушателя мыши в версии приложения «Рисование пузырьков» для компьютера: нам нужно добавить слушатель в конструктор. Найдите в файле конструктор `BubbleView()`, закомментируйте функцию `testBubbles()` и добавьте следующую строку кода:

```
    public BubbleView(Context context, AttributeSet attributeSet) {
        super(context, attributeSet);
```

```
bubbleList = new ArrayList<Bubble>();
    // testBubbles();
    setOnTouchListener(this);
}
```

Мы закомментировали вызов метода `testBubbles()`, поскольку нам больше не нужны 100 тестовых пузырьков. Мы собираемся добавлять пузырьки, касаясь пальцами экрана на нашем устройстве Android или щелкая и перетаскивая мышь, имитируя касания на эмуляторе устройства Android. Команда `setOnTouchListener(this)` добавляет слушатель к событиям касания, при этом ключевое слово `this` говорит компилятору языка Java, что объект класса `BubbleView` будет обрабатывать события касания *сам*.

Рис. 11.8. Щелкните и перетащите мышь, чтобы имитировать на эмуляторе касание одним пальцем экрана, и вы увидите, как пузырьки разлетаются в разные стороны

Сделав эти изменения можно снова запустить наше приложение. Сохраните свой код и запустите приложение на своем эмуляторе. Щелкайте мышью и перетаскивайте ее по окну эмулятора, имитируя движения вашего пальца по экрану. Вы увидите, как пузырьки разлетаются от места, которого вы касаетесь, как показано на рис. 11.8.

Вы можете запустить это приложение на своем реальном устройстве Android. Мы рассмотрим, как это сделать в следующем разделе, добавив возможность обработки одновременно нескольких касаний экране в одно и то же время.

Использование множественных касаний для рисования одновременно 10 пальцами!

Возможно, вам доводилось играть с приложением, которое использует множественные касания. Например, игру для двух игроков, в которой вы управляете объектами на одной стороне экрана

одним или несколькими пальцами, а ваш друг управляет объектами на другой стороне экрана. Если такой опыт у вас был, то вы понимаете огромную мощь, которую множественные касания приносят в приложения и игры.

Надо сказать, что код для обработки нескольких событий касаний в устройстве Android почти так же прост, как и код для обработки одного касания. Фактически, мы просто добавим команду и изменим пару строк в методе `onTouch()`, и наше приложение будет обрабатывать события множественных касаний.

Метод `getPointerCount()` сообщает нам, сколько событий касания происходит одновременно. Он возвращает количество *указателей* — событий касаний или пальцев на экране в текущем событии `MotionEvent`.

Мы можем добавить цикл `for` для добавления пузырьков для каждого указателя в текущем событии касания:

```
    public boolean onTouch(View view, MotionEvent motionEvent) {
❶       for (int n = 0; n < motionEvent.getPointerCount(); n++) {
❷           int x = (int) motionEvent.getX(n);
❸           int y = (int) motionEvent.getY(n);
            int s = rand.nextInt(size) + size;
            bubbleList.add(new Bubble(x, y, s));
❹       }
        return true;
    }
```

В строке ❶ мы добавляем цикл `for`, в котором переменная `n` изменяется от `0` до числа указателей в текущем событии касания. Функция `motionEvent.getPointerCount()` возвращает количество указателей. Если есть только одно событие касания, метод `getPointerCount()` вернет `1`, и цикл будет работать только один раз, для `n = 0`. При двух событиях касания `n` будет равняться `0` и `1` и так далее.

В строке ❷ мы модифицируем метод `motionEvent.getX()`, вставив `n` в круглые скобки после `getX()`. Указатели пронумерованы в событиях касания, поэтому передав переменную `n` в качестве аргумента в метод `motionEvent.getX()`, в качестве результата работы метода мы получим координату `x` *n*-го указателя касания в текущем событии касания. Таким образом, `getX(0)` вернет координату `x` первого касания, `getX`❶ вернет координату `x` второго касания и так далее. В строке ❸ мы делаем то же самое для

координат `y` каждого касания с помощью методов `getY(n)`. Наконец, не забудьте закрыть фигурную скобку в конце тела цикла `for` в строке ❹.

Это все, что нужно! Язык программирования Java и ОС Android упрощают обработку событий множественных касаний.

Тестирование событий множественных касаний на устройстве Android

Сохраните ваш код. На этот раз мы запустим приложение на реальном устройстве Android. К сожалению, мы не сможем имитировать события множественных касаний с помощью одной мыши на эмуляторе устройства Android, поэтому вам придется запустить приложение на реальном смартфоне или планшете.

Прежде всего, подключите с помощью USB-кабеля устройство Android к компьютеру, на котором запущена среда разработки Android Studio. Разрешите отладку через USB с компьютера на устройстве (см. раздел «Запуск приложения на реальном устройстве Android» главы 4). Если эмулятор работает, закройте там приложение «Рисование пузырьков». Затем нажмите кнопку запуска или выберите команду меню **Run** ⇒ **Run *'app'*** (Выполнить ⇒ Выполнить *'приложение'*).

В окне **Select Deployment Target** (Выбор цели развертывания) найдите и выберите ваше устройство (у меня это смартфон Asus Nexus 7) и нажмите кнопку **OK**. Программа Android Studio перекомпилирует приложение и развернет его на вашем устройстве.

