Роджерс Кейденхед

8-Е ИЗДАНИЕ

Java™

за 24 часа

ДИАЛЕКТИКА

Pearson

Java™

за 24 часа

8-Е ИЗДАНИЕ

Rogers Cadenhead

SAMS **Teach Yourself**

Java™

in 24 Hours

EIGHTH EDITION

Pearson

Роджерс Кейденхед

Java™

за 24 часа

8-Е ИЗДАНИЕ

Диалектика

Москва ♦ Санкт Петербург
2019

ББК 32.973.26-018.2.75
К33
УДК 681.3.07

Компьютерное издательство "Диалектика"

Перевод с английского *А.П. Сергеева*

Под редакцией *В.Р. Гинзбурга*

По общим вопросам обращайтесь в издательство "Диалектика" по адресу:
info@dialektika.com, http://www.dialektika.com

Кейденхед, Роджерс

К33 Java за 24 часа, 8-е издание. : Пер. с англ. — СПб. : ООО "Диалектика", 2019. — 480 с. : ил. — Парал. тит. англ.

ISBN 978-5-6041394-6-2 (рус.)

ББК 32.973.26-018.2.75

Научно-популярное издание

Роджерс Кейденхед

Java за 24 часа, 8-е издание

Подписано в печать 11.04.2019. Формат 70x100/16.
Гарнитура Times
Усл. печ. л. 38,7. Уч.-изд. л. 21,53
Доп. тираж 400 экз. Заказ № 3351

Отпечатано в АО "Первая Образцовая типография"
Филиал "Чеховский Печатный Двор"
142300, Московская область, г. Чехов, ул. Полиграфистов, д. 1
Сайт: www.chpd.ru, E-mail: sales@chpd.ru, тел. 8 (499) 270-73-59

ООО "Диалектика", 195027, Санкт-Петербург, Магнитогорская ул., д. 30, лит. А, пом. 848

ISBN 978-5-6041394-6-2 (рус.)
ISBN 978-0-672-33794-9 (англ.)

Оглавление

Содержание

Об авторе

Роджерс Кейденхед — писатель, программист и разработчик веб-приложений. Автор более 25 книг, посвященных Java и Интернету. Активный блогер и администратор нескольких популярных сайтов. С Роджерсом можно связаться в Твиттере: `@rcade`.

Введение

Будучи автором компьютерных книг, я частенько захожу в книжные магазины и тайком наблюдаю за поведением читателей. По моим оценкам, после того как читатель взял в руки книгу и открыл введение, в распоряжении автора остается ровно 13 секунд, чтобы убедить его купить эту книгу.

Так что буду краток: программировать на Java гораздо проще, чем может показаться на первый взгляд.

Я уже не говорю о том, что тысячи программистов воспользовались своими знаниями Java, чтобы получить высокооплачиваемую работу в IT-сфере. При достаточной настойчивости и наличии небольшого количества свободного времени любой человек может освоить основы этого самого популярного языка программирования на планете. И вы сможете быстро изучить Java, если будете каждый день выполнять упражнения из данной книги.

Java — один из лучших языков программирования для изучения, поскольку это полезная, мощная, современная технология, используемая множеством компаний по всему миру.

Данная книга предназначена как для новичков в программировании, так и для более опытных программистов, которые хотят освоить новые приемы работы с Java. В книге рассматриваются две наиболее современные версии: Java 9 и Java 10.

Благодаря своим широким возможностям Java завоевал просто невероятную популярность. Он позволяет создавать приложения с графическим интерфейсом, подключающиеся к веб-сайтам, работающие на смартфонах или планшетах Android и делающие много других вещей.

Этот язык программирования может применяться самым удивительным и неожиданным образом. К примеру, одна из известнейших и популярнейших игр, Minecraft, была целиком написана на Java. (В книге вы научитесь создавать собственные моды Minecraft!)

Данная книга научит вас программированию на Java с нуля. Здесь вы найдете не только пошаговые инструкции по созданию рабочих программ, но

и объяснения важных концепций простым и понятным языком. Пройдя 24 занятия, вы начнете писать свои собственные программы на Java. Вы также приобретете полезные навыки, такие как создание веб-приложений, разработка графического интерфейса и объектно-ориентированное программирование.

Многие термины, связанные с программированием, могут быть пока что вам непонятны. Но не переживайте заранее. Если вы способны загружать фотографии на Facebook или создавать электронные таблицы в Excel, вам вполне по силам писать компьютерные программы. Для этого вам всего лишь нужно прочесть данную книгу.

Сайт книги

Если в процессе чтения книги у вас возникнут какие-либо вопросы, посетите сайт книги, доступный по следующему адресу:

`http://workbench.cadenhead.org/book/java-9-24-hours/`

На этом сайте вы найдете:

- исходный код файлов классов и ресурсы, требуемые для создания всех примеров книги;
- ответы на упражнения к занятиям 1–8, включая исходный код.

Все исходные коды примеров можно также скачать на сайте издательства "Диалектика":

`http://www.williamspublishing.com/Books/978-5-6041394-6-2.html`

Соглашения, принятые в книге

Ниже приведены соглашения, принятые в книге.

ПРИМЕЧАНИЕ

Содержит дополнительную информацию по рассмотренной теме.

ПРЕДУПРЕЖДЕНИЕ

Привлекает внимание к проблемам и/или побочным эффектам, которые могут возникнуть в тех или иных ситуациях.

СОВЕТ

Содержит рекомендацию и уточнение, что позволит повысить эффективность программирования на языке Java.

В книге используются следующие шрифтовые выделения.

- Для того чтобы отделить код на Java от обычного текста, фрагменты листингов программ представлены моноширинным шрифтом.
- Заполнители, замещающие реальные значения аргументов, которые необходимо будет набрать в коде, выделяются *моноширинным курсивом*.
- Новые или важные термины выделяются *курсивом*.
- Все строки в листингах пронумерованы для удобства описания. В реальной программе строки нумеровать не нужно.

Ждем ваших отзывов!

Вы, читатель этой книги, и есть главный ее критик. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересны любые ваши замечания в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам бумажное или электронное письмо либо просто посетить наш сайт и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится ли вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Отправляя письмо или сообщение, не забудьте указать название книги и ее авторов, а также свой обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию новых книг.

Наши электронные адреса:

E-mail: info@dialektika.com

WWW: www.dialektika.com

Часть I
Первое знакомство

В этой части...

- Занятие 1 Готовимся программировать на Java
- Занятие 2 Создаем первую программу
- Занятие 3 Путешествие в мир Java
- Занятие 4 Принципы работы программ на Java

ЗАНЯТИЕ 1
Готовимся программировать на Java

На этом занятии вы узнаете...

- зачем изучать Java;
- как работает программа на Java;
- как выбрать инструмент разработки Java;
- как подготовиться к написанию первой программы.

Наверное, вам не раз доводилось слышать о том, что компьютерное программирование является безумно сложным. Вас убеждают в том, что для этого нужна ученая степень по информатике, тысячи долларов, вкладываемые в компьютерное оборудование и программное обеспечение, острый аналитический ум, бесконечное терпение и безграничная любовь к кофе и прочим энергетикам.

В действительности это не так, помимо утверждения о кофе и прочих энергетиках. Программирование гораздо проще, чем кажется многим. Некоторые программисты специально распускают подобные слухи, чтобы уменьшить конкуренцию за высокооплачиваемые рабочие места.

В наши дни сформировались идеальные условия для обучения программированию. В Интернете доступно бесчисленное множество инструментальных средств разработки, а тысячи программистов распространяют свои творения на условиях программного обеспечения с открытым кодом. Этот код могут просматривать другие люди, устранять ошибки и вносить свои улучшения. Во многих странах труд программиста становится весьма востребованным.

Поскольку Java используется повсеместно, целесообразно приступить к изучению именно этого языка. Операционная система Android установлена на миллиардах мобильных устройств, а приложения, выполняемые в ней, написаны на Java. Поэтому, если вы являетесь счастливым обладателем смартфона с Android, вы будете наслаждаться плодами труда программистов на

Java всякий раз, когда смотрите фильм, слушаете радио либо играете в Angry Birds.

Эта книга поможет научиться программировать на Java следующим категориям читателей.

1. боязливым новичкам, которые никогда не программировали прежде;
2. начинающим программистам, которые уже пытались программировать, но невзлюбили это занятие;
3. нетерпеливым интеллектуалам, которые знают другой язык программирования и хотят быстро научиться программировать на Java.

Материал книги излагается на понятном языке, без использования невразумительного технического жаргона. Все новые термины детально объясняются по мере их появления.

Если я преуспел в писательском искусстве, то после прочтения книги вы овладеете набором навыков, достаточных для занятия программированием на весьма высоком уровне. Вы сможете писать программы, быстро осваивать программистские курсы и книги, а также изучать новые языки программирования. К тому же вы получите навыки программирования на Java, который является самым распространенным языком программирования.

Первое занятие представляет собой введение в программирование и краткое руководство по настройке среды разработки программ. После выполнения предварительной настройки вы сможете использовать свой компьютер для написания и выполнения программ на Java.

Выбор языка программирования

Если вы достаточно хорошо знакомы с компьютером, чтобы подготовить красиво выглядящее резюме, свести бухгалтерский баланс либо поделиться отпускными фотографиями в Instagram, то вполне сможете создавать компьютерные программы.

Ключевым фактором в обучении программированию является выбор подходящего языка программирования. Зачастую результат этого выбора зависит от стоящих перед вами задач. Каждому языку программирования присущи свои преимущества и недостатки. Например, в дни моей юности был популярен язык программирования BASIC, специально предназначенный для начинающих программистов.

ПРИМЕЧАНИЕ

Язык программирования BASIC предназначался в помощь новичкам в программировании (буква 'B' в слове BASIC означает "Beginner" — "начинающий"). Но при изучении этого языка можно легко приобрести привычки неряшливого программирования.

Наиболее популярным языком программирования, созданным на основе BASIC, является Visual Basic (сокр. VB). Он был разработан компанией Microsoft и уже давно перерос возможности своего предка. Этот язык используется для разработки программ, предназначенных для выполнения на компьютерах и мобильных устройствах под управлением Windows. Не менее популярен и PHP — язык написания сценариев, предназначенный для создания сайтов. Другие широко используемые языки программирования, о которых вы, возможно, слышали, — C++, Ruby, JavaScript и Python.

Каждый из перечисленных языков программирования имеет своих сторонников, но именно Java обычно преподается на компьютерных курсах и в вузах.

Язык программирования Java, предложенный компанией Oracle, сложнее в изучении, чем такие языки, как VB и PHP, зато он перспективнее в силу ряда причин. Одно из преимуществ, связанных с изучением Java, — возможность его применения в разных операционных системах и вычислительных средах. С помощью Java можно разрабатывать программы для настольных компьютеров, веб-приложения, веб-серверы, приложения для Android и прочий код, выполняемый под управлением Windows, Mac, Linux и других операционных систем. Подобная универсальность отражена в следующем слогане, появившемся еще на ранних этапах развития Java: "Написано один раз, выполняется везде".

ПРИМЕЧАНИЕ

Первые программисты на Java использовали не столь лестный слоган: "Написано один раз, отлаживается везде". Этот язык программирования прошел длинный путь развития, поскольку его первая версия появилась еще в 1996 году.

Еще одно важное преимущество Java заключается в том, что для создания работающих программ требуется использование высокоорганизованного подхода. Следует проявлять особую осторожность при написании программ и хорошо обдумывать, как они хранят и изменяют данные.

Если вы только начинаете писать программы на Java, то вряд ли сможете оценить по достоинству этот язык программирования. Вас может утомить необходимость исправлять ошибки в процессе написания программы до ее запуска на выполнение. Но благодаря подобным усилиям гарантируется создание более надежных, полезных и свободных от ошибок программ.

На последующих занятиях вы узнаете обо всех правилах Java и подводных камнях, которых следует избегать.

Язык программирования Java появился благодаря усилиям канадского компьютерного инженера Джеймса Гослинга. Этот язык позиционировался своим создателем в качестве лучшего средства написания компьютерных

программ. В 1991 году во время работы в компании Sun Microsystems Гослинг был весьма недоволен тем, как используется язык программирования C++ в текущем проекте. Поэтому он создал новый язык, который позволил выполнять работу лучше. Вполне естественно, что появление Java вызвало многочисленные дискуссии относительно его преимуществ по сравнению с другими языками программирования. Но успешный опыт использования этого языка наглядно продемонстрировал его преимущества. В настоящее время Java выполняется на 15 млрд устройств по всему миру. Этому языку посвящено более 1000 книг, среди которых есть и написанные мною.

Даже если Java не является лучшим языком программирования, вне всякого сомнения он достоин изучения. На занятии 2 вы получите первую возможность попрактиковаться в программировании на Java.

Знание одного языка программирования существенно облегчает изучение других языков. Поскольку многие языки программирования похожи друг на друга, вам не придется начинать все сначала при переходе к изучению другого языка. Например, многие программисты на языках C++ и Smalltalk довольно легко освоили Java, поскольку он был создан на их основе. Аналогичным образом C# во многом подобен Java, поэтому легко осваивается программистами на Java.

ПРИМЕЧАНИЕ

Язык C++ уже несколько раз упоминался нами, поэтому у вас может возникнуть вопросы, касающиеся произношения данного названия. Если вы не в курсе, то C++ произносится как "Си-плас-плас" (на английском), а по-русски чаще произносят "Си-плюс-плюс". Этот язык разработан датским компьютерным инженером Бьярне Страуструпом в компании Bell Laboratories. Язык C++ является расширением языка C, отсюда "++" в его названии. Почему бы этот язык не назвать просто C+? На самом деле "++" — всего лишь шутка в мире компьютерного программирования, суть которой вы поймете позднее.

Подскажите компьютеру, что ему делать

Компьютерная программа, или *программное обеспечение*, представляет собой набор указаний компьютеру о необходимости выполнения той или иной задачи. Все действия, выполняемые компьютером, от загрузки до отключения, фактически проделывает программа. Операционная система Mac OS X — это программа; Minecraft — это программа; драйвер, управляющий принтером, — тоже программа; даже "голубой экран смерти", появляющийся в случае серьезного сбоя компьютера с установленной операционной системой Windows, — это тоже программа.

Компьютерные программы состоят из списков команд. После запуска программы эти команды выполняются компьютером в определенном порядке.

Рассмотрим следующий пример. Предположим, вы наняли управляющего и даете ему следующий подробный набор команд.

> Уважаемый мистер Дживс,
>
> Пока я буду заниматься важными государственными делами, пожалуйста, выполните следующее.
>
> Пункт 1. Пропылесосьте гостиную.
>
> Пункт 2. Сходите в магазин.
>
> Пункт 3. Купите соевый соус, васаби и как можно больше калифорнийских роллов.
>
> Пункт 4. Возвращайтесь домой.
>
> С уважением, ваш господин и хозяин,
> Берти Вустер

Если вы имеете дело с управляющим-человеком, то порядок выполнения выданных команд не является жестко определенным. Если не будет калифорнийских роллов, Дживс купит и принесет домой филадельфийские.

Компьютерам не присуща подобная свобода действий: они следуют командам буквально, и написанные вами программы будут выполнены буквально.

Ниже приводится пример компьютерной программы, состоящей из трех строк и написанной на языке BASIC. Посмотрите на нее, но пока что не думайте о значении этих строк.

```
1 PRINT "Привет, Том! Это Боб из соседнего офиса."
2 PRINT "Рад видеть тебя, приятель! Как дела?"
3 INPUT A$
```

После перевода на русский язык эта программа будет эквивалентна следующему списку команд.

> Дорогой персональный компьютер,
>
> Пункт 1. Отобрази сообщение: "Привет, Том! Это Боб из соседнего офиса."
>
> Пункт 2. Задай вопрос: "Рад видеть тебя, приятель! Как дела?"
>
> Пункт 3. Дайте пользователю возможность ответить на вопрос.
>
> С уважением, твой господин и хозяин,
> Петр Кодер

Каждая строка в компьютерной программе называется *инструкцией*. Компьютер обрабатывает каждую инструкцию в определенном порядке, подобно тому, как повар готовит блюдо по рецепту или как управляющий, мистер

Дживс, следует указаниям Берти Вустера. В BASIC номера строк применяются для расположения инструкций в нужном порядке. В других языках, таких как Java, номера строк не используются. Вместо этого применяются разные способы, позволяющие сообщить компьютеру о порядке выполнения программы.

В силу специфики выполнения программ бессмысленно обвинять компьютер в случае появления каких-либо проблем. Компьютер выполняет в точности то, что вы ему говорите, поэтому вина за допущенные ошибки обычно ложится на программиста.

Хорошая новость заключается в том, что вы не сможете нанести существенного вреда компьютеру из-за неправильно написанной программы. Поэтому можете смело учиться программировать на Java — что бы вы ни делали, компьютер не пострадает.

Как работают программы

Набор инструкций, образующих компьютерную программу, называется *исходным кодом*.

Большинство компьютерных программ пишется так же, как и обычные электронные письма, — путем ввода каждой инструкции в текстовом окне. Некоторые инструменты программирования включают собственный редактор исходного кода, другие могут использоваться совместно с любым текстовым редактором.

После создания компьютерной программы запишите файл на жесткий диск компьютера. Компьютерные программы часто используют собственные расширения имен файлов, определяющие тип файла. Программы, написанные на Java, имеют расширение `.java`, например `Calculator.java`.

ПРИМЕЧАНИЕ

Компьютерные программы должны быть подготовлены в виде текстовых файлов без специального форматирования. Для сохранения файлов в виде неформатированного текста можно воспользоваться текстовым редактором Блокнот из комплекта поставки Windows. Для создания текстовых неформатированных файлов можно также воспользоваться редактором TextEdit, установленным на компьютерах Macintosh, или текстовым редактором *vi* или *emacs*, работающим под управлением Linux. Более простое решение будет предложено позже на этом занятии.

Чтобы выполнить программу, сохраненную в виде файла, вам потребуется помощь, конкретный вид которой зависит от используемого языка программирования. В некоторых языках для выполнения программ нужен *интерпретатор*. В этом случае проверяется и выполняется каждая строка кода, после

чего происходит переход к следующей строке. Многие версии BASIC относятся к категории интерпретируемых языков.

Наибольшее преимущество интерпретируемых языков — большая скорость тестирования. В процессе написания программы на языке BASIC вы сможете тут же попробовать ее выполнить, устранить выявленные ошибки, а потом снова запустить на выполнение. Основной недостаток интерпретируемых языков программирования заключается в меньшей скорости выполнения по сравнению с другими языками. Каждая строка кода должна быть преобразована в инструкции, которые последовательно выполняются компьютером.

Другие языки программирования нуждаются в *компиляторе*. Исходная программа преобразуется в форму, воспринимаемую компьютером. Компилятор также оптимизирует программу. Скомпилированная программа может выполняться непосредственно, не нуждаясь в интерпретаторе.

Скомпилированные программы работают быстрее интерпретируемых, но требуют больше времени для тестирования. Прежде чем запустить такую программу на выполнение, нужно написать ее полностью, а затем скомпилировать. Если при этом будут обнаружены и исправлены ошибки, потребуется повторно скомпилировать программу.

Язык Java необычен в том, что требует как компилятор, так и интерпретатор. Компилятор преобразует инструкции программы в *байт-код*. После успешного создания байт-кода он запускается с помощью интерпретатора, который называется *виртуальной машиной Java*.

Виртуальная машина Java, также называемая JVM, позволяет запускать одну и ту же программу на Java в разных операционных системах и на разных типах вычислительных устройств. Виртуальная машина преобразует байт-код в инструкции, которые может выполнять операционная система конкретного устройства.

ПРИМЕЧАНИЕ

В Java 9 появилось новое инструментальное средство, называемое JShell. Это интерпретатор команд, выполняющий инструкции Java сразу же после их ввода. Оболочка JShell помещает инструкцию в программу на Java, компилирует ее в байт-код, а затем выполняет его. Этот инструмент весьма полезен как для обучения, так и для тестирования.

Когда программы не работают

Многие начинающие программисты буквально впадают в ступор, когда сталкиваются с обилием ошибок, которые появляются практически всюду. Одни из них являются синтаксическими, допущенными при вводе

инструкций программы, другие относятся к категории логических. Эти ошибки можно выявить лишь на этапе тестирования программы, хотя они могут остаться незамеченными. Зачастую логические ошибки приводят к выполнению компьютером непредвиденных действий.

Как только вы начнете писать свои собственные программы, вы тут же столкнетесь с ошибками — естественной частью процесса программирования. Программные ошибки часто называют *багами*. Этот термин уже более ста лет используется для описания ошибок в технических устройствах.

Процесс поиска и устранения ошибок называется *отладкой*.

Существует множество способов описания ошибок. Хотите вы того или нет, но вы также получите опыт отладки при изучении программирования.

ПРИМЕЧАНИЕ

Один из первых компьютерных багов был обнаружен в далеком 1947 году усилиями команды, в которой работала американский специалист в области компьютерных наук Грейс Хоппер. Она искала причину неисправности реле, установленного в компьютере Гарвардского университета. Выявленная причина поломки заключалась не в программном сбое. Это был самый настоящий "баг" (от англ. *bug* — насекомое). В процессе обследования неисправного компьютера Грейс Хоппер нашла и удалила дохлую моль, которая вывела из строя реле. После этого она поместила ее в журнал, сопроводив следующей записью: "Первый случай фактического обнаружения бага". Этот "баг" и страницу журнала можно увидеть на сайте, доступном по следующему адресу:

`http://www.doncio.navy.mil/CHIPS/ArticleDetails.aspx?id=3489`

Выбор инструмента программирования на Java

Чтобы создавать программы на Java, нужно располагать соответствующим инструментом программирования на Java. Доступно несколько подобных средств, включая простую среду разработки Java Development Kit и более сложные среды разработки Eclipse, IntelliJ IDEA и NetBeans. Последние три инструмента относятся к категории интегрированных сред разработки (Integrated Development Environment — IDE), применяемых профессиональными программистами.

Всякий раз, когда компания Oracle выпускает новую версию Java, первоочередная поддержка этой версии появляется в Java Development Kit (JDK).

Для создания программ из этой книги вам понадобится JDK версии 9 или инструментальное средство, которое работает поверх нее. JDK — это набор свободно распространяемых утилит командной строки, предназначенных для разработки программ на Java. Эти инструменты лишены графического интерфейса пользователя, поэтому, если вы никогда не работали в текстовой

среде, такой как командная строка Windows или интерфейс командной строки Linux, вам будет затруднительно пользоваться JDK.

Интегрированная среда разработки NetBeans, также бесплатно предлагаемая компанией Oracle, позволяет написать и протестировать код Java с гораздо меньшими усилиями, чем JDK. Графическая среда NetBeans включает редактор исходного кода, дизайнер интерфейса пользователя и менеджер проектов. Эта среда работает совместно с JDK, поэтому при разработке программ на Java вам придется установить оба этих инструмента.

Большинство программ, описанных в книге, было разработано в среде NetBeans, которую можно загрузить и установить независимо от JDK. Можно использовать и другие инструменты Java, если они поддерживают JDK 9 или JDK 10.

ПРИМЕЧАНИЕ

Вовсе не обязательно использовать NetBeans для выполнения примеров, описанных в книге. Если вы можете использовать JDK или другой инструмент для создания, компиляции или выполнения программы, воспользуйтесь им. Описание среды разработки NetBeans включено здесь по той причине, что для читателей предыдущего издания книги эта среда оказалась проще в применении, чем JDK. К тому же я использую NetBeans для создания большинства программ на языке Java.

На занятии 24 вы будете использовать интегрированную среду разработки Android Studio. Компания Google, которая разработала Android, рекомендует этот свободно распространяемый инструмент разработчикам программ для Android.

Установка среды разработки Java

На каждом занятии будут рассмотрены проекты, выполнение которых позволит вам лучше усвоить материал. Имейте в виду, что вы не сможете выполнять какие-либо операции, если на вашем компьютере не установлен инструмент программирования на Java. Если вы уже установили инструмент, позволяющий разрабатывать программы на Java, воспользуйтесь им для разработки примеров программ на следующих 23 занятиях. Но вы должны знать, как использовать этот инструмент. Одновременное изучение Java и возможностей интегрированной среды разработки может оказаться непростой задачей.

Для выполнения примеров из книги рекомендуется использовать пакет NetBeans, который можно бесплатно загрузить на сайте Oracle (`http://netbeans.org`). Несмотря на то что NetBeans обладает набором расширенных средств, изучение которых потребует времени, его использование облегчает создание и выполнение простых приложений Java.

Инструкции по загрузке и установке NetBeans приведены в приложении А.

Резюме

На этом занятии вы ознакомились с концепцией компьютерного программирования, суть которой заключается в передаче компьютеру набора команд по выполнению определенных действий, а также узнали о том, почему следует выбирать именно язык программирования Java.

Возможно, вы уже загрузили и установили среду разработки программ на Java, которой воспользуетесь для создания программ на следующих занятиях.

Задайте десяти программистам вопрос о том, какой язык программирования они считают наилучшим, и вы получите десять ответов, которые включают фразу "Мой язык программирования лучше вашего" либо шуточки вроде "Ваш исходный код непомерно раздут". Как бы там ни было, но Java котируется очень высоко, поскольку он позволяет писать адаптируемые программы, к тому же это невероятно гибкий и грамотно спроектированный язык. Зная Java, можно достичь очень многого, а также быстрее выучить другие языки программирования.

Если вы до сих пор не можете разобраться в программах, языках программирования или в Java в целом, не паникуйте. Все эти понятия будут разъяснены на следующем занятии, когда вы ознакомитесь с этапами создания программы на Java.

Вопросы и ответы

В. BASIC? C++? Smalltalk? Java? Что означают названия этих языков программирования?

О. BASIC — это сокращение от Beginner's All-purpose Symbolic Instruction Code (универсальный код символических инструкций для начинающих).

Язык программирования C++ является расширением языка C, который, в свою очередь, является результатом совершенствования языка B. Smalltalk — это инновационный объектно-ориентированный язык программирования, разработанный в 1970-е годы, идеи которого были заимствованы при создании Java.

Название "Java" идет вразрез с традицией именовать языки программирования с применением аббревиатур либо других значащих терминов. Просто это название понравилось разработчикам языка.

В. Почему интерпретируемые языки программирования медленнее компилируемых?

О. Эти языки медленнее в силу тех же причин, что и человек, который выполняет синхронный перевод. Синхронист делает свою работу медленнее, чем переводчик печатного текста. В процессе синхронного перевода приходится думать о точном переводе каждого утверждения, а при переводе печатного текста можно представлять себе текст в целом и применять технические приемы, ускоряющие перевод. Компилируемые языки могут быть намного быстрее, поскольку используют технические приемы, позволяющие кардинально улучшить эффективность выполнения программы.

Коллоквиум

Проверьте знание материала, рассмотренного на этом занятии, ответив на следующие вопросы.

Контрольные вопросы

1. Что из нижеперечисленного НЕ является причиной негативного отношения к программированию?
   - **А.** Программисты специально распространяют слухи о трудностях программирования, чтобы уменьшить конкуренцию на рынке труда.
   - **Б.** Повсеместно используемые жаргон и аббревиатуры.
   - **В.** Люди, которые находят программирование слишком сложным, могут рассчитывать на государственную помощь.
2. Какое инструментальное средство выполняет компьютерную программу построчно?
   - **А.** Медленный инструмент.
   - **Б.** Интерпретатор.
   - **В.** Компилятор.
3. Почему Джеймс Гослинг надолго закрылся в своем офисе и создал Java?
   - **А.** Ему не нравился язык программирования, используемый в текущем проекте.
   - **Б.** Рок-группа под его руководством не пользовалась успехом.
   - **В.** Если вы не можете просматривать ролики YouTube на работе, Интернет довольно скучен.

Ответы

1. **В.** Авторы компьютерных книг вообще не получают государственную помощь.

2. **Б.** Интерпретаторы выполняют программу построчно. Компиляторы заранее создают выполняемый код, поэтому программа выполняется быстрее.
3. **А.** Он был разочарован возможностями языка C++. В 1991 году, когда создавался Java, сервиса YouTube еще не существовало.

Упражнения

Если хотите лучше ознакомиться с Java и программированием, выполните следующие упражнения.

- Чтобы больше узнать о мотивах изучения Java, прочтите следующее сообщение в блоге Oracle University:

  ```
  https://blogs.oracle.com/oracleuniversity/
  10-reasons-why-you-should-consider-learning-java
  ```

- Используя предложения на английском или русском языке, напишите набор инструкций для преобразования температуры из шкалы Цельсия в шкалу Фаренгейта. По возможности разбейте инструкции на короткие строки, содержащие одно предложение.

Ответы на эти упражнения можно найти на сайте книги, адрес которого указан во введении.

ЗАНЯТИЕ 2
Создаем первую программу

На этом занятии вы узнаете...

- как вводить код программы на Java с помощью текстового редактора;
- как структурировать программу с помощью скобок;
- как хранить информацию в переменной;
- как отображать информацию, сохраненную в переменной;
- как сохранить, скомпилировать и выполнить программу.

Как вы узнали на занятии 1, компьютерная программа представляет собой набор инструкций, которые указывают компьютеру, что ему нужно делать. Эти инструкции даются компьютеру на языке программирования.

На этом занятии вы создадите свою первую программу на Java путем ввода соответствующего кода с помощью текстового редактора. По завершении ввода программы вы сохраните ее, скомпилируете и протестируете. Если в процессе тестирования обнаружатся ошибки, нужно исправить их и повторно запустить программу на выполнение.

Что нужно для создания программ

Как отмечалось на занятии 1, для создания программ на Java понадобится инструмент программирования, который поддерживает Java Development Kit (JDK). В качестве такого инструмента можно использовать интегрированную среду разработки NetBeans. Вам понадобятся инструмент, который может компилировать и выполнять программы на Java, и текстовый редактор, применяемый для написания этих программ.

При использовании большинства языков программирования компьютерные программы создаются путем ввода текста в текстовом редакторе (также называется *редактором исходного кода*). Некоторые языки программирования располагают собственным редактором. Интегрированная среда разработки

NetBeans также имеет свой собственный редактор, предназначенный для написания программ на Java.

Программы на Java — это простые текстовые файлы без какого-либо специального форматирования, такого как центрированный или полужирный текст. Редактор исходного кода NetBeans подобен простому текстовому редактору, только содержащему ряд полезных усовершенствований, предназначенных для программистов. Одно из таких усовершенствований заключается в том, что вводимый текст окрашивается в разные цвета, чтобы идентифицировать различные элементы языка программирования. В NetBeans также создаются отступы строк в нужных местах, а справиться с затруднениями поможет справка, встроенная в текстовый редактор.

Поскольку программы на Java являются текстовыми файлами, вы сможете открывать и редактировать их с помощью любого текстового редактора. Например, можно создать программу на Java в среде NetBeans, открыть ее в редакторе Блокнот Windows, внести изменения, а потом снова открыть в среде NetBeans.

Программа `Saluton`

Первая программа на Java, которая будет создана на этом занятии, просто отображает приветствие `"Saluton mondo!"`.

Прежде чем приступить непосредственно к разработке программы, нужно создать новый проект `Java24`. Для этого выполните следующие действия.

1. Выберите пункт меню **Файл⇨Создать проект**. На экране появится диалоговое окно **Создать проект**.
2. На панели **Категории** выберите пункт **Java**, а на панели **Проекты** — **Приложение Java**. После этого щелкните на кнопке **Далее**.
3. В качестве имени проекта введите **`Java24`**. (Если ранее уже создавался проект с этим именем, появится сообщение об ошибке "Папка проекта уже существует и содержит данные".)
4. Снимите флажок **Создать главный проект**.
5. Щелкните на кнопке **Готово**.

Только что созданный проект `Java24` находится в своей собственной папке, и вы сможете использовать его для создания и отладки программ на Java по мере чтения книги.

Приступаем к созданию программы

Среда NetBeans группирует родственные программы в одном проекте. Если вы еще не открыли проект `Java24`, сделайте это сейчас, выполнив следующие действия.

1. Выберите пункт меню **Файл⇨Открыть проект**. На экране появится соответствующее диалоговое окно.
2. Найдите папку `NetBeansProjects` и перейдите в нее (если до сих пор еще этого не сделали).
3. Выберите проект `Java24` и щелкните на кнопке **Открытие проекта**.

Проект `Java24` появится на панели **Проекты** справа от значка с изображением чашки кофе. Значок **+** предназначен для раскрытия списка, в котором находятся файлы и папки проекта.

Чтобы добавить новую программу на Java в текущий открытый проект, выберите пункт меню **Файл⇨Создать файл**. На экране появится окно **Создать файл**, показанное на рис. 2.1.

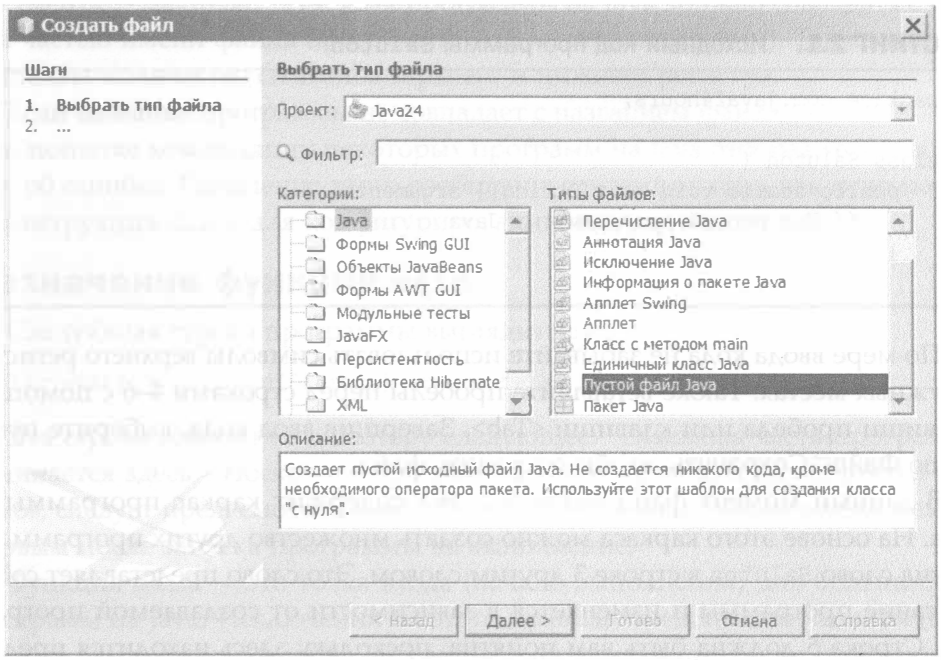


РИС. 2.1. Окно мастера Создать файл

На панели **Категории** перечислены разные типы программ на Java, доступные для создания. Щелкните на папке `Java`, находящейся на этой панели, чтобы увидеть, какие типы файлов относятся к этой категории. Для первого проекта выберите тип файла `Пустой файл Java` (в нижней части панели **Типы файлов**) и щелкните на кнопке **Далее**.

На экране появится диалоговое окно **Пустой файл Java**. Чтобы приступить к созданию новой программы, выполните следующие действия.

1. В поле **Имя класса** введите **`Saluton`**.
2. В поле **Пакет** введите **`com.java24hours`**.
3. Щелкните на кнопке **Готово**.

Теперь можно приступить к созданию программы. При этом используется редактор исходного кода, в котором открывается пустой файл `Saluton.java`. Начните с ввода строк кода из листинга 2.1, которые называются исходным кодом программы.

ПРЕДУПРЕЖДЕНИЕ

Не вводите номера строк и двоеточия в начале каждой строки — они используются в книге для ссылки на соответствующие строки.

ЛИСТИНГ 2.1. Исходный код программы `Saluton`

```
1: package com.java24hours;
2:
3: class Saluton {
4:    public static void main(String[] arguments) {
5:       // Код первой программы на Java
6:    }
7: }
```

По мере ввода кода не забывайте использовать символы верхнего регистра в нужных местах. Также вставляйте пробелы перед строками 4–6 с помощью клавиши пробела или клавиши <Tab>. Завершив ввод кода, выберите пункт меню **Файл⇨Сохранить**, чтобы сохранить файл.

В данный момент файл `Saluton.java` содержит каркас программы на Java. На основе этого каркаса можно создать множество других программ, заменив слово `Saluton` в строке 3 другим словом. Это слово представляет собой название программы и изменяется в зависимости от создаваемой программы. Строка 5 должна быть вам понятна, поскольку здесь находится предложение, написанное на русском языке. Другие строки требуют дополнительных пояснений.

Инструкция `class`

Первая строка программы имеет следующий вид:

```
package com.java24hours;
```

Пакет (`package`) представляет собой способ группирования программ на Java. Эта строка указывает компьютеру на то, что `com.java24hours` — это название пакета, используемого в программе.

После пустой строки следует третья строка, которая имеет следующий вид:

```
class Saluton {
```

В переводе на русский язык эта строка означает: "Компьютер, присвой моей программе название Saluton".

Как мы уже поясняли на занятии 1, каждая команда, которую вы даете компьютеру, записывается в виде отдельной строки кода и называется *инструкцией*. Инструкция `class` обеспечивает способ именования программы и может также использоваться для выполнения других операций в программе, как будет показано далее. Отметим, что программы на Java также называются *классами*.

В данном примере имя программы `Saluton` совпадает с названием файла документа — `Saluton.java`. Имя программы на Java должно совпадать с первой частью имени файла программы — до символа точки (`.`). Также должно совпадать количество символов верхнего и нижнего регистра.

Если название программы не совпадает с названием файла программы, то при попытке компиляции некоторых программ на Java отобразится сообщение об ошибке. Появление этого сообщения зависит от того, как используется инструкция `class` для конфигурирования программы.

Назначение функции `main`

Следующая строка программы выглядит так:

```
public static void main(String[] arguments) {
```

Эта строка говорит компьютеру следующее: "Основная часть программы начинается здесь". Поскольку программы на Java разбиты на несколько разделов, следует предусмотреть способ идентификации раздела, выполняемого первым после запуска программы на выполнение.

Функция `main` — это точка входа (начало выполнения) для большинства программ на Java. Исключения из этого правила представляют аплеты (программы, выполняемые на веб-странице в браузере), сервлеты (программы, выполняемые веб-сервером) и мобильные приложения, выполняемые на мобильных устройствах.

Большинство программ, которые вам предстоит написать на последующих занятиях, в качестве начальной точки использует функцию `main`. Это связано с тем, что данные программы будут выполняться непосредственно с вашего компьютера. Аплеты, мобильные приложения и сервлеты запускаются на выполнение опосредованно, с помощью другой программы или устройства.

В дальнейшем программы, запускаемые непосредственно с компьютера, будем называть *приложениями*.

Фигурные скобки

В строках 3, 4, 6 и 7 программы `Saluton` находятся фигурные скобки `{` или `}`, которые применяются для группирования строк программы определенным образом (точно так же, как круглые скобки применяются в предложении для группирования слов). Все, что находится между открывающей фигурной скобкой, `{`, и закрывающей, `}`, относится к одной и той же группе.

Подобные группировки называются *блоком*. В коде из листинга 2.1 открывающая фигурная скобка в строке 3 связана с закрывающей фигурной скобкой в строке 7. В результате вся программа заключается в один блок. В данном случае фигурные скобки обозначают начало и конец программы.

Блоки могут находиться внутри других блоков (подобно тому, как используются круглые скобки в этом предложении (второй вложенный набор скобок)). Программа `Saluton` имеет фигурные скобки в строках 4 и 6, которые образуют другой блок, начинающийся с функции `main`. Строки, находящиеся внутри блока `main`, начинают выполняться после запуска программы.

СОВЕТ

Среда NetBeans подсказывает, где начинается и заканчивается блок. Щелкните на одной из фигурных скобок в исходном коде программы `Saluton`, и эта скобка окрасится в желтый цвет (вместе с соответствующей закрывающей скобкой). Инструкции Java, находящиеся между двумя фигурными скобками, образуют блок. Выделение фигурных скобок желтым цветом не требуется в таких коротких программах, как `Saluton`. Если же вы пишете большие программы, то этот прием будет весьма полезным.

В данном случае в блоке находится единственная инструкция:

```
// Код первой программы на Java
```

Эта строка является *заполнителем*. Символы `//`, находящиеся в начале строки, указывают компьютеру на то, что эту строку следует игнорировать, поскольку она предназначена исключительно для облегчения понимания исходного кода. Подобные строки называются *комментариями*.

Итак, вы написали завершенную программу на Java. Эту программу можно скомпилировать, но если запустить ее на выполнение, то ничего не произойдет. Причина заключается в том, что вы не дали указания компьютеру что-либо сделать. Блок `main` включает лишь единственный комментарий, игнорируемый компьютером. Теперь нужно добавить несколько инструкций в блок `main`, находящийся между открывающей и закрывающей фигурными скобками.

Сохранение информации в переменной

В программах, которые вы пишете, нужно предусмотреть место для хранения информации на протяжении короткого периода времени. Для этого можно использовать переменную, которая способна хранить такую информацию, как целые числа, числа с плавающей запятой, значения `true/false`, символы и текстовые строки. Информация, которая хранится в переменной, может изменяться, а получить к ней доступ можно по имени переменной.

В файле `Saluton.java` замените строку 5 следующей строкой:

```
String greeting = "Saluton mondo!";
```

Эта инструкция указывает компьютеру на то, что фраза `"Saluton mondo!"` должна храниться в переменной `greeting`.

В программе на Java нужно указать компьютеру тип информации, которая хранится в переменной. В нашей программе значение переменной `greeting` представляет собой текстовую строку, которая состоит из букв, цифр, знаков пунктуации и других символов. Включение ключевого слова `String` в инструкцию, описывающую переменную, приводит к тому, что переменная может хранить строковые значения.

При вводе этой инструкции в программу в конце строки нужно добавлять точку с запятой. Инструкции Java обязательно завершаются точкой с запятой, подобно тому, как предложения завершаются точками. Компьютер использует точку с запятой для того, чтобы понять, где заканчивается одна инструкция и начинается другая.

Чтобы сделать программу более понятной, помещайте в каждой строке одну инструкцию.

Отображение содержимого переменной

Если вы сейчас запустите программу, все равно будет казаться, что ничего не происходит. Команда, сохраняющая текст в переменной `greeting`, выполняется "за кулисами". Чтобы компьютер отобразил результаты вычислений, нужно вывести содержимое этой переменной.

В программе `Saluton` после инструкции `String greeting = "Saluton mondo!"` вставьте еще одну пустую строку, а затем добавьте следующую инструкцию:

```
System.out.println(greeting);
```

Эта инструкция предписывает компьютеру отобразить значение, которое хранится в переменной `greeting`. Функция `System.out.println()` указывает компьютеру отобразить информацию на системном устройстве вывода — вашем мониторе.

Сохранение завершенного проекта

Теперь ваша программа должна напоминать код из листинга 2.2, хотя в строках 5 и 6 могут быть другие отступы. Внесите необходимые исправления и сохраните файл с помощью команд Файл⇨Сохранить.

ЛИСТИНГ 2.2. Завершенная версия программы `Saluton`

```
1: package com.java24hours;
2:
3: class Saluton {
4:    public static void main(String[] arguments) {
5:       String greeting = "Saluton mondo!";
6:       System.out.println(greeting);
7:    }
8: }
```

Когда компьютер запускает эту программу, он выполняет каждую из инструкций блока `main`, находящихся в строках 5 и 6. В листинге 2.3 показана аналогичная программа, написанная на русском языке.

ЛИСТИНГ 2.3. Программа `Saluton` в переводе на русский язык

```
1: Поместите программу в пакет com.java24hours.
2:
3: Здесь начинается программа Saluton:
4:    Здесь начинается основная часть программы:
5:    Сохранить текст "Saluton mondo!" в переменной greeting
         типа String.
6:    Отобразить содержимое переменной greeting.
7:    Здесь завершается основная часть программы.
8: Здесь завершается программа Saluton.
```

Компилирование программы в файл класса

Перед запуском программы на Java ее нужно скомпилировать. В процессе компиляции программы инструкции, передаваемые компьютеру в программе, преобразуются в форму, которую лучше понимает компьютер.

Среда NetBeans компилирует программы автоматически сразу же после их сохранения. Как только вы введете код из листинга 2.2, программа скомпилируется автоматически.

После завершения компиляции создается новый файл `Saluton.class`. Все программы на Java компилируются в файлы классов с расширением `.class`.

Программа на Java может состоять из нескольких классов, которые работают совместно. Но такие простые программы, как `Saluton`, используют лишь один класс.

Компилятор преобразует исходный код Java в *байт-код*, выполняемый виртуальной машиной Java (Java Virtual Machine — JVM).

ПРИМЕЧАНИЕ

Компилятор Java выдает сообщения только в случае обнаружения ошибок. Если же в процессе компиляции ошибок не обнаружено, то на экране ничего не отображается.

Устранение ошибок

При создании программы в редакторе исходного кода NetBeans выявленные ошибки помечаются красным значком предупреждения, который отображается в левой части панели редактора (рис. 2.2).

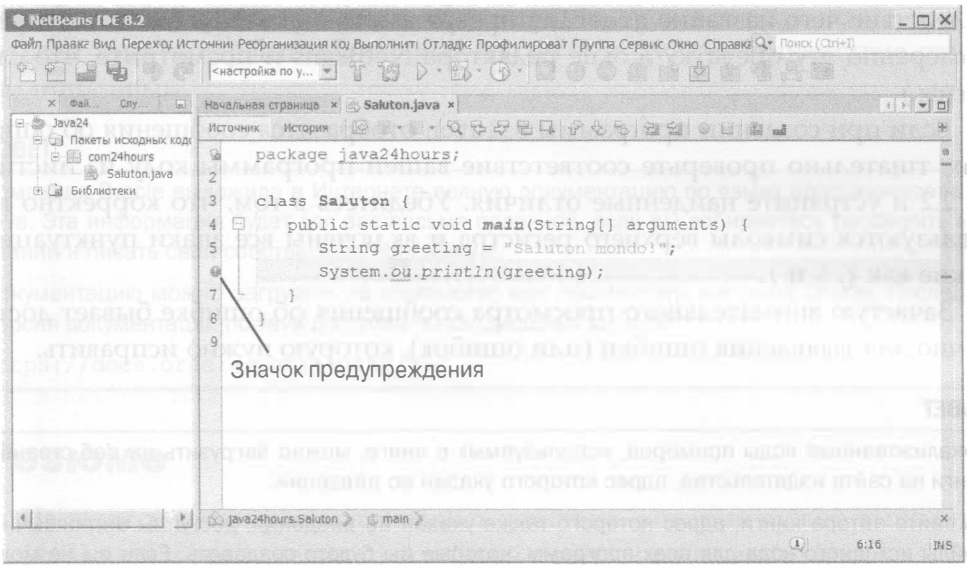


РИС. 2.2. Выделение ошибок в редакторе исходного кода

Значок появляется в строке, которая вызвала ошибку. Щелкните на этом значке, чтобы отобразить диалоговое окно, в котором подробно объясняется ошибка, обнаруженная компилятором, включая следующие сведения:

- название программы на Java;
- тип ошибки;
- строка, в которой найдена ошибка.

Ниже приводится пример диалогового окна с ошибкой, которое может появиться в процессе компиляции программы `Saluton`.

```
cannot find symbol.
symbol : variable greting
location: class Saluton
```

Сообщение об ошибке `cannot find symbol` (невозможно найти символ) отображается в первой строке диалогового окна. Если смысл сообщения об ошибке понять невозможно, не теряйте время на напрасные попытки. Лучше проанализировать строку с ошибкой, чтобы проверить наиболее очевидные причины ее появления.

Для начала ответьте на вопрос, содержит ли следующая инструкция ошибку:

```
System.out.println(greting);
```

Да, причина появления ошибки — опечатка в названии переменной, вследствие чего название `greeting` превратилось в `greting`. (Добавьте преднамеренно эту опечатку в окне редактора NetBeans и посмотрите, что случится.)

Если при создании программы `Saluton` отобразятся сообщения об ошибках, тщательно проверьте соответствие вашей программы коду из листинга 2.2 и устраните найденные отличия. Убедитесь в том, что корректно используются символы верхнего регистра и включены все знаки пунктуации, такие как `{`, `}` и `;`.

Зачастую внимательного просмотра сообщения об ошибке бывает достаточно для выявления ошибки (или ошибок), которую нужно исправить.

СОВЕТ

Локализованные коды примеров, используемых в книге, можно загрузить на веб-странице книги на сайте издательства, адрес которого указан во введении.

На сайте автора книги, адрес которого также указан во введении, доступны англоязычные файлы исходного кода для всех программ, которые вы будете создавать. Если вы не можете понять причину появления ошибок в программе `Saluton`, загрузите с сайта издательства файл `Saluton.java` и попытайтесь выполнить его.

Выполнение программы на Java

Для проверки программы `Saluton` запустите на выполнение файл класса с помощью интерпретатора Java. В среде NetBeans выберите пункт меню **Выполнить⇨Выполнить файл**. Под окном исходного кода появится панель **Вывод** с результатами выполнения программы (в случае отсутствия ошибок), как показано на рис. 2.3.

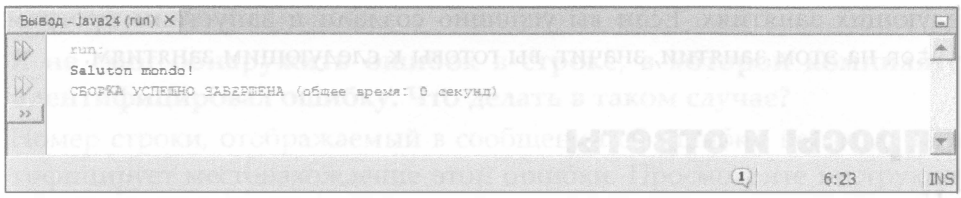


РИС. 2.3. Результат выполнения вашей первой программы на Java

Если вы увидите на экране текст "Saluton mondo!", значит, ваша первая программа работает успешно! Ваш компьютер только что поздоровался с окружающим миром в лучших традициях компьютерного программирования.

В этом месте книги у читателей может возникнуть вопрос: почему в данном случае используется фраза "Saluton mondo!"? Дело в том, что в переводе с эсперанто эта фраза означает "Здравствуй, мир!". Для справки: эсперанто — это искусственный язык, созданный Людвигом Заменхофом в 1887 году для облегчения международного общения.

Если язык эсперанто когда-либо войдет в моду, это приветствие станет традиционным.

СОВЕТ

Компания Oracle выложила в Интернете полную документацию по языку программирования Java. Эта информация будет для вас весьма полезной, если вы собираетесь расширить круг знаний и писать свои собственные программы.

Документацию можно загрузить на компьютер или просмотреть на сайте Oracle. Последняя версия документации по Java доступна по следующему адресу:

`https://docs.oracle.com/javase/10/`

Резюме

На этом занятии рассматривались действия, которые нужно выполнить для создания и выполнения простейшей программы на языке Java.

- ввести текст программы с помощью текстового редактора или редактора интегрированной среды разработки, такой как NetBeans;
- скомпилировать программу в файл класса;
- дать виртуальной машине Java указание выполнить класс;
- посмотреть на результат выполнения программы.

Также на этом занятии были рассмотрены некоторые понятия компьютерного программирования, такие как компиляторы, интерпретаторы, блоки, инструкции и переменные, которые мы дополнительно разъясним на

следующих занятиях. Если вы успешно создали и запустили программу `Saluton` на этом занятии, значит, вы готовы к следующим занятиям.

Вопросы и ответы

В. **Нужно ли включать определенное количество пробелов в строках программы на Java?**

О. Если речь идет о компьютере, то пробелы не требуются. Они нужны исключительно людям, просматривающим текст компьютерной программы, а компилятору Java абсолютно все равно, есть пробелы или нет. Можно написать программу `Saluton` вообще без пробелов или создать отступы в строках с помощью клавиши <Tab>, и она будет успешно скомпилирована.

Несмотря на то что количество пробелов перед строками не имеет какого-либо значения для компьютера, рекомендуется использовать согласованные интервалы и отступы в программах на Java. Почему? Потому что интервалы и отступы облегчают просмотр структуры программы и идентификацию программных блоков, к которым относятся рассматриваемые инструкции.

Текст программы должен быть понятен другим программистам, а также вам, когда вы будете просматривать код через несколько недель или даже месяцев для поиска ошибок или усовершенствования кода. Согласованное использование пробелов и отступов относится к стилю программирования. Хорошим программистам присущ свой стиль, который они последовательно используют в своей работе.

В. **Программа на Java может быть описана как класс или группа классов. Верно ли это?**

О. Верны оба утверждения. Простые программы на Java, которые будут созданы на нескольких следующих занятиях, компилируются в один файл с расширением `.class`. Этот файл можно запускать с помощью виртуальной машины Java. Программы на Java могут также состоять из нескольких классов, которые работают совместно. Эта тема будет подробно рассмотрена на занятии 10.

В. **Если в конце каждой инструкции нужно указывать точку с запятой, почему же в конце строки комментария (`// Код первой программы на Java`) они отсутствуют?**

О. Компилятор полностью игнорирует комментарии. Если он встречает в строке программы символы `//`, то будет игнорировать все, что находится справа от них. В следующем примере показан комментарий, находящийся в одной строке с инструкцией:

```
System.out.println(greeting); // Здравствуй, мир!
```

В. Я не смог обнаружить ошибок в строке, в которой компилятор идентифицировал ошибку. Что делать в таком случае?

О. Номер строки, отображаемый в сообщении об ошибке, не всегда идентифицирует местонахождение этой ошибки. Просмотрите инструкции, которые предшествуют сообщению об ошибке, на наличие опечаток или других ошибок. Обычно ошибка находится в том же программном блоке.

Коллоквиум

Проверьте знание материала, рассмотренного на этом занятии, ответив на следующие вопросы.

Контрольные вопросы

1. Что происходит при компилировании программы на Java?
   - **А.** Программа сохраняется на диске.
   - **Б.** Программа преобразуется в форму, лучше понимаемую компьютером.
   - **В.** Программа добавляется в коллекцию.
2. Что такое переменная?
   - **А.** Что-то, что качается, но не падает.
   - **Б.** Текст программы, игнорируемый компилятором.
   - **В.** Место в программе, предназначенное для хранения информации.
3. Как называется процесс поиска и устранения ошибок?
   - **А.** Размораживание.
   - **Б.** Отладка.
   - **В.** Декомпозиция.

Ответы

1. **Б.** В процессе компилирования программы файл `.java` преобразуется в файл `.class` или в набор файлов `.class`.
2. **В.** Переменные — это одно из мест, предназначенных для хранения информации. Позднее будут описаны и другие такие места, например массивы и константы.
3. **Б.** Процесс поиска и устранения ошибок называется *отладкой*. Некоторые среды разработки снабжены отладчиками, которые помогают при

поиске и устранении ошибок. Среда NetBeans содержит один из лучших отладчиков.

Упражнения

Если хотите глубже исследовать темы, рассмотренные на этом занятии, выполните следующие упражнения.

- С помощью переводчика Google (`http://translate.google.com`) можно перевести английскую фразу "Hello world!" на другие языки. Напишите программу, которая позволит компьютеру приветствовать мир на французском, итальянском или португальском языке.
- Вернитесь к программе `Saluton` и добавьте в нее одну или две ошибки. Например, удалите точку с запятой в конце строки или измените слово `println` на `println` (вместо буквы `l` введите цифру `1`). Сохраните программу и попробуйте ее скомпилировать. Сравните отображаемые сообщения об ошибках с ошибками, добавленными вами.

Ответы на эти упражнения можно найти на сайте книги, адрес которого указан во введении.

ЗАНЯТИЕ 3
Путешествие в мир Java

На этом занятии вы узнаете...

- об истории Java;
- о преимуществах этого языка;
- о примерах использования Java.

Прежде чем углубиться в дебри программирования на Java, немного отвлечемся. На этом занятии будет кратко описана история Java и на основе примеров приложений рассмотрены возможности языка. Также будут приведены примеры сайтов, на которых можно найти программы на Java.

Чтобы приступить к этому занятию, вам понадобится смартфон или планшет, на котором могут выполняться программы на Java.

Включите мобильное устройство и подготовьтесь к увлекательному путешествию вместе с Java. Но при этом вам не придется покидать свой дом. Конечно, вы не сможете ощутить простые радости туризма, такие как посещение экзотических мест, общение с аборигенами, знакомство с местной кухней и прочие прелести подобного рода. Но зато вам не придется преодолевать трудности путешествия, не понадобится загранпаспорт и не нужно запасаться водой и продуктами.

Первая остановка: Oracle

Наше путешествие в мир Java начинается с сайта `www.java.com`, который поддерживается Oracle, компанией-разработчиком этого языка.

Программа на Java, которая выполняется на мобильном устройстве, называется *мобильным приложением*. Мобильные приложения Java преимущественно предназначены для устройств Android. Java также применяется для разработки интернет-приложений, относящихся к трем категориям: программы для настольных компьютеров и ноутбуков, которые могут запускаться

с помощью веб-браузеров; сервлеты Java, которые запускаются веб-серверами для доставки веб-приложений; аплеты Java, отображаемые на веб-страницах в окнах браузера.

На рис. 3.1 показано окно многопользовательской онлайн-игры Celtic Heroes, основанной на Java и предназначенной для платформы Android. В эту игру можно играть бесплатно, установив ее из Google Play или магазина приложений вашего устройства.

РИС. 3.1. Игра Celtic Heroes, основанная на Java

После запуска игры можно в течение нескольких секунд создать персонаж и приступить к исследованию увлекательного мира фэнтези.

Подразделение Java компании Oracle курирует разработку языка Java. На сайте `Java.com` доступен бесплатный онлайн-журнал *Java Magazine*. Он включает примеры использования Java на смартфонах Android, сайтах и других платформах.

Компания Oracle также поддерживает сайт, ориентированный на программистов Java (`www.oracle.com/technetwork/java`) и содержащий последние версии сред разработки NetBeans и Java Development Kit, а также другие ресурсы для разработчиков.

Наше путешествие в мир Java начинается с платформы Android, поскольку именно для нее создается большинство программ на Java. После изучения основ Java вы сможете применить полученные навыки для разработки собственных приложений с помощью Android Software Development Kit (SDK).

Эта среда разработки доступна на бесплатной основе и может применяться на платформах Windows, Mac OS и Linux.

На сегодняшний день создано более 300000 приложений для смартфонов Android и других устройств с мобильными операционными системами. Методики, применяемые для создания таких приложений, будут рассмотрены на занятии 24.

Краткая история Java

Билли Джой, один из руководителей компании Sun Microsystems, после завершения работ по разработке Java заявил, что "в результате 15-летней работы был предложен более качественный и надежный способ написания компьютерных программ". Работа по созданию Java оказалась длительной и непростой.

В 1990-х годах Джеймс Гослинг разработал язык программирования Java в качестве "мозга" умных устройств (интерактивные телевизоры, микроволновые печи, путешествующие во времени терминаторы, военные спутники SkyNet, порабощающие человечество, и прочие подобные устройства). А все началось с того, что Гослинг оказался недоволен результатами, полученными в результате использования языка программирования C++. В порыве вдохновения он заперся в своем кабинете и начал разрабатывать новый, более подходящий язык программирования.

Гослинг назвал новый язык Oak (от англ. "oak" — дуб), поскольку именно это дерево было видно из окна его офиса. Создание данного языка было частью стратегии его компании по зарабатыванию денег в эпоху, когда интерактивное телевидение превратилось в индустрию с оборотом в миллионы долларов. Пока что им не удалось воплотить эти мечты в реальность, хотя успех таких компаний, как Amazon, Apple, Roku и ряда других, свидетельствовал о том, что игра стоит свеч. Новый язык программирования, изобретенный Гослингом, ждала особая судьба.

В силу сложившихся обстоятельств многие качества языка Oak позволили адаптировать его для применения в Интернете. Команда Гослинга разработала способ безопасного управления программами на веб-страницах и выбрала новое название для языка — Java.

ПРИМЕЧАНИЕ

Иногда название "Java" расшифровывают как "Just Another Vague Acronym" (еще один непонятный акроним). Возможно, это просто марка любимого кофе Гослинга.

На самом деле это название, скорее всего, было выбрано из-за его благозвучного звучания.

На данный момент вышло более 10 основных версий языка Java, каждая из которых включает новые функции. Ниже перечислены первые десять версий и вкратце описаны их ключевые новинки.

- **Java 1.0.** Оригинальная версия (1995 г.).
- **Java 1.1.** JDBC (Java Database Connectivity — соединение с базами данных на Java), улучшенные графические интерфейсы пользователя (1997 г.).
- **Java 2, версия 1.2.** Внутренние классы, подключаемый модуль Java для веб-браузеров и структуры данных (1998 г.).
- **Java 2, версия 1.3.** Расширенные мультимедийные возможности (2000 г.).
- **Java 2, версия 1.4.** Улучшенная поддержка Интернета, обработка XML-документов и утверждений (2002 г.).
- **Java 5.** Обобщенные типы данных, новые возможности для циклов, аннотации и автоматическое преобразование данных (2005 г.).
- **Java 6.** Встроенная база данных Derby и веб-сервисы (2006 г.).
- **Java 7.** Улучшенное управление памятью и ресурсами, графический интерфейс пользователя Nimbus (2011 г.).
- **Java 8.** Замыкания (2014 г.).
- **Java 9.** HTTP-клиент, JShell (2017 г.).

Весной 2018 года появилась очередная версия — Java 10. Она включает ряд интересных новинок, в том числе объявление типов переменных с помощью ключевого слова `var`.

Если перечисленные понятия, такие как внутренние классы, обобщенные типы данных или HTTP-клиент, вам незнакомы, не переживайте, все они будут подробно рассмотрены на последующих занятиях.

ПРИМЕЧАНИЕ

Версии Java имеют довольно причудливую нумерацию: 1, 2, 6. Причем седьмая версия называется Java 6, а номера некоторых версий выражаются числами с плавающей точкой. Подобная несколько хаотичная схема нумерации связана с тем, что разработчики Java первоначально не уделяли этому вопросу особого внимания. Но с 2006 года номера версий выражаются целыми последовательными числами.

Идем в школу вместе с Java

В Интернете доступно множество ресурсов, предназначенных для преподавателей и студентов. Поскольку приложениям Java присуща большая степень интерактивности, чем веб-страницам, этот язык является более подходящим для учебных курсов.

В качестве примера рассмотрим приложение FreeBalls, разработанное российским программистом Иваном Макляковым и представляющее собой бесплатный симулятор движения частиц. Эта программа использует Java для демонстрации анимации движения множества жидких или твердых частиц. Движением частиц можно управлять путем наклона экрана (рис. 3.2).

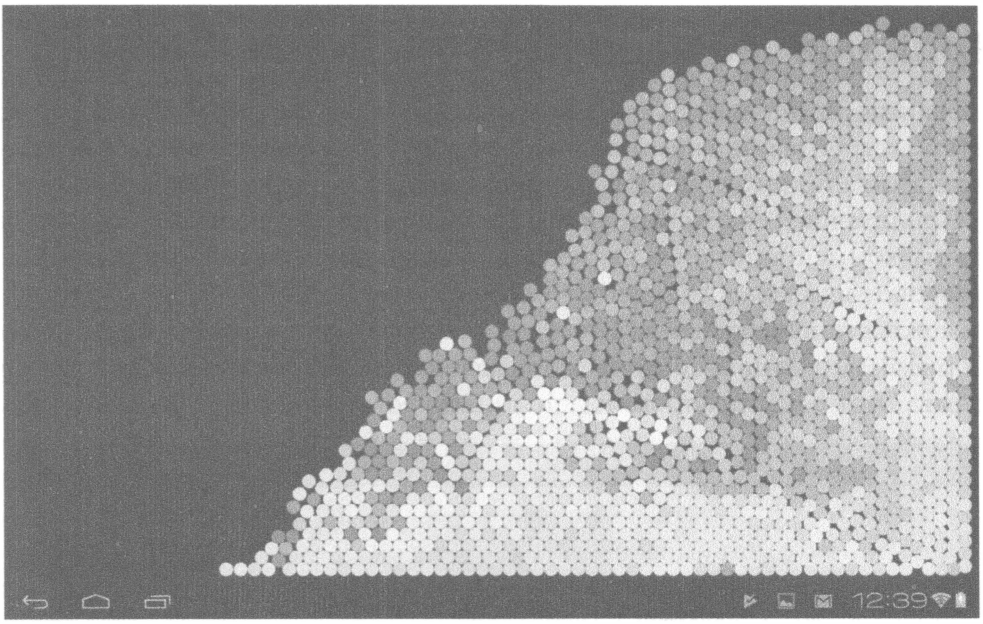


РИС. 3.2. С помощью Java можно написать физический симулятор

Существует множество учебных программ, предназначенных для разных операционных систем, но данная программа выделяется своей доступностью. Она может выполняться на любом устройстве Android. И вообще, программы на Java могут выполняться на любом компьютере, на котором установлена виртуальная машина Java (JVM).

Виртуальная машина Java, загруженная на мобильное устройство или в веб-браузер, идентична виртуальной машине Java, применяемой для выполнения приложения `Saluton` (см. занятие 2).

Программу на Java, такую как физический симулятор, не нужно создавать под конкретную операционную систему. И поскольку операционные системы, такие как Windows, также называются *платформами*, подобное свойство программ на Java называется *платформенной независимостью*. Язык программирования Java создавался для использования на разных операционных системах. Разработчики Java сделали его мультиплатформенным, поскольку он предназначается для использования на различных гаджетах и других электронных устройствах.

Пользователи могут выполнять программы, написанные на Java, без какой-либо дополнительной адаптации этих программ. При использовании корректного подхода к программированию на Java не потребуется создавать специальные версии программ, предназначенные для разных операционных систем и устройств.

Приложение Food Network in the Kitchen

Если после чтения двух предыдущих разделов вы уже успели проголодаться, устройте обеденный перерыв вместе с Food Network in the Kitchen — бесплатным приложением Java, разработанным кулинарным телеканалом.

Приложение Food Network in the Kitchen предлагает рецепты, кулинарные советы, комментарии пользователей и видеоролики от каждого из известных поваров телеканала. Одно из преимуществ этого приложения заключается в возможности предоставления доступа к интерактивным функциям и статьям. На рис. 3.3 показан рецепт, предоставленный шеф-поваром Джадой Де Лаурентис. Этот рецепт сопровождается видеоуроком по приготовлению блюда.

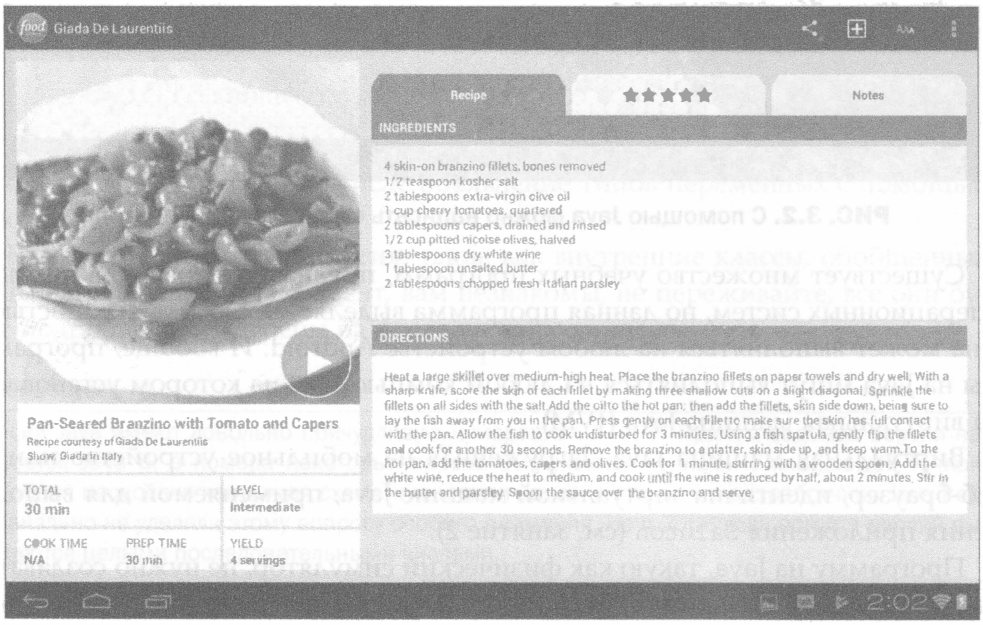


РИС. 3.3. Рецепт, отображаемый в окне приложения Food Network in the Kitchen

Магазин приложений, подобный тому, в котором находится приложение Food Network in the Kitchen, предлагает пользователям тысячи программ. Поскольку эти программы созданы разными разработчиками, для защиты пользователей и устройств должны предприниматься определенные меры

безопасности. Одна из проблем, которая обсуждалась с момента появления Java, связана с безопасностью этого языка.

Важность обеспечения безопасности связана со способом выполнения программ на Java, поставляемых в виде приложений. Приложения, рассматриваемые на этом занятии, загружаются на смартфон или планшет, после чего их можно запустить на выполнение.

Как правило, большинство приложений публикуются незнакомыми вам людьми. С точки зрения безопасности запуск подобных приложений не слишком отличается от предоставления собственного компьютера в пользование посторонним людям. Если бы язык Java не обладал встроенными средствами обеспечения безопасности, то написанные на нем программы могли бы заражать систему вирусами, удалять файлы, демонстрировать рекламу и выполнять другие неожиданные действия.

В Java имеется несколько средств, обеспечивающих безопасность при запуске приложений или выполнении программ с веб-страниц. Этот язык программирования считается довольно безопасным для использования в Интернете, но выявленные в последнее время прорехи в системе безопасности привели к тому, что некоторые эксперты рекомендуют пользователям полностью отключить Java в браузере. Большинство популярных современных браузеров препятствует запуску аплетов Java на сайтах.

С Java чаще имеют дело пользователи мобильных приложений, серверных программ и программ для настольных компьютеров. Поэтому на данном занятии вы будете иметь дело с приложениями, а не с аплетами Java.

Наблюдаем за небом

Первая послеобеденная остановка нашего тура по стране Java — путешествие в космос вместе с NASA (государственное агентство США, которое интенсивно использует возможности Java). Один из примеров такого использования — бесплатное приложение ISS Detector, разработанное компанией RunaR и позволяющее отслеживать положение МКС и некоторых других искусственных спутников Земли.

Приложение ISS Detector накладывает текущее положение и траекторию небесных объектов на карту ночного неба. Как показано на рис. 3.4, приложение отображает положение спутника Iridium 96 (GALEX), перемещающегося в восточной части неба между Ураном и Венерой.

Это приложение перерисовывает положение каждого отслеживаемого спутника по мере его перемещения и изменяет ориентацию звездного неба в зависимости от направления взгляда пользователя. Подобное обновление в режиме реального времени возможно в силу того, что язык Java является многопоточным, благодаря чему компьютер может одновременно выполнять

несколько операций. Одна часть программы выполняет одну задачу, другая часть — другую, причем обе части не оказывают влияния друг на друга. Каждая часть программы в этом примере называется *потоком*.

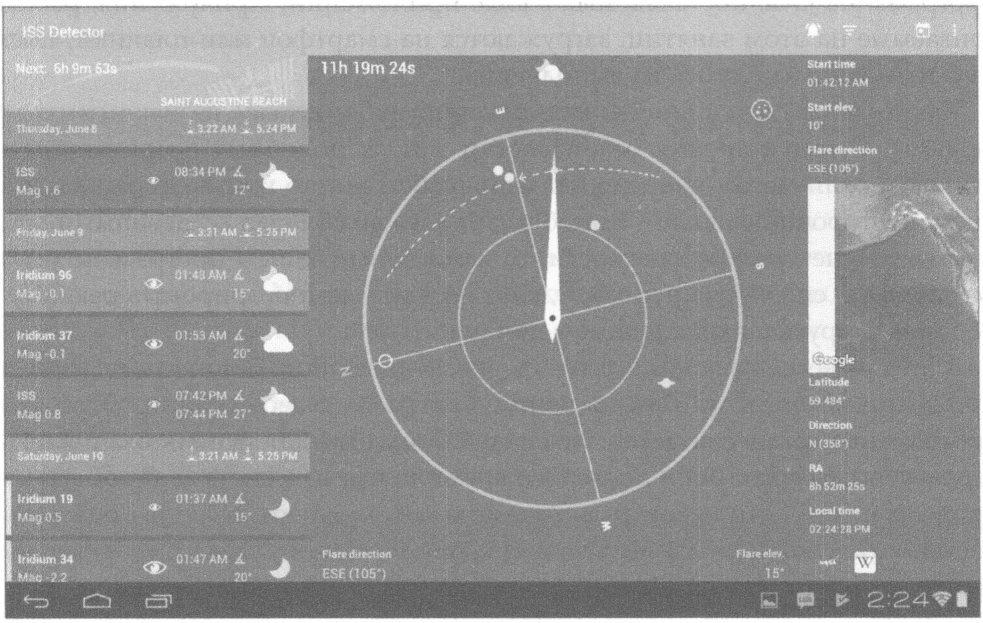


РИС. 3.4. С помощью приложения ISS Detector можно отслеживать положение и траекторию движения искусственных спутников Земли

В программе ISS Detector положение каждого спутника может вычисляться в своем собственном потоке. Если вы работаете в такой операционной системе, как Windows 10, то благодаря присущей ей многопоточности можно одновременно выполнять несколько программ. Например, можно играть в игру Minecraft в одном окне, составлять отчет о продажах компании в другом, общаться с другом с помощью Skype в третьем, и все это выполнять одновременно!

Пришло время заняться бизнесом

У читателей, наверное, сложилось впечатление, будто Java в основном используется для наблюдений за космическими объектами, обмена кулинарными рецептами и участия в войнах эльфов. Чтобы развеять это обманчивое впечатление, рассмотрим применение Java для бизнеса.

С сайта uInvest Studio загрузите и установите приложение Realtime Stock Quotes, которое отображает обновленные цены акций из пользовательского портфеля. На рис. 3.5 показаны текущие данные для биржевого тикера GOOG (материнская компания Google под названием Alphabet Inc.).

РИС. 3.5. Данные рынка акций, отображаемые в окне приложения Realtime Stock Quotes

Создать программу, подобную Realtime Stock Quotes, можно несколькими способами. Один из них заключается в том, чтобы представлять программу как объект. Под *объектом* понимают сущность, которая существует в мире, занимает место в пространстве и может выполнять определенные операции. Согласно принципам объектно-ориентированного программирования, применяемого для создания программ на Java (см. занятие 10), компьютерная программа создается в виде группы объектов. Каждый объект выполняет определенные функции и знает, каким образом взаимодействовать с другими объектами. Например, программа биржевого тикера может быть настроена в виде группы следующих объектов:

- объект `quote` (котировка), который представляет котировку отдельных акций;
- объект `portfolio` (портфель), включающий набор котировок для конкретных акций;
- объект `ticker` (тикер), отображающий портфель;
- объект `Internet`, `user` (пользователь) и ряд других объектов.

В рамках этой модели программа биржевого тикера представляет собой коллекцию всех объектов, требуемых для выполнения работы.

Благодаря использованию принципов ООП возможно написание программ с расширенным набором полезных функций. Например, рассмотрим

ту же программу биржевого тикера. Если программист захочет использовать возможности котировок из этой программы в другой программе, достаточно лишь включить в новую программу объект `quote`, и не придется вносить какие-либо дополнительные изменения.

Программы, созданные с применением объектов, структурированы, поэтому их легче поддерживать. Объект содержит данные, требуемые для его функционирования, и код, используемый для выполнения требуемых функций. Объекты также придают программам расширяемость. На основе существующего объекта можно создать новый объект, наделив его иными возможностями.

Хранилище программ

А сейчас рассмотрим хранилище программ, доступное на сайте Source Forge. Здесь вы найдете программы, написанные на Java (или каком-либо другом языке программирования). Сайт этого хранилища доступен по адресу `www.sourceforge.net`.

На сайте SourceForge размещено колоссальное хранилище универсальных программных проектов. Если вы заинтересованы в привлечении других программистов к разработке своей программы, начните создавать проект на сайте SourceForge и откройте общий доступ к файлам. На этом сайте находятся более 430000 проектов с открытым кодом, благодаря чему программисты могут обмениваться *исходным кодом*, т.е. текстовыми файлами, на основе которых создаются компьютерные программы. Пример исходного кода — файл `Saluton.java`, который был создан на занятии 2.

Если в поле поиска на начальной странице SourceForge ввести слово **`Java`**, в папке проектов отобразится более 61000 листингов кода.

Одно из приложений Java, которое можно найти на сайте SourceForge, — JSoko, Java-версия игры Sokoban (рис. 3.6). Эта игра-головоломка включает анимацию, графику и звук, а управление осуществляется с помощью клавиатуры. Чтобы загрузить это приложение и просмотреть соответствующий исходный код на Java, посетите следующий сайт:

```
https://sourceforge.net/projects/jsokoapplet
```

В Java имеется огромная библиотека классов, которые можно использовать в разрабатываемых вами программах. В игре JSoko используется библиотечный класс `Image` (файл `java.awt`) для отображения графики и класс `AudioInputStream` (файл `javax.sound.sampled`) для воспроизведения звуков, имитирующих перемещение погрузчика и передвижение ящиков.

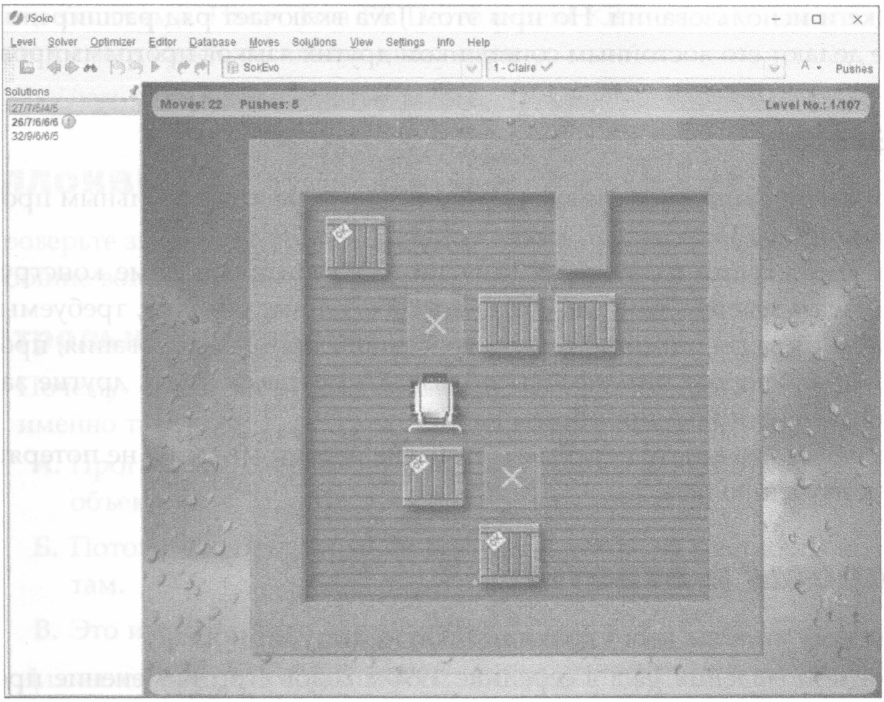


РИС. 3.6. На сайте SourceForge можно найти исходный код для программ на Java, например для версии игры Sokoban с открытым исходным кодом, которая называется JSoko

Одна из причин засилья программ, написанных на Java, в хранилище SourceForge и в других хранилищах — простота изучения и богатый набор функциональных возможностей Java.

При разработке языка программирования Java ставилась цель максимального упрощения его изучения. В 1990-х годах Джеймс Гослинг пытался использовать язык программирования C++ в своем проекте интеллектуального устройства. Большая часть кода Java основана на коде языка C++, поэтому программистам, владеющим C++, будет проще освоить Java. В то же время некоторые из элементов языка C++, которые трудны в освоении и применении, отсутствуют в Java.

Как уже отмечалось, Java проще в изучении, чем C++, который предназначен для опытных пользователей, желающих "выжать" все возможности компьютера. Языки типа C++ включают специальные конструкции, которые востребованы лишь опытными программистами.

Язык Java не предлагает своим пользователям продвинутые возможности — он настолько прост, насколько может быть простым объектно-ориентированный язык программирования. При разработке Java ставилась задача создать язык программирования, который был бы простым в изучении,

отладке и использовании. Но при этом Java включает ряд расширений, которые делают его достойным соперником других языков программирования.

Резюме

По завершении этого занятия пришло время заняться реальным программированием на Java.

На протяжении следующих занятий вы освоите основные конструкции языка Java, узнаете о том, как создавать собственные объекты, требуемые для выполнения задач объектно-ориентированного программирования, проектировать графические интерфейсы пользователя и выполнять другие задачи, связанные с программированием на Java.

Не забывайте делать перерывы, чтобы не переутомиться и не потерять интерес к изучению Java.

Вопросы и ответы

В. Почему аплеты Java утратили былую популярность?

О. После появления Java в середине 1990-х годов его применение преимущественно ограничивалось написанием аплетов. В те времена лишь с помощью Java можно было создавать интерактивные программы, выполнявшиеся в браузере. Спустя годы появились альтернативные способы обеспечения интерактивности, такие как Macromedia Flash, Microsoft Silverlight и новый стандарт публикаций в Интернете HTML5.

Былой популярности аплетов сильно навредили большое время загрузки, задержки во внедрении поддержки новых версий Java разработчиками браузеров и бреши в системе безопасности, используемые хакерами. И хотя аплеты исчезают из Интернета, Java остается по-прежнему популярным. Этому способствует эволюция Java до уровня сложного универсального языка программирования.

В. Чем отличается Java SE (Standard Edition) от Java EE (Enterprise Edition)? Распространяется ли Java EE на платной основе?

О. Версия Java Enterprise Edition является расширением версии Java Standard Edition и включает пакеты, предназначенные для поддержки передовых технологий, таких как Enterprise JavaBeans, обработка XML-документов и разработка сервлетов (программы на Java, которые выполняются на веб-сервере). Версия Java EE также включает сервер приложений, сложную среду выполнения приложений Java, предназначенную для корпораций и других крупных организаций с большими вычислительными

потребностями. Пакет Java EE Development Kit можно бесплатно загрузить с сайта Oracle по следующему адресу:

```
http://www.oracle.com/technetwork/java/javaee
```

Коллоквиум

Проверьте знание материала, рассмотренного на этом занятии, ответив на следующие вопросы.

Контрольные вопросы

1. Почему объектно-ориентированное программирование называется именно так?
   - **А.** Программы рассматриваются как группа совместно работающих объектов.
   - **Б.** Потому что программисты ориентируются на местности по объектам.
   - **В.** Это имя получено от родителей.
2. Для какой среды предназначены мобильные приложения Java?
   - **А.** Для браузеров.
   - **Б.** Для смартфонов либо планшетов Android.
   - **В.** Для настольных компьютеров.
3. Что требуется компьютеру или устройству для выполнения программ на Java?
   - **А.** Компилятор Java.
   - **Б.** Виртуальная машина Java.
   - **В.** Компилятор и виртуальная машина Java.

Ответы

1. **А.** Для обозначения объектно-ориентированного программирования также используется аббревиатура ООП.
2. **Б.** Все приложения Android написаны на Java. Они могут выполняться в среде браузеров и на настольных компьютерах, но прежде всего ориентированы на использование мобильных устройств и планшетов.
3. **Б.** Виртуальная машина Java (JVM) — это интерпретатор, преобразующий байт-код Java в инструкции, которые могут выполняться на компьютере или устройстве. Компилятор нужен для создания программ на Java, но не для их выполнения.

Упражнения

Чтобы лучше закрепить материал этого занятия, выполните следующие упражнения.

- На сайте SourceForge (`www.sourceforge.net`) найдите карточные игры, разработанные с помощью Java.
- С помощью Google Play на устройстве Android найдите физические симуляторы. Выберите один из наиболее интересных симуляторов и установите на свое устройство или компьютер.

Ответы на эти упражнения можно найти на сайте книги, адрес которого указан во введении.

ЗАНЯТИЕ 4
Принципы работы программ на Java

На этом занятии вы узнаете...

- как работают приложения;
- как передавать аргументы приложениям;
- как организованы программы на Java;
- как использовать библиотеку классов Java;
- как работать с JShell;
- как создавать объекты в приложениях.

При написании программ на Java важно понимать, где они будут выполняться. Одни программы предназначены для настольных компьютеров, другие — для смартфонов и планшетов.

Программы на Java, которые выполняются локально на компьютере, называются *приложениями*, программы, выполняемые на веб-серверах, — *сервлетами*, а программы, выполняемые на мобильных устройствах, — *мобильными приложениями*.

На этом занятии будет создано приложение, предназначенное для выполнения на компьютере.

Создание приложения

Примером приложения Java может служить программа `Saluton`, которая была создана на занятии 2. На этом занятии будет создано приложение, вычисляющее квадратный корень числа и отображающее полученное значение на экране.

В среде разработки NetBeans откройте проект `Java24` и начните создавать новое приложение.

1. Выберите пункт меню **Файл⇨Создать файл**. На экране появится окно мастера создания нового файла.

2. На панели категорий выберите пункт **Java**, а на панели типов файлов — **Пустой файл Java** и щелкните на кнопке **Далее**.
3. Введите имя класса **Root**.
4. Введите название пакета **com.java24hours**.
5. Щелкните на кнопке **Готово**.

После этого среда разработки NetBeans создаст файл `Root.java` и откроет пустой файл в окне редактора исходного кода. Введите код из листинга 4.1, но не вводите номера строк и двоеточия, находящиеся в левой части листинга, поскольку они используются лишь для пометки строк программы. Завершив ввод кода, сохраните файл, щелкнув на кнопке **Сохранить все** панели инструментов.

ЛИСТИНГ 4.1. Исходный код программы `Root.java`

```
 1: package com.java24hours;
 2:
 3: class Root {
 4:    public static void main(String[] arguments) {
 5:       int number = 225;
 6:       System.out.println("Квадратный корень из "
 7:          + number
 8:          + " равен "
 9:          + Math.sqrt(number)
10:       );
11:    }
12: }
```

Приложение `Root` работает следующим образом.

- Строка 1. Приложение помещается в пакет `com.java24hours`.
- Строка 5. Целочисленное значение `225` присваивается переменной `number`.
- Строки 6–10. Отображается исходное целое число и значение квадратного корня из этого числа. Инструкция `Math.sqrt(number)`, находящаяся в строке 9, вычисляет значение квадратного корня.

Если вы ввели без каких-либо опечаток код из листинга 4.1, правильно расставили все знаки пунктуации и символы верхнего регистра, запустите файл в NetBeans, выбрав пункт меню **Выполнить⇨Выполнить файл**. Результат выполнения приложения на панели **Вывод** показан на рис. 4.1.

После запуска приложения Java виртуальная машина Java (JVM) ищет блок `main()` и начинает выполнять инструкции Java, находящиеся в нем. Если блок `main()` отсутствует в программе, JVM отобразит сообщение об ошибке.

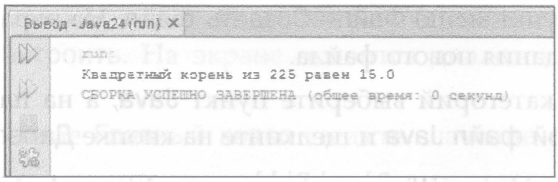


РИС. 4.1. Результат выполнения приложения `Root`

Инструкция `Math.sqrt(number)` в строке 9 содержит встроенную функцию языка Java, вычисляющую квадратный корень числа. В данном случае используется программа на Java под названием `Math`, которая с помощью метода `sqrt()` вычисляет квадратный корень числа.

Программа `Math` входит в библиотеку классов Java, которая будет рассмотрена далее.

Передача аргументов приложениям

Приложения Java можно запускать в командной строке с помощью программы `java`, вызывающей виртуальную машину Java. Среда NetBeans использует эту программу в фоновом режиме при выполнении приложений Java. Если программа `java` выполняется как команда, виртуальная машина Java загружает приложение. Команда может включать дополнительную информацию, как показано в следующем примере:

```
java TextDisplayer readme.txt /p
```

Любая дополнительная информация, передаваемая программе, называется *аргументом*. Первый аргумент (при его наличии) указывается через пробел после названия приложения. Каждый дополнительный аргумент также отделяется пробелом от предыдущего аргумента. В предыдущем примере аргументами являются `readme.txt` и `/p`.

Если нужно включить пробел внутри аргумента, заключите его в кавычки, как показано в следующем примере:

```
java TextDisplayer readme.txt /p "Page Title"
```

В этом примере выполняется программа `TextDisplayer` с тремя аргументами: `readme.txt`, `/p` и `"Page Title"`. Благодаря кавычкам предотвращается трактовка `Page` и `Title` в качестве отдельных аргументов.

Приложению Java можно передавать произвольное количество аргументов (в пределах разумного). Чтобы иметь возможность обработки аргументов, следует создать соответствующие инструкции в приложении.

Чтобы увидеть, каким образом аргументы используются в приложении, в проекте `Java24` создайте новый класс.

1. Выберите пункт меню **Файл⇨Создать файл**. На экране появится окно мастера создания нового файла.
2. На панели категорий выберите пункт **Java**, а на панели типов файлов — **Пустой файл Java** и щелкните на кнопке **Далее**.
3. Присвойте классу имя `BlankFiller`, пакету — название `com.java24hours` и щелкните на кнопке **Готово**.

Введите текст из листинга 4.2 в окне редактора исходного кода и сохраните файл. Скомпилируйте программу и исправьте ошибки, которые могут появиться при вводе кода.

ЛИСТИНГ 4.2. Исходный код программы `BlankFiller.java`

```
 1: package com.java24hours;
 2:
 3: class BlankFiller {
 4:    public static void main(String[] arguments) {
 5:       System.out.println("The " + arguments[0]
 6:          + " " + arguments[1] + " fox "
 7:          + "jumped over the "
 8:          + arguments[2] + " dog."
 9:       );
10:    }
11: }
```

Это приложение успешно компилируется и может быть запущено на выполнение, но в случае попытки его запуска с помощью команды меню **Выполнить⇨Выполнить файл** на экране появится сообщение об ошибке компиляции, имеющее довольно сложную структуру (рис. 4.2).

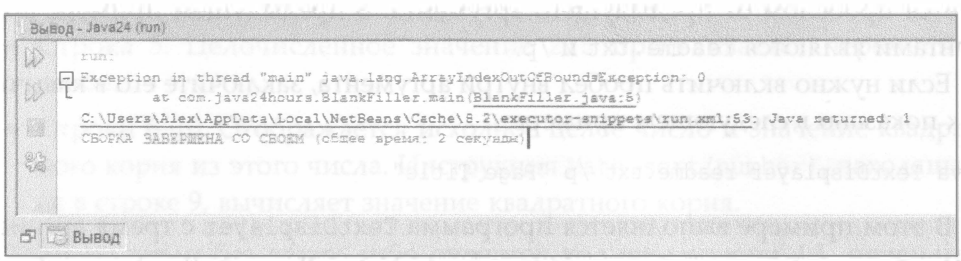


РИС. 4.2. Первая попытка компиляции завершилась провалом

Причина появления этой ошибки — ожидание программой трех аргументов, получаемых после ее запуска. Эти аргументы можно указать путем настройки проекта в NetBeans.

1. Выберите пункт меню Выполнить⇨Установить конфигурацию проекта⇨Настроить. На экране появится диалоговое окно Свойства проекта.
2. В текстовое поле Главный класс введите значение **com.java24hours.BlankFiller**.
3. В поле Аргументы введите **retromingent purple lactose-intolerant** и щелкните на кнопке OK.

Поскольку настройки проекта были изменены, следует использовать немного иной способ его запуска на выполнение. Выберите пункт меню Выполнить⇨Запустить проект. Приложение использует указанные аргументы как прилагательные в предложении (рис. 4.3).

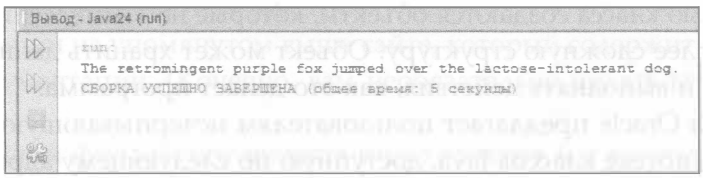


РИС. 4.3. Результат выполнения приложения `BlankFiller`

Вернитесь в диалоговое окно Свойства проекта и укажите три прилагательных, которые будут использоваться как аргументы. Это позволит гарантировать, что всегда указаны как минимум три значения аргументов.

С помощью аргументов обеспечивается простой способ изменения поведения программы. Для хранения аргументов используется особый тип переменной — *массив*. (Массивы будут подробно рассмотрены на занятии 9.)

Библиотека классов Java

В этой книге описывается использование Java для создания пользовательских программ с нуля. Вы изучите все ключевые слова и операторы языка и сможете на их основе формировать инструкции, с помощью которых компьютер будет выполнять полезные и интересные операции.

Используемый в книге подход является наилучшим для изучения Java, хотя он довольно трудоемкий. Дело в том, что вам придется выполнять много действий, которые необязательны для создания и запуска программы на Java.

Обычно программистам на Java не приходится делать работу с нуля, поскольку многое уже сделано за них. Обычно они используют готовые компоненты, которые требуют минимальной доработки.

В комплект поставки Java входит огромная коллекция кода — *библиотека классов Java*, которую можно применять в пользовательских программах.

В состав этой библиотеки входят тысячи классов, многие из которых можно применять для создания пользовательских программ.

ПРИМЕЧАНИЕ

Существует множество библиотек классов, предлагаемых другими компаниями. Создатели веб-сервера Apache поддерживают проект Apache Project, в рамках которого опубликовано более десятка проектов Java с открытым исходным кодом. Один из таких проектов — Tomcat — включает набор классов, предназначенных для создания веб-сервера, который может выполнять веб-приложения, реализованные в виде сервлетов Java.

Для получения дополнительных сведений о проекте посетите сайт `http://tomcat.apache.org`. Чтобы просмотреть все Java-проекты Apache, посетите сайт `https://projects.apache.org`.

Классы могут использоваться в программах так же, как и переменные.

С помощью класса создаются объекты, которые напоминают переменные, но имеют более сложную структуру. Объект может хранить данные подобно переменной и выполнять действия, как это делает программа.

Компания Oracle предлагает пользователям исчерпывающую документацию по библиотеке классов Java, доступную по следующему адресу:

`https://docs.oracle.com/javase/9/docs/api/overview-summary.html`

Страница этого сайта показана на рис. 4.4.

РИС. 4.4. Документация по библиотеке классов Java

Классы Java организованы в виде пакетов, которые выполняют те же функции, что и папки с файлами на компьютере. Программы, которые были созданы ранее, относятся к пакету `com.java24hours`.

Начальная страница сайта с документацией разделена на фреймы. В наибольшем фрейме отображаются все пакеты, которые формируют библиотеку классов Java.

Названия пакетов помогают определять их назначение. Например, пакет `java.io` — это набор классов, предназначенных для ввода и вывода данных с дисков, серверов Интернета и других источников данных. Пакет `java.time` содержит классы, предназначенные для работы со значениями даты и времени. Пакет `java.util` включает полезные служебные классы.

В главном фрейме, находящемся на начальной странице документации, отображается список пакетов, сопровождаемый коротким описанием. Щелкните на названии пакета, чтобы получить дополнительные сведения о нем. Страница загрузит перечень классов, находящихся в пакете.

Каждый класс в библиотеке классов Java имеет свою собственную страницу документации на упомянутом выше сайте, который содержит более 22000 страниц документации. (Конечно, вам необязательно читать всю эту документацию.)

При создании финального проекта этого занятия мы воспользуемся библиотекой классов Java и одним из содержащихся в ней классов Java, который и будет выполнять необходимую работу.

Программа `Dice` использует класс `Random` из пакета `java.util` для генерирования случайных чисел. Первое действие, выполняемое программой, заключается в импорте соответствующего пакета с помощью следующей инструкции:

```
import java.util.*;
```

После выполнения этой инструкции можно ссылаться на класс `Random` без указания его полного имени (`java.util.Random`), т.е. в виде `Random`. Звездочка означает, что можно ссылаться на все классы в пакете по их коротким именам.

Если же нужно импортировать только класс `Random`, воспользуйтесь следующей инструкцией:

```
import java.util.Random;
```

Программу на Java можно рассматривать как объект, который выполняет ту или иную задачу (или задачи). Класс `Random` используется в качестве шаблона объекта `Random`. Для создания объекта применяется ключевое слово `new`, за которым следуют название класса и круглые скобки.

```
Random generator = new Random();
```

В результате выполнения этой инструкции создается переменная `generator`, которая сохраняет новый объект `Random`, представляющий собой

генератор случайных чисел. Задачи, выполняемые объектом, называются *методами*.

В данной программе метод `nextInt()` объекта будет использован для создания случайного целочисленного значения.

```
int value = generator.nextInt();
```

Целые числа в Java находятся в диапазоне от –2 147 483 648 до 2 147 483 647. Генератор случайным образом выбирает одно из целых чисел и присваивает его переменной `value`.

Если бы не класс `Random` из библиотеки классов Java, то вам пришлось бы писать собственную программу, генерирующую случайные числа. Разработка подобной программы является весьма сложной задачей. Случайные числа применяются в играх, образовательных и других программах, которые должны выполнять какие-либо случайные операции.

В среде NetBeans создайте новый пустой класс Java, присвойте ему имя `Dice` и поместите в пакет `com.java24hours`. После открытия окна редактора исходного кода введите текст из листинга 4.3 и щелкните на кнопке **Сохранить** (или же выберите пункт меню **Файл⇨Сохранить**).

ЛИСТИНГ 4.3. Исходный код программы `Dice.java`

```
 1: package com.java24hours;
 2:
 3: import java.util.*;
 4:
 5: class Dice {
 6:    public static void main(String[] arguments) {
 7:       Random generator = new Random();
 8:       int value = generator.nextInt();
 9:       System.out.println("Случайное число "
10:      + value);
11:    }
12: }
```

Выберите пункт меню **Выполнить⇨Выполнить файл**, чтобы запустить программу. Результат выполнения программы показан на рис. 4.5. Сгенерированное вашей программой случайное число может отличаться от числа, показанного на рис. 4.5. (Вероятность совпадения этих чисел меньше одного шанса на четыре миллиарда, что значительно меньше шанса выигрыша в лотерею.)

Теперь у вас появилось сгенерированное случайное число, которое вы сможете назвать своим любимым числом.

Класс `Random`, подобно всем другим классам в библиотеке классов Java, снабжен обширной документацией, которую можно найти на сайте Oracle.

В этой документации описывается назначение класса, пакет, к которому он относится, способы создания объекта на основании этого класса и методы, которые могут вызываться для выполнения каких-либо действий.

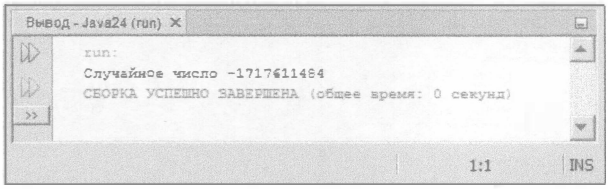


РИС. 4.5. Результат выполнения программы `Dice`

Чтобы получить доступ к документации, выполните следующие действия.

1. В окне браузера перейдите по следующему адресу:

   `https://docs.oracle.com/javase/9/docs/api/overview-summary.html`

2. На панели навигации, находящейся в левой верхней части страницы, щелкните на ссылке **Frames** (Фреймы). Документация откроется в нескольких фреймах.
3. Во фрейме **Modules** (Модули), находящемся в левой верхней части окна, щелкните на ссылке `java.base`.
4. В главном фрейме выполняйте прокрутку вниз до тех пор, пока не появится ссылка на пакет `java.util`. Щелкните на этой ссылке, и на экране отобразится документация по данному пакету.
5. В главном фрейме выполните прокрутку вниз и щелкните на ссылке `Random`.

Если ваш опыт программирования на Java ограничивается четырьмя занятиями из этой книги, то вам будет сложно разобраться в документации. Что вполне закономерно, поскольку она адресована в первую очередь опытным программистам.

Но по мере чтения книги у вас будет возникать потребность в изучении встроенных классов. И в таком случае без официальной документации не обойтись. Чаще всего вы будете искать информацию о применении методов класса, каждый из которых решает ту или иную задачу.

Если, например, потребуется информация об использовании метода `nextInt()`, который указан в строке 8 листинга 4.3, прокрутите документацию по классу `Random`, пока не отобразится соответствующий раздел (рис. 4.6).

ПРИМЕЧАНИЕ

В книге описаны все классы Java, используемые в примерах, поэтому вам вряд ли понадобится онлайн-документация для изучения примеров. Но поскольку эти классы обладают множеством дополнительных возможностей, описание которых выходит за рамки книги для начинающих, документация по библиотеке классов Java все же будет полезной.

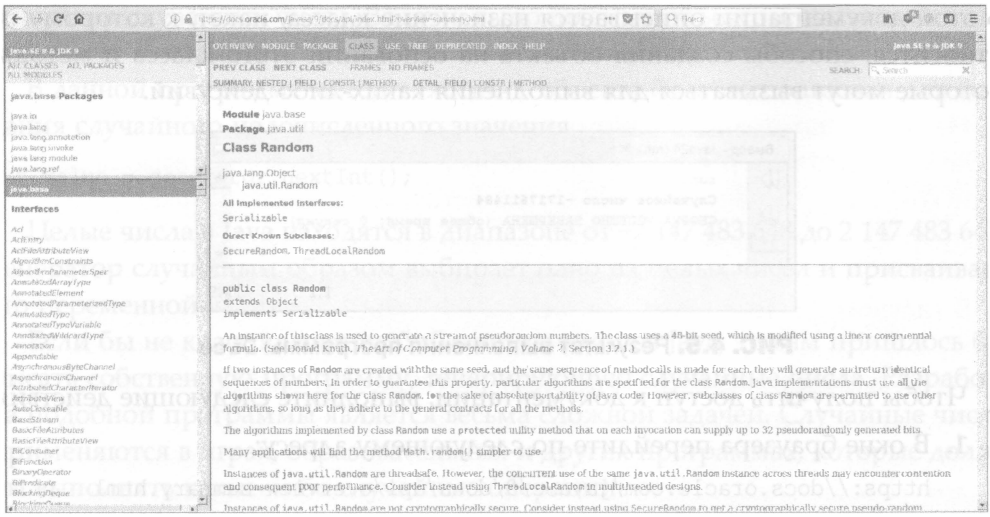


РИС. 4.6. Документация Oracle для класса `Random`

Знакомство с JShell

В Java 9 появилось новое средство — JShell, облегчающее изучение языка. Эта утилита командной строки включена в набор Java Development Kit (JDK). Она позволяет последовательно вводить инструкции Java и сразу же видеть результаты их выполнения.

JShell — это *оболочка языка*, представляющая собой интерактивную среду разработки, которая позволяет "играть" с Java путем ввода команд. Каждая вводимая команда является инструкцией Java. Как только вы нажмете клавишу <Enter>, инструкция будет выполнена виртуальной машиной Java, как если бы она была частью завершенной программы на Java.

Чтобы запустить JShell, найдите в вашей файловой системе папку, в которой установлен пакет JDK. В этой папке находится подпапка `bin`. Откройте эту папку и дважды щелкните на значке `jshell`.

В открывшемся окне введите код Java и посмотрите, что он делает. Окно JShell показано на рис. 4.7.

Команды JShell вводятся в командной строке после приглашения `jshell>`. На рис. 4.7 показан набор из трех введенных инструкций и полученных результатов.

1. `System.out.println(Math.sqrt(19600));` — вычисление квадратного корня из целого числа.
2. `int x = 100;` — присваивание значения переменной `x`.
3. `System.out.println(x);` — вывод значения переменной `x`.

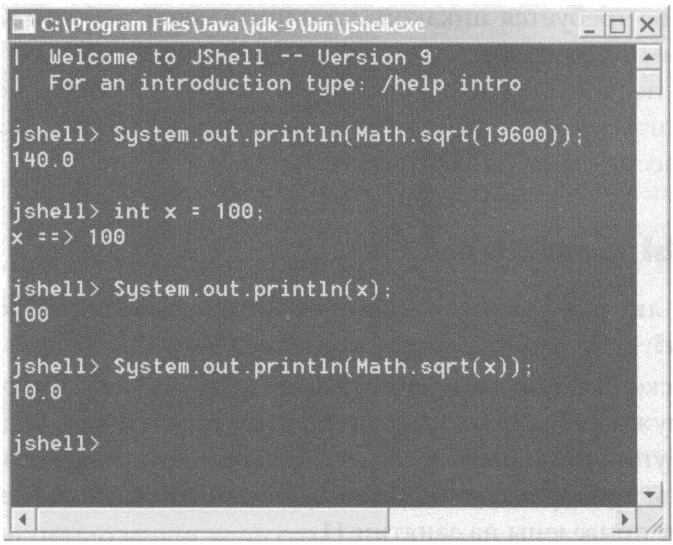


РИС. 4.7. Играем в песочнице Java с помощью JShell

4. `System.out.println(Math.sqrt(x));` — вычисление квадратного корня значения переменной `x`.

После ввода каждой инструкции JShell отображает значение, сгенерированное этой инструкцией. Когда создается переменная, она сохраняется в памяти и может использоваться в последующих инструкциях, как переменная `x` на рис. 4.7.

Инструкции в JShell не обязательно должны завершаться точкой с запятой (`;`), в отличие от инструкций в обычной программе на Java.

Для выхода из среды JShell используйте команду `/exit`.

Оболочка языка, такая как JShell, также называется *REPL* (Read-Eval-Print Loop — цикл "чтение — вычисление — вывод"). Этот термин позаимствован из языка программирования Lisp, в котором была столь любимая программистами интерактивная оболочка.

Благодаря REPL обеспечивается удобный способ экспериментирования с программными конструкциями, изучаемыми в данной книге.

Резюме

На этом занятии вы создали программу на Java, передали ей аргументы и получили доступ к программам из библиотеки классов Java.

На нескольких следующих занятиях будут и далее рассматриваться приложения, по мере того как вы будете приобретать опыт программирования на Java. Приложения проще и быстрее тестировать, поскольку для их

выполнения не требуется никакой дополнительной работы, как в случае с другими типами программ.

Это было первое занятие, на котором рассматривалось использование объектов и относящихся к ним методов в программе на Java. Подробнее объекты будут рассмотрены на занятии 10.

Вопросы и ответы

В. Должны ли все аргументы, переданные приложению Java, быть строками?

О. При запуске приложения Java хранит все аргументы в строковом виде. Если же нужно использовать один из этих аргументов в виде целого числа или другого типа данных, отличающегося от строкового типа, нужно выполнить преобразование типов. Дополнительные сведения по этой теме будут приведены на занятии 11.

Коллоквиум

Проверьте знание материала, рассмотренного на этом занятии, ответив на следующие вопросы.

Контрольные вопросы

1. Какой тип программы на Java может выполняться на мобильном устройстве?
   - **А.** Мобильные приложения.
   - **Б.** Приложения.
   - **В.** Никакой.
2. Что означает аббревиатура JVM?
   - **А.** Journal of Vacation Marketing.
   - **Б.** Jacksonville Veterans Memorial.
   - **В.** Java Virtual Machine.
3. Каким образом нужно передавать информацию приложению Java?
   - **А.** С помощью строк.
   - **Б.** С помощью аргументов.
   - **В.** С помощью функций.

Ответы

1. **А.** Мобильные приложения предназначены для выполнения на смартфонах и планшетах, в то время как приложения могут выполняться практически на любой платформе.
2. **А., Б. и В.** Провокационный вопрос! Все три варианта расшифровки аббревиатуры верны, хотя в данном случае правильный вариант — Java Virtual Machine (виртуальная машина Java). Именно этот вариант будет использоваться на следующих 20 занятиях.
3. **Б.** Приложения получают информацию в форме аргументов.

Упражнения

Чтобы лучше закрепить материал этого занятия, выполните следующие упражнения.

- На основе приложения `Root` создайте приложение `NewRoot`, в результате выполнения которого отображается квадратный корень из числа 625.
- На основе приложения `Root` создайте приложение `NewRoot`, отображающее квадратный корень числа, переданного в качестве аргумента.

Ответы на эти упражнения можно найти на сайте книги, адрес которого указан во введении.

Часть II
Основы программирования на Java

В этой части...

- Занятие 5 Сохранение и изменение информации в программе
- Занятие 6 Работа со строками
- Занятие 7 Условные инструкции
- Занятие 8 Циклы

ЗАНЯТИЕ 5
Сохранение и изменение информации в программе

На этом занятии вы узнаете...

- как создать переменную;
- как использовать различные типы переменных;
- как хранить значения в переменных;
- как применять переменные в математических выражениях;
- как присвоить значение одной переменной другой переменной;
- как увеличивать или уменьшать значение переменной.

На занятии 2 мы создавали переменную, предназначенную для хранения информации. Данные, хранящиеся в переменных, могут изменяться при выполнении программы. В вашей самой первой программе переменная содержала строку текста. Следует отметить, что информация, находящаяся в переменной, не ограничивается строками. Переменная может также хранить символы, целые числа, числа с плавающей точкой и объекты.

На этом занятии будет подробно рассмотрено использование переменных в программах на Java.

Инструкции и выражения

Компьютерные программы представляют собой наборы команд, указывающие компьютеру на то, что ему нужно делать. Каждая такая команда называется *инструкцией*. Ниже приведен пример инструкции Java.

```
int highScore = 450000;
```

Для группирования инструкций в программе на Java можно использовать фигурные скобки. Все, что находится в фигурных скобках, называется *блоком инструкций*. Рассмотрим следующий пример кода.

```
1: public static void main(String[] arguments) {
2:    int a = 3;
3:    int b = 4;
4:    int c = 8 * 5;
5: }
```

Блок инструкций находится в строках 2–4. Открывающая фигурная скобка в строке 1 обозначает начало блока, а закрывающая фигурная скобка в строке 5 — конец блока.

Некоторые инструкции называются *выражениями*, поскольку они включают математические выражения и возвращают результат. Строка 4 предыдущего примера представляет собой выражение, поскольку она устанавливает значение переменной `c`, равное числу 8, умноженному на 5.

На этом занятии будет подробно рассмотрена работа с выражениями.

Назначение типов переменным

Переменные хранят значения, используемые в программе. В программе `Saluton`, рассмотренной на занятии 2, использовалась переменная `greeting` для хранения значения `"Saluton mondo!"`. Компьютеру нужно запомнить текст, чтобы позже отобразить его в виде сообщения.

В программе на Java переменные создаются с помощью инструкции инициализации, которая должна включать следующее:

- имя переменной;
- тип данных, которые хранятся в переменной.

Переменным можно присваивать значения.

Чтобы познакомиться с различными типами переменных и способами их создания, откройте среду NetBeans и создайте пустой файл Java. Назовите содержащийся в нем класс `Variable`.

Начнем создавать программу со следующих инструкций:

```
package com.java24hours;

class Variable {
   public static void main(String[] arguments) {
      // Скоро появятся: переменные
   }
}
```

Не забудьте сохранить введенный код в файле `Variable.java`.

Целые числа и числа с плавающей точкой

На данный момент программа `Variable` состоит из блока `main()`, включающего единственную инструкцию, которая представляет собой комментарий:

```
// Скоро появятся: переменные
```

Удалите комментарий и введите вместо него инструкцию

```
int tops;
```

которая создает переменную `tops`. Переменной `tops` значение не присваивается, поэтому она представляет собой пустое пространство, отведенное для хранения данных. Ключевое слово `int` в начале инструкции помечает `tops` как переменную, которая предназначена для хранения целых чисел. Тип данных `int` позволяет хранить целые числа в диапазоне от –2,14 млрд до 2,14 млрд.

После инструкции `int tops` добавьте пустую строку, а затем введите следующую инструкцию:

```
float gradePointAverage;
```

Эта инструкция создает переменную `gradePointAverage`. Ключевое слово `float` обозначает числа с плавающей точкой. Тип данных `float` предназначен для хранения десятичных чисел, содержащих до 38 цифр. Тип данных `double` позволяет хранить десятичные числа, содержащие до 300 цифр.

Символы и строки

Поскольку до сих пор рассматривались числовые переменные, у читателей может сложиться мнение о том, что все переменные используются исключительно для хранения чисел. Но это вовсе не так. Переменные можно использовать также для хранения текста. Существуют две разновидности текста, которые можно хранить в переменных: символы и строки. Символ — это одиночная буква, число или знак пунктуации. Строка — это последовательность символов.

Следующий этап в написании программы `Variable` — создание переменных типа `char` и `String`. Добавьте две соответствующие инструкции после строки `float gradePointAverage`:

```
char key = 'C';
String productName = "Larvets";
```

В этих двух инструкциях текстовые значения заключены в разные кавычки: символьные значения — в одинарные, а строковые — в двойные.

Благодаря кавычкам предотвращается путаница между символьными или строковыми данными и названиями переменных. Рассмотрим следующий пример:

```
String productName = Larvets;
```

На первый взгляд создается впечатление, будто эта инструкция создает строковую переменную productName, которой присваивается текстовое значение Larvets. Но поскольку слово Larvets не взято в кавычки, фактически переменной productName присваивается значение переменой Larvets. (Если же в программе переменная Larvets отсутствует, то при попытке скомпилировать ее появится сообщение об ошибке.)

После объявления переменных типа char и String программа будет выглядеть так, как показано в листинге 5.1. Внесите в программу необходимые изменения и сохраните файл Variable.java.

ЛИСТИНГ 5.1. Исходный код программы Variable.java

```
 1: package com.java24hours;
 2:
 3: class Variable {
 4:    public static void main(String[] arguments) {
 5:       int tops;
 6:       float gradePointAverage;
 7:       char key = 'C';
 8:       String productName = "Larvets";
 9:    }
10: }
```

Двум последним переменным в программе Variable присваиваются начальные значения с помощью знака =, означающего операцию присваивания. Аналогичным образом можно присваивать значения любым переменным, создаваемым в программе на Java, как будет продемонстрировано на этом занятии.

ПРИМЕЧАНИЕ

Ключевые слова, обозначающие типы переменных, записываются символами нижнего регистра (int, float и char). Исключение составляют лишь строковые переменные, при создании которых используется ключевое слово String. Строковые переменные отличаются от других переменных, используемых в инструкциях. Подробнее это отличие будет рассмотрено на занятии 6.

Эту программу можно запустить на выполнение, но она не выводит никаких результатов.

Другие числовые типы переменных

В предыдущих примерах рассматривались основные типы переменных, используемые в большинстве программ на Java. В этом разделе будут рассмотрены менее распространенные типы переменных.

Один из таких типов называется `byte`. Он предназначен для хранения целых чисел в диапазоне от –128 до 127. Следующая инструкция создает переменную `escapeKey` и присваивает ей значение `27`:

```
byte escapeKey = 27;
```

Второй тип переменных, `short`, может применяться для хранения целочисленных значений, находящихся в меньшем диапазоне, чем `int`, — от –32,768 до 32,767.

```
short roomNumber = 222;
```

Последний из числовых типов переменных, `long`, применяется для хранения целочисленных значений, которые слишком велики для типа `int`. Значение, которое хранится в переменной типа `long`, может находиться в диапазоне от –9,22 квинтиллиона до 9,22 квинтиллиона. Этого достаточно, чтобы удовлетворить любые запросы.

При работе с огромными числами в Java могут возникать трудности, связанные с распознаваний значений, как в следующей инструкции:

```
long salary = 264400000000L;
```

Навскидку трудно определить, что это число в действительности равно 264,4 млрд долл. В Java можно оформлять большие числа с помощью символов подчеркивания (`'_'`), например:

```
long salary = 264_400_000_000L;
```

Символы подчеркивания игнорируются, поэтому переменная сохранит прежнее значение, просто его будет легче анализировать.

Прописная буква `'L'` в конце числа указывает на то, что это значение типа `long`. Если суффикс `'L'` пропущен, компилятор Java будет предполагать, что выбран тип данных `int`, и отобразит сообщение об ошибке "integer number too large" (целочисленное значение слишком велико).

ПРЕДУПРЕЖДЕНИЕ

Если при использовании символов подчеркивания в числе редактор исходного кода NetBeans отображает сообщение об ошибке, причина этого, скорее всего, заключается в том, что среда разработки настроена на использование устаревшей версии Java. Для устранения этой проблемы на панели **Проекты** щелкните правой кнопкой мыши на названии текущего проекта (например, `Java24`) и в контекстном меню выберите пункт **Свойства**. На экране появится диалоговое окно **Свойства проекта** с выбранной вкладкой **Исходные файлы** на панели **Категории**. Раскройте список **Формат исходного/бинарного файла**, чтобы убедиться в том, что выбрана текущая версия Java.

Тип boolean

В Java существует тип переменной boolean, который может хранить значение true или false. На первый взгляд, польза от переменной типа boolean сомнительна, если только вы не собираетесь писать программы для викторин (с вопросами, на которые нужно отвечать только "да" или "нет"). Но это не совсем так, поскольку переменные типа boolean могут использоваться для моделирования разных ситуаций в программах. Ниже приведены примеры вопросов, для ответа на которые могут применяться переменные типа boolean.

- Нажал ли пользователь клавишу?
- Закончилась ли игра?
- Не обнулился ли мой банковский счет?
- Подходят ли мне эти брюки?

Переменные типа boolean применяются для хранения ответов виде "да/нет" или "истина/ложь".

Следующая инструкция создает переменную типа boolean под названием gameOver:

```
boolean gameOver = false;
```

Эта переменная получает начальное значение false, поэтому подобная инструкция может, например, указывать на то, что игра еще не закончилась. Позднее, в случае наступления события, вызывающего завершение игры, переменной gameOver присваивается значение true.

Два возможных значения переменной типа boolean выглядят подобно строкам, но их не нужно заключать в кавычки. Рассмотрение переменных типа boolean будет продолжено на занятии 7.

ПРИМЕЧАНИЕ

Булевы числа получили свое название в честь Джорджа Буля (1815–1864). Этот английский математик изобрел булеву алгебру, которая служит фундаментом компьютерного программирования, цифровой электроники и логики.

Именование переменных

Имена переменных в Java могут начинаться с буквы, символа подчеркивания ('_') или знака доллара ($). Оставшаяся часть имени может состоять из любых букв и цифр. Переменным можно присвоить практически любое имя, но при этом нужно быть последовательным. В данном разделе описывается общепринятый способ именования переменных.

Язык Java чувствителен к регистру символов. Поэтому нужно всегда использовать один и тот же регистр в именах переменных. Например, если на переменную `gameOver` в тексте программы сослаться как `GameOver`, то при попытке компиляции программы отобразится сообщение об ошибке.

Имя переменной должно каким-то образом соответствовать ее назначению. Имя переменной должно начинаться со строчной буквы, а если оно состоит из нескольких слов, то первая буква каждого последующего слова должна быть прописной. Например, чтобы создать целочисленную переменную, предназначенную для хранения лучших результатов игры, можно воспользоваться следующей инструкцией:

```
int allTimeHighScore;
```

В имени переменной не допускается использовать знаки пунктуации или пробелы, поэтому ни одна из следующих инструкций не будет работать:

```
int all-TimeHigh Score;
int all Time High Score;
```

Если вы попытаетесь использовать эти имена в программе, NetBeans пометит ошибку предупреждающим красным значком рядом со строкой с ошибкой в окне редактора исходного кода.

ПРИМЕЧАНИЕ

В Java чувствительны к регистру символов не только имена переменных. Это относится ко всему, чему можно присвоить имя в программе, в том числе к классам, пакетам и методам.

В качестве имен переменных нельзя использовать ключевые слова, такие как `public`, `class`, `true` и `false`.

В версии Java 9 появилось еще одно ограничение: в качестве имени переменной нельзя использовать единственный символ подчеркивания (`'_'`). Ранее эта инструкция была разрешена в Java:

```
int _ = 747;
```

Но в Java 9 отобразится сообщение об ошибке, в котором будет сказано, что `'_'` является ключевым словом.

Хранение информации в переменных

Переменной можно присвоить значение либо в момент ее создания в программе, либо позже. Чтобы присвоить начальное значение переменной в момент ее создания, используйте знак равенства (=). Ниже показан пример создания переменной `pi` типа `double`, которой присваивается значение `3.14`.

```
double pi = 3.14;
```

Аналогичным образом можно присваивать начальные значения любым числовым переменным. Если же значение присваивается символьной или строковой переменной, следует заключить его в кавычки, как описывалось ранее.

Если переменные имеют один и тот же тип, то можно присвоить значение одной переменной другой переменной. Рассмотрим следующий пример.

```
int mileage = 300;
int totalMileage = mileage;
```

Сначала создается целочисленная переменная `mileage` с исходным значением `300`, а затем — целочисленная переменная `totalMileage`, имеющая то же значение, что и `mileage`. Эти переменные имеют одинаковое начальное значение `300`. На следующих занятиях будет рассмотрено, как преобразовать значение, присвоенное одной переменной, с учетом типа другой переменной.

ПРЕДУПРЕЖДЕНИЕ

Если вы создали переменную, но не присвоили ей начальное значение, то придется сделать это перед использованием переменной в другой инструкции. В противном случае в процессе компиляции программы может появиться сообщение об ошибке, гласящее о том, что переменная не была инициализирована.

Как уже говорилось, в Java имеются похожие числовые типы с разными диапазонами значений. Значения типа `int` и `long` являются целыми числами, но `long` охватывает больший диапазон возможных значений. Значения типа `float` и `double` предназначены для хранения чисел с плавающей точкой, но тип `double` может хранить большие значения.

Чтобы указать тип значения, можно добавить букву к числу, как показано в следующей инструкции:

```
float pi = 3.14F;
```

Буква `'F'` после значения `3.14` указывает на то, что это значение типа `float`. Если же буква пропущена, Java предполагает, что значение `3.14` имеет тип `double`.

Буква `'L'` применяется для обозначения целочисленных значений типа `long`, а буква `'D'` — для значений с плавающей точкой типа `double`.

В соответствии с еще одним соглашением об именовании в Java предусматривается использование всех символов верхнего регистра в именах переменных, значение которых не изменяется. Такие переменные называются *константами*. Ниже приведен пример создания четырех констант.

```
final int TOUCHDOWN = 6;
final int FIELDGOAL = 3;
```

```
final int CONVERSION = 2;
final int PAT = 1;
```

Поскольку значения констант никогда не меняются, возникает сомнение в целесообразности их использования. Ведь вместо констант можно указывать сами значения. Ключевым преимуществом констант является то, что благодаря им облегчается понимание программ.

В предыдущих четырех инструкциях названия констант записаны с помощью символов верхнего регистра. Это не является обязательным требованием в Java, но считается общепринятым соглашением среди программистов для выделения констант среди других переменных.

Все об операторах

Инструкции могут включать математические выражения, построенные на основе операторов +, -, *, / и %, с помощью которых в программах Java можно обрабатывать числа.

Например, выражение суммирования в Java использует оператор +, как показано в следующих инструкциях.

```
double weight = 205;
weight = weight + 10;
```

Во второй инструкции используется оператор + для присваивания переменной `weight` текущего значения, к которому прибавляется 10.

В выражении вычитания используется оператор -.

```
weight = weight - 15;
```

В этом выражении переменной `weight` присваивается текущее значение, из которого вычитается 15.

Операция деления обозначается символом /.

```
weight = weight / 3;
```

В этом выражении переменной `weight` присваивается текущее значение, деленное на 3.

Чтобы найти остаток от деления, используйте оператор % (это называется операцией деления по модулю). В следующей инструкции вычисляется остаток от деления 245 на 3:

```
int remainder = 245 % 3;
```

При выполнении операции умножения используется оператор *. Ниже приведен пример, в котором операция умножения является частью более сложной инструкции.

```
int total = 500 + (score * 12);
```

В выражении `score * 12` значение переменной `score` умножается на 12. Во всей инструкции осуществляется умножение значения переменной `score` на 12 с последующим добавлением к результату 500. Если значение переменной `score` равно 20, то результат вычисления всего выражения равен 740: 500 + (20 * 12).

Операции инкремента и декремента

Одна из наиболее часто выполняемых операций — изменение значения переменной на единицу. Можно увеличить значение на единицу, выполнив тем самым операцию *инкремента*, либо уменьшить значение на единицу, выполнив операцию *декремента*. Для выполнения подобных операций существуют специальные операторы.

Чтобы увеличить значение переменной на единицу, используйте оператор ++, как показано в следующей инструкции:

```
power++;
```

Чтобы уменьшить значение переменной на единицу, используйте оператор --.

```
rating--;
```

Операторы инкремента и декремента можно ставить перед именем переменной.

```
++power;
--rating;
```

Оператор, стоящий перед именем переменной, называется *префиксным*, а после имени переменной — *постфиксным*.

ПРИМЕЧАНИЕ

Немного запутались? На самом деле все не так уж и сложно. Достаточно вспомнить о приставках (префиксах) и суффиксах из школьного курса грамматики. Подобно тому, как приставка стоит перед корнем слова, префиксный оператор находится перед именем переменной. А постфиксный оператор стоит после имени переменной, подобно суффиксу, находящемуся после корня слова.

Рассмотрим следующие инструкции.

```
int x = 3;
int answer = x++ * 10;
```

Чему будет равно значение переменной `answer` после выполнения этих инструкций? Можно ожидать, что оно будет равно 40. Это значение будет получено после инкремента числа 3 и умножения полученного значения (4) на 10.

На самом же деле значение этой переменной будет равно 30, поскольку вместо префиксного оператора используется постфиксный.

В случае применения в выражении постфиксного оператора значение переменной не изменяется до тех пор, пока выражение не будет полностью вычислено. Инструкция `int answer = x++ * 10` выполняет те же действия и в том же порядке, что и следующие две инструкции.

```
int answer = x * 10;
x++;
```

В случае применения префиксного оператора все происходит немного иначе. Если этот оператор используется по отношению к переменой, находящейся в выражении, значение переменной изменяется перед вычислением выражения.

Рассмотрим следующие инструкции.

```
int x = 3;
int answer = ++x * 10;
```

В результате выполнения этих инструкций переменная `answer` становится равной 40. Применение префиксного оператора приводит к изменению значения переменной `x` до вычисления выражения. Поэтому инструкция `int answer = ++x * 10` будет выполнять те же действия и в том же порядке, что и следующие две инструкции.

```
x++;
int answer = x * 10;
```

Как видите, использование операторов `++` и `--` порой сопряжено с трудностями, поскольку они не столь просты, как можно подумать.

В принципе, операторы инкремента и декремента не обязательно использовать в программах. Аналогичные результаты можно получить с помощью операторов `+` и `-`:

```
x = x + 1;
y = y - 1;
```

Операторы инкремента и декремента весьма полезны, но не следует пренебрегать и другими классическими методами.

ПРИМЕЧАНИЕ

На занятии 1 упоминалось о том, что название языка программирования C++ — это просто шутка, смысл которой мы объясним позднее. Возможно, после знакомства с оператором инкремента `++` вы уже догадываетесь, почему в названии C++ используются два символа "плюс" вместо одного. Это связано с тем, что в C++ появились новые средства и функции, которые отсутствовали в исходном языке C. Поэтому C++ можно рассматривать как инкрементное улучшение языка C.

Приоритет операторов

При использовании выражения, состоящего из нескольких операторов, нужно знать, в каком порядке вычисляются эти операторы при обработке выражения. Рассмотрим следующие инструкции.

```
int y = 10;
x = y * 3 + 5;
```

Если не знать порядок выполнения математических операций в этих инструкциях, то невозможно предугадать значение переменной x. Это значение может быть равным как 35, так и 80, в зависимости от того, вычисляется ли сначала операция y * 3 или 3 + 5.

Ниже описан порядок выполнения математических операций в выражениях.

1. Инкремент и декремент.
2. Умножение, деление и деление с остатком.
3. Сложение и вычитание.
4. Операции сравнения.
5. Присваивание значения переменной с помощью знака равенства.

Поскольку умножение выполняется перед сложением, можно вернуться к предыдущему примеру и вычислить ответ: сначала y умножается на 3 и к результату 30 добавляется 5. В итоге переменной x присваивается значение 35.

Операции сравнения будут рассмотрены на занятии 7. Остальные необходимые сведения вам уже известны, поэтому вы сможете оценить результат выполнения следующих инструкций.

```
int x = 5;
int number = x++ * 6 + 4 * 10 / 2;
```

В данном случае переменной number присваивается значение 50.

Каким же образом компьютер вычислил эту сумму? Сначала выполняется оператор инкремента x++, в результате чего переменная x получает значение 6. Но обратите внимание на то, что здесь применяется постфиксный оператор ++, а значит, при вычислении остального выражения используется исходное значение x (5).

В результате выражение приобретает следующий вид:

```
int number = 5 * 6 + 4 * 10 / 2;
```

После этого выполняются операции умножения и деления: сначала 5 умножается на 6, затем 4 умножается на 10 и результат делится на 2 (4*10/2). В результате выражение преобразуется в следующую форму:

```
int number = 30 + 20;
```

В итоге значение переменной `number` становится равным 50.

Чтобы изменить порядок вычисления выражения, с помощью круглых скобок сгруппируйте части выражения, которые должны вычисляться в первую очередь. Например, в результате вычисления выражения `x = 5 * 3 + 2;` значение переменной `x` будет равно 17, поскольку умножение выполняется раньше сложения. А теперь рассмотрим измененную форму выражения:

```
x = 5 * (3 + 2);
```

В этом случае первым вычисляется выражение в круглых скобках, поэтому переменная получает значение 25. В выражении можно использовать произвольное количество скобок.

Выражения

Помните, как часто в школе вы возмущались, что знания математики никогда не пригодятся вам в реальной жизни? Мне жаль вас разочаровывать, но учитель был прав: математика важна при изучении программирования. Впрочем, это не повод паниковать, ведь компьютер с готовностью выполняет любые математические операции, которые вы от него просите.

Математические выражения применяются в компьютерных программах для решения следующих задач:

- изменение значения переменной;
- подсчет количества чего-либо, происходящего в программе;
- ввод математических формул в программу.

Выражения строятся на основе простых арифметических правил и создаются на основе таких операций, как сложение, вычитание, умножение, деление и деление с остатком.

Откройте среду NetBeans и создайте новый класс Java с именем `PlanetWeight`. Эта программа отслеживает вес космонавта, путешествующего по планетам Солнечной системы. В редакторе исходного кода введите текст из листинга 5.2.

ЛИСТИНГ 5.2. Исходный код программы `PlanetWeight.java`

```
1: package com.java24hours;
2:
3: class PlanetWeight {
4:    public static void main(String[] arguments) {
5:       System.out.print("Ваш вес на Земле будет ");
6:       double weight = 80;
```

```
 7:        System.out.println(weight);
 8:
 9:        System.out.print("Ваш вес на Меркурии будет ");
10:       double mercury = weight * .378;
11:       System.out.println(mercury);
12:
13:       System.out.print("Ваш вес на Луне будет ");
14:       double moon = weight * .166;
15:       System.out.println(moon);
16:
17:       System.out.print("Ваш вес на Юпитере будет ");
18:       double jupiter = weight * 2.364;
19:       System.out.println(jupiter);
20:    }
21: }
```

После завершения ввода кода сохраните файл. При этом произойдет автоматическая компиляция. Выберите команду **Выполнить⇨Выполнить файл**, чтобы запустить программу. Результат выполнения программы на панели **Вывод** показан на рис. 5.1.

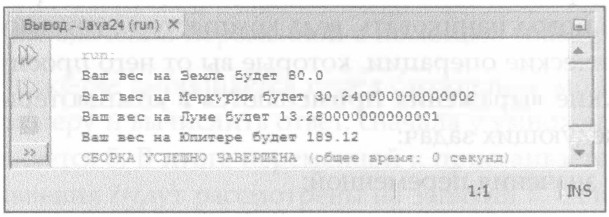


РИС. 5.1. Результат выполнения программы `PlanetWeight`

Как и в случае с другими ранее созданными программами, все действия в программе `PlanetWeight` выполняет блок инструкций `main()`. Этот блок можно разбить на следующие четыре раздела.

1. Строки 5–7. Вес человека изначально равен 80 кг.
2. Строки 9–11. Вычисляется вес человека на Меркурии.
3. Строки 13–15. Вычисляется вес человека на Луне.
4. Строки 17–19. Вычисляется вес человека на Юпитере.

В строке 6 создается переменная `weight` типа `double`, предназначенная для хранения чисел с плавающей точкой. Этой переменной присваивается начальное значение `80`, используемое в программе для отслеживания веса человека.

Рассмотрим следующую инструкцию:

```
System.out.println(weight);
```

Функция `System.out.println()` отображает строку, заключенную в круглые скобки. Команда в строке 5 отображает текст `"Ваш вес на Земле будет"`. В программе также содержатся еще несколько инструкций `System.out.print()` и `System.out.println()`.

Различия между инструкциями `print()` и `println()` заключаются в том, что первая не вставляет в конце символ новой строки, а вторая вставляет.

ПРИМЕЧАНИЕ

В качестве начального значения программа `PlanetWeight` использует 80 кг, поскольку, согласно статистическим данным, это средний вес взрослого человека.

Резюме

Теперь, когда вы ознакомились с переменными и выражениями, вы сможете давать своему компьютеру широкий набор инструкций.

Используя навыки, полученные на этом занятии, вы сможете писать программы, выполняющие многие из тех задач, на которые способен электронный калькулятор, включая сложные математические уравнения.

На этом занятии также была создана программа, вычисляющая вес человека на разных планетах Солнечной системы.

В переменных могут храниться не только числовые значения, но и отдельные символы, символьные строки и специальные значения `true` или `false`, относящиеся к типу `Boolean`. На следующем занятии будут рассмотрены переменные типа `String`.

Вопросы и ответы

В. **Правда ли, что строка и инструкция в программе Java — это одно и то же?**

О. Нет. В программах, создаваемых в данной книге, в каждой строке помещается одна инструкция, но это сделано для удобства чтения и понимания. На самом деле инструкция может занимать несколько строк.

В процессе компиляции программы компилятор Java не учитывает символы перевода строк, пробелы и другие символы форматирования. Компилятор лишь распознает точки с запятой в конце каждой инструкции.

Следующая строка будет корректно скомпилирована в Java:

```
int x = 12; x = x + 1;
```

Включение нескольких инструкций в одну строку усложняет восприятие исходного кода программы читателями, поэтому подобная практика не рекомендуется.

В. Почему в качестве первой буквы имени переменной используется символ нижнего регистра, например `gameOver`?

О. Причина — в общепринятом соглашении о наименовании переменных, которое полезно в двух отношениях. Во-первых, так облегчается отслеживание переменных среди других элементов программы на Java. Во-вторых, благодаря согласованному стилю именования переменных устраняются ошибки, которые могут возникать при использовании переменной в нескольких местах программы. Данный стиль уже много применяется большинством программистов на Java.

В. Можно ли в Java задавать целые числа в двоичном виде?

О. Можно. Для этого укажите префикс `0b` перед битовым представлением числа. Например, двоичным эквивалентом числа 13 является 1101. Следующая инструкция преобразует число 13 в двоичное представление:

```
int z = 0b0000_1101;
```

Символ подчеркивания служит лишь для улучшения наглядности кода, он игнорируется компилятором Java.

Для представления шестнадцатеричных значений следует указывать префикс `0x`, например `0x33`.

Коллоквиум

Проверьте знание материала, рассмотренного на этом занятии, ответив на следующие вопросы.

Контрольные вопросы

1. Как называется группа инструкций, которые находятся между открывающей и закрывающей фигурными скобками?
   - **А.** Блок инструкций.
   - **Б.** Групповое программное обеспечение.
   - **В.** Фигурные инструкции.
2. Переменная типа `boolean` используется для хранения значения `true` или `false`.
   - **А.** Верно.
   - **Б.** Неверно.
   - **В.** Спасибо, я уже сыт.

3. Какие символы нельзя ставить в начале имени переменной?

   А. Знак доллара.

   Б. Две косые черты (//).

   В. Букву.

Ответы

1. **А.** Сгруппированные инструкции называются блоком.
2. **А.** Переменная типа `boolean` может принимать лишь значение `true` или `false`.
3. **Б.** Имя переменной может начинаться с буквы, знака доллара (`$`) или символа подчеркивания (_). Если же имя переменной начинается двумя символами косой черты, то оставшаяся часть строки будет игнорироваться, поскольку эти символы применяются для обозначения строки комментария.

Упражнения

Чтобы лучше закрепить материал этого занятия, выполните следующие упражнения.

- Дополните программу `PlanetWeight` возможностью отслеживания веса человека на Венере (90,7% от веса на Земле) и на Уране (88,9% от веса на Земле).
- Напишите короткую программу, которая создает целочисленные переменные `x` и `y` и отображает результат в виде $x^2 + y^2$.

Ответы на эти упражнения можно найти на сайте книги, адрес которого указан во введении.

ЗАНЯТИЕ 6
Работа со строками

На этом занятии вы узнаете...

- как отображать строки в программе;
- как включать специальные символы в строку;
- как объединять строки;
- как включать переменные в строки;
- как сравнить две строки;
- как вычислить длину строки.

Любая компьютерная программа, в отличие от человека, может молча выполнять свою работу, не отвлекаясь на общение с себе подобными. Но как только программе нужно о чем-то поведать миру, она использует наиболее простой способ: строки.

В программах на Java строки применяются в качестве основного средства общения с пользователем. *Строки* представляют собой цепочки текстовых символов: букв, цифр, знаков пунктуации и др. На этом занятии будут рассмотрены способы работы со строками.

Сохранение текста в строках

В строках хранится текст, выводимый на экран. Простейшим элементом строки является *символ*, в качестве которого может выступать буква, цифра, знак пунктуации или другой символ.

Символ — это один из типов данных, которые могут храниться в переменной. Для создания символьных переменных используется тип данных `char`, как показано в следующей инструкции:

```
char keyPressed;
```

Эта инструкция создает переменную keyPressed, которая может хранить символы. При создании символьных переменных можно присваивать им начальные значения, как показано в следующем примере:

```
char quitKey = '@';
```

Значение символа нужно заключать в одинарные кавычки.

Строка — это последовательность символов. Ниже представлена инструкция, которая помещает строковое значение в переменную.

```
String fullName = "Фин Шеперд";
```

Эта инструкция создает строковую переменную fullName, включающую текст "Фин Шеперд" — имя одного из главных героев фильма "Акулий торнадо". В инструкции Java строка помечается с помощью двойных кавычек, которые сами по себе не являются частью строки.

В отличие от использованных ранее других типов переменных (int, float, char, boolean и т.п.), название типа данных String пишется с прописной буквы. В Java строки представляют собой особый вид информации, называемой *объектами*. Названия типов всех объектов записываются с прописной буквы (об этом мы поговорим на занятии 10).

Отображение строк в программах

Простейший способ отобразить строку в программе на Java — воспользоваться функцией System.out.println(), которая отображает значения, указанные в круглых скобках, на системном устройстве вывода (по умолчанию это экран компьютера). Рассмотрим пример:

```
System.out.println("Мы не можем ждать акульего дождя.");
```

Эта инструкция отображает следующий текст:

```
Мы не можем ждать акульего дождя.
```

Отображение текста на экране часто называют *печатью*. Отсюда функция println() и получила свое название (от англ. *print line* — печать строки). С помощью инструкции System.out.println() можно отображать текст, заключенный в двойные кавычки, и значения переменных. Все, что будет отображаться на экране, должно находиться в круглых скобках.

Еще один способ отображения текста на экране заключается в использовании функции System.out.print(). Она тоже отображает строки и другие переменные, находящиеся в круглых скобках, но, в отличие от функции System.out.println(), не вставляет в конце символ новой строки.

С помощью функции `System.out.print()` можно отобразить в одной строке несколько символьных строк.

```
System.out.print("В ");
System.out.print("вашем ");
System.out.print("бассейне ");
System.out.print("плавает ");
System.out.print("акула ");
System.out.print(".");
System.out.println();
```

В результате выполнения этих инструкций выводится следующий текст:

```
В вашем бассейне плавает акула.
```

Вызов функции `println()` без аргументов приводит к завершению текущей строки.

Использование специальных символов в строках

При создании или отображении строки нужно заключить ее в двойные кавычки. При этом сами символы кавычек не включаются в строку, что неизбежно вызывает следующий вопрос: как отобразить символы двойных кавычек?

Для этого нужно включить в строку специальное обозначение `\"`. Как только этот код встречается в строке, он тут же заменяется символом двойной кавычки. Рассмотрим следующий пример:

```
System.out.println("Энтони Ферранте - режиссер фильма
   \"Акулий торнадо \".");
```

Этот код отображает следующую фразу:

```
Энтони Ферранте - режиссер фильма "Акулий торнадо".
```

Аналогичным образом можно включать в строку и другие специальные символы, которые приведены в следующей таблице. Обратите внимание на то, что каждый из них предваряется обратной косой чертой (\).

Специальный символ	Что отображается
`\'`	Одинарная кавычка
`\"`	Двойная кавычка
`\\`	Обратная косая черта
`\t`	Табуляция
`\b`	Происходит возврат на одну позицию

Специальный символ	Что отображается
\r	Происходит возврат в начало строки
\f	Выполняется прогон страницы
\n	Новая строка

Задание символа новой строки приводит к тому, что следующий символ выводится в начале следующей строки. Например:

```
System.out.println("Сценарий написан\nТандером Левиным ");
```

В результате выполнения этой инструкции отображается следующий текст.

```
Сценарий написан
Тандером Левиным
```

Объединение строк

При использовании функции `System.out.println()` и вообще при работе со строками можно объединять строки с помощью оператора +, напоминающего оператор сложения.

В случае объединения строк оператор + действует иначе, чем при работе с числами. Вместо арифметической операции сложения выполняется объединение строк. В результате на основе двух меньших строк создается одна большая строка.

Операция объединения строк называется *конкатенацией*.

ПРИМЕЧАНИЕ

Отметим, что термины *конкатенация* и *объединение строк* идентичны. Поэтому можете использовать их взаимозаменяемо.

В следующей инструкции оператор + применяется для отображения длинной строки.

```
System.out.println("\"\'Акулий торнадо\' - это полтора часа вашей"
   + "жизни, которые никогда не вернутся. \nИ вы не захотите, чтобы"
   + "они вернулись.\"\n"
   + "\t-- Дэвид Хинкли, New York Daily News);
```

Для улучшения читаемости кода здесь используется оператор +, который разбивает длинные инструкции на несколько частей. В результате выполнения этой инструкции будет получен следующий текст.

```
"'Акулий торнадо' - это полтора часа вашей жизни, которые никогда
не вернутся. И вы не захотите, чтобы они вернулись."
   -- Дэвид Хинкли, New York Daily News
```

В этой инструкции используются несколько специальных символов: `\"`, `\'`, `\n` и `\t`. Чтобы лучше понять их назначение, сравните текст инструкции `System.out.println()` и результат ее выполнения.

Использование других переменных вместе со строками

Оператор + может применяться для объединения не только двух строк, но и строк с переменными. Рассмотрим следующий пример.

```
int length = 86;
char rating = 'R';
System.out.println("Время воспроизведения: " + length + " минут");
System.out.println("Рейтинг " + rating);
```

В результате выполнения этого кода отображаются следующие строки.

```
Время воспроизведения: 86 минут
Рейтинг R
```

Этот пример иллюстрирует уникальный аспект обработки строк с помощью оператора +: переменные, не являющиеся строками, могут трактоваться как строки при отображении на экране. Целочисленной переменной `length` присвоено значение `86`, которое выводится между строками `"Время воспроизведения: "` и `"минут"`. Инструкция `System.out.println()` отображает одну строку, потом выводит целочисленное значение, а затем — другую строку. Это допустимо, поскольку как минимум одна часть отображаемой группы является строкой.

При обработке строк обычно приходится включать в них другие строки, как показано в следующем примере.

```
String searchKeywords = "";
searchKeywords = searchKeywords + "акула ";
searchKeywords = searchKeywords + "ураган ";
searchKeywords = searchKeywords + "опасность";
```

В результате переменной `searchKeywords` присваивается строка `"акула ураган опасность"`. Первая инструкция создает переменную `searchKeywords` и присваивает ей пустую строку, поскольку между двумя двойными кавычками ничего нет. Вторая инструкция присваивает переменной `searchKeywords` текущую строку, к которой добавляется значение `"акула"`. В следующих двух

инструкциях аналогичным образом добавляются значения "ураган" и "опасность".

В процессе вставки дополнительного текста имя переменной отображается дважды. В Java имеется более короткий способ сделать то же самое — оператор +=. Он сочетает в себе функции операторов = и +. При работе со строками этот оператор добавляет указанные символы в конец существующей строки. Показанный выше пример может быть упрощен путем использования оператора +=, как показано в следующих инструкциях.

```
String searchKeywords = "";
searchKeywords += "акула ";
searchKeywords += "ураган ";
searchKeywords += "опасность";
```

В результате будет получен аналогичный результат: переменной searchKeywords присваивается значение "акула ураган опасность".

Дополнительные приемы обработки строк

Анализировать и изменять значение строковой переменной можно и другими способами. Эти способы основаны на объектной природе строк в языке Java и рассматриваются в следующих разделах.

Сравнение двух строк

В программах часто приходится сравнивать строки, и для этого предназначен метод equals(), как показано в следующем примере.

```
String favorite = "бензопила";
String guess = "кий для пула";
System.out.println("Любимое оружие Фина " + guess + "?");
System.out.println("Ответ: " + favorite.equals(guess));
```

В этом примере используются две строковые переменные. Одна из них, favorite, хранит название любимого оружия Фина — бензопилы. Вторая, guess, хранит предположение о том, каково любимое оружие Фина. Предполагается, что это кий для пула.

В третьей строке выводится текст "Любимое оружие Фина", после которого отображается значение переменной guess, а затем — вопросительный знак. В четвертой строке отображается текст "Ответ:", после которого выводится название оружия:

```
favorite.equals(guess)
```

Эта часть инструкции является ссылкой на *метод*, который представляет собой одну из разновидностей функции. Назначение метода equals()

заключается в установлении равенства значений двух строк. Если две строковые переменные имеют одно и то же значение, то в результате вызова метода отображается текст `true`. Если же переменные имеют разные значения, отображается текст `false`. Ниже приводится результат выполнения этого примера.

```
Любимое оружие Фина кий для пула?
Ответ: false
```

В результате вызова метода `equals()` генерируется значение типа `Boolean`, которое может быть сохранено в переменной. Рассмотрим переработанный вариант инструкции:

```
boolean checker = favorite.equals(guess);
```

Если строки `favorite` и `guess` имеют одно и то же значение, переменная `checker` получает значение `true`. В противном случае эта переменная получит значение `false`.

При работе со строками часто требуется проверять, являются ли они пустыми. Пустая строка, которая также называется нулевой строкой, не содержит никакого текста, заключенного в двойные кавычки. Соответствующая инструкция выглядит следующим образом.

```
if (favorite.equals("")) {
   System.out.println("Любимое оружие не задано");
}
```

Можно также использовать метод `equalsIgnoreCase()`, который выполняет сравнение строк без учета регистра символов. Если значение переменной `favorite` равно `"бензопила"`, а другая строковая переменная имеет значение `"бензопила"`, `"Бензопила"` или `"БЕНЗОПИЛА"`, то обе переменные будут считаться равными.

Определение длины строки

Можно также определить длину строки, выраженную в символах. Для этого применяется метод `length()`, который работает так же, как и метод `equals()`, только в данном случае используется одна строковая переменная. Рассмотрим следующий пример.

```
String cinematographer = "Бен Демари";
int nameLength = cinematographer.length();
```

Здесь целочисленной переменной `nameLength` присваивается значение 10. Метод `cinematographer.length()` подсчитывает количество символов в строковой переменной `cinematographer` и сохраняет его в качестве значения целочисленной переменной `nameLength`.

Преобразование регистра символов

Компьютеры понимают все буквально, поэтому их легко ввести в заблуждение. Человеку нетрудно понять, что строки Йен Зиринг и ЙЕН ЗИРИНГ означают одно и то же, однако компьютер не настолько понятлив. Например, рассмотренный ранее метод `equals()` авторитетно утверждает, что строка "Йен Зиринг" не совпадает со строкой "ЙЕН ЗИРИНГ".

Для решения указанных проблем Java предлагает методы, которые на основе исходного значения строковой переменной создают новое значение, содержащее все символы нижнего либо верхнего регистра (методы `toLowerCase()` и `toUpperCase()` соответственно). В следующем примере показано использование метода `toUpperCase()`.

```
String fin = "Йен Зиринг";
String change = fin.toUpperCase();
```

Этот код преобразовывает значение переменной `fin` в символы верхнего регистра (ЙЕН ЗИРИНГ), после чего присваивает его переменной `change`. Метод `toLowerCase()` работает аналогичным образом, но возвращает все символы нижнего регистра.

Отметим, что метод `toUpperCase()` не изменяет регистр символов исходной строковой переменной. В предыдущем примере значение переменной `fin` остается равным "Йен Зиринг".

Поиск подстрок

Еще одна распространенная задача при обработке строк заключается в поиске подстрок. Чтобы найти подстроку внутри строки, воспользуйтесь методом `indexOf()`. Для этого заключите искомую подстроку в круглые скобки. Если подстрока не найдена, метод `indexOf()` возвращает значение –1. В противном случае метод возвращает целочисленное значение, представляющее позицию, в которой начинается подстрока.

Номера позиций в строке начинаются с нуля, соответствующего первому символу в строке. В строке "Акулий торнадо" слово "торнадо" начинается в 7-й позиции.

Одно из возможных применений метода `indexOf()` заключается в поиске в тексте сценария фильма "Акулий торнадо" нужного фрагмента. Итак, герои фильма летят на вертолете к центру торнадо, чтобы сбросить бомбу. Вертолет начинает трясти, и Нова восклицает: "Нам нужен более устойчивый вертолет".

Если сценарий фильма "Акулий торнадо" хранится в строке `script`, выполните поиск нужной цитаты с помощью следующей инструкции:

```
int position = script.indexOf("Нам нужен более устойчивый вертолет");
```

Если в строке `script` найдена искомая подстрока, переменной `position` присваивается номер позиции, в которой начинается найденная подстрока. Если же подстрока не найдена, переменная `position` получает значение –1.

Если нужно выяснить, содержится ли подстрока в строке, однако номер позиции нас не интересует, воспользуйтесь методом `contains()`, возвращающим булево значение. Если подстрока найдена, возвращается значение `true`, если не найдена — значение `false`. Соответствующий пример приведен ниже.

```
if (script.contains("В вашем бассейне плавает акула ")) {
   int stars = 4;
}
```

ПРЕДУПРЕЖДЕНИЕ

Поскольку методы `indexOf()` и `contains()` чувствительны к регистру символов, они распознают только тот текст, регистр символов которого в точности совпадает с регистром символов искомой строки. Если же регистр символов отличается, метод `indexOf()` возвращает **–1**, а метод `contains()` — `false`.

Метод `indexOf()` может также применяться для поиска символа в строке.

Отображение титров

Чтобы применить на практике функции обработки строк, рассмотренные на этом занятии, напишем программу на Java, которая отображает титры фильма.

Вернитесь к проекту `Java24` в NetBeans, создайте новый пустой класс `Credits` и включите его в пакет `com.java24hours`. В окне редактора исходного кода введите код из листинга 6.1 и сохраните полученный файл `Credits.java`.

ЛИСТИНГ 6.1. Исходный код программы `Credits.java`

```
 1: package com.java24hours;
 2:
 3: class Credits {
 4:    public static void main(String[] arguments) {
 5:       // Информация о фильме
 6:       String title = "Акулий торнадо";
 7:       int year = 2013;
 8:       String director = "Энтони Ферранте";
 9:       String role1 = "Фин";
10:       String actor1 = "Йен Зиринг";
11:       String role2 = "Эйприл";
12:       String actor2 = "Тара Рид";
```

```
13:        String role3 = "Джордж";
14:        String actor3 = "Джон Херд";
15:        String role4 = "Нова";
16:        String actor4 = "Кэсси Сербо";
17:        // отображение информации
18:        System.out.println(title + " (" + year + ")\n" +
19:           "Режиссер" + "\t" + director + "\n" +
20:           role1 + "\t" + actor1 + "\n" +
21:           role2 + "\t" + actor2 + "\n" +
22:           role3 + "\t" + actor3 + "\n" +
23:           role4 + "\t" + actor4);
24:     }
25: }
```

Изучите код программы, чтобы понять, как она работает. Ниже приведен пошаговый анализ программы.