Когда приложение запускается, оно выглядит как черный экран с надписью **BubbleDraw** в строке заголовка. Однако если вы коснетесь экрана одним или несколькими пальцами, все станет намного интереснее, как показано на рис. 11.9.

Когда вы касаетесь пальцами экрана, кажется, что пузырьки вылетают от ваших прикосновений, не важно, используете ли вы один палец, два, четыре или даже десять!

Есть еще одна интересная функция, о которой мы еще не говорили: чтобы очистить экран, просто поверните устройство набок (убедитесь, что положение фиксации ориентации вашего устройства установлено в положение **Auto-rotate** (Авто-поворот), а не **Portrait** (Портрет)). Изменение ориентации заставляет приложение повторно инициализировать класс `BubbleView`, что приводит к сбросу массива `bubbleList` и очистке экрана. Это

классный эффект, и он делает приложение еще более осязаемым и интерактивным.

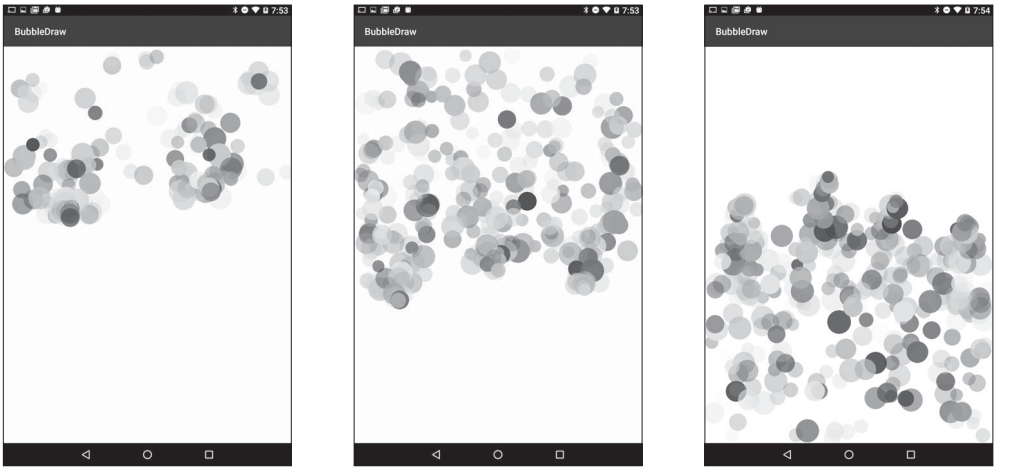


Рис. 11.9. Прикосновение к экрану устройства Android двумя (слева), тремя (в центре) или четырьмя пальцами (справа) создает несколько потоков разноцветных, отражающихся от краев пузырьков

Осталась еще одна настройка, которую мы сделаем в нашем приложении «Рисование пузырьков». Мы заменим значок приложения, который по умолчанию присвоило ему устройство Android, на свой собственный.

Изменение значка запуска приложения

До сих пор все наши приложения использовали значки запуска, предоставляемые устройством Android по умолчанию, на которых изображен дружелюбный зеленый дроид — символ устройств Android. Но что, если вы хотите использовать свой собственный значок для приложения «Рисование пузырьков», например, логотип вашей компании или скриншот из приложения, как показано на рис. 11.10?

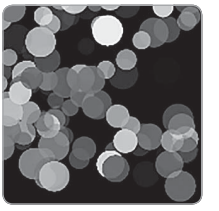


Рис. 11.10. Значок для приложения

Чтобы ваше приложение использовало свой собственный значок, вам нужно создать файл *ic_launcher.png*, поместить его в папку *app* ⇒ *src* ⇒ *main* ⇒ *res* ⇒ *drawable*, а затем изменить код файла *AndroidManifest.xml*, в котором определить новый файл как значок запуска вашего приложения.

Создание значка

По умолчанию устройство Android для значка запуска приложения использует файл с именем *ic_launcher.png*. Если вы откроете папку *main* ⇒ *res* ⇒ *app* ⇒ *src* ⇒ *mipmap*, вы увидите несколько файлов *ic_launcher.png* разных размеров, хранящихся внутри папок с именами *mipmap_mdpi*, *mipmap_xxhdpi* и так далее. Это связано с различными размерами экрана различных смартфонов и планшетов.

Мы также назовем наше новое изображение *ic_launcher.png*, просто для удобства. Используя графический редактор, создайте файл формата PNG с изображением, которое вы хотите использовать в качестве значка запуска приложения. (На сайте **www.gimp.org** доступен отличный бесплатный редактор изображений, а с помощью сайта **www.pixlr.com/editor/** можно бесплатно редактировать изображения прямо в Интернете.) Лучше всего использовать квадратное изображение, но это необязательно. Размер моего изображение составляет 156 × 156 пикселей, но, в принципе, подойдут изображения, размер которых лежит в диапазоне между 64 × 64 и 256 × 256 пикселей.