- В строке 3 программе присваивается имя `Credits`.
- В строке 4 начинается блок инструкций `main()`, в котором выполняется вся работа.
- В строках 6–16 переменным присваивается информация о фильме, режиссере и актерах. Одна из переменных, `year`, является целочисленной, остальные переменные — строковые.
- В строках 18–23 находится одна длинная инструкция `System.out.println()`. Все, что находится между открывающей круглой скобкой в строке 18 и закрывающей круглой скобкой в строке 23, отображается на экране. Символ новой строки (`\n`) приводит к тому, что весь последующий текст отображается с новой строки. Знак табуляции (`\t`) приводит к вставке пробелов при выводе результатов. Все остальное — это значения текстовых или символьных переменных.
- Строка 24 завершает блочную инструкцию `main()`.
- Строка 25 завершает программу.

Если при вводе кода программы `Credits` были допущены ошибки, исправьте их и сохраните код повторно. В NetBeans компиляция программы происходит в автоматическом режиме. После запуска программы будет получен результат, показанный на рис. 6.1.

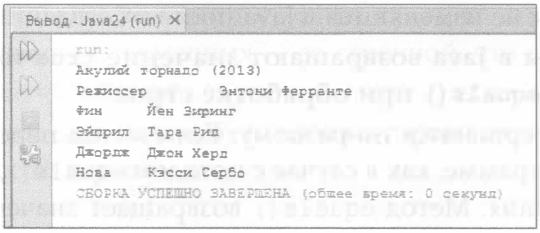


РИС. 6.1. Результат выполнения программы `Credits`

Резюме

Если результат выполнения вашей версии программы `Credits` напоминает рис. 6.1, можете поздравить себя. На шести пройденных занятиях вы создавали все более длинные и сложные программы. Именно со строками вы всякий раз имеете дело, когда беретесь писать программы.

Вопросы и ответы

В. **Каким образом можно присвоить пустое значение строковой переменной?**

О. Воспользуйтесь пустой строкой, представляющей собой пару двойных кавычек, между которыми отсутствует какой-либо текст. Следующий код создает новую строковую переменную `georgeSays` и присваивает ей пустое значение:

```
String georgeSays = "";
```

В. **У меня не получается с помощью метода `toUpperCase()` изменить строку так, чтобы она содержала только прописные буквы. Что я делаю неправильно?**

О. Если вызвать метод `toUpperCase()` для объекта `String`, то сам объект `String` не изменяется. Просто создается новая строка, состоящая из прописных букв. Рассмотрим следующие инструкции:

```
String firstName = "Баз";
String changeName = firstName.toUpperCase();
System.out.println("Имя: " + firstName);
```

Этот код отображает текст `"Имя: Баз"`, поскольку переменная `firstName` содержит исходную строку. Если же изменить последнюю инструкцию, чтобы отображалось значение переменной `changeName`, будет выведен текст `"Имя: БАЗ"`.

Значения строк не изменяются в Java после их создания.

В. Все ли методы в Java возвращают значение `true` или `false`, как это делает метод `equals()` при обработке строк?

О. Все методы завершаются по-разному. Если метод передает значение вызывающей программе, как в случае с методом `equals()`, то это называется *возвратом значения*. Метод `equals()` возвращает значение типа `boolean`. Другие методы могут возвращать строки, целые числа, другие типы переменных либо ничего из вышеперечисленного (`void`).

Коллоквиум

Проверьте знание материала, рассмотренного на этом занятии, ответив на следующие вопросы.

Контрольные вопросы

1. Что такое конкатенация?
   - **А.** Создание незаконной организации.
   - **Б.** Плетение каната.
   - **В.** Объединение строк.
2. Почему ключевое слово `String` начинается с прописной буквы, а ключевое слово `int` и названия других типов данных — со строчной?
   - **А.** Потому что `String` — это полное слово, а `int` — сокращение.
   - **Б.** Подобно всем стандартным классам Java, название класса `String` пишется с прописной буквы.
   - **В.** Из-за плохого контроля качества со стороны компании Oracle.
3. Какое из нажеперечисленных обозначений вставляет одинарную кавычку в строку?
   - **А.** `<quote>`
   - **Б.** `\'`
   - **В.** `'`

Ответы

1. **В.** Конкатенация — это термин, обозначающий операцию по объединению двух строк. Такая операция выполняется с помощью операторов + и +=, операндами которых являются строки.

2. **Б.** Названия типов объектов, доступных в Java, начинаются с прописной буквы, а названия переменных — со строчной, что позволяет не путать переменные с объектами.
3. **Б.** Для вставки специального символа в строку его следует предварить обратной косой чертой.

Упражнения

Чтобы лучше закрепить материал этого занятия, выполните следующие упражнения.

- Напишите небольшую программу на Java под названием `Favorite`, которая помещает код из раздела "Сравнение двух строк" в блок `main()`. Проверьте код, чтобы убедиться в его работоспособности. Код должен отобразить сообщение о том, что кий для пула не является любимым оружием Фина против акул. Затем измените начальное значение переменной `guess` с `"кий для пула"` на `"бензопила"`. Посмотрите, что получится.
- Измените программу `Credits` таким образом, чтобы имена режиссера и актеров отображались исключительно прописными буквами.

Ответы на эти упражнения можно найти на сайте книги, адрес которого указан во введении.

ЗАНЯТИЕ 7
Условные инструкции

На этом занятии вы узнаете...

- как использовать инструкцию `if` для проверки простых условий;
- как сравнивать значения;
- как применять инструкцию `else`;
- как объединить несколько условных проверок в цепочку;
- как использовать инструкцию `switch` для проверки сложных условий;
- как применять тернарный оператор.

В процессе создания программы компьютеру передается список команд, называемых *инструкциями*, которые выполняются беспрекословно. Можно приказать компьютеру вычислить громоздкие математические формулы или отобразить на экране какую-то информацию, и он выполнит это.

Но иногда требуется, чтобы компьютер не выполнял все, что ему прикажут, буквально, а имел определенную свободу выбора. Например, если вы создаете программу, которая будет контролировать баланс чековой книжки, желательно отображать предупреждающее сообщение, когда превышен кредит по счету. Компьютер должен выводить такое сообщение только в том случае, когда кредит действительно превышен, иначе подобное сообщение лишено смысла.

Для подобных целей в Java имеются *условные инструкции*, которые запускают те или иные действия в программе лишь в том случае, если выполняется конкретное условие. На этом занятии будет рассмотрено использование условных инструкций `if`, `else` и `switch`. Также мы изучим условные операторы `==`, `!=`, `<`, `>`, `<=`, `>=` и `?`.

Инструкция `if`

Простейший способ проверки условия в Java заключается в использовании инструкции `if`, которая выясняет, является *ли* условие истинным. Если это так, выполняется определенное действие.

Инструкция `if` задается вместе с проверяемым условием, как показано в следующем примере.

```
long account = -17_000_000_000_000L;
if (account < 0) {
   System.out.println("Превышен кредит по счету; вам нужна помощь");
}
```

Инструкция `if` проверяет, является *ли* значение переменной `account` отрицательным. При этом используется оператор "меньше чем" (<). Если условие истинно, выполняются инструкции в блоке `if`, и на экране отображается соответствующее сообщение.

Блок инструкций выполняется только в том случае, если условие истинно. Как показано в предыдущем примере, когда значение переменной `account` равно нулю или положительно, функция `println()` игнорируется. Отметим, что проверяемое условие должно быть заключено в круглые скобки, например `(account < 0)`.

Оператор "меньше чем" (<) — это один из нескольких операторов, которые можно использовать в условных инструкциях.

Операторы "меньше чем" и "больше чем"

В предыдущем разделе оператор < использовался так же, как и на уроках математики, — в качестве знака "меньше чем". Существует также условный оператор "больше чем" (>).

```
int elephantWeight = 900;
int elephantTotal = 13;
int cleaningExpense = 200;

if (elephantWeight > 780) {
   System.out.println("Слон слишком велик для перевозки");
}

if (elephantTotal > 12) {
   cleaningExpense = cleaningExpense + 150;
}
```

Первая инструкция `if` проверяет, будет *ли* значение переменной `elephantWeight` превышать 780. Вторая инструкция `if` проверяет, будет *ли* значение переменной `elephantTotal` больше 12.

Если бы значение переменной `elephantWeight` было равно 600, а значение переменной `elephantTotal` было равно 10, то инструкции в каждом блоке `if` были бы игнорируются.

С помощью оператора <= можно определить, будет ли одно значение меньше другого либо равно ему.

```
if (account <= 0) {
   System.out.println("Вы очень милы");
}
```

Для проверки условия может также использоваться оператор "больше или равно" (>=).

Операторы "равно" и "не равно"

Еще одно условие, часто проверяемое в программах, — равенство. Равно ли значение переменной указанному значению? Равно ли значение одной переменной значению другой переменной? На эти вопросы можно ответить с помощью оператора ==, как показано в следующих примерах.

```
char answer = 'b';
char rightAnswer = 'c';
int studentGrade = 85;

if (answer == rightAnswer) {
   studentGrade = studentGrade + 10;
}

if (studentGrade == 100) {
   System.out.println("Супер!");
}
```

ПРЕДУПРЕЖДЕНИЕ

Оператор, применяемый для проверки равенства, состоит из двух знаков равенства: ==. Его можно легко перепутать с оператором =, который применяется для присваивания значения переменной. Будьте внимательны!

Чтобы проверить неравенство двух значений, используйте оператор !=, как показано в следующем примере.

```
if (answer != rightAnswer) {
   score = score - 5;
}
```

Операторы == и != можно использовать с любыми типами переменных, за исключением строк (тип `String`), поскольку строки — это объекты.

Структурирование программы с помощью блоков

Рассмотренные выше инструкции `if` являются блочными, поскольку содержат фигурные скобки, `{` и `}`, обозначающие начало и конец блока.

Ранее мы уже познакомились с блоком `main()`, который является основной точкой входа в программу на Java. При выполнении программы обрабатывается каждая инструкция, находящаяся в блоке `main()`.

Инструкцию `if` не обязательно дополнять блоком, поскольку она может занимать лишь одну строку, как показано в следующем примере:

```
if (account <= 0) System.out.println("На счету денег нет");
```

Если блок отсутствует, инструкция, которая следует за условием `if`, выполняется только в случае истинности условия.

В следующем примере инструкция `if` используется для подсчета количества очков в американском футболе. В случае тачдауна к сумме очков добавляется 7, после гола с игры — 3.

```
int total = 0;
int score = 7;

if (score == 7) {
   System.out.println("Вы сделали тачдаун!");
}
if (score == 3) {
   System.out.println("Вы забили гол с игры!");
}
total = total + score;
```

Блочная инструкция `if` позволяет выполнить несколько операций в случае истинности условия. Ниже приведен пример инструкции `if`, включающей блок команд.

```
int playerScore = 12000;
int playerLives = 3;
int difficultyLevel = 10;

if (playerScore > 9999) {
   playerLives++;
   System.out.println("Дополнительная попытка!");
   difficultyLevel = difficultyLevel + 5;
}
```

Фигурные скобки применяются для группирования всех команд, которые относятся к инструкции `if`. Если значение переменной `playerScore` больше 9999, происходит следующее:

- значение переменной `playerLives` увеличивается на 1 (благодаря использованию оператора инкремента ++);
- отображается текст "`Дополнительная попытка!`";
- значение переменной `difficultyLevel` увеличивается на 5.

Если значение переменной `playerScore` не превышает 9999, то ничего не происходит: все три инструкции, находящиеся в блоке `if`, игнорируются.

Инструкция `if-else`

Зачастую возникают ситуации, когда нужно выполнить определенные действия, если условие истинно, и другие действия, если условие ложно. В этом случае можно использовать инструкцию `else` вместе с инструкцией `if`, как показано в следующем примере.

```
int answer = 17;
int correctAnswer = 13;

if (answer == correctAnswer) {
   score += 10;
   System.out.println("Все верно. Вы получили 10 очков");
} else {
   score -= 5;
   System.out.println("Извините, неправильно. Вы потеряли 5 очков");
}
```

Инструкция `else` не включает условие, в отличие от инструкции `if`. Дело в том, что инструкция `else` всегда сопоставляется с непосредственно предшествующей инструкцией `if`. С помощью инструкции `else` можно также создавать вложенные инструкции `if`, как показано ниже.

```
char grade = 'A';

if (grade == 'A') {
   System.out.println("Вы получили оценку A. Отлично!");
} else if (grade == 'B') {
   System.out.println("Вы получили оценку B. Хорошо!");
} else if (grade == 'C') {
   System.out.println("Вы получили оценку C. Удовлетворительно!");
} else {
   System.out.println("Вы получили оценку F. Вас выберут в Госдуму!");
}
```

Путем объединения инструкций `if` и `else` можно обрабатывать самые разные условия. В предыдущем примере студентам, получившим оценки A, B, C, и будущим законодателям выводятся разные сообщения.

Инструкция switch

Инструкции if и else хороши для ситуаций с двумя возможными исходами (да/нет), но бывают ситуации, когда количество условий больше двух.

В предыдущем разделе рассматривался пример объединения инструкций if и else в цепочку, чтобы обрабатывать несколько условий.

В случае большого количества условий проще использовать инструкцию switch. Ниже приведен пример из предыдущего раздела, переписанный с применением инструкции switch.

```
char grade = 'B';

switch (grade) {
   case 'A':
      System.out.println("Вы получили оценку A. Отлично!");
      break;
   case 'B':
      System.out.println("Вы получили оценку B. Хорошо!");
      break;
   case 'C':
      System.out.println("Вы получили оценку C. Удовлетворительно!");
      break;
   default:
      System.out.println("Вы получили оценку F. Вас выберут в Госдуму!");
}
```

Первая строка инструкции switch задает проверяемую переменную — здесь это grade. Далее с помощью фигурных скобок { и } формируется блок инструкций.

В каждом разделе case значение проверяемой переменной инструкции switch сравнивается с определенной константой. Это может быть символ, число или строка. В предыдущем примере в качестве тестируемых значений использовались символы 'A', 'B' и 'C'. В каждой ветви находится одна или две инструкции. Если одна из проверок дает истинный результат, выполняются все инструкции соответствующего раздела case до тех пор, пока не встретится инструкция break.

Например, если переменная grade равна 'B', отображается текст "Вы получили оценку B. Хорошо!". Далее идет инструкция break, поэтому ничего другого в блоке switch не выполняется. Инструкция break указывает компьютеру на необходимость выхода из блока switch.

Отсутствие инструкции break в разделах case инструкции switch может приводить к нежелательным результатам. Если в предыдущем примере убрать инструкции break, то независимо от того, будет ли значение

переменной grade равно 'A', 'B' или 'C', отображаются первые три сообщения "Вы получили...".

Раздел default используется в ситуации, когда ни одно из условий предыдущих разделов case не является истинным. В рассматриваемом примере эта ситуация имеет место, если значение переменной grade не равно 'A', 'B' или 'C'. Раздел default не обязателен в инструкции switch. Если он опущен и ни одна из проверок case не является истинной, ничего не произойдет.

Первый проект, рассматриваемый на этом занятии, — класс Commodity (листинг 7.1). В нем используется инструкция switch для покупки или продажи некоего товара. Цена товара составляет 20 долл. при покупке, а при продаже он приносит прибыль 15 долл.

Инструкция switch проверяет значение строки command. Если оно равно "ПОКУПКА", то выполняется один блок, а если "ПРОДАЖА" — другой.

В среде NetBeans создайте пустой класс Java с именем Commodity, включите его в пакет com.java24hours, введите код листинга 7.1 и сохраните файл Commodity.java.

ЛИСТИНГ 7.1. Исходный код программы Commodity.java

```
 1: package com.java24hours;
 2:
 3: class Commodity {
 4:    public static void main(String[] arguments) {
 5:       String command = "ПОКУПКА";
 6:       int balance = 550;
 7:       int quantity = 42;
 8:
 9:       switch (command) {
10:          case "ПОКУПКА":
11:             quantity += 5;
12:             balance -= 20;
13:             break;
14:          case "ПРОДАЖА":
15:             quantity -= 5;
16:             balance += 15;
17:       }
18:       System.out.println("Баланс: $" + balance + "\n" +
19:                          "Количество: " + quantity);
20:    }
21: }
```

В строке 5 переменной command присваивается значение "ПОКУПКА". После проверки в инструкции switch в строке 9 выполняется блок case в строках 11–13. Количество товара увеличивается на 5, а баланс уменьшается на 20 долл.

После запуска программы `Commodity` выводится результат, показанный на рис. 7.1.

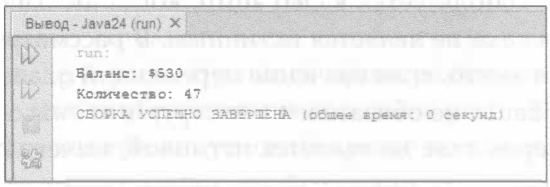


РИС. 7.1. Результат выполнения программы `Commodity`

Тернарный оператор

Наиболее сложной условной инструкцией в Java является тернарный оператор `?`. Он выполняет требуемое действие на основании значения условия. В качестве примера рассмотрим видеоигру, в которой значение переменной `numberOfEnemies` вычисляется на основе того, будет ли значение переменной `skillLevel` больше 5. Один из способов реализации такого сценария заключается в использовании инструкции `if-else`.

```
if (skillLevel > 5) {
   numberOfEnemies = 20;
} else {
   numberOfEnemies = 10;
}
```

То же самое можно сделать с помощью тернарного оператора, реализующий более короткий, но и более сложный способ. Тернарное выражение состоит из пяти частей:

- проверяемое условие, заключенное в круглые скобки, например `(skillLevel > 5)`;
- знак вопроса (`?`);
- значение, используемое при истинном условии;
- двоеточие (`:`);
- значение, используемое при ложном условии.

Чтобы применить тернарный оператор для присваивания значения переменной `numberOfEnemies` на основе значения переменной `skillLevel`, можно воспользоваться такой инструкцией:

```
int numberOfEnemies = (skillLevel > 5) ? 20 : 10;
```

Можно также использовать тернарный оператор для определения того, какую информацию отображать. Рассмотрим пример программы, которая выводит текст "Госпожа" или "Господин" в зависимости от значения переменной gender. Вот соответствующий код.

```
String gender = "женщина";
System.out.print( (gender.equals("женщина")) ? "Госпожа" :
   "Господин" );
```

Безусловно, тернарный оператор может быть весьма полезным, но он также является одной из наиболее сложных условных инструкций для начинающих. По мере изучения Java вы вряд ли встретите ситуации, когда вместо инструкции if-else лучше использовать тернарный оператор.

Часы

В этом разделе будет рассмотрен последний проект данного занятия. В нем применяется встроенная функция Java, которая отслеживает текущую дату и время.

Откройте NetBeans или другую среду разработки, предназначенную для создания программ на Java. В редакторе исходного кода введите код листинга 7.2 и сохраните файл под именем Clock.java, включив его в пакет com.java24hours. Несмотря на то что эта программа весьма длинная, бо́льшая ее часть состоит из похожих условных инструкций, состоящих из нескольких строк.

ЛИСТИНГ 7.2. Исходный код программы Clock.java

```
 1: package com.java24hours;
 2:
 3: import java.time.*;
 4: import java.time.temporal.*;
 5:
 6: class Clock {
 7:    public static void main(String[] arguments) {
 8:       // Определить текущее время и дату
 9:       LocalDateTime now = LocalDateTime.now();
10:       int hour = now.get(ChronoField.HOUR_OF_DAY);
11:       int minute = now.get(ChronoField.MINUTE_OF_HOUR);
12:       int month = now.get(ChronoField.MONTH_OF_YEAR);
13:       int day = now.get(ChronoField.DAY_OF_MONTH);
14:       int year = now.get(ChronoField.YEAR);
15:
16:       // Вывести приветствие
17:       if (hour < 12) {
18:          System.out.println("Доброе утро!\n");
```

```
19:       } else if (hour < 17) {
20:          System.out.println("Добрый день!\n");
21:       } else {
22:          System.out.println("Добрый вечер!\n");
23:       }
24:
25:       // Начало формирования строки, отображающей время
26:       System.out.print("Сейчас ");
27:
28:       // Сколько часов
29:       System.out.print(hour);
30:       System.out.print( (hour != 1 & hour != 21) ?
31:                         " часа (часов) " : " час ");
32:       // Сколько минут
33:       if (minute != 0) {
34:          System.out.print(minute);
35:          System.out.print( (minute != 1 & minute != 21 &
36:             minute != 31 & minute != 41 & minute != 51) ?
37:             " минуты (минут)" :" минута");
38:       }
39:       // Отображение числа месяца
40:       System.out.print("\n" + day +" ");
41:
42:       // Название месяца
43:       switch (month) {
44:          case 1:
45:             System.out.print("января");
46:             break;
47:          case 2:
48:             System.out.print("февраля");
49:             break;
50:          case 3:
51:             System.out.print("марта");
52:             break;
53:          case 4:
54:             System.out.print("апреля");
55:             break;
56:          case 5:
57:             System.out.print("мая");
58:             break;
59:          case 6:
60:             System.out.print("июня");
61:             break;
62:          case 7:
63:             System.out.print("июля");
64:             break;
65:          case 8:
66:             System.out.print("августа");
67:             break;
```

```
68:             case 9:
69:                System.out.print("сентября");
70:                break;
71:             case 10:
72:                System.out.print("октября");
73:                break;
74:             case 11:
75:                System.out.print("ноября");
76:                break;
77:             case 12:
78:                System.out.print("декабря");
79:          }
80:
81:          // Отображение года
82:          System.out.println(", " + year);
83:       }
84: }
```

Перед запуском программы внимательно просмотрите код, чтобы понять, как работают условные проверки.

ПРЕДУПРЕЖДЕНИЕ

Поскольку код программы рассчитан на использование Java 9 или более поздней версии, то в случае, когда текущий проект NetBeans настроен на использование более ранней версии Java, программа может не скомпилироваться. Чтобы убедиться в выборе корректной настройки, выполните команду **Файл⇨Свойства проекта** и в диалоговом окне **Свойства проекта** убедитесь в том, что в списке **Формат исходного/бинарного файла** выбрано значение **JDK 9.** или **JDK 10.**

За исключением строк 3–4 и 8–14, программа `Clock` содержит код, который был рассмотрен в предыдущих разделах. После того как ряду переменных были присвоены текущие значения даты и времени, с помощью условных инструкций `if` и `switch` определяется отображаемая информация.

В программе несколько раз вызываются функции `System.out.println()` и `System.out.print()` для отображения строк.

В строках 8–14 используется переменная типа `LocalDateTime`, которая называется `now`. Тип переменной `LocalDateTime` записывается с прописной буквы, поскольку это имя класса.

Создание объектов и работа с ними будут рассмотрены на занятии 10, а пока сосредоточимся на том, что происходит в этих строках.

Программа `Clock` состоит из следующих частей.

- В строке 3 программа подключает класс, используемый для отслеживания текущей даты и времени: `java.time.LocalDateTime`.

- В строке 4 программа подключает класс `java.time.temporal.ChronoField`.
- В строках 6 и 7 начинается код программы `Clock` и блока инструкций `main()`.
- В строке 9 создается объект `LocalDateTime` с именем `now`, который содержит значения текущей даты и времени. Объект `now` изменяется при каждом запуске программы.
- В строках 10–14 создаются переменные, предназначенные для хранения значений часов, минут, месяца, дня и года. Значения переменных берутся из объекта `LocalDateTime`, который является хранилищем этой информации. Данные, заключенные в круглые скобки, такие как `ChronoField.DAY_OF_MONTH`, указывают на то, какие значения присваиваются соответствующим переменным.
- В строках 16–23 отображается одно из трех возможных приветствий: `"Доброе утро!"`, `"Добрый день!"` или `"Добрый вечер!"`. Выбор приветствия зависит от значения переменной `hour`.
- В строках 25–31 начинает отображаться текущее время. Сначала выводится строка `"Сейчас"`. Далее с помощью тернарного оператора формируется строка с описанием текущего часа. Тернарный оператор в строках 30–31 отображает текст `"часа (часов)"` или `"час"` в зависимости от количества часов. Оператор `&` в условном выражении означает, что обе проверки (слева и справа) должны дать истинный результат, чтобы все выражение было истинным.
- В строках 32–38 отображается текущее значение минут. Если значение переменной `minute` равно 0, строки 34–37 игнорируются. Проверка значения этой переменной выполняется с помощью инструкции `if` в строке 33. Эта инструкция предотвращает вывод таких сообщений, как `"5 часов 0 минуты (минут)"`. В строке 34 выводится текущее значение переменной `minute`. Тернарная инструкция в строках 35–37 отображает текст `"минуты (минут)"` или `"минута"` в зависимости от количества минут.
- В строке 40 отображается число месяца.
- В строках 42–79, занимающих практически половину программы, находится длинная инструкция `switch`, отображающая различные названия месяцев в зависимости от целочисленного значения, которое хранится в переменной `month`.
- В строке 82 программа сообщает, какой сейчас год.
- В строках 83 и 84 закрывается блок инструкций `main()` и завершается программа `Clock`.

После запуска программы на панели вывода появляется сообщение, содержащее информацию о текущей дате и времени (рис. 7.2).

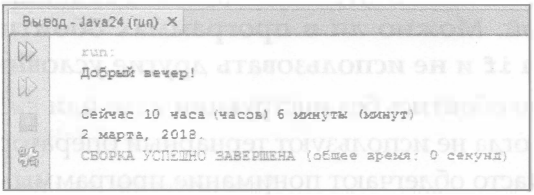


РИС. 7.2. Результат выполнения программы `Clock`

Запустите программу несколько раз, чтобы увидеть, как изменяются показания часов.

ПРИМЕЧАНИЕ

Приложение `Clock` использует библиотеку Date/Time API, которая появилась в Java 9. Для работы со значениями даты и времени в более ранних версиях Java использовался другой набор классов. Как отмечалось на занятии 4, библиотека классов Java включает тысячи классов, выполняющих различные полезные задачи. Используемые в программе пакеты `java.time` и `java.time.temporal` входят в состав в Date/Time API.

Резюме

Теперь, когда в вашем распоряжении появились условные инструкции, ваши программы на Java сильно "поумнеют". Они смогут оценивать входные данные и на основе полученных результатов по-разному реагировать в разных ситуациях, даже если данные изменяются при запуске программы. При этом осуществляется выбор среди нескольких альтернатив на основе определенных условий.

В процессе программирования приходится разбивать программу на логические наборы выполняемых действий и принимаемых решений. Использование инструкции `if` и других условных инструкций в программировании также способствует формированию логического мышления, которое пригодится и в повседневной жизни. Например:

- "если зима наступит в ноябре, я поеду в Египет";
- "если я выиграю миллион, то куплю себе дом";
- "если я закончу работу, то пойду вечером в театр".

Вопросы и ответы

В. **Создается впечатление, будто инструкция `if` является наиболее распространенной. Можно ли в программах обойтись одними лишь инструкциями `if` и не использовать другие условные инструкции?**

О. Конечно, можно обойтись без инструкции `else` или `switch`, а многие программисты никогда не используют тернарный оператор `?`. Но инструкции `else` и `switch` часто облегчают понимание программы. Однообразный набор инструкций `if` будет громоздким и трудным для понимания.

В. **На этом занятии открывающие и закрывающие фигурные скобки, `{` и `}`, не всегда используются в инструкции `if`, занимающей одну строку. Обязательно ли использовать фигурные скобки?**

О. Необязательно. Фигурные скобки могут применяться в любой инструкции `if`. Они обрамляют программный блок, выполнение которого зависит от проверки условия. Использование фигурных скобок является хорошей практикой, поскольку они предотвращают распространенные ошибки, которые могут возникать при доработке программы. Когда инструкция `if` занимает одну строку, фигурные скобки не нужны. Но что произойдет, если позже вы решите дополнить ее другими командами и при этом забудете добавить фигурные скобки? Это не сулит ничего хорошего, поэтому не забывайте о фигурных скобках.

В. **Нужно ли использовать инструкцию `break` в каждом блоке инструкций, следующих за инструкцией `case`?**

О. Инструкцию `break` использовать не обязательно. Если она не стоит в конце блока инструкций, то все оставшиеся инструкции в блоке `switch` будут выполнены, причем независимо от значения условия `case`.

Но в большинстве случаев в конце каждого блока `case` все же нужна инструкция `break`.

Коллоквиум

Проверьте знание материала, рассмотренного на этом занятии, ответив на следующие вопросы.

Контрольные вопросы

1. Условные проверки возвращают либо значение `true`, либо `false`. Какой тип переменной это напоминает?

   А. Никакой. И не задавайте мне подобные вопросы.

   Б. Тип `long`.

В. Тип `boolean`.

2. Какой раздел служит для выполнения универсальной обработки в инструкции `switch`?

   А. `default`.

   Б. `otherwise`.

   В. `onTheOtherHand`.

3. Что такое условная инструкция?

   А. Инструкция, которая может выполняться, а может не выполняться.

   Б. Часть программы, которая проверяет истинность условия.

   В. Иллюзорная инструкция, которой на самом деле нет.

Ответы

1. **В.** Лишь переменная типа `boolean` может принимать значение `true` или `false`, подобно результатам проверки условий.
2. **А.** Раздел `default` обрабатывается в том случае, если никакие другие условия `case` не соответствуют значению переменной инструкции `switch`.
3. **Б.** Все другие ответы неверны.

Упражнения

Чтобы закрепить материал этого занятия, выполните следующие упражнения.

- Превратите одну из инструкций `break` в программе `Clock` в комментарий с помощью символов `//`. Скомпилируйте программу и посмотрите, что произойдет после ее запуска. Удалите еще несколько инструкций `break` и попробуйте снова запустить программу.
- Напишите короткую программу, которая хранит значение оценки от 1 до 100 в целочисленной переменной `grade`. Используйте переменную `grade` вместе с условной инструкцией для вывода различных сообщений в зависимости от того, соответствует ли оценка уровню "отлично" (A), "хорошо" (B), "удовлетворительно" (C) и "неудовлетворительно" (F). Напишите эту программу сначала с инструкцией `if`, а затем с инструкцией `switch`.

Ответы на эти упражнения можно найти на сайте книги, адрес которого указан во введении.

ЗАНЯТИЕ 6
Циклы

На этом занятии вы узнаете...

- что такое циклы;
- как использовать цикл `while`;
- как использовать цикл `do while`;
- как преднамеренно выйти из цикла;
- как присвоить имя циклу.

Одно из наказаний для школьников заключается в том, чтобы заставлять их постоянно писать одни и те же фразы на школьной доске. Например, Барт Симпсон, главный герой мультипликационного сериала "Симпсоны", часто выходил к доске, чтобы написать: "Я перестану спрашивать, когда Санта-Клаус придет в ванную". Компьютер же по своей природе предназначен для выполнения повторяющихся действий. Компьютерная программа может с легкостью повторять одну и ту же задачу сколько угодно раз.

Благодаря циклам программы идеально приспособлены для повторения одних и тех же действий. *Цикл* — это инструкция или блок инструкций, которые повторяются при выполнении программы. Одни циклы могут выполняться определенное количество раз, другие — бесконечно.

В Java имеется три вида циклов: `for`, `do` и `while`. Каждый из них работает подобно своим "собратьям", но лучше научиться работать со всеми разновидностями циклов. Зачастую можно упростить циклический блок программы, выбрав наиболее подходящую инструкцию цикла.

Цикл `for`

В программировании часто возникают ситуации, когда без цикла не обойтись. Циклы применяются в тех случаях, когда нужно несколько раз выполнить определенные действия (например, антивирусная программа проверяет

каждое новое письмо на наличие вирусов). Можно также использовать цикл, чтобы приостановить работу компьютера на короткий период времени, например в анимированных часах, в которых минутная стрелка перемещается на одно деление каждую минуту.

Выполнение инструкции цикла приводит к тому, что компьютерная программа возвращается к одной и той же инструкции снова и снова, подобно гимнасту, который "крутит солнышко".

Наиболее сложная инструкция цикла в Java — `for`. Этот вид цикла повторяет соответствующую часть программы определенное количество раз. Рассмотрим следующий пример.

```
for (int dex = 0; dex < 1000; dex++) {
   if (dex % 12 == 0) {
      System.out.println("#: " + dex);
   }
}
```

Этот цикл выводит на экран каждое число от 0 до 999, которое без остатка делится на 12.

В цикле `for` имеется переменная, которая определяет, когда цикл должен начинаться и завершаться. Эта переменная называется *счетчиком цикла*. Роль счетчика в предыдущем примере выполняла переменная `dex`.

Цикл `for` состоит из трех частей, заключенных в круглые скобки, которые следуют за ключевым словом `for`: инициализация, условие и изменение. Эти части разделены точками с запятой (`;`).

Ниже описано назначение этих трех частей.

- Раздел инициализации. В первой части переменной `dex` присваивается начальное значение, равное нулю.
- Раздел условия. Во второй части цикла проверяется условие, подобно условию в инструкции `if`: `dex < 1000`.
- Раздел изменения. Третья часть представляет собой инструкцию, изменяющую значение переменной `dex`, в данном случае с помощью оператора инкремента.

В разделе инициализации устанавливается значение счетчика цикла. Можно создать переменную напосредственно в области действия инструкции `for`, как показано в предыдущем примере, где создается целочисленная переменная `dex`. Но можно также применить переменную, созданную в любом месте программы. В этом разделе нужно присвоить начальное значение счетчику. Переменная счетчика будет иметь это значение в начале выполнения цикла.

Во втором разделе задается условие, которое должно иметь значение `true`, чтобы цикл мог выполниться. Как только значение условия станет равным

false, выполнение цикла тут же прекратится. В данном примере цикл завершается после того, как переменная dex становится равной 1000.

В последнем разделе цикла for находится инструкция, которая изменяет значение переменной-счетчика. Эта инструкция выполняется на каждой итерации цикла. Помните, что счетчик должен каким-то образом изменяться, иначе цикл никогда не завершится. В данном примере значение переменной dex увеличивается на единицу. Если этого не сделать, переменная всегда будет равна исходному значению (нулю), а значение условного выражения dex < 1000 всегда будет равно true.

Блок инструкции for выполняется на каждой итерации цикла.

В предыдущем примере в цикле for содержались следующие инструкции:

```
if (dex % 12 == 0) {
   System.out.println("#: " + dex);
}
```

Эти инструкции выполняются 1000 раз. Выполнение цикла начинается с присваивания переменной dex нулевого значения. На каждой итерации цикла к значению переменной dex прибавляется единица. Как только значение переменной dex становится равным 1000, цикл прекращается.

ПРИМЕЧАНИЕ

При описании циклов используется термин *итерация*, под которым понимается один проход цикла. Переменная-счетчик, применяемая для управления циклом, также называется *итератором*.

Как и инструкция if, цикл for не требует фигурные скобки, если содержит единственную инструкцию.

```
for (int p = 0; p < 500; p++)
   System.out.println("Я не продаю чудодейственные средства");
```

Этот цикл 500 раз отображает фразу "Я не продаю чудодейственные средства". Даже несмотря на то что единственную инструкцию в цикле не обязательно заключать в фигурные скобки, их использование улучшит читабельность кода.

```
for (int p = 0; p < 500; p++) {
   System.out.println("Я не продаю чудодейственные средства");
}
```

Первая программа, которую мы на этом занятии, выводит первые 200 значений, кратных 9: 9, 18, 27 и т.д., вплоть до 1800 (9×200). В среде разработки NetBeans создайте новый пустой класс Java с именем Nines и включите его

в пакет `com.java24hours`. После этого введите код листинга 8.1 и сохраните файл `Nines.java`.

ЛИСТИНГ 8.1. Исходный код программы `Nines.java`

```
 1: package com.java24hours;
 2:
 3: class Nines {
 4:     public static void main(String[] arguments) {
 5:        for (int dex = 1; dex <= 200; dex++) {
 6:           int multiple = 9 * dex;
 7:           System.out.print(multiple + " ");
 8:        }
 9:        System.out.println();
10:     }
11: }
```

В строке 5 программы `Nines` находится инструкция `for`, которая включает следующие три раздела.

- Инициализация (`int dex = 1`). В этом разделе создается целочисленная переменная `dex`, которой присваивается начальное значение `1`.
- Условие (`dex <= 200`). Значение условия должно быть равно `true` на каждой итерации цикла. Как только условие перестанет быть равным `true`, цикл завершается.
- Изменение (`dex++`). Увеличение значения переменной `dex` на единицу на каждой итерации цикла.

Запустите программу, выбрав в среде NetBeans команду **Выполнить⇨ Выполнить файл**. Результат выполнения программы показан на рис. 8.1.

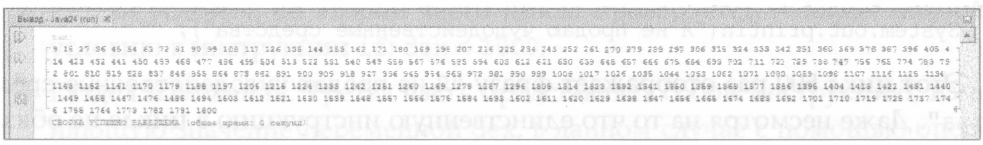


РИС. 8.1. Результат выполнения программы `Nines`

Текст в окне NetBeans не переносится, поэтому все числа выводятся в одной длинной строке. Чтобы включить перенос текста, щелкните правой кнопкой мыши в любом месте панели **Вывод** и в контекстном меню выберите команду **Перенос по словам**.

Цикл while

Структура цикла while проще, чем у цикла for. В нем требуется лишь раздел условия, как показано в следующем примере.

```
int gameLives = 3;
while (gameLives > 0) {
   // Инструкции цикла
}
```

Цикл выполняется до тех пор, пока значение переменной gameLives не станет меньше нуля.

Условие while проверяется в самом начале, до выполнения каких-либо инструкций в цикле. Если проверяемое условие будет равно false еще до первой итерации, то инструкции в цикле игнорируются.

Если условие while равно true, цикл выполняется один раз, после чего условие проверяется снова. Если проверяемое условие никогда не изменяется во время выполнения цикла, происходит зацикливание, т.е. цикл начинает выполняться бесконечно.

В следующем фрагменте цикл while используется для повторного отображения одной и той же строки текста.

```
int limit = 5;
int count = 1;
while (count < limit) {
   System.out.println("Солнце - это звезда");
   count++;
}
```

В цикле while используется несколько переменных, которым были присвоены значения до начала цикла. В данном случае это две целочисленные переменные: limit, которой присвоено значение 5, и count, значение которой равно 1.

Цикл while четыре раза выводит текст "Солнце – это звезда". Если присвоить переменной count начальное значение 6 вместо 1, текст не будет отображаться вообще.

Цикл do while

Цикл do while подобен циклу while, но проверка условия выполняется в другом месте. Рассмотрим пример.

```
int gameLives = 0;
do {
```

```
   // Здесь находятся инструкции цикла
} while (gameLives > 0);
```

Подобно циклу `while`, этот цикл выполняется до тех пор, пока значение переменной `gameLives` остается большей нуля. Отличие заключается в том, что проверка условия осуществляется после выполнения инструкций цикла.

Когда цикл достигается первый раз после запуска программы, автоматически выполняются все инструкции цикла, после чего проверяется условие цикла. Если условие равно `true`, цикл будет выполнен повторно. Если же условие равно `false`, выполнение цикла завершается. Чтобы условие цикла изменилось, инструкции, находящиеся между `do` и `while`, должны что-то сделать, иначе цикл будет выполняться бесконечно. Инструкции цикла `do while` выполняются хотя бы один раз.

В следующем фрагменте цикл `do while` отображает одну и ту же строку текста несколько раз.

```
int limit = 5;
int count = 1;
do {
   System.out.println("Жизнь хороша");
   count++;
} while (count < limit);
```

Подобно циклу `while`, в цикле `do while` используется несколько переменных, которым присваиваются значения до начала цикла.

Цикл четыре раза отображает строку `"Жизнь хороша"`. Если переменной `count` присвоить начальное значение 6 вместо 1, текст будет выведен ровно один раз, даже несмотря на то что значение переменной `count` никогда не будет меньшим значения переменной `limit`.

Инструкции в цикле `do while` выполняются как минимум один раз, даже если изначально условие цикла равно `false`. В этом и проявляется отличие цикла `do while` от цикла `while`.

Выход из цикла

Обычно выход из цикла осуществляется в том случае, когда проверяемое условие равно `false`. Это справедливо для всех трех типов циклов в Java. Но бывает и так, что нужно завершить цикл немедленно, даже если условие цикла остается равным `true`. Для этого можно воспользоваться инструкцией `break`.

```
int index = 0;
while (index <= 1000) {
   index = index + 5;
   if (index == 400) {
```

```
      break;
   }
}
```

Инструкция `break` завершает цикл, в котором она содержится.

В этом примере цикл `while` выполняется до тех пор, пока значение переменной `index` не станет больше 1000. Но из этого правила есть исключение, которое ведет к более раннему выходу из цикла: если значение переменной `index` равно 400, вызывается инструкция `break`, которая немедленно завершает цикл.

Еще одна специальная инструкция, которую можно использовать в цикле, — `continue`. Она вызывает выход из текущей итерации цикла и возврат к первой инструкции цикла. Рассмотрим следующий код.

```
int index = 0;
while (index <= 1000) {
   index = index + 5;
   if (index == 400) {
      continue;
   }
   System.out.println("Значение переменной index равно " + index);
}
```

Инструкции, находящиеся в этом цикле, обрабатываются обычным образом до тех пор, пока значение переменной `index` не станет равно 400. В этом случае инструкция `continue` вызывает обратный переход к началу цикла инструкции `while` вместо привычного выполнения инструкции `System.out.println()`. Из-за наличия инструкции `continue` следующий текст никогда не выводится:

```
Значение переменной index равно 400
```

Инструкции `break` и `continue` разрешается использовать во всех трех видах циклов.

С помощью инструкции `break` можно создать цикл, который будет выполняться в программе бесконечно, как показано в следующем примере.

```
while (true) {
   if (quitKeyPressed == true) {
      break;
   }
}
```

В этом примере предполагается, что выполнение некоего вычисления в другой части программы приводит к тому, что значение переменной `quitKeyPressed` становится равным `true`. А пока этого не произойдет, цикл

while будет выполняться бесконечно, поскольку условие цикла всегда равно true и никогда не изменяется.

Именование циклов

Подобно другим инструкциям в программах на Java, циклы могут находиться в других циклах. Ниже представлен цикл for, который вложен в цикл while.

```
int points = 0;
int target = 100;
while (target <= 100) {
   for (int i = 0; i < target; i++) {
      if (points > 50) {
         break;
      }
      points = points + i;
   }
}
```

В этом примере использование инструкции break приводит к тому, что цикл for завершается, как только значение переменной points станет больше 50. В данном случае цикл while никогда не заканчивается, поскольку значение переменной target никогда не превышает 100.

Иногда нужно прервать выполнение нескольких циклов одновременно. Для этого нужно присвоить имя внешнему циклу (в данном случае циклу while). Чтобы присвоить имя циклу, укажите это имя в строке, предшествующей началу цикла, и поставьте после него двоеточие (:).

Если циклу присвоено имя, укажите его в инструкции break или continue, чтобы задать цикл, по отношению к которому будет выполнена данная инструкция. Следующий пример повторяет предыдущий с единственным исключением: если значение переменной points больше 50, завершается выполнение обоих циклов.

```
int points = 0;
int target = 100;
targetLoop:
while (target <= 100) {
   for (int i = 0; i < target; i++) {
      if (points > 50) {
         break targetLoop;
      }
      points = points + i;
   }
}
```

Имя цикла, указываемое в инструкции `break` или `continue`, не должно включать двоеточие.

Сложные циклы `for`

Цикл `for` может быть более сложным, чем было показано до сих пор. В частности, он может включать несколько переменных-счетчиков, задаваемых в разделе инициализации и проверяемых в разделе условия, и несколько инструкций в разделе изменения, как показано в следующем примере.

```
int i, j;
for (i = 0, j = 0; i * j < 1000; i++, j += 2) {
   System.out.println(i + " * " + j + " = " + (i * j));
}
```

В каждом разделе цикла `for`, отделенном точкой с запятой (`;`), переменные разделяются с помощью запятой, например `i = 0, j = 0`. В этом примере цикла отображается список уравнений, в которых перемножаются переменные `i` и `j`. На каждой итерации цикла переменная `i` увеличивается на 1, а переменная `j` — на 2. Как только результат умножения `i` на `j` становится равным или большим 1000, выполнение цикла прекращается.

Разделы цикла `for` могут быть также пустыми. Например, переменная — счетчик цикла уже могла быть создана, и ей присвоено начальное значение в другой части программы, как показано в следующем примере.

```
int displayCount = 1;
int endValue = 13;
for ( ; displayCount < endValue; displayCount++) {
   // Инструкции цикла
}
```

Тестирование быстродействия компьютера

Следующий проект, рассматриваемый на этом занятии, — программа, выполняющая тест производительности. С помощью такого теста можно оценить быстродействие аппаратного и программного обеспечения. Программа `Benchmark` использует цикл для повторного выполнения следующего математического выражения:

```
double x = Math.sqrt(index);
```

Эта инструкция вызывает метод `Math.sqrt()`, используемый для вычисления квадратного корня числа. Дополнительные сведения о методах будут приведены на занятии 11.

С помощью теста производительности подсчитывается, сколько раз программа может вычислить квадратный корень за одну минуту.

В среде NetBeans создайте новый пустой класс Java с именем `Benchmark` и включите его в пакет `com.java24hours`. Введите код листинга 8.2 и сохраните файл `Benchmark.java`.

ЛИСТИНГ 8.2. ИСХОДНЫЙ КОД ПРОГРАММЫ `Benchmark.java`

```
 1: package com.java24hours;
 2:
 3: class Benchmark {
 4:    public static void main(String[] arguments) {
 5:       long startTime = System.currentTimeMillis();
 6:       long endTime = startTime + 60000;
 7:       long index = 0;
 8:       while (true) {
 9:          double x = Math.sqrt(index);
10:          long now = System.currentTimeMillis();
11:          if (now > endTime) {
12:             break;
13:          }
14:          index++;
15:       }
16:       System.out.println(index + " циклов за одну минуту.");
17:    }
18: }
```

При выполнении программы происходит следующее.

- Строка 5. Создается переменная `startTime`, которой присваивается текущее время, выраженное в миллисекундах. При этом используется метод `currentTimeMillis()` из библиотечного класса `System`.
- Строка 6. Создается переменная `endTime`, которой присваивается значение, на 60000 большее, чем значение переменной `startTime`. Поскольку в одной минуте содержится 60000 миллисекунд, этой переменной присваивается значение, которое на одну минуту превышает значение переменной `startTime`.
- Строка 7. Переменной `index` типа `long` присваивается начальное значение 0.
- Строка 8. Инструкция `while` начинает цикл, используя условие `true`. В результате цикл будет выполняться бесконечно (другими словами, он будет выполняться до тех пор, пока его не остановит какое-либо внешнее событие).

- Строка 9. Вычисляется квадратный корень из значения переменной `index`, который присваивается переменной `x`.
- Строка 10. С помощью метода `currentTimeMillis()` создается переменная `now`, которой присваивается значение текущего времени.
- Строки 11–13. Если значение переменной `now` больше значения переменной `endTime`, то это означает, что цикл выполнялся одну минуту, после чего инструкция `break` завершает выполнение цикла. В противном случае цикл продолжается.
- Строка 14. Значение переменной `index` увеличивается на единицу на каждой итерации цикла.
- Строка 16. После завершения цикла программа отображает количество операций по вычислению квадратного корня.

Результат выполнения приложения показан на рис. 8.2.

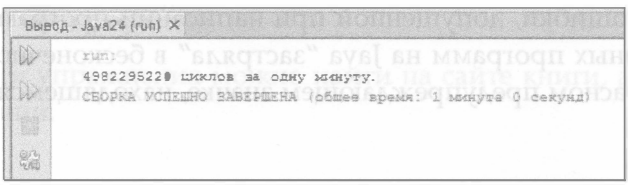


РИС. 8.2. Результат выполнения программы `Benchmark`

Программа `Benchmark` является превосходным средством оценки быстродействия компьютера. В процессе тестирования программы мой ноутбук выполнил 4,98 млрд вычислений. Если ваш компьютер покажет лучшие результаты, не спешите выражать мне соболезнования. Лучше купите больше моих книг, чтобы я мог обновить свой компьютер.

Резюме

Циклы являются фундаментальной частью большинства языков программирования. Например, одна из множества задач, которая не может быть выполнена в Java или другом языке программирования без циклов, — это анимация, создаваемая путем отображения последовательности из нескольких картинок.

Каждый поход Барта Симпсона к доске был задокументирован в Интернете. Чтобы увидеть этот список, перейдите по следующему адресу:

`http://simpsons.wikia.com/wiki/List_of_chalkboard_gags`

Вопросы и ответы

В. В главе несколько раз встретился термин "инициализация". Что он означает?

О. Этот термин означает присваивание чему-либо начального значения. Если создать переменную и присвоить ей начальное значение, то эта переменная будет инициализирована.

В. Если цикл выполняется бесконечно, то каким образом его можно остановить?

О. Обычно в программе с бесконечным циклом имеются какие-то другие инструкции, которые могут остановить выполнение цикла. Например, цикл в игровой программе может выполняться бесконечно до тех пор, пока у игрока есть жизни.

В то же время бесконечный цикл может возникать непреднамеренно, в результате ошибки, допущенной при написании программы. Если одна из запущенных программ на Java "застряла" в бесконечном цикле, щелкните на красном предупреждающем значке, находящемся слева от панели Вывод.

Коллоквиум

Проверьте знание материала, рассмотренного на этом занятии, ответив на следующие вопросы.

Контрольные вопросы

1. С помощью какого знака отделяются друг от друга разделы инструкции `for`?
   - **А.** Запятая.
   - **Б.** Точка с запятой.
   - **В.** Звездочка.
2. Какая инструкция вынуждает программу вернуться к началу цикла?
   - **А.** `continue`.
   - **Б.** `next`.
   - **В.** `skip`.
3. Какая инструкция цикла в Java всегда выполняется хотя бы один раз?
   - **А.** `for`.
   - **Б.** `while`.
   - **В.** `do while`.

Ответы

1. **Б.** Запятые применяются для разделения переменных цикла внутри раздела, а точки с запятой — для отделения разделов друг от друга.
2. **А.** Инструкция `break` полностью завершает цикл, а инструкция `continue` вызывает переход к следующей итерации цикла.
3. **В.** Условие цикла `do while` не проверяется до тех пор, пока цикл не выполнится хотя бы один раз.

Упражнения

Если вам еще не надоели циклы, выполните следующие упражнения.

- Измените программу `Benchmark` так, чтобы проверить выполнение простой математической операции, например умножения или деления.
- С помощью циклов напишите короткую программу, которая находит первые 400 чисел, кратные 13.

Ответы на эти упражнения можно найти на сайте книги, адрес которого указан во введении.

Часть III
Объекты и массивы в Java

В этой части...

- Занятие 9 Массивы
- Занятие 10 Создание объектов
- Занятие 11 Работа с объектами
- Занятие 12 Повторное использование объектов

ЗАНЯТИЕ 9
Массивы

На этом занятии вы узнаете...

- как создать массив;
- как настроить размер массива;
- как сохранить значение в массива;
- как изменить информацию в массиве;
- как создать многомерный массив;
- как отсортировать массив.

Наверное, появление компьютера принесло больше всего пользы Санта-Клаусу, ведь на протяжении столетий ему приходилось собирать и обрабатывать информацию вручную. Традиционно старину Санту интересует следующее:

- непослушные дети;
- хорошие дети;
- желаемые подарки.

Компьютеры стали настоящим благом для Санта-Клауса, ведь они идеально подходят для хранения, классификации и изучения информации.

Простейший способ сохранить информацию в компьютерной программе — занести данные в переменную. До сих пор все переменные, с которыми мы имели дело, могли хранить единственный элемент информации, например число с плавающей точкой или строку.

Список хороших детей, составленный Сантой, — пример большой коллекции однотипной информации. Для работы с подобными списками лучше использовать массивы.

Массив — это группа связанных переменных одного и того же типа. Любой тип информации, который может храниться в виде переменной, может стать элементом массива. Массивы применяются для управления более сложными

типами данных, но их почти так же легко создавать и обрабатывать, как и простую переменную.

Создание массивов

Массивы представляют собой переменные, сгруппированные под общим именем. Подобно переменным, массивы создаются путем указания типа хранимых значений и имени самого массива. За названием типа следует пара квадратных скобок `[]`, которые позволяют отличать массивы от переменных.

Массивы могут содержать любой тип информации, допустимый для переменной. Следующая инструкция создает массив строк:

```
String[] naughtyChild;
```

Ниже представлены две инструкции, которые создают массивы целочисленных и булевых значений соответственно.

```
int[] reindeerWeight;
boolean[] hostileAirTravelNations;
```

ПРИМЕЧАНИЕ

В Java квадратные скобки можно также помещать после имени переменной, как показано в следующем примере:

```
String niceChild[];
```

Чтобы облегчить отслеживание массивов в программах, следует придерживаться единого стиля. В этой книге квадратные скобки всегда ставятся после обозначения типа.

В предыдущих примерах создавались переменные массивов, но они не содержали никаких значений. Чтобы организовать хранение данных, используйте инструкцию `new`, указав после него тип значений, либо задайте список значений в фигурных скобках. В случае ключевого слова `new` нужно также указать количество ячеек в массиве, называемых *элементами*. Следующая инструкция создает массив и выделяет память для его элементов:

```
int[] elfSeniority = new int[250];
```

В этом примере создается массив целых чисел под названием `elfSeniority`. В массиве хранится 250 элементов.

При создании массива с помощью инструкции `new` требуется указывать размер массива. Каждый элемент массива получает начальное значение, которое зависит от типа массива. Все числовые элементы получают начальное значение `0`, элементы типа `char` — `'\0'`, а элементы типа `boolean` — `false`. Объекты типа `String` и любые другие объекты создаются с начальным значением `null`.

Если массивы не слишком большие, можно присваивать им начальные значенияи прямо в момент создания. В следующем примере создается массив строк, которому присвоены начальные значения:

```
String[] reindeerNames = { "Дэшер", "Дэнсер", "Прэнсер", "Виксен",
   "Комет", "Кьюпид", "Дондер", "Блитцен" };
```

Информация, которая должна храниться в элементах массива, приводится в фигурных скобках. Сами элементы разделяются запятыми. Количество элементов в массиве устанавливается равным количеству элементов списка, разделенных запятыми.

Элементы массива нумеруются с нуля (номер первого элемента). Чтобы получить доступ к определенному элементу, нужно указать его номер в квадратных скобках. Предыдущая инструкция выполняет ту же задачу, что и следующий код.

```
String[] reindeerNames = new String[8];
reindeerNames[0] = "Дэшер";
reindeerNames[1] = "Дэнсер";
reindeerNames[2] = "Прэнсер";
reindeerNames[3] = "Виксен";
reindeerNames[4] = "Комет";
reindeerNames[5] = "Кьюпид";
reindeerNames[6] = "Дондер";
reindeerNames[7] = "Блитцен";
```

Все элементы массива должны быть одного типа. Например, в предыдущем коде для хранения имен оленей Санта-Клауса используется строковый тип.

После создания массива невозможно выделить место для дополнительных элементов. Даже если вы вспомните имя самого известного оленя Санты, вы не сможете добавить `"Рудольф"` в качестве девятого элемента массива `reindeerNames`. Компилятор Java не позволит бедному Рудольфу присоединиться к дружной упряжке оленей Санты.

Работа с массивами

Массивы можно использовать в программе точно так же, как и другие переменные, но с одним исключением: номер элемента массива должен заключаться в квадратные скобки и следовать за именем массива. Элемент массива можно использовать там же, где применяется переменная, как показано в следующих инструкциях.

```
elfSeniority[193] += 1;
niceChild[9428] = "Eli";
currentNation = 413;
```

```
if (hostileAirTravelNations[currentNation] == true) {
   sendGiftByMail();
}
```

Поскольку первый элемент массива имеет номер 0, а не 1, это означает, что самый большой номер элемента массива будет на единицу меньше, чем размер массива. Рассмотрим следующую инструкцию:

```
String[] topGifts = new String[10];
```

Эта инструкция создает массив строковых переменных, которые нумеруются от 0 до 9. Если же где-то в программе указать ссылку `topGifts[10]`, будет выдано сообщение об ошибке типа `ArrayIndexOutOfBoundsException`.

Исключение — это еще один термин, применяемый для обозначения ошибок в программах на Java. Указанное исключение является ошибкой типа "array index out of bounds" (индекс выходит за пределы массива). Это означает, что программа пытается обратиться к элементу массива, который не существует в заданных границах массива. Дополнительные сведения об исключениях будут приведены на занятии 14.

Для контроля верхней границы массива с каждым массивом связывается переменная `length`, которая имеет целочисленное значение, равное размеру массива. В следующем примере показано создание массива и отображение информации о длине этого массива:

```
String[] reindeerNames = { "Дэшер", "Дэнсер", "Прэнсер", "Виксен",
              "Комет", "Кьюпид", "Дондер", "Блитцен", "Рудольф" };
System.out.println("Всего " + reindeerNames.length + " оленей.");
```

В рассматриваемом примере значение переменной `reindeerNames.length` равно 9, т.е. самый большой номер элемента будет равен 8.

С текстом в Java можно работать как со строкой или как с массивом символов. При работе со строками можно воспользоваться одной полезной методикой, суть которой заключается в том, чтобы поместить каждый символ строки в собственный элемент символьного массива. Для этого вызовите метод строки `toCharArray()`, создающий символьный массив, количество символов в котором совпадает с длиной строки.

В первом проекте данного занятия используются обе методики, рассмотренные в этом разделе. Программа `SpaceRemover` отображает строку, в которой все символы пробела заменены точками.

Откройте проект в среде NetBeans, выполните команду **Файл⇨Создать файл**, чтобы создать новый пустой класс Java с именем `SpaceRemover`, и включите его в пакет `com.java24hours`. В редакторе исходного кода введите код листинга 9.1 и сохраните полученный файл `SpaceRemover.java`.

ЛИСТИНГ 9.1. Исходный код программы SpaceRemover.java

```
 1: package com.java24hours;
 2:
 3: class SpaceRemover {
 4:    public static void main(String[] arguments) {
 5:       String mostFamous = "Рудольф - это олень с красным
 6:                            светящимся носом ";
 7:       char[] mfl = mostFamous.toCharArray();
 8:       for (int dex = 0; dex < mfl.length; dex++) {
 9:          char current = mfl[dex];
10:          if (current != ' ') {
11:             System.out.print(current);
12:          } else {
13:             System.out.print('.');
14:          }
15:       }
16:       System.out.println();
17:    }
18: }
```

С помощью команды Выполнить⇨Выполнить файл запустите программу на выполнение и посмотрите на результат (рис. 9.1).

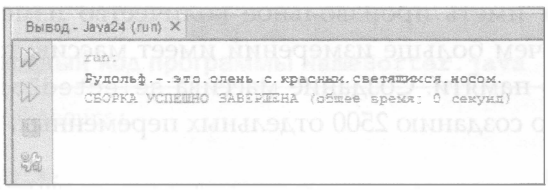


РИС. 9.1. Результат выполнения программы SpaceRemover

Приложение `SpaceRemover` сохраняет текст `"Рудольф – это олень с красным светящимся носом"` в двух местах: в строке `mostFamous` и в символьном массиве `mfl`. Массив создается в строке 7 путем вызова метода `toCharArray()` объекта `mostFamous`, который заносит в каждый элемент массива один текстовый символ. Символ `'Р'` попадает в элемент 0, символ `'у'` — в элемент 1 и т.д., а последний символ `'м'` становится элементом 46.

Цикл `for` в строках 8–15 анализирует каждый символ массива `mfl`. Если символ не является пробелом, он отображается. Если же встречается пробел, вместо него выводится символ `'.'`.

Многомерные массивы

Рассмотренные ранее массивы имели одно измерение, поэтому можно было осуществлять выборку элемента с помощью единственного индекса. Для хранения некоторых типов информации, например точек в системе координат *(x,y)*, требуются многомерные (в данном случае двумерные) массивы. Одно измерение массива может применяться для хранения координаты *x*, а второе — для хранения координаты *y*.

При работе с двумерным массивом необходимо указывать дополнительный набор квадратных скобок, как показано ниже.

```
boolean[][] selectedPoint = new boolean[50][50];
selectedPoint[4][13] = true;
selectedPoint[7][6] = true;
selectedPoint[11][22] = true;
```

В этом примере создается массив значений типа `Boolean` под названием `selectedPoint`. Он включает 50 элементов в первом измерении и 50 элементов во втором, т.е. всего 2500 отдельных элементов массива (50×50). При создании массива каждому элементу присваивается значение по умолчанию `false`. Далее трем элементам присваивается значение `true`. Точки получают значения координат *(x,y)*, равные *(4,13)*, *(7,6)* и *(11,22)*.

Массивы могут иметь произвольное количество измерений, но следует учитывать, что чем больше измерений имеет массив, тем больше нужно выделять для него памяти. Создание массива `selectedPoint` размерностью 50×50 эквивалентно созданию 2500 отдельных переменных.

Сортировка массива

Если в массиве сгруппировано множество похожих элементов, то их можно переупорядочивать. В следующем примере происходит обмен значениями двух элементов в целочисленном массиве `numbers`.

```
int[] numbers = { 3, 7, 9, 12, 5, 0, 8, 19 };
int temporary = numbers[5];
numbers[5] = numbers[6];
numbers[6] = temporary;
```

В результате элементы массива `numbers[5]` и `numbers[6]` меняются местами. Целочисленная переменная `temporary` применяется в качестве временного хранилища для обмениваемых значений. Сортировка — это процесс упорядочения списка связанных элементов определенным образом, например в порядке возрастания чисел.

Санта-Клаус мог бы использовать сортировку для упорядочения по фамилиям (от А до Я) получателей подарков.

Сортировка массива в Java упрощается благодаря тому, что вся работа по сортировке выполняется с помощью класса `Arrays`, который входит в пакет `java.util` и может переупорядочивать массивы, содержащие переменные любых типов.

Чтобы использовать класс `Arrays` в программе, выполните следующие действия.

1. С помощью инструкции `import java.util.*` подключите к программе пакет `java.util`.
2. Создайте массив.
3. Воспользуйтесь методом `sort()` класса `Arrays` для переупорядочения массива.

Массив, отсортированный с помощью класса `Arrays`, переупорядочивается по возрастанию числовых элементов. Символы и строки сортируются в алфавитном порядке.

Чтобы увидеть, как работает сортировка на практике, создайте новый пустой класс Java с именем `NameSorter` в редакторе исходного кода и включите его в пакет `com.java24hours`. Введите код листинга 9.2 и сохраните полученный файл `NameSorter.java`.

ЛИСТИНГ 9.2. Исходный код программы `NameSorter.java`

```
 1: package com.java24hours;
 2:
 3: import java.util.*;
 4:
 5: class NameSorter {
 6:    public static void main(String[] arguments) {
 7:       String names[] = { "Иван", "Семен", "Дмитрий", "Наталья",
 8:           "Ольга", "Валентина", "Татьяна", "Андрей", "Борис" };
 9:       System.out.println("До сортировки:");
10:       for (int i = 0; i < names.length; i++) {
11:          System.out.println(i + ": " + names[i]);
12:       }
13:       System.out.println();
14:       Arrays.sort(names);
15:       System.out.println("После сортировки:");
16:       for (int i = 0; i < names.length; i++) {
17:          System.out.println(i + ": " + names[i]);
18:       }
19:       System.out.println();
20:    }
21: }
```

После запуска программы отображается список из девяти имен в исходном порядке, а затем происходит сортировка имен и выводится отсортированный список имен (рис. 9.2).

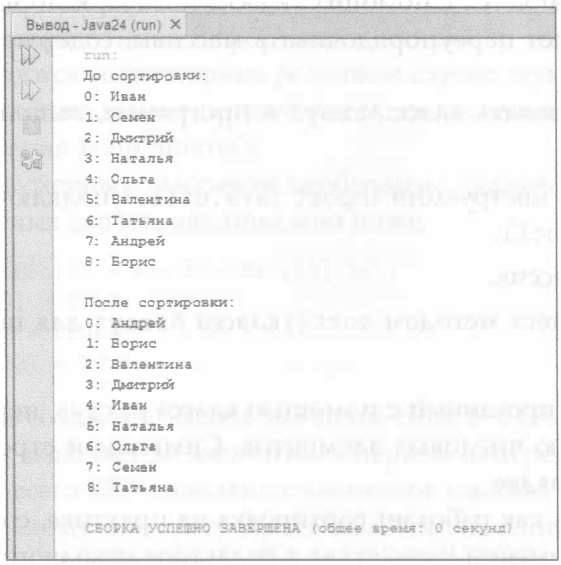


РИС. 9.2. Результат выполнения программы `NameSorter`

При работе со строками и базовыми типами переменных, такими как целые числа и числа с плавающей точкой, сортировку с помощью класса `Arrays` можно выполнять только по возрастанию. Если требуется сортировка другого рода, нужно написать собственный код.

Подсчет количества символов в строке

Следующая программа на Java подсчитывает частоту появления букв в различных фразах и выражениях. Для подсчета числа появлений каждой буквы будет применяться массив.

В среде NetBeans создайте новый пустой файл Java, присвойте ему имя `Wheel.java` и включите его в пакет `com.java24hours`. Добавьте в этот файл код листинга 9.3. Между строками 6 и 20 можете вставлять свои собственные фразы.

ЛИСТИНГ 9.3. Исходный код программы `Wheel.java`

```
1: package com.java24hours;
2:
3: class Wheel {
4:    public static void main(String[] arguments) {
5:       String phrase[] = {
```

```
 6:          "ВСЫПАТЬ ПО ПЕРВОЕ ЧИСЛО",
 7:          "ГНАТЬСЯ ЗА ДЛИННЫМ РУБЛЕМ",
 8:          "В ЧАС ПО ЧАЙНОЙ ЛОЖКЕ",
 9:          "КАЖДЫЙ КУЛИК СВОЕ БОЛОТО ХВАЛИТ",
10:          "ЗА ТРИДЕВЯТЬ ЗЕМЕЛЬ",
11:          "ЗА ДУШОЙ НИЧЕГО НЕТ",
12:          "СЕМЬ ПЯДЕЙ ВО ЛБУ",
13:          "ЗАРУБИТЬ НА НОСУ",
14:          "ПРИНЦ НА БЕЛОМ КОНЕ",
15:          "ДОМА И СТЕНЫ ПОМОГАЮТ",
16:          "ОТ ЗАБОРА ДО ОБЕДА",
17:          "ВЕРНЕМСЯ К НАШИМ БАРАНАМ",
18:          "ВНЕСТИ СВОЮ ЛЕПТУ",
19:          "НЕ МЫТЬЕМ, ТАК КАТАНЬЕМ",
20:          "ЭТОТ ДИВНЫЙ НОВЫЙ МИР"
21:       };
22:       int[] letterCount = new int[32];
23:       for (int count = 0; count < phrase.length; count++) {
24:          String current = phrase[count];
25:          char[] letters = current.toCharArray();
26:          for (int count2 = 0; count2 < letters.length; count2++) {
27:             char lett = letters[count2];
28:             if ( (lett >= 'А') & (lett <= 'Я') ) {
29:                letterCount[lett - 'А']++;
30:             }
31:          }
32:       }
33:       for (char count = 'А'; count <= 'Я'; count++) {
34:          System.out.print(count + ": " +
35:                           letterCount[count - 'А'] +
36:                           " ");
37:          if (count == 'М') {
38:             System.out.println();
39:          }
40:       }
41:       System.out.println();
42:    }
43: }
```

После запуска программы будет получен результат, показанный на рис. 9.3.

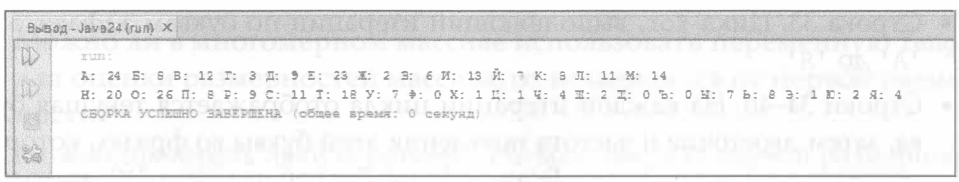


РИС. 9.3. Результат выполнения программы `Wheel`

В программе Wheel выполняются следующие действия.

- Строки 5–21. Перечислены фразы, которые хранятся в строковом массиве phrase.
- Строка 22. Целочисленный массив letterCount содержит 32 элемента и применяется для хранения частоты появления каждой буквы. Элементы следуют в порядке от 'А' до 'Я'. Элемент letterCount[0] хранит частоту появления буквы 'А', элемент letterCount[1] — частоту появления буквы 'Б' и т.д., вплоть до элемента letterCount[31], который содержит частоту появления буквы 'Я'.
- Строка 23. Цикл for, выполняющий итерации по фразам, которые хранятся в массиве phrase. Переменная phrase.length применяется для завершения цикла по достижении последней фразы.
- Строка 24. Строковой переменной current присваивается значение текущего элемента массива phrase.
- Строка 25. Символьный массив, в котором хранятся все символы текущей фразы.
- Строка 26. Цикл for, выполняющий обход всех букв текущей фразы. Переменная letters.length применяется для завершения цикла по достижении последней буквы.
- Строка 27. Символьной переменной lett присваивается значение текущей буквы. Помимо текстового значения, символы имеют числовое значение. Поскольку элементы массива пронумерованы, числовое значение каждого символа используется для определения номера соответствующего элемента.
- Строки 28–30. Инструкция if исключает все символы, которые не являются алфавитными, такие как знаки пунктуации и пробелы. Элемент массива letterCount увеличивается на 1 в зависимости от числового значения текущего символа, которое хранится в переменной lett. Числовые значения (коды ASCII) символов алфавита находятся в диапазоне от 192 (буква 'А') до 223 (буква 'Я'). Поскольку массив letterCount начинается с элемента 0 и завершается элементом 32, 'А' (192) вычитается из lett для определения элемента массива, который будет увеличиваться.
- Строка 33. Цикл for, выполняющий итерации по буквам алфавита от 'А' до 'Я'.
- Строки 34–40. На каждой итерации цикла отображается текущая буква, затем двоеточие и частота появления этой буквы во фразах, которые хранятся в массиве phrase. Если текущей буквой является 'М', выводится символ новой строки, который разбивает вывод на две строки.

Этот проект демонстрирует, каким образом два вложенных цикла могут использоваться для последовательного посимвольного просмотра набора фраз. В Java с каждым символом связано числовое значение, с которым проще работать, чем с самим символом, находящимся в массиве.

ПРИМЕЧАНИЕ

Числовые значения, присущие каждому символу от `'A'` до `'Я'`, — это значения из таблицы ASCII. Набор символов ASCII является частью Unicode, полного набора символов, поддерживаемого языком Java. Стандарт Unicode включает более 60000 разных символов, используемых в письменных языках мира. В ASCII же поддерживается лишь 256 символов.

Резюме

Благодаря массивам в программе можно хранить и обрабатывать сложные типы информации. Массивы идеально подходят для всего, что может быть упорядочено в виде списка. С массивами можно легко работать с помощью инструкций циклов, которые были рассмотрены на занятии 8.

Конечно, потребности Санта-Клауса по обработке информации могут существенно превышать скромные возможности массивов. Количество детей в мире каждый год стремительно растет, соответственно увеличивается количество подарков, которые они хотят получить на Новый год и Рождество.

Массивы можно использовать для хранения информации, если она слишком сложна для обработки с помощью переменных.

Вопросы и ответы

В. **Является ли диапазон ASCII-кодов, соответствующий латинскому алфавиту (от 65 для буквы 'A' до 90 для буквы 'Z'), частью базового языка Java? И если это так, то для чего зарезервированы коды ASCII от 1 до 64?**

О. Коды от 1 до 64 включают числа, знаки пунктуации и некоторые непечатаемые символы, такие как символы прокрутки страницы, перевода строки и возврата назад. В Java применяется система нумерации Unicode. Первые 127 символов относятся к набору символов ASCII и могут применяться в различных языках программирования.

В. **Можно ли в многомерном массиве использовать переменную `length` для оценки размерностей массива, отличающихся от первой размерности?**

О. Можно проверять любую размерность массива. Для первой размерности используйте переменную `length` вместе с названием массива, например `x.length`. Другие размерности можно оценить с помощью переменной

length вместе с элементом [0], соответствующим данной размерности. Рассмотрим массив data, который создан с помощью следующей инструкции:

```
int[][][] data = new int[12][13][14];
```

Размерности этого массива можно оценить с помощью переменной data.length (для первой размерности), data[0].length (для второй размерности) и data[0][0].length (для третьей размерности).

Коллоквиум

Проверьте знание материала, рассмотренного на этом занятии, ответив на следующие вопросы.

Контрольные вопросы

1. Для каких типов информации лучше всего подходят массивы?
   - **А.** Для списков.
   - **Б.** Для пар связанных данных.
   - **В.** Ни для каких.
2. Какую переменную можно использовать для проверки верхней границы массива?
   - **А.** top.
   - **Б.** length.
   - **В.** limit.
3. Сколько оленей у Санта-Клауса (вместе с Рудольфом)?
   - **А.** 8.
   - **Б.** 9.
   - **В.** 10.

Ответы

1. **А.** Массивы лучше всего подходят для хранения списков, которые могут содержать строки, числа и другую подобную информацию.
2. **Б.** Переменная length содержит информацию о количестве элементов массива.
3. **Б.** Согласно легенде, у Санта-Клауса было восемь маленьких северных оленей, соответственно олень Рудольф — девятый по счету.

Упражнения

Чтобы углубить познания в области массивов, выполните следующие упражнения.

- Создайте программу, которая использует многомерный массив для хранения сведений об успеваемости студентов. Первая размерность массива предназначена для хранения порядкового номера каждого студента, а вторая — для оценок студента. Отобразите среднее значение по всем оценкам, полученным каждым студентом, и общее среднее значение по всем оценкам, полученным студентами.
- Напишите программу, которая хранит в массиве первые 400 чисел, которые кратны 13.

ЗАНЯТИЕ 10
Создание объектов

На этом занятии вы узнаете...

- как создавать объекты;
- как описать объект с помощью атрибутов;
- как определять поведение объектов;
- как наследовать объекты;
- как преобразовывать объекты и другие переменные.

В книге вам не раз встретится термин *объектно-ориентированное программирование* (ООП), который описывает принципы построения компьютерной программы.

До появления ООП компьютерная программа обычно описывалась простейшим способом — в виде набора хранящихся в файле инструкций, обрабатываемых в жестко заданном порядке.

Рассматривая программу в качестве совокупности объектов, вы определяете выполняемые ею задачи и назначаете эти задачи соответствующим объектам.

Основы объектно-ориентированного программирования

Создаваемые программы на Java можно представлять себе как объекты, напоминающие физические объекты в реальном мире. Объекты существуют независимо друг от друга, могут взаимодействовать определенным образом и способны объединяться с другими объектами, формируя нечто большее. Если исходить из предположения, что компьютерная программа является группой взаимодействующих объектов, то можно создать программу, которая будет более надежной, проще в понимании и будет повторно использоваться в других проектах.

На занятии 22 нам предстоит создать программу на Java, отображающую круговые диаграммы, — круги с цветными секторами, которые представляют данные. Пример такой диаграммы приведен на рис. 10.1.

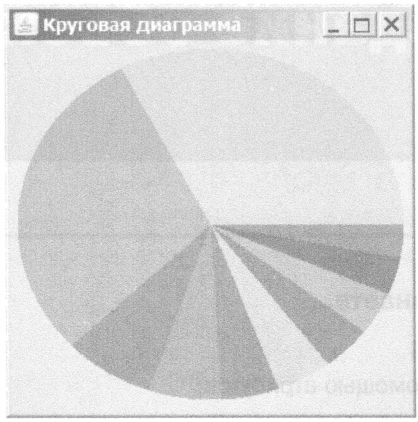


РИС. 10.1. Круговая диаграмма, созданная с помощью программы на Java

Каждый объект обладает характеристиками, отличающими его от других объектов. Круговые диаграммы имеют круглую форму, в то время как гистограммы представляют данные в виде ряда прямоугольников. Круговая диаграмма — это объект, который состоит из меньших по размерам объектов: отдельных разноцветных фрагментов, легенды, которая описывает каждый фрагмент, и заголовка. Если заняться разбиением компьютерных программ аналогично разбиению круговой диаграммы, то, по сути, вы будете заниматься ООП.