Если вы хотите использовать скриншот приложения, как я, сделать снимок экрана очень просто: нажмите и удерживайте кнопку режима ожидания устройства Android и кнопку уменьшения громкости одновременно до тех пор, пока экран не «мигнет», давая понять, что изображение сохранено. В приложении «Фотографии» найдите папку «Скриншоты», в которой вы должны увидеть сохраненное изображение. Вы можете отредактировать изображение, сделав его размером 256 × 256 пикселей или меньше, либо на ПК, предварительно отправив его по электронной почте самому себе, либо непосредственно на устройстве Android.

Сохраните или экспортируйте свой файл из редактора изображений под именем *ic_launcher.png*. На следующем шаге мы скопируем это изображение и вставим его в проект BubbleDraw в редакторе Android Studio.

После того как вы создали свой собственный файл со значком *ic_launcher.png*, откройте папку, в которой вы его сохранили, и скопируйте файл.

В среде разработки Android Studio на панели **Project Explorer** (Обозреватель проекта) для приложения BubbleDraw найдите папку *app* ⇒ *res* ⇒ *drawable* или *app* ⇒ *src* ⇒ *main* ⇒ *res* ⇒ *drawable* и вставьте в нее новое изображение *ic_launcher.png*, как показано на рис. 11.11.

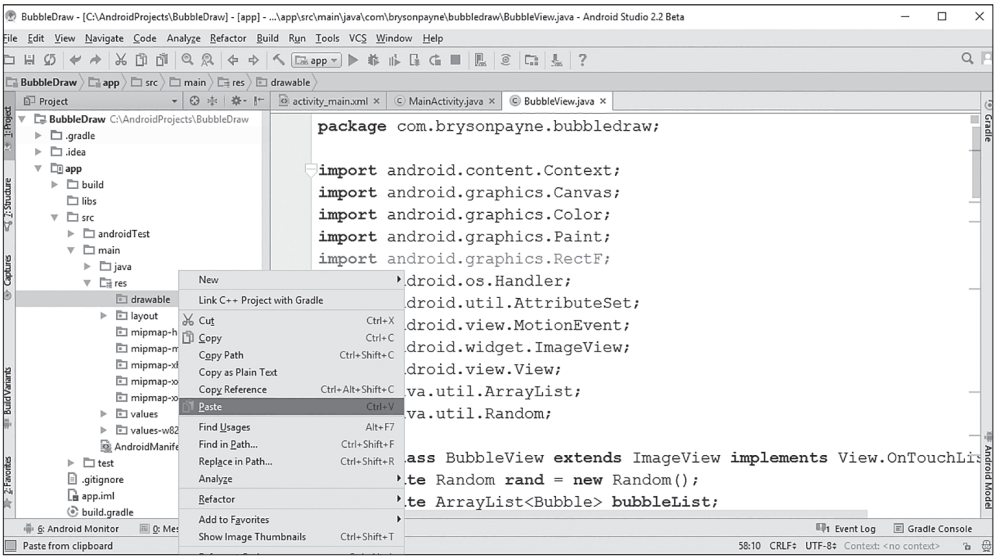


Рис. 11.11. Поместите новый значок приложения в папку *app* ⇒ *src* ⇒ *main* ⇒ *res4* ⇒ *drawable*

Появится окно, проверяющее местоположение файла. Нажмите кнопку **ОК**.

После того, как значок будет скопирован в папку *drawable*, вы сможете открыть папку и, дважды щелкнув мышью по файлу *ic_launcher.png*, просмотреть его в программе Android Studio.

Теперь у нас появился новый значок, который мы создали в структуре проекта BubbleDraw, поэтому мы можем дать указание устройству Android использовать это изображение в качестве значка для нашего приложения.

Отображение нового значка

Чтобы сделать этот последний шаг, мы отредактируем файл *AndroidManifest.xml*. Файл *AndroidManifest.xml* описывает базовую структуру, свойства и функциональность вашего приложения в операционной системе Android.

Найдите папку *app* ⇒ *src* ⇒ *main* (или *app* ⇒ *manifests*) и откройте файл *AndroidManifest.xml*. В верхней части файла найдите запись `android: icon` и замените ее значение ссылкой на новый файл, который вы только что разместили в папке *drawable*:

```
<application
    android: allowBackup=»true»
    android: icon="@drawable/ic_launcher"
```

Мы сообщаем устройству Android, чтобы необходимо заглянуть в папку *drawable* этого проекта и найти там файл изображения с именем *ic_launcher*. В манифесте расширение *.png* мы убираем.

Теперь сохраните файл *AndroidManifest.xml* и нажмите кнопку запуска, чтобы скомпилировать и развернуть приложение с новым значком. После обновления приложения на вашем эмуляторе или устройстве Android вы увидите новый значок для приложения «Рисование пузырьков», как показано на рис. 11.12.

Созданное нами изображение в формате PNG теперь является значком для приложения в устройстве Android.

Изменение названия приложения

Вы также можете изменить название приложения, которое отображается под его значком и в строке заголовка внутри приложения. На вкладке в левой части панели **Project Explorer** (Обозреватель проекта) откройте файл *strings.xml*, расположенный в папке *app* ⇒ *src* ⇒ *main* ⇒ *res* ⇒ *values*. В файле *strings.xml* есть строка `app_name`, содержащая имя приложения, которое используется как на значке запуска, как показано на рис. 11.12, так и в строке заголовка приложения во время его работы.

Поскольку название **BubbleDraw** без пробела между словами выглядит немного странно, мы добавим пробел между словами в строке XML-кода, которая определяет переменную `app_name` в *strings.xml*:

```
<resources>
<string name="app_name">Bubble Draw</string>
</resources>
```

Естественно, вы можете назвать приложение как угодно, например «Рисование пузырьков — программа *ваше имя*», но только около 11 или 12 символов будут показаны под значком, поэтому вы увидите что-то вроде «Ваше имя...» на стартовом экране. Кроме того, следует экранировать символом обратного слеша все специальные символы в имени приложения, включая одинарные кавычки.

Теперь создайте собственные значки для игры «Больше-Меньше» и приложения «Секретные сообщения» Продолжайте просматривать ваши приложения и пробовать новые возможности. Написание кода лучше всего разовьет ваши навыки программирования.

Рис. 11.12. Мы настроили значок запуска для приложения «Рисование пузырьков»

Что вы изучили

С начала главы 1 вы прошли долгий путь. Создав три полноценных приложения с мобильными версиями и версиями для компьютеров, вы не только приобрели значимые навыки в программировании, но и использовали их. При этом вы научились шаг за шагом добавлять все больше и больше функциональных возможностей в приложение, пока оно не начнет делать именно то, что вы хотите. И это, возможно, самое главное.

В этой главе вы укрепили несколько навыков и добавили следующие способности в ваш арсенал инструментов программирования:

- использование графики на устройстве Android;
- добавление нового класса в проект в редакторе Android Studio;
- настройка переменных и импорт классов;
- создание конструктора класса с нуля;

- преобразование графики из класса Java AWT в графику для Android Canvas;
- рисование объекта `ImageView` с помощью метода `onDraw()`;
- создание объекта интерфейса `Runnable` для реализации многопоточности на языке Java;
- использование обработчика `Handler` для связи с отдельными потоками при программировании на языке Java;
- использование многопоточности для улучшения эффективности работы с анимацией;
- обработка событий множественных касаний и использование методов `MotionEvent` для определения координат точек касания на экране;
- изменение значка и названия приложения.

Дополнительные задачи

Попробуйте выполнить следующие задания. Таким образом вы повторите пройденный материал, попрактикуетесь в программировании и получите новые навыки. Если у вас возникнут проблемы, вы можете загрузить примеры решений на сайте по адресу **https://eksmo.ru/files/learn_java_easy_way.zip**.

Задача № 1: Объединение событий касания одним пальцем и множественных касаний, версия 1.0

В этой главе мы научились обрабатывать события одиночного и множественных касаний. В этой задаче вам нужно отразить разницу между событиями одного и множественных касаний. Измените логику внутри метода `onTouch()`, чтобы при нажатии на экран одним пальцем рисовались пузырьки большего размера, и пузырьки меньшего размера в случае множественных касаний.

Помните, вы можете узнать количество событий касания с помощью метода `getPointerCount()` объекта `MotionEvent` внутри метода `onTouch()`.

Кроме того, можно написать программу, рисующую пузырьки тем меньше размером, чем больше пальцев одновременно касается экрана.

Задача № 2: Объединение событий касания одним пальцем и множественных касаний, версия 2.0

После решения задачи № 1 попробуйте сделать следующее. Измените класс `Bubble` и слушатель событий `onTouch()` так, чтобы пузырьки, нарисованные с помощью одного пальца, получали одни и те же значения переменных скорости `xspeed` и `yspeed`, а нарисованные множественными касаниями — разлетались в разных направлениях с разной скоростью.

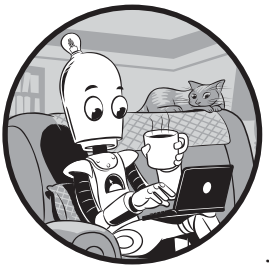
Для этого потребуется вести изменения в слушатель событий `onTouch()`, аналогичные тем, которые требуются в задаче № 1. Но, чтобы сгруппировать пузырьки, вам потребуется научиться рисовать одни пузырьки с фиксированной скоростью и другие — со случайной. Пузырьки, нарисованные одним касанием, должны двигаться с одинаковой скоростью, чтобы выглядеть единой группой, а пузырькам, нарисованным с помощью множественных касаний, скорость должна по-прежнему присваиваться случайным образом.

Для решения этой задачи можно создать второй конструктор `Bubble()`. Он будет похож на конструктор `Bubble(x, y, size)`, который мы создали для класса `Bubble` изначально, но у него будет другое количество параметров.

Например, можно создать второй конструктор `Bubble()`, который принимает пять параметров: `x`, `y`, `size`, `xspeed` и `yspeed`. Затем вы можете использовать этот конструктор, когда пользователь прикасается к экрану только одним пальцем, чтобы дать всем пузырькам одинаковые значения скорости. При этом оригинальный конструктор надо использовать, когда пользователь коснется экрана более, чем одним пальцем, чтобы пузырьки летели в случайных направлениях.