В ООП объект обладает атрибутами и поведением. *Атрибуты* описывают объект и показывают, чем он отличается от других объектов. *Поведение* описывает действия объекта.

Для создания объектов Java в качестве шаблона используется класс. *Класс* — это своего рода мастер-копия объекта, определяющая его атрибуты и поведение. Термин "класс" должен быть вам знаком, поскольку программы на Java называются классами. Каждая программа, создаваемая в Java, — это класс, который можно использовать в качестве шаблона для создания новых объектов. Например, любая программа на Java, применяющая строки, использует объекты, созданные на основе класса `String`. Этот класс содержит атрибуты, описывающие строку, и поведение, поддерживаемое объектами типа `String`.

В ООП компьютерная программа — это набор объектов, которые совместно работают, чтобы сделать что-то полезное. В случае простых программ может сложиться впечатление, будто они состоят только из одного

объекта — файла класса. Но даже такие программы используют в своей работе другие объекты.

Объекты в действии

Рассмотрим программу, которая отображает круговую диаграмму. Объект `PieChart` включает следующее:

- поведение, вычисляющее размер каждого сектора;
- поведение, создающее диаграмму;
- атрибут, в котором хранится заголовок диаграммы.

Второй пункт может вызвать недоуменный вопрос: каким образом диаграмма может нарисовать саму себя? На самом деле в ООП это вполне естественно. Благодаря такой возможности облегчается включение объектов в другие программы. Если бы, например, объект `PieChart` не знал, как рисовать самого себя, то всякий раз при использовании этого объекта в другой программе вам пришлось бы самостоятельно создавать соответствующее поведение.

В качестве еще одного примера использования ООП рассмотрим программу автодозвона, которую применял герой Мэтью Бродерика для взлома компьютеров в классическом хакерском фильме "Военные игры".

ПРИМЕЧАНИЕ

Программа автодозвона использует модем для обзвона последовательности телефонных номеров. Назначение такой программы — найти другие компьютеры, которые отвечают на телефонные звонки, чтобы обзвонить их позднее.

Читателям младше 30 лет трудно представить, что когда-то компьютеры общались между собой с помощью телефонных звонков. В те времена для подключения к компьютеру нужно было знать его телефонный номер, а если компьютер уже общался с другим компьютером, то вы слышали сигнал "занято".

В настоящее время использование программы автодозвона может вызвать вопросы у вашей телефонной компании и органов правопорядка. Но в 1980-х годах это был хороший способ взлома какого-либо компьютера, не выходя из дома. Дэвид Лайтман (герой фильма "Военные игры", роль которого талантливо сыграл Бродерик) использовал свой автодозвонщик для доступа к закрытой компьютерной системе компании по разработке видеоигр. Это позволило ему сыграть в новую игру еще до ее официального выпуска. Совершенно случайно Лайтман подключился к секретному правительственному компьютеру, который позволял играть как в безобидные шахматы, так и начать глобальную ядерную войну.

Программу автодозвона, как и любую другую компьютерную программу, можно рассматривать как набор совместно работающих объектов.

- Объект `Modem`, который знает свои атрибуты, например скорость подключения. Этот объект также имеет поведение, которое позволяет модему дозваниваться по указанному номеру и получать отклик другого компьютера на вызов.
- Объект `Monitor`, который отслеживает вызываемые номера и определяет, какие из них принадлежат компьютерам.

Каждый объект существует независимо от других объектов. Благодаря тому, что объект `Modem` абсолютно независим от других объектов, он может применяться в других программах, в которых требуются возможности модема.

Еще одна причина использования автономных объектов — упрощение отладки. Компьютерные программы быстро разрастаются настолько, что отслеживать связи в них становится проблематично. Если же осуществляется отладка объекта `Modem`, который не зависит от других объектов, то можно сконцентрироваться на проверке того, что он выполняет требуемые от него действия и располагает необходимой информацией.

Объектно-ориентированный язык типа Java удобно изучать в качестве первого языка программирования, так как вам не придется избавляться от вредных привычек, характерных для других стилей программирования.

Структура объекта

При создании объектов в качестве шаблона используется класс. Класс объявляется следующим образом.

```
public class Modem {
}
```

Объект, созданный на основе такого класса, не делает ничего полезного, поскольку не имеет ни атрибутов, ни поведения. Поэтому нужно добавить их в класс.

```
public class Modem {
   int speed;

   public void displaySpeed() {
      System.out.println("Быстродействие: " + speed);
   }
}
```

Теперь класс `Modem` начинает напоминать классы, созданные на занятиях 1–9. Код класса `Modem` начинается инструкцией `class`, которая включает

ключевое слово public. Это означает, что класс является общедоступным и может применяться любой программой, в которой требуются объекты типа Modem.

Первая часть определения класса Modem включает объявление целочисленной переменной speed, которая является атрибутом объекта.

Вторая часть определения класса содержит описание метода displaySpeed(), который является частью поведения объекта и включает одну инструкцию, System.out.println(), отображающую сведения о быстродействии модема.

Переменные, создаваемые в классе, называются *переменными экземпляра*, или *переменными-членами*.

Чтобы использовать объект Modem в программе, нужно создать объект с помощью инструкции

```
Modem device = new Modem();
```

которая создает объект device класса Modem. После создания объекта можно присваивать значения его переменным и вызывать его методы. Чтобы присвоить значение переменной speed объекта device, воспользуйтесь следующей инструкцией:

```
device.speed = 28800;
```

Для отображения сведений о быстродействии модема нужно вызвать метод displaySpeed().

```
device.displaySpeed();
```

В ответ на это объект device отобразит текст "Быстродействие: 28800".

Наследование

Одно из ключевых преимуществ ООП — механизм *наследования*, который позволяет одному объекту наследовать поведение и атрибуты другого объекта.

В процессе создания объектов нередко обнаруживается, что новый объект во многом подобен ранее созданным.

Предположим, что Дэвид Лайтман хочет создать объект, который мог бы выполнять коррекцию ошибок и реализовывать другие функции модема, которых попросту не существовало в 1983 году, когда на экраны вышел фильм "Военные игры". Лайтман вполне мог бы создать новый класс ErrorCorrectionModem путем копирования инструкций из класса Modem и последующей их модификации. Но если поведение и атрибуты объекта ErrorCorrectionModem в основном совпадают с таковыми у объекта Modem, то нет смысла выполнять столько лишней работы. К тому же в распоряжении

Лайтмана оказались бы две программы, каждую из которых впоследствии пришлось бы обновлять, если бы потребовалось вносить какие-то изменения.

С помощью наследования можно создать новый класс объектов, указав лишь отличия этого класса от предыдущего. На основе такой методики Лайтман мог быть создать класс `ErrorCorrectionModem`, унаследованный от класса `Modem`. Ему достаточно было бы добавить описания функций, присущих модемам с коррекцией ошибок.

Класс объектов наследуется от другого класса с помощью ключевого слова `extends`. Ниже показан каркас класса `ErrorCorrectionModem`, наследуемого от класса `Modem`.

```
public class ErrorCorrectionModem extends Modem {
   // Код класса
}
```

Создание иерархии наследования

Благодаря наследованию можно без особых усилий создавать различные связанные классы. При этом можно передавать код сверху вниз от одного класса другому. Подобное группирование классов "предок–потомок" называется *иерархией классов*. Все стандартные классы, которые можно использовать в программах на Java, являются частью этой иерархии.

Понять структуру иерархии будет гораздо проще, если разобраться с понятиями подклассов и суперклассов. Класс, который наследуется из другого класса, называется *подклассом*, а класс, от которого наследуется другой класс, — *суперклассом*.

В примере с фильмом "Военные игры" класс `Modem` является суперклассом для класса `ErrorCorrectionModem`, а класс `ErrorCorrectionModem` — подклассом для класса `Modem`.

От одного класса в иерархии могут наследоваться несколько классов. Например, от класса `Modem` может наследоваться класс `ISDNModem`, поскольку поведение и атрибуты модемов ISDN могут отличаться от поведения и атрибутов модемов с коррекцией ошибок. Если бы существовал подкласс класса `ErrorCorrectionModem`, например `InternalErrorCorrectionModem`, то он наследовался бы от всех классов, которые находятся выше его в иерархии (классы `ErrorCorrectionModem` и `Modem`). Эти отношения наследования проиллюстрированы на рис. 10.2.

Классы, составляющие язык программирования Java, широко используют наследование, поэтому столь важно понимать суть этого понятия. Подробнее наследование будет рассмотрено на занятии 12.

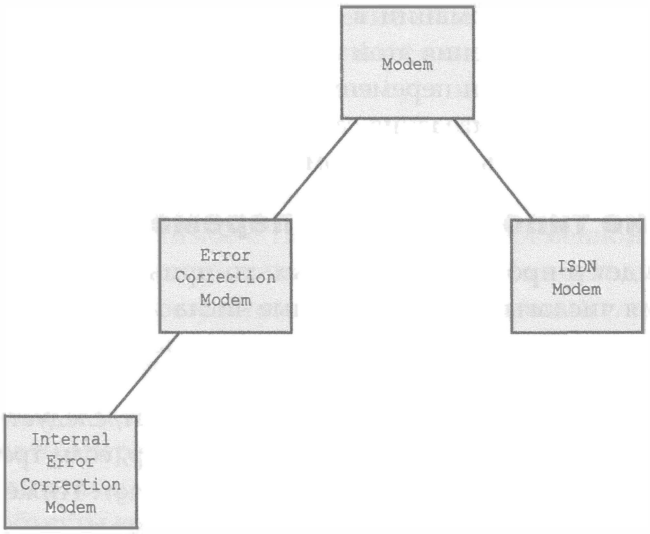


РИС. 10.2. Пример иерархии классов

Преобразование объектов и простых переменных

Одна из наиболее часто выполняемых операций в Java — преобразование информации из одной формы в другую. Ниже приведены примеры некоторых преобразований:

- преобразование объекта в другой объект;
- приведение простой переменной к другому типу;
- использование объекта для создания простой переменной;
- использование простой переменной для создания объекта.

Простые переменные — это переменные базовых типов (см. занятие 5), таких как `int`, `float`, `char`, `long`, `double`, `byte` и `short`.

Когда в программе встречается метод или выражение, в них нужно использовать данные корректного типа. Например, метод, который ожидает объект `Calendar`, должен получать именно такой объект. Если методу, который имеет единственный целочисленный аргумент, будет передано число с плавающей точкой, то при попытке скомпилировать программу будет выдано сообщение об ошибке.

ПРИМЕЧАНИЕ

Если методу требуется строковый аргумент, то с помощью оператора + в этом аргументе можно скомбинировать несколько разных типов информации. В результате будет сформирована строка, используемая в качестве аргумента.

Преобразование информации в новую форму называется *приведением типа*. В результате выполнения этой операции формируется новое значение, которое имеет не такой тип переменной или объекта, как исходные данные. Фактически в процессе приведения само значение не изменяется, а создается новая переменная или объект в нужном формате.

Приведение типов простых переменных

Если речь идет о простых переменных, то приведение типов чаще всего используют для числовых значений (целые числа и числа с плавающей точкой). Имейте в виду, что приведение типов невозможно в случае булевых переменных.

Чтобы преобразовать переменную в новый формат, следует предварить ее названием нового типа в круглых скобках. Например, если требуется выполнить приведение к типу `long`, то нужно указать `(long)`. Ниже показано, как преобразовать значение типа `float` в значение типа `int`:

```
float source = 7.00F;
int destination = (int) source;
```

Если целевая переменная может хранить больший диапазон значений, чем исходная, то преобразование значений происходит без особого труда. Например, тип данных `byte` легко приводится к типу данных `int`. Тип данных `byte` охватывает диапазон значений от –128 до 127, тогда как тип данных `int` — от –2,1 млрд до 2,1 млрд. Независимо от того, какое значение содержит переменная типа `byte`, новая переменная типа `int` предоставляет достаточно места для хранения этого значения.

Иногда можно использовать переменную в другом формате, не выполняя приведение типов. Например, с переменными типа `char` можно работать так, как если бы они были переменными типа `int`. Также можно использовать переменные типа `int` вместо переменных типа `long`, а любой числовой тип данных может использоваться вместо типа `double`.

Когда целевая переменная обеспечивает больше места для хранения информации, чем исходная, данные преобразуются без изменения значений. Исключения из этого правила имеют место в том случае, когда переменные типа `int` или `long` приводятся к типу `float` либо переменная типа `long` приводится к типу `double`.

Если исходная переменная обеспечивает хранение большего диапазона, чем целевая, то нужно обязательно выполнять явное приведение типов, как показано ниже.

```
int xNum = 103;
byte val = (byte) xNum;
```

В данном случае целочисленное значение xNum преобразуется в переменную val типа byte. Здесь целевая переменная охватывает меньший диапазон значений, чем исходная. Переменная типа byte может хранить целочисленные значения в диапазоне от –128 до 127, а переменная int — намного больший диапазон значений.

Если переменная-источник, применяемая в операции приведения типов, имеет значение, не совместимое с типом целевой переменной, Java изменяет значение, чтобы успешно выполнить операцию. Но поскольку это может привести к неожиданным результатам, подобных ситуаций следует избегать.

Приведение типов объектов

Если исходный и целевой объекты связаны в рамках наследования, то можно выполнять операции приведения типов объектов. При этом один класс должен быть подклассом другого.

Некоторые объекты вообще не требуют приведения типов. Объект можно использовать в любых операциях, в которых ожидается какой-либо из его суперклассов. Все объекты в Java являются подклассами класса Object, поэтому любой объект можно использовать в качестве аргумента там, где требуется класс Object.

Объект можно также использовать там, где требуется любой из его подклассов. Но поскольку подклассы обычно содержат больше информации, чем их суперклассы, часть информации может быть утеряна. Например, если объект не включает метод, который находится в подклассе, то в случае обращения к этому методу в программе появится сообщение об ошибке.

Чтобы использовать объект класса вместо одного из его подклассов, необходимо выполнить явное приведение типов, как показано в следующем примере:

```
public void paintComponent(Graphics comp) {
   Graphics2D comp2D = (Graphics2D) comp;
}
```

Здесь объект comp класса Graphics приводится к типу Graphics2D объекта comp2D. Вы не потеряете никакой информации в ходе такого приведения типов, зато получите доступ ко всем методам и переменным, определенным в подклассе.

Преобразование простых переменных в объекты и обратно

Каждому из простых типов переменных соответствуют аналогичные классы в Java: Boolean, Byte, Character, Double, Float, Integer, Long и Short. Названия всех классов начинаются с прописной буквы, поскольку все они являются объектами, а не простыми типами переменных.

Работать с этими объектами не сложнее, чем с соответствующими типами данных. Например, следующая инструкция создает объект `Integer`, которому присвоено значение `5309`:

```
Integer suffix = 5309;
```

Полученная объектная переменная `suffix` представляет целочисленное значение 5309.

Такой объект можно использовать подобно любому другому объекту. В частности, его можно присвоить простой переменной:

```
int newSuffix = suffix;
```

В результате выполнения этой инструкции переменная `newSuffix` получает значение 5309, выраженное теперь как значение типа `int`.

Преобразование базовых типов данных в классы и обратное преобразование классов в базовые типы данных выполняется в Java с помощью процедур автоупаковки и распаковки. *Автоупаковка* позволяет преобразовать значение простой переменной в соответствующий класс, а *распаковка* позволяет преобразовать объект в соответствующее простое значение. Автоупаковка и распаковка выполняются в фоновом режиме. Если, например, требуется простой тип данных, такой как `float`, объект преобразуется в соответствующий тип данных с тем же значением. Если же нужен объект типа `Float`, соответствующий тип данных будет автоматически преобразован в этот объект.

Одно из наиболее распространенных преобразований из объекта в переменную выполняется с помощью строк в числовых выражениях. Чтобы преобразовать строковое значение в целочисленное, можно воспользоваться методом `parseInt()` из класса `Integer`.

```
String count = "25";
int myCount = Integer.parseInt(count);
```

Этот код преобразует строку с текстом "25" в целочисленное значение 25. Если строковое значение не является допустимым целым числом, преобразование не сработает.

А сейчас мы создадим проект `NewRoot`. Это приложение преобразует строковое значение аргумента командной строки в числовое. Подобная методика часто применяется при обработке данных, вводимых пользователем в командной строке.

В среде NetBeans вернитесь к проекту `Java24` и выполните команду **Файл⇨ Создать файл**. Затем выберите пункт **Пустой файл Java**, присвойте файлу имя `NewRoot.java` и включите его в пакет `com.java24hours`. В редакторе исходного кода введите код листинга 10.1 и сохраните файл.

ЛИСТИНГ 10.1. Исходный код программы `NewRoot.java`

```
 1: package com.java24hours;
 2:
 3: class NewRoot {
 4:    public static void main(String[] arguments) {
 5:       int number = 100;
 6:       if (arguments.length > 0) {
 7:          number = Integer.parseInt(arguments[0]);
 8:       }
 9:       System.out.println("Квадратный корень из "
10:          + number
11:          + " равен "
12:          + Math.sqrt(number)
13:       );
14:    }
15: }
```

Прежде чем запустить программу на выполнение, нужно сконфигурировать среду NetBeans для работы с аргументами командной строки.

1. Выберите команду Выполнить⇨Установить конфигурацию проекта⇨Настроить. На экране появится окно Свойства проекта.
2. В поле Главный класс введите **`com.java24hours.NewRoot`**.
3. В поле Аргументы введите число **`9025`**.
4. Щелкните на кнопке OK, чтобы закрыть диалоговое окно.

Чтобы запустить эту программу, выберите команду Выполнить⇨Запустить проект (вместо Выполнить⇨Выполнить файл). Программа сообщит, какое число получено и чему равен его квадратный корень (рис. 10.3).

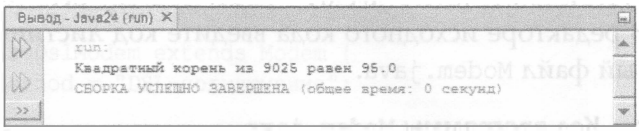


РИС. 10.3. Результат выполнения программы `NewRoot`

Программа `NewRoot` является результатом модификации приложения, рассмотренного на занятии 4, которое извлекало квадратный корень из числа 225.

Эта программа была бы более полезной, если бы принимала число, введенное пользователем, и вычисляла его квадратный корень. Для этого требуется выполнять преобразование из строки в целое число. Все аргументы командной строки хранятся в виде элементов массива `String`, поэтому их

следует преобразовывать в числа перед использованием в математических выражениях.

Чтобы создать целочисленное значение на основе строки, нужно вызвать метод `Integer.parseInt()` со строкой в качестве аргумента, как показано в строке 7:

```
number = Integer.parseInt(arguments[0]);
```

Элемент массива `arguments[0]` хранит первый аргумент командной строки, переданный при запуске приложения. Если программа была запущена с аргументом "9025", то строка "9025" преобразуется в целое число 9025.

Попробуйте изменить аргумент командной строки с 9025 на другое число и выполните программу несколько раз.

Создание объекта

Следующий проект, который мы рассмотрим на этом занятии, посвящен демонстрации наследования классов. В рамках этого проекта создаются классы для двух типов объектов: кабельные модемы, представленные классом `CableModem`, и DSL-модемы, представленные классом `DslModem`. Мы сфокусируемся на простых атрибутах и поведении этих объектов:

- каждый объект должен отображать скорость работы модема;
- каждый объект должен иметь возможность подключаться к Интернету.

Кабельные модемы и DSL-модемы имеют определенную скорость работы. Поскольку этот атрибут является общим для обеих классов устройств, его можно включить в класс, который является суперклассом для классов `CableModem` и `DslModem`. Назовем новый класс `Modem`. В среде NetBeans создайте новый пустой класс Java с именем `Modem` и включите его в пакет `com.java24hours`. В редакторе исходного кода введите код листинга 10.2 и сохраните полученный файл `Modem.java`.

ЛИСТИНГ 10.2. Код программы `Modem.java`

```
1: package com.java24hours;
2:
3: public class Modem {
4:    int speed;
5:
6:    public void displaySpeed() {
7:       System.out.println("Быстродействие: " + speed);
8:    }
9: }
```

Этот файл компилируется автоматически, в результате чего генерируется файл `Modem.class`. Такую программу нельзя запустить на выполнение непосредственно, но ее можно использовать в других классах. Класс `Modem` способен отображать скорость работы модема — поведение, поддерживаемое обеими подклассами: `CableModem` и `DslModem`. Указав ключевое слово `extends` при создании классов `CableModem` и `DslModem`, вы сделаете каждый из них подклассом класса `Modem`.

В среде NetBeans создайте новый пустой класс Java с именем `CableModem` и включите его в пакет `com.java24hours`. Введите код листинга 10.3 и сохраните полученный файл `CableModem.java`.

ЛИСТИНГ 10.3. Исходный код программы `CableModem.java`

```
 1: package com.java24hours;
 2:
 3: public class CableModem extends Modem {
 4:    String method = "кабельное подключение";
 5:
 6:    public void connect() {
 7:       System.out.println("Подключение к Интернету...");
 8:       System.out.println("Используем " + method);
 9:    }
10: }
```

В среде NetBeans создайте третий класс, `DslModem`, и включите его в пакет `com.java24hours`. Введите код листинга 10.4 и сохраните файл под именем `DslModem.java`.

ЛИСТИНГ 10.4. Исходный код программы `DslModem.java`

```
 1: package com.java24hours;
 2:
 3: public class DslModem extends Modem {
 4:    String method = "DSL-подключение";
 5:
 6:    public void connect() {
 7:       System.out.println("Подключение к Интернету...");
 8:       System.out.println("Используем " + method);
 9:    }
10: }
```

Если при вводе кода не было допущено ошибок, в вашем распоряжении окажутся три файла классов: `Modem.class`, `CableModem.class` и `DslModem.class`. Но ни один из них невозможно запустить на выполнение, поскольку в них отсутствуют блоки `main()`, в отличие от других ранее созданных

программ. Для проверки созданной иерархии классов понадобится создать небольшое приложение.

Вернитесь в среду NetBeans, создайте новый пустой класс Java с именем `ModemTester` и включите его в пакет `com.java24hours`. В редакторе исходного кода введите код листинга 10.5 и сохраните полученный файл `ModemTester.java`.

ЛИСТИНГ 10.5. Код программы `ModemTester.java`

```
 1: package com.java24hours;
 2:
 3: public class ModemTester {
 4:    public static void main(String[] arguments) {
 5:       CableModem surfBoard = new CableModem();
 6:       DslModem gateway = new DslModem();
 7:       surfBoard.speed = 500000;
 8:       gateway.speed = 400000;
 9:       System.out.println("Попытка подключения кабельного модема:");
10:       surfBoard.displaySpeed();
11:       surfBoard.connect();
12:       System.out.println("Попытка подключения DSL-модема:");
13:       gateway.displaySpeed();
14:       gateway.connect();
15:    }
16: }
```

После запуска программы будет получен результат, показанный на рис. 10.4.

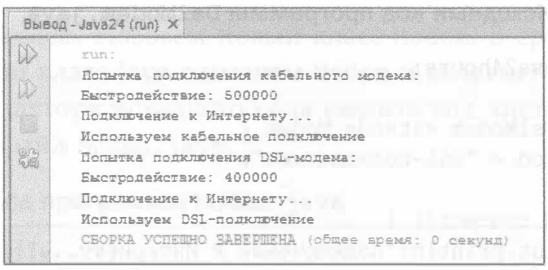


РИС. 10.4. Результат выполнения программы `ModemTester`

Проанализируем работу программы.

- Строки 5 и 6. Созданы два новых объекта: объект `surfBoard` класса `CableModem` и объект `gateway` класса `DslModem`.
- Строка 7. Переменной `speed` объекта `surfBoard` присвоено значение `500000`.

- Строка 8. Переменной `speed` объекта `gateway` присвоено значение `400000`.
- Строка 10. Вызывается метод `displaySpeed()` объекта `surfBoard`. Этот метод наследуется из класса `Modem`. Даже если метод отсутствует в классе `CableModem`, его можно вызвать из родительского класса.
- Строка 11. Вызывается метод `connect()` объекта `surfBoard`.
- Строка 13. Вызывается метод `displaySpeed()` объекта `gateway`.
- Строка 14. Вызывается метод `connect()` объекта `gateway`.

Резюме

Создав свой первый класс объектов и упорядочив несколько классов в виде иерархии, вы получили представление об объектно-ориентированном программировании (ООП). Дополнительные сведения о поведении и атрибутах объекта будут приведены на следующих двух занятиях в процессе создания более сложных объектов.

Термины "программа", "класс" и "объект" станут для вас понятнее после того, как вы овладеете объектно-ориентированными приемами разработки. Конечно, на освоение ООП уйдет время, но в результате вы изучите эффективный способ проектирования, разработки и отладки компьютерных программ.

Вопросы и ответы

В. Могут ли классы наследоваться от нескольких классов?

О. Это возможно в некоторых языках программирования (например, в C++), но не в Java. Множественное наследование — это мощное средство, но оно же затрудняет изучение и использование ООП. Наверное, по этой причине разработчики языка Java решили ограничить наследование одним суперклассом для любого класса, но при этом класс может иметь несколько подклассов. Можно обойти данное ограничение путем наследования методов из класса специального типа, называемого *интерфейсом*. Дополнительные сведения об интерфейсах будут изложены на занятии 15.

В. В каких ситуациях уместно создавать метод, который не является общедоступным (`public`)?

О. Такие методы предназначены исключительно для одной программы. Если, например, вы создаете компьютерную игру и метод `shootRayGun()` предназначен исключительно для нее, значит, он должен быть закрытым

(private). Чтобы превратить общедоступный метод в закрытый, удалите ключевое слово public из объявления метода.

В. Почему значения типа char могут использоваться в качестве значений типа int?

О. Возможность использования символов в качестве переменных типа int объясняется тем, что каждому символу соответствует определенный числовой код, который отражает его положение в наборе символов. Например, если к переменной k, имеющей значение 67, применить инструкцию (char) k, то будет получено символьное значение 'C'. Дело в том, что в соответствии с таблицей ASCII прописной букве C соответствует код 67. Набор символов ASCII является частью стандарта Unicode, поддерживаемого в языке Java.

Коллоквиум

Проверьте знание материала, рассмотренного на этом занятии, ответив на следующие вопросы.

Контрольные вопросы

1. Какое ключевое слово используется для указания наследования одного класса из другого класса?
   - **А.** inherits.
   - **Б.** extends.
   - **В.** handItOverAndNobodyGetsHurt.
2. Почему скомпилированные программы на Java сохраняются с расширением .class?
   - **А.** Потому что разработчики Java считают, что это классный язык.
   - **Б.** Это проявление уважения к преподавателям.
   - **В.** Каждая программа на Java является классом.
3. Какие две сущности характеризуют объект?
   - **А.** Атрибуты и поведение.
   - **Б.** Команды и файлы данных.
   - **В.** Душа и тело.

Ответы

1. **Б.** Это ключевое слово extends, поскольку подкласс является расширением атрибутов и поведения суперкласса и любых других суперклассов, находящихся выше него в иерархии наследования.

2. **В.** Программы на Java всегда состоят по крайней мере из одного главного класса и любых других необходимых классов.
3. **А.** В каком-то смысле вариант **Б** тоже правильный, поскольку команды сопоставимы с поведением, а файлы данных аналогичны атрибутам.

Упражнения

Чтобы углубить познания в области объектов, выполните следующие упражнения.

- Создайте класс `Commodore64Modem`, в котором скорость работы модема равна 300 и используется собственный метод `connect()`.
- В один из классов проекта `Modem` добавьте метод `disconnect()`, в котором принимается решение об отключении кабельного модема, DSL-модема или модема Commodore 64.

ЗАНЯТИЕ 11
Работа с объектами

На этом занятии вы узнаете...

- как создавать переменные для объекта или класса;
- как вызывать методы объектов и классов;
- как создавать конструкторы;
- как передавать аргументы методу;
- как применять ключевое слово `this` для ссылки на объекты;
- как образом создавать новые объекты.

На предыдущем занятии вы узнали о том, как структурировать программы с помощью объектов. Объекты включают атрибуты и поведение.

Атрибуты — это информация, которая хранится в объекте. В качестве такой информации могут использоваться простые переменные (например, целочисленные, символьные и булевы) либо объекты (например, `String` и `Calendar`). *Поведение* — это набор инструкций, применяемых для выполнения определенных операций в объекте. Каждый из таких наборов называется *методом.*

До сих пор вы использовали методы и переменные объектов, даже не подозревая об этом. Всякий раз, когда в инструкции встречается точка (`.`), не являющаяся частью строки или числа, имеет место ссылка на объект.

Создание переменных

На этом занятии будет рассмотрен представитель класса объектов `Gremlin`, единственное предназначение которого заключается в самовоспроизведении. Этот объект выполняет несколько операций, реализованных в виде поведения класса. Информация, требуемая для методов, сохранена в виде атрибутов.

Атрибуты объекта представляют собой переменные, необходимые для функционирования объекта. В качестве переменных могут использоваться

целые числа, символы, числа с плавающей точкой, а также массивы и объекты, такие как `String` или `Calendar`. Переменные объекта, называемые переменными экземпляра, можно сводобно использовать в пределах класса, в любом из методов, которые содержатся в объекте. В соответствии с соглашением переменные объявляются сразу же после ключевого слова `class`, перед объявлением методов.

Для объекта `Gremlin` требуется уникальный идентификатор, который позволит отличать его от других объектов данного класса. Идентификатор объекта `Gremlin` хранится в целочисленной переменной `guid`. Начнем создавать класс `Gremlin`, включающий атрибут `guid` и два других атрибута.

```
public class Gremlin {
   public int guid;
   public String creator = "Крис Коламбус";
   int maximumAge = 240;
}
```

Все три переменные — `guid`, `maximumAge` и `creator` — являются атрибутами класса.

Ключевое слово `public`, добавляемое к объявлению переменной, — это *спецификатор доступа*, определяющий, каким образом другие объекты, созданные на основе других классов, могут использовать данную переменную и могут ли они в принципе обращаться к ней.

Благодаря спецификатору `public` обеспечивается возможность изменения переменной из другой программы, которая использует объект `Gremlin`.

Если, например, другая программа хочет изменить имя создателя, переменной `creator` нужно присвоить новое значение. Следующие инструкции создают объект `gizmo` класса `Gremlin` и присваивают переменной `creator` соответствующее значение.

```
Gremlin gizmo = new Gremlin();
gizmo.creator = "Джо Данте";
```

Переменная `creator` в классе `Gremlin` имеет тип доступа `public`, поэтому может свободно изменяться другими программами. Другая переменная, `maximumAge`, может использоваться только в самом классе.

Если переменная класса имеет тип доступа `public`, класс теряет контроль над ее использованием со стороны других программ. Во многих случаях это не является проблемой. Например, переменной `creator` можно присвоить любое значение, которое идентифицирует создателя гремлинов.

Ограничение доступа к переменной позволяет избежать ошибок, которые могут иметь место в случае некорректного изменения значения этой переменной со стороны другой программы. Например, переменная `maximumAge` задает длительность существования гремлина в его нынешней форме (в

часах). Поэтому отрицательное значение этой переменной лишено всякого смысла. Чтобы защитить класс объектов `Gremlin` от появления подобных проблем, нужно выполнить следующее:

- выбрать более ограничивающий тип доступа к переменной: `protected` или `private`;
- добавить поведение, которое изменяет значение переменной и возвращает это значение другим программам.

Переменную с типом доступа `protected` можно использовать в том же классе, в котором находится исходная переменная, а также в любом подклассе этого класса и в классах, находящихся в том же пакете. *Пакет* — это набор связанных классов, имеющих одно и то же назначение. В качестве примера можно привести пакет `java.util`, содержащий полезные вспомогательные классы, в частности, для архивирования файлов и работы со значениями даты/времени. Если в программе на Java встречается инструкция `import` со звездочкой, например `import java.util.*`, то это означает, что в программе можно использовать все классы данного пакета.

Переменная с типом доступа `private` еще более ограничена, чем переменная с типом доступа `protected`. Ее можно использовать лишь в ее собственном классе. Чтобы ограничить возможности по изменению значения переменной, выберите тип доступа `private` или `protected`.

Следующая инструкция назначает переменной `guid` тип доступа `private`:

```
private int guid;
```

Если хотите, чтобы другие программы могли обращаться к переменной `guid`, нужно создать поведение, которое сделает возможным подобное обращение. Соответствующий пример будет рассмотрен далее.

Также возможен другой тип контроля доступа, когда при создании переменной вообще не указывается спецификатор `public`, `private` или `protected`.

В большинстве программ, созданных на предыдущих занятиях, тип доступа не указывался. Если контроль доступа не применяется, переменная будет доступна лишь для классов, находящихся в том же пакете. В данном случае имеет место так называемый доступ по умолчанию, или пакетный доступ.

Создание переменных класса

При создании объект получает собственную версию всех переменных, которые являются частью класса. Каждый объект, созданный на основе класса `Gremlin`, имеет собственную копию переменных `guid`, `maximumAge` и `creator`. Если одна из переменных объекта была изменена, это не окажет влияния на ту же самую переменную в другом объекте `Gremlin`.

Бывают ситуации, когда атрибут должен описывать целый класс объектов, а не один конкретный объект. В таком случае мы имеем дело с переменными класса. Например, чтобы отслеживать количество объектов `Gremlin`, используемых в программе, соответствующую информацию можно хранить в переменной класса. Во всем классе существует лишь одна копия такой переменной. Переменные, которые до сих пор создавались для объектов, были переменными экземпляров, поскольку они связаны с конкретным объектом.

Оба типа переменных создаются и используются одинаковым образом, но при создании переменных класса дополнительно указывается ключевое слово `static`. Следующая инструкция создает переменную класса `Gremlin`:

```
static int gremlinCount = 0;
```

Изменение значения переменной класса ничем не отличается от изменения объектных переменных. Например, переменную `gremlinCount` из класса `Gremlin`, для которого создан объект с именем `stripe`, можно изменить с помощью следующей инструкции:

```
stripe.gremlinCount++;
```

Поскольку переменные класса относятся ко всему классу, можно также использовать имя класса вместо имени объекта:

```
Gremlin.gremlinCount++;
```

Обе инструкции выполняют одни и те же действия, но предпочтительнее все же указывать имя класса. В таком случае сразу же становится очевидно, что `gremlinCount` является переменной класса, а не переменной экземпляра. Если при работе с переменными класса всегда указывать имена объектов, придется анализировать исходный код, чтобы понять их назначение.

Переменные класса также называются *статическими переменными*.

ПРЕДУПРЕЖДЕНИЕ

Несмотря на всю пользу переменных класса, не следует чрезмерно увлекаться ими. Эти переменные существуют до тех пор, пока существует класс. Если в переменных класса хранится большой массив объектов, это приведет к большим затратам памяти, которая никогда не будет освобождена.

Создание методов

С помощью атрибутов задается информация о классе, но для того чтобы класс выполнял то, что от него требуется, нужно создать поведение. Поведение — это функциональные фрагменты класса, которые выполняют определенные операции. Каждый из таких фрагментов называется *методом*.

До сих пор вы использовали методы во всех своих программах, даже не подозревая об этом. Вспомните, например, метод `println()`, который отображает текст на экране. Подобно переменным, методы применяются в связке с объектом или классом. После имени объекта либо класса ставится точка, а затем указывается имя метода, например `object2.move()` или `Integer.parseInt()`.

ПРИМЕЧАНИЕ

Выражение `System.out.println()` может вызывать недоумение, поскольку содержит две точки вместо одной. Наличие двух точек объясняется тем, что в инструкции используются два класса: `System` и `PrintStream`. Класс `System` содержит статическую переменную `out`, являющуюся объектом класса `PrintStream`. Метод `println()` относится именно к классу `PrintStream`. Таким образом, инструкция `System.out.println()` означает следующее: "Вызвать метод `println()` для переменной `out` класса `System`".

Объявление метода

Инструкция создания метода напоминает инструкцию, применяемую для создания класса. Обе инструкции могут иметь аргументы, заключенные в круглые скобки (указываются после названия метода или класса). Тело метода или класса обрамляется фигурными скобками. Различие заключается в том, что методы могут возвращать значение вызывающей программе. Это значение может быть одним из простых типов данных, таким как целое число или строка, либо объектом.

Ниже представлен пример метода из класса `Gremlin`, который используется для создания нового гремлина.

```
public Gremlin replicate(String creator) {
   Gremlin noob = new Gremlin();
   noob.creator = "Стивен Спилберг";
   return noob;
}
```

Этот метод имеет единственный аргумент — строковую переменную `creator`, которая представляет имя создателя гремлинов.

В объявлении метода слово `Gremlin` предваряет имя `replicate`. Это означает, что метод возвращает объект класса `Gremlin`. Само возвращаемое значение указывается в инструкции `return`. В данном случае возвращается объект `noob`.

Если метод не возвращает никакого значения, в объявлении метода следует указать ключевое слово `void`.

В случае, когда метод возвращает значение, он может стать частью выражения. Например, если создан объект `haskins` класса `Gremlin`, можно использовать следующие инструкции.

```
if (haskins.active() == true) {
   System.out.println(haskins.guid + " активный.");
}
```

Метод, возвращающий значение, может использоваться везде, где допустима переменная.

Ранее уже было показано, как изменить тип доступа к переменной `guid` с `public` на `private`, чтобы предотвратить чтение или изменение значения этой переменной другими программами.

Если переменная экземпляра является закрытой (`private`), то работать с ней следует по-другому. Необходимо создать общедоступные методы в классе `Gremlin`, которые возвращают значение переменной `guid` и присваивают ей новое значение. Чтобы эти методы могли свободно вызываться из других программ, их следует объявить как `public` (в отличие от самой переменной `guid`, которая объявлена как `private`).

Рассмотрим оба метода.

```
public int getGuid() {
   return guid;
}
public void setGuid(int newValue) {
   if (newValue > 0) {
      guid = newValue;
   }
}
```

Эти методы называются *методами доступа*, поскольку они позволяют получить доступ к переменной `guid` из других объектов.

Метод `getGuid()` применяется для получения текущего значения переменной `guid`. У него отсутствуют какие-либо аргументы, но все равно после имени метода нужно поставить круглые скобки. Метод `setGuid()` имеет один целочисленный аргумент `newValue`, представляющий новое значение переменной `guid`. Если аргумент `newValue` больше нуля, значение переменной будет изменено.

В рассматриваемом примере класс `Gremlin` контролирует способ использования переменной `guid` другими классами. Данный процесс называется *инкапсуляцией*, и это одна из фундаментальных концепций ООП. Чем лучше объекты смогут защитить себя от некорректного использования, тем больше от них пользы в других программах.

Несмотря на то что переменная `guid` имеет тип доступа `private`, новые методы `getGuid()` и `setGuid()` могут работать с ней, поскольку относятся к тому же самому классу.

СОВЕТ

В интегрированной среде разработки (IDE) методы доступа часто создаются автоматически. Подобная возможность предлагается и в среде NetBeans. Чтобы ознакомиться с соответствующим примером, откройте класс `Modem`, который рассматривался на занятии 10, щелкните в любом месте редактора исходного кода и выберите команду **Средство реорганизации кода⇨Инкапсулировать поля**. В появившемся диалоговом окне щелкните на кнопке **Выбрать все**, а затем — на кнопке **Средство реорганизации кода**. В результате NetBeans изменит тип доступа к переменной `speed` на `private` и создаст новые методы `getSpeed()` и `setSpeed()`.

Похожие методы с разными аргументами

На примере метода `setGuid()` было показано, что методу можно передавать аргументы, влияющие на его работу. Обычно у методов разные имена, но методы могут иметь и одинаковые имена, даже когда у них разные аргументы.

Два метода могут иметь одно и то же имя и при этом располагать разным числом аргументов. К тому же сами аргументы могут быть разных типов. Например, класс `Gremlin` может включать два метода `tauntHuman()`. У первого из них отсутствуют аргументы, и он выводит стандартное сообщение. Во втором методе выводимое сообщение задается в виде строкового аргумента.

```
void tauntHuman() {
   System.out.println("Это будет больно!");
}

void tauntHuman(String taunt) {
   System.out.println(taunt);
}
```

Оба метода называются одинаково, но имеют разные списки аргументов. У первого метода аргументы отсутствуют, а второй метод имеет единственный аргумент типа `String`. Аргументы метода формируют его *сигнатуру*. Класс может включать разные методы с одним и тем же именем, при условии, что у каждого из них разная сигнатура.

Конструкторы

Для создания нового объекта в программе предназначена инструкция `new`, как показано в следующем примере:

```
Gremlin clorr = new Gremlin();
```

Данная инструкция создает новый объект `clorr` класса `Gremlin`. При использовании инструкции `new` запускается специальный метод класса, называемый *конструктором*, который выполняет всю необходимую работу по

созданию объекта. Назначение конструктора заключается в настройке всех переменных и вызове методов, которые должны быть выполнены для корректного функционирования объекта.

Конструкторы создаются подобно другим методам, но они не могут возвращать значений. Ниже представлены два конструктора класса `Gremlin`.

```
public Gremlin() {
   creator = "Майкл Финнелл"; // атрибут creator - это строка
   maximumAge = 240; // атрибут maximumAge - это значение типа int
}

public Gremlin(String name, int size) {
   creator = name;
   maximumAge = size;
}
```

Подобно другим методам, конструкторы могут иметь аргументы, что позволяет определить несколько конструкторов в классе. В данном примере первый конструктор вызывается, когда инструкция `new` вводится следующим образом:

```
Gremlin blender = new Gremlin();
```

Второй конструктор вызывается только в том случае, если в качестве аргументов инструкции `new` передаются строка и целочисленное значение.

```
Gremlin plate = new Gremlin("Зак Гэллиган", 960);
```

Если у класса нет ни одного конструктора, из суперкласса будет наследоваться единственный конструктор без аргументов. В зависимости от используемого суперкласса могут наследоваться и другие конструкторы.

В любом классе должен существовать конструктор, который имеет то же количество и тип аргументов, что и инструкция `new`, используемая для создания объектов этого класса. В примере с классом `Gremlin`, который включает конструкторы `Gremlin()` и `Gremlin(String name, int size)`, можно создавать объекты с двумя разными типами инструкций `new`: одна без аргументов и вторая с двумя аргументами (строка и целое число).

ПРЕДУПРЕЖДЕНИЕ

Если подкласс определяет конструктор с одним или несколькими аргументами, он перестает наследовать конструктор без аргументов из своего суперкласса. Поэтому, если класс включает другие конструкторы, нужно всегда определять конструктор без аргументов.

Методы класса

Как и переменные класса, методы класса позволяют реализовать функциональные средства, присущие всему классу, а не отдельному объекту.

Метод класса следует использовать в тех случаях, когда он не предназначен для воздействия на отдельный объект класса. На занятии 10 применялся метод `parseInt()` из класса `Integer` для преобразования строки в переменную типа `int`:

```
int fontSize = Integer.parseInt(fontText);
```

Это метод класса. Чтобы превратить метод в метод класса, перед спецификацией метода вставьте ключевое слово `static`, как показано в следующем примере.

```
static void showGremlinCount() {
   System.out.println("Здесь " + gremlinCount + " гремлинов");
}
```

Переменная класса `gremlinCount` уже применялась для отслеживания количества объектов `Gremlin`, созданных в программе. Метод `showGremlinCount()` является методом класса, который отображает общее количество гремлинов. Для вызова этого метода используется следующая инструкция:

```
Gremlin.showGremlinCount();
```

Область видимости переменных в методах

Если в методе создается переменная или объект, их можно использовать только в этом методе. Причина заключается в концепции области видимости переменной. Область видимости — это блок программы, в пределах которого существует переменная. Если выйти за пределы области видимости переменной, то получить доступ к этой переменной не удастся.

Фигурные скобки в программе задают границы области видимости переменных. Любая переменная, созданная в этой области, не может использоваться за ее пределами. Рассмотрим, к примеру, следующие инструкции.

```
if (numFiles < 1) {
   String warning = "Больше не осталось файлов.";
}
System.out.println(warning);
```

Этот код неработоспособен и даже не компилируется в NetBeans, поскольку переменная `warning` была создана внутри блока `if`. Здесь фигурные скобки определяют область видимости переменной. Переменная `warning` недоступна за пределами фигурных скобок, поэтому метод `System.out.println()` не может использовать ее в качестве аргумента.

Если одни фигурные скобки вложены в другие, необходимо следить за областью видимости вложенных переменных. Рассмотрим следующий пример.

```
if (humanCount < 5) {
   int status = 1;
   if (humanCount < 1) {
      boolean firstGremlin = true;
      status = 0;
   } else {
      firstGremlin = false;
   }
}
```

Замечаете какие-либо проблемы? В этом примере переменная `status` может применяться в любом месте блока, но выполнение инструкции, которая присваивает значение `false` переменной `firstGremlin`, вызовет ошибку компилятора. Поскольку переменная `firstGremlin` создана в блоке инструкции `if (humanCount < 1)`, она недоступна в области видимости следующей за ней инструкции `else`.

Чтобы устранить эту проблему, нужно создать переменную `firstGremlin` за пределами обоих блоков, тогда ее область видимости будет охватывать их. Одно из возможных решений — объявить переменную `firstGremlin` сразу после переменной `status`.

Правила, задающие область видимости, облегчают отладку программы, поскольку область видимости ограничивает сферу применения переменной. Это позволяет избежать типичных ошибок программирования, когда одна и та же переменная используется двумя разными способами в разных разделах программы.

Концепция области видимости также относится к методам, поскольку они определяются открывающей и закрывающей скобками. Переменная, созданная в методе, не может использоваться другими методами. К переменной можно обращаться в нескольких методах только в том случае, когда она создана как переменная класса.

Вложенные классы

Программа на Java обычно называется классом, но во многих ситуациях для работы программы требуется несколько классы. Подобные программы состоят из главного класса и нескольких вспомогательных классов.

Существуют два способа определения вспомогательных классов. Первый способ заключается в отдельном описании каждого класса, как показано в следующем примере.

```
public class Wrecker {
   String creator = "Фиби Кейтс";

   public void destroy() {
```

```
      GremlinCode gc = new GremlinCode(1024);
   }
}

class GremlinCode {
   int vSize;

   GremlinCode(int size) {
      vSize = size;
   }
}
```

В этом примере класс `GremlinCode` является вспомогательным для класса `Wrecker`. Вспомогательные классы часто определяются в том же файле исходного кода, что и класс, которому они "помогают". После компиляции файла исходного кода создаются несколько файлов классов. В результате компиляции предыдущего примера были созданы файлы `Wrecker.class` и `GremlinCode.class`.

ПРЕДУПРЕЖДЕНИЕ

Если в одном файле исходного кода определено несколько классов, лишь один из них может быть общедоступным (`public`). В объявлениях других классов не должно использоваться ключевое слово `public`. Имя файла исходного кода должно соответствовать общедоступному классу, который в нем определен.

При создании главного и вспомогательного классов последний можно включить в главный класс. В этом случае вспомогательный класс называется *внутренним* или *вложенным*.

Внутренний класс можно объявить внутри фигурных скобок другого класса.

```
public class Wrecker {
   String creator = "Хойт Акстон";

   public void destroy() {
      GremlinCode vic = new GremlinCode(1024);
   }

   class GremlinCode {
      int vSize;

      GremlinCode(int size) {
         vSize = size;
      }
   }
}
```

Внутренний класс используется точно так же, как и любой другой вспомогательный класс. Основное отличие внутренних классов от вспомогательных проявляется в процессе их компиляции. По завершении компиляции файлам внутренних классов не присваивается имя, указанное в соответствующей инструкции `class`. Вместо этого компилятор присваивает таким файлам имя, которое включает имя главного класса.

В предыдущем примере компилятор сгенерировал файлы `Wrecker.class` и `Wrecker$GremlinCode.class`.

ПРИМЕЧАНИЕ

При работе с внутренними классами можно использовать сложные методики программирования на Java, которые станут более понятными по мере приобретения опыта. Если говорить точнее, то через пять занятий. Мы вернемся к этой теме на занятии 16.

Использование ключевого слова `this`

Поскольку можно ссылаться на переменные и методы как собственных, так и других классов, порой могут возникать неоднозначные ситуации. Чтобы избежать этого, для ссылки на собственный объект программы используют ключевое слово `this`.

При ссылке на методы или переменные объекта нужно поместить имя объекта перед именем метода или переменной, разделив их точкой. Рассмотрим следующие примеры.

```
Gremlin mogwai = new Gremlin();
mogwai.creator = "LoveHandles";
mogwai.setGuid(75);
```

Эти инструкции создают новый объект `mogwai` класса `Gremlin`, присваивают значение переменной `creator` объекта `mogwai` и вызывают его метод `setGuid()`.

Бывают ситуации, когда нужно обратиться к текущему объекту, т.е. к объекту, представленному самой программой. Например, в классе `Gremlin` может быть метод, который имеет собственную переменную `creator`.

```
public void checkCreator() {
   String creator = null;
}
```

В данном примере переменная `creator` находится в области видимости метода `checkCreator()`, но это не та же переменная, что и переменная класса `creator`. Чтобы обратиться к переменной `creator` текущего объекта, нужно использовать ключевое слово `this`:

```
System.out.println(this.creator);
```

С помощью ключевого слова this конкретизируется переменная или метод, на который устанавливается ссылка. Это ключевое слово можно использовать в любом месте класса, где требуется ссылаться на объект по имени. Если, например, нужно передать текущий объект в качестве аргумента метода, можно использовать следующую инструкцию:

```
verifyData(this);
```

Во многих случаях для уточнения ссылки на переменные и методы объекта ключевое слово this не требуется. С другой стороны, оно никогда не будет лишним, если нужно быть уверенным в том, что вы ссылаетесь на правильный объект.

Ключевое слово this может использоваться в конструкторе при установке значений переменных экземпляра. Возьмем, к примеру, класс Gremlin, который включает переменные creator и maximumAge. Его конструктор может выглядеть так.

```
public Gremlin(String creator, int maximumAge) {
   this.creator = creator;
   this.maximumAge = maximumAge;
}
```

Использование методов и переменных класса

В рамках первого проекта этого занятия мы создадим простой класс Gremlin, который подсчитывает количество объектов Gremlin в программе и выводит результат.

В среде NetBeans выполните команду Файл⇨Создать файл и создайте новый пустой класс Java с именем Gremlin. Включите этот класс в пакет com.java24hours. В редакторе исходного кода введите код из листинга 11.1 и сохраните полученный файл Gremlin.java.

ЛИСТИНГ 11.1. Исходный код программы Gremlin.java

```
 1: package com.java24hours;
 2:
 3: public class Gremlin {
 4:    static int gremlinCount = 0;
 5:
 6:    public Gremlin() {
 7:       gremlinCount++;
 8:    }
 9:
10:    static int getGremlinCount() {
```

```
11:        return gremlinCount;
12:     }
13: }
```

При сохранении файл будет автоматически скомпилирован в среде NetBeans. Поскольку в этом классе отсутствует метод `main()`, его невозможно запустить на выполнение непосредственно. Чтобы протестировать класс `Gremlin`, нужно создать второй класс, который будет создавать объекты `Gremlin`.

Класс `GremlinLab` — это простое приложение, которое создает объекты `Gremlin`, а затем подсчитывает количество созданных объектов с помощью метода `getGremlinCount()` класса `Gremlin`.

В среде NetBeans создайте новый файл, включите его в пакет `com.java24hours`, введите в этот файл код из листинга 11.2 и сохраните его под именем `GremlinLab.java`.

ЛИСТИНГ 11.2. Исходный код программы **`GremlinLab.java`**

```
 1: package com.java24hours;
 2:
 3: public class GremlinLab {
 4:    public static void main(String[] arguments) {
 5:       int numGremlins = Integer.parseInt(arguments[0]);
 6:       if (numGremlins > 0) {
 7:          Gremlin[] gremlins = new Gremlin[numGremlins];
 8:          for (int i = 0; i < numGremlins; i++) {
 9:             gremlins[i] = new Gremlin();
10:          }
11:          System.out.println("Здесь " + Gremlin.getGremlinCount() +
12:                             " гремлинов.");
13:       }
14:    }
15: }
```

Приложению `GremlinLab` необходимо передать один аргумент командной строки, определяющий количество создаваемых объектов `Gremlin`. Чтобы задать аргумент командной строки в NetBeans, выполните следующие действия.

1. Выберите команду Выполнить⇨Установить конфигурацию проекта⇨ Настроить. На экране появится диалоговое окно Свойства проекта.
2. В поле Главный класс введите **`GremlinLab`**, а в поле Аргументы — количество создаваемых объектов `Gremlin`.
3. Щелкните на кнопке OK, чтобы закрыть диалоговое окно.

Чтобы запустить программу с аргументами командной строки, в NetBeans выберите команду Выполнить⇨Запустить главный проект.

Аргументы считываются приложением в виде строкового массива, который передается методу `main()`. В классе `GremlinLab` это происходит в строке 4.

Чтобы работать с аргументом как с целым числом, следует преобразовать его из объекта `String` в целое число, воспользовавшись методом `parseInt()` класса `Integer`. В строке 5 на основе первого аргумента, переданного программе в командной строке, создается переменная `numGremlins` типа `int`.

Если значение переменной `numGremlins` больше нуля, приложение `GremlinLab` выполняет следующие действия.

- Строка 7. Создается массив объектов `Gremlin`, размер которого определяется значением переменной `numGremlins`.
- Строки 8–10. Цикл `for`, в котором вызывается конструктор для каждого объекта `Gremlin` в массиве.
- Строки 11 и 12. После того как все объекты `Gremlin` созданы, с помощью метода `getGremlinCount()` класса `Gremlin` подсчитывается количество таких объектов. Это количество должно совпадать со значением аргумента, который был задан при запуске приложения `GremlinLab`.

Если значение переменной `numGremlins` не является положительным, приложение `GremlinLab` не выполняет никаких действий.

После того как файл `GremlinLab.java` будет скомпилирован, запустите его на выполнение с аргументом командной строки, выбранным на ваше усмотрение. Количество создаваемых объектов `Gremlin` зависит от объема оперативной памяти вашей системы. На ноутбуке автора книги число, превышающее 80 млн гремлинов, приводило к аварийному завершению программы с выводом сообщения `OutOfMemoryError`.

Если количество объектов `Gremlin` не превышает максимально допустимое, результат выполнения программы будет подобен показанному на рис. 11.1.

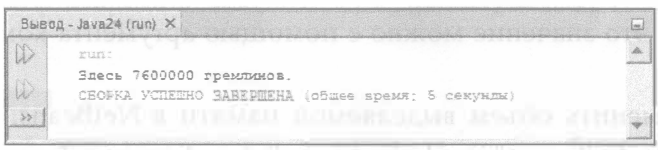


РИС. 11.1. Результат выполнения программы `GremlinLab`

Резюме

Итак, мы завершили два из трех занятий, посвященных объектно-ориентированному программированию. Вы узнали о том, как создавать объекты,

задавать поведение и атрибуты объектов и классов и преобразовывать объекты и переменные с помощью операции приведения типов.

Выработка объектного мышления является одной из наиболее сложных задач в процессе изучения Java. Но как только вы освоите подобное мышление, вы сразу же поймете, что Java основан на использовании объектов и классов.

На следующем занятии вы узнаете о том, как создавать иерархии наследования объектов.

Вопросы и ответы

В. Нужно ли создавать объект для использования переменных или методов класса?

О. Нет. Поскольку переменные и методы класса не связаны с конкретным объектом, не нужно создавать объект исключительно для их использования. В качестве примера можно привести метод `Integer.parseInt()`. В его случае не нужно создавать новый объект `Integer` лишь для преобразования строки в значение типа `int`.

В. Существует ли список всех встроенных методов, поддерживаемых в Java?

О. Компания Oracle предлагает исчерпывающую документацию по всем классам Java, включая описание всех общедоступных методов. Эта документация доступна по следующему адресу:

```
https://docs.oracle.com/javase/10/docs/api/overview-summary.html
```

В. Можно ли управлять объемом памяти, выделяемым программам Java?

О. Объем памяти, доступный виртуальной машине Java (JVM) при запуске приложения, определяется двумя факторами: общим объемом физической памяти, доступной на компьютере, и объемом используемой памяти, сконфигурированным в JVM. По умолчанию выделяется 256 Мбайт. Изменить это значение можно с помощью аргумента командной строки `-Xmx`.

Чтобы изменить объем выделяемой памяти в NetBeans, выберите команду Выполнить⇨Установить конфигурацию проекта⇨Настроить. На экране появится диалоговое окно Свойства проекта с выбранной вкладкой Выполнение. Чтобы выделить 1024 Мбайт оперативной памяти для JVM, в поле Параметры VM введите **`-Xmx1024M`**. Измените это значение в случае необходимости. Также введите данные в поля Главный класс и

Аргументы, после чего выполните программу с помощью команды Выполнить⇨Запустить главный проект.

Коллоквиум

Проверьте знание материала, рассмотренного на этом занятии, ответив на следующие вопросы.

Контрольные вопросы

1. Примером чего является метод в классе Java?
   - **А.** Атрибутов.
   - **Б.** Инструкций.
   - **В.** Поведения.
2. Какое ключевое слово применяется для определения переменной класса?
   - **А.** `new`.
   - **Б.** `public`.
   - **В.** `static`.
3. Как называется раздел программы, в котором существует переменная?
   - **А.** Гнездо.
   - **Б.** Область видимости.
   - **В.** Аллея переменных.

Ответы

1. **В.** Метод состоит из инструкций, но является примером поведения.
2. **В.** Если не указано ключевое слово `static`, то переменная будет не переменной класса, а переменной экземпляра.
3. **Б.** Если переменная используется за пределами области видимости, компилятор аварийно завершит свою работу и выдаст сообщение об ошибке.

Упражнения

Если вам еще не надоело говорить о гремлинах, углубите свои познания по этой теме, выполнив следующие упражнения.

- Добавьте в класс `Gremlin` переменную `guid` с типом доступа `private` для хранения целочисленного значения. Создайте методы, которые возвращают значение переменной `guid` и изменяют ее значение только в том случае, когда оно находится в диапазоне 1 000 000–9 999 999.

- Напишите приложение Java, которое получает строковый аргумент, преобразует его в переменную типа `float`, затем преобразует ее в объект `Float` и, наконец, преобразует последний в переменную типа `int`. Выполните это приложение несколько раз с разными аргументами и посмотрите, как изменяются результаты.

ЗАНЯТИЕ 12
Повторное использование объектов

На этом занятии вы узнаете...

- как создавать суперклассы и подклассы;
- как формировать иерархию наследования;
- как переопределять методы.

Объекты Java идеально подходят для создания потомков. После создания объекта, представляющего собой набор атрибутов и поведения, в вашем распоряжении оказывается сущность, готовая передать свои качества потомству. Дочерние объекты наследуют атрибуты и поведение, присущие родительскому объекту. Но дочерние объекты также могут выполнять аналогичные операции иначе, чем родительские объекты.

Описанная выше система называется наследованием, и это то, что каждый суперкласс (родительский класс) передает своим подклассам (дочерним классам). Наследование является одним из наиболее важных аспектов объектно-ориентированного программирования (ООП), и именно оно является предметом рассмотрения данного занятия.

Еще один полезный аспект ООП заключается в возможности создания объекта, который можно использовать при работе с разными программами. Благодаря повторному использованию облегчается разработка надежных безошибочных программ.

Принципы наследования

В действительности наследование используется всякий раз, когда применяется один из стандартных классов Java, такой как `String` или `Integer`. Классы Java образуют пирамидальную иерархию, в которой все классы происходят от класса `Object`.

Класс объектов наследуется от всех суперклассов, которые находятся выше его в иерархии. Чтобы получить представление о том, как это работает, рассмотрим класс `InputStreamReader`, который является суперклассом для класса `FileReader`, считывающего текстовые строки из файла. Класс `FileReader`, соответственно, является подклассом класса `InputStreamReader`.

На рис. 12.1 показана часть генеалогического древа класса `InputStream Reader`. В каждом прямоугольнике находится класс, а линии соединяют суперкласс с подклассами, находящимися под ним.

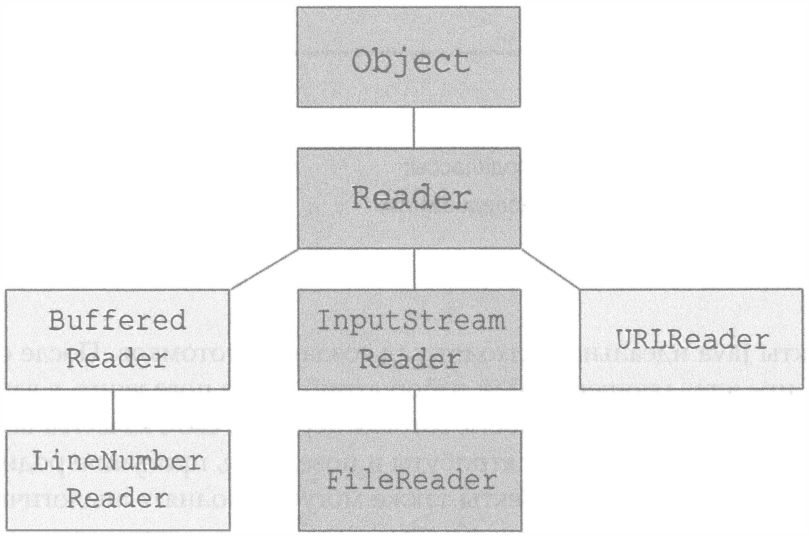


РИС. 12.1. Генеалогическое древо класса `InputStreamReader`

На вершине иерархии находится класс `Object`. Класс `InputStreamReader` имеет два суперкласса, находящиеся выше него в иерархии: `Reader` и `Object`. У него также есть один подкласс, находящийся ниже в иерархии: `FileReader`.

Класс `InputStreamReader` наследует атрибуты и поведение от классов `Reader` и `Object`, поскольку каждый из них находится непосредственно над ним в иерархии суперклассов. В то же время он не наследует ничего от классов `BufferedReader`, `UrlReader` и `LineNumberReader`, поскольку они не находятся над ним в иерархии.

Если все это кажется вам немного запутанным, то представьте иерархию как генеалогическое древо. Класс `InputStreamReader` наследуется от своего родительского класса и его предков, но не наследуется от своих "братьев" и "кузенов".

Создание нового класса сводится к следующему: нужно определить, чем новый класс отличается от существующего. Оставшаяся часть работы будет выполнена автоматически.

Наследование поведения и атрибутов

Поведение и атрибуты класса представляют собой комбинацию его собственного поведения и атрибутов, а также поведения и атрибутов, которые он наследует от своих суперклассов.

Ниже описана часть поведения и атрибутов класса `InputStreamReader`.

- Метод `equals()` определяет, имеет ли объект `InputStreamReader` то же значение, что и другой объект.
- Метод `getEncoding()` возвращает название кодировки символов, используемой потоком.
- Метод `reset()` перемещает текущую позицию потока обратно, т.е. в исходную точку.
- Метод `mark()` помечает текущую позицию.

Класс `InputStreamReader` может использовать все эти методы, при том что лишь метод `getEncoding()` не наследуется от родительского класса. Метод `equals()` определен в классе `Object`, а методы `reset()` и `mark()` наследуются от класса `Reader`.

Переопределение методов

Некоторые методы, которые определены в классе `InputStreamReader`, также определены в одном из его суперклассов. В качестве примера можно привести метод `close()`, который имеется в классах `InputStreamReader` и `Reader`. Если метод определен в подклассе и его суперклассе, то используется метод подкласса. Благодаря этому подкласс может изменять, замещать или полностью устранять определенные методы и атрибуты своих суперклассов.

Создание нового метода в подклассе для изменения поведения, унаследованного от суперкласса, называется *переопределением метода*. Необходимость в переопределении метода возникает в тех случаях, когда унаследованное поведение вызывает нежелательный результат либо нужно выполнить какое-то другое действие.

Настройка наследования

Класс определяется как подкласс другого класса с помощью ключевого слова `extends`, как показано в следующем примере.

```
class FileReader extends InputStreamReader {
   // Здесь находятся поведение и атрибуты
}
```

В данном случае класс `FileReader` объявляется потомком класса `InputStreamReader`. Все объекты чтения файлов должны быть потомками объектов чтения входного потока. Они используют функции этого класса для чтения файла из входного потока.

Один из методов, который класс `FileReader` может переопределить, — это метод `close()`, который выполняет операции, требуемые для закрытия файла после прочтения текста. Метод `close()`, реализованный классом `Reader`, наследуется вниз по иерархии вплоть до класса `FileReader`.

Чтобы переопределить метод, нужно объявить его таким же образом, как он был объявлен в суперклассе, из которого унаследован. Метод с доступом `public` должен оставаться общедоступным, значение, возвращаемое методом, должно быть того же типа, количество и тип аргументов метода тоже не должны меняться.

Сигнатура метода `read()` в классе `Reader` выглядит так:

```
public int read(char[] buffer) {
```

Если класс `FileReader` переопределяет этот метод, он должен начинаться так:

```
public int read(char[] cbuf) {
```

Единственное различие заключается в имени массива аргументов, что не имеет значения при определении того, созданы ли методы аналогичным образом. Эти две сигнатуры совпадают в силу справедливости следующих утверждений:

- оба метода имеют тип доступа `public`;
- оба метода возвращают целочисленное значение;
- оба метода используют массив типа `char` в качестве единственного аргумента.

В процессе переопределения метода в подклассе нужно предварять объявление метода директивой `@Override`, как показано в следующем примере.

```
@Override
public void setPosition(int x, int y) {
   if ( (x > 0) & (y > 0) ) {
      super.setPosition(x, y);
   }
}
```

Директива `@Override` является специальным комментарием, называемым *аннотацией*, которая сообщает компилятору Java все необходимые сведения. Благодаря наличию аннотации `@Override` компилятор получает возможность

проверить, действительно ли метод переопределяет метод суперкласса. Если указанные выше правила не соблюдены, например пропущен второй целочисленный аргумент, то программа не будет скомпилирована.

При отсутствии аннотации компилятор не сможет обнаружить эту потенциальную проблему. Метод, лишенный второго целочисленного аргумента, будет выглядеть подобно любому другому методу.

Использование ключевых слов this и super в подклассе

При работе с подклассами чрезвычайно полезны ключевые слова `this` и `super`.

Как вы помните из предыдущего занятия, ключевое слово `this` используется для ссылки на текущий объект. Если создается класс и нужно обратиться к конкретному объекту этого класса, можно указать ключевое слово `this`:

```
this.title = "Cagney";
```

Эта инструкция присваивает переменной экземпляра `title` текущего объекта значение `"Cagney"`.

Ключевое слово `super` позволяет сослаться на непосредственный суперкласс объекта. Его можно использовать несколькими способами:

- для ссылки на конструктор суперкласса, например `super("Adam", 12);`
- для ссылки на переменную суперкласса, например `super.hawaii = 50;`
- для ссылки на метод суперкласса, например `super.dragnet()`.

Один из вариантов использования ключевого слова `super` — в конструкторе подкласса. Поскольку подкласс наследует поведение и атрибуты своего суперкласса, необходимо связывать каждый конструктор подкласса с конструктором его суперкласса. Если этого не сделать, некоторые методы и атрибуты могут быть сконфигурированы некорректно, и подкласс будет функционировать неправильно.

Переменная, доступ к которой осуществляется с помощью ключевого слова `super`, не должна быть объявлена как `private`, иначе она станет недоступна в подклассе.

Чтобы связать конструкторы, в качестве первой инструкции конструктора подкласса нужно вызвать конструктор суперкласса. При этом нужно использовать ключевое слово `super`, как показано ниже.

```
public DataReader(String name, int length) {
   super(name, length);
}
```

Это пример конструктора подкласса, который использует инструкцию `super(name, length)` для вызова аналогичного конструктора своего суперкласса.

Если не использовать метод `super()` для вызова конструктора суперкласса, Java автоматически вызывает метод `super()` без аргументов в начале конструктора подкласса. Если у суперкласса нет такого конструктора или он ведет себя непредсказуемо, могут возникать ошибки, поэтому лучше вызывать конструктор суперкласса самостоятельно.

Использование существующих объектов

В ООП поощряется повторное использование объектов. Если вы разрабатываете объект для проекта Java, необходимо предусмотреть возможность использовать его в другом проекте Java без каких-либо изменений.

Когда класс Java хорошо спроектирован, он может применяться в других программах. Чем больше готовых объектов доступно в программе, тем проще ее разрабатывать. Если, например, существует превосходно работающий объект проверки орфографии и грамматики, лучше включить его в программу в готовом виде, чем разрабатывать свой собственный объект.

Когда язык Java только появился, механизмы совместного использования объектов в основном были неформальными. Программисты старались делать свои объекты максимально независимыми и защищали их от неправомерного использования с помощью закрытых переменных и общедоступных методов, применяемых для чтения и записи этих переменных.

Но лучше, когда имеется стандартный подход к разработке повторно используемых объектов (как, например, в случае JavaBeans). Ниже описаны преимущества стандартного подхода.

- Не нужно документировать всю функциональность объекта, поскольку каждый, кто ознакомлен со стандартным подходом, уже многое знает о работе объекта.
- Можно создавать стандартизированные инструменты разработки, которые облегчают работу с такими объектами.
- Два объекта, соответствующих стандарту, могут взаимодействовать друг с другом, не требуя специального программирования, направленного на достижение совместимости.

Для объектов JavaBeans существуют строгие правила создания и использования. Класс Java, который следует этим правилам, называется *bean-компонентом*. Эти компоненты могут создаваться в среде NetBeans. Вводный курс

по использованию объектов JavaBeans в среде NetBeans доступен по следующему адресу:

```
http://wiki.netbeans.org/NetBeansJavaBeansTutorial
```

Хранение объектов одного класса в списках массивов

Прежде чем приступить к написанию программы, следует выбрать хранилище данных. На предыдущих занятиях были рассмотрены следующие информационные хранилища:

- простые типы данных, такие как `int` и `char` (или же соответствующие им классы);
- массивы;
- объекты типа `String`.

В действительности существует гораздо больше информационных хранилищ, поскольку данные могут храниться в любом классе Java. Одним из наиболее полезных в этом плане будет класс `ArrayList`. Эта структура данных применяется для хранения объектов, относящихся к одному классу или общему суперклассу.

Как следует из названия, *списки массивов* подобны массивам в том плане, что могут хранить однотипные элементы. Основное же отличие заключается в том, что размеры списков могут динамически изменяться. Их еще называют *динамическими массивами*.

Класс `ArrayList` входит в пакет `java.util`, который является одним из наиболее часто используемых в библиотеке классов Java. Чтобы подключить его к программе, необходимо задать следующую инструкцию `import`:

```
import java.util.ArrayList;
```

Список массивов содержит объекты, которые принадлежат к одному классу или относятся к одному и тому же суперклассу. Такие списки создаются путем указания двух классов: класса `ArrayList` и класса, объекты которого будут содержаться в списке.

Имя класса объектов заключается между символами < и >, как показано в следующем примере:

```
ArrayList<String> structure = new ArrayList<String>();
```

Эта инструкция создает список строк. При подобной идентификации класса применяются обобщенные типы, с помощью которых задаются типы

объектов, хранящихся в структурах данных наподобие списков массивов. В старых версиях Java требовалось применять следующий конструктор:

```
ArrayList structure = new ArrayList();
```

Можно и сейчас так поступать, но благодаря использованию обобщенных типов повышается надежность кода, поскольку появление многих ошибок предотвращается еще на этапе компиляции. В частности, компилятор не позволит включить в список массивов объекты неподходящего класса. Если, например, попытаться добавить объект `Integer` в список, предназначенный для хранения объектов `String`, компилятор выдаст сообщение об ошибке.

В отличие от массива, список массивов не создается с фиксированным количеством элементов. Изначально резервируются 10 элементов. Если заранее известно, что в списке будет храниться больше объектов, можно в качестве аргумента конструктора указать предельный размер списка. Вот инструкция, которая создает список из 300 элементов:

```
ArrayList<String> structure = new ArrayList<String>(300);
```

Чтобы добавить объект в список, нужно вызвать метод `add()`. При этом в качестве единственного аргумента указывается добавляемый объект.

```
structure.add("Василий");
structure.add("Валентин");
structure.add("Валентина");
```

Объекты добавляются по порядку, так что, если это были первые три добавленных объекта, строка `"Василий"` станет элементом 0, строка `"Валентин"` — элементом 1, а строка `"Валентина"` — элементом 2.

Чтобы выбрать элемент из списка, вызовите метод `get()`, указав в качестве аргумента индекс элемента:

```
String name = structure.get(1);
```

Эта инструкция сохраняет элемент `"Валентин"` в переменной `name`.

Чтобы узнать, содержится ли объект в качестве одного из элементов списка, вызовите метод `contains()`, указав требуемый объект в качестве аргумента.

```
if (structure.contains("Валентина")) {
   System.out.println("Найдена Валентина");
}
```

Чтобы удалить элемент из списка, укажите сам элемент или его индекс.

```
structure.remove(0);
structure.remove("Валентин");
```

После выполнения этих инструкций в качестве единственной строки в списке останется "Валентина".

Циклический обход списка массивов

В Java имеется специальный тип цикла for, который облегчает загрузку списка массивов и получение доступа к каждому из элементов этого списка. Этот цикл состоит из двух частей, а не трех, в отличие от циклов for, которые рассматривались на занятии 8.

Первая часть представляет собой раздел инициализации. В нем указывается класс и имя переменной, в которой будет храниться каждый объект, извлекаемый из списка. Эта переменная должна быть того же класса, что и содержимое списка.

Вторая часть цикла идентифицирует сам список.

Ниже представлен пример кода, в котором осуществляется циклический обход списка structure, и каждое содержащееся в нем имя отображается на экране.

```
for (String name : structure) {
   System.out.println(name);
}
```

Первым на данном занятии будет создан проект StringLister. Это приложение работает со списками массивов, для обработки которых применяется специальный тип цикла for. Приложение отображает список строк в алфавитном порядке. Список формируется на основании массива и аргументов командной строки.

В среде NetBeans откройте проект Java24 и выполните команду **Файл⇨ Создать файл**. Затем создайте новый пустой класс Java с именем StringLister и включите его в пакет com.java24hours. В редакторе исходного кода введите код из листинга 12.1 и сохраните полученный файл StringLister.java.

ЛИСТИНГ 12.1. Исходный код программы **StringLister.java**

```
 1: package com.java24hours;
 2:
 3: import java.util.*;
 4:
 5: public class StringLister {
 6:    String[] names = { "Андрей", "Борис", "Татьяна", "Светлана",
 7:       "Наталья", "Александр", "Дмитрий", "Мария", "Нина" };
 8:
 9:    public StringLister(String[] moreNames) {
10:       ArrayList<String> list = new ArrayList<String>();
```

```
11:        for (int i = 0; i < names.length; i++) {
12:           list.add(names[i]);
13:        }
14:        for (int i = 0; i < moreNames.length; i++) {
15:           list.add(moreNames[i]);
16:        }
17:        Collections.sort(list);
18:        for (String name : list) {
19:           System.out.println(name);
20:        }
21:     }
22:
23:     public static void main(String[] arguments) {
24:        StringLister lister = new StringLister(arguments);
25:     }
26: }
```

Прежде чем запустить приложение, выберите команду **Выполнить⇨Установить конфигурацию проекта⇨Изменить** и задайте `com.java24hours.StringLister` в качестве главного класса. Введите также один или несколько аргументов командной строки, разделенных пробелами, например **Максим Тамара Семен**. После этого выполните команду **Выполнить⇨Запустить главный проект** и посмотрите на полученные результаты.

Имена, указанные в командной строке, добавляются к именам, которые хранятся в массиве (строки 6 и 7). Поскольку общее количество имен неизвестно до момента запуска программы, строки лучше хранить в списке массивов, а не в обычном массиве.

Строки списка сортируются в алфавитном порядке с помощью метода `sort()` класса `Collections`.

```
Collections.sort(list);
```

Этот класс, как и `ArrayList`, входит в пакет `java.util`. В Java списки массивов и другие подобные структуры данных называются *коллекциями*.

В результате запуска программы отобразится список имен, отсортированных в алфавитном порядке (рис. 12.2). Благодаря динамически изменяемому размеру массива в структуру данных можно добавлять дополнительные имена, а затем сортировать их наравне с другими именами.