Приложение

ОТЛАДКА И ПРЕДОТВРАЩЕНИЕ ТИПИЧНЫХ ОШИБОК В ЯЗЫКЕ JAVA



Во время работы с этой книгой вы, скорее всего, сделали несколько опечаток или ошибок во время написания программ. В этом приложении я рассмотрю несколько распространенных ошибок программирования. Надеюсь, что это поможет вам избежать их в будущем. У сред разработки, таких как Eclipse и Android Studio, есть одна хорошая функция, которая особенно полезна при отладке — *подсветка синтаксиса*. Имена классов, функции, типы переменных, строки, комментарии и другие элементы по-разному окрашены в этих программах.

Подсветка синтаксиса может помочь вам быстро обнаружить опечатки и другие проблемы. Например, если вы забудете закрыть кавычки в конце строковой переменной, точка с запятой в конце этой строки будет окрашена не так, как другие точки с запятой на экране. Когда мы рассмотрим эти распространенные ошибки, попробуйте намеренно сделать их в одном из приложений и посмотрите на предупреждения, которые покажут программы Eclipse и Android Studio. Ничего страшного, если вы испортите приложение — вы всегда можете вернуться назад и все исправить, либо используя исходный код

в книге, либо нажав сочетание клавиш **Ctrl+Z** (⌘+**Z** в macOS) или выбрав команду меню **Edit** ⇒ **Undo** (Правка ⇒ Отменить).

Ошибки и регистр

Правильное написание важно в любом языке программирования, но в языке Java важную роль также играют прописные и строчные буквы. Например, если вы напишите слова `scanner` или `string` с маленькой буквы `s`, текстовый редактор в среде разработки Eclipse подчеркнет имя класса красным цветом, а в программе Android Studio текст будет окрашен красным. Это может показаться глупым, но компилятор языка Java понимает только ключевое слово `String`, но не `string`.

То же самое касается имен переменных, таких как `playAgain` или `theNumber`. Например, если мы случайно напишем букву `P` прописной и используем строчную букву `a`, то у нас получится `Playagain`. Компилятор языка Java не поймет, что мы имеем в виду переменную `playAgain`. Давайте рассмотрим, как каждая среда разработки помогает нам находить и исправлять опечатки, подобные этим. Начнем с программы Eclipse, а затем перейдем к среде разработки Android Studio.

Исправление опечаток в среде разработки Eclipse

Программа Eclipse предупреждает нас об ошибке, подчеркивая слово с ошибкой красным цветом, а Android окрасит красным весь текст. На рис. A.1 показано, как среда разработки Eclipse выделяет две ошибки, которые я сделал в игре «Больше-Меньше» из главы 2.

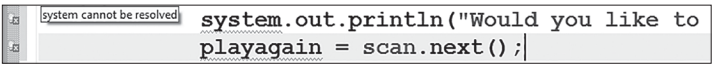


Рис. A.1. Eclipse помогает нам находить опечатки и ошибки в регистре букв

В этом примере я написал строчную букву `s` в слове `system`, и среда разработки Eclipse подчеркнула ошибку. Кроме того, у левой границы экрана добавлено предупреждение, которое можно увидеть, наведя на него указатель мыши. Компилятор языка Java выдает ошибку `system cannot be resolved` (Переменная `system` не может быть решена). Это означает, что компилятор не может понять, о какой переменной идет речь, поскольку он понимает только слово `System` с прописной буквой `S`. То же самое

программа Eclipse делает с переменной `playagain` на следующей строке — там должно быть написано `playAgain`.

Помните, что вы можете использовать возможность автозаполнения в среде Eclipse для исправления многих ошибок, подобных этим. Щелкните мышью по одному из слов с ошибками, как показано на рис. А.1, и вы увидите меню автозаполнения, похожее на показанное на рис. А.2.

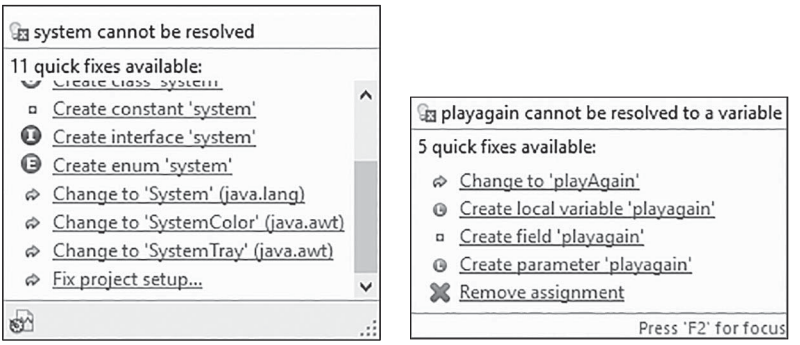


Рис. А.2. Функция автозаполнения в среде разработки Eclipse предоставляет информацию об ошибках и предлагает возможные варианты их исправления

Для первой ошибки программа Eclipse не только сообщает нам, что она не понимает слово `system`, но и предлагает несколько возможных исправлений, в том числе и четвертый снизу на рис. А.2, **Change to 'System' (java.lang)** (Изменить на 'System' (java.lang)). У функции автозаполнения не всегда есть правильное решение, но в данном случае одним из 11 предлагаемых исправлений является правильный регистр слова `System`. Для второй ошибки правильное решение стоит первым в списке предложений среды разработки: **Change to 'playAgain'** (Изменить на 'playAgain'). Если функция автозаполнения программы Eclipse предлагает правильное решение, щелкните соответствующую запись, и ваша ошибка орфографии или регистра будет исправлена.