Создание подкласса

Чтобы показать, как работает наследование, в следующем проекте мы создадим класс `Point3D`, который представляет точку в трехмерном пространстве. Точка в двумерном пространстве задается с помощью координат

(x,y). Но в трехмерном пространстве добавляется третья координата, которая обычно называется *z*.

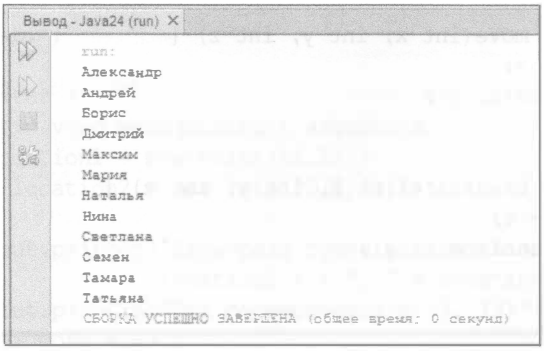


РИС. 12.2. Результат выполнения программы `StringLister`

Класс `Point3D` может выполнять следующее:

- отслеживать координаты объекта *(x,y,z)*;
- перемещать объект в точку, определяемую новыми координатами *(x,y,z)*;
- перемещать объект на расстояние, определяемое значениями *x*, *y* и *z*.

В Java есть стандартный класс `Point`, который представляет двумерные точки и находится в пакете `java.awt`. Он содержит две целочисленные переменные, `x` и `y`, которые хранят местоположение *(x,y)* объектов класса `Point`. Он также включает метод `move()`, перемещающий точку в указанное место, и метод `translate()`, который перемещает объект на расстояние, заданное значениями `x` и `y`.

В среде NetBeans выберите проект `Java24`, создайте новый пустой класс Java с именем `Point3D` и включите его в пакет `com.java24hours`. В редакторе исходного кода введите код из листинга 12.2 и сохраните полученный файл `Point3D.java`.

ЛИСТИНГ 12.2. Исходный код программы `Point3D.java`

```
1: package com.java24hours;
2:
3: import java.awt.*;
4:
5: public class Point3D extends Point {
6:     public int z;
7:
8:     public Point3D(int x, int y, int z) {
```

```
 9:        super(x,y);
10:        this.z = z;
11:     }
12:
13:     public void move(int x, int y, int z) {
14:        this.z = z;
15:        super.move(x, y);
16:     }
17:
18:     public void translate(int x, int y, int z) {
19:        this.z += z;
20:        super.translate(x, y);
21:     }
22: }
```

Класс `Point3D` не включает метод `main()`, поэтому вы не сможете запустить его как самостоятельное приложение, но зато его можно использовать в Java-программах, моделирующих трехмерные точки.

Класс `Point3D` должен выполнять лишь ту работу, которая не выполняется его суперклассом `Point`. Эта работа прежде всего включает отслеживание целочисленной переменной `z` и ее использование в качестве аргумента метода `move()`, метода `translate()` и конструктора `Point3D()`.

Во всех методах используются ключевые слова `super` и `this`. Ключевое слово `this` применяется для ссылки на текущий объект `Point3D`, поэтому инструкция `this.z = z` в строке 10 устанавливает переменную экземпляра `z` равной значению `z`, передаваемому в качестве аргумента в строке 8.

Ключевое слово `super` ссылается на суперкласс текущего объекта: `Point`. Оно применяется для присваивания значений переменных и вызова методов, которые наследуются классом `Point3D`. Инструкция `super(x,y)` в строке 9 вызывает конструктор `Point(x,y)` суперкласса, который тем самым назначает координаты *(x,y)* объекта `Point3D`. Поскольку класс `Point` уже умеет обрабатывать координаты осей *x* и *y*, было бы излишним добавлять подобную функциональность в класс `Point3D`.

Чтобы протестировать новый класс `Point3D`, создайте программу, которая использует объекты `Point` и `Point3D`, выполняя их перемещение. Создайте новый класс `PointTester` в среде NetBeans, включите его в пакет `com.java24hours`, введите код из листинга 12.3 и сохраните полученный файл `PointTester.java`. После сохранения файл будет скомпилирован автоматически.

ЛИСТИНГ 12.3. Исходный код программы `PointTester.java`

```
 1: package com.java24hours;
 2:
 3: import java.awt.*;
 4:
 5: class PointTester {
 6:    public static void main(String[] arguments) {
 7:       Point location1 = new Point(11,22);
 8:       Point3D location2 = new Point3D(7,6,64);
 9:
10:       System.out.println("Двумерная точка находится в (" +
11:                          location1.x + ", " + location1.y + ")");
12:       System.out.println("Она перемещается в (4, 13)");
13:       location1.move(4,13);
14:       System.out.println("Теперь двумерная точка находится в (" +
15:                          location1.x + ", " + location1.y + ")");
16:       System.out.println("Она перемещена на -10 единиц по оси " +
17:                          "x и по оси y");
18:       location1.translate(-10,-10);
19:       System.out.println("Теперь двумерная точка находится в (" +
20:                          location1.x + ", " + location1.y + ")\n");
21:
22:       System.out.println("Трехмерная точка находится в (" +
23:                          location2.x + ", " + location2.y + ", " +
24:                          location2.z + ")");
25:       System.out.println("Она перемещается в (10, 22, 71)");
26:       location2.move(10,22,71);
27:       System.out.println("Теперь трехмерная точка находится в (" +
28:                          location2.x + ", " + location2.y + ", " +
29:                          location2.z + ")");
30:       System.out.println("Она перемещена на -20 единиц по осям " +
31:                          "x, y и оси z");
32:       location2.translate(-20,-20,-20);
33:       System.out.println("Теперь трехмерная точка находится в (" +
34:                          location2.x + ", " + location2.y + ", " +
35:                          location2.z + ")");
36:    }
37: }
```

После запуска программы с помощью команд **Выполнить⇨Выполнить файл** будет получен результат, показанный на рис. 12.3. Если же программа не выполняется, посмотрите, не появился ли красный предупреждающий значок в левой части окна исходного кода, свидетельствующий о наличии ошибки.

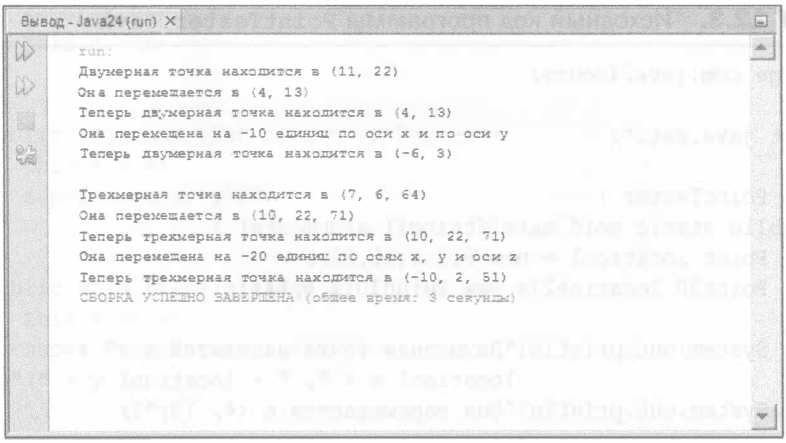


РИС. 12.3. Результат выполнения программы `PointTester`

Резюме

То, как люди представляют себе таинство рождения, существенно отличается от того, как порождаются подклассы в Java и как наследуются атрибуты и поведение в иерархии классов. Если бы люди рождались так же, как объекты в ООП, то каждый потомок Моцарта мог бы стать великим композитором, а все наследники Марка Твена могли бы описывать приключения Тома Сойера и Гекльберри Финна. И вы бы могли без малейшего труда использовать все умения и навыки, присущие вашим предкам.

Конечно, наследование в ООП устроено гораздо проще, чем таинство рождения живых существ. Тем не менее оно обеспечивает возможность разработки программного обеспечения без лишних усилий.

Вопросы и ответы

В. В созданных нами ранее программах на Java не использовалось ключевое слово `extends`, означающее наследование от суперкласса. Означает ли это, что данные программы не входят в иерархию классов?

О. Все классы, созданные в Java, входят в иерархию классов, поскольку при отсутствии ключевого слова `extends` по умолчанию используется суперкласс `Object`. Методы `equals()` и `toString()`, относящиеся ко всем классам, являются частью поведения, которое автоматически наследуется от суперкласса `Object`.

Коллоквиум

Проверьте знание материала, рассмотренного на этом занятии, ответив на следующие вопросы.

Контрольные вопросы

1. Если метод суперкласса неприемлем для подкласса, то что можно сделать в этом случае?
   А. Удалить метод из суперкласса.
   Б. Переопределить метод в подклассе.
   В. Написать гневное письмо разработчикам.
2. Какие методы можно использовать для выборки элемента из списка массивов?
   А. `get()`.
   Б. `read()`.
   В. `elementAt()`.
3. Какое ключевое слово можно использовать для ссылки на методы и переменные текущего объекта?
   А. `this`.
   Б. `that`.
   В. `theOther`.

Ответы

1. **Б.** Поскольку в Java разрешается переопределять методы, вам не понадобится изменять способ его работы суперкласса.
2. **А.** Метод `get()` имеет лишь один аргумент — индекс элемента.
3. **А.** Ключевое слово `this` ссылается на текущий объект.

Упражнения

Чтобы закрепить и расширить свои познания в области наследования объектов, выполните следующие упражнения.

- Создайте класс `Point4D`, который добавляет координату *t* к системе координат *(x,y,z)*, созданной классом `Point3D`. Координата *t* означает время, поэтому убедитесь в том, что она всегда положительна.
- Создайте иерархию классов, которая отражает характеристики игроков футбольной команды в зависимости от их позиции на поле: вратарь,

защитник, полузащитник и нападающий. Причем общие характеристики должны находиться выше в иерархии. Например, число забитых голов должно наследоваться классами `Защитник`, `Полузащитник` и `Нападающий`, а число пропущенных голов — только классом `Вратарь`.

Часть IV
Продвинутые методики программирования

В этой части...

- Занятие 13 Хранение объектов в структурах данных
- Занятие 14 Обработка ошибок в программе
- Занятие 15 Создание многопоточной программы
- Занятие 16 Использование внутренних классов и замыканий

ЗАНЯТИЕ 13
Хранение объектов в структурах данных

На этом занятии вы узнаете...

- как создавать списки массивов;
- как добавлять и удалять элементы списка;
- как применять обобщенные классы в списках;
- как найти объект в списке;
- как просмотреть содержимое списка;
- как создать хеш-карту ключей и значений.

В компьютерном программировании много времени тратится на сбор информации и поиск места для ее хранения. Информация может быть представлена примитивным типом данных, таким как `float`, либо в виде объекта определенного класса. Данные могут считываться с диска, запрашиваться с сервера, вводиться пользователем или собираться другими способами.

После того как информация окажется в вашем распоряжении, следует подумать, куда ее поместить, пока программа выполняется в виртуальной машине Java. Элементы, имеющие одинаковый тип данных или относящиеся к одному классу, можно хранить в массиве. Этого вполне достаточно для многих целей, но по мере усложнения программ потребности в упорядоченном хранении данных будут расти.

На этом занятии вы узнаете о двух структурах Java, предназначенных для хранения информации: это списки массивов и хеш-карты.

Списки массивов

На занятии 9 вы ознакомились с массивами — чрезвычайно удобным средством работы с группами переменных и объектов в программах. Массивы объединяют элементы, которые относятся к одному и тому же типу или классу.

Несмотря на всю полезность массивов, они все же довольно ограничены, так как их размеры не изменяются. Если массив предназначен для хранения 90 элементов, то его невозможно приспособить для хранения большего или меньшего числа элементов, поскольку его размер фиксирован.

В пакет `java.util` включен класс `ArrayList`, который позволяет снять все ограничения массива.

Список массивов — это структура данных, которая включает объекты того же класса или общего суперкласса. Размер списка может изменяться в процессе выполнения программы.

Простейший способ создания списка массивов заключается в вызове соответствующего конструктора без аргументов:

```
ArrayList servants = new ArrayList();
```

Списки массивов можно создавать, указывая начальный размер, который дает некоторое представление о количестве элементов, планируемых для включения в список. Размер задается в виде целочисленного аргумента конструктора, как показано в следующей инструкции, в которой размер массива равен 30:

```
ArrayList servants = new ArrayList(30);
```

Описанный процесс во многом напоминает создание массива и определение его точного размера, хотя в данном случае размер не задается точно. Если изначально заданный размер списка превышается, выполняется соответствующая перестройка списка. Просто чем точнее изначально будет оценен размер списка, тем эффективнее будут расходоваться ресурсы программы.

Список массивов хранит объекты, которые относятся к одному и тому же классу или совместно используют один и тот же суперкласс.

При создании списка массивов обычно известен класс или суперкласс хранящихся в нем элементов. Имя класса можно указать в угловых скобках (< и >) при вызове конструктора. Это называется *обобщенным типом*. Ниже показан вызов конструктора списка, в котором хранятся объекты `String`.

```
ArrayList<String> servants = new ArrayList<String>();
```

Эта инструкция создает список массивов, содержащий объекты `String`. Только объекты этого класса или его подклассов могут храниться в списке. Если же элемент будет иметь другой тип, компилятор сообщит об ошибке.

Чтобы добавить объект в список, вызовите метод `add(`*`объект`*`)`, указав требуемый объект в качестве аргумента. Следующие инструкции добавляют в список пять строк.

```
servants.add("Борис");
servants.add("Анна");
```

```
servants.add("Михаил");
servants.add("Василий");
servants.add("Дмитрий");
```

Каждый элемент добавляется в конец списка, поэтому первая строка в списке `servants` будет `"Борис"`, а последняя — `"Дмитрий"`.

Существует также метод `remove(`*объект*`)`, который удаляет объект из списка:

```
servants.remove("Василий");
```

Размер списка массивов соответствует количеству находящихся в нем элементов. Чтобы получить эту информацию, вызовите метод `size()`, который возвращает целое число.

```
int servantCount = servants.size();
```

Если класс элементов списка задавался с помощью обобщенного типа, то можно использовать цикл `for` для циклического обхода всех элементов.

```
for (String servant : servants) {
   System.out.println(servant);
}
```

Первый параметр, задаваемый в инструкции цикла, — это переменная, в которой будут храниться элементы, второй параметр — сам список массивов.

Метод `add(`*объект*`)` добавляет объект в конец списка. Можно также указать позицию, в которой должен храниться объект. При этом используется синтаксис `add(`*позиция*`, `*объект*`)`, где первый аргумент должен быть целым числом.

```
ArrayList<String> aristocrats = new ArrayList<String>();
aristocrats.add(0, "Лорд Роберт");
aristocrats.add(1, "Леди Мэри");
aristocrats.add(2, "Леди Эдит");
aristocrats.add(3, "Леди Сибил");
aristocrats.add(0, "Леди Джуди");
```

Последняя инструкция добавляет запись `"Леди Джуди"` не в конец списка, а в начало, выше записи `"Лорд Роберт"` и остальных.

Номер позиции, указанный в качестве первого аргумента, не должен превышать значение, возвращаемое методом `size()`. Если, например, попытаться добавить запись `"Лорд Роберт"` в позицию 1, а не 0, то программа завершится аварийно, сгенерировав исключение `IndexOutOfBoundsException`.

Чтобы удалить конкретный элемент из списка, нужно указать его позицию в качестве аргумента метода `remove(`*позиция*`)`:

```
aristocrats.remove(4);
```

Чтобы осуществить выборку элемента в заданной позиции списка, нужно вызвать метод `get(позиция)`. Вот цикл, который осуществляет выборку каждой строки списка с последующим ее отображением.

```
for (int i = 0; i < aristocrats.size(); i++) {
   String aristocrat = aristocrats.get(i);
   System.out.println(aristocrat);
}
```

Зачастую нужно выяснить, содержит ли список массивов определенный объект. Для этого нужно вызвать метод `indexOf(объект)`, который возвращает позицию указанного объекта или `-1`, если объект отсутствует в списке.

```
int hasCarson = servants.indexOf("Carson");
```

Также имеется метод `contains()`, который возвращает значение `true` или `false`, в зависимости от того, найден ли указанный объект. Следующая инструкция позволяет определить, содержится ли объект `Carson` в списке.

```
boolean hasCarson = servants.contains("Carson");
```

Первый проект данного занятия представляет собой вариант игры "Морской бой". Суть игры заключается в том, что игрок производит "выстрелы" по точкам с координатами *(x,y)* в пределах поля размером 10×10 клеток. В некоторых точках расположены цели — "боевые корабли", другие точки являются пустыми.

Цели представлены с помощью класса `Point` из пакета `java.awt`. Точка создается путем вызова конструктора `Point(целое_число, целое_число)`, в качестве аргументов которого указываются координаты *x* и *y*.

Следующая инструкция создает точку с координатами *(5,9)*:

```
Point p1 = new Point(5,9);
```

На рис. 13.1 показано поле размером 9×9, на котором эта точка обозначена символом `X`, а пустые места — символами `'.'`.

РИС. 13.1. Игровое поле размером 9×9

Столбцы следуют слева направо и представляют координату *x*, а строки — сверху вниз и представляют координату *y*.

Ранее было рассмотрено, каким образом хранить строки в списках массивов. Но эти структуры могут также хранить класс `Point` или любой другой класс объектов. Следующая инструкция создает список точек:

```
ArrayList<Point> targets = new ArrayList<Point>();
```

Компилятор Java не позволит добавить в этот список объекты иного класса, чем `Point` или его подклассы.

В NetBeans или другой среде программирования создайте класс Java с именем `Battlepoint` и включите его в пакет `com.java24hours`. В редакторе исходного кода введите код из листинга 13.1 и сохраните полученный файл `Battlepoint.java`.

ЛИСТИНГ 13.1. Исходный код программы `Battlepoint.java`

```
 1: package com.java24hours;
 2:
 3: import java.awt.*;
 4: import java.util.*;
 5:
 6: public class Battlepoint {
 7:    ArrayList<Point> targets = new ArrayList<Point>();
 8:
 9:    public Battlepoint() {
10:       // Создание целей
11:       createTargets();
12:       // Отображение игрового поля
13:       showMap();
14:       // Стрельба по трем точкам
15:       shoot(7,4);
16:       shoot(3,3);
17:       shoot(9,2);
18:       // Повторное отображение игрового поля
19:       showMap();
20:    }
21:
22:    private void showMap() {
23:       System.out.println("\n   1  2  3  4  5  6  7  8  9");
24:       for (int row = 1; row < 10; row++) {
25:          for (int column = 1; column < 10; column++) {
26:             if (column == 1) {
27:                System.out.print(row + " ");
28:             }
29:             System.out.print(" ");
30:             Point cell = new Point(column, row);
31:             if (targets.indexOf(cell) > -1) {
```

```
32:                // Цель находится в этой точке
33:                System.out.print("X");
34:             } else {
35:                // Цель отсутствует
36:                System.out.print(".");
37:             }
38:             System.out.print(" ");
39:          }
40:          System.out.println();
41:       }
42:       System.out.println();
43:    }
44:
45:    private void createTargets() {
46:       Point p1 = new Point(5,9);
47:       targets.add(p1);
48:       Point p2 = new Point(4,5);
49:       targets.add(p2);
50:       Point p3 = new Point(9,2);
51:       targets.add(p3);
52:    }
53:
54:    private void shoot(int x, int y) {
55:       Point shot = new Point(x,y);
56:       System.out.print("Стреляем по (" + x + "," + y + ") ... ");
57:       if (targets.indexOf(shot) > -1) {
58:          System.out.println("Вы потопили боевой корабль!");
59:          // Удаление потопленной цели
60:          targets.remove(shot);
61:       } else {
62:          System.out.println("Промах.");
63:       }
64:    }
65:
66:    public static void main(String[] arguments) {
67:       new Battlepoint();
68:    }
69: }
```

Комментарии в приложении `Battlepoint` описывают каждую часть конструктора и важные части условной логики программы.

Приложение создает цели в виде трех объектов `Point` и добавляет их в массив (строки 45–52). Эти цели отображаются на игровом поле (строки 22–43).

Затем производятся выстрелы по трем точкам путем вызова метода `shoot(`*целое_число*`,` *целое_число*`)` (строки 54–64). После каждого выстрела приложение сообщает, была ли поражена цель. Если цель поражена, она удаляется из списка массивов.

По окончании игровое поле отображается снова, и приложение завершает работу. Результат выполнения приложения показан на рис. 13.2.

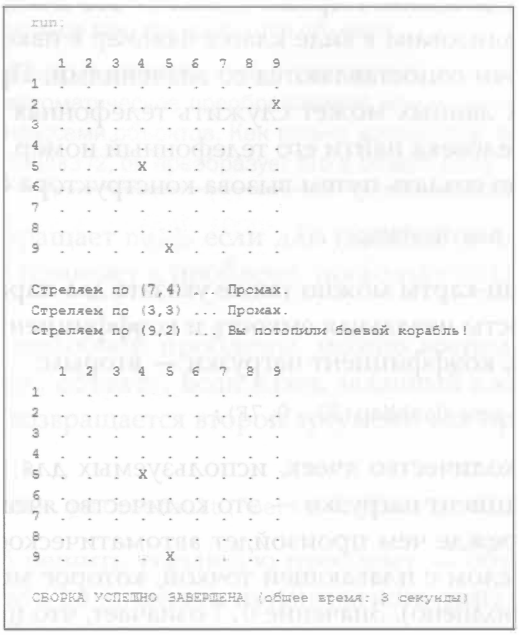


РИС. 13.2. Включение объектов `Point` в список массивов и стрельба по ним

На рис. 13.2 можно увидеть три цели на верхнем игровом поле. Одна цель была успешно поражена. Нижнее поле отражает это изменение.

После поражения цели она удаляется из списка целей путем вызова метода `remove(`*объект*`)`, в качестве аргумента которого задается точка выстрела.

ПРИМЕЧАНИЕ

В методе `shoot()` приложения `Battlepoint` объект `Point`, удаляемый из списка массивов, совпадает с объектом, представляющим выстрел. Этот объект имеет те же координаты *(x,y)*, что и цель, по которой была произведена стрельба.

Хеш-карты

Зачастую в программировании одна часть информации используется для доступа к другой части. Простейший пример — список массивов, представляющий собой одну из разновидностей структур данных, где индекс применяется для выборки объекта из списка. Ниже представлен пример, в котором из списка массивов `aristocrats` выбирается первая строка.

```
String first = aristocrats.get(0);
```

Индексы также применяются в массивах для доступа к их элементам.

Хеш-карты — это структуры данных в Java, которые используют объекты для выборки других объектов. Первый объект — это *ключ*, второй — *значение*. Такая структура реализована в виде класса `HashMap` в пакете `java.util`.

В хеш-карте ключи сопоставляются со значениями. Примером подобных структурированных данных может служить телефонная книга, которая позволяет по имени человека найти его телефонный номер.

Хеш-карту можно создать путем вызова конструктора без аргументов:

```
HashMap phonebook = new HashMap();
```

При создании хеш-карты можно также указать два параметра, определяющих ее эффективность: начальная емкость и коэффициент нагрузки. Емкость указывается первой, коэффициент нагрузки — вторым:

```
HashMap phonebook = new HashMap(30, 0.7F);
```

Емкость — это количество ячеек, используемых для хранения значений хеш-карты. Коэффициент нагрузки — это количество ячеек, которые должны быть заполнены, прежде чем произойдет автоматическое увеличение емкости. Он задается числом с плавающей точкой, которое может изменяться от `0` (пусто) до `1.0` (заполнено). Значение `0.7` означает, что при заполнении 70% ячеек емкость хранилища увеличивается. По умолчанию емкость равна `16`, а коэффициент нагрузки — `0.75`. В большинстве случаев этого достаточно.

Для обозначения классов для ключей и значений следует использовать обобщенные типы. Они указываются между символами < и > и разделяются запятыми, как показано в следующем примере:

```
HashMap<String, Long> phonebook = new HashMap<>();
```

Эта инструкция создает хеш-карту `phonebook` с ключами строкового типа и значениями типа `Long`. Вторая запись в угловых скобках < и > пуста, поскольку предполагается использование тех же классов, что были указаны в предыдущем случае.

Чтобы сохранить объекты в хеш-карте, нужно вызвать метод `put(`*`объект`*`, `*`объект`*`)` с двумя аргументами: ключом и значением.

```
phonebook.put("Butterball Turkey Line", 8002888372L);
```

В результате в хеш-карту записывается элемент с ключом `"Butterball Turkey Line"` и значением `8002888372` типа `Long` (телефонный номер данной службы).

Для выборки объекта из хеш-карты по ключу используется метод `get(`*`объект`*`)`. Здесь ключ является единственным аргументом метода.

```
long number = phonebook.get("Butterball Turkey Line");
```

ПРИМЕЧАНИЕ

В этом примере объект типа `Long` сохраняется в хеш-карте в виде значения типа `long`. В ранних версиях Java это привело бы к ошибке, поскольку примитивные типы данных, такие как `long`, не могли использоваться там, где требуются объекты.

Благодаря появлению функций автоупаковки и распаковки эта ошибка была устранена. Теперь Java выполняет автоматическое преобразование между примитивными типами данных и эквивалентными классами объектов. Как только компилятор Java встречает значение типа `long`, например `8002888372`, он преобразует его в объект `Long`.

Метод `get()` возвращает `null`, если для указанного ключа нет значения. В данном случае это приведет к проблеме, поскольку `null` не является значением типа `long`.

Чтобы избежать подобной проблемы, можно воспользоваться методом `getOrDefault(`*объект*`, `*объект*`)`. Если ключ, заданный в качестве первого аргумента, не найден, возвращается второй аргумент, как продемонстрировано ниже:

```
long number = phonebook.getOrDefault("Betty Crocker", -1L);
```

Еще один способ решить указанную проблему — объявить переменную `number` в виде объекта `Long` вместо примитивного типа `long`. Для объекта `Long` значение `null` будет допустимым.

Если в хеш-карте найдено число, соответствующее ключу `"Betty Crocker"`, оно будет возвращено, в противном случае возвращается `-1`.

Методы `containsKey(`*объект*`)` и `containsValue(`*объект*`)` сообщают о наличии указанного ключа или значения в карте и возвращают булево значение `true` или `false`.

Хеш-карты, как и списки массивов, включают метод `size()`, который позволяет подсчитать количество элементов в структуре данных.

Для циклического просмотра хеш-карты можно использовать набор записей, представляющий собой коллекцию всех записей карты. Метод `entrySet()` возвращает все записи в виде объекта `Set` (интерфейс `Set` входит в пакет `java.util`).

Каждая запись в наборе представляется с помощью внутреннего класса `Map.Entry`, находящегося в классе `Map` из пакета `java.util`. При наличии объекта `Entry` можно вызвать его метод `getKey()` для получения ключа и метод `getValue()` для получения значения.

В следующем цикле `for` наборы и записи используются для получения доступа ко всем ключам и значениям в хеш-карте `phonebook`.

```
for (Map.Entry<String, Long> entry : phonebook.entrySet()) {
   String key = entry.getKey();
```

```
    Long value = entry.getValue();
    // ...
}
```

В нашем следующем проекте FontMapper хеш-карта применяется для управления коллекцией шрифтов. Класс Font из пакета java.awt предназначен для создания шрифтов и отображения текста в графическом интерфейсе пользователя. Описание шрифта включает его имя, размер (в пунктах) и начертание (обычный, полужирный или курсив).

Хеш-карты могут содержать объекты любого класса. В среде NetBeans создайте новый класс Java с именем FontMapper, включите его в пакет com.java24hours, введите код из листинга 13.2 и сохраните полученный файл FontMapper.java.

ЛИСТИНГ 13.2. Исходный код программы FontMapper.java

```
 1: package com.java24hours;
 2:
 3: import java.awt.*;
 4: import java.util.*;
 5:
 6: public class FontMapper {
 7:    public FontMapper() {
 8:       Font times = new Font("Times New Roman", Font.BOLD, 12);
 9:       Font verdana = new Font("Verdana", Font.ITALIC, 25);
10:       Font courier = new Font("Courier New", Font.PLAIN, 6);
11:       HashMap<String, Font> fonts = new HashMap<>();
12:       fonts.put("Основной текст", times);
13:       fonts.put("Заголовок", verdana);
14:       fonts.put("Мелкий шрифт", courier);
15:       for (Map.Entry<String, Font> entry : fonts.entrySet()) {
16:          String key = entry.getKey();
17:          Font value = entry.getValue();
18:          System.out.println(key + ": " + value.getSize() +
19:                             "-pt " + value.getFontName());
20:       }
21:    }
22:
23:    public static void main(String[] arguments) {
24:       new FontMapper();
25:    }
26: }
```

Приложение FontMapper создает три объекта Font (строки 8–10) и добавляет их в хеш-карту fonts (строки 12–14). Объекты шрифтов хранятся в карте вместе со строковыми ключами, описывающими назначение шрифтов: "Основной текст", "Заголовок" и "Мелкий шрифт".

В цикле `for` (строки 15–20) выполняется обход хеш-карты с помощью набора записей.

Результат выполнения этого приложения показан на рис. 13.3.

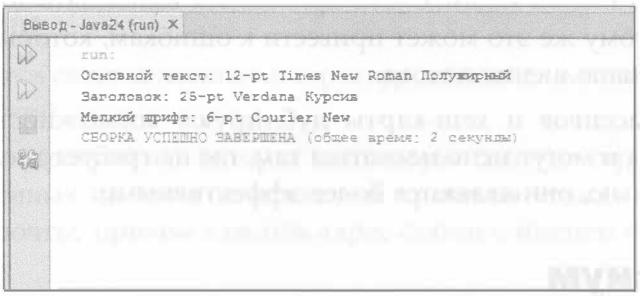


РИС. 13.3. Объекты `Font` хранятся в хеш-карте

Резюме

Списки массивов и хеш-карты — это две чаще всего используемые структуры данных из пакета `java.util`. Класс списка массивов расширяет функциональные возможности массивов, позволяя преодолеть ограничения, связанные с фиксированным размером массива. Хеш-карты обеспечивают возможность применения любого объекта в качестве ключа для выборки значения.

Существуют и другие типы структур данных, в частности, битовые наборы (класс `BitSet`), в которых хранятся значения битов 0 и 1; стеки (класс `Stack`), которые представляют собой списки элементов, работающие по принципу "последним пришел — первым ушел"; отсортированные структуры `TreeMap` и `TreeSet`, а также свойства (класс `Properties`), представляющие собой специализированную хеш-карту, которая хранит конфигурационные настройки программы в файле или другом постоянном хранилище.

Вопросы и ответы

В. Что такое синхронизация класса?

О. Благодаря *синхронизации* виртуальная машина Java гарантирует, что переменные и методы объекта будут представлены в согласованном виде для других пользователей этого объекта.

Данная концепция станет более понятной на занятии 15, после знакомства с потоками. Программы на Java могут выполнять одновременно несколько задач, каждая из которых работает в своем собственном потоке.

Если несколько потоков получают доступ к одному и тому же объекту, важно, чтобы этот объект вел себя одинаково в каждом потоке. Поэтому, если метод класса требует синхронизации, как в случае векторов и хеш-таблиц, виртуальной машине Java приходится выполнять дополнительную работу. К тому же это может привести к ошибкам, которые вызовут прекращение выполнения потока.

Списки массивов и хеш-карты дублируют возможности векторов и хеш-таблиц и могут использоваться там, где не требуется синхронизация. Следовательно, они являются более эффективными.

Коллоквиум

Проверьте знание материала, рассмотренного на этом занятии, ответив на следующие вопросы.

Контрольные вопросы

1. Какая из следующих структур не может изменять свои размеры после создания?
   - **А.** Списки массивов.
   - **Б.** Массивы.
   - **В.** Хеш-карты.
2. Какой метод применяется для определения количества элементов в списке массивов или хеш-карте?
   - **А.** `length()`.
   - **Б.** `size()`.
   - **В.** `count()`.
3. Какой тип данных или класс может использоваться в качестве ключей хеш-карты?
   - **А.** `int` или `Integer`.
   - **Б.** `String`.
   - **В.** Любой класс.

Ответы

1. **Б.** Размер массива определяется при его создании и остается неизменным. Размеры списков массивов и хеш-карт изменяются по мере необходимости.
2. **Б.** Метод `size()` возвращает текущее количество элементов в структуре данных.

3. **А, Б или В.** Хеш-карты могут использовать любой класс в качестве ключей или значений. Если задан тип данных `int`, он автоматически преобразуется в тип `Integer`.

Упражнения

Воспользуйтесь своими знаниями структур данных для выполнения следующих упражнений.

- Напишите программу, которая использует одну из структур данных, рассмотренных на этом занятии, для хранения списка адресов электронной почты, причем каждый адрес связан с именем определенного человека.
- Расширьте программу `FontMapper` таким образом, чтобы имя, размер и начертание нового шрифта задавались в виде аргументов командной строки. Эти данные должны добавляться в хеш-карту перед отображением ключей и значений.

ЗАНЯТИЕ 14
Обработка ошибок в программе

На этом занятии вы узнаете...

- почему ошибки называются исключениями;
- как реагировать на исключения в программах на Java;
- как создавать методы, которые игнорируют исключения;
- как использовать методы, которые генерируют исключения;
- как создавать собственные исключения.

Ошибки всех видов и сортов — естественная составляющая процесса разработки программ. Одни ошибки обнаруживаются компилятором и не позволяют вам создать класс, другие обнаруживаются интерпретатором в ответ на возникновение проблемы, которая не позволяет ему успешно выполняться. В Java ошибки делятся на следующие две категории.

- **Исключения.** События, которые сигнализируют о необычных обстоятельствах, возникающих при выполнении программы.
- **Ошибки выполнения.** События, которые сигнализируют о том, что интерпретатор столкнулся с проблемой, возможно, не связанной с вашей программой.

Как правило, ошибки выполнения не могут быть устранены самой программой, поэтому они не рассматриваются на этом занятии. Например, ошибка `OutOfMemoryError` (Нехватка памяти) означает, что программа настолько велика, что ей не хватает памяти компьютера. Программа не сможет обработать ошибку подобного рода и просто завершит свое выполнение, выдав сообщение об ошибке.

А вот исключения зачастую могут быть обработаны таким образом, чтобы программа продолжила корректно выполняться.

Исключения

Вы уже имели дело с исключениями на предыдущих 13 занятиях. Эти ошибки возникают в том случае, когда программа успешно компилируется, но сталкивается с проблемами при выполнении.

Например, распространенная ошибка — сослаться на несуществующий элемент массива, как в следующем примере.

```
String[] greek = { "альфа", "бета", "гамма" };
System.out.println(greek[3]);
```

Массив `greek` типа `String` содержит три элемента. Но поскольку первый элемент массива имеет номер 0, а не 1, ссылка на него выглядит так: `greek[0]`. Соответственно, ссылка на второй элемент — `greek[1]`, на третий — `greek[2]`. В результате инструкция, пытающаяся отобразить элемент `greek[3]`, приводит к ошибке. Сами инструкции компилируются успешно, но при попытке выполнения программы виртуальная машина Java выдаст следующее сообщение об ошибке.

```
Exception in thread "main" java.lang.ArrayIndexOutBoundsException:
   3 at SampleProgram.main(SampleProgram.java:4)
```

Оно указывает на то, что программа сгенерировала исключение, которое привело к ее останову.

Сообщение об ошибке ссылается на класс `ArrayIndexOutOfBoundsExcep tion` в пакете `java.lang`. Это класс исключения, т.е. объекта, описывающего исключительные обстоятельства, которые возникли при выполнении программы.

Обработка исключений — это способ взаимодействия классов в случае возникновения ошибок или других необычных обстоятельств. Как только класс Java сталкивается с исключением, он предупреждает пользователя об ошибке. В данном случае пользователем класса является виртуальная машина Java.

ПРИМЕЧАНИЕ

При описании процесса обработки исключений используют два термина: *генерирование* (throw) и *перехват* (catch). Объекты генерируют исключения, чтобы предупредить другие объекты о факте их возникновения. Возникшие исключения перехватываются другими объектами или виртуальной машиной Java.

Все исключения являются подклассами класса `Exception` из пакета `java. lang`. Уже упоминавшееся исключение `ArrayIndexOutOfBoundsException` (выход за границы массива) свидетельствует о том, что произошла попытка доступа к элементу массива, находящемуся за пределами массива.

В Java существуют сотни исключений. Многие из них, такие как выход за границы массива, указывают на наличие проблемы, которая может быть устранена программным способом. Они подобны ошибкам компилятора. После устранения ситуации, вызвавшей проблему, об исключении можно будет забыть.

Есть и такие исключения, с которыми вы столкнетесь при каждом выполнении программы. При этом используются пять новых ключевых слов: `try`, `catch`, `finally`, `throw` и `throws`.

Перехват исключений в блоке `try-catch`

До сего момента обработка исключений сводилась к устранению проблем, вызвавших эти исключения. Но бывает и так, что справиться с исключением подобным образом не получается, и тогда приходится выполнять обработку исключения в классе Java.

Чтобы понять природу исключений, введите код короткого приложения из листинга 14.1 в новый класс Java, включите его в пакет `com.java24hours` и сохраните этот файл под именем `Calculator.java`.

ЛИСТИНГ 14.1. Исходный код программы `Calculator.java`

```
 1: package com.java24hours;
 2:
 3: public class Calculator {
 4:    public static void main(String[] arguments) {
 5:       float sum = 0;
 6:       for (String argument : arguments) {
 7:          sum = sum + Float.parseFloat(argument);
 8:       }
 9:       System.out.println("Сумма этих чисел равна " + sum);
10:    }
11: }
```

Программа `Calculator` получает одно или несколько чисел в виде аргументов командной строки, суммирует их и сообщает, чему равна сумма.

Поскольку все аргументы командной строки в Java представлены строками, перед выполнением операции сложения программа должна преобразовать их в числа с плавающей точкой. Эту операцию выполняет метод `Float.parseFloat()`, вызываемый в строке 7. После завершения преобразования полученный результат прибавляется к значению переменной `sum`.

Перед запуском программы задайте следующие аргументы командной строки в NetBeans с помощью команды **Выполнить⇨Установить конфигурацию проекта ⇨Настроить: 7 4 8 1 4 1 4**. Чтобы выполнить программу, выберите команду **Выполнить⇨Запустить главный проект**. Результат показан на рис. 14.1.

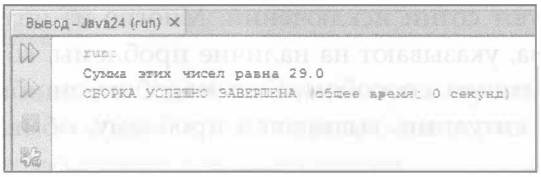


РИС. 14.1. Результат выполнения программы `Calculator`

Запустите программу несколько раз с разными числовыми аргументами. Никаких ошибок возникнуть не должно, и вы вправе спросить: а причем здесь исключения?

Чтобы увидеть их в действии, измените аргументы командной строки на `1 3 5x`. Третий аргумент содержит опечатку (символ `x` после `5`). Но программа `Calculator` не понимает, что это ошибка, поэтому пытается добавить `5x` к общей сумме, что приводит к появлению исключения (рис. 14.2).

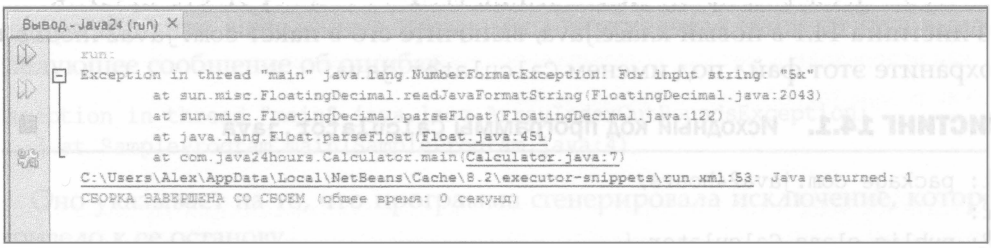


РИС. 14.2. Исключение, которое возникает из-за ошибки в аргументе

Это сообщение может быть весьма информативным для программиста, но вряд ли интересно для рядового пользователя. Поэтому лучше не отображать подобные сообщения, а обрабатывать исключения в программе.

Программы на Java обрабатывают собственные исключения с помощью блока инструкций `try-catch`, который имеет следующий синтаксис.

```
try {
   // Инструкции, которые могут привести к исключению
} catch (Exception e) {
   // Операции, выполняемые при возникновении исключения
}
```

Блок `try-catch` нужен для каждого исключения, которое вы хотите обрабатывать. Объект `Exception`, указываемый в инструкции `catch`, должен быть одной из следующих сущностей:

- классом исключения, которое может возникнуть;
- несколькими классами исключения, разделенными символом канала `'|'`;

- суперклассом для нескольких разных исключений, которые могут возникнуть.

Раздел `try` блока `try-catch` содержит инструкцию (или инструкции), которая может вызвать исключение. В программе `Calculator` вызов метода `Float.parseFloat(argument)` в строке 7 (листинг 14.1) приводит к генерированию исключения `NumberFormatException` всякий раз, когда используется строковый аргумент, который не может быть преобразован в значение с плавающей точкой.

Чтобы улучшить программу `Calculator`, предотвратив ее останов в случае возникновения подобных ошибок, можно воспользоваться блоком `try-catch`.

Создайте новый класс `NewCalculator`, включите его в пакет `com.java24hours`, введите код из листинга 14.2 и сохраните полученный файл `NewCalculator.java`.

ЛИСТИНГ 14.2. Исходный код программы `NewCalculator.java`

```
 1: package com.java24hours;
 2:
 3: public class NewCalculator {
 4:    public static void main(String[] arguments) {
 5:       float sum = 0;
 6:       for (String argument : arguments) {
 7:          try {
 8:             sum = sum + Float.parseFloat(argument);
 9:          } catch (NumberFormatException e) {
10:             System.out.println(argument + " - это не число.");
11:          }
12:       }
13:       System.out.println("Сумма этих чисел равна " + sum);
14:    }
15: }
```

После сохранения программы настройте конфигурацию проекта, выбрав в качестве главного класса `com.java24hours.NewCalculator`. Затем введите аргументы командной строки **1 3 5x** и запустите программу. Результат показан на рис. 14.3.

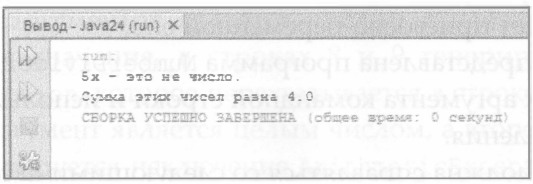


РИС. 14.3. Результат выполнения программы `NewCalculator`

Блок `try-catch` в строках 7–11 обрабатывает исключения `NumberFormatException`, генерируемые методом `Float.parseFloat()`. Эти исключения перехватываются в классе `NewCalculator`. Для любого аргумента, не являющегося числом, выводится сообщение об ошибке. Поскольку исключение обрабатывается внутри класса, виртуальная машина Java не сообщает об ошибках. Зачастую проблемы, связанные с вводом пользователем некорректных данных или наличием непредвиденных значений, решаются с помощью блоков `try-catch`.

Перехват нескольких разных исключений

Блок `try-catch` может применяться для обработки нескольких разных типов исключений, даже когда они генерируются разными инструкциями.

Один из способов обработки нескольких классов исключений заключается в том, чтобы назначить каждый блок `catch` отдельному классу, как показано в следующем фрагменте кода.

```
String textValue = "35";
int value;
try {
   value = Integer.parseInt(textValue);
} catch (NumberFormatException exc) {
   // Код обработки исключения
} catch (ArithmeticException exc) {
   // Код обработки исключения
}
```

Можно также обрабатывать несколько исключений в одном и том же блоке `catch`. При этом исключения разделяются символом канала `'|'`, а сам список завершается именем объекта исключения. Ниже представлен соответствующий пример.

```
try {
   value = Integer.parseInt(textValue);
} catch (NumberFormatException | ArithmeticException exc) {
   // Код обработки исключений
}
```

Если перехватывается исключение `NumberFormatException` или `ArithmeticException`, оно будет присвоено переменной `exc`.

В листинге 14.3 представлена программа `NumberDivider`, которая получает два целочисленных аргумента командной строки и использует их в операции целочисленного деления.

Эта программа должна справляться со следующими двумя потенциальными проблемами, возникающими при вводе данных пользователем:

- аргументы не являются целочисленными;
- деление на нуль.

Создайте новый класс `NumberDivider`, включите его в пакет `com.java24hours`, в редакторе исходного кода введите код из листинга 14.3 и сохраните полученный файл `NumberDivider.java`.

ЛИСТИНГ 14.3. Исходный код программы `NumberDivider.java`

```
 1: package com.java24hours;
 2:
 3: public class NumberDivider {
 4:    public static void main(String[] arguments) {
 5:       if (arguments.length == 2) {
 6:          int result = 0;
 7:          try {
 8:             result = Integer.parseInt(arguments[0]) /
 9:                      Integer.parseInt(arguments[1]);
10:             System.out.println(arguments[0] + " нацело разделить" +
11:                   " на " + arguments[1] + " будет " + result);
12:          } catch (NumberFormatException e) {
13:             System.out.println("Оба аргумента должны быть
14:                                 целочисленными.");
15:          } catch (ArithmeticException e) {
16:             System.out.println("Деление на нуль запрещено.");
17:          }
18:       }
19:    }
20: }
```

Аргументы командной строки могут иметь целочисленное и нечисловое значение, а также быть числами с плавающей точкой.

С помощью инструкции `if` в строке 5 проверяется факт передачи приложению двух аргументов. Если число аргументов иное, программа просто завершает свое выполнение.

Поскольку программа `NumberDivider` выполняет целочисленное деление, результатом будет целое число. К примеру, результат деления 5 на 2 будет 2, а не 2,5.

Если один из аргументов является числом с плавающей точкой или имеет нечисловое значение, в строках 8 и 9 генерируется исключение `NumberFormatException`, которое перехватывается в строке 12.

Если первый аргумент является целым числом, а второй равен нулю, то в строках 8 и 9 генерируется исключение `ArithmeticException`, которое перехватывается в строке 15.

Результат успешного выполнения программы показан на рис. 14.4.

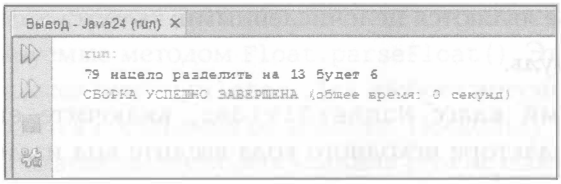


РИС. 14.4. Результат выполнения программы `NumberDivider`

Обработка инструкций, следующих после исключения

При обработке нескольких исключений с помощью блоков `try` и `catch` иногда нужно выполнить какие-то действия в конце блока, независимо от того, произошло исключение или нет.

Для этого нужно использовать блок `try-catch-finally`, синтаксис которого показан ниже.

```
try {
   // Инструкции, которые могут привести к исключению
} catch (Exception e) {
   // Действия, выполняемые при возникновении исключения
} finally {
   // Инструкции, выполняемые в любом случае
}
```

Инструкция (или инструкции), находящаяся в разделе `finally` блока, выполняется после всех других инструкций блока, независимо от того, возникло ли исключение.

Раздел `finally` может использоваться в программе, которая считывает данные из файла (она будет рассмотрена на занятии 20). При получении доступа к данным исключения могут возникать в силу следующих причин: файл не существует, произошла ошибка чтения с диска и т.п. Если инструкции, выполняющие чтение данных с диска, находятся в разделе `try`, а ошибки обрабатываются в разделе `catch`, то можно закрыть файл в разделе `finally`. Благодаря этому гарантируется закрытие файла, независимо от того, было ли сгенерировано исключение в процессе чтения данного файла.

Генерирование исключений

При вызове метода из другого класса порядок использования данного метода контролируется самим классом путем генерирования исключений.

При работе с классами из библиотеки классов Java компилятор часто отображает сообщение о некорректной обработке исключения. Вот соответствующий пример

```
NetReader.java:14: unreported exception
java.net.MalformedURLException;
must be caught or declared to be thrown
```

Всякий раз, когда появляется сообщение об ошибке, которое говорит о том, что исключение "must be caught or declared to be thrown" (должно быть перехвачено или объявлено как генерируемое), оно указывает на то, что метод, который вы пытаетесь использовать, генерирует данное исключение.

Любой класс, который вызывает такие методы, например созданное вами приложение, должен выполнять одну из следующих операций:

- обрабатывать исключение с помощью блока `try-catch`;
- генерировать исключение;
- обрабатывать ошибку с помощью блока `try-catch`, после чего генерировать исключение.

До сих пор мы говорили только о том, как обработать исключение. Если нужно сгенерировать исключение после его обработки, следует использовать инструкцию `throw`, указав в ней объект исключения.

В следующем фрагменте выполняется обработка исключения `NumberFormatException` в блоке `catch`, после чего исключение генерируется повторно.

```
float principal;
try {
   principal = Float.parseFloat(loanText) * 1.1F;
} catch (NumberFormatException e) {
   System.out.println(arguments[0] + " не является числом.");
   throw e;
}
```

В следующем фрагменте перехватывается любое исключение, которое может быть сгенерировано в блоке `try`, после чего оно генерируется повторно.

```
float principal;
try {
   principal = Float.parseFloat(loanText) * 1.1F;
} catch (Exception e) {
   System.out.println("Ошибка " + e.getMessage());
   throw e;
}
```

Класс `Exception` является родительским для всех классов исключений, поэтому инструкция `catch` перехватит любое исключение из иерархии `Exception`.

Генерирование исключения с помощью инструкции `throw` в общем случае означает, что вы не сделали все, что нужно сделать для обработки данного исключения.

Рассмотрим гипотетическую программу CreditCardChecker, которая верифицирует покупки, сделанные с помощью кредитной карты. Эта программа использует класс Database, который выполняет следующие действия.

1. устанавливает соединение с компьютером организации, выпустившей кредитную карту;
2. запрашивает у компьютера, корректный ли номер кредитной карты владельца;
3. запрашивает у компьютера, достаточно ли у владельца средств, чтобы совершить покупку.

Что произойдет в том случае, когда класс Database не сможет подключиться к компьютеру организации, которая выпустила кредитную карту? Подобного рода ошибки подключения обрабатываются блоком try-catch в самом классе Database.

Если класс Database обрабатывает эту ошибку сам по себе, то программа CreditCardChecker вообще не будет знать о том, что исключение было сгенерировано. И это не очень хорошо, ведь программа должна быть в курсе того, что подключение не может быть установлено, и сообщить об этом пользователю.

Один из способов уведомить программу CreditCardChecker о возникшем исключении — перехватить исключение в блоке catch класса Database, а затем сгенерировать его повторно с помощью инструкции throw. Исключение, которое генерируется в классе Database, должно обрабатываться точно так же, как и любое другое исключение.

Если инструкция throw находится в блоке catch, который перехватывает родительский класс исключения, такой как Exception, то генерируется исключение именно этого класса. В подобном случае часть информации о возникшей ошибке теряется, поскольку подкласс типа NumberFormatException может сообщить гораздо больше сведений о проблеме, чем класс Exception.

В Java имеется возможность отслеживать необходимые сведения. Для этого нужно указать в блоке catch ключевое слово final.

```
try {
   principal = Float.parseFloat(loanText) * 1.1F;
} catch (final Exception e) {
   System.out.println("Ошибка " + e.getMessage());
   throw e;
}
```

При наличии ключевого слова final в инструкции catch инструкция throw работает по-другому. Теперь она генерирует исключение конкретного класса, который был перехвачен.

Игнорирование исключений

А теперь рассмотрим, как проигнорировать исключение. Метод класса может игнорировать исключения с помощью ключевого слова `throws`, которое указывается в объявлении метода.

Следующий метод генерирует исключение `MalformedURLException`, которое может возникать при работе с веб-адресами в программе на Java.

```
public void loadURL(String address) throws MalformedURLException {
   URL page = new URL(address);
   // Код, загружающий веб-страницу
}
```

Вторая инструкция в этом примере создает объект `URL`, который представляет адрес в Интернете. Конструктор класса `URL` генерирует исключение `MalformedURLException` в случае, если указан некорректный веб-адрес. Например, следующая инструкция не сможет подключиться к заданному URL-адресу:

```
URL source = new URL("http:www.java24hours.com");
```

Строка `http:www.java24hours.com` не является корректным URL-адресом, так как в ней пропущены символы пунктуации: два символа косой черты (//) после двоеточия.

Поскольку метод `loadURL()` объявлен как генерирующий исключение `MalformedURLException`, обработка исключения в самом методе не требуется. Ответственность за перехват этого исключения возлагается на любой метод, который вызывает метод `loadURL()`.

Исключения, которые не нужно обрабатывать в блоке `catch`

Далеко не все исключения должны перехватываться с помощью блока `try-catch` либо объявляться с помощью ключевого слова `throws`. Есть исключения, которые не должны никак обрабатываться. Компилятор не будет никак реагировать при обнаружении игнорируемого исключения. Такие исключения называются *непроверяемыми*, в отличие от остальных исключений, которые называются *проверяемыми*.

Непроверяемые исключения являются подклассами класса `RuntimeException` из пакета `java.lang`. В качестве типичного примера можно привести исключение `IndexOutOfBoundsException`, которое указывает на то, что индекс, применяемый для доступа к массиву, строке или списку массива, выходит за пределы этой структуры данных. Если, например, массив содержит пять элементов и предпринимается попытка получить доступ к десятому элементу, то генерируется это исключение.

Еще одним примером может служить исключение NullPointerException, которое генерируется в случае использования объекта, не имеющего значения. Например, объектным переменным изначально присваивается значение null. Некоторые методы также возвращают значение null, когда не может быть возвращен объект. Если инструкция некорректно предполагает, что используемый объект имеет значение, генерируется исключение NullPointerException.

Это два примера того, что программист может (и должен) предотвращать в коде, а не того, что требует обработки исключений. Если вы пишете программу, которая получает доступ к элементу массива, находящемуся за его пределами, исправьте некорректный код и повторно скомпилируйте его. Если вы ожидаете появление объекта, а вместо него получаете null, проверьте это с помощью условия if перед использованием объекта.

Следует избегать непроверяемых исключений в Java путем тщательного контроля создаваемого кода. Необходимость частой обработки подобных исключений приводит к неоправданному усложнению программы. Например, исключение NullPointerException может возникать в каждой инструкции, в которой вызываются методы объекта.

Даже если исключение можно проигнорировать, это вовсе не означает, что так следует поступать. У вас по-прежнему остается возможность использовать ключевые слова try, catch и throws вместе с непроверяемыми исключениями.

Генерирование и перехват исключений

В последнем проекте этого занятия будет создан класс, который использует исключения для передачи другому классу информации об ошибке. В этом проекте используется класс HomePage, который представляет личную домашнюю страницу в Интернете, и класс PageCatalog, соответствующий приложению, которое каталогизирует эти страницы.

Введите код из листинга 14.4 в новый класс HomePage, включите его в пакет com.java24hours и сохраните полученный файл HomePage.java.

ЛИСТИНГ 14.4. Исходный код приложения HomePage.java

```
1: package com.java24hours;
2:
3: import java.net.*;
4:
5: public class HomePage {
6:    String owner;
7:    URL address;
8:    String category = "нет";
```

```
 9:
10:    public HomePage(String inOwner, String inAddress)
11:       throws MalformedURLException {
12:
13:       owner = inOwner;
14:       address = new URL(inAddress);
15:    }
16:
17:    public HomePage(String inOwner, String inAddress,
18:                    String inCategory)
19:       throws MalformedURLException {
20:
21:       this(inOwner, inAddress);
22:       category = inCategory;
23:    }
24: }
```

Класс `HomePage` можно использовать в других программах. Он представляет персональные веб-страницы и включает три переменных экземпляра. Переменная `address` является объектом `URL`, содержащим адрес веб-страницы, переменная `owner` соответствует владельцу веб-страницы, а переменная `category` содержит краткий комментарий, описывающий основную тематику страницы.

Подобно любому классу, который создает объекты `URL`, класс `HomePage` должен обрабатывать исключение `MalformedURLException` в блоке `try-catch` или объявлять о его генерировании.

В строках 11 и 19 используется ключевое слово `throws` в двух методах конструктора. Это позволяет классу `HomePage` не обрабатывать исключение `MalformedURLException` самостоятельно.

Чтобы создать приложение, которое использует класс `HomePage`, вернитесь в среду NetBeans и создайте класс `PageCatalog`. Включите этот класс в пакет `com.java24hours`, введите код из листинга 14.5 и сохраните полученный файл `PageCatalog.java`.

ЛИСТИНГ 14.5. Исходный код программы `PageCatalog.java`

```
 1: package com.java24hours;
 2:
 3: import java.net.*;
 4:
 5: public class PageCatalog {
 6:    public static void main(String[] arguments) {
 7:       HomePage[] catalog = new HomePage[5];
 8:       try {
 9:          catalog[0] = new HomePage("Майк Глайер",
10:             "http://www.file770.com", "научная фантастика");
```

```
11:           catalog[1] = new HomePage("Шелли Пауэрс",
12:              "http://burningbird.net", "экология");
13:           catalog[2] = new HomePage("Роджерс Кейденхед",
14:              "http://workbench.cadenhead.org", "программирование");
15:           catalog[3] = new HomePage("Таеган Годдард",
16:              "https://politicalwire.com", "политика");
17:           catalog[4] = new HomePage("Мэнтон Риц",
18:              "www.manton.org");
19:           for (int i = 0; i < catalog.length; i++) {
20:              System.out.println(catalog[i].owner + ": " +
21:                 catalog[i].address + " -- " +
22:                 catalog[i].category);
23:           }
24:        } catch (MalformedURLException e) {
25:           System.out.println("Ошибка: " + e.getMessage());
26:        }
27:     }
28: }
```

Результат выполнения скомпилированного приложения показан на рис. 14.5.

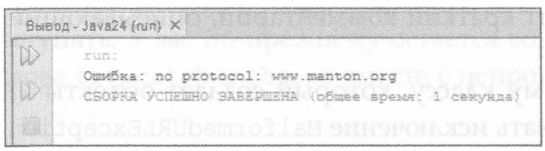


РИС. 14.5. Результат выполнения приложения `PageCatalog` некорректен

Приложение `PageCatalog` создает массив объектов `HomePage` и отображает содержимое этого массива. При создании каждого объекта `HomePage` используется до трех аргументов:

- имя владельца страницы;
- адрес страницы (как объект `String`, а не `URL`);
- категория страницы.

Третий аргумент является необязательным и не используется в строках 17 и 18.

Конструкторы класса `HomePage` генерируют исключение `MalformedURLException`, когда получают строку, которая не может быть преобразована в корректный объект `URL`. Это исключение обрабатывается с помощью блока `try-catch` в приложении `PageCatalog`.

Чтобы устранить проблему, вызывающую ошибку "no protocol" (не указан протокол), измените строку 18 таким образом, чтобы адрес начинался

с `http://`, как другие веб-адреса в строках 10–16. Результат повторного запуска приложения показан на рис. 14.6.

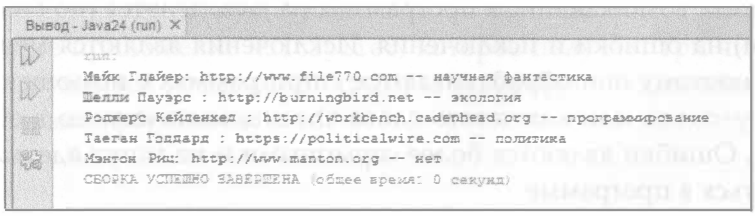


РИС. 14.6. Корректный результат выполнения приложения `PageCatalog`

Резюме

Итак, на этом занятии вы освоили способы обработки исключений в Java. Можно применять следующие методики:

- перехват исключения с последующей обработкой;
- игнорирование исключения и передача его другому классу или виртуальной машине Java;
- перехват нескольких разных исключений в одном и том же блоке `try-catch`;
- генерирование собственных исключений.

Благодаря управлению исключениями программы на Java становятся более надежными, гибкими и простыми в применении, а пользователи не будут получать малопонятные сообщения об ошибках.

Вопросы и ответы

В. Можно ли создавать свои собственные исключения?

О. Это не просто возможно, но и весьма полезно. Благодаря созданию пользовательских исключений классы становятся более надежными. Пользовательские исключения можно создавать в виде подкласса существующего исключения, такого как `Exception`, который является суперклассом всех исключений. В классе `Exception` имеются два конструктора, которые можно переопределить: `Exception()` без аргументов и `Exception()` с аргументом типа `String`. Во втором случае строка должна быть сообщением, которое описывает возникшую ошибку.

В. Почему на этом занятии в дополнение к исключениям не рассматривается генерирование и перехват ошибок?

О. Проблемы, возникающие в программах на Java, делятся (по степени серьезности) на ошибки и исключения. Исключения являются менее серьезными, поэтому они обрабатываются в программах с помощью инструкций `try-catch` или ключевого слова `throws`, включенного в объявление метода. Ошибки являются более серьезными и не могут адекватно обрабатываться в программе.

Примеры ошибок — "stack overflow" (переполнение стека) и "out of memory" (нехватка памяти). Ошибки могут привести к сбою виртуальной машины Java, к тому же их невозможно устранить в программе, выполняемой под управлением JVM.

Коллоквиум

Проверьте знание материала, рассмотренного на этом занятии, ответив на следующие вопросы.

Контрольные вопросы

1. Сколько исключений может обрабатывать одна инструкция `catch`?
   - **А.** Одно.
   - **Б.** Несколько разных исключений.
   - **В.** Ответ на этот вопрос неизвестен.
2. Когда выполняются инструкции в разделе `finally`?
   - **А.** После выполнения блока `try-catch` с генерированием исключения.
   - **Б.** После выполнения блока `try-catch` без генерирования исключения.
   - **В.** Варианты **А** и **Б**.

Ответы

1. **Б.** Объект `Exception` в инструкции `catch` может обрабатывать все исключения своего собственного класса и его подклассов. Также можно указать несколько классов, разделенных символом канала: `'|'`.
2. **В.** Инструкция (или инструкции) в блоке `finally` всегда выполняется после других инструкций блока `try-catch`, независимо от того, возникло исключение или нет.

Упражнения

Чтобы закрепить пройденный материал, постарайтесь сделать как можно меньше ошибок при выполнении следующих упражнений.

- Измените приложение `NumberDivider` таким образом, чтобы оно генерировало все перехватываемые исключения, и запустите его, чтобы посмотреть на полученный результат.
- Создайте приложение `Multiplier`, которое получает три числа в качестве аргументов командной строки и генерирует исключение в случае нечисловых аргументов.

ЗАНЯТИЕ 15
Создание многопоточной программы

На этом занятии вы узнаете...

- как использовать программный интерфейс;
- как создавать потоки;
- как запускать, останавливать и приостанавливать потоки;
- как перехватывать ошибки.

Стремительный темп современной жизни требует многозадачности, когда одновременно приходится делать несколько вещей, например просматривать веб-страницы, участвовать в видеоконференции и упражняться с кистевым эспандером. На многозадачном компьютере тоже можно одновременно выполнять несколько программ.

В Java можно создавать программы, которые выполняют сразу несколько задач. Для этого в языке предусмотрен класс объектов, называемых *потоками*.

Потоки

В программе на Java задачи, которые одновременно выполняются на компьютере, называются *потоками*, а процесс управления потоками называется *многопоточностью*. Многопоточность применяется при создании анимации и во многих других сферах.

Благодаря потокам программа может одновременно выполнять несколько задач, каждая из которых помещается в свой собственный поток и зачастую реализуется в виде отдельного класса.

Потоки представлены с помощью класса `Thread` и интерфейса `Runnable`, которые включены в пакет `java.lang`. Вам не потребуется задавать инструкцию `import`, чтобы ссылаться на них в своих программах.

Одно из простейших применений класса `Thread` — замедление выполнения программы.

Замедление выполнения программы

Класс `Thread` включает метод `sleep()`, который можно вызвать, если необходимо приостановить выполнение программы на короткий промежуток времени. Эта методика часто применяется в программах анимации, где она позволяет не выводить изображения быстрее, чем виртуальная машина Java может обработать их.

Чтобы использовать метод `sleep()`, укажите длительность паузы в миллисекундах, как показано в следующем примере:

```
Thread.sleep(5000);
```

В результате виртуальная машина Java приостанавливается на пять секунд (5000 миллисекунд). Если в силу каких-либо причин JVM не может сделать столь длительную паузу, метод `sleep()` генерирует исключение `InterruptedException`.

Вероятность появления данного исключения означает, что его следует обрабатывать при использовании метода `sleep()`. Один из вариантов — поместить инструкцию `Thread.sleep()` в блок `try-catch`.

```
try {
   Thread.sleep(5000);
} catch (InterruptedException e) {
   // Досрочное пробуждение
}
```

Если нужно, чтобы программа на Java выполняла несколько задач одновременно, разбейте ее в потоки. Программа может включать произвольное количество потоков, которые могут работать одновременно, не влияя друг на друга.

Создание потока

Класс Java, который может выполняться в виде потока, называется *выполняемым* (или *потоковым*). В предыдущем разделе потоки применялись для приостановки выполнения программы на несколько секунд, однако программисты часто используют потоки с противоположной целью — для ускорения выполнения программы. Если поместить задачи, требующие больших вычислительных ресурсов, в свои собственные потоки, остальная часть программы будет выполняться быстрее. Этот подход часто применяется для улучшения отзывчивости графического интерфейса программы при выполнении трудоемкого задания.

Например, в случае приложения, которое загружает из файла сведения о котировках акций на фондовом рынке и формирует статистику по ним, больше всего времени занимает загрузка данных из файла. И если в приложении

не используются потоки, то в процессе загрузки данных реагирование интерфейса сильно замедлится. Вряд ли это обрадует пользователя.

Для того чтобы поместить задачу в свой собственный поток, можно воспользоваться одним из двух подходов:

- включить задачу в класс, который реализует интерфейс `Runnable`;
- включить задачу в подкласс класса `Thread`.

Для реализации интерфейса `Runnable` необходимо при создании класса указать ключевое слово `implements`.

```
public class LoadStocks implements Runnable {
   // Тело класса
}
```

Если класс реализует интерфейс, то это означает, что помимо собственных методов класс содержит дополнительное поведение.

Классы, которые реализуют интерфейс `Runnable`, должны включать метод `run()`.

```
public void run() {
   // Тело метода
}
```

Метод `run()` предназначен для управления задачей, которая выполняется в созданном потоке. В примере с фондовым рынком метод `run()` может содержать инструкции, предназначенные для загрузки данных с диска и формирования статистики по ним.

В потоковом приложении инструкции в методе `run()` не выполняются автоматически. Потоки в Java можно запускать и останавливать, и поток не начнет выполняться, пока вы не предпримите два действия:

- создадите объект в потоковом классе путем вызова конструктора `Thread`;
- запустите поток на выполнение путем вызова его метода `start()`.

Конструктор `Thread` имеет единственный аргумент: объект, содержащий метод `run()` потока. В качестве аргумента зачастую можно использовать ключевое слово `this`, которое указывает на то, что текущий класс включает метод `run()`.

В листинге 15.1 показан код приложения Java, которое находит первый миллион простых чисел и сохраняет их в буфере `StringBuffer`. Найденные числа отображаются на панели вывода. В среде NetBeans создайте новый класс `PrimeFinder`, включите его в пакет `com.java24hours`, введите в него код листинга и сохраните полученный файл `PrimeFinder.java`.

ЛИСТИНГ 15.1. Программа `PrimeFinder.java`

```
 1: package com.java24hours;
 2:
 3: public class PrimeFinder implements Runnable {
 4:    Thread go;
 5:    StringBuffer primes = new StringBuffer();
 6:    int time = 0;
 7:
 8:    public PrimeFinder() {
 9:       start();
10:       while (primes != null) {
11:          System.out.println(time);
12:          try {
13:             Thread.sleep(1000);
14:          } catch (InterruptedException exc) {
15:             // Ничего не делать
16:          }
17:          time++;
18:       }
19:    }
20:
21:    public void start() {
22:       if (go == null) {
23:          go = new Thread(this);
24:          go.start();
25:       }
26:    }
27:
28:    public void run() {
29:       int quantity = 1_000_000;
30:       int numPrimes = 0;
31:       // Кандидат на простое число
32:       int candidate = 2;
33:       primes.append("\nПервые ").append(quantity).append(" " +
34:                     "простых чисел:"\n\n");
35:       while (numPrimes < quantity) {
36:          if (isPrime(candidate)) {
37:             primes.append(candidate).append(" ");
38:             numPrimes++;
39:          }
40:          candidate++;
41:       }
42:       System.out.println(primes);
43:       primes = null;
44:       System.out.println("\nПрошло: " + time + " секунд");
45:    }
46:
47:    public static boolean isPrime(int checkNumber) {
```

```
48:        double root = Math.sqrt(checkNumber);
49:        for (int i = 2; i <= root; i++) {
50:           if (checkNumber % i == 0) {
51:              return false;
52:           }
53:        }
54:        return true;
55:     }
56:
57:     public static void main(String[] arguments) {
58:        new PrimeFinder();
59:     }
60: }
```

Поиск миллиона простых чисел занимает много времени, поэтому лучше выделить отдельный поток в Java для выполнения этой задачи. Пока поток делает свою работу в фоновом режиме, программа `PrimeFinder` отображает затраченное время (в секундах). После завершения потока выводятся найденные простые числа и программа завершает свою работу. Часть вывода программы показана на рис. 15.1.

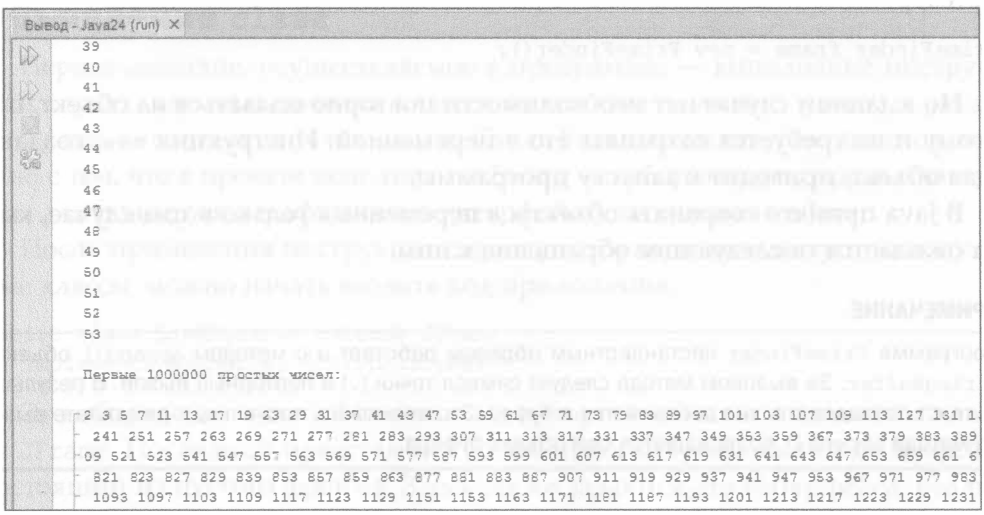


РИС. 15.1. Результат выполнения программы `PrimeFinder`

Большинство инструкций программы предназначено для поиска простых чисел. Для реализации потоков в программе используются следующие инструкции.

- Строка 3. Объявляется класс `PrimeFinder`, реализующий интерфейс `Runnable`.

- Строка 4. Создается переменная `go` класса `Thread`, которой пока что ничего не присвоено.
- Строки 21–26. Если объектная переменная `go` равна `null`, значит, поток еще не создан. В таком случае создается новый объект `Thread`, который сохраняется в данной переменной. Для запуска потока вызывается метод `start()`, который, в свою очередь, вызывает метод `run()` класса `PrimeFinder`.
- Строки 28–45. Метод `run()` ищет последовательность простых чисел, которая начинается с 2, и сохраняет каждое найденное число в строковом буфере `primes`. Для добавления чисел в буфер используется метод `append()`.

В методе `main()` создается объект `PrimeFinder`, предназначенный для запуска программы, но происходит это нестандартно. Рассмотрим следующую инструкцию:

```
new PrimeFinder();
```

Обычно созданный объект присваивается объектной переменной, например:

```
PrimeFinder frame = new PrimeFinder();
```

Но в данном случае нет необходимости повторно ссылаться на объект, поэтому и не требуется сохранять его в переменной. Инструкция `new`, создающая объект, приводит к запуску программы.

В Java принято сохранять объекты в переменных только в том случае, когда ожидается последующее обращение к ним.

ПРИМЕЧАНИЕ

Программа `PrimeFinder` нестандартным образом работает и с методом `append()` объекта `StringBuffer`. За вызовом метода следует символ точки (`.`) и повторный вызов. В результате текст последовательно добавляется в буфер. Это возможно, поскольку в результате вызова метода `append()` возвращается сам объект буфера.

Работа с потоками

Для запуска потока предназначен метод `start()`. Логично предположить, что существует и метод `stop()`, прекращающий выполнение потока.

Действительно, в классе `Thread` имеется метод `stop()`, но он является не рекомендуемым. В Java *не рекомендуемый* элемент — это класс, интерфейс, метод или переменная, которые заменены другим элементом, работающим лучше.

ПРЕДУПРЕЖДЕНИЕ

Обращайте внимание на предупреждения о не рекомендуемых элементах. Компания Oracle отнесла метод `stop()` к категории не рекомендуемых, поскольку он может вызвать проблемы у других потоков, выполняемых в JVM. К этой категории также относятся методы класса `resume()` и `suspend()`.

Следующий проект, который рассматривается на этом занятии, будет посвящен тому, как остановить выполнение потока. Эта программа циклически обходит список заголовков веб-сайтов и их адресов.

Заголовок каждой страницы и веб-адрес отображаются в непрерывном цикле. Для посещения сайта, отображаемого в данный момент, нужно щелкнуть на соответствующей кнопке графического интерфейса приложения. Данная программа работает в течение определенного периода времени, последовательно отображая сведения о каждом сайте. Подобная периодичность обеспечивается благодаря использованию потоков, которые обеспечивают наилучший способ управления такой программой.

Прежде чем вводить код приложения `LinkRotator` в редакторе исходного кода NetBeans, подробно рассмотрим каждый раздел программы.

Объявление `class`

Первое действие, осуществляемое в программе, — выполнение инструкции `import` для подключения классов из пакетов `java.awt`, `java.io`, `java.net`, `java.awt.event` и `javax.swing`. Столь большое количество пакетов связано с тем, что в проекте используется Swing — набор пакетов, предназначенных для поддержки графического интерфейса пользователя в Java.

После применения инструкции `import`, которая делает доступными нужные классы, можно начать вводить код приложения.

```
public class LinkRotator extends JFrame
   implements Runnable, ActionListener {
```

Это объявление класса `LinkRotator`, который является подклассом класса `JFrame`. Последний представляет собой простой графический интерфейс, состоящий из пустого фрейма. Здесь также задаются два интерфейса, реализуемые создаваемым классом: `Runnable` и `ActionListener`. Благодаря реализации класса `Runnable` можно запустить поток на выполнение с помощью метода `run()`, а благодаря интерфейсу `ActionListener` программа реагирует на щелчки мышью.

Настройка переменных

Первое, что нужно сделать в классе `LinkRotator`, — создать переменные и объекты. Мы создадим шестиэлементный массив `pageTitle` объектов `String` и шестиэлементный массив `pageLink` объектов `URI`.

```
String[] pageTitle = new String[6];
URI[] pageLink = new URI[6];
```

В массиве `pageTitle` хранятся заголовки шести отображаемых сайтов. Класс `URI` хранит значение адреса сайта и включает поведение и атрибуты, используемые для работы с веб-адресом.

Теперь осталось создать целочисленную переменную, объект `Thread` и метку интерфейса пользователя.

```
int current = 0;
Thread runner;
JLabel siteLabel = new JLabel();
```

Переменная `current` позволяет отслеживать, какой сайт отображается в текущий момент при циклическом обходе сайтов. Объект `runner` класса `Thread` представляет поток, запускаемый этой программой. При запуске, остановке и приостановке программы вызываются методы объекта `runner`.

Конструктор

Конструктор программы автоматически выполняется при запуске программы и применяется для инициализации массивов `pageTitle` и `pageLink`, а также для создания кнопки в интерфейсе пользователя. Тело конструктора показано ниже.

```
pageTitle = new String[] {
   "Сайт Oracle Java",
   "Сообщество ServerSide",
   "Сообщество JavaWorld",
   "Освой самостоятельно Java за 24 часа",
   "Издательство Sams Publishing",
   "Сайт Роджерса Кейденхеда"
};
pageLink[0] = getURI("http://www.oracle.com/technetwork/java");
pageLink[1] = getURI("http://www.theserverside.com");
pageLink[2] = getURI("http://www.javaworld.com");
pageLink[3] = getURI("http://www.java24hours.com");
pageLink[4] = getURI("http://www.samspublishing.com");
pageLink[5] = getURI("http://workbench.cadenhead.org");
Button visitButton = new Button("Посетить сайт");
goButton.addActionListener(this);
add(visitButton);
```

Заголовки страниц хранятся в шести элементах массива `pageTitle`, который инициализируется с помощью шести строк. Элементам массива `pageLink` присваивается значение, возвращаемое методом `getURI()`, который еще будет создан.

Последние три инструкции метода `init()` создают кнопку с надписью Посетить сайт и добавляют ее во фрейм приложения.

Перехват ошибок при настройке URL-адресов

При настройке объекта `URI` следует убедиться в корректности формата адреса. Например, адреса

```
http://workbench.cadenhead.org
```

и

```
http://www.samspublishing.com
```

корректны, тогда как адрес

```
http:www.javaworld.com
```

некорректен, поскольку отсутствуют две косые черты (//) после точки.

Метод `getURI(`*строка*`)` в качестве аргумента получает адрес сайта и возвращает объект `URI`, соответствующий этому адресу. Если строка не является корректным адресом, метод возвращает значение `null`.

```
URI getURI(String urlText) {
   URI pageURI = null;
   try {
      pageURI = new URI(urlText);
   } catch (URISyntaxException m) {
      // Ничего не делать
   }
   return pageURI;
}
```

Блок `try-catch` обрабатывает любые исключения `URISyntaxLException`, которые возникают при создании объектов `URI`. И поскольку в случае генерирования исключения ничего делать не нужно, блок `catch` включает лишь комментарий.

Запуск потока

В рассматриваемой программе поток `runner` запускается после вызова метода `start()` потока.

Метод `start()` вызывается в качестве последней инструкции конструктора. Вот тело метода.

```
public void start() {
   if (runner == null) {
      runner = new Thread(this);
      runner.start();
   }
}
```

Данный метод запускает поток `runner`, если он еще не запущен.

Инструкция `runner = new Thread(this)` создает новый объект `Thread` с единственным аргументом — ключевым словом `this`, которое ссылается на саму программу, обозначая ее как класс, выполняемый в потоке.

Вызов метода `runner.start()` приводит к запуску потока. Как только поток начинает выполняться, вызывается метод `run()` потока. В данном случае поток `runner` — это само приложение `LinkRotator`, поэтому вызывается его метод `run()`.

Выполнение потока

В методе `run()` находится сам поток. Он содержит следующие инструкции.

```
public void run() {
   Thread thisThread = Thread.currentThread();
   while (runner == thisThread) {
      current++;
      if (current > 5) {
         current = 0;
      }
      siteLabel.setText(pageTitle[current]);
      repaint();
      try {
         Thread.sleep(1000);
      } catch (InterruptedException e) {
         // Ничего не делать
      }
   }
}
```

Первое, что делает метод `run()`, — создает объект `Thread` с именем `thisThread`. Метод `currentThread()` класса `Thread` возвращает объект текущего потока.

Остальные инструкции в этом методе являются частью цикла `while`, который сравнивает объект `runner` с объектом `thisThread`. Оба объекта являются потоками, и пока они ссылаются на один и тот же объект, цикл `while` будет выполняться. В этом цикле отсутствует инструкция, с помощью которой объектам `runner` и `thisThread` присваиваются иные значения, поэтому цикл будет выполняться бесконечно до тех пор, пока один из указанных объектов не будет изменен за его пределами.

Метод `run()` вызывает метод `repaint()`. После этого значение переменной `current` увеличивается на единицу, и если оно превышает 5, то снова сбрасывается в 0. Переменная `current` предназначена для определения отображаемого сайта и применяется в качестве индекса строкового массива `pageTitle`, а заголовок сайта становится текстом компонента пользовательского интерфейса `siteLabel`.

Метод `run()` включает блок `try-catch` для обработки исключений. Инструкция `Thread.sleep(1000)` приостанавливает поток на 1 секунду, и этого времени достаточно для чтения имени сайта и его адреса. Инструкция `catch` обрабатывает исключение `InterruptedException`, которое может появиться при выполнении инструкции `Thread.sleep()`. Подобные ошибки возникают в тех случаях, когда по какой-то причине поток выходит из состояния сна.

Обработка щелчков мышью

Осталось рассмотреть последний раздел программы `LinkRotator`, в котором выполняется обработка событий, связанных с вводом данных пользователем. Всякий раз, когда пользователь щелкает на кнопке **Посетить сайт**, приложение должно открыть сайт в окне браузера. При этом используется метод `actionPerformed()`, требуемый интерфейсом `ActionListener`. Этот метод вызывается всякий раз, когда выполняется щелчок на кнопке.

Ниже приведен код метода `actionPerformed()` из класса `LinkRotator`.

```
public void actionPerformed(ActionEvent event) {
   Desktop desktop = Desktop.getDesktop();
   if (pageLink[current] != null) {
      try {
         desktop.browse(pageLink[current]);
         runner = null;
         System.exit(0);
      } catch (IOException exc) {
         // Ничего не делать
      }
   }
}
```

Сначала этот метод создает объект `Desktop`. Класс `Desktop` из пакета `java.awt` представляет среду рабочего стола компьютера, на котором выполняется приложение. С помощью этого объекта можно запустить программу электронной почты, используя ссылку "mailto:", открыть файл для редактирования в другой программе, вывести файл на печать или запустить любую внешнюю программу.

В данном случае объект `Desktop` применяется для открытия веб-страницы с помощью браузера, заданного по умолчанию.

Метод `browse(`*`URI-идентификатор`*`)` загружает указанную веб-страницу в браузер. Если элемент `pageLink[current]` является корректным адресом, то метод `browse()` запрашивает загрузку этой страницы браузером.

Отображение сменяемых ссылок

Теперь вы готовы создать и протестировать программу. Создайте новый класс `LinkRotator`, включите его в пакет `com.java24hours`, введите код из листинга 15.2 и сохраните полученный файл `LinkRotator.java`.

ЛИСТИНГ 15.2. Исходный код программы `LinkRotator.java`

```
 1: package com.java24hours;
 2:
 3: import java.awt.*;
 4: import java.awt.event.*;
 5: import java.io.*;
 6: import javax.swing.*;
 7: import java.net.*;
 8:
 9: public class LinkRotator extends JFrame
10:    implements Runnable, ActionListener {
11:
12:    String[] pageTitle = new String[6];
13:    URI[] pageLink = new URI[6];
14:    int current = 0;
15:    Thread runner;
16:    JLabel siteLabel = new JLabel();
17:
18:    public LinkRotator() {
19:       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
20:       setSize(300, 100);
21:       FlowLayout flo = new FlowLayout();
22:       setLayout(flo);
23:       add(siteLabel);
24:       pageTitle = new String[] {
25:          "Сайт Oracle Java",
26:          "Сообщество Server Side",
27:          "Сообщество JavaWorld",
28:          "Освой самостоятельно Java за 24 часа",
29:          "Издательство Sams Publishing",
30:          "Сайт Роджерса Кейденхеда"
31:       };
32:       pageLink[0] = getURI("http://www.oracle.com/technetwork/java");
33:       pageLink[1] = getURI("http://www.theserverside.com");
34:       pageLink[2] = getURI("http://www.javaworld.com");
35:       pageLink[3] = getURI("http://www.java24hours.com");
```

```
36:       pageLink[4] = getURI("http://www.samspublishing.com");
37:       pageLink[5] = getURI("http://workbench.cadenhead.org");
38:       Button visitButton = new Button("Посетить сайт");
39:       visitButton.addActionListener(this);
40:       add(visitButton);
41:       setVisible(true);
42:       start();
43:    }
44:
45:    private URI getURI(String urlText) {
46:       URI pageURI = null;
47:       try {
48:          pageURI = new URI(urlText);
49:       } catch (URISyntaxException ex) {
50:         // Ничего не делать
51:       }
52:       return pageURI;
53:    }
54:
55:    public void start() {
56:       if (runner == null) {
57:          runner = new Thread(this);
58:          runner.start();
59:       }
60:    }
61:
62:    public void run() {
63:       Thread thisThread = Thread.currentThread();
64:       while (runner == thisThread) {
65:          current++;
66:          if (current > 5) {
67:             current = 0;
68:          }
69:          siteLabel.setText(pageTitle[current]);
70:          repaint();
71:          try {
72:             Thread.sleep(2000);
73:          } catch (InterruptedException exc) {
74:             // Ничего не делать
75:          }
76:       }
77:    }
78:
79:    public void actionPerformed(ActionEvent event) {
80:       Desktop desktop = Desktop.getDesktop();
81:       if (pageLink[current] != null) {
82:          try {
83:             desktop.browse(pageLink[current]);
84:             runner = null;
```

```
85:             System.exit(0);
86:           } catch (IOException exc) {
87:             // Ничего не делать
88:           }
89:       }
90:    }
91:
92:    public static void main(String[] arguments) {
93:       new LinkRotator();
94:    }
95: }
```

На рис. 15.2 показаны два окна, открытых на рабочем столе компьютера. Меньшее окно по центру — это выполняющееся приложение `LinkRotator`. Большее окно открывается после щелчка на одной из ссылок в программе — в данном случае это сайт издательства Sams Publishing.

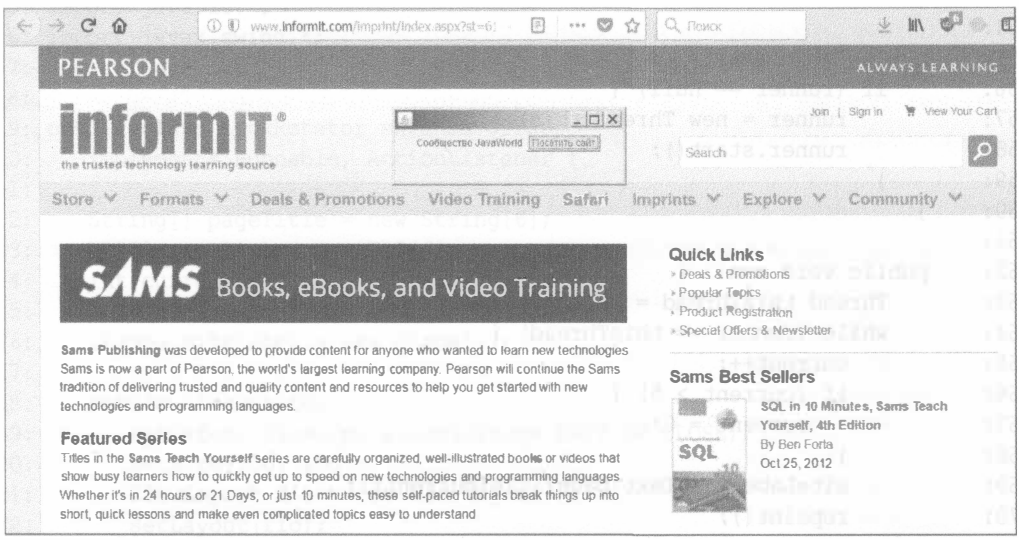


РИС. 15.2. Отображение ссылок в приложении

Останов потока

Приложение `LinkRotator` само по себе не может остановить выполнение потока, но оно написано так, что это несложно сделать. Ниже приведен метод, который останавливает выполнение потока.

```
public void stop() {
   if (runner != null) {
      runner = null;
   }
}
```

Инструкция `if` проверяет, равен ли объект `runner` значению `null`. Если это так, значит, отсутствует активный поток, который нужно остановить. В противном случае инструкция присваивает объекту `runner` значение `null`.

Присваивание объекту `runner` значения `null` приведет к тому, что значение этого объекта будет отличаться от значения объекта `thisThread`. В результате произойдет выход из цикла `while` в методе `run()`.

ПРИМЕЧАНИЕ

На этом занятии мы немного забежали наперед, использовав Swing и объекты графического интерфейса пользователя для управления потоками. Более детальное знакомство с программированием графического интерфейса пользователя в Java ожидает вас на занятии 17.

Резюме

Потоки реализованы в Java с привлечением небольшого числа классов и интерфейсов. Благодаря поддержке многопоточности можно существенно ускорять выполнение программ.

Вопросы и ответы

В. Есть ли какие-то причины ничего не делать в инструкции `catch`, как в приложении `LinkRotator`?

О. Все зависит от типа исключения, которое нужно перехватить. В приложении `LinkRotator` причина появления исключения заранее известна в обеих инструкциях `catch`, поэтому мы уверены в том, что дополнительная обработка не требуется. В методе `getURI()` исключение `URISyntaxException` генерируется только в том случае, когда URI-адрес, отсылаемый методу, некорректен.

Еще одно исключение, которое не требует обработки в блоке `catch`, — это `InterruptedException`. За более чем 20-летнюю карьеру программиста на Java я не встречал программы, которая генерировала бы это исключение.

Большинство исключений требует как минимум регистрации ошибки.

Коллоквиум

Проверьте знание материала, рассмотренного на этом занятии, ответив на следующие вопросы.

Контрольные вопросы

1. Какой интерфейс следует реализовать в программе, использующей потоки?
   А. `Runnable`.
   Б. `Thread`.
   В. Интерфейс не требуется.
2. Если интерфейс включает три разных метода, сколько из них нужно определить в классе, который реализует интерфейс?
   А. Ни одного.
   Б. Все.
   В. Я знаю, но ничего не скажу.
3. Вы восхищаетесь работой другого программиста, который написал программу, способную одновременно выполнять четыре задачи. Что вы ему скажете?
   А. "Я куплю эту программу за доллар".
   Б. "Я восхищен твоей работой".
   В. "Все это благодаря потокам!"

Ответы

1. **А.** Интерфейс `Runnable` следует указать после ключевого слова `implements`. Интерфейс `Thread` используется в многопоточной программе, но его не требуется указывать в объявлении класса.
2. **Б.** Поддержка интерфейса означает, что класс реализует все методы интерфейса.
3. **В.** Именно благодаря потокам можно написать программу, которая способна одновременно выполнять несколько задач.

Упражнения

Чтобы закрепить пройденный материал, выполните следующие упражнения.

- Создайте новую версию программы `LinkRotator`, которая отображает ваши любимые шесть сайтов.
- Добавьте второй поток в программу `PrimeFinder` для нахождения первого миллиона чисел, которые не делятся без остатка на 3. Остановите счетчик времени после завершения обоих потоков.

ЗАНЯТИЕ 16
Использование внутренних классов и замыканий

На этом занятии вы узнаете:

- как добавить внутренний класс к объекту;
- как создать анонимный внутренний класс;
- как использовать класс адаптера вместе с интерфейсами;
- как создавать лямбда-выражения;
- как заменить анонимный внутренний класс лямбда-выражением.

Язык Java, появившийся в 1995 году, был ограничен в своих возможностях, что делало его простым в освоении. В те времена библиотека классов Java включала всего лишь 250 классов. Язык разрабатывался в основном для того, чтобы интерактивные программы, называемые аплетами, могли выполняться веб-браузерами. Поскольку подобной технологии прежде не существовало, язык Java получил бурное развитие и был взят на вооружение тысячами программистов.

Благодаря удачно продуманному дизайну Java превратился в универсальный язык, ставший конкурентом C++, который в те времена был наиболее популярным и распространенным языком программирования в мире.

Миллионы программистов в конечном итоге сделали свой выбор в пользу Java. Каждая новая версия этого языка расширяет возможности разработчиков, позволяя применять новые методики написания программ.

Одна из самых восхитительных новых возможностей — замыкания, или лямбда-выражения, реализующие методологию, которая называется *функциональным программированием*.

На этом занятии вы узнаете о внутренних классах и анонимных внутренних классах, благодаря которым становится возможным применять лямбда-выражения.

Внутренние классы

В Java при создании класса определяются его атрибуты и поведение. Атрибуты — это переменные класса и экземпляра, хранящие данные, а поведение — это методы, выполняющие действия над этими данными.

Класс может также включать дополнительный элемент, состоящий из атрибутов и поведения: *внутренний класс*.

Внутренние классы — это вспомогательные классы, которые находятся в содержащем их классе. У многих пользователей возникает вопрос: зачем они вообще нужны, если можно легко создать программу, которая будет содержать ровно столько новых классов, сколько вам нужно? Если вы пишете программу `CheckBook`, в которой нужны объекты для каждого чека, выписанного пользователем, вы создаете класс `Check`. Если программа поддерживает повторяющиеся ежемесячные платежи, вы добавляете класс `Autopayment`.

Внутренние классы не являются обязательными, но когда вы ознакомитесь с их возможностями, вы поймете, что они невероятно удобны.

Ниже перечислены преимущества внутренних классов.

1. Если вспомогательный класс используется только одним классом, имеет смысл определить его внутри этого класса.
2. Внутренние классы могут получать доступ к закрытым методам и переменным классов, в которых они содержатся. При реализации в виде отдельных классов это было бы невозможно.
3. Внутренний класс в программном коде располагается как можно ближе к тому месту, где он будет использоваться другим классом.

Внутренний класс создается с помощью ключевого слова `class`, подобно любому другому классу, но объявляется во включающем его классе. Обычно внутренний класс записывается рядом с объявлением переменных класса и экземпляра.

Ниже приведен пример внутреннего класса `InnerSimple`, созданного в классе `Simple`.

```
public class Simple {

   class InnerSimple {
      InnerSimple() {
         System.out.println("Я внутренний класс!");
      }
   }

   public Simple() {
      // Пустой конструктор
   }
```

```
   public static void main(String[] arguments) {
      Simple program = new Simple();
      Simple.InnerSimple inner = program.new InnerSimple();
   }
}
```

Внутренний класс структурируется подобно любому другому классу, но записывается в пределах фигурных скобок, соответствующих включающему классу.

Для создания внутреннего класса нужен объект внешнего класса. Для этого объекта вызывается оператор `new`:

```
Simple.InnerSimple inner = program.new InnerSimple();
```

Имя класса включает имя внешнего класса, точку и имя внутреннего класса. В предыдущей инструкции имя класса — `Simple.InnerSimple`.

Первый проект этого занятия появился в результате переработки приложения `PageCatalog`, созданного на занятии 14, где требовался вспомогательный класс `HomePage`. В программе `Catalog`, исходный код которой приведен в листинге 16.1, отдельный класс заменен внутренним классом.

Создайте новый класс `Catalog`, включите его в пакет `com.java24hours`, введите код из листинга 16.1 и сохраните полученный файл `Catalog.java`.

ЛИСТИНГ 16.1. Исходный код программы `Catalog.java`

```
 1: package com.java24hours;
 2:
 3: import java.net.*;
 4:
 5: public class Catalog {
 6:    class HomePage {
 7:       String owner;
 8:       URL address;
 9:       String category = "нет";
10:
11:       public HomePage(String inOwner, String inAddress)
12:             throws MalformedURLException {
13:          owner = inOwner;
14:          address = new URL(inAddress);
15:       }
16:
17:       public HomePage(String inOwner, String inAddress,
18:                       String inCategory)
19:             throws MalformedURLException {
20:
21:          this(inOwner, inAddress);
22:          category = inCategory;
```

```
23:       }
24:    }
25:
26:    public Catalog() {
27:       Catalog.HomePage[] catalog = new Catalog.HomePage[5];
28:       try {
29:          catalog[0] = new HomePage("Майк Глир",
30:             "http://www.file770.com", "научная фантастика");
31:          catalog[1] = new HomePage("Шелли Пауэрс",
32:             "http://burningbird.net", "окружающая среда");
33:          catalog[2] = new HomePage("Роджерс Кейденхед",
34:             "http://workbench.cadenhead.org", "программирование");
35:          catalog[3] = new HomePage("Таеган Годдард",
36:             "https://politicalwire.com", "политика");
37:          catalog[4] = new HomePage("Мэнтон Рид",
38:             "http://manton.org");
39:          for (int i = 0; i < catalog.length; i++) {
40:             System.out.println(catalog[i].owner + ": " +
41:                catalog[i].address + " -- " +
42:                catalog[i].category);
43:          }
44:       } catch (MalformedURLException e) {
45:          System.out.println("Ошибка: " + e.getMessage());
46:       }
47:    }
48:
49:    public static void main(String[] arguments) {
50:       new Catalog();
51:    }
52: }
```

Внутренний класс определен в строках 6–24. Он включает два конструктора: первому передается имя владельца и URI сайта (строки 11–15), а второму — имя владельца, URI и категория сайта (строки 18–23).

Класс `Catalog` использует внутренний класс (строка 27) для создания массива объектов `HomePage`. Имя внутреннего класса записывается в виде `Catalog.HomePage`.

Результат выполнения программы показан на рис. 16.1.

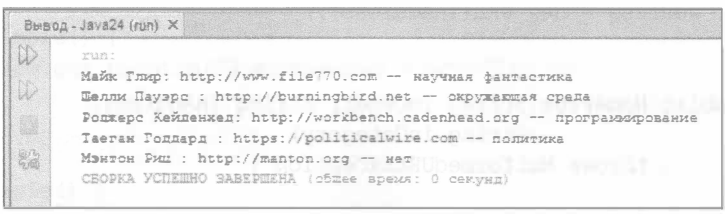


РИС. 16.1. Результат выполнения программы `Catalog`

Анонимные внутренние классы

Распространенная задача в программировании на Java — создание простого класса, который будет использоваться однократно. Для этого хорошо подходит специальная разновидность внутреннего класса — *анонимный внутренний класс*, у которого нет имени. Он одновременно и объявляется, и создается.

Чтобы использовать такой класс, нужно заменить ссылку на объект инструкцией `new`, добавить вызов конструктора и сразу же — определение класса, заключенное в фигурные скобки.

Вот пример кода, в котором не используется анонимный внутренний класс.

```
WorkerClass worker = new WorkerClass();
Thread main = new Thread(worker);
main.start();
```

Объект `worker` реализует интерфейс `Runnable` и может запускаться как поток.

Если код класса `WorkerClass` короткий и простой, а сам класс используется лишь один раз, то целесообразно объявить его как анонимный внутренний класс. Новая версия класса приведена ниже.

```
Thread main = new Thread(new Runnable() {
   public void run() {
      // Здесь поток выполняет свою работу
   }
});
main.start();
```

Анонимный внутренний класс заменил ссылку на объект `worker` следующим кодом.

```
new Runnable() {
   public void run() {
      // Здесь поток выполняет свою работу
   }
}
```

В результате выполнения этого кода создается анонимный класс, который реализует интерфейс `Runnable` и переопределяет его метод `run()`. Инструкции, заданные в этом методе, будут выполнять любую работу, требуемую классом.

Эту концепцию будет проще понять, если рассмотреть более полный пример, демонстрирующий способы создания анонимного внутреннего класса и особенности его применения.

Приложение Java может получать данные с клавиатуры с помощью пакетов Swing. Контроль за вводом данных с клавиатуры осуществляется с помощью объекта, реализующего интерфейс `KeyListener`.

Класс, который реализует этот интерфейс, должен реализовать три метода — `keyTyped()`, `keyPressed()` и `keyReleased()`, как продемонстрировано в следующем коде.

```
public void keyTyped(KeyEvent input) {
   char key = input.getKeyChar();
   keyLabel.setText("Вы нажали " + key);
}
public void keyPressed(KeyEvent txt) {
   // Ничего не делать
}
public void keyReleased(KeyEvent txt) {
   // Ничего не делать
}
```

В том же классе, в котором находятся эти инструкции, регистрируется слушатель событий клавиатуры:

```
keyText.addKeyListener(this);
```

С помощью анонимного внутреннего класса и класса `KeyAdapter` из пакета `java.awt.event` обеспечивается более удобный способ создания слушателя и его добавления к графическому интерфейсу пользователя.

Класс `KeyAdapter` реализует интерфейс `KeyListener` с заглушками всех трех его методов. Это облегчает создание слушателя событий клавиатуры, поскольку можно создать подкласс, переопределяющий только нужный метод.

Вот каркас слушателя для приложения `KeyViewer`.

```
public class KeyViewerListener extends KeyAdapter {
   public void keyTyped(KeyEvent input) {
      // Выполнение каких-либо действий
   }
}
```

Этот класс может быть зарегистрирован в качестве слушателя событий клавиатуры.

```
KeyViewerListener kvl = new KeyViewerListener();
keyText.addKeyListener(kvl);
```

Для реализации данного подхода требуется отдельный вспомогательный класс `KeyViewerListener`, а также объект этого класса, который должен быть создан и присвоен переменной.

Другой подход заключается в создании слушателя в качестве анонимного внутреннего класса.

```
keyText.addKeyListener(new KeyAdapter() {
   public void keyTyped(KeyEvent input) {
      char key = input.getKeyChar();
      keyLabel.setText("Вы нажали " + key);
   }
});
```

Слушатель создается анонимно путем вызова `new KeyAdapter()`, за которым следует определение класса. Класс переопределяет метод `keyTyped()`, поэтому при нажатии клавиши вызывается метод `getKeyChar()`. Затем выводится сообщение, созданное путем установки содержимого объекта `keyLabel`, который является компонентом `JLabel`.

Анонимный внутренний класс делает то, что обычный вспомогательный класс делать не может, — получает доступ к переменной экземпляра `keyLabel`, которая относится к классу `KeyViewer`. Внутренние классы могут получать доступ к методам и переменным включающего их класса.

В среде NetBeans создайте новый класс `KeyViewer`, включите его в пакет `com.java24hours`, введите код из листинга 16.2 и сохраните полученный файл `KeyViewer.java`.

ЛИСТИНГ 16.2. Исходный код программы **`KeyViewer.java`**

```
 1: package com.java24hours;
 2:
 3: import javax.swing.*;
 4: import java.awt.event.*;
 5: import java.awt.*;
 6:
 7: public class KeyViewer extends JFrame {
 8:    JTextField keyText = new JTextField(80);
 9:    JLabel keyLabel = new JLabel("Нажмите любую клавишу " +
10:                                 "в текстовом поле.");
11:    public KeyViewer() {
12:       super("KeyViewer");
13:       setSize(350, 100);
14:       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
15:       keyText.addKeyListener(new KeyAdapter() {
16:          public void keyTyped(KeyEvent input) {
17:             char key = input.getKeyChar();
18:             keyLabel.setText("Вы нажали " + key);
19:          }
20:       });
21:       BorderLayout bord = new BorderLayout();
22:       setLayout(bord);
```

```
23:        add(keyLabel, BorderLayout.NORTH);
24:        add(keyText, BorderLayout.CENTER);
25:        setVisible(true);
26:     }
27:
28:     public static void main(String[] arguments) {
29:        new KeyViewer();
30:     }
31: }
```

Анонимный внутренний класс создается и используется в строках 15–20. Он отслеживает ввод данных с клавиатуры, используя единственный метод интерфейса `KeyListener`, который необходим в программе, и сообщает о том, какая клавиша нажата, путем обновления значения переменной `keyLabel`.

Результат выполнения программы показан на рис. 16.2.

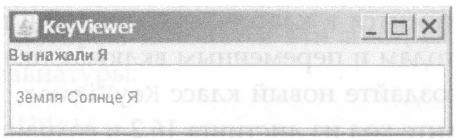


РИС. 16.2. Результат отслеживания ввода с клавиатуры с помощью приложения `KeyViewer`

Анонимные внутренние классы не могут определять конструкторы, поэтому они более ограничены, чем обычные внутренние классы. Это сложное средство программирования, которое нелегко освоить, но благодаря им можно сделать программу более компактной.

Замыкания

В этом разделе будет рассмотре́но наиболее востребованное средство языка Java — *замыкания*, также называемые *лямбда-выражениями*. Они позволяют создать объект с единственным методом, используя лишь оператор ->, при условии соблюдения нескольких правил.

```
Runnable runner = () -> { System.out.println("Выполнить!"); };
```

Эта единственная строка кода создает объект, который реализует интерфейс `Runnable` и создает следующий метод `run()`.

```
void run() {
   System.out.println("Выполнить!");
}
```

В лямбда-выражении инструкция, находящаяся справа от оператора ->, становится методом, который реализует интерфейс.

Все это можно сделать лишь в том случае, если интерфейс содержит единственный реализуемый метод. Пример — интерфейс `Runnable`, содержащий только метод `run()`. В Java интерфейс с одним методом называется *функциональным интерфейсом*.

В лямбда-выражении есть также часть, стоящая слева от оператора ->. Она задает список аргументов, передаваемых методу функционального интерфейса. В предыдущем примере это пустой набор скобок. Поскольку метод `run()` не имеет аргументов в интерфейсе `Runnable`, в лямбда-выражении аргументы тоже не требуются.

Рассмотрим второй пример лямбда-выражения, в котором в круглых скобках задан один аргумент.

```
ActionListener al = (ActionEvent act) -> {
   System.out.println(act.getSource());
}
```

Это выражение эквивалентно следующему коду в объекте, который реализует интерфейс `ActionListener` из пакета `java.awt.event`.

```
public void actionPerformed(ActionEvent act) {
   System.out.println(act.getSource());
}
```

Интерфейс `ActionListener` предназначен для реагирования на события действий, таких как щелчок на кнопке. Данный функциональный интерфейс содержит единственный метод `actionPerformed(`*`ActionEvent`*`)`. Его аргументом является объект `ActionEvent`, описывающий действие пользователя, которое вызвало событие.

В правой части лямбда-выражения задается метод `actionPerformed()`, состоящий из одной инструкции. Он отображает информацию о компоненте интерфейса, вызвавшем событие.

В левой части выражения указано, что аргументом метода является объект `act` класса `ActionEvent`. Этот объект используется в теле метода. Создается впечатление, что ссылка на объект `act` в левой части выражения находится за пределами области видимости метода, задаваемого в правой части. На самом деле это не так, поскольку замыкания позволяют коду ссылаться на переменные другого метода вне их области видимости.

Одно из преимуществ, связанных с использованием лямбда-выражений, заключается в том, что они существенно уменьшают объем кода. Всего лишь одно выражение создает объект и реализует интерфейс.

Благодаря поддержке в Java целевой типизации можно еще больше сократить код.

В лямбда-выражении можно вывести класс аргумента или аргументов, отправленных методу. Рассмотрим еще один пример. Поскольку функциональный интерфейс `ActionListener` включает метод, имеющий единственный аргумент типа `ActionEvent`, имя этого класса можно не указывать. Приведем сокращенную версию лямбда-выражения.

```
ActionListener al = (act) -> {
   System.out.println(act.getSource());
}
```

Следующие две программы более наглядно продемонстрируют трансформацию программного кода, вызванную появлением замыканий в Java.

Приложение `ColorFrame`, код которого приведен в листинге 16.3, отображает три кнопки, позволяющие изменять цвет фрейма. В этой версии приложения для отслеживания щелчков пользователя на трех кнопках вместо лямбда-выражения используется анонимный внутренний класс.

В среде NetBeans создайте новый класс `ColorFrame`, включив его в пакет `com.java24hours`. Ввведите код из листинга 16.3 и сохраните полученный файл `ColorFrame.java`.

ЛИСТИНГ 16.3. Исходный код программы `ColorFrame.java`

```
 1: package com.java24hours;
 2:
 3: import java.awt.*;
 4: import java.awt.event.*;
 5: import javax.swing.*;
 6:
 7: public class ColorFrame extends JFrame {
 8:    JButton red, green, blue;
 9:
10:    public ColorFrame() {
11:       super("ColorFrame");
12:       setSize(322, 122);
13:       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14:       FlowLayout flo = new FlowLayout();
15:       setLayout(flo);
16:       red = new JButton("Красный");
17:       add(red);
18:       green = new JButton("Зеленый");
19:       add(green);
20:       blue = new JButton("Синий");
21:       add(blue);
22:       // Начало анонимного внутреннего класса
23:       ActionListener act = new ActionListener() {
24:          public void actionPerformed(ActionEvent event) {
25:             if (event.getSource() == red) {
```

```
26:                getContentPane().setBackground(Color.RED);
27:             }
28:             if (event.getSource() == green) {
29:                getContentPane().setBackground(Color.GREEN);
30:             }
31:             if (event.getSource() == blue) {
32:                getContentPane().setBackground(Color.BLUE);
33:             }
34:          }
35:       };
36:       // Конец анонимного внутреннего класса
37:       red.addActionListener(act);
38:       green.addActionListener(act);
39:       blue.addActionListener(act);
40:       setVisible(true);
41:    }
42:
43:    public static void main(String[] arguments) {
44:       new ColorFrame();
45:    }
46: }
```

Результат выполнения этой программы показан на рис. 16.3.

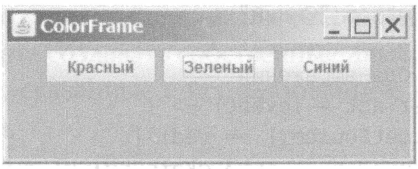


РИС. 16.3. Отслеживание событий интерфейса с помощью анонимного внутреннего класса

В строках 23–35 создается слушатель событий для класса `ColorFrame` с использованием анонимного внутреннего класса. Этот класс без имени реализует единственный метод `actionPerformed(ActionEvent)` интерфейса `ActionListener`.

В методе `actionPerformed()` панель содержимого фрейма выбирается путем вызова метода `getContentPane()`. Анонимные внутренние классы имеют доступ к методам и переменным экземпляра окружающего класса. Отдельный вспомогательный класс не имел бы такого доступа.

Метод панели содержимого `setBackground(цвет)` изменяет цвет фона фрейма. При этом цвет трех кнопок остается неизменным.

Теперь рассмотрим приложение `NewColorFrame`. В среде NetBeans создайте новый класс `NewColorFrame` и включите его в пакет `com.java24hours`.

Введите код из листинга 16.4 и сохраните полученный файл `NewColorFrame.java`.

ЛИСТИНГ 16.4. Исходный код программы `NewColorFrame.java`

```
 1: package com.java24hours;
 2:
 3: import java.awt.*;
 4: import java.awt.event.*;
 5: import javax.swing.*;
 6:
 7: public class NewColorFrame extends JFrame {
 8:    JButton red, green, blue;
 9:
10:    public NewColorFrame() {
11:       super("NewColorFrame");
12:       setSize(322, 122);
13:       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14:       FlowLayout flo = new FlowLayout();
15:       setLayout(flo);
16:       red = new JButton("Красный");
17:       add(red);
18:       green = new JButton("Зеленый");
19:       add(green);
20:       blue = new JButton("Синий");
21:       add(blue);
22:       // Начало лямбда-выражения
23:       ActionListener act = (event) -> {
24:          if (event.getSource() == red) {
25:             getContentPane().setBackground(Color.RED);
26:          }
27:          if (event.getSource() == green) {
28:             getContentPane().setBackground(Color.GREEN);
29:          }
30:          if (event.getSource() == blue) {
31:             getContentPane().setBackground(Color.BLUE);
32:          }
33:       };
34:       // Конец лямбда-выражения
35:       red.addActionListener(act);
36:       green.addActionListener(act);
37:       blue.addActionListener(act);
38:       setVisible(true);
39:    }
40:
41:    public static void main(String[] arguments) {
42:       new NewColorFrame();
43:    }
44: }
```

Приложение `NewColorFrame` реализует слушатель событий (строки 23–33). Вовсе не обязательно знать имя метода в интерфейсе `ActionListener`, чтобы использовать его в программе. Также нет необходимости указывать класс `ActionEvent` для аргумента метода.

Лямбда-выражения реализуют методологию функционального программирования, появившуюся в версии Java 8.

В этом разделе вы ознакомились с базовым синтаксисом лямбда-выражений и двумя наиболее распространенными способами их использования в программах.

Функциональное программирование на Java является столь мощной и революционной парадигмой, что на эту тему написаны целые книги.

На данный момент вы сможете распознавать лямбда-выражения, а также использовать их для реализации произвольных функциональных интерфейсов, включающих единственный метод.

Резюме

Внутренние и анонимные внутренние классы, а также лямбда-выражения являются наиболее сложными для изучения средствами языка Java. Эти средства предназначены в первую очередь для опытных программистов, которые готовы использовать сложные методики для сокращения объема кода.

Прежде чем вы начнете создавать собственные лямбда-выражения, следует оценить преимущества, связанные с использованием внутренних и анонимных внутренних классов.

Внутренний (не анонимный) класс структурирован подобно отдельному вспомогательному классу, но определен в другом классе вместе с переменными экземпляра, переменными класса, методами экземпляра и методами класса. В отличие от вспомогательного класса, внутренний класс может получать доступ к закрытым переменным и методам окружающего класса.

Благодаря использованию анонимного внутреннего класса устраняется необходимость в создании объекта однократно используемого класса, например слушателя событий, связанного с компонентом графического интерфейса в Swing.

Лямбда-выражения напоминают анонимные внутренние классы, но для их создания требуется лишь указать оператор `->`. Благодаря лямбда-выражениям возможности программирования на Java многократно возрастают.

Вопросы и ответы

В. Анонимные внутренние классы — довольно сложная тема. Обязательно ли использовать их при разработке программ?

О. Нет. Как и в случае с другими сложными средствами языка Java, вовсе не обязательно использовать анонимные внутренние классы, если можно решить поставленную задачу другим способом.

Но как бы там ни было, следует ознакомиться с этим средством, поскольку вы наверняка столкнетесь с ним в программировании на Java.

При изучении кода приложений Java, написанных опытными программистами, вы будете часто видеть как внутренние, так и анонимные внутренние классы. Поэтому, даже если вы пока не готовы самостоятельно создавать такие классы, имеет смысл узнать о том, что они собой представляют и как работают.

Коллоквиум

Проверьте знание материала, рассмотренного на этом занятии, ответив на следующие вопросы.

Контрольные вопросы

1. Почему некоторые внутренние классы являются анонимными?
   - **А.** Они реализуют интерфейс.
   - **Б.** У них отсутствует имя.
   - **В.** И то и другое.
2. Как называется интерфейс, который содержит только один метод?
   - **А.** Абстрактный интерфейс.
   - **Б.** Класс.
   - **В.** Функциональный интерфейс.
3. Как называется процесс подбора лямбда-выражением класса аргумента для метода?
   - **А.** Целевая типизация.
   - **Б.** Приведение типа.
   - **В.** Вывод класса.

Ответы

1. **В.** Анонимный внутренний класс реализует интерфейс с помощью ключевого слова `new`, при этом объявление самого класса пропускается.

2. **B.** В Java 8 и более поздних версиях — функциональный интерфейс. В более ранних версиях эти классы назывались интерфейсами одного абстрактного метода.
3. **A.** Целевая типизация позволяет определить класс любого аргумента метода функционального интерфейса.

Упражнения

В завершение этого занятия выполните следующие упражнения.

- Перепишите класс `LinkRotator` (см. занятие 15) с учетом использования лямбда-выражения для слушателя событий.
- Добавьте четвертый фоновый цвет `Color.YELLOW` в приложение `NewColorFrame`.

Часть V
Разработка графического интерфейса пользователя

В этой части...

- Занятие 17 Создание простого пользовательского интерфейса
- Занятие 18 Компоновка элементов интерфейса
- Занятие 19 Получение данных от пользователя

ЗАНЯТИЕ 17
Создание простого пользовательского интерфейса

На этом занятии вы узнаете...

- как создавать компоненты пользовательского интерфейса, такие как кнопки;
- как создавать метки, текстовые поля и другие компоненты;
- как группировать компоненты;
- как помещать компоненты в другие компоненты;
- как прокручивать компоненты по вертикали и по горизонтали;
- как открывать и закрывать окна.

На этом занятии вы создадите свой первый графический интерфейс (Graphical User Interface, GUI) с помощью Java.

Пользователи компьютеров чаще всего рассчитывают на то, что программы оснащены графическим интерфейсом и принимают ввод пользовательских данных с помощью мыши. И даже несмотря на то, что некоторые пользователи до сих пор работают в средах с поддержкой командной строки, таких как командные оболочки Linux/Unix, большинству вряд ли понравится программа, которая не поддерживает графический интерфейс, как в Windows или Mac OS.

В Java разрабатывать графические интерфейсы можно с помощью Swing. Это библиотека классов, представляющих различные кнопки, текстовые поля, ползунки и другие компоненты, которые могут быть частью графического интерфейса. В библиотеку также входят классы, которые нужны для обработки пользовательских данных, принимаемых этими компонентами.

На этом и следующем занятиях мы будем создавать и структурировать графические интерфейсы в Java. Затем на занятии 19 мы "научим" эти интерфейсы поддерживать щелчки мышью и другие способы взаимодействия с пользователем.

AWT (Abstract Windowing Toolkit) и Swing

Поскольку Java является кросс-платформенным языком, позволяющим писать программы для многих операционных систем, инструменты, предназначенные для создания графических интерфейсов, должны обеспечивать необходимую степень гибкости и поддерживать не только интерфейсы в стиле Windows или Macintosh, но и универсальные решения, которые могли бы выполняться на любой платформе.

Разработка пользовательского интерфейса в Java ведется с помощью Swing и более ранней библиотеки классов, которая называется AWT (Abstract Windowing Toolkit). Эти классы позволяют создавать графические компоненты и получать данные, вводимые пользователем.

В Swing есть все, что нужно для написания программ, имеющих графический интерфейс. С помощью библиотечных классов можно создавать следующие компоненты:

- кнопки, флажки, метки и другие простые компоненты;
- текстовые поля, ползунки и другие более сложные компоненты;
- всплывающие и раскрывающиеся меню;
- окна, фреймы, диалоговые окна, панели и окна аплетов.

ПРИМЕЧАНИЕ

В Swing доступны десятки компонентов, которые можно настраивать множеством способов. В книге описаны лишь самые распространенные компоненты и их наиболее часто используемые методы. Вы сможете узнать больше о каждом компоненте и найти описание новых методов в официальной документации Oracle, посвященной библиотеке классов Java. Эта документация доступна по следующему адресу:

`https://docs.oracle.com/javase/10/docs/api/overview-summary.html`

На сайте Oracle доступна также дополнительная справочная документация по каждому классу и интерфейсу в Java 10.

Использование компонентов

В Java каждому компоненту графического интерфейса соответствует класс в пакете Swing. Кнопки представлены с помощью класса `JButton`, окна — с помощью класса `JWindow`, текстовые поля — с помощью класса `JTextField` и т.п.

Чтобы сформировать и отобразить интерфейс на экране, нужно создать определенные объекты, настроить их атрибуты и вызвать их методы. При этом используются те же методики, что и в объектно-ориентированном программировании (ООП).

В процессе объединения компонентов графического интерфейса вы будете использовать два вида объектов: компоненты и контейнеры. *Компонент* — это

отдельный элемент пользовательского интерфейса, такой как кнопка или ползунок. *Контейнер* — это компонент, который можно использовать для хранения других компонентов.

Первый шаг на пути к формированию интерфейса заключается в создании контейнера, предназначенного для хранения компонентов. В приложении в качестве такого контейнера чаще всего выступает окно или фрейм.

Окна и фреймы

Окна и фреймы — это контейнеры, которые могут отображаться в интерфейсе пользователя и включать другие компоненты. *Окна* — это простые контейнеры, лишенные строки заголовка и каких-либо других кнопок, которые обычно отображаются вдоль верхнего края. *Фреймы* — это окна, имеющие все привычные оконные элементы, такие как кнопки для закрытия, развертывания и свертывания окна.

Контейнеры создаются с помощью классов Swing `JWindow` и `JFrame`. Чтобы не нужно было каждый раз указывать полное имя класса, используйте следующую инструкцию:

```
import javax.swing.*;
```

Она импортирует имена классов из пакета `javax.swing`. Существуют и другие пакеты, которые можно использовать вместе со Swing.

Один из способов создания фрейма в приложении Java — сделать приложение подклассом класса `JFrame`. Программа наследует поведение, требуемое для функционирования в качестве фрейма. Вот как создать подкласс класса `JFrame`.

```
import javax.swing.*;

public class MainFrame extends JFrame {
   public MainFrame() {
      // Настройка фрейма
   }
}
```

Здесь создается только оболочка фрейма, но сам фрейм не настраивается. В конструкторе фрейма нужно выполнить несколько операций:

- вызвать конструктор суперкласса, `JFrame`;
- присвоить заголовок фрейму;
- установить размер фрейма;
- настроить внешний вид фрейма;
- определить, что будет происходить при закрытии фрейма пользователем.

Также необходимо сделать фрейм видимым, если только по каким-то причинам он не должен отображаться при запуске приложения.

Первое, что должен сделать конструктор, — вызвать один из конструкторов класса `JFrame` с помощью метода `super()`, например:

```
super();
```

Предыдущая инструкция вызывает конструктор класса `JFrame` без аргументов. Можно также вызвать конструктор с заголовком фрейма в качестве аргумента:

```
super("Основной фрейм");
```

Заголовок фрейма отображается в строке заголовка вдоль верхнего края окна.

Если заголовок не был задан в конструкторе, то впоследствии можно вызвать метод фрейма `setTitle(`*`строка`*`)` с заголовком в качестве аргумента:

```
setTitle("Основной фрейм");
```

Чтобы настроить размер фрейма, нужно вызвать метод `setSize(`*`целое_число`*`, `*`целое_число`*`)` с двумя аргументами: шириной и высотой. Следующая инструкция создает фрейм шириной 350 пикселей и высотой 125 пикселей:

```
setSize(350, 125);
```

Еще один способ настроить размер фрейма — заполнить его компонентами, а затем вызвать метод фрейма `pack()` без аргументов.

```
pack();
```

С помощью метода `pack()` устанавливается размер фрейма, который будет достаточен для хранения каждого компонента предпочтительного размера внутри фрейма (но не больше). Каждый компонент интерфейса имеет предпочтительный размер, хотя иногда это обстоятельство игнорируется (в зависимости от расположения компонентов в интерфейсе). Перед вызовом метода `pack()` не нужно явно указывать размер фрейма, поскольку метод сам устанавливает подходящий размер.

Каждый фрейм отображается с кнопкой в строке заголовка, которая служит для закрытия фрейма. В Windows эта кнопка обозначается значком × в правом верхнем углу фрейма. Чтобы задать действие, происходящее после щелчка на кнопке, вызовите метод фрейма `setDefaultCloseOperation(`*`целое_число`*`)` с одной из четырех констант класса `JFrame` в качестве аргумента.

- `EXIT_ON_CLOSE`. Выход из программы после щелчка на кнопке.
- `DISPOSE_ON_CLOSE`. Закрыть фрейм и продолжить выполнение приложения.

- DO_NOTHING_ON_CLOSE. Оставить фрейм открытым и продолжить выполнение приложения.
- HIDE_ON_CLOSE. Закрыть фрейм и продолжить выполнение приложения.

На практике чаще всего применяется первый вариант, поскольку закрытие фрейма обычно означает, что приложение должно прекратить выполнение.

```
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Внешний вид графического интерфейса, созданного с помощью Swing, можно настраивать. При этом используется визуальная тема, которая определяет внешний вид и поведение кнопок и других компонентов.

В Java доступны расширенные возможности по настройке пользовательского интерфейса, реализуемые с помощью оболочки Nimbus. Чтобы воспользоваться этими возможностями в классе, нужно активизировать Nimbus. Для настройки внешнего вида интерфейса нужно вызвать метод `setLookAndFeel()` класса `UIManager`, определенного в главном пакете Swing. Метод имеет единственный аргумент — полное имя класса графической оболочки.

Следующая инструкция активизирует оболочку Nimbus для настройки внешнего вида интерфейса.

```
UIManager.setLookAndFeel("javax.swing.plaf.nimbus.NimbusLookAndFeel");
```

Теперь осталось отобразить фрейм. Для этого необходимо вызвать метод `setVisible()` фрейма, указав `true` в качестве аргумента:

```
setVisible(true);
```

Это приведет к открытию фрейма с заданными значениями ширины и высоты. Чтобы скрыть фрейм, вызовите метод `setVisible()` с аргументом `false`.

Листинг 17.1 содержит исходный код, описанный в этом разделе. Создайте класс `SalutonFrame`, включите его в пакет `com.java24hours`, введите код из листинга 17.1 и сохраните полученный файл `SalutonFrame.java`.

ЛИСТИНГ 17.1. Исходный код программы `SalutonFrame.java`

```
1: package com.java24hours;
2:
3: import javax.swing.*;
4:
5: public class SalutonFrame extends JFrame {
6:     public SalutonFrame() {
7:         super("Saluton mondo!");
8:         setLookAndFeel();
9:         setSize(450, 200);
```

```
10:       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11:       setVisible(true);
12:    }
13:
14:    private void setLookAndFeel() {
15:       try {
16:          UIManager.setLookAndFeel(
17:             "javax.swing.plaf.nimbus.NimbusLookAndFeel"
18:          );
19:       } catch (Exception exc) {
20:          // Игнорировать ошибки
21:       }
22:    }
23:
24:    public static void main(String[] arguments) {
25:       SalutonFrame frame = new SalutonFrame();
26:    }
27: }
```

В строках 24–26 листинга 17.1 определяется метод `main()`, в котором создается фрейм приложения. После запуска приложения отобразится окно, показанное на рис. 17.1.

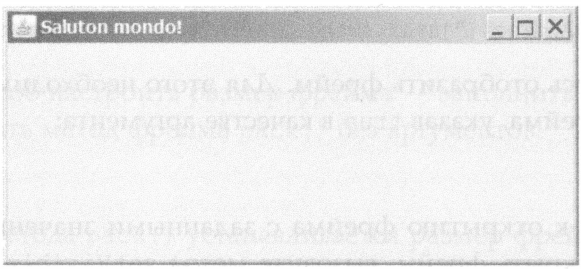


РИС. 17.1. Отображение фрейма в приложении

Единственное, что отображает приложение `SalutonFrame`, — это строка заголовка `"Saluton mondo!"`. На данный момент фрейм является пустым окном, поскольку не содержит компонентов.

Чтобы добавить компоненты во фрейм, нужно создать компоненты и включить их в контейнер. Каждый компонент имеет метод `add()`, у которого единственный аргумент — отображаемый компонент.

Класс `SalutonFrame` содержит метод `setLookAndFeel()`, который назначает Nimbus в качестве темы оформления фрейма. Для этого в строках 16–18 вызывается метод `setLookAndFeel()` класса `UIManager`. Вызов данного метода помещается в блок `try-catch`, что позволяет обрабатывать возникающие ошибки. В случае каких-либо ошибок вместо внешнего вида, заданного оболочкой Nimbus, применяется внешний вид, заданный по умолчанию.

СОВЕТ

В строках 16–18 применяется необычный прием: здесь одна инструкция разбита на три строки. Это сделано для улучшения наглядности кода. Компилятор Java игнорирует дополнительные пробелы, поэтому инструкция может занимать несколько строк. Главное, чтобы она была корректной и заканчивалась точкой с запятой.

Кнопки

Один из простых компонентов, который можно добавить в контейнер, — это объект `JButton`. Подобно другим компонентам, с которыми вы будете работать на этом занятии, он является частью пакета `java.awt.swing`.

Объект `JButton` — это кнопка с меткой, которая описывает выполняемое кнопкой действие. Метка может быть текстовой, графической или комбинированной. Следующая инструкция создает объект `JButton` с именем `okButton` и присваивает ему текстовую метку `OK`:

```
JButton okButton = new JButton("OK");
```

После создания компонента, такого как `JButton`, его нужно добавить в контейнер путем вызова соответствующего метода `add()`:

```
add(okButton);
```

При добавлении компонентов в контейнер не нужно указывать, где конкретно в контейнере они должны отображаться. Расположение компонентов в контейнере задается с помощью объекта, который называется *менеджер компоновки*. Простейший из таких менеджеров — класс `FlowLayout` из пакета `java.awt`.

Чтобы контейнер мог использовать определенный менеджер компоновки, нужно сначала создать объект класса данного менеджера. Объект `FlowLayout` можно создать путем вызова его конструктора без аргументов.

```
FlowLayout flo = new FlowLayout();
```

После создания менеджера компоновки вызовите метод `setLayout()`, чтобы связать менеджер с контейнером:

```
setLayout(flo);
```

Эта инструкция задает объект `flo` в качестве менеджера компоновки.

Следующее приложение Java отображает фрейм с тремя кнопками. Создайте класс `Playback`, включите его в пакет `com.java24hours`, введите код из листинга 17.2 и сохраните полученный файл `Playback.java`.

Программа `Playback` создает менеджер компоновки `FlowLayout` (строка 12) и подключает его к фрейму (строка 13). После добавления трех кнопок

во фрейм (строки 17–19) они будут перекомпонованы с помощью этого менеджера.

ЛИСТИНГ 17.2. Исходный код программы `Playback.java`

```
 1: package com.java24hours;
 2:
 3: import javax.swing.*;
 4: import java.awt.*;
 5:
 6: public class Playback extends JFrame {
 7:    public Playback() {
 8:       super("Playback");
 9:       setLookAndFeel();
10:       setSize(450, 200);
11:       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12:       FlowLayout flo = new FlowLayout();
13:       setLayout(flo);
14:       JButton play = new JButton("Воспроизведение");
15:       JButton stop = new JButton("Останов");
16:       JButton pause = new JButton("Пауза");
17:       add(play);
18:       add(stop);
19:       add(pause);
20:       setVisible(true);
21:    }
22:
23:    private void setLookAndFeel() {
24:       try {
25:          UIManager.setLookAndFeel(
26:             "javax.swing.plaf.nimbus.NimbusLookAndFeel"
27:          );
28:       } catch (Exception exc) {
29:          // Игнорировать ошибки
30:       }
31:    }
32:
33:    public static void main(String[] arguments) {
34:       Playback frame = new Playback();
35:    }
36: }
```

После запуска приложения результат будет примерно таким, как на рис. 17.2. Несмотря на то что можно щелкать на кнопках, в ответ ничего не происходит, поскольку программа не содержит никаких методов, предназначенных для получения пользовательских данных и реагирования на них. Дополнительные сведения по этой теме будут приведены на занятии 19.

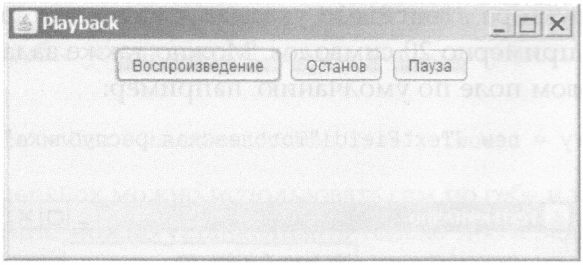


РИС. 17.2. Отображение кнопок в графическом интерфейсе

Аналогичным образом можно добавить в контейнер самые разные компоненты Swing.

ПРИМЕЧАНИЕ

Поскольку на этом занятии рассматривается множество компонентов пользовательского интерфейса, полный исходный код, требуемый для создания каждого рисунка, не приводится. Соответствующие программы можно найти на сайте издательства, адрес которого указан во введении.

Метки и текстовые поля

Компонент `JLabel` отображает информацию, которую пользователь не может изменить. Это может быть текст, графика или их сочетание. Такие компоненты часто используются для маркирования других компонентов интерфейса, в особенности текстовых полей.

Компонент `JTextField` представляет собой область, в которой пользователь может ввести единственную строку текста. При создании текстового поля можно настроить его ширину.

Следующие инструкции создают компоненты `JLabel` и `JTextField`, а затем добавляют их в контейнер.

```
JLabel pageLabel = new JLabel("Адрес веб-страницы: ", JLabel.RIGHT);
JTextField pageAddress = new JTextField(20);
FlowLayout flo = new FlowLayout();
setLayout(flo);
add(pageLabel);
add(pageAddress);
```

На рис. 17.3 показаны метка и текстовое поле, расположенные рядом.

Метке `pageLabel` присвоено значение `"Адрес веб-страницы: "`. Конструктору компонента также передается аргумент `JLabel.RIGHT`, задающий выравнивание метки по правому краю. Значение `JLabel.LEFT` приводит к выравниванию метки по левому краю, а значение `JLabel.CENTER` — по центру.

Аргумент конструктора `JTextField` указывает на то, что ширина тестового поля составляет примерно 20 символов. Можно также задать текст, отображаемый в текстовом поле по умолчанию, например:

```
JTextField country = new JTextField("Тоголезская республика", 29);
```

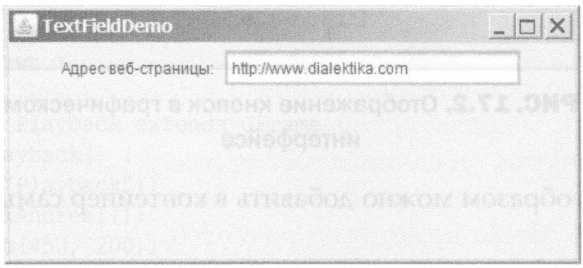


РИС. 17.3. Отображение метки и текстового поля

Эта инструкция создаст объект `JTextField`, ширина которого составит 29 символов. В поле данного объекта будет находиться текст `"Тоголезская республика"`.

Для получения текста, хранящегося в поле, используется метод `getText()`, который возвращает строку.

```
String countryChoice = country.getText();
```

Как нетрудно догадаться, объекту можно также присвоить текст с помощью соответствующего метода:

```
country.setText("Отдельная таможенная территория Тайваня, Пенгу,
                 Кинмен и Мацу");
```

Эта инструкция задает официальное название Китайского Тайбэя. Это название является длиннейшим в мире, на втором месте находится официальное название Великобритании: Соединенное Королевство Великобритании и Северной Ирландии.

Флажки

Компонент `JCheckBox` представляет собой квадратик, находящийся после строки текста. Его можно выделять меткой либо сбрасывать. Следующие инструкции создают объект `JCheckBox` и добавляют его в контейнер.

```
JCheckBox jumboSize = new JCheckBox("Jumbo Size");
FlowLayout flo = new FlowLayout();
setLayout(flo);
add(jumboSize);
```

Аргумент конструктора `JCheckBox()` задает текст, который отображается возле флажка. Чтобы установить флажок, воспользуйтесь следующей инструкцией:

```
JCheckBox jumboSize = new JCheckBox("Jumbo Size", true);
```

Компонент `JCheckBox` можно использовать сам по себе или в составе группы. Флажки в группе можно устанавливать только поочередно. Чтобы включить объект `JCheckBox` в группу, нужно создать объект `ButtonGroup`. Рассмотрим следующий код.

```
JCheckBox jumboSize = new JCheckBox("Jumbo Size", true);
JCheckBox frogLegs = new JCheckBox("Frog Leg Grande", true);
JCheckBox fishTacos = new JCheckBox("Fish Taco Platter", false);
JCheckBox emuNuggets = new JCheckBox("Emu Nuggets", false);
FlowLayout flo = new FlowLayout();
ButtonGroup meals = new ButtonGroup();
meals.add(jumboSize);
meals.add(frogLegs);
meals.add(fishTacos);
meals.add(emuNuggets);
setLayout(flo);
add(jumboSize);
add(frogLegs);
add(fishTacos);
add(emuNuggets);
```

Этот код создает четыре флажка, сгруппированных в объекте `ButtonGroup`, который называется `meals` (рис. 17.4).

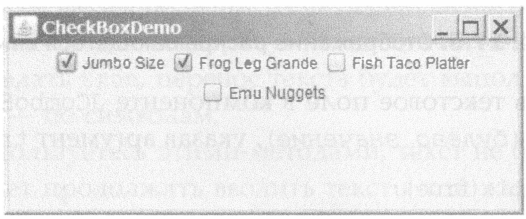


РИС. 17.4. Отображение флажков

Изначально установлены флажки `Jumbo Size` и `Frog Leg Grande`, но если пользователь установит один из оставшихся флажков, то автоматически отменится установка флажка `Frog Leg Grande`.

Раскрывающиеся списки

Компонент `JComboBox` представляет собой раскрывающийся список с возможностью ввода текста. Если включены оба режима, то можно выбрать нужный вариант с помощью мыши либо ввести текст с клавиатуры.

Раскрывающийся список имеет то же предназначение, что и группа флажков. Разница заключается в том, что в случае свернутого раскрывающегося списка отображается лишь один из вариантов выбора.

Чтобы отобразить раскрывающийся список, создайте объект `JComboBox` и добавьте в него варианты выбора, как показано ниже.

```
JComboBox profession = new JComboBox();
FlowLayout flo = new FlowLayout();
profession.addItem("Мясник");
profession.addItem("Пекарь");
profession.addItem("Изготовитель подсвечников");
profession.addItem("Лучник");
profession.addItem("Боксер");
profession.addItem("Разработчик документации");
setLayout(flo);
add(profession);
```

В этом примере создается компонент `JComboBox`, отображающий шесть вариантов выбора (рис. 17.5). Как только будет выбран один из вариантов, он тут же отобразится в поле списка.

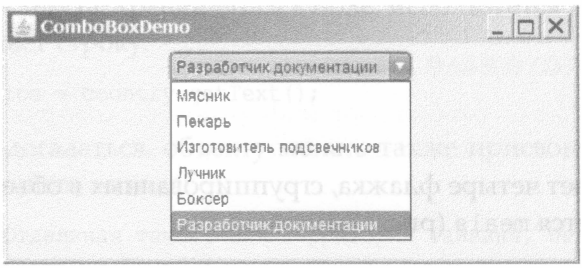


РИС. 17.5. Отображение раскрывающегося списка

Чтобы включить текстовое поле в компоненте `JComboBox`, нужно вызвать метод `setEditable(`*булево_значение*`)`, указав аргумент `true`.

```
profession.setEditable(true);
```

Этот метод следует вызвать перед добавлением компонента в контейнер.

Текстовые области

Компонент `JTextArea` — это текстовое поле, в которое можно вводить несколько строк. Ширина и высота компонента задаются пользователем. Следующие инструкции создают компонент `JTextArea` шириной 40 символов и высотой 8 строк и добавляют его в контейнер.

```
JTextArea comments = new JTextArea(8, 40);
FlowLayout flo = new FlowLayout();
```

```
setLayout(flo);
add(comments);
```

На рис. 17.6 показана текстовая область, отображаемая во фрейме.

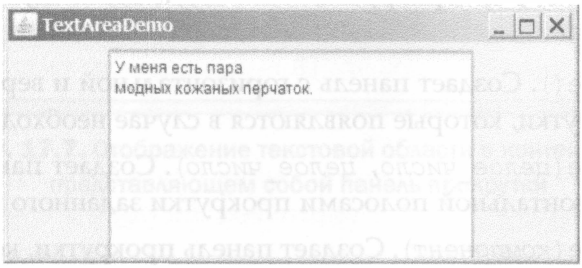


РИС. 17.6. Отображение текстовой области

В конструкторе `JTextArea()` можно указать строку, отображаемую в текстовой области. Символ новой строки `\n` позволяет переносить текст на новую строку.

```
JTextArea comments = new JTextArea("У меня есть пара\n" +
                                   "модных кожаных перчаток.", 10, 25);
```

У компонента `JTextArea` есть два метода, которые определяют поведение компонента в случае, когда вводимый пользователем текст достигает правой границы. Чтобы текст переносился на новую строку, вызовите метод `setLineWrap(`*`булево_значение`*`)`, указав аргумент `true`:

```
comments.setLineWrap(true);
```

Чтобы определить, каким образом текст переносится на следующую строку, воспользуйтесь методом `setWrapStyleWord(`*`булево_значение`*`)`. Если в качестве аргумента задать `true`, перенос текста будет выполняться по словам, в противном случае — по символам.

Если вы не воспользуетесь этими методами, текст не будет переноситься. Пользователь может продолжать вводить текст в той же строке, пока не нажмет клавишу <Enter>, дойдя до правого края области.

Компоненты `JTextArea` могут вести себя непредсказуемо: растягиваться по достижении пользователем нижней части области и не включать панели прокрутки вдоль правого или нижнего края. Чтобы получить более корректную реализацию, следует поместить текстовую область в контейнер, называемый *панелью прокрутки*.

Компоненты графического интерфейса пользователя часто имеют большие размеры, чем область, предназначенная для их отображения. Чтобы перемещаться от одной части компонента к другой, следует использовать вертикальные и горизонтальные панели прокрутки.

В Swing поддержка прокрутки включается путем добавления компонента на панель прокрутки, которая представляет собой контейнер класса `JScrollPane`.

Чтобы создать панель прокрутки, воспользуйтесь следующими конструкторами.

- `JScrollPane()`. Создает панель с горизонтальной и вертикальной полосами прокрутки, которые появляются в случае необходимости.
- `JScrollPane(`*целое_число*`, `*целое_число*`)`. Создает панель с вертикальной и горизонтальной полосами прокрутки заданного типа.
- `JScrollPane(`*компонент*`)`. Создает панель прокрутки, которая содержит указанный компонент пользовательского интерфейса.
- `JScrollPane(`*компонент*`, `*целое_число*`, `*целое_число*`)`. Создает панель с указанным компонентом, вертикальной и горизонтальной полосами прокрутки заданного типа.

Целочисленные аргументы в этих конструкторах определяют, каким образом полосы прокрутки используются на панели. В качестве этих аргументов используйте следующие константы:

- `JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED` **или** `JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED`;
- `JScrollPane.VERTICAL_SCROLLBAR_NEVER` **или** `JScrollPane.HORIZONTAL_SCROLLBAR_NEVER`;
- `JScrollPane.VERTICAL_SCROLLBAR_ALWAYS` **или** `JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS`.

Если созданная панель прокрутки не содержит компоненты, их можно добавить с помощью метода `add(`*компонент*`)`. После настройки панели прокрутки ее следует добавить в контейнер вместо компонента.

Ниже приведен предыдущий пример, переписанный с учетом добавления текстовой области на панель прокрутки.

```
FlowLayout flo = new FlowLayout();
setLayout(flo);
JTextArea comments = new JTextArea(8, 40);
comments.setLineWrap(true);
comments.setWrapStyleWord(true);
JScrollPane scroll = new JScrollPane(comments,
   JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
   JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
add(scroll);
```

На рис. 17.7 показан пример отображения этого компонента во фрейме.

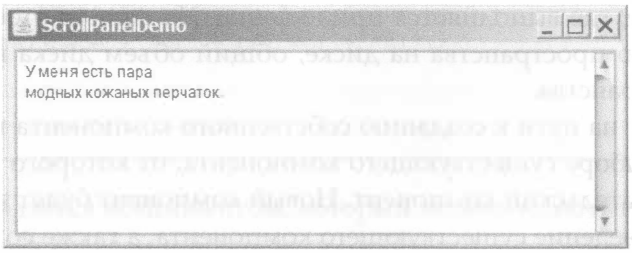


РИС. 17.7. Отображение текстовой области в контейнере, представляющем собой панель прокрутки

Панели

Последний компонент, который будет рассмотрен на этом занятии, — панель, создаваемая в Swing с помощью класса `JPanel`. Объекты `JPanel` являются простейшей разновидностью контейнера, доступной в пользовательском интерфейсе. Назначение объектов `JPanel` заключается в разделении области отображения на группы компонентов. В каждой группе можно использовать разные менеджеры компоновки.

Следующий код создает объект `JPanel` и назначает его менеджеру компоновки.

```
JPanel topRow = new JPanel();
FlowLayout flo = new FlowLayout();
topRow.setLayout(flo);
add(topRow);
```

Панели часто применяются для упорядочения компонентов интерфейса, как будет продемонстрировано на занятии 18.

Чтобы добавить компоненты на панель, следует вызвать соответствующий метод `add()`. Можно также назначить менеджер компоновки непосредственно панели, вызвав ее метод `setLayout()`.

Панели удобно использовать, когда нужно, например, вывести изображение, хранящееся в файле.

С помощью класса `JPanel` можно создавать собственные компоненты, добавляемые в другие классы. Это будет продемонстрировано в завершающем проекте данного занятия.

Создание пользовательского компонента

Преимущество объектно-ориентированного программирования заключается в возможности повторного использования классов в разных проектах. В следующем проекте будет создан специальный компонент панели, который можно повторно использовать в других программах на Java. Этот компонент, `FreeSpacePanel`, выводит сведения об объеме дискового пространства

на компьютере, где выполняется приложение. На панели отображается размер свободного пространства на диске, общий объем диска и процент свободного пространства.

Первый шаг на пути к созданию собственного компонента интерфейса заключается в выборе существующего компонента, от которого будет наследоваться пользовательский компонент. Новый компонент будет наследовать все атрибуты и поведение существующего компонента, а также содержать все то, что вы измените и добавите сами.

Компонент `FreeSpacePanel`, код которого приведен в листинге 17.3, является подклассом класса `JPanel`. Введите код листинга в новый пустой файл Java, включите его в пакет `com.java24hours` и сохраните файл под именем `FreeSpacePanel.java`.

ЛИСТИНГ 17.3. Код программы `FreeSpacePanel.java`

```
 1: package com.java24hours;
 2:
 3: import java.io.IOException;
 4: import java.nio.file.*;
 5: import javax.swing.*;
 6:
 7: public class FreeSpacePanel extends JPanel {
 8:    JLabel spaceLabel = new JLabel("Объем диска: ");
 9:    JLabel space = new JLabel();
10:
11:    public FreeSpacePanel() {
12:       super();
13:       add(spaceLabel);
14:       add(space);
15:       try {
16:          setValue();
17:       } catch (IOException ioe) {
18:          space.setText("Ошибка");
19:       }
20:    }
21:
22:    private final void setValue() throws IOException {
23:       // Получение сведений о хранилище текущего файла
24:       Path current = Paths.get("");
25:       FileStore store = Files.getFileStore(current);
26:       // Определение свободного места в хранилище
27:       long totalSpace = store.getTotalSpace();
28:       long freeSpace = store.getUsableSpace();
29:       // Вычисление объема в % (с двумя знаками после запятой)
30:       double percent = (double)freeSpace / (double)totalSpace * 100;
31:       percent = (int)(percent * 100) / (double)100;
32:       // Настройка текста метки
```

```
33:        space.setText(freeSpace + " свободно из " +
34:           totalSpace + " (" + percent + "%)");
35:
36:     }
37: }
```

Этот класс является компонентом, который можно включить в любой графический интерфейс, если понадобится отобразить информацию об объеме свободного места на диске. Как самостоятельное приложение он запускаться не может.

Метод `setValue()` в классе `FreeSpacePanel` устанавливает текст метки, сообщающей информацию о наличии места на диске. Данный метод объявлен как `final` (строка 22).

```
private final void setValue() {
   // ...
}
```

Благодаря наличию ключевого слова `final` предотвращается переопределение метода в подклассах. Это необходимое условие для того, чтобы компонент `FreeSpacePanel` был GUI-компонентом.

Панель создается в конструкторе (строки 11–20), при этом выполняются следующие действия.

- Строка 12. Метод `super()` вызывает конструктор класса `JPanel`, гарантируя его корректную настройку.
- Строка 13. На панель добавляется новая метка `spaceLabel`, созданная в строке 8 в качестве переменной экземпляра и содержащая текст `"Объем диска:"`. Метка добавляется с помощью метода `add(spaceLabel)`.
- Строка 14. На панель добавляется пустая метка, представляющая собой переменную экземпляра без текста, созданную в строке 9.
- Строка 16. Вызывается метод `setValue()`, в котором задается текст для пустой метки.

Вызов метода `setValue()` в строке 16 заключен в блок `try-catch`, который напоминает блок, применяемый при настройке внешнего вида панели. Дополнительные сведения об обработке ошибок будут приведены на занятии 18, сейчас же вам достаточно знать, что блок `try-catch` позволяет обработать одну или несколько ошибок, которые могут произойти в произвольной части программы на Java.

Подобные действия необходимы, поскольку вызов метода `setValue()` может сгенерировать исключение при получении доступа к файловой системе

компьютера. В данном случае это исключение класса `IOException`, которое представляет ошибки ввода/вывода.

В блоке `try` находится код, который может вызывать ошибки. Блок `catch` идентифицирует тип ошибки (в скобках) и содержит код, выполняемый в случае возникновения ошибки.

Если произошла ошибка, в строке 18 пустой метке присваивается текст `Ошибка`. В результате пользователь будет знать: что-то пошло не так.

Наш пользовательский компонент выполняет ряд действий, связанных с обработкой файлов. Подробнее об этом мы поговорим на занятии 20, здесь же будет дана лишь общая информация.

Чтобы определить размер свободного места на диске, нам понадобятся четыре класса из пакета `java.nio.file`, который используется для получения доступа к файловой системе компьютера.

Объект `Path` хранит местоположение файла или папки. Класс `Paths` содержит метод `get(`*`строка`*`)`, который преобразует строку в путь, соответствующий этой строке. В случае вызова с аргументом `""` (пустая строка) возвращается путь к текущей папке, в которой было запущено приложение Java.

```
Path current = Paths.get("");
```

Информация о дисковом хранилище представлена в Java объектом `File Store`, который соответствует пулу файлового хранилища. Чтобы получить доступ к файловому хранилищу, нужно задать путь к нему. Для этого используется метод `getFileStore(`*`путь`*`)` класса `Files`.

```
FileStore store = Files.getFileStore(current);
```

Получив объект пула, можно вызвать методы `getTotalSpace()` и `getUsableSpace()`, чтобы определить объем свободного места на текущем диске.

```
long totalSpace = store.getTotalSpace();
long freeSpace = store.getUsableSpace();
double percent = (double)freeSpace / (double)totalSpace * 100;
```

ПРЕДУПРЕЖДЕНИЕ

Выражение, которое вычисляет процентную долю, включает множество ссылок на тип `double`, что может показаться необычным. Обратите внимание на приведение переменных `freeSpace` и `totalSpace` к типу `double`. В результате предотвращается генерирование выражением целочисленного значения вместо числа с плавающей точкой. В Java тип выражения определяется на основании типов используемых в нем переменных. Значение типа `long`, деленное на значение типа `long` и умноженное на число 100 типа `integer`, даст результат типа `long`, а не требуемое в данном случае процентное значение.

Процентная доля, вычисляемая последней инструкцией, может иметь очень много цифр после десятичной точки, например `64.8675309`. Следующее выражение преобразует это значение в число, содержащее не более двух знаков после десятичной точки:

```
percent = (int)(percent * 100) / (double)100;
```

Первая часть выражения умножает процентную долю на 100, перемещая десятичную точку на два знака вправо, после чего приводит значение к целочисленному типу. В результате число `64.8675309` становится равно `6487`.

Вторая часть выражения делит значение на 100, перемещая десятичную точку обратно, на два знака влево, чтобы получить приблизительную процентную долю. В результате число `6487` преобразуется в `64.87`.

После того как процентная доля была ограничена двумя знаками после десятичной точки, текст метки настраивается таким образом, чтобы отобразить сведения о свободном и занятом месте на диске.