Исправление опечаток в среде разработки Android Studio

Программа Android Studio помогает нам устранять ошибки, окрашивая возможный источник проблемы в красный цвет. В примере, показанном на рис. А.3, я ошибся в написании ключевого слова `public`, набрав его как `Public` с прописной буквой `P`.

Обратите внимание, что ошибка похожа на ту, что мы видели чуть раньше в среде разработки Eclipse: компилятор языка Java

говорит нам, что не может разрешить или понять символ 'Public'. Однако программа Android Studio не может всегда обеспечивать такой же уровень поддержки в исправлении ошибок, как на рис. А.4.

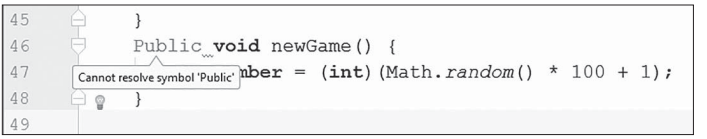


Рис. А.3. Среда разработки Android Studio предупреждает нас о ключевом слове, набранном в неправильном регистре

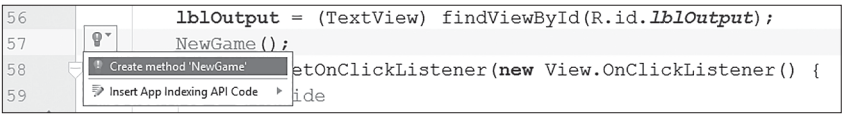


Рис. А.4. После нажатия сочетания клавиш **Alt+Enter** (⌥ + ⏎ в операционной системе macOS) программа Android Studio предлагает варианты решения, которые не всегда могут решить проблему

На рис. А.4 я напечатал `NewGame()` вместо `newGame()` с прописной буквой `N`. Программа Android Studio правильно окрасила код красным цветом, сообщая мне об ошибке. Однако давайте посмотрим, что произойдет, когда я щелкну мышью по коду и нажму сочетание клавиш **Alt+Enter**, чтобы попытаться получить совет по решению проблемы. Вместо того чтобы предложить правильное написание `newGame`, среда разработки Android Studio предлагает создать другой метод с именем `NewGame`. Это неправильное решение для ошибки в имени метода, поэтому вам придется найти место в программе, где метод `public void newGame()` был определен, уточнить корректное написание его имени и вручную исправить опечатку.

Обе среды разработки, как Android Studio, так и Eclipse, пытаются помочь вам в поиске ошибок и часто предлагают правильные пути их исправления, однако понимание того, что ошибки в написании и регистре являются весьма распространенными, может помочь вам избежать их и быстро исправить, если они появятся в вашем коде.

Предотвращение других распространенных ошибок в написании

Особенно трудно отлавливать ошибки, когда допущена опечатка в имени обработчика события или другого переопределенного метода. Эти опечатки могут варьироваться от ввода `OnCreate()`

или OnDraw() вместо правильных onCreate() и onDraw() до опечаток в public void actionPerformed (ActionEvent e) для кнопки в приложении на языке Java с графическим интерфейсом внутри метода ActionListener или потеря букв «ed» в обработчиках событий mousePressed(), mouseClicked() или mouseDragged().

Чтобы избежать подобных ошибок, существует конвенции программирования. Если вы назвали метод или переменную именем, начинающимся с прописной буквы, а класс — со строчной, это далеко не всегда вызовет ошибку. Однако такое именование является дурным тоном. Если вы с самого начала возьмете себе за правило следовать распространенным конвенциями программирования на языке Java, то сэкономите время и избежите разочарований в будущем.

Проблемы сравнения

Помните, что знак двойного равенства (==) является оператором сравнения «равно». Не путайте его с оператором присваивания (=). Например, чтобы присвоить значение 5 переменной с именем number, вы должны ввести:

```
int number = 5; // Присваивает значение 5 переменной number
```

Но для сравнения значения переменной используется знак двойного равенства:

```
if (number == 5) // Если значение переменной number "равно" 5
```

Также помните, что строки сравниваются с помощью метода equals(), а не оператора ==. Например, условие if (playAgain == "y") всегда будет оцениваться как false (ложь). Правильная запись инструкции if будет следующей:

```
if (playAgain.equals("y"))
```

Метод equals() класса String проверяет, совпадает ли *содержимое* строк, что мы обычно и хотим при сравнении строк. Метод equals() также используется при сравнении двух объектов, например if (bubble1.equals(bubble2)), позволяет понять, ссылаются ли, две переменные Bubble на один и тот же объект Bubble в приложении «Рисование пузырьков».

Символы группировки

При программировании на языке Java важно не запутаться в круглых, квадратных и фигурных скобках. Мы называем их *символами группировки*, потому что они группируют программные элементы вместе, и в программе не должно быть открывающего символа группировки без соответствующего ему закрывающего символа. Среды разработки Eclipse и Android Studio предлагают разные способы обнаружения и исправления ошибок незакрытых символов группировки.