```
space.setText(freeSpace + " свободно из " + totalSpace + " (" +
              percent + "%)");
```

Панель невозможно увидеть до тех пор, пока она не будет добавлена в графический интерфейс пользователя. Чтобы увидеть панель `FreeSpacePanel`, создайте класс `FreeSpaceFrame` и включите его в пакет `com.java24hours`. Введите код из листинга 17.4 и сохраните полученный файл `FreeSpaceFrame.java`.

ЛИСТИНГ 17.4. Исходный код программы `FreeSpaceFrame.java`

```
 1: package com.java24hours;
 2:
 3: import java.awt.*;
 4: import javax.swing.*;
 5:
 6: public class FreeSpaceFrame extends JFrame {
 7:    public FreeSpaceFrame() {
 8:       super("Свободное пространство на диске");
 9:       setLookAndFeel();
10:       setSize(500, 120);
11:       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12:       FlowLayout flo = new FlowLayout();
13:       setLayout(flo);
14:       FreeSpacePanel freePanel = new FreeSpacePanel();
15:       add(freePanel);
16:       setVisible(true);
17:    }
18:
19:    private void setLookAndFeel() {
```

```
20:        try {
21:           UIManager.setLookAndFeel(
22:              "javax.swing.plaf.nimbus.NimbusLookAndFeel"
23:           );
24:        } catch (Exception exc) {
25:           // Игнорировать ошибки
26:        }
27:     }
28:
29:     public static void main(String[] arguments) {
30:        FreeSpaceFrame frame = new FreeSpaceFrame();
31:     }
32: }
```

Это приложение создает компонент `FreeSpacePanel` в строке 14 и добавляет панель в соответствующий фрейм в строке 15.

После запуска приложения отобразятся сведения о системном диске вашего компьютера (рис 17.8).

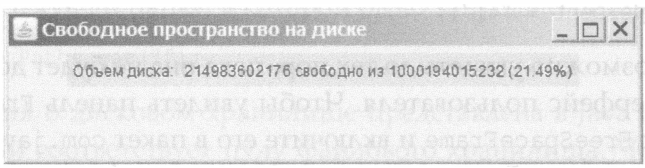


РИС. 17.8. Отображение пользовательского компонента, сообщающего сведения о диске

Резюме

Пользователям нравится использовать визуальную среду для запуска программ. Это усложняет создание программного обеспечения, но, по крайней мере, в Java имеется удобная библиотека Swing, включающая все необходимые классы для создания графического интерфейса пользователя.

Разработка графических приложений Java с помощью библиотеки Swing послужит хорошей практикой в объектно-ориентированном программировании. Каждый компонент, контейнер и менеджер компоновки представлен собственным классом. Многие из этих классов наследуют свое общее поведение из одного и того же суперкласса. Например, все компоненты пользовательского интерфейса, рассмотренные на этом занятии, являются подклассами суперкласса `javax.swing.JComponent`.

На следующем занятии мы поговорим о создании графического интерфейса с помощью новых менеджеров компоновки, которые позволяют указывать расположение компонентов в контейнере более сложным образом, чем стандартный менеджер `FlowLayout`.

Вопросы и ответы

В. **Как упорядочиваются компоненты, если контейнеру не назначен менеджер компоновки?**

О. В случае простого контейнера, такого как панель, компоненты по умолчанию упорядочиваются с помощью менеджера `FlowLayout`. Каждый компонент добавляется слева направо и сверху вниз (по достижении конца строки менеджер переходит к следующей строке). При размещении фреймов, окон и аплетов используется заданный по умолчанию стиль компоновки `BorderLayout`, который будет рассмотрен на следующем занятии.

В. **Почему так много классов графического интерфейса имеют префикс `J` в названии, например `JFrame` или `JLabel`?**

О. Эти классы являются частью пакета Swing, который был создан в результате второй попытки построения фреймворка классов графического интерфейса в библиотеке классов Java. Первая попытка — это пакет AWT (Abstract Windowing Toolkit). Тогда имена классов были проще, например `Frame` и `Label`.

Классы AWT относятся к пакету `java.awt` и связанным с ним пакетам, тогда как классы Swing относятся к пакету `javax.swing` и родственным пакетам. Потенциально имена классов AWT и Swing могли совпадать, поэтому благодаря префиксу `J` в именах предотвращается возможная путаница в использовании классов.

Классы Swing входят в набор JFC (Java Foundation Classes).

Коллоквиум

Проверьте знание материала, рассмотренного на этом занятии, ответив на следующие вопросы.

Контрольные вопросы

1. Какой пользовательский компонент используется в качестве контейнера, предназначенного для хранения других компонентов?

   А. `TupperWare`.

   Б. `JPanel`.

   В. `Choice`.

2. Что из нижеперечисленного нужно сделать в первую очередь в контейнере?

   А. Установить менеджер компоновки.

Б. Добавить компоненты.

В. Не имеет значения.

3. Какой метод определяет способ упорядочения компонентов в контейнере?

   А. `setLayout()`.

   Б. `setLayoutManager()`.

   В. `setVisible()`.

Ответы

1. **Б.** `JPanel`. Можно добавить компоненты на панель, а затем включить панель в другой контейнер, такой как фрейм.
2. **А.** Сначала нужно задать менеджер компоновки, чтобы потом можно было корректно добавлять компоненты.
3. **А.** Метод `setLayout()` имеет единственный аргумент — объект менеджера компоновки, который определяет, как должны располагаться компоненты.

Упражнения

Чтобы закрепить знания в области дизайна графического интерфейса пользователя, выполните следующие упражнения.

- Измените приложение `SalutonFrame` так, чтобы оно отображало сообщение `"Saluton Mondo!"` в основной области фрейма, а не в строке заголовка.
- Усовершенствуйте компонент `FreeSpacePanel` таким образом, чтобы отображать пробелы между разрядами чисел в строке данных о диске. Для этого можно использовать объект `StringBuilder`, метод `charAt(`*`целое_число`*`)` класса `String` и цикл `for` для итерирования по строке.

ЗАНЯТИЕ 18
Компоновка элементов интерфейса

На этом занятии вы узнаете...

- как создать менеджер компоновки;
- как назначить менеджер компоновки контейнеру;
- как применять панели для группировки компонентов в интерфейсе;
- как работать с необычными компоновками;
- как создать прототип для приложения Java.

В процессе разработки графических интерфейсов для приложений Java следует учитывать возможность перемещения компонентов. Как только размеры контейнера изменятся, например вследствие масштабирования фрейма пользователем, компоненты перераспределятся в пределах контейнера автоматически, чтобы вписаться в новое пространство.

Благодаря подобной адаптивности интерфейса можно легко настраивать отображение компонентов в разных операционных системах. Например, простая кнопка может выглядеть по-разному в Windows, Linux или в Mac OS.

Компоненты интерфейса упорядочиваются на экране с помощью так называемых *менеджеров компоновки*, которые определяют порядок отображения компонентов в контейнере. Каждый контейнер в интерфейсе может иметь свой собственный менеджер компоновки.

Использование менеджеров компоновки

В Java способ расположения компонентов в контейнере зависит от размеров этих компонентов и высоты/ширины контейнера. На размещение кнопок, текстовых полей и других компонентов могут оказывать влияние следующие факторы:

- размер контейнера;
- размеры других компонентов и контейнеров;
- применяемый менеджер компоновки.

Доступно несколько менеджеров компоновки, которые влияют на порядок отображения компонентов. По умолчанию в качестве менеджера компоновки для панелей используется класс `FlowLayout` из пакета `java.awt` (он применялся на прошлом занятии). В этом случае компоненты располагаются слева направо и сверху вниз (при достижении правой границы компонент переносится на следующую строку).

Следующий фрагмент кода предназначен для фрейма, поскольку при добавлении компонентов применяется компоновка с перетеканием (`FlowLayout`).

```
FlowLayout topLayout = new FlowLayout();
setLayout(topLayout);
```

Можно также настроить менеджер компоновки для работы с конкретным контейнером, например с объектом `JPanel`. В таком случае нужно использовать метод `setLayout()` объекта этого контейнера.

В качестве первого проекта занятия мы создадим приложение `Crisis` с графическим интерфейсом, включающим пять кнопок. Создайте новый класс `Crisis` и включите его в пакет `com.java24hours`. Введите код из листинга 18.1 и сохраните полученный файл `Crisis.java`.

ЛИСТИНГ 18.1. Исходный код программы `Crisis.java`

```
 1: package com.java24hours;
 2:
 3: import java.awt.*;
 4: import javax.swing.*;
 5:
 6: public class Crisis extends JFrame {
 7:    JButton panicButton;
 8:    JButton dontPanicButton;
 9:    JButton blameButton;
10:    JButton mediaButton;
11:    JButton saveButton;
12:
13:    public Crisis() {
14:       super("Crisis");
15:       setLookAndFeel();
16:       setSize(348, 128);
17:       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
18:       FlowLayout flo = new FlowLayout();
19:       setLayout(flo);
20:       panicButton = new JButton("Тревога");
21:       dontPanicButton = new JButton("Не паникуйте");
22:       blameButton = new JButton("Виноваты другие");
23:       mediaButton = new JButton("Известите СМИ");
24:       saveButton = new JButton("Спасите себя");
```

```
25:        add(panicButton);
26:        add(dontPanicButton);
27:        add(blameButton);
28:        add(mediaButton);
29:        add(saveButton);
30:        setVisible(true);
31:     }
32:
33:     private void setLookAndFeel() {
34:        try {
35:           UIManager.setLookAndFeel(
36:              "com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel"
37:           );
38:        } catch (Exception exc) {
39:           // Игнорировать ошибки
40:        }
41:     }
42:
43:     public static void main(String[] arguments) {
44:        Crisis frame = new Crisis();
45:     }
46: }
```

На рис. 18.1 показан результат выполнения приложения.

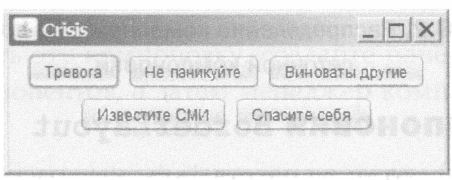


РИС. 18.1. Расположение компонентов в соответствии с компоновкой `FlowLayout`

Класс `FlowLayout` использует размеры контейнера как ориентир для размещения компонентов. Измените размеры окна приложения, и вы увидите, как компоненты мгновенно перегруппируются. Сделайте окно в два раза шире, и все компоненты `JButton` выстроятся в линию. Сузьте окно, и все компоненты отобразятся в виде вертикальной колонки.

Менеджер компоновки `GridLayout`

Класс `GridLayout` из пакета `java.awt` упорядочивает все компоненты контейнера таким образом, чтобы они занимали определенное количество строк и столбцов. При этом каждому компоненту выделяется одинаковое пространство. Если, к примеру, задать сетку размером три столбца в ширину и

три строки в высоту, то контейнер будет разделен на девять областей одинакового размера.

Менеджер компоновки `GridLayout` помещает все компоненты по мере их добавления в выделенные ячейки сетки. Компоненты добавляются в направлении слева направо до заполнения строки, после чего заполняется крайний левый столбец следующей строки.

Следующие инструкции создают контейнер и настраивают его для использования сеточной компоновки (`GridLayout`), ширина которой составляет три столбца, а высота — две строки.

```
GridLayout grid = new GridLayout(2, 3);
setLayout(grid);
```

На рис. 18.2 показано, как выглядит приложение `Crisis` после выбора сеточной компоновки.

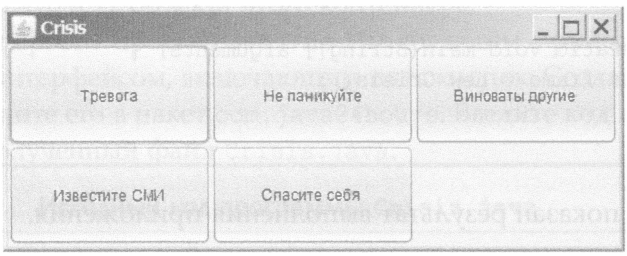


РИС. 18.2. Распределение компонентов с помощью сеточной компоновки

Менеджер компоновки `BorderLayout`

Класс `BorderLayout`, тоже из пакета `java.awt`, распределяет компоненты по указанным позициям в контейнере, которые идентифицируются одним из пяти направлений: север, юг, восток, запад и центр.

Менеджер компоновки `BorderLayout` распределяет компоненты по пяти областям: четыре в соответствии со сторонами света и пятая область — в центре. Если добавить компонент в соответствии с этой компоновкой, то метод `add()` будет включать второй аргумент, задающий местоположение компонента. В качестве этого аргумента используется одна из пяти констант класса `BorderLayout`: `NORTH`, `SOUTH`, `EAST`, `WEST` или `CENTER`.

Подобно классу `GridLayout`, класс `BorderLayout` выделяет компонентам все доступное пространство. Компоненту, находящемуся в центре, отводится все пространство, не занимаемое четырьмя граничными компонентами. Поэтому центральный компонент обычно самый большой.

Следующие инструкции создают контейнер, который использует граничную компоновку (`BorderLayout`).

```
BorderLayout crisisLayout = new BorderLayout();
setLayout(crisisLayout);
add(panicButton, BorderLayout.NORTH);
add(dontPanicButton, BorderLayout.SOUTH);
add(blameButton, BorderLayout.EAST);
add(mediaButton, BorderLayout.WEST);
add(saveButton, BorderLayout.CENTER);
```

На рис. 18.3 показан результат применения этого кода в приложении Crisis.

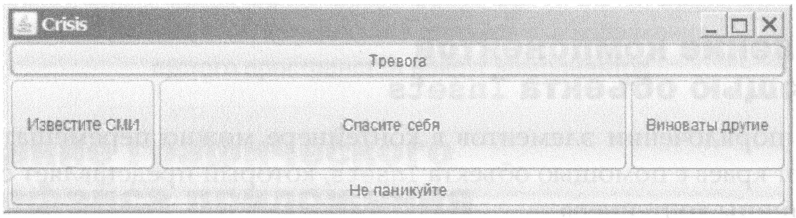


РИС. 18.3. Размещение компонентов с помощью граничной компоновки

Менеджер компоновки BoxLayout

Еще один удобный менеджер компоновки, BoxLayout из пакета javax.swing, выстраивает компоненты в одной строке, расположенной по горизонтали или по вертикали.

Чтобы использовать эту компоновку, создайте панель, предназначенную для размещения компонентов, а затем менеджер компоновки с двумя аргументами:

- компонент, предназначенный для использования блочной компоновки;
- константа BoxLayout.Y_AXIS для выравнивания по вертикали или BoxLayout.X_AXIS для выравнивания по горизонтали.

Ниже приведен код, применяемый для размещения компонентов приложения Crisis.

```
JPanel pane = new JPanel();
BoxLayout box = new BoxLayout(pane, BoxLayout.Y_AXIS);
pane.setLayout(box);
pane.add(panicButton);
pane.add(dontPanicButton);
pane.add(blameButton);
pane.add(mediaButton);
pane.add(saveButton);
add(pane);
```

На рис. 18.4 представлен результат выполнения этого кода.

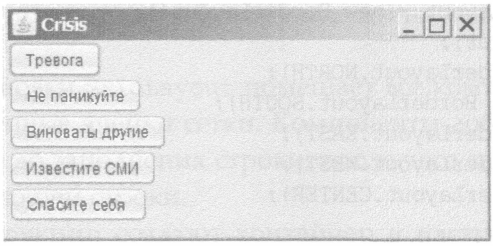


РИС. 18.4. Размещение компонентов с применением блочной компоновки

Разделение компонентов с помощью объекта `Insets`

При упорядочении элементов в контейнере можно перемещать их подальше от краев с помощью объекта `Insets`, который представляет собой область границы контейнера.

Класс `Insets`, входящий в пакет `java.awt`, содержит конструктор с четырьмя аргументами, определяющими расстояние до верхней, левой, нижней и правой границ контейнера. Аргументы конструктора задаются в пикселях, как и при указании размеров фрейма.

Следующая инструкция создает объект `Insets`:

```
Insets around = new Insets(10, 6, 10, 3);
```

Объект `around` формирует границу контейнера, которая на 10 пикселей отстоит от верхнего края, на 6 пикселей — от левого края, на 10 пикселей — от нижнего края и на 3 пикселя — от правого.

Чтобы использовать объект `Insets` в контейнере, нужно переопределить метод `getInsets()` контейнера, который не имеет аргументов и возвращает объект `Insets`.

```
public Insets getInsets() {
   Insets squeeze = new Insets(50, 15, 10, 15);
   return squeeze;
}
```

На рис. 18.5 показано, как этот код изменит интерфейс, скомпонованный с помощью менеджера компоновки `FlowLayout` (см. рис. 18.1).

ПРИМЕЧАНИЕ

Контейнер `JFrame` имеет встроенный отступ, резервирующий пространство для заголовка фрейма. Если переопределить метод `getInsets()` и установить собственные значения, низкое значение отступа приведет к тому, что компоненты в контейнере будут отображаться непосредственно под строкой заголовка.

Контейнер, показанный на рис. 18.5, имеет пустую границу, которая на 15 пикселей отстоит от левого края, на 10 пикселей — от нижнего, на 15 пикселей — от правого и на 50 пикселей — от верхнего.

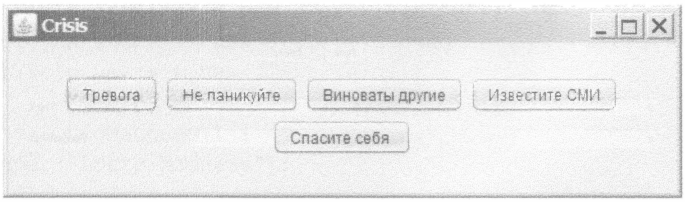


РИС. 18.5. Использование отступов для добавления пустого пространства вокруг компонентов

Создание графического интерфейса приложения

Менеджеры компоновки, которые рассматривались до сих пор, применялись ко всему фрейму. При этом все компоненты располагались в соответствии с одними и теми же правилами. Подобная методика удобна для отдельных программ, но, разрабатывая графические интерфейсы с помощью Swing, вы часто будете сталкиваться с тем, что ни один из менеджеров компоновки не подходит.

Один из способов решения этой проблемы заключается в использовании группы объектов `JPanel` в качестве контейнеров, предназначенных для хранения разных секций графического интерфейса. Для каждой из секций можно задать свои правила компоновки, которые будут применяться к панелям `JPanel`. После того как панели будут содержать все необходимые компоненты, можно добавить их непосредственно во фрейм.

А сейчас мы разработаем полноценный интерфейс для программы, которая будет написана на занятии 19. Эта программа оценивает шанс пользователя выиграть один из призов в лотерею. Шанс вычисляется путем случайного перебора лотерейных розыгрышей снова и снова до тех пор, пока комбинация пользователя не выиграет. На рис. 18.6 показан графический интерфейс, разработанный для этого приложения.

Создайте новый класс `LottoMadness` и включите его в пакет `com.java24hours`. В редакторе исходного кода введите код из листинга 18.2 и сохраните полученный файл `LottoMadness.java`.

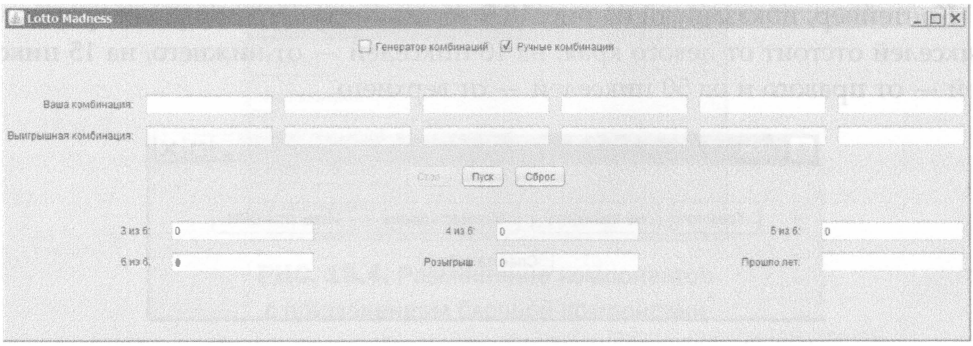


РИС. 18.6. Графический интерфейс приложения `LottoMadness`

ЛИСТИНГ 18.2. Исходный код программы `LottoMadness.java`

```
 1: package com.java24hours;
 2:
 3: import java.awt.*;
 4: import javax.swing.*;
 5:
 6: public class LottoMadness extends JFrame {
 7:
 8:    // Настройка строки 1
 9:    JPanel row1 = new JPanel();
10:    ButtonGroup option = new ButtonGroup();
11:    JCheckBox quickpick = new JCheckBox("Генератор комбинаций",
12:                                        false);
13:    JCheckBox personal = new JCheckBox("Ручные комбинации", true);
14:    // Настройка строки 2
15:    JPanel row2 = new JPanel();
16:    JLabel numbersLabel = new JLabel("Ваша комбинация: ",
17:                                     JLabel.RIGHT);
18:    JTextField[] numbers = new JTextField[6];
19:    JLabel winnersLabel = new JLabel("Выигрышная комбинация: ",
20:                                     JLabel.RIGHT);
21:    JTextField[] winners = new JTextField[6];
22:    // Настройка строки 3
23:    JPanel row3 = new JPanel();
24:    JButton stop = new JButton("Стоп");
25:    JButton play = new JButton("Пуск");
26:    JButton reset = new JButton("Сброс");
27:    // Настройка строки 4
28:    JPanel row4 = new JPanel();
29:    JLabel got3Label = new JLabel("3 из 6: ", JLabel.RIGHT);
30:    JTextField got3 = new JTextField("0");
31:    JLabel got4Label = new JLabel("4 из 6: ", JLabel.RIGHT);
32:    JTextField got4 = new JTextField("0");
33:    JLabel got5Label = new JLabel("5 из 6: ", JLabel.RIGHT);
```

```
34:    JTextField got5 = new JTextField("0");
35:    JLabel got6Label = new JLabel("6 из 6: ", JLabel.RIGHT);
36:    JTextField got6 = new JTextField("0", 10);
37:    JLabel drawingsLabel = new JLabel("Розыгрыш: ", JLabel.RIGHT);
38:    JTextField drawings = new JTextField("0");
39:    JLabel yearsLabel = new JLabel("Прошло лет: ", JLabel.RIGHT);
40:    JTextField years = new JTextField();
41:
42:    public LottoMadness() {
43:       super("Lotto Madness");
44:
45:       setSize(550, 400);
46:       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
47:       GridLayout layout = new GridLayout(5, 1, 10, 10);
48:       setLayout(layout);
49:
50:       FlowLayout layout1 = new FlowLayout(FlowLayout.CENTER,
51:                                           10, 10);
52:       option.add(quickpick);
53:       option.add(personal);
54:       row1.setLayout(layout1);
55:       row1.add(quickpick);
56:       row1.add(personal);
57:       add(row1);
58:
59:       GridLayout layout2 = new GridLayout(2, 7, 10, 10);
60:       row2.setLayout(layout2);
61:       row2.add(numbersLabel);
62:       for (int i = 0; i < 6; i++) {
63:          numbers[i] = new JTextField();
64:          row2.add(numbers[i]);
65:       }
66:       row2.add(winnersLabel);
67:       for (int i = 0; i < 6; i++) {
68:          winners[i] = new JTextField();
69:          winners[i].setEditable(false);
70:          row2.add(winners[i]);
71:       }
72:       add(row2);
73:
74:       FlowLayout layout3 = new FlowLayout(FlowLayout.CENTER,
75:                                           10, 10);
76:       row3.setLayout(layout3);
77:       stop.setEnabled(false);
78:       row3.add(stop);
79:       row3.add(play);
80:       row3.add(reset);
81:       add(row3);
82:
```

```
 83:        GridLayout layout4 = new GridLayout(2, 3, 20, 10);
 84:        row4.setLayout(layout4);
 85:        row4.add(got3Label);
 86:        got3.setEditable(false);
 87:        row4.add(got3);
 88:        row4.add(got4Label);
 89:        got4.setEditable(false);
 90:        row4.add(got4);
 91:        row4.add(got5Label);
 92:        got5.setEditable(false);
 93:        row4.add(got5);
 94:        row4.add(got6Label);
 95:        got6.setEditable(false);
 96:        row4.add(got6);
 97:        row4.add(drawingsLabel);
 98:        drawings.setEditable(false);
 99:        row4.add(drawings);
100:        row4.add(yearsLabel);
101:        years.setEditable(false);
102:        row4.add(years);
103:        add(row4);
104:
105:        setVisible(true);
106:     }
107:
108:     private static void setLookAndFeel() {
109:        try {
110:           UIManager.setLookAndFeel(
111:              "com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel"
112:           );
113:        } catch (Exception exc) {
114:          // Игнорировать ошибки
115:        }
116:     }
117:
118:     public static void main(String[] arguments) {
119:        LottoMadness.setLookAndFeel();
120:        LottoMadness frame = new LottoMadness();
121:     }
122: }
```

Несмотря на то что программа всего лишь создает графический интерфейс приложения, можно запустить ее на выполнение. Это позволит убедиться в том, что графический интерфейс корректно сформирован и собирает требуемую информацию.

В приложении используется несколько менеджеров компоновки. Чтобы получить более четкое представление о структуре интерфейса, обратитесь

к рис. 18.7. Интерфейс разделен на четыре горизонтальные строки (панели), выделенные на рисунке черными линиями. Каждая строка является объектом `JPanel`. Интерфейсом в целом управляет менеджер компоновки `GridLayout`, создающий сетку из четырех строк и одного столбца.

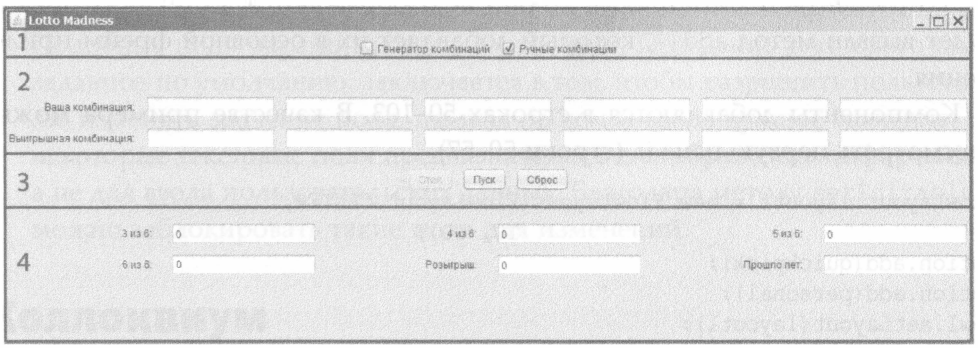


РИС. 18.7. Разделение окна приложения `LottoMadness` на панели

В панелях используются разные менеджеры компоновки, что позволяет точно настроить расположение компонентов. Для первой и третьей панелей задан менеджер `FlowLayout`. Строки 50 и 51 программы демонстрируют, как создаются соответствующие объекты.

```
FlowLayout layout1 = new FlowLayout(FlowLayout.CENTER,
                                    10, 10);
```

Конструктору `FlowLayout()` передаются три аргумента. Первый аргумент, `FlowLayout.CENTER`, указывает на то, что компоненты следует выровнять по центру в содержащем их контейнере. В качестве такого контейнера используется горизонтальная панель `JPanel`. Последние два аргумента определяют расстояние по ширине и высоте, на которое каждый компонент должен быть отделен от других компонентов. Расстояние в 10 пикселей по ширине и высоте создает дополнительное небольшое пространство между компонентами.

Вторая панель имеет вид сетки, высота которой составляет две строки, а ширина — семь столбцов. Конструктор `GridLayout()` также указывает на то, что компоненты должны отделяться от других компонентов на 10 пикселей по ширине и высоте. Эта сетка настраивается в строках 59 и 60.

```
GridLayout layout2 = new GridLayout(2, 7, 10, 10);
row2.setLayout(layout2);
```

Аналогичным образом в четвертой панели используется компоновка `GridLayout` для распределения компонентов в виде сетки, высота которой составляет две строки, а ширина — три столбца.

В приложении LottoMadness используется большая группа компонентов, настраиваемых в строках 9–40. Инструкции сгруппированы по панелям интерфейса. Сначала создается объект JPanel для панели, после чего настраивается каждый компонент, добавляемый на эту панель. Программа создает все компоненты и контейнеры, но они не отображаются до тех пор, пока не будет вызван метод add(), который добавляет их в основной фрейм приложения.

Компоненты добавляются в строках 50–103. В качестве примера можно рассмотреть первую панель (строки 50–57).

```
FlowLayout layout1 = new FlowLayout(FlowLayout.CENTER,
                                    10, 10);
option.add(quickpick);
option.add(personal);
row1.setLayout(layout1);
row1.add(quickpick);
row1.add(personal);
add(row1);
```

Созданный объект менеджера компоновки подключается к объекту JPanel (панель row1) с помощью метода setLayout(). Далее на панель добавляются компоненты с помощью метода add(), а в конце весь объект row1 добавляется во фрейм. При этом используется собственный метод фрейма add().

Внешний вид интерфейса LottoMadness настраивается не так, как в предыдущих приложениях Swing. Здесь метод setLookAndFeel() создается как метод класса (обратите внимание на использование ключевого слова static в строке 108) и вызывается в методе main() (строка 119).

В предыдущих приложениях метод setLookAndFeel() был методом экземпляра, который вызвался в конструкторе объекта. В приложении LottoMadness этот способ не применяется, поскольку внешний вид должен быть выбран до того, как будут созданы какие-либо переменные экземпляра.

Резюме

Если вы впервые сталкиваетесь с разработкой графического интерфейса для программы на Java, то вряд ли оцените все преимущества менеджеров компоновки. Просто поработайте с ними, и вы поймете, насколько они удобны для гибкой настройки расположения компонентов.

На следующем занятии вы больше узнаете о том, как функционирует графический интерфейс пользователя, и увидите, как применяется программа LottoMadness.

Вопросы и ответы

В. Почему одни текстовые поля в приложении `LottoMadness` окрашены в серый цвет, а другие — в белый?

О. Метод `setEditable()` позволяет запретить изменение значений полей. Подобные поля окрашиваются в серый цвет. Поведение текстового поля, заданное по умолчанию, заключается в том, чтобы разрешить пользователю изменять значение поля, щелкая на нем и вводя нужное значение. Но некоторые текстовые поля предназначены для отображения информации, а не для ввода пользовательских данных. Благодаря методу `setEditable()` можно заблокировать такие поля для изменений.

Коллоквиум

Проверьте знание материала, рассмотренного на этом занятии, ответив на следующие вопросы.

Контрольные вопросы

1. Какой контейнер чаще всего применяется при разделении интерфейса на разные менеджеры компоновки?
   - **А.** `JWindow`.
   - **Б.** `JPanel`.
   - **В.** `Container`.
2. Какой менеджер компоновки по умолчанию применяется для панели?
   - **А.** `FlowLayout`.
   - **Б.** `GridLayout`.
   - **В.** Не задан.
3. Откуда произошло название класса `BorderLayout`?
   - **А.** От границы (border) каждого компонента.
   - **Б.** От способа распределения компонентов вдоль границ контейнера.
   - **В.** Так захотелось разработчикам Java.

Ответы

1. **Б.** `JPanel`, простейший из контейнеров.
2. **А.** Панели используют компоновку с перетеканием (`FlowLayout`), тогда как для фреймов и окон по умолчанию применяется граничная компоновка (`BorderLayout`).

3. **Б.** При добавлении компонентов в контейнер с компоновкой `BorderLayout` следует указать их расположение вдоль границ с помощью таких констант, как `BorderLayout.WEST` и `BorderLayout.EAST`.

Упражнения

Выполните следующие упражнения, чтобы закрепить изученный материал.

- Создайте измененную версию приложения `Crisis`, в которой кнопки `panic` и `dontPanic` упорядочены с помощью одного менеджера компоновки, а оставшиеся три кнопки — с помощью другого менеджера.
- Создайте копию файла `LottoMadness.java`, переименовав ее в `New Madness.java`. Внесите изменения в программу, чтобы вместо верхних двух флажков использовался раскрывающийся список, а кнопки пуска, останова и сброса были реализованы в виде флажков.

ЗАНЯТИЕ 19
Получение данных от пользователя

На этом занятии вы узнаете...

- как создавать программы, которые реагируют на события;
- как заставить компонент генерировать события;
- как обрабатывать события в программе;
- как хранить информацию в интерфейсе;
- как преобразовывать значения, сохраненные в текстовых полях.

Графический интерфейс, который мы разрабатывали на двух предыдущих занятиях, представляет интерес сам по себе. Пользователи могут щелкать на кнопках, вводить текст в поля и изменять размеры окон. Но рано или поздно даже самый неприхотливый пользователь захочет большего. Графический интерфейс программы должен выполнять определенные действия в ответ на щелчок мыши или ввод данных с клавиатуры.

Все это становится возможным в том случае, когда приложение Java способно реагировать на пользовательские события. Данный процесс называется *обработкой событий* и будет рассмотрен на этом занятии.

Как научить программу "слушать"

Пользовательское событие в Java происходит тогда, когда пользователь выполняет действие с помощью мыши, клавиатуры или другого устройства ввода.

Прежде чем вы сможете обрабатывать события, следует научить объект слушать. Для реагирования на пользовательские события нужно использовать один или несколько интерфейсов `EventListener`. Интерфейс — это элемент объектно-ориентированного программирования на Java, позволяющий классу наследовать требуемое поведение. Это своего рода контракт, который гарантирует включение в класс определенных методов.

Интерфейс EventListener содержит методы, которые позволяют получать данные определенного типа, вводимые пользователем.

Для подключения интерфейса EventListener требуется соблюдение двух условий. Во-первых, поскольку классы-слушатели входят в пакет java.awt.event, следует обеспечить доступ к этому пакету с помощью следующей инструкции:

```
import java.awt.event.*;
```

Во-вторых, класс должен объявляться с ключевым словом implements, реализуя поддержку одного или нескольких интерфейсов-слушателей. Следующая инструкция создает класс, который реализует интерфейс ActionListener, отвечающий за щелчки на кнопках и пунктах меню:

```
public class Graph implements ActionListener {
```

Интерфейсы EventListener позволяют компонентам графического интерфейса генерировать пользовательские события. В случае отсутствия нужного интерфейса компонент не сможет ничего сделать, чтобы быть услышанным другими частями программы. Программа должна включать интерфейс-слушатель для каждого типа компонента, от которого она хочет получать события. Чтобы программа реагировала на щелчок мыши на кнопке или на нажатие клавиши <Enter> в текстовом поле, следует включить интерфейс ActionListener. Чтобы программа могла реагировать на выбор вариантов из списка или установку флажков, потребуется интерфейс ItemListener.

Если требуется несколько интерфейсов в одном и том же классе, поместите их названия, разделенные запятыми, после ключевого слова implements, как показано в следующем примере.

```
public class Graph3D implements ActionListener, MouseListener {
   // ...
}
```

Настройка компонентов для прослушивания

После того как реализован интерфейс для определенного компонента, необходимо заставить этот компонент генерировать пользовательские события. Интерфейс ActionListener позволяет реагировать на события действия, такие как щелчок на кнопке или нажатие клавиши <Enter>.

Чтобы объект JButton начал генерировать события, воспользуйтесь методом addActionListener(), как показано ниже.

```
JButton fireTorpedos = new JButton("Пуск торпед");
fireTorpedos.addActionListener(this);
```

Этот код создает кнопку `fireTorpedos` и вызывает метод кнопки `addActionListener()`. Ключевое слово `this`, которое используется в качестве аргумента метода `addActionListener()`, указывает на то, что текущий объект обрабатывает пользовательские события.

ПРИМЕЧАНИЕ

Ключевое слово `this` указывает на объект, в котором оно появляется. Если, например, создать класс `LottoMadness` и использовать ключевое слово `this` в одном из его методов, оно будет ссылаться на объект `LottoMadness`, для которого был вызван данный метод.

Обработка пользовательских событий

Если пользовательское событие генерируется компонентом, у которого есть слушатель, автоматически вызывается метод-обработчик. Этот метод должен содержаться в классе, указанном при подключении слушателя к компоненту.

Каждый слушатель включает разные методы, которые вызываются для реагирования на соответствующие события. Интерфейс `ActionListener` отсылает события методу `actionPerformed()`, который определяется так.

```
public void actionPerformed(ActionEvent event) {
   // Тело метода
}
```

Все события, полученные программой, передаются этому методу. Если только один компонент генерирует события действий, в метод включаются инструкции обработки полученного события. Если же события могут генерироваться несколькими компонентами, нужно проверить объект, переданный методу.

Аргументом метода `actionPerformed()` является объект `ActionEvent`. Пользовательские события, которые могут генерироваться в программе, представляются несколькими классами. Эти классы содержат методы, позволяющие определить компонент, который вызвал событие. Если объект `ActionEvent` в методе `actionPerformed()` называется `event`, то можно идентифицировать компонент с помощью следующей инструкции:

```
String cmd = event.getActionCommand();
```

Метод `getActionCommand()` возвращает строку. Если компонент является кнопкой, строка будет надписью на этой кнопке. Если же компонент

является текстовым полем, строка будет текстом, введенным в поле. Метод `getSource()` возвращает сам объект, сгенерировавший событие.

Можно использовать следующий метод `actionPerformed()` для получения событий от трех компонентов: кнопки `start` (объект `JButton`), текстового поля `speed` (объект `JTextField`) и еще одного текстового поля `viscosity`.

```
public void actionPerformed(ActionEvent event) {
   Object source = event.getSource();
   if (source == speed) {
      // Событие сгенерировано полем speed
   } else if (source == viscosity) {
      // Событие сгенерировано полем viscosity
   } else {
      // Событие сгенерировано кнопкой start
   }
}
```

Вызывайте метод `getSource()` для всех пользовательских событий, чтобы идентифицировать конкретный объект, который сгенерировал событие.

События флажка и поля со списком

Поля со списком и флажки требуют наличия интерфейса `ItemListener`. Вызовите метод компонента `addItemListener()`, чтобы заставить его генерировать события. Следующие инструкции создают флажок `superSize`, который генерирует пользовательские события в случае его выбора или отмены выбора.

```
JCheckBox superSize = new JCheckBox("Super Size", true);
superSize.addItemListener(this);
```

Эти события перехватываются методом `itemStateChanged()`, которому в качестве аргумента передается объект `ItemEvent`. Если требуется узнать, какой объект сгенерировал событие, следует вызвать метод `getItem()` объекта события.

Чтобы определить, был ли флажок выбран или сброшен, сравните значение, возвращаемое методом `getStateChange()`, с константой `ItemEvent.SELECTED` или `ItemEvent.DESELECTED`.

```
public void itemStateChanged(ItemEvent item) {
   int status = item.getStateChange();
   if (status == ItemEvent.SELECTED) {
      // Элемент выбран
   }
}
```

Чтобы определить, какое значение было выбрано в объекте JComboBox, используйте метод getItem() и преобразуйте полученное значение в строку.

```
Object which = item.getItem();
String answer = which.toString();
```

События клавиатуры

Если программа должна реагировать сразу же после нажатия клавиши, необходимо обрабатывать события клавиатуры и использовать интерфейс KeyListener.

В первую очередь необходимо зарегистрировать компонент, получающий код нажатой клавиши, вызвав его метод addKeyListener(). Аргументом метода должен быть объект, реализующий интерфейс KeyListener. Если это текущий класс, используйте аргумент this.

Объект, который обрабатывает события клавиатуры, должен реализовывать следующие три метода.

- keyPressed(*СобытиеКлавиатуры*). Вызывается в момент нажатия клавиши.
- keyReleased(*СобытиеКлавиатуры*). Вызывается в момент отпускания клавиши.
- keyTyped(*СобытиеКлавиатуры*). Вызывается после того, как клавиша была нажата и отпущена.

Эти методы не возвращают никаких значений (void), и аргументом каждого из них является объект KeyEvent, который содержит методы, позволяющие получить дополнительную информацию о событии. Например, с помощью метода getKeyChar() можно узнать о том, какая клавиша была нажата. Он возвращает значение типа char, которое может содержать только буквы, цифры и знаки пунктуации.

Если нужно отслеживать нажатия произвольных клавиш, включая клавиши <Enter>, <Home>, <Page Up> и <Page Down>, вызовите метод getKeyCode(), который возвращает целочисленный код клавиши. Далее можно вызвать метод getKeyText(), указав полученный код в качестве аргумента. Это позволит получить объект String, содержащий название клавиши (например, Home, F1 и т.п.).

В листинге 19.1 содержится код приложения Java, которое идентифицирует нажатые клавиши с помощью метода getKeyChar(). Приложение реализует интерфейс KeyListener, поэтому содержит методы keyTyped(), keyPressed() и keyReleased(). Но единственный метод, который выполняет реальные действия, — это keyTyped() (строки 26–29). Создайте новый класс KeyViewer, включите его в пакет com.java24hours, введите код из листинга 19.1 и сохраните полученный файл KeyViewer.java.

ЛИСТИНГ 19.1. Исходный код программы `KeyViewer.java`

```
 1: package com.java24hours;
 2:
 3: import javax.swing.*;
 4: import java.awt.event.*;
 5: import java.awt.*;
 6:
 7: public class KeyViewer extends JFrame implements KeyListener {
 8:    JTextField keyText = new JTextField(80);
 9:    JLabel keyLabel = new JLabel("Нажмите любую клавишу
10:                                  в текстовом поле.");
11:
12:    public KeyViewer() {
13:       super("KeyViewer");
14:       setLookAndFeel();
15:       setSize(350, 100);
16:       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
17:       keyText.addKeyListener(this);
18:       BorderLayout bord = new BorderLayout();
19:       setLayout(bord);
20:       add(keyLabel, BorderLayout.NORTH);
21:       add(keyText, BorderLayout.CENTER);
22:       setVisible(true);
23:    }
24:
25:    @Override
26:    public void keyTyped(KeyEvent input) {
27:       char key = input.getKeyChar();
28:       keyLabel.setText("Вы нажали " + key);
29:    }
30:
31:    @Override
32:    public void keyPressed(KeyEvent txt) {
33:       // Ничего не делать
34:    }
35:
36:    @Override
37:    public void keyReleased(KeyEvent txt) {
38:       // Ничего не делать
39:    }
40:
41:    private void setLookAndFeel() {
42:       try {
43:          UIManager.setLookAndFeel(
44:             "com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel"
45:          );
46:       } catch (Exception exc) {
47:          // Игнорировать ошибку
```

```
48:        }
49:     }
50:
51:     public static void main(String[] arguments) {
52:        KeyViewer frame = new KeyViewer();
53:     }
54: }
```

После запуска приложения появится окно, показанное на рис. 19.1. Приложение реализует три метода интерфейса `KeyListener` (строки 25–39), два из которых являются пустыми. Они не нужны самому приложению, но должны формально включаться в рамках контракта по реализации интерфейса, заключенного в строке 7 с помощью ключевого слова `implements`.

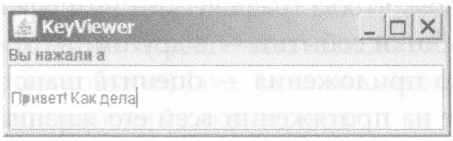


РИС. 19.1. Обработка событий клавиатуры в программе

Активизация и отключение компонентов

Наверняка вам не раз доводилось видеть неактивные программные компоненты, окрашенные в серый цвет. Подобное окрашивание означает, что пользователи ничего не могут сделать с компонентом, поскольку он не активен. Активизация и отключение компонентов осуществляются с помощью метода компонента `setEnabled()`. В качестве аргумента метода используется булево значение, поэтому вызов `setEnabled(true)` активизирует компонент, а `setEnabled(false)` отключает его.

Следующие инструкции создают кнопки с надписями Назад, Далее и Готово и отключают первую кнопку.

```
JButton previousButton = new JButton("Назад");
JButton nextButton = new JButton("Далее");
JButton finishButton = new JButton("Готово");
previousButton.setEnabled(false);
```

Это эффективный способ предотвратить генерирование пользовательских событий компонентом. Например, если создается приложение, которое собирает адреса пользователей с помощью текстовых полей, можно отключить кнопку Сохранить адрес до тех пор, пока пользователь не введет адрес, название города и области, а также почтовый индекс.

Завершенное графическое приложение

Чтобы проиллюстрировать применение классов обработки событий Swing, мы вернемся к приложению `LottoMadness` (см. занятие 18), имитирующему лотерею.

В данный момент приложение `LottoMadness` — это просто графический интерфейс, и если вы будете щелкать на кнопках и вводить текст в поля, в ответ ничего не произойдет. На этом занятии мы создадим новый класс `LottoEvent`, который получает пользовательские данные, проводит розыгрыш лотереи и отслеживает количество выигрышей. После создания класса `LottoEvent` мы добавим несколько строк кода в приложение `LottoMadness`, чтобы задействовать этот класс. Зачастую бывает удобно разделить проекты Swing таким образом, чтобы графический интерфейс находился в одном классе, а методы обработки событий — в другом.

Назначение нашего приложения — оценить шанс пользователя угадать шесть чисел в лотерее на протяжении всей его жизни. На рис. 19.2 показан снимок экрана выполняющейся программы.

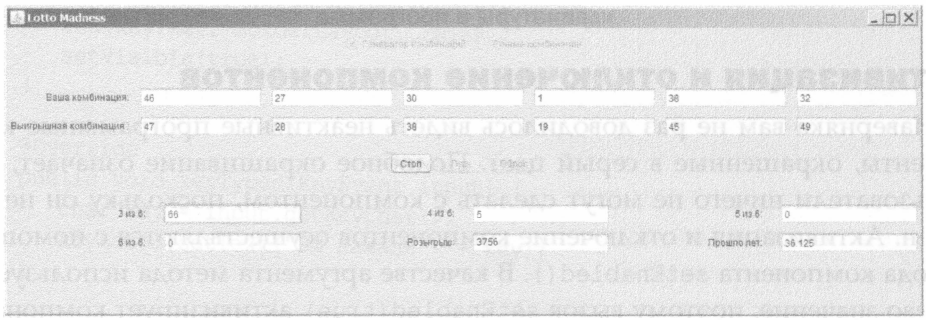


РИС. 19.2. Выполняющееся приложение `LottoMadness`

Вместо того чтобы решить эту задачу с помощью инструментов теории вероятности, программа быстро перебирает лотерейные комбинации до тех пор, пока не найдет выигрышную. Поскольку выигрышная комбинация "6 из 6" встречается чрезвычайно редко, программа отображает сообщения о любой выигрышной комбинации, состоящей из трех, четырех или пяти чисел.

Интерфейс включает 12 текстовых полей, предназначенных для чисел лотереи, и два флажка, которые называются `Генератор комбинаций` и `Ручные комбинации`. Шесть текстовых полей, заблокированных для ввода чисел, служат для отображения выигрышных номеров каждой комбинации. Оставшиеся шесть текстовых полей предназначены для ввода номеров лотереи пользователем. При установке флажка **Генератор комбинаций** выбирается шесть случайных чисел. Если же установлен флажок **Ручные комбинации**, то пользователь может ввести лотерейные номера вручную.

Пользователь управляет действиями программы с помощью трех кнопок: Стоп, Пуск и Сброс. После щелчка на кнопке Пуск программа запускает поток `playing` и генерирует лотерейные комбинации. После щелчка на кнопке Стоп выполнение потока прекращается, а после щелчка на кнопке Сброс очищаются все поля, чтобы пользователь мог все ввести заново (подробнее о потоках см. занятие 15).

Класс `LottoEvent` реализует три интерфейса: `ActionListener`, `ItemListener` и `Runnable`. Интерфейс `Runnable` связан с потоками и был рассмотрен на занятии 15. Слушатели требуются для обработки пользовательских событий, генерируемых кнопками и флажками приложения. Программе не нужно обрабатывать события, связанные с текстовыми полями, поскольку они применяются исключительно для хранения номеров, выбранных пользователем. Подобное хранение реализуется автоматически.

К программе необходимо подключить основной пакет Swing, `javax.swing`, и пакет обработки событий Java, `java.awt.event`.

Класс содержит две переменные экземпляра:

- `gui`, объект `LottoMadness`;
- `playing`, объект `Thread`, используемый для проведения непрерывного розыгрыша лотереи.

Переменная `gui` применяется для взаимодействия с объектом `LottoMadness`, который содержит графический интерфейс программы. Если нужно внести какие-то изменения в интерфейс или извлечь значение из одного из текстовых полей, используются переменные экземпляра объекта `gui`.

Например, переменная `play` объекта `LottoMadness` представляет кнопку Пуск. Чтобы отключить эту кнопку в объекте `LottoEvent`, воспользуйтесь такой инструкцией:

```
gui.play.setEnabled(false);
```

Чтобы извлечь содержимое поля `got3` (объект `JTextField`), введите следующую инструкцию:

```
String got3value = gui.got3.getText();
```

В листинге 19.2 содержится полный код класса `LottoEvent`. Создайте новый класс, включите его в пакет `com.java24hours`, введите код листинга и сохраните полученный файл `LottoEvent.java`.

ЛИСТИНГ 19.2. Исходный код программы `LottoEvent.java`

```
1: package com.java24hours;
2:
3: import javax.swing.*;
```

```
 4: import java.awt.event.*;
 5:
 6: public class LottoEvent implements ItemListener,
 7:                                     ActionListener, Runnable {
 8:
 9:    LottoMadness gui;
10:    Thread playing;
11:
12:    public LottoEvent(LottoMadness in) {
13:       gui = in;
14:    }
15:
16:    @Override
17:    public void actionPerformed(ActionEvent event) {
18:       String command = event.getActionCommand();
19:       if (command.equals("Пуск")) {
20:          startPlaying();
21:       }
22:       if (command.equals("Стоп")) {
23:          stopPlaying();
24:       }
25:       if (command.equals("Сброс")) {
26:          clearAllFields();
27:       }
28:    }
29:
30:    void startPlaying() {
31:       playing = new Thread(this);
32:       playing.start();
33:       gui.play.setEnabled(false);
34:       gui.stop.setEnabled(true);
35:       gui.reset.setEnabled(false);
36:       gui.quickpick.setEnabled(false);
37:       gui.personal.setEnabled(false);
38:    }
39:
40:    void stopPlaying() {
41:       gui.stop.setEnabled(false);
42:       gui.play.setEnabled(true);
43:       gui.reset.setEnabled(true);
44:       gui.quickpick.setEnabled(true);
45:       gui.personal.setEnabled(true);
46:       playing = null;
47:    }
48:
49:    void clearAllFields() {
50:       for (int i = 0; i < 6; i++) {
51:          gui.numbers[i].setText(null);
52:          gui.winners[i].setText(null);
```

```
53:       }
54:       gui.got3.setText("0");
55:       gui.got4.setText("0");
56:       gui.got5.setText("0");
57:       gui.got6.setText("0");
58:       gui.drawings.setText("0");
59:       gui.years.setText("0");
60:    }
61:
62:    @Override
63:    public void itemStateChanged(ItemEvent event) {
64:       Object item = event.getItem();
65:       if (item == gui.quickpick) {
66:          for (int i = 0; i < 6; i++) {
67:             int pick;
68:             do {
69:                pick = (int) Math.floor(Math.random() * 50 + 1);
70:             } while (numberGone(pick, gui.numbers, i));
71:             gui.numbers[i].setText("" + pick);
72:          }
73:       } else {
74:          for (int i = 0; i < 6; i++) {
75:             gui.numbers[i].setText(null);
76:          }
77:       }
78:    }
79:
80:    void addOneToField(JTextField field) {
81:       int num = Integer.parseInt("0" + field.getText());
82:       num++;
83:       field.setText("" + num);
84:    }
85:
86:    boolean numberGone(int num, JTextField[] pastNums, int count) {
87:       for (int i = 0; i < count; i++) {
88:          if (Integer.parseInt(pastNums[i].getText()) == num) {
89:             return true;
90:          }
91:       }
92:       return false;
93:    }
94:
95:    boolean matchedOne(JTextField win, JTextField[] allPicks) {
96:       for (int i = 0; i < 6; i++) {
97:          String winText = win.getText();
98:          if ( winText.equals( allPicks[i].getText() ) ) {
99:             return true;
100:         }
101:      }
```

```
102:       return false;
103:    }
104:
105:    @Override
106:    public void run() {
107:       Thread thisThread = Thread.currentThread();
108:       while (playing == thisThread) {
109:          addOneToField(gui.drawings);
110:          int draw = Integer.parseInt(gui.drawings.getText());
111:          float numYears = (float) draw / 104;
112:          gui.years.setText("" + numYears);
113:
114:          int matches = 0;
115:          for (int i = 0; i < 6; i++) {
116:             int ball;
117:             do {
118:                ball = (int) Math.floor(Math.random() * 50 + 1);
119:             } while (numberGone(ball, gui.winners, i));
120:             gui.winners[i].setText("" + ball);
121:             if (matchedOne(gui.winners[i], gui.numbers)) {
122:                matches++;
123:             }
124:          }
125:          switch (matches) {
126:             case 3:
127:                addOneToField(gui.got3);
128:                break;
129:             case 4:
130:                addOneToField(gui.got4);
131:                break;
132:             case 5:
133:                addOneToField(gui.got5);
134:                break;
135:             case 6:
136:                addOneToField(gui.got6);
137:                gui.stop.setEnabled(false);
138:                gui.play.setEnabled(true);
139:                playing = null;
140:          }
141:          try {
142:             Thread.sleep(100);
143:          } catch (InterruptedException e) {
144:             // Ничего не делать
145:          }
146:       }
147:    }
148: }
```

Класс LottoEvent содержит единственный конструктор LottoEvent (*LottoMadness*). Объект LottoMadness, указанный в качестве аргумента, идентифицирует объект, который полагается на объект LottoEvent для обработки пользовательских событий и проведения розыгрышей лотереи.

В классе используются следующие методы.

- clearAllFields(). Очищает все текстовые поля приложения. Вызывается после щелчка на кнопке Сброс.
- addOneToField(). Преобразует содержимое текстового поля в целочисленное значение, увеличивает это значение на единицу, а затем преобразует его обратно в содержимое текстового поля. Поскольку все текстовые поля хранятся в виде строк, нужно выполнять дополнительные действия, чтобы использовать их в выражениях.
- numberGone(). Имеет три аргумента: число из розыгрыша лотереи, массив объектов JTextField и целочисленная переменная count. Метод исключает дублирование номеров при проведении розыгрыша лотереи.
- matchedOne(). Имеет два аргумента: объект JTextField и массив из шести объектов JTextField. Метод проверяет, соответствует ли один из номеров, введенных пользователем, номерам из текущего розыгрыша лотереи.

Метод actionPerformed() получает объект события, сгенерированного после щелчка пользователя на кнопке. Применяемый в нем метод getActionCommand() возвращает надпись на кнопке. Это позволяет определить, на какой из кнопок был выполнен щелчок.

При щелчке на кнопке Пуск вызывается метод startPlaying(), отключающий четыре компонента. При щелчке на кнопке Стоп вызывается метод stopPlaying(), который активизирует каждый компонент, за исключением кнопки Стоп.

Метод itemStateChanged() обрабатывает пользовательские события, вызванные установкой флажков Генератор комбинаций и Ручные комбинации. Вызываемый в нем метод getItem() возвращает объект, представляющий флажок, на котором был выполнен щелчок мышью. Если был установлен флажок Генератор комбинаций, лотерейным номерам пользователя присваиваются шесть случайных значений, находящихся в диапазоне от 1 до 50. В противном случае текстовые поля, в которых хранятся лотерейные номера, будут очищены.

Класс LottoEvent использует номера от 1 до 50 для каждого шара в лотерейном розыгрыше. Эта методика реализуется в строке 118, в которой значение, возвращаемое методом Math.random(), умножается на 50, к результату добавляется 1, и полученное значение используется в качестве аргумента

метода `Math.floor()`. Конечный результат представляет собой случайное число в диапазоне от 1 до 50. Если заменить число 50 другим значением здесь и в строке 69, то можно будет использовать приложение `LottoMadness` для лотерейных розыгрышей с бо́льшим или меньшим диапазоном значений.

В приложении `LottoMadness` отсутствуют переменные, используемые для отслеживания количества лотерейных розыгрышей, выигрышей и текстовых полей с лотерейными номерами. Вместо этого интерфейс сам хранит значения и автоматически отображает их.

Повторно откройте в NetBeans файл `LottoMadness.java`. Чтобы это приложение могло использовать класс `LottoEvent`, необходимо добавить в него всего лишь шесть строк кода.

Прежде всего добавьте новую переменную экземпляра, которая будет хранить объект `LottoEvent`.

```
LottoEvent lotto = new LottoEvent(this);
```

Далее в конструкторе `LottoMadness()` вызовите методы `addItemListener()` и `addActionListener()` для каждого компонента интерфейса, который принимает данные, вводимые пользователем.

```
// Добавить слушателей событий
quickpick.addItemListener(lotto);
personal.addItemListener(lotto);
stop.addActionListener(lotto);
play.addActionListener(lotto);
reset.addActionListener(lotto);
```

Эти инструкции следует добавить в конструктор сразу же после вызова метода `setLayout()`, который задает менеджер компоновки `GridLayout` для фрейма приложения.

В листинге 19.3 содержится полный код приложения `LottoMadness.java` после внесения изменений. Добавленные строки выделены серым цветом, остальные строки не изменились.

ЛИСТИНГ 19.3. Исходный код программы `LottoMadness.java`

```
 1: package com.java24hours;
 2:
 3: import java.awt.*;
 4: import javax.swing.*;
 5:
 6: public class LottoMadness extends JFrame {
 7:     LottoEvent lotto = new LottoEvent(this);
 8:
 9:     // Настройка строки 1
10:     JPanel row1 = new JPanel();
```

```
11:    ButtonGroup option = new ButtonGroup();
12:    JCheckBox quickpick = new JCheckBox("Генератор комбинаций ",
13:                                        false);
14:    JCheckBox personal = new JCheckBox("Ручные комбинации", true);
15:    // Настройка строки 2
16:    JPanel row2 = new JPanel();
17:    JLabel numbersLabel = new JLabel("Ваша комбинация: ",
18:                                     JLabel.RIGHT);
19:    JTextField[] numbers = new JTextField[6];
20:    JLabel winnersLabel = new JLabel("Выигрышная комбинация: ",
21:                                     JLabel.RIGHT);
22:    JTextField[] winners = new JTextField[6];
23:    // Настройка строки 3
24:    JPanel row3 = new JPanel();
25:    JButton stop = new JButton("Стоп");
26:    JButton play = new JButton("Пуск");
27:    JButton reset = new JButton("Сброс");
28:    // Настройка строки 4
29:    JPanel row4 = new JPanel();
30:    JLabel got3Label = new JLabel("3 из 6: ", JLabel.RIGHT);
31:    JTextField got3 = new JTextField("0");
31:    JLabel got4Label = new JLabel("4 из 6: ", JLabel.RIGHT);
33:    JTextField got4 = new JTextField("0");
34:    JLabel got5Label = new JLabel("5 из 6: ", JLabel.RIGHT);
35:    JTextField got5 = new JTextField("0");
36:    JLabel got6Label = new JLabel("6 из 6: ", JLabel.RIGHT);
37:    JTextField got6 = new JTextField("0", 10);
38:    JLabel drawingsLabel = new JLabel("Розыгрыш: ", JLabel.RIGHT);
39:    JTextField drawings = new JTextField("0");
40:    JLabel yearsLabel = new JLabel("Прошло лет: ", JLabel.RIGHT);
41:    JTextField years = new JTextField("0");
42:
43:    public LottoMadness() {
44:       super("Lotto Madness");
45:
46:       setSize(550, 400);
47:       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
48:       GridLayout layout = new GridLayout(5, 1, 10, 10);
49:       setLayout(layout);
50:
51:        // Добавить слушателей
52:        quickpick.addItemListener(lotto);
53:        personal.addItemListener(lotto);
54:        stop.addActionListener(lotto);
55:        play.addActionListener(lotto);
56:        reset.addActionListener(lotto);
57:
58:       FlowLayout layout1 = new FlowLayout(FlowLayout.CENTER,
59:                                           10, 10);
```

```
60:       option.add(quickpick);
61:       option.add(personal);
62:       row1.setLayout(layout1);
63:       row1.add(quickpick);
64:       row1.add(personal);
65:       add(row1);
66:
67:       GridLayout layout2 = new GridLayout(2, 7, 10, 10);
68:       row2.setLayout(layout2);
69:       row2.add(numbersLabel);
70:       for (int i = 0; i < 6; i++) {
71:          numbers[i] = new JTextField();
72:          row2.add(numbers[i]);
73:       }
74:       row2.add(winnersLabel);
75:       for (int i = 0; i < 6; i++) {
76:          winners[i] = new JTextField();
77:          winners[i].setEditable(false);
78:          row2.add(winners[i]);
79:       }
80:       add(row2);
81:
82:       FlowLayout layout3 = new FlowLayout(FlowLayout.CENTER,
83:                                           10, 10);
84:       row3.setLayout(layout3);
85:       stop.setEnabled(false);
86:       row3.add(stop);
87:       row3.add(play);
88:       row3.add(reset);
89:       add(row3);
90:
91:       GridLayout layout4 = new GridLayout(2, 3, 20, 10);
92:       row4.setLayout(layout4);
93:       row4.add(got3Label);
94:       got3.setEditable(false);
95:       row4.add(got3);
96:       row4.add(got4Label);
97:       got4.setEditable(false);
98:       row4.add(got4);
99:       row4.add(got5Label);
100:      got5.setEditable(false);
101:      row4.add(got5);
102:      row4.add(got6Label);
103:      got6.setEditable(false);
104:      row4.add(got6);
105:      row4.add(drawingsLabel);
106:      drawings.setEditable(false);
107:      row4.add(drawings);
108:      row4.add(yearsLabel);
```

```
109:        years.setEditable(false);
110:        row4.add(years);
111:        add(row4);
112:
113:        setVisible(true);
114:     }
115:
116:     private static void setLookAndFeel() {
117:        try {
118:           UIManager.setLookAndFeel(
119:              "com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel"
120:           );
121:        } catch (Exception exc) {
122:           // Игнорировать ошибки
123:        }
124:     }
125:
126:     public static void main(String[] arguments) {
127:        LottoMadness.setLookAndFeel();
128:        LottoMadness frame = new LottoMadness();
129:     }
130: }
```

После добавления выделенных строк запустите приложение, чтобы оценить свои шансы на выигрыш в лотерею. Как и следовало ожидать, игра в лотерею — бесполезное занятие. Шанс угадать все шесть номеров настолько мал, что вам и жизни не хватит, даже если вы будете жить так долго, как библейские патриархи.

ПРИМЕЧАНИЕ

После запуска программы `LottoMadness` было просчитано 410 732 244 возможных розыгрыша, что заняло бы около 3,9 млн лет при условии проведения розыгрышей дважды в неделю. В результате было определено 6 364 880 победителей, угадавших 3 из 6 номеров, 337 285 победителей, которые угадали 4 из 6 номеров, 6 476 победителей, которые угадали 5 из 6 номеров, и 51 победитель, который угадал 6 из 6 номеров (приблизительно один победитель на каждые 8 млн розыгрышей). Первый выигрыш был получен после участия в 241 225 розыгрышах, на что было потрачено почти 2 320 лет.

Резюме

Благодаря библиотеке Swing можно создавать профессионально выглядящие программы, прилагая для этого совсем небольшие усилия. Несмотря на то что приложение `LottoMadness` больше по размерам, чем многие из примеров, которые были рассмотрены на предыдущих занятиях, в действительности половина кода этой программы состоит из инструкций, предназначенных для создания интерфейса.

Если вы какое-то время поработаете с приложением, имитирующим проведение лотереи, то станете еще больше завидовать удаче людей, которые таки угадали заветные шесть номеров. Когда я последний раз запустил программу, я пришел к выводу, что мне придется потратить 17 000 долл. и лучшие 165 лет моей жизни, чтобы один раз угадать 5 номеров из 6, 10 раз угадать 4 номера из 6 и 264 раза угадать 3 номера из 6. По сравнению с этими сомнительными выгодами шансы улучшить навыки программирования на Java после прочтения этой книги близки к 100%.

Вопросы и ответы

В. Нужно ли вызывать метод `paint()` или `repaint()`, чтобы указать на изменение текстового поля?

О. После использования метода `setText()` для изменения значения текстового компонента вам не придется делать еще что-то. В подобном случае все необходимые действия автоматически выполняются в Swing.

В. Почему так часто приходится импортировать класс и один из его подклассов, как в листинге 19.1, где выполнялись инструкции `import java.awt.*` и `java.awt.event.*`? Может ли первая из этих инструкций замещать вторую?

О. Несмотря на то что имена пакетов `java.awt` и `java.awt.event` подразумевают их родство, в Java сами пакеты не наследуются. Один пакет не может входить в состав другого.

В случае использования символа звездочки в инструкции импорта все классы пакета становятся доступными в программе. Но это касается только имен классов, а не пакетов. Одиночная инструкция `import` загружает классы из одного пакета.

Коллоквиум

Проверьте знание материала, рассмотренного на этом занятии, ответив на следующие вопросы.

Контрольные вопросы

1. Почему события действия называются именно так?

   А. Они происходят в ответ на что-либо.

   Б. Они указывают на то, что в ответ следует предпринять какое-то действие.

   В. Они весьма действенны.

2. Что означает ключевое слово `this` в качестве аргумента метода `addActionListener()`?

   А. Для обработки события должен применяться текущий слушатель.

   Б. Текущее событие получает приоритет над другими событиями.

   В. События будут обрабатываться текущим объектом.

3. Какой текстовый компонент хранит вводимые пользователем данные в виде целых чисел?

   А. `JTextField`.

   Б. `JTextArea`.

   В. Ни А, ни Б.

Ответы

1. **Б.** События действий включают щелчки на кнопке и выбор элемента в раскрывающемся меню.
2. **В.** Ключевое слово `this` ссылается на текущий объект. Если вместо `this` указать имя объекта, то этот объект будет перехватывать события и, как ожидается, обрабатывать их.
3. **В.** Компоненты `JTextField` и `JTextArea` хранят свои значения в виде текста, поэтому, прежде чем использовать их в качестве целочисленных значений, чисел с плавающей точкой или других нетекстовых значений, необходимо выполнить соответствующее преобразование.

Упражнения

Выполните следующие упражнения, чтобы закрепить изученный материал.

- Добавьте в приложение `LottoMadness` текстовое поле, которое используется совместно с инструкцией `Thread.sleep()` в классе `LottoEvent`, чтобы замедлить скорость проведения розыгрышей.
- Измените проект `LottoMadness` таким образом, чтобы можно было угадать пять номеров из диапазона 1–90.

Часть VI
Создание интернет-приложений

В этой части...

- Занятие 20 Чтение и запись файлов
- Занятие 21 Использование HTTP-клиента
- Занятие 22 Создание двумерной графики
- Занятие 23 Создание модов для Minecraft с помощью Java
- Занятие 24 Создание приложений для Android

ЗАНЯТИЕ 20
Чтение и запись файлов

На этом занятии вы узнаете...

- как считывать байты из файла;
- как создать новый файл;
- как сохранить байтовый массив в файле;
- как внести изменения в файл.

Существует множество способов представления данных на компьютере. Вы уже применяли один из них, заключающийся в создании объектов. Объект содержит переменные и ссылки на другие объекты, а также методы, использующие данные для решения задач.

Для работы с другими видами данных, такими как файлы на жестком диске и документы на веб-сервере, можно использовать классы из пакета `java.io`. Суффикс "io" означает "input/output" (ввод/вывод). Эти классы служат для получения доступа к источнику данных, например к жесткому диску, DVD или памяти компьютера.

Данные можно получать в программе и отправлять из программы с помощью коммуникационного механизма, называемого *потоками ввода-вывода*, которые являются объектами, передающими информацию из одного места в другое.

Потоки ввода-вывода

Поток ввода-вывода (stream) — это объект, который извлекает информацию из одного источника, а затем пересылает ее в другое место. Потоки ввода-вывода подключаются к различным источникам, включая компьютерные программы, жесткие диски, серверы Интернета, компьютерную память и флеш-накопители. После того как вы научитесь работать с одним типом

данных, используя потоки ввода-вывода, вы сможете аналогичным образом работать с другими типами данных.

На этом занятии вы будете использовать потоки ввода-вывода для чтения и записи данных, которые хранятся в файлах на вашем компьютере.

Существуют два вида потоков ввода-вывода:

- входные, которые считывают данные из источника;
- выходные, которые записывают данные в источник.

Все входные и выходные потоки состоят из байтов, представляющих собой целые числа в диапазоне от 0 до 255. Подобный формат удобен для работы с такими данными, как исполняемые программы, документы и музыкальные файлы в формате MP3, хотя это лишь малая толика того, что можно представлять с помощью байтов.

ПРИМЕЧАНИЕ

Файлы классов Java хранятся в байтовом формате, который называется *байт-кодом*. Виртуальная машина Java выполняет любой байт-код, не обязательно сгенерированный программой на Java. Это может быть скомпилированный байт-код, созданный программами, которые написаны на других языках, включая Scala, Groovy и Jython.

Более специализированный способ управления данными заключается в работе на уровне символов — отдельных букв, цифр, знаков пунктуации и т.п. Символьный поток можно использовать при чтении и записи текстового источника данных.

Независимо от того, работаете ли вы с потоками байтов, символов или другими видами информации, выполняется один и тот же процесс:

- создается объект потока, связанный с данными;
- вызываются методы потока, применяемые для занесения информации в поток или извлечения из него данных;
- поток закрывается с помощью метода `close()`.

Файлы

В Java файлы представлены классом `File`, который входит в пакет `java.io`. Файлы могут считываться с жестких дисков, DVD и других устройств хранения данных.

Объект `File` может соответствовать уже существующему файлу или же файлу, который вы только намереваетесь создать. Чтобы создать объект `File`, укажите имя файла в качестве аргумента конструктора, как показано в следующем примере:

```
File book = new File("address.dat");
```

В результате выполнения этой инструкции создается объект для файла `address.dat` в текущей папке. В состав имени файла можно также включить путь.

```
File book = new File("data\\address.dat");
```

ПРИМЕЧАНИЕ

Этот пример работает в Windows, где символ обратной косой черты (\) служит разделителем в путевых именах. (Он удваивается вследствие экранирования, поскольку является одним из специальных символов в Java.) В Linux и других Unix-системах применяется символ обычной косой черты. Чтобы создать программу на Java, которая ссылается на файлы платформенно-независимым способом, вместо косой черты (обычной или обратной) используйте переменную `File.pathSeparator`:

```
File book = new File("data" + File.pathSeparator + "address.dat");
```

Создание объекта `File` не приводит к автоматическому созданию файла на компьютере. Это просто ссылка на файл, которого еще может и не быть.

У объекта `File` есть несколько полезных методов:

- `exists()` — возвращает `true`, если файл существует, и `false` в противном случае;
- `getName()` — возвращает имя файла в виде объекта `String`;
- `length()` — возвращает размер файла в виде значения `long`;
- `createNewFile()` — создает файл с тем же именем, если он еще не существует;
- `delete()` — удаляет существующий файл;
- `renameTo(`*`File`*`)` — переименовывает файл на основе имени объекта *`File`*, указанного в качестве аргумента.

Объект `File` может также представлять папку, а не файл. Укажите имя папки в конструкторе `File`, которое может быть абсолютным (например, `C:\\Documents\\`) или относительным (например, `java\\database`).

После того как в вашем распоряжении окажется объект, представляющий папку, вызовите метод `listFiles()`, чтобы получить содержимое папки. Этот метод возвращает массив объектов `File`, соответствующих вложенным файлам и папкам.

Чтение данных из потока

Первый проект, который мы создадим на этом занятии, считывает данные из файла с помощью входного потока. При этом используется класс `FileInputStream`, представляющий входные потоки, которые считываются из файла в байтовом виде.

Входной файловый поток можно создать путем указания имени файла или объекта `File` в качестве аргумента конструктора `FileInputStream()`.

Методы, которые выполняют чтение и запись файлов, могут завершаться аварийно, генерируя исключение `IOException` в случае ошибки доступа к файлу. Поэтому зачастую такие методы приходится заключать в блок `try-catch`. Класс указанного исключения находится в пакете `java.io`.

Потоки ввода-вывода относятся к ресурсам Java, которые следует закрывать, если они больше не используются. Если оставить такой поток открытым, это приведет к существенной утечке ресурсов виртуальной машины JVM.

Чтобы гарантировать закрытие ресурса, такого как входной файловый поток, в случае, когда он больше не нужен, следует использовать специальную разновидность инструкции `try`, которая называется инструкцией `try` с ресурсами. После ключевого слова `try` в круглых скобках указывается одна или несколько инструкций, объявляющих переменные, которые считывают или записывают данные через контролируемый ресурс.

Ниже приведен пример кода, в котором текстовый файл `cookie.web` считывается с помощью входного файлового потока `stream`.

```
File cookie = new File("cookie.web");
try (FileInputStream stream = new FileInputStream(cookie)) {
   System.out.println("Длина файла: " + cookie.length());
} catch (IOException ioe) {
   System.out.println("Невозможно считать файл.");
}
```

Поскольку поток ввода-вывода создается в инструкции `try`, он автоматически закрывается после завершения блока `try-catch` (если он не был закрыт в явном виде).

Входные файловые потоки считывают байтовые данные. Чтобы прочитать один байт, вызовите метод потока `read()` без аргумента. Если по достижении конца файла в потоке не остается доступных байтов, возвращается значение `-1`.

Процесс считывания входного потока начинается с первого байта потока, например первого байта файла. Чтобы пропустить некоторые байты в потоке, вызовите метод `skip()` с целочисленным аргументом, задающим количество пропускаемых байтов. Следующая инструкция пропускает 1024 байта в потоке `scanData`:

```
scanData.skip(1024);
```

Чтобы прочитать несколько байтов за один раз, выполните следующие действия.

- Создайте байтовый массив, размер которого в точности соответствует количеству считываемых байтов.

- Вызовите метод потока `read()`, указав созданный байтовый массив в качестве аргумента. Массив будет заполнен байтами, прочитанными из потока.

Наша первая программа считывает метаданные (теги) ID3 из MP3-файла. MP3 — это популярный формат музыкальных файлов. В конец MP3-файла часто добавляются 128 байтов метаданных ID3, предназначенных для хранения сведений о музыкальной композиции (название, исполнитель, альбом и т.п.).

Приложение `ID3Reader` считывает MP3-файл из входного потока, пропуская все, кроме последних 128 байтов. Оставшиеся байты проверяются на наличие метаданных ID3, и если таковые имеются, то первые три байта будут числами 84, 65 и 71.

ПРИМЕЧАНИЕ

В наборе символов ASCII, который включен в стандартный набор Unicode, поддерживаемый в Java, эти три числа соответствуют заглавным буквам "T," "A" и "G" соответственно.

Создайте новый класс `ID3Reader`, включите его в пакет `com.java24hours`, введите код из листинга 20.1 и сохраните полученный файл `ID3Reader.java`.

ЛИСТИНГ 20.1. Исходный код программы `ID3Reader.java`

```
 1: package com.java24hours;
 2:
 3: import java.io.*;
 4:
 5: public class ID3Reader {
 6:    public static void main(String[] arguments) {
 7:       File song = new File(arguments[0]);
 8:       try (FileInputStream file = new FileInputStream(song)) {
 9:          int size = (int) song.length();
10:          file.skip(size - 128);
11:          byte[] last128 = new byte[128];
12:          file.read(last128);
13:          String id3 = new String(last128);
14:          String tag = id3.substring(0, 3);
15:          if (tag.equals("TAG")) {
16:             System.out.println("Название: " +
17:                                id3.substring(3, 32));
18:             System.out.println("Исполнитель: " +
19:                                id3.substring(33, 62));
20:             System.out.println("Альбом: " +
21:                                id3.substring(63, 91));
22:             System.out.println("Год: " +
23:                                id3.substring(93, 97));
24:          } else {
25:             System.out.println(arguments[0] + " не содержит " +
```

```
26:                                    "информацию ID3.");
27:          }
28:          file.close();
29:       } catch (IOException ioe) {
30:          System.out.println("Ошибка -- " + ioe.toString());
31:       }
32:    }
33: }
```

Прежде чем запустить программу, следует указать MP3-файл в качестве аргумента командной строки (с помощью команды Выполнить⇨Установить конфигурацию проекта⇨Настроить). Это может быть любой MP3-файл, например `Come On And Gettit.mp3` — песня, которую в 1973 году исполнил соул-певец Марион Блэк. Если файл найден в вашей системе, то приложение отобразит примерно такой результат, как на рис. 20.1.

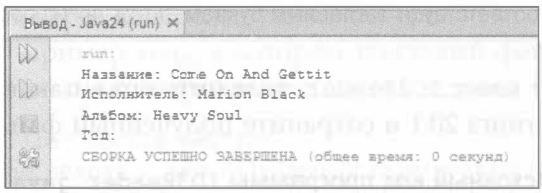


РИС. 20.1. Результат выполнения приложения `ID3Reader`

СОВЕТ

Композицию `Come On And Gettit.mp3` можно найти в Интернете.

Программа считывает последние 128 байтов из файла MP3 (строки 11 и 12) и сохраняет их в байтовом массиве, который используется в строке 13 для создания объекта `String`, содержащего символы, представленные этими байтами.

Если первые три символа в строке — "TAG", значит, проверяемый файл MP3 содержит теги ID3.

В строках 16–23 вызывается метод `substring()`, возвращающий фрагменты строки. В формате ID3 сведения об исполнителе, альбоме, названии композиции и годе выхода альбома находятся в одних и тех же позициях в последних 128 байтах.

Некоторые MP3-файлы могут не включать метаданные ID3 либо содержать их в формате, который не поддерживается данным приложением. В нашем случае заполнены все поля метаданных, кроме года.

После того как все теги ID3 были прочитаны из входного потока MP3-файла, поток закрывается (строка 28). Всегда закрывайте потоки после завершения работы с ними, чтобы не расходовать понапрасну ресурсы виртуальной машины Java.

Буферизованные потоки ввода

Один из способов повышения производительности программы, которая считывает входные потоки, — буферизировать входные данные. *Буферизация* — это процесс сохранения данных в памяти для дальнейшего использования программой по мере необходимости. Если программа нуждается в данных из буферизованного входного потока, она сначала ищет их в буфере. Это существенно быстрее, чем считывать данные из источника, такого как файл.

Чтобы использовать буферизованный входной поток, необходимо сначала создать входной поток, например объект `FileInputStream`, а затем передать его конструктору `BufferedInputStream(`*`ВходнойПоток`*`)`. Данные буферизуются по мере их считывания из входного потока.

Для чтения данных из буферизованного потока вызовите его метод `read()` без аргументов. В результате возвращается целочисленное значение от 0 до 255, которое представляет следующий байт данных в потоке. Если же байтов в потоке больше нет, возвращается значение `-1`.

Чтобы продемонстрировать использование буферизованных потоков, мы напишем программу, которая реализует ввод данных с консоли. *Консольный ввод* — это возможность чтения символов из консоли (ее называют командной строкой) при выполнении приложения.

Класс `System`, который содержит переменную `out`, используемую в инструкциях `System.out.print()` и `System.out.println()`, включает также переменную `in`. Она представляет объект `InputStream`, который обрабатывает данные, вводимые с клавиатуры, и делает их доступными в виде потока.

С этим входным потоком можно работать точно так же, как и с любыми другими потоками. Следующая инструкция создает буферизованный входной поток, связанный с входным потоком `System.in`:

```
BufferedInputStream bin = new BufferedInputStream(System.in);
```

В следующем проекте, рассмотренном на данном занятии, программа содержит метод класса, который можно использовать для получения консольного ввода в любом из ваших приложений Java. Создайте новый класс `Console`, включите его в пакет `com.java24hours`, введите код из листинга 20.2 и сохраните полученный файл `Console.java`.

ЛИСТИНГ 20.2. Исходный код программы `Console.java`

```
1: package com.java24hours;
2:
3: import java.io.*;
4:
5: public class Console {
```

```
 6:    public static String readLine() {
 7:       StringBuilder response = new StringBuilder();
 8:       try {
 9:          BufferedInputStream bin = new
10:             BufferedInputStream(System.in);
11:          int in = 0;
12:          char inChar;
13:          do {
14:             in = bin.read();
15:             inChar = (char) in;
16:             if (in != -1) {
17:                response.append(inChar);
18:             }
19:          } while ((in != -1) & (inChar != '\n'));
20:          bin.close();
21:          return response.toString();
22:       } catch (IOException e) {
23:          System.out.println("Исключение: " + e.getMessage());
24:          return null;
25:       }
26:    }
27:
28:    public static void main(String[] arguments) {
29:       System.out.print("You are standing at the end of the ");
30:       System.out.print("road before a small brick building. ");
31:       System.out.print("Around you is a forest. A small ");
32:       System.out.print("stream flows out of the building ");
33:       System.out.println("and down a gully.\n");
34:       System.out.print("> ");
35:       String input = Console.readLine();
36:       System.out.println("That's not a verb I recognize.");
37:    }
38: }
```

Программа включает метод `main()`, с помощью которого демонстрируется использование консольного ввода. Результат выполнения данного приложения показан на рис. 20.2.

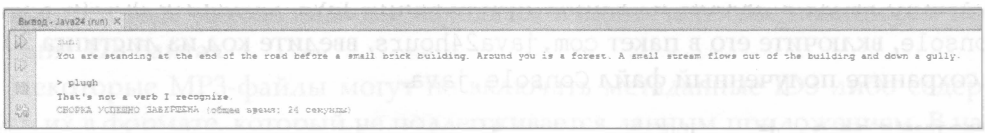


РИС. 20.2. Результат выполнения приложения `Console`

Класс `Console` содержит единственный метод класса, `readLine()`, предназначенный для чтения символов с консоли. После нажатия клавиши <Enter> метод `readLine()` возвращает строковый объект, содержащий прочитанные символы.

Запись данных в поток

В пакете `java.io` содержатся парные классы, предназначенные для работы с потоками. Это классы `FileInputStream` и `FileOutputStream`, предназначенные для работы с байтовыми потоками, классы `FileReader` и `FileWriter` для работы с символьными потоками и множество других пар для работы с иными типами потоковых данных.

Чтобы осуществить запись данных, нужно сначала создать объект `File`, связанный с выходным потоком. При этом сам файл может не существовать в системе.

Объект `FileOutputStream` можно создать двумя способами. Чтобы добавить байты в существующий файл, вызовите конструктор `FileOutputStream()` с двумя аргументами: объектом `File`, представляющим файл, и булевым значением `true`. Байты, записываемые в поток, добавляются в конец файла.

Чтобы записать байты в новый файл, вызовите конструктор `FileOutputStream()` с единственным аргументом: объектом `File`.

После создания выходного потока можно вызывать разные методы `write()` для записи байтов в этот поток.

- Вызовите метод `write()` с единственным аргументом типа `byte`, чтобы записать этот байт в поток.
- Вызовите метод `write()` с аргументом в виде байтового массива, чтобы записать все байты массива в поток.
- Если необходимо записать в поток фрагмент байтового массива, укажите три аргумента метода `write(byte[], int, int)`: байтовый массив, целое число, представляющее первый элемент массива, и количество записываемых байтов.

Следующая инструкция создает байтовый массив, включающий 10 байтов, и записывает последние четыре байта в выходной поток.