Быстрые исправления в программе Eclipse

Среда разработки Eclipse может помочь нам с группировкой символов несколькими способами. Если вы забудете круглую скобку в условии или функции, программа Eclipse укажет на ошибку, подчеркнув красной линией ближайшее к отсутствующей скобке слово, как показано на рис. А.5.

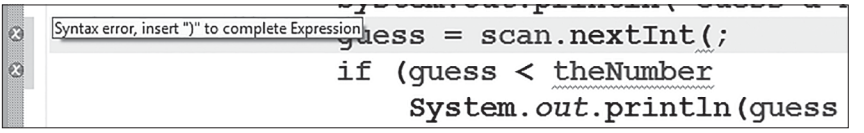


Рис. А.5. Среда разработки Eclipse помогает находить ошибки с круглыми скобками и иногда предлагает их решение

Кроме того, программа Eclipse пытается помочь вам выявить ошибки во время ввода. На рис. А.5 я забыл поставить закрывающиеся круглые скобки в конце двух строк, которые оказались помечены красным символом ошибки. Щелчок по красному символу ошибки в первой строке в левом поле редактора приводит к появлению предложения вставить закрывающую скобку.

Если вы поместите курсор рядом с закрывающей или открытой фигурной скобкой, среда разработки Eclipse выделит парную ей фигурную скобку. Посмотрите как это работает, щелкнув мышью рядом с фигурной скобкой в любой из ваших программ.

Кроме того, не забывайте, что редактор Eclipse может автоматически исправить отступы. Для этого надо выделить нужный фрагмент кода и нажать **Ctrl+I**. Для коротких программ это будет всего лишь приятным удобством, но для более длинных программ,

охватывающих несколько страниц, исправление отступов может быть очень полезно для предотвращения ошибок отсутствующих или неверно расставленных фигурных скобок.

Завершение кода в программе Android Studio

Среда разработки Android Studio предлагает еще большую помощь по исправлению ошибок, связанных с символами группировки. Функция завершения кода в Android Studio может автоматически добавить открывающиеся и закрывающиеся круглые, квадратные или фигурные скобки.

Например, откройте проект *GuessingGame* из главы 4 и внутри функции `onCreate()`, найдите строку, в которой запускается новая игра:

```
lblOutput = (TextView) findViewById(R.id.lblOutput);
newGame();
btnGuess.setOnClickListener(new View.OnClickListener() {
```

Удалите круглые скобки и точку с запятой в конце строки:

```
lblOutput = (TextView) findViewById(R.id.lblOutput);
newGame
btnGuess.setOnClickListener(new View.OnClickListener() {
```

Программа Android Studio окрасит текст красным цветом, чтобы вы знали об ошибке, а затем вы можете использовать завершение кода для добавления отсутствующих круглых скобок и точки с запятой. Чтобы автоматически исправить инструкцию, расположите курсор сразу после `newGame`, как показано на рис. А.6, и нажмите сочетание клавиш **Ctrl+Пробел**, чтобы выполнить базовое завершение кода.

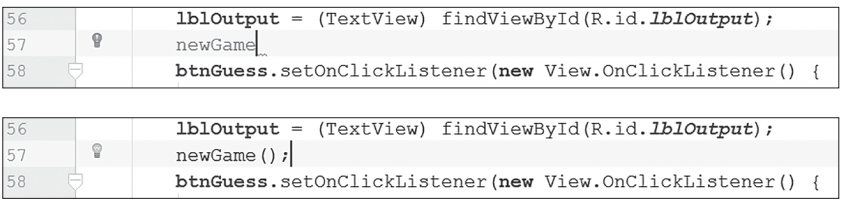


Рис. А.6. Поместите курсор рядом с отсутствующими скобками (вверху) и нажмите сочетание клавиш **Ctrl+Пробел**, чтобы дописать инструкцию с помощью функции автоматического завершения кода среды разработки Android Studio (внизу)

Обратите внимание, что функция завершения кода добавила как отсутствующие скобки, так и точку с запятой. Комбинация клавиш **Ctrl+Пробел** — это всего лишь одно из трех сочетаний клавиш для завершения кода в программе Android Studio. Второе сочетание, **Ctrl+Shift+Пробел**, называется *интеллектуальным завершением* и показывает соответствующие параметры во всплывающем окне. Дважды нажмите эту комбинацию клавиш, чтобы развернуть список вариантов завершения кода. Наконец, *завершение команды* (**Ctrl+Shift+Enter** в операционной системе Windows и Linux, а также ⌘+⇧+⏎ в операционной системе macOS) добавит закрывающие круглые, квадратные или фигурные скобки и конечную точку с запятой, если она нужна. Посмотрите, как работает завершение команды. Для этого удалите только закрывающую круглую скобку из метода, инструкции `if` или цикла `for`, и нажмите сочетание клавиш **Ctrl+Shift+Enter**. Функция завершения команды в среде разработки Android Studio закроет символы группировки и добавит точки с запятой или фигурные скобки там, где они нужны, тем самым упрощая и ускоряя работу и помогая избежать ошибок при программировании.

Итоги

Это далеко не все ошибки, с которыми вы столкнетесь во время программирования на языке Java. Однако, как вы могли заметить, обе среды, Eclipse и Android Studio, предлагают очень полезные инструменты для начинающих и опытных программистов. Чем больше вы будете программировать на языке Java, тем быстрее вы будете находить ошибки и исправлять их. Однако ваш код, скорее всего, никогда не будет полностью безошибочным. Я программирую уже более 30 лет, и все равно мне нужно отлаживать код после его написания. Чтобы не делать ошибок, а сделанные быстро находить и исправлять, нужно с самого начала учиться правильным приемам программирования и пользоваться поддержкой, встроенной в профессиональные инструменты, такие как среды разработки Eclipse и Android Studio.

ОБ АВТОРЕ

Брайсон Пэйн (Bryson Payne) — профессор информатики в Университете Северной Джорджии, где с 1998 года он преподает начинающим программистам. Брайсон — сертифицированный специалист по информационной безопасности (CISSP) и сертифицированный этичный хакер (CEH), обладающий и другими сертификатами в этой области. Он также был первым заведующим кафедры информатики в Университете Северной Джорджии и директором-учредителем Центра по обучению деятельности в киберпространстве, при Центре академического мастерства по киберзащите в Агентстве Национальной безопасности (NSA). Ему нравится работать со школьниками и способствовать обучению информатике во всем мире.

Брайсон печатается в журналах CIO, Campus Technology и Wall Street Journal, а также написал книгу-бестселлер «Python для детей и родителей. Играй и программируй» (Эксмо, 2017), изданную на пяти языках (корейский, португальский, русский, китайский и английский). Помимо этого в качестве инструктора он ведет нескольких онлайн-курсов, в том числе высокорейтинговый курс «Практический этичный хакинг» (2017) на обучающем портале Udemy. Его курсы прошли свыше 20 000 учащихся из более чем 150 стран. Брайсон Пэйн защитил диссертацию в области информатики в Государственном университете Джорджии и публикуется в научных и отраслевых журналах, а также регулярно выступает на региональных и национальных конференциях по информатике и кибербезопасности.

Брайсон Пэйн живет на севере Атланты, штат Джорджия, со своей прекрасной женой Бев, двумя сыновьями Алексом и Максом и тремя кошками Лео, Роки и Пикселем.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

КОГДА ВЫ ДАРИТЕ КНИГУ, ВЫ ДАРИТЕ ЦЕЛЫЙ МИР

ХОТИТЕ ЗНАТЬ БОЛЬШЕ?

Заходите на сайт:
https://eksmo.ru/b2b/

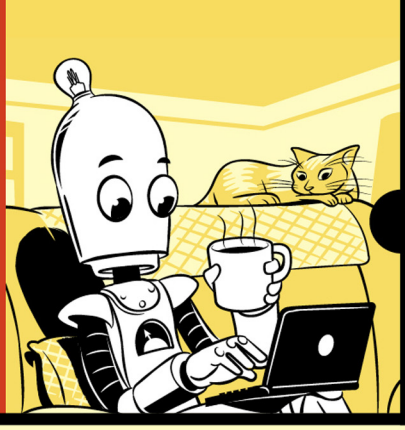
Звоните по телефону:
+7 495 411-68-59, доб. 2261

ВАШ ЛОГОТИП НА ОБЛОЖКЕ

ВАШ ЛОГОТИП НА КОРЕШКЕ

ОБРАЩЕНИЕ К КЛИЕНТАМ НА ОБЛОЖКЕ

JAVA
БЕЗ ГОЛОВНОЙ БОЛИ



ВКЛЮЧАЕТ ВЕРСИИ JAVA 8 и 9

Java - один из самых популярных языков программирования в мире, однако выучить его – непростая задача. «Легкий способ выучить Java» - это книга, которая сделает процесс обучения эффективным и понятным, ведь с первых страниц вы начнете создавать реальные функционирующие приложения!
Начните знакомство с Java с JShell, затем приступайте к разработке несложных игр. Вы научитесь разрабатывать игры как под ПК, так и под Android и использовать такие «золотые стандарты» отрасли, как среды разработки Eclipse и Android Studio.

Создавая эти приложения, вы научитесь:

- **Выполнять вычисления и управлять текстовыми строками**
- **Использовать условия, циклы и методы**
- **Создавать функции, позволяющие использовать код повторно**
- **Создавать элементы графического интерфейса: кнопки, меню, слайдеры**
- **Пользоваться Eclipse и Android Studio для отладки кода**

«Книга Брайсона Пейна – это больше, чем просто еще одно пособие по языку программирования Java. Читателю предлагается увлекательное путешествие в удивительный и полный драматизма мир Java. Здесь есть все – и базовые приемы создания программ, и нетривиальные подходы и концепции. А еще – неповторимый авторский стиль, который никого не оставит равнодушным. Изучать Java по такой книге не только легко, но и приятно».

Алексей Васильев,
доктор физико-математических наук,
автор книги «Программирование на Java для начинающих»

no starch press
www.nostarch.com

ISBN 978-5-04-093540-6
9 785040 935406 >

БОМБОРА

Бомбора — это новое название Эксмо Non-fiction, лидера на рынке полезных и вдохновляющих книг. Мы любим книги и создаем их, чтобы вы могли творить, открывать мир, пробовать новое, расти. Быть счастливыми. Быть на волне.

bomborabooks
www.bombora.ru