```
File dat = new File("data.dat");
FileOutputStream datStream = new FileOutputStream(dat);
byte[] data = new byte[] { 5, 12, 4, 13, 3, 15, 2, 17, 1, 18 };
datStream.write(data, 6, 4);
```

При записи байтов в поток можно преобразовать текст в байтовый массив путем вызова метода `getBytes()` объекта `String`.

```
String name = "Puddin N. Tane";
byte[] nameBytes = name.getBytes();
```

Записав байты в поток, закройте его, вызвав метод потока `close()`.

Следующий проект, создаваемый на этом занятии, — простое приложение `ConfigWriter`, которое записывает несколько строк текста в файл путем записи байтов в выходной поток. Создайте новый класс Java, включите его в пакет `com.java24hours`, введите код из листинга 20.3 и сохраните полученный файл `ConfigWriter.java`.

ЛИСТИНГ 20.3. Исходный код программы `ConfigWriter.java`

```
 1: package com.java24hours;
 2:
 3: import java.io.*;
 4:
 5: public class ConfigWriter {
 6:    String newline = System.getProperty("line.separator");
 7:
 8:    public ConfigWriter() {
 9:       try {
10:          File file = new File("program.properties");
11:          FileOutputStream fileStream = new FileOutputStream(file);
12:          write(fileStream, "username=max");
13:          write(fileStream, "score=12550");
14:          write(fileStream, "level=5");
15:          fileStream.close();
16:       } catch (IOException ioe) {
17:          System.out.println("Невозможно записать файл");
18:       }
19:    }
20:
21:    void write(FileOutputStream stream, String output)
22:       throws IOException {
23:
24:       output = output + newline;
25:       byte[] data = output.getBytes();
26:       stream.write(data, 0, data.length);
27:    }
28:
29:    public static void main(String[] arguments) {
30:       new ConfigWriter();
31:    }
32: }
```

После запуска приложения будет создан файл `program.properties`, содержащий три строки текста.

```
username=max
score=12550
level=5
```

Файл создается в строке 10 и связывается с выходным потоком в строке 11. Три строки свойств записываются в поток в строках 12–14.

Приложение, запущенное в среде NetBeans, сохраняет созданный файл (или файлы) в основной папке проекта, если не указано иное. Чтобы просмотреть файл `program.properties` в NetBeans, щелкните на вкладке **Файлы** панели **Проекты**. Название файла отображается в верхней части папки `Java24`. В этой папке дважды щелкните на названии `program.properties`, чтобы открыть файл, содержимое которого показано на рис. 20.3.

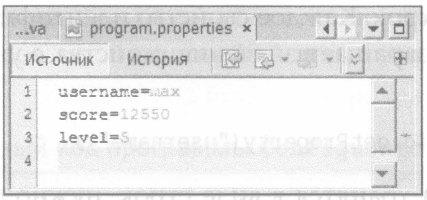


РИС. 20.3. Содержимое файла `program.properties`

Считывание и запись конфигурационных параметров

Программы на Java становятся более универсальными, когда их можно настраивать с помощью аргументов командной строки, как уже было продемонстрировано в примерах, рассмотренных на предыдущих занятиях. Пакет `java.util` включает класс `Properties`, позволяющий загружать конфигурационные настройки из другого источника: текстового файла.

Конфигурационный файл можно считывать так же, как и другие файловые источники в Java:

1) создайте объект `File`, который представляет файл;
2) на основе объекта `File` создайте объект `FileInputStream`;
3) вызовите метод `load()` объекта `Properties` для чтения свойств из этого входного потока.

Файл свойств содержит набор строк, в каждой из которых указано имя свойства и его значение после знака равенства. Вот соответствующий пример.

```
username=lepton
lastCommand=open database
windowSize=32
```

Каждое свойство занимает отдельную строку, поэтому в данном примере устанавливаются свойства `username`, `lastCommand` и `windowSize`, которым

присваиваются значения lepton, open database и 32 соответственно. (Аналогичный формат применялся приложением ConfigWriter.)

Следующий код загружает файл свойств config.dat.

```
File configFile = new File("config.dat");
FileInputStream inStream = new FileInputStream(configFile);
Properties config = new Properties();
config.load(inStream);
```

Конфигурационные настройки, называемые *свойствами*, хранятся в объекте Properties в виде строк. Каждое свойство идентифицируется ключом. Метод getProperty() возвращает значение свойства с указанным ключом, как показано ниже:

```
String username = config.getProperty("username");
```

Поскольку свойства хранятся в виде строк, нужно выполнить определенное преобразование для использования числовых значений, как показано в следующем примере.

```
String windowProp = config.getProperty("windowSize");
int windowSize;
try {
   windowSize = Integer.parseInt(windowProp);
} catch (NumberFormatException exception) {
   // Ничего не делать
}
```

Чтобы задать значение свойства, нужно вызвать метод setProperty() с двумя аргументами: ключом и значением:

```
config.setProperty("username", "max");
```

Для отображения всех свойств можно воспользоваться методом list(*PrintStream*) объекта Properties. Тип PrintStream уже использовался ранее: это класс переменной экземпляра out из класса System, которая фигурирует в инструкциях вывода System.out.println() и System.out.print(). Следующая инструкция вызывает метод list() для отображения всех свойств:

```
config.list(System.out);
```

После внесения изменений в свойства можно записать их обратно в файл. Для этого выполните следующие действия:

- создайте объект File, который представляет файл;
- на основе объекта File создайте объект FileOutputStream;

- вызовите метод `store(OutputSream, String)` объекта `Properties`, чтобы сохранить свойства в указанном выходном потоке, предоставив описание файла свойств в виде строки.

Заключительный проект, создаваемый на этом занятии, представляет собой приложение `Configurator`, которое записывает несколько настроек программы в файл. Программа считывает эти настройки из файла свойств, добавляет новое свойство `runtime` со значениями текущей даты и времени и сохраняет измененный файл.

Создайте новый класс `Configurator`, включите его в пакет `com.java24hours`, введите код из листинга 20.4 и сохраните полученный файл `Configurator.java`.

ЛИСТИНГ 20.4. Исходный код программы `Configurator.java`

```
 1: package com.java24hours;
 2:
 3: import java.io.*;
 4: import java.util.*;
 5:
 6: public class Configurator {
 7:
 8:    public Configurator() {
 9:       try {
10:          // Загрузка файла свойств
11:          File configFile = new File("program.properties");
12:          FileInputStream inStream = new
13:                                FileInputStream(configFile);
14:          Properties config = new Properties();
15:          config.load(inStream);
16:          // Создание нового свойства
17:          Date current = new Date();
18:          config.setProperty("runtime", current.toString());
19:          // Сохранение файла свойств
20:          FileOutputStream outStream = new
21:                                 FileOutputStream(configFile);
22:          config.store(outStream, "Properties settings");
23:          inStream.close();
24:          config.list(System.out);
25:       } catch (IOException ioe) {
26:           System.out.println("Ошибка ввода/вывода " +
27:                              ioe.getMessage());
28:       }
29:    }
30:
31:    public static void main(String[] arguments) {
```

```
32:         new Configurator();
33:     }
34: }
```

При выполнении этого приложения создается объект `File` для файла `program.properties`, который ассоциируется с входным потоком в строках 11–13. Содержимое файла загружается из потока в объект `Properties` в строках 14 и 15.

В строках 17 и 18 создается новое свойство для текущей даты/времени. Далее файл свойств связывается с выходным потоком (строки 20 и 21), и в этот поток записывается весь файл свойств (строка 22).

Результат выполнения приложения `Configurator` показан на рис. 20.4.

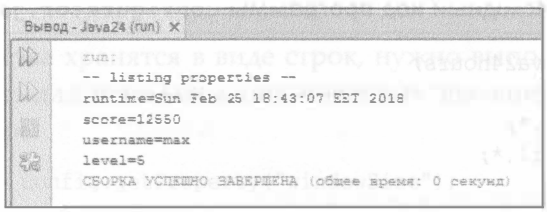


РИС. 20.4. Результат выполнения приложения `Configurator`

Теперь файл `program.properties` содержит следующий текст.

```
-- listing properties --
runtime=Sun Feb 25 18:43:07 EET 2018
score=12550
username=max
level=5
```

Резюме

На этом занятии вы поработали с входными и выходными байтовыми потоками. Это простейший способ передачи данных в потоковом виде.

Пакет `java.io` включает множество других классов, предназначенных для работы с иными видами потоков. Существует также пакет `java.net`, содержащий классы для чтения и записи потоков через интернет-подключение.

Байтовые потоки находят множество применений, поскольку байты можно легко конвертировать в другие типы данных, такие как целые числа, символы и строки.

Вопросы и ответы

В. Почему некоторые методы байтовых потоков, рассмотренные на этом занятии, в качестве аргументов используют целые числа? Не должны ли они использовать аргументы типа `byte`?

О. Байты в потоке и байты, представленные классом `byte`, интерпретируются по-разному. Значения типа `byte` в Java изменяются в диапазоне от –128 до 127, тогда как байт в потоке может иметь значение от 0 до 255. По этой причине при работе с байтовыми потоками приходится использовать тип данных `int`, поскольку он позволяет хранить диапазон значений от 128 до 255, недоступный для типа `byte`.

Коллоквиум

Проверьте знание материала, рассмотренного на этом занятии, ответив на следующие вопросы.

Контрольные вопросы

1. Какая из перечисленных ниже методик может применяться для преобразования массива байтов в строку?
   - **А.** Вызов метода массива `toString()`.
   - **Б.** Преобразование каждого байта в символ с последующим присваиванием каждого символа элементу массива `String`.
   - **В.** Вызов конструктора `String()` с аргументом в виде массива.
2. Какой тип потока применяется для чтения файла в программе на Java?
   - **А.** Входной поток.
   - **Б.** Выходной поток.
   - **В.** Оба.
3. С помощью какого метода класса `File` можно определить размер файла?
   - **А.** `getSize()`.
   - **Б.** `read()`.
   - **В.** `length()`.

Ответы

1. **В.** С каждым байтом можно работать отдельно, как и предполагает ответ Б, но легче создавать строки на основе других типов данных.
2. **А.** Входной поток создается на основе объекта `File` или путем предоставления имени файла конструктору входного потока.

3. **B.** Этот метод возвращает значение типа `long`, представляющее количество байтов в потоке.

Упражнения

Выполните следующие упражнения, чтобы закрепить изученный материал.

- Создайте приложение, которое считывает теги ID3 всех MP3-файлов в папке и переименовывает файлы на основе сведений об исполнителе, песне или альбоме (при наличии подобной информации).
- Напишите программу, которая считывает файл исходного кода Java и записывает его обратно без каких-либо изменений, но под новым именем.

ЗАНЯТИЕ 21
Использование HTTP-клиента

На этом занятии вы узнаете...

- как добавить модуль в проект Java;
- как создать HTTP-объект веб-браузера;
- как создать GET-запрос к веб-серверу;
- как получить данные из веб-запроса;
- как загрузить файл изображения из Интернета;
- как отправить данные веб-серверу.

Веб-серверы уже давно вышли за рамки своего изначального предназначения — отображения веб-страниц в окнах браузеров. Возможность взаимодействия по протоколу HTTP используется многими программами, включая RSS-агрегаторы, веб-службы, программные обновления и операционные системы.

В Java 9 появилась библиотека HTTP Client, обеспечивающая более простой и надежный способ передачи данных через Интернет. Но она не включается автоматически в библиотеку классов Java. Вместо этого она добавляется через новое средство языка — *модули*.

На этом занятии вы узнаете о модулях и способах создания веб-подключений по протоколу HTTP.

Модули Java

Готовые программы на Java упаковываются в файлы архивов Java (Java Archive, JAR) вместе с библиотеками классов, используемыми программой. Файлы JAR, необходимые для проекта, помещаются в каталоги, указанные в переменной среды `Classpath`, что позволяет виртуальной машине Java (JVM) получать к ним доступ при выполнении программы.

Неэффективность этого подхода уже давно стала очевидной. В состав файла JAR могут входить сотни или даже тысячи классов, сгруппированных

по пакетам. Один и тот же класс может находиться в двух разных каталогах Classpath, что ведет к путанице в процессе выполнения программы, использующей этот класс.

В версии Java 9 и выше обеспечивается больший контроль над развертыванием программ с помощью модулей. Чтобы новая библиотека HTTP Client, которая представляет собой пакет jdk.incubator.http, стала доступна для пользовательских программ, нужно включить ее в модуль.

Выполните следующие действия в среде NetBeans, чтобы добавить модуль в проект Java24.

- Выберите команду **Файл⇨Создать файл**, а затем на панели **Категории** выберите пункт **Java**.
- На панели **Типы файлов** выберите пункт **Информация о пакете Java** и щелкните на кнопке **Далее**.
- В поле **Имя класса** отобразится имя module-info, которое нельзя изменить. Щелкните на кнопке **Готово**.

Файл module-info.java откроется в редакторе исходного кода. Он должен включать следующие три строки кода.

```
module Java24 {
   requires jdk.incubator.httpclient;
}
```

После сохранения файла классы из пакета jdk.incubator.httpclient смогут использоваться любой программой Java в этом проекте.

Создание HTTP-запроса

Веб-сервер взаимодействует с браузерами и другими клиентскими веб-приложениями путем обмена сообщениями, называемыми HTTP-запросами.

Запрос может применяться для передачи информации в обоих направлениях. Данные могут приниматься и отсылаться между сервером и клиентом.

В первом проекте данного занятия демонстрируется применение библиотеки HTTP Client для подключения к веб-серверу, запрашивания документа и получения информации о сервере. И хотя это простой процесс, он требует использования нескольких классов.

Чтобы создать веб-запрос с помощью этой библиотеки, выполните следующие действия.

1. Создайте объект браузера (класс HttpClient).
2. Создайте построитель запросов (внутренний класс HttpRequest.Builder).

3. Постройте запрос в виде объекта `HttpRequest`.
4. С помощью браузера отправьте запрос веб-серверу.
5. Получите обратно объект `HttpResponse`.

Объект браузера создается с помощью фабричного метода класса `HttpClient`:

```
HttpClient browser = HttpClient.newHttpClient();
```

Как и следовало ожидать, функция браузера заключается в передаче запроса серверу.

При создании запроса применяется класс-построитель. Построителю требуется веб-адрес сервера (также называемый URI или URL). Этот адрес создается с помощью класса `URI` из пакета `java.net`:

```
URI link = new URI("https://www.oracle.com/");
```

Если веб-адрес, указанный в качестве аргумента конструктора, некорректно отформатирован, будет сгенерировано исключение `URISyntaxException` (из пакета `java.net`).

Чтобы создать построитель запросов с помощью объекта `URI`, необходимо воспользоваться фабричным методом `HttpRequest.newBuilder(`*`URI`*`)`:

```
HttpRequest.Builder bob = HttpRequest.newBuilder(uri);
```

Метод `build()` построителя создает запрос, который является объектом класса `HttpRequest`:

```
HttpRequest request = bob.build();
```

Теперь, когда в вашем распоряжении имеются браузер и запрос, можно отправить запрос веб-серверу и получить отклик. Ответ сервера представляет собой объект `HttpResponse`, который использует обобщенные типы, поскольку может получать информацию в нескольких форматах.

Метод браузера `send()` имеет два аргумента:

- объект `HttpRequest`;
- обработчик, который настраивает формат отклика.

Для создания обработчика следует вызвать метод внутреннего класса `HttpResponse.BodyHandler`, как показано в следующем примере.

```
HttpResponse<String> response = browser.send(request,
   HttpResponse.BodyHandler.asString());
```

Обобщенный тип в выражении `HttpResponse<String>` определяет ответ сервера как строку. Метод `asString()` обработчика заставляет метод браузера `send()` вернуть строку.

Обработчик также включает метод `asFile(`*путь*`)` для возврата отклика в виде файла и метод `asByteArray()` для возврата отклика в виде массива байтов (`byte[]`).

Каждый HTTP-запрос имеет заголовки, которые содержат дополнительные сведения об отклике и о сервере, который вернул ответ. Один из этих заголовков, `Server`, включает имя и номер версии программы, которая выполняет сервер. Некоторые серверы не указывают номер версии, руководствуясь соображениями безопасности.

Метод `headers()` отклика возвращает заголовки в виде объекта `Http Headers`. Вызовите метод `firstValue(`*строка*`)` этого объекта для получения первого заголовка, соответствующего указанному имени. Ниже представлен соответствующий код.

```
HttpHeaders headers = response.headers();
Optional<String> server = headers.firstValue("Server");
```

Объект, возвращаемый в результате вызова метода `firstValue()`, представляет собой структуру данных из пакета `java.util`. Эта структура упрощает обработку объектов, имеющих значение `null`. При обращении к пустому объекту генерируется исключение `NullPointerException`, которое является одним из наиболее распространенных исключений в Java.

Класс `Optional` предотвращает появление подобного исключения с помощью метода `isPresent()`, который возвращает значение `true` при наличии допустимого значения и `false` в противном случае.

Объект `server` содержит строку, на что указывает обобщенный тип в выражении `Optional<String>`. Эту строку можно безопасно отобразить с помощью следующего кода, позволяющего избежать проблем со значением `null`.

```
if (server.isPresent()) {
   System.out.println("Сервер: " + server.get());
}
```

Все эти методики применяются в приложении `ServerCheck`, которое запрашивает домашние страницы шести технологических компаний и сообщает о том, какие серверные программы используются в каждом случае.

Создайте в среде NetBeans новый класс `ServerCheck` и включите его в пакет `com.java24hours`. В редакторе исходного кода введите код из листинга 21.1 и сохраните полученный файл `ServerCheck.java`.

ЛИСТИНГ 21.1. Исходный код программы `ServerCheck.java`

```
 1: package com.java24hours;
 2:
 3: import java.io.*;
 4: import java.net.*;
```

```
 5: import java.util.*;
 6: import jdk.incubator.http.*;
 7:
 8: public class ServerCheck {
 9:    public ServerCheck() {
10:       String[] sites = {
11:          "https://www.apple.com",
12:          "https://www.microsoft.com",
13:          "http://www.compaq.com",
14:          "http://www.hp.com",
15:          "https://www.oracle.com/",
16:          "http://www.informit.com/"
17:       };
18:       try {
19:          load(sites);
20:       } catch (URISyntaxException oops) {
21:          System.out.println("Неподходящий URI: " +
22:                             oops.getMessage());
23:       } catch (IOException | InterruptedException oops) {
24:          System.out.println("Ошибка: " + oops.getMessage());
25:       }
26:    }
27:
28:    public void load(String[] sites) throws URISyntaxException,
29:                                IOException, InterruptedException {
30:
31:       for (String site : sites) {
32:          System.out.println("\nСайт: " + site);
33:          // Создание веб-клиента
34:          HttpClient browser = HttpClient.newHttpClient();
35:          // Создание запроса к сайту
36:          URI uri = new URI(site);
37:          HttpRequest.Builder bob = HttpRequest.newBuilder(uri);
38:          HttpRequest request = bob.build();
39:          // Отправка запроса
40:          HttpResponse<String> response = browser.send(request,
41:             HttpResponse.BodyHandler.asString());
42:          // Поиск заголовка сервера
43:          HttpHeaders headers = response.headers();
44:          Optional<String> server = headers.firstValue("Server");
45:          if (server.isPresent()) {
46:             System.out.println("Сервер: " + server.get());
47:          } else {
48:             System.out.println("Сервер не идентифицирован ");
49:          }
50:       }
51:    }
52:
53:    public static void main(String[] arguments) {
```

```
54:         new ServerCheck();
55:     }
56: }
```

Результат работы программы показан на рис. 21.1.

РИС. 21.1. Использование HTTP-клиента для запроса заголовков сервера

Подобно классам ввода-вывода, которые рассматривались на предыдущем занятии, методы HTTP-классов должны обрабатывать ошибки, возникающие при передаче данных. Эти методы генерируют исключения `IOException` и `InterruptedException` из пакета `java.io`.

Приложение `ServerCheck` содержит метод `load()`, который выполняет все действия, требуемые для получения шести домашних страниц от различных веб-серверов.

Три возможных исключения, которые могут возникнуть в данном случае, помещаются в блок `try-catch` конструктора. В инструкции `catch` в строке 23 указываются несколько классов исключений.

Сохранение файла, полученного из Интернета

Данные, которые могут передаваться по протоколу HTTP, не ограничиваются текстовыми файлами, такими как веб-страницы. Это могут быть любые данные, представленные байтами, в том числе изображения, видеоклипы и исполняемые файлы.

В следующем проекте, рассматриваемом на этом занятии, с помощью библиотеки HTTP Client из блога автора загружается изображение, которое затем сохраняется на локальном компьютере.

Приложение `ImageDownloader` выполняет те же действия, что и предыдущее приложение, вплоть до момента отправки запроса. Сначала создается объект браузера, после чего на основе URI создается построитель запросов и генерируется сам запрос.

Прежде чем отослать запрос веб-серверу, нужно создать файл, в котором будет храниться содержимое полученного изображения.

```
Path temp = Files.createTempFile("lighthouse", ".jpg");
```

Класс `Files` из пакета `java.nio` позволяет создать временный файл путем вызова метода `createTempFile(`*`строка`*`, `*`строка`*`)`. Аргументы представляют собой текстовый идентификатор, используемый в имени файла, и расширение файла. Генерируемое имя файла содержит идентификатор, за которым следует число и расширение, например `lighthouse3994062538481620758.jpg`.

После создания файла его можно указать в качестве аргумента обработчика ответов в методе `send()` браузера. В этом файле будет храниться отклик, полученный от сервера. Соответствующая инструкция выглядит так:

```
HttpResponse<Path> response = browser.send(request,
   HttpResponse.BodyHandler.asFile(temp));
```

Запрос представляет собой объект `HttpRequest`, созданный с помощью построителя на основе веб-адреса изображения (URI). Метод тела обработчика `asFile(`*`путь`*`)` назначает указанный файл в качестве целевого для хранения данных изображения.

Этот временный файл можно сохранить в виде постоянного файла, переименовав его.

```
File perm = new File("lighthouse.jpg");
temp.toFile().renameTo(perm);
```

Откройте среду NetBeans, создайте новый класс `ImageDownloader` и включите его в пакет `com.java24hours`. Введите код из листинга 21.2 и сохраните полученный файл `ImageDownloader.java`.

ЛИСТИНГ 21.2. Исходный код программы `ImageDownloader.java`

```
 1: package com.java24hours;
 2:
 3: import java.io.*;
 4: import java.net.*;
 5: import java.nio.file.*;
```

```
 6: import jdk.incubator.http.*;
 7:
 8: public class ImageDownloader {
 9:    public ImageDownloader() {
10:       String uri = "http://workbench.cadenhead.org/
11:                     media/lighthouse.jpg";
12:       try {
13:          load(uri);
14:       } catch (URISyntaxException oops) {
15:          System.out.println("Неподходящий URI: " +
16:                             oops.getMessage());
17:       } catch (IOException | InterruptedException oops) {
18:          System.out.println("Ошибка: " + oops.getMessage());
19:       }
20:    }
21:
22:    public void load(String imageUri) throws URISyntaxException,
23:                              IOException, InterruptedException {
24:
25:       // Создание веб-клиента
26:       HttpClient browser = HttpClient.newHttpClient();
27:       // Создание запроса к изображению
28:       URI uri = new URI(imageUri);
29:       HttpRequest.Builder bob = HttpRequest.newBuilder(uri);
30:       HttpRequest request = bob.build();
31:       // Создание файла для хранения данных изображения
32:       Path temp = Files.createTempFile("lighthouse", ".jpg");
33:       // Выполнение запроса и получение данных
34:       HttpResponse<Path> response = browser.send(request,
35:          HttpResponse.BodyHandler.asFile(temp));
36:       System.out.println("Изображение сохранено в " +
37:                          temp.toFile().getAbsolutePath());
38:       // Сохранение файла на постоянной основе
39:       File perm = new File("lighthouse.jpg");
40:       temp.toFile().renameTo(perm);
41:       System.out.println("Изображение перемещено в " +
42:                          perm.getAbsolutePath());
43:    }
44:
45:    public static void main(String[] arguments) {
46:       new ImageDownloader();
47:    }
48: }
```

После запуска приложения `ImageDownloader` файл `lighthouse.jpg` появляется в главной папке проекта. Выберите вкладку **Файлы**, чтобы вывести панель **Файлы** (она является такой же частью NetBeans, как и панель **Проекты**),

а затем дважды щелкните на файле, чтобы отобразить его на основной панели (рис. 21.2).

Программа выводит сведения о местоположении временного и постоянного файлов изображения (в строках 36–37 и 41–42 соответственно).

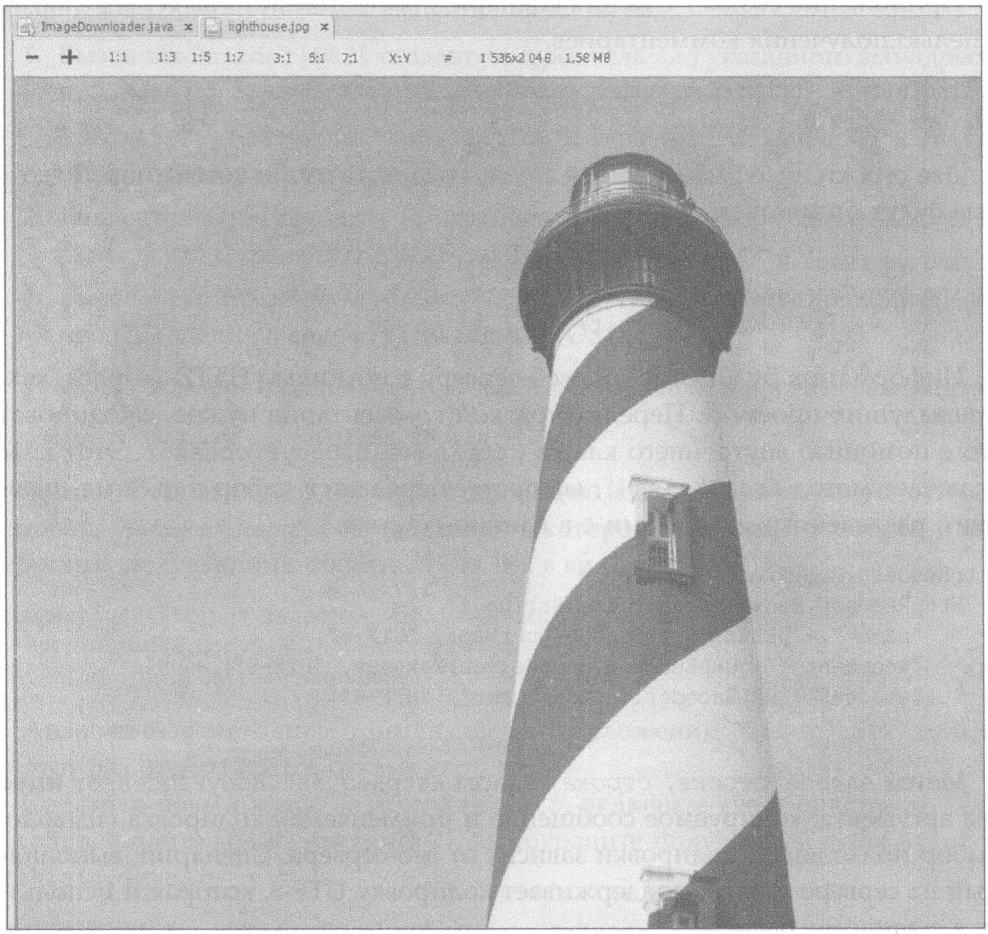


РИС. 21.2. Загрузка файла изображения из Интернета

Публикация данных в Интернете

До сих пор на занятии рассматривалось использование HTTP для получения данных из Интернета. Теперь пришло время научиться отправлять данные. Заключительный проект, рассматриваемый на этом занятии, — `SalutonVerkisto`. Это приложение Java предназначено для отправки сообщения веб-серверу автора с использованием запроса POST.

Запрос POST позволяет кодировать большие объемы данных для передачи на сервер, например публикации в блоге и даже файлы изображений и видео. Запрос GET тоже позволяет передавать информацию серверу, но он ограничен тем, что может включаться в строку URI.

Приложение создает URI для сценария, выполняемого на сервере автора с целью получения комментариев.

```
String site = "http://workbench.cadenhead.org/post-a-comment.php";
URI uri = new URI(site);
```

Две строки, `yourName` и `yourMessage`, содержат имя и комментарий, которые будут отсылаться.

```
String yourName = "Sam Snett of Indianapolis";
String yourMessage = "Your book is pretty good, if I do say
                      so myself.";
```

Информация будет отослана веб-серверу с помощью HTTP-запроса, как в предыдущих проектах. Перед отправкой комментария нужно закодировать его с помощью внутреннего класса `HttpRequest.BodyProcessor`. Этот класс включает метод `fromString()`, которому передается набор пар "имя–значение", разделенных символами `&` в длинной строке.

```
HttpRequest.BodyProcessor proc =
   HttpRequest.BodyProcessor.fromString (
      "name=" + URLEncoder.encode(yourName, "UTF-8") +
      "&comment=" + URLEncoder.encode(yourMessage, "UTF-8") +
      "&mode=" + URLEncoder.encode("demo", "UTF-8")
);
```

Метод `encode(`*строка*`, `*строка*`)` класса `HttpRequest.BodyProcessor` имеет два аргумента: кодируемое сообщение и применяемая кодировка символов. Выбор подходящей кодировки зависит от веб-сервера. Сценарий, выполняемый на сервере автора, поддерживает кодировку UTF-8, которая и используется в данном случае.

Рассмотренный вызов метода определяет три пары "имя–значение": `name`, `comment` и `mode`. Первые две пары принимают значения из переменных `yourName` и `yourMessage`. Третья пара, `mode`, имеет значение `demo`, что информирует серверный сценарий о назначении сообщения. (Это также отсеивает спамеров, которые могут отправлять нежелательные сообщения сценарию.)

Обработанное сообщение `proc` можно использовать для создания построителя запросов с помощью метода `newBuilder(`*URI*`)`, после которого делаются еще три вызова. Приведем соответствующий код.

```
HttpRequest.Builder newBuilder = HttpRequest.newBuilder(uri)
   .header("Content-Type", "application/x-www-form-urlencoded")
```

```
    .header("Accept", "text/plain")
    .POST(proc);
```

Здесь в одной инструкции объединены четыре вызова. Такое объединение становится возможным, поскольку каждый вызов возвращает объект построителя. Ниже описан порядок выполнения вызовов.

1. Вызов `newBuilder(`*`URI`*`)` создает построитель для указанного веб-адреса.
2. Вызов `header(`*`строка`*`, `*`строка`*`)` присваивает заголовку запроса `Content-Type` значение `"application/x-www-form-urlencoded"`. Это сообщает серверу о том, что передается веб-форма.
3. Еще один вызов `header()` присваивает атрибуту `Accept` значение `"text/plain"`, что определяет MIME-тип запроса.
4. Вызов `post(`*`HttpRequest.BodyProcessor`*`)` форматирует закодированное сообщение в виде HTTP-запроса POST.

Теперь, когда построитель подготовлен, можно сформировать сам запрос.

```
HttpRequest request = newBuilder.build();
```

Запрос отправляется серверу с помощью метода браузера `send(`*`HttpRequest`*`, `*`HttpResponse.BodyHandler`*`)`, в котором указываются запрос и обработчик, получающий обратный отклик в виде строки.

```
HttpResponse<String> response = client.send(request,
   HttpResponse.BodyHandler.asString());
System.out.println(response.body());
```

Аналогичная методика применялась в приложении `ServerCheck`, только на этот раз отображается отклик сервера.

Создайте новый класс `SalutonVerkisto`, включите его в пакет `com.java24hours`, введите код из листинга 21.3 и сохраните полученный файл `SalutonVerkisto.java`.

ЛИСТИНГ 21.3. Исходный код программы `SalutonVerkisto.java`

```
 1: package com.java24hours;
 2:
 3: import java.io.*;
 4: import java.net.*;
 5: import jdk.incubator.http.*;
 6:
 7: public class SalutonVerkisto {
 8:
 9:    public SalutonVerkisto() {
10:       String site = "http://workbench.cadenhead.org/
11:                     post-a-comment.php";
```

```
12:       try {
13:          postMessage(site);
14:       } catch (URISyntaxException oops) {
15:          System.out.println("Неподходящий URI: " +
16:                             oops.getMessage());
17:       } catch (IOException | InterruptedException oops) {
18:          System.out.println("Ошибка: " + oops.getMessage());
19:       }
20:    }
21:
22:    public void postMessage(String server) throws IOException,
23:       URISyntaxException, InterruptedException {
24:
25:       HttpClient client = HttpClient.newHttpClient();
26:
27:       // Адрес сервера
28:       URI uri = new URI(server);
29:
30:       // Настройка сообщения
31:       String yourName = "Sam Snett of Indianapolis";
32:       String yourMessage = "Your book is pretty good,
33:                             if I do say so myself.";
34:
35:       // Кодировка сообщения
36:       HttpRequest.BodyProcessor proc =
37           HttpRequest.BodyProcessor.fromString(
38:          "name=" + URLEncoder.encode(yourName, "UTF-8") +
39:          "&comment=" + URLEncoder.encode(yourMessage, "UTF-8") +
40:          "&mode=" + URLEncoder.encode("demo", "UTF-8")
41:       );
42:
43:       // Подготовка запроса
44:       HttpRequest.Builder newBuilder = HttpRequest.newBuilder(uri)
45:       .header("Content-Type", "application/x-www-form-urlencoded")
46:       .header("Accept", "text/plain")
47:       .POST(proc);
48:
49:       // Завершение построения запроса
50:       HttpRequest request = newBuilder.build();
51:
52:       // Получение ответа от сервера
53:       System.out.println("Метод: " + request.method() + "\n");
54:       HttpResponse<String> response = client.send(request,
55:          HttpResponse.BodyHandler.asString());
56:       System.out.println(response.body());
57:    }
58:
59:    public static void main(String[] arguments) {
```

```
60:         new SalutonVerkisto();
61:     }
62: }
```

Перед запуском приложения `SalutonVerkisto` измените строки 31–33: присвойте переменной `yourName` ваше имя и адрес, а переменной `yourMessage` — ваше сообщение. Вся эта информация будет доступна всем желающим в блоге автора книги.

Приложение возвращает текст комментариев, которые были получены от читателей (рис. 21.3).

РИС. 21.3. Отправка данных веб-серверу с помощью запроса `POST`

Резюме

Использование HTTP-запросов программами и веб-службами было неизбежным. Этот протокол реализован повсеместно, а брандмауэры сконфигурированы так, чтобы пропускать трафик через порт 80, в противном случае веб-браузеры не смогут работать.

Новая библиотека HTTP Client, которая появилась в Java 9, была создана для упрощения работы с этим полезным двусторонним каналом передачи информации.

Возможно, у вас вызвало недоумение странное название последнего проекта этого занятия: `SalutonVerkisto`. Вспомните программу `Saluton`, которая была создана на занятии 2. Она отображала сообщение `"Saluton, mondo!"`, которое представляет собой аналог фразы “Здравствуй, мир!”, но на языке

эсперанто. Слово "saluton" переводится как "привет". Таким образом, фраза "Saluton verkisto" означает "Привет тебе, автор".

Вопросы и ответы

В. Почему при выполнении трех программ этого занятия отображается предупреждение об использовании инкубатора?

О. Разработчики Java внедрили концепцию инкубатора в Java 9. Новая библиотека HTTP Client пока находится на стадии инкубации, т.е. на ранней стадии разработки.

Проект, находящийся на стадии инкубации, скорее всего изменится с выходом следующей версии Java. Это обстоятельство следует учитывать при использовании классов из пакета `jdk.incubator.http`.

В. Каким был первый сайт в Интернете?

О. Первым сайтом Интернета был `http://info.cern.ch`, который до сих пор "в строю". Тим Бернерс-Ли, британский физик из Европейской организации по ядерным исследованиям (CERN), использовал этот сайт для описания своего нового проекта — Всемирной паутины (World Wide Web).

Первая веб-страница имела адрес `http://info.cern.ch/hypertext/WWW/TheProject.html` и ежедневно обновлялась по мере роста интереса со стороны пользователей, издателей и разработчиков программ.

На этом сайте Интернет был определен как "широкомасштабная инициатива по извлечению гипермедийной информации, направленная на предоставление универсального доступа к огромному пространству документов".

Коллоквиум

Проверьте знание материала, рассмотренного на этом занятии, ответив на следующие вопросы.

Контрольные вопросы

1. Какой тип запроса применяется для отправки данных веб-серверу?

   А. `GET`.

   Б. `POST`.

   В. `GET` и `POST`.

2. Какой класс хранит информацию, полученную от веб-сервера в ответ на запрос?

А. `HttpClient`.

Б. `HttpResponse`.

В. `HttpRequest`.

3. Какова одна из причин использования структуры данных `Optional`?

А. Чтобы избежать появления ошибок `NullPointerException`.

Б. Чтобы создать коллекцию значений `null`.

В. Чтобы ускорить выполнение программы.

Ответы

1. **В.** Метод `POST` позволяет передавать большие объемы данных. Метод `GET` передает данные в строке веб-адреса (URI-идентификатор).
2. **Б.** Объект `HttpResponse` возвращается в результате вызова метода браузера `send()`.
3. **А.** Метод `isPresent()` проверяет, содержит ли эта структура значение `null` вместо корректного объекта.

Упражнения

Выполните следующие упражнения, чтобы закрепить изученный материал.

- Добавьте пять дополнительных страниц технологических компаний в программу `ServerCheck`.
- Напишите программу, которая использует заголовки HTTP-отклика "Content-Type" и "Content-Encoding" для отображения сведений о MIME-типе и символьной кодировке, поддерживаемых сервером.

ЗАНЯТИЕ 22
Создание двумерной графики

На этом занятии вы узнаете...

- как задать шрифт и цвет текста;
- как настроить фоновый цвет контейнера;
- как рисовать линии, прямоугольники и другие фигуры;
- как создавать графику GIF и JPEG;
- как рисовать окрашенные и неокрашенные фигуры.

На этом занятии вы узнаете о том, как превращать контейнеры — невыразительные серые панели и фреймы, содержащие компоненты графического интерфейса, — в художественный холст, на котором можно рисовать шрифты, цвета, фигуры и графику.

Использование класса `Font`

Цвета и шрифты представлены в Java классами `Color` и `Font`, находящимися в пакете `java.awt`. Благодаря этим классам можно форматировать текст с помощью различных шрифтов, а также изменять цвет текста и графики. Шрифты создаются с помощью конструктора `Font(`*`String`*`, `*`int`*`, `*`int`*`)`, который имеет три аргумента.

- Гарнитура шрифта, которая представлена либо обобщенным именем ("Dialog", "DialogInput", "Monospaced", "SanSerif" или "Serif"), либо фактическим именем шрифта ("Arial Black", "Helvetica", или "Courier New").
- Стиль, представленный одной из трех констант класса: `Font.BOLD`, `Font.ITALIC` или `Font.PLAIN`.
- Размер шрифта (кегль), выраженный в пунктах.

Следующая инструкция создает курсивный шрифт с засечками (Serif), размер которого составляет 12 пунктов:

```
Font current = new Font("Serif", Font.ITALIC, 12);
```

Если применяется конкретный шрифт, а не один из обобщенных, этот шрифт должен быть предварительно установлен на компьютере пользователя, выполняющего программу.

Стили шрифтов можно комбинировать, как показано ниже:

```
Font headline = new Font("Courier New", Font.BOLD + Font.ITALIC, 72);
```

При наличии шрифта можно вызвать метод `setFont(`*`шрифт`*`)` компонента `Graphics2D`, чтобы назначить его текущим шрифтом. Все последующие операции рисования будут использовать этот шрифт до тех пор, пока в качестве текущего не будет выбран следующий шрифт. Представленный ниже код создает объект шрифта "Comic Sans" и назначает его в качестве текущего перед рисованием текста.

```
public void paintComponent(Graphics comp) {
   Graphics2D comp2D = (Graphics2D) comp;
   Font font = new Font("Comic Sans", Font.BOLD, 15);
   comp2D.setFont(font);
   comp2D.drawString("Привет!", 5, 50);
}
```

В Java поддерживается сглаживание, что позволяет отрисовывать более плавные шрифты, которые лучше выглядят при просмотре на экране. Чтобы активизировать сглаживание, следует настроить подсказку отображения с помощью метода `setRenderingHint(`*`int`*`, `*`int`*`)` объекта `Graphics2D`, который имеет два аргумента:

- ключ подсказки отображения;
- значение, связанное с этим ключом.

Эти значения представляют собой переменные класса `RenderingHints` из пакета `java.awt`. Чтобы включить сглаживание, вызовите метод `setRendering Hint()` с двумя аргументами.

```
comp2D.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
   RenderingHints.VALUE_ANTIALIAS_ON);
```

Объект `comp2D` в этом примере представляет собой компонент `Graphics2D`, формирующий среду рисования контейнера.

Использование класса Color

Цвета в Java представляются с помощью класса Color, который содержит следующие константы, являющиеся переменными класса: black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white и yellow. С помощью этих констант можно задать фоновый цвет компонента в контейнере. Для этого следует вызвать метод setBackground(*цвет*).

```
setBackground(Color.orange);
```

Как и текущий шрифт, текущий цвет нужно задать перед выполнением рисования. Для выбора текущего цвета предназначен метод setColor(*цвет*). В следующем фрагменте программы в качестве текущего цвета задается синий, после чего выводится синяя надпись.

```
public void paintComponent(Graphics comp) {
   Graphics2D comp2D = (Graphics2D) comp;
   comp2D.setColor(Color.blue);
   comp2D.drawString("Привет!", 5, 50);
}
```

В отличие от метода setBackground(), который можно вызывать непосредственно в контейнере, метод setColor() следует вызывать для объекта Graphics2D.

Создание пользовательских цветов

Для создания пользовательских цветов в Java нужно указать значения красной, зеленой и синей составляющей цвета в модели RGB (Red, Green, Blue). Пользовательский цвет образуется в результате смешения трех цветовых составляющих. Каждое значение изменяется от 0 (отсутствие цвета) до 255 (максимальная насыщенность).

Аргументы конструктора Color(*int*, *int*, *int*) соответствуют красной, зеленой и синей составляющей. Следующая программа рисует панель, на которой отображается золотистый текст (159 красного, 121 зеленого, 44 синего) на сине-зеленом фоне (0 красного, 101 зеленого, 118 синего).

```
import java.awt.*;
import javax.swing.*;

public class Jacksonville extends JPanel {
   Color gold = new Color(159, 121, 44);
   Color teal = new Color(0, 101, 118);

   public void paintComponent(Graphics comp) {
      Graphics2D comp2D = (Graphics2D) comp;
```

```
        comp2D.setColor(teal);
        comp2D.fillRect(0, 0, 200, 100);
        comp2D.setColor(gold);
        comp2D.drawString("Привет!", 5, 50);
    }
}
```

В этом примере вызывается метод `fillRect()` объекта `Graphics2D` для рисования прямоугольника с заливкой текущим цветом.

ПРИМЕЧАНИЕ

Цветовая модель RGB позволяет создавать 16,5 млн возможных комбинаций, но большинство компьютерных мониторов не способно отобразить все это богатство красок.

Рисование линий и фигур

В Java рисовать фигуры, такие как линии и прямоугольники, так же легко, как и отображать текст. Все, что вам нужно, — это объект `Graphics2D`, формирующий поверхность для рисования, и объекты фигур, которые рисуются на этой поверхности.

Объект `Graphics2D` содержит метод для рисования текста.

```
comp2D.drawString("Привет!", 15, 40);
```

Эта инструкция рисует текст `"Draw, pardner!"` в точке с координатами *(15, 40)*. Методы рисования используют ту же систему координат *(x, y)*, что и текст. Координата *(0, 0)* находится в левом верхнем углу контейнера, значения *x* увеличиваются слева направо, а значения *y* — сверху вниз. Чтобы определить максимальную координату *(x, y)*, которую можно использовать во фрейме или в другом контейнере, выполните следующие инструкции.

```
int maxXValue = getSize().width;
int maxYValue = getSize().height;
```

За исключением линий, рисуемые фигуры могут быть заполненными или незаполненными. Для заливки фигуры используется текущий цвет, он также применяется для рисования границы незаполненной фигуры.

Рисование линий

Двумерный контур объекта создается с помощью соответствующей нарисованной фигуры. Классы рисуемых фигур входят в пакет `java.awt.geom`.

Класс `Line2D.Float` создает линию, которая соединяет начальную и конечную точки, имеющие координаты *(x, y)*. Следующая инструкция создает линию, которая начинается в точке *(40, 200)* и заканчивается в точке *(70, 130)*:

```
Line2D.Float line = new Line2D.Float(40F, 200F, 70F, 130F);
```

Аргументы с суффиксом `F` являются значениями с плавающей точкой. Если эта буква отсутствует, Java трактует аргументы как целые числа.

ПРИМЕЧАНИЕ

В названии класса `Line2D.Float` используется точка, что отличает его от большинства классов, которые встречались нам ранее. Это связано с тем, что `Float` является статическим внутренним классом класса `Line2D` (см. занятие 16).

Для рисования всех фигур (за исключением линий) используется один из методов класса `Graphics2D`. Для рисования контуров применяется метод `draw()`, а для заливки фигур — метод `fill()`.

Следующая инструкция рисует объект `line`, созданный в предыдущем примере:

```
comp2D.draw(line);
```

Рисование прямоугольников

В Java можно создавать заполненные и незаполненные прямоугольники, которые могут иметь прямые или скругленные углы. Для этого предназначен конструктор `Rectangle2D.Float(`*`int`*`, `*`int`*`, `*`int`*`, `*`int`*`)`, имеющий следующие аргументы:

- координата *x* левого верхнего угла прямоугольника;
- координата *y* левого верхнего угла прямоугольника;
- ширина прямоугольника;
- высота прямоугольника.

Следующая инструкция создает незаполненный прямоугольник с прямыми углами:

```
Rectangle2D.Float box = new Rectangle2D.Float(245F, 65F, 20F, 10F);
```

В данном случае создается прямоугольник, у которого левый верхний угол находится в точке с координатами *(245, 65)*, ширина равна 20 пикселям, высота — 10 пикселям. Чтобы нарисовать этот прямоугольник в виде контура, воспользуйтесь следующей инструкцией:

```
comp2D.draw(box);
```

Чтобы нарисовать прямоугольник с заливкой, воспользуйтесь методом `fill()`:

```
comp2D.fill(box);
```

Для создания прямоугольников со скругленными углами предназначен класс `RoundRectangle2D.Float`. Конструктор этого класса имеет те же четыре аргумента, что и конструктор класса `Rectangle2D.Float`, плюс добавляются следующие два аргумента:

- количество пикселей в направлении *x* от угла прямоугольника;
- количество пикселей в направлении *y* от угла прямоугольника.

С помощью этих аргументов определяется точка, в которой начинается скругление угла.

Следующая инструкция создает скругленный прямоугольник.

```
RoundRectangle2D.Float ro = new RoundRectangle2D.Float(
   10F, 10F,
   100F, 80F,
   15F, 15F);
```

Левый верхний угол прямоугольника находится в точке с координатами *(10, 10)*. Третий и четвертый аргументы задают ширину и высоту прямоугольника. В данном случае ширина прямоугольника составляет 100 пикселей, высота — 80 пикселей.

Последние два аргумента метода указывают на то, что для всех четырех углов скругление должно иметь радиус 15 пикселей.

Рисование эллипсов и окружностей

Для рисования эллипсов и окружностей используется один и тот же класс, `Ellipse2D.Float`, конструктор которого имеет четыре аргумента:

- координата *x*;
- координата *y*;
- ширина;
- высота.

В действительности координаты *(x, y)* не задают центр эллипса или окружности, как можно было бы ожидать. Вместе со значениями ширины и высоты они определяют невидимый прямоугольник, в который вписан эллипс. Координаты *(x, y)* соответствуют левому верхнему углу этого прямоугольника. Если ширина и высота прямоугольника одинаковы, эллипс будет окружностью.

Следующая инструкция создает окружность, вписанную в прямоугольник, координаты левого верхнего угла которого *(245, 45)*, а ширина и высота равны 5 пикселям:

```
Ellipse2D.Float cir = new Ellipse2D.Float(245F, 45F, 5F, 5F);
```

Рисование дуг

Еще одна круглая фигура, которую можно нарисовать в Java, — дуга, представляющая собой часть эллипса или окружности. Дуги создаются с помощью класса `Arc2D.Float`, большинство аргументов которого уже рассматривалось ранее. Для рисования дуги указывается эллипс, часть эллипса, которая должна отображаться (в градусах), и место на эллипсе, где должна начинаться дуга.

Чтобы создать дугу, задайте следующие целочисленные аргументы конструктора:

- координата *x* невидимого прямоугольника, в который вписан эллипс;
- координата *y* прямоугольника;
- ширина прямоугольника;
- высота прямоугольника;
- точка эллипса, в которой должна начинаться дуга (в градусах от 0 до 359);
- размер дуги (также в градусах);
- тип дуги.

Углы для определения начальной точки и размера дуги измеряются от 0 до 359 градусов в направлении против часовой стрелки. Точка 0 градусов соответствует 3 часам пополудни (рис. 22.1).

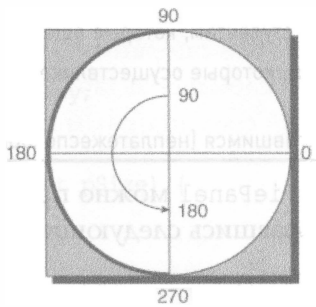


РИС. 22.1. Определение дуги в градусах

Тип дуги указывается с помощью переменных класса: `PIE` — сектор круговой диаграммы, `CLOSED` — конечные точки дуги соединены прямой линией, `OPEN` — конечные точки дуги не соединяются.

Следующая инструкция рисует открытую дугу, начиная с точки с координатами *(100, 50)*. Длина дуги составляет 120 градусов, начинается она в точке с угловой координатой 30 градусов, ширина составляет 65 градусов, а высота — 75 градусов.

```
Arc2D.Float smile = new Arc2D.Float(100F, 50F, 65F, 75F, 30F, 120F,
                                    Arc2D.Float.OPEN);
```

Рисование круговой диаграммы

А теперь создадим объект `PiePanel` — компонент графического интерфейса, который отображает круговую диаграмму. Этот компонент является подклассом класса `JPanel`, представляющего собой простой контейнер, в котором удобно рисовать различные объекты.

Программы, использующие класс `PiePanel`, должны выполнить следующие действия:

- создать объект `PiePanel` с помощью конструктора `PiePanel(int)`, где целочисленный аргумент задает количество срезов круговой диаграммы;
- вызвать метод `addSlice(Color, float)`, чтобы присвоить срезу указанные цвет и значение.

Значение среза в классе `PiePanel` определяет долю, которую он занимает на диаграмме.

Для примера в табл. 22.1 представлены данные о статусе погашения студенческих кредитов в США на протяжении первых 38 лет действия кредитной программы (согласно данным Министерства образования).

Таблица 22.1. Выплаты по студенческим кредитам в США

Сумма, выплаченная студентами	101 млрд долл.
Сумма, предоставленная будущим студентам, которые еще учатся в школе	68 млрд долл.
Сумма, предоставленная студентам, которые осуществляют платежи (платежеспособные студенты)	91 млрд долл.
Сумма, предоставленная обанкротившимся (неплатежеспособным) студентам	25 млрд долл.

С помощью компонента `PiePanel` можно представить эти данные на круговой диаграмме, воспользовавшись следующими инструкциями.

```
PiePanel loans = new PiePanel(4);
loans.addSlice(Color.green, 101F);
loans.addSlice(Color.yellow, 68F);
loans.addSlice(Color.blue, 91F);
loans.addSlice(Color.red, 25F);
```

На рис. 22.2 показан результат, отображенный во фрейме приложения, содержащего один компонент — `PiePanel`, созданный на основе данных о студенческих кредитах.

При создании объекта `PiePanel` количество срезов указывается в конструкторе. Чтобы нарисовать каждый срез, нужно знать следующую информацию:

- цвет среза, представленный объектом `Color`;
- доля, занимаемая срезом;
- общая сумма значений всех срезов.

РИС. 22.2. Отображение данных о студенческих кредитах на круговой диаграмме

Для представления каждого среза на круговой диаграмме используется новый вспомогательный класс `PieSlice`.

```
import java.awt.*;

class PieSlice {
   Color color = Color.lightGray;
   float size = 0;

   PieSlice(Color pColor, float pSize) {
      color = pColor;
      size = pSize;
   }
}
```

Для построения среза необходимо вызвать конструктор `PieSlice(`*`Color`*`, `*`float`*`)`. Сумма значений всех срезов хранится в закрытой переменной `totalSize` класса `PiePanel`. Существуют также переменные экземпляра для фонового цвета панели (`background`) и счетчика числа срезов (`current`).

```
private int current = 0;
private float totalSize = 0;
private Color background;
```

Теперь, когда в вашем распоряжении появился класс `PieSlice`, можно создать массив объектов `PieSlice` в виде переменной экземпляра.

```
private PieSlice[] slice;
```

При создании объекта `PiePanel` срезам не присваивается ни цвет, ни размер. Единственное, что нужно сделать в конструкторе, — определить размер массива `slice` и сохранить фоновый цвет панели.

```
public PiePanel(int sliceCount) {
   slice = new PieSlice[sliceCount];
   background = getBackground();
}
```

Используйте метод `addSlice(Color, float)` для добавления среза на панель.

```
public void addSlice(Color sColor, float sSize) {
   if (current <= slice.length) {
      slice[current] = new PieSlice(sColor, sSize);
      totalSize += sSize;
      current++;
   }
}
```

Для того чтобы поместить каждый срез в собственный элемент массива `slice`, мы используем переменную экземпляра `current`. Переменная `length` массива содержит количество элементов, которые будут храниться в массиве. Пока значение переменной `current` не превышает значение `slice.length`, можно продолжать добавлять срезы на панель.

Класс `PiePanel` выполняет все операции рисования с помощью метода `paintComponent()`. Наиболее сложная задача заключается в рисовании дуг, которые представляют каждый срез диаграммы. Для этого применяются следующие инструкции.

```
float start = 0;
for (int i = 0; i < slice.length; i++) {
   float extent = slice[i].size * 360F / totalSize;
   comp2D.setColor(slice[i].color);
   Arc2D.Float drawSlice = new Arc2D.Float(xInset, yInset, width,
      height, start, extent, Arc2D.Float.PIE);
   start += extent;
   comp2D.fill(drawSlice);
}
```

Переменная `start` отслеживает точку начала рисования дуги, а переменная `extent` — размер дуги. Если известен общий размер всех срезов диаграммы и размер отдельного среза, можно подсчитать значение переменной `extent`, умножив размер среза на 360 и разделив на общий размер срезов.

Все дуги рисуются в цикле `for`. После вычисления значения переменной `extent` строится сама дуга, а значение переменной `extent` добавляется к значению переменной `start`. Это приводит к тому, что каждый срез начинается

там, где заканчивается предыдущий. Для рисования дуги вызывается метод `fill()` объекта `Graphics2D`.

Теперь сведем все воедино. Создайте новый класс `PiePanel`, включите его в пакет `com.java24hours`, введите код из листинга 22.1 и сохраните полученный файл `PiePanel.java`.

ЛИСТИНГ 22.1. Исходный код программы `PiePanel.java`

```
 1: package com.java24hours;
 2:
 3: import java.awt.*;
 4: import javax.swing.*;
 5: import java.awt.geom.*;
 6:
 7: public class PiePanel extends JPanel {
 8:    private PieSlice[] slice;
 9:    private int current = 0;
10:    private float totalSize = 0;
11:    private Color background;
12:
13:    public PiePanel(int sliceCount) {
14:       slice = new PieSlice[sliceCount];
15:       background = getBackground();
16:    }
17:
18:    public void addSlice(Color sColor, float sSize) {
19:       if (current <= slice.length) {
20:          slice[current] = new PieSlice(sColor, sSize);
21:          totalSize += sSize;
22:          current++;
23:       }
24:    }
25:
26:    public void paintComponent(Graphics comp) {
27:       super.paintComponent(comp);
28:       Graphics2D comp2D = (Graphics2D) comp;
29:       int width = getSize().width - 10;
30:       int height = getSize().height - 15;
31:       int xInset = 5;
32:       int yInset = 5;
33:       if (width < 5) {
34:          xInset = width;
35:       }
36:       if (height < 5) {
37:          yInset = height;
38:       }
39:       comp2D.setColor(background);
40:       comp2D.fillRect(0, 0, getSize().width, getSize().height);
```

```
41          comp2D.setColor(Color.lightGray);
42:         Ellipse2D.Float pie = new Ellipse2D.Float(
43:            xInset, yInset, width, height);
44:         comp2D.fill(pie);
45:         float start = 0;
46:         for (int i = 0; i < slice.length; i++) {
47:            float extent = slice[i].size * 360F / totalSize;
48:            comp2D.setColor(slice[i].color);
49:            Arc2D.Float drawSlice = new Arc2D.Float(
50:               xInset, yInset, width, height, start, extent,
51:               Arc2D.Float.PIE);
52:            start += extent;
53:            comp2D.fill(drawSlice);
54:         }
55:      }
56: }
57:
58: class PieSlice {
59:    Color color = Color.lightGray;
60:    float size = 0;
61:
62:    PieSlice(Color pColor, float pSize) {
63:       color = pColor;
64:       size = pSize;
65:    }
66: }
```

В листинге 22.1 определяется класс `PiePanel` (строки 1–56) и вспомогательный класс `PieSlice` (строки 58–66). Класс `PiePanel` может использоваться в качестве компонента в графическом интерфейсе любой программы на Java. Чтобы протестировать класс `PiePanel`, нужно создать другой класс, использующий его.

В листинге 22.2 содержится код приложения `PieFrame`, использующего панели `PiePanel`. Создайте новый класс `PieFrame`, включите его в пакет `com.java24hours`, введите код из листинга 22.2 и сохраните полученный файл `PieFrame.java`.

ЛИСТИНГ 22.2. Исходный код программы `PieFrame.java`

```
1: package com.java24hours;
2:
3: import javax.swing.*;
4: import java.awt.*;
5:
6: public class PieFrame extends JFrame {
7:    Color uneasyBeingGreen = new Color(0xCC, 0xCC, 0x99);
8:    Color zuzusPetals = new Color(0xCC, 0x66, 0xFF);
```

```
 9:    Color zootSuit = new Color(0x66, 0x66, 0x99);
10:    Color sweetHomeAvocado = new Color(0x66, 0x99, 0x66);
11:    Color shrinkingViolet = new Color(0x66, 0x66, 0x99);
12:    Color miamiNice = new Color(0x33, 0xFF, 0xFF);
13:    Color inBetweenGreen = new Color(0x00, 0x99, 0x66);
14:    Color norwegianBlue = new Color(0x33, 0xCC, 0xCC);
15:    Color purpleRain = new Color(0x66, 0x33, 0x99);
16:    Color freckle = new Color(0x99, 0x66, 0x33);
17:
18:    public PieFrame() {
19:       super("Круговая диаграмма");
20:       setLookAndFeel();
21:       setSize(320, 290);
22:       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
23:       setVisible(true);
24:
25:       PiePanel pie = new PiePanel(10);
26:       pie.addSlice(uneasyBeingGreen, 1350);
27:       pie.addSlice(zuzusPetals, 1221);
28:       pie.addSlice(zootSuit, 316);
29:       pie.addSlice(sweetHomeAvocado, 251);
30:       pie.addSlice(shrinkingViolet, 201);
31:       pie.addSlice(miamiNice, 193);
32:       pie.addSlice(inBetweenGreen, 173);
33:       pie.addSlice(norwegianBlue, 164);
34:       pie.addSlice(purpleRain, 143);
35:       pie.addSlice(freckle, 127);
36:       add(pie);
37:    }
38:
39:    private void setLookAndFeel() {
40:       try {
41:          UIManager.setLookAndFeel(
42:             "com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel"
43:          );
44:       } catch (Exception exc) {
45:          // Игнорировать ошибки
46:       }
47:    }
48:
49:    public static void main(String[] arguments) {
50:       PieFrame pf = new PieFrame();
51:    }
52: }
```

Класс `PieFrame` представляет собой простой графический интерфейс, который содержит единственный компонент — объект `PiePanel`, создаваемый

в строке 25. В строках 26–35 десять раз вызывается метод объекта `addSlice()`, который добавляет срезы в круговую диаграмму.

После запуска приложения класс `PieFrame` выводит круговую диаграмму, отражающую численность населения в 10 наиболее населенных странах мира (в миллионах человек). При этом используются данные за июнь 2017 года, полученные из международной базы данных Бюро переписи населения США. Согласно этим данным, в Китае проживало 1,379 млрд, в Индии — 1,282 млрд, в США — 327 млн, в Индонезии — 261 млн, в Бразилии — 207 млн, в Пакистане — 205 млн, в Нигерии —191 млн, в Бангладеш — 158 млн, в России — 142 млн, в Японии — 126 млн.

Поскольку в Java определено лишь несколько цветов в классе `Color`, в программе создаются 10 новых цветов, которым присваиваются описательные названия. Цветовые составляющие выражаются шестнадцатеричными значениями, которые в Java предваряются префиксом `0x`. Впрочем, в конструкторе `Color()` можно указывать и десятичные значения.

На рис. 22.3 показан результат выполнения программы.

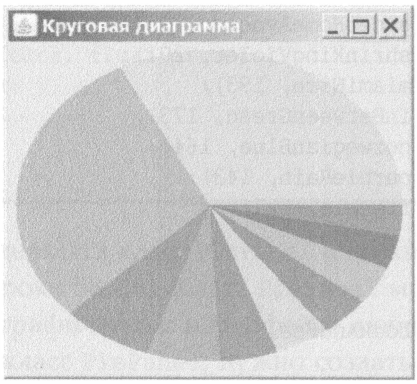


РИС. 22.3. Круговая диаграмма, отражающая численность населения в 10 наиболее населенных странах

Резюме

Благодаря использованию шрифтов, цветов и графики можно привлечь больше внимания к компонентам ваших программ и сделать их более привлекательными для пользователей.

Рисование фигур может показаться вам сложным занятием, хотя в сущности ничего сложного там нет. К тому же использование векторных фигур обеспечивает два преимущества по сравнению с растровой графикой, загружаемой из файлов изображений.

- **Быстродействие.** Даже небольшие растровые элементы, такие как пиктограммы, требуют больше времени для загрузки и отображения, чем набор векторных фигур.
- **Масштабирование.** Чтобы изменить размер всего изображения, использующего векторные фигуры, достаточно изменить значения, применяемые при создании фигур. Например, можно добавить функцию, которая умножает все точки *(x, y)* в каждой фигуре на 2 перед созданием фигур. В итоге изображение увеличится в два раза. Векторные изображения масштабируются гораздо быстрее, чем растровые, и результат выглядит лучше.

Вопросы и ответы

В. Как нарисовать дугу по часовой стрелке, а не против?

О. Для этого нужно указать размер дуги в виде отрицательного числа. В результате дуга будет начинаться в той же самой точке, но будет направлена в противоположном направлении по эллиптическому пути. Например, следующая инструкция рисует открытую дугу из точки с координатами *(35, 20)*. Дуга имеет ширину 15 пикселей, высоту 20 пикселей, длину 90 градусов, начинается на отметке 0 градусов и рисуется по часовой стрелке.

```
Arc2D.Float smile = new Arc2D.Float(35F, 20F, 15F, 20F,
   0F, -90F, Arc2D.Float.OPEN);
```

В. Эллипсы и окружности лишены углов. Что же определяют координаты *(x, y)* в конструкторе `Ellipses.Float`?

О. Они соответствуют наименьшим координатам *x* и *y* эллипса или окружности. Если вписать эллипс или окружность в невидимый прямоугольник, то его левый верхний угол будет находиться в точке с координатами *x* и *y*, заданными в качестве аргументов метода.

Коллоквиум

Проверьте знание материала, рассмотренного на этом занятии, ответив на следующие вопросы.

Контрольные вопросы

1. Какое из следующих значений не является константой, используемой для выбора цвета?

   А. `Color.cyan`.

   Б. `Color.purple`.

В. `Color.magenta`.

2. Если изменить цвет какого-либо компонента и перерисовать его в контейнере, то что нужно сделать, чтобы компонент стал видимым?
   А. Вызвать метод `drawColor()`.
   Б. Воспользоваться методом `repaint()`.
   В. Ничего не нужно делать.
3. Что означает аббревиатура RGB?
   А. Roy G. Biv.
   Б. Red, Green, Blue.
   В. Lucy in the Sky with Diamonds.

Ответы

1. **Б.** Фиолетовый цвет не представлен в классе `Color`.
2. **Б.** Вызов метода `repaint()` приводит к вызову метода `paintComponent()`.
3. **Б.** Остальные варианты — это не более чем аллегория.

Упражнения

Выполните следующие упражнения, чтобы закрепить изученный материал.

- Создайте версию класса `PieFrame`, которая получает значения цветов и размеры срезов в качестве аргументов командной строки (они не должны указываться в исходном коде приложения).
- Создайте приложение, которое рисует на панели знак "СТОП" с помощью цветов, фигур и шрифтов.

ЗАНЯТИЕ 23
Создание модов для Minecraft с помощью Java

На этом занятии вы узнаете...

- как установить сервер Minecraft на компьютере;
- как написать сценарий для запуска сервера;
- как настроить NetBeans для программирования модов;
- как создать мод, порождающий мобы;
- как "оседлать" одного моба другим;
- как найти всех мобов в игре и атаковать их;
- как создать мода-строителя.

На этом занятии мы рассмотрим явление, которое стало феноменом наших дней.

Мои сыновья-подростки являются заядлыми геймерами, обожая многопользовательскую игру Minecraft. Их двоюродные братья такого же возраста тоже играют в нее. И как только я говорю какому-нибудь юноше о том, что пишу книги по программированию на Java, все они спрашивают одно и то же: "А там рассказывается, как создавать моды Minecraft".

Моды — это расширения игры, которые разрабатываются и используются самими игроками. Minecraft написана на Java, поэтому моды должны создаваться на этом же языке.

Первый шаг на пути к созданию модов Minecraft — изучение языка Java, а второй шаг — освоение материала этого занятия.

После успешного завершения занятия на вашем компьютере будет загружен, установлен и запущен ваш собственный сервер Minecraft. Кроме того, вы научитесь создавать моды на языке Java, развертывать их на сервере и применять в игре.

Установка сервера Minecraft

Для создания модов требуется доступ к серверу Minecraft. Если вы только начинаете разрабатывать моды, то лучше всего установить и запустить сервер на своем собственном компьютере. Поскольку не существует стандартного подхода к добавлению модов в игру, разные серверы позволяют делать это по-разному.

Проще всего обратиться к проекту Spigot, представляющему собой сервер Minecraft и специализированную библиотеку классов Java. Spigot является бесплатным проектом и загружается с сайта `www.javaminecraft.com/spigot`. При этом вы получаете файл `spigotserver.jar`, который включает сервер Minecraft и библиотеку классов.

Spigot API — это набор пакетов Java, используемых при создании модов. После загрузки библиотеки создайте новую папку на компьютере и сохраните в ней загруженные файлы. На своем компьютере Windows я создал папку `c:\minecraft\server` и скопировал в нее файл `spigotserver.jar`, который применяется для запуска сервера Minecraft.

Сервер запускается такой командой:

```
java -Xms1024M -Xmx1024M -jar spigotserver.jar
```

Она указывает виртуальной машине Java на необходимость запуска приложения, упакованного в JAR-файл `spigotserver.jar`, и выделения этой программе 1024 Мбайт памяти.

Чтобы не вводить эту команду при каждом запуске сервера, создайте для нее .bat-файл и запускайте его. Для создания такого файла в Windows откройте текстовый редактор, например Блокнот, и введите код из листинга 23.1. Сохраните этот файл под именем `start-server.bat` в той же папке, где находится файл `spigotserver.jar`.

ЛИСТИНГ 23.1. Исходный код файла `start-server.bat`

```
1: java -Xms1024M -Xmx1024M -jar spigotserver.jar
2: pause
```

Дважды щелкните на значке этого файла, чтобы запустить сервер. Если сервер успешно запустится, откроется окно, в котором отображаются действия запущенного сервера. Но, скорее всего, вы увидите сообщение об ошибке `"Failed to load eula.txt"` (рис. 23.1).

Прежде чем запустить сервер Minecraft, нужно принять соглашение с конечным пользователем (EULA). Чтобы просмотреть текст этого соглашения, посетите следующую веб-страницу:

```
https://account.mojang.com/documents/minecraft_eula
```

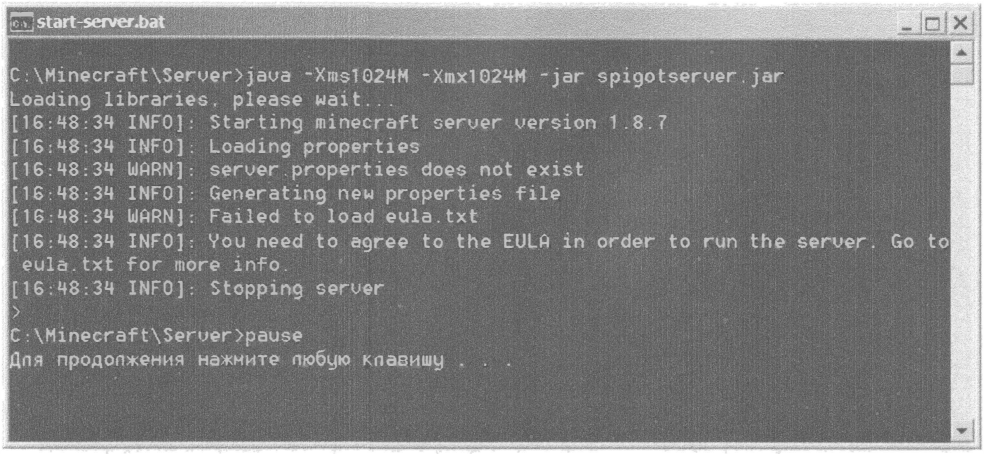


РИС. 23.1. Сервер не прошел проверку EULA

Обратите внимание на файл `eula.txt`, который находится в той же папке, что и сервер. В этом файле есть строка следующего вида:

```
eula=false
```

Если вы согласны с EULA, откройте этот файл в редакторе, измените указанную строку на `eula=true`, сохраните измененный файл и снова запустите сервер.

При первом запуске сервера создается несколько десятков файлов и подпапок и строится карта игрового мира. После успешного завершения этих действий появится завершающее сообщение `"[Info] Done"`, а в командной строке отобразится мигающий курсор рядом с приглашением >.

Команда `help` выводит список команд, которые применяются для управления сервером и игровым миром. Команда `stop` завершает выполнение сервера. (Не делайте пока что этого!)

На рис. 23.2 показан выполняющийся сервер Spigot. Это окно должно оставаться открытым до тех пор, пока работает сервер. Если ваше окно сервера выглядит подобным образом, можете переходить к разделу "Подключение к серверу".

Устранение проблем при работе сервера

После загрузки и установки сервера могут иметь место две распространенные ошибки, препятствующие его выполнению. В первом случае (рис. 23.3) появляется сообщение `"Unable to access jarfile spigotserver.jar"` (Невозможно получить доступ к JAR-файлу spigotserver.jar).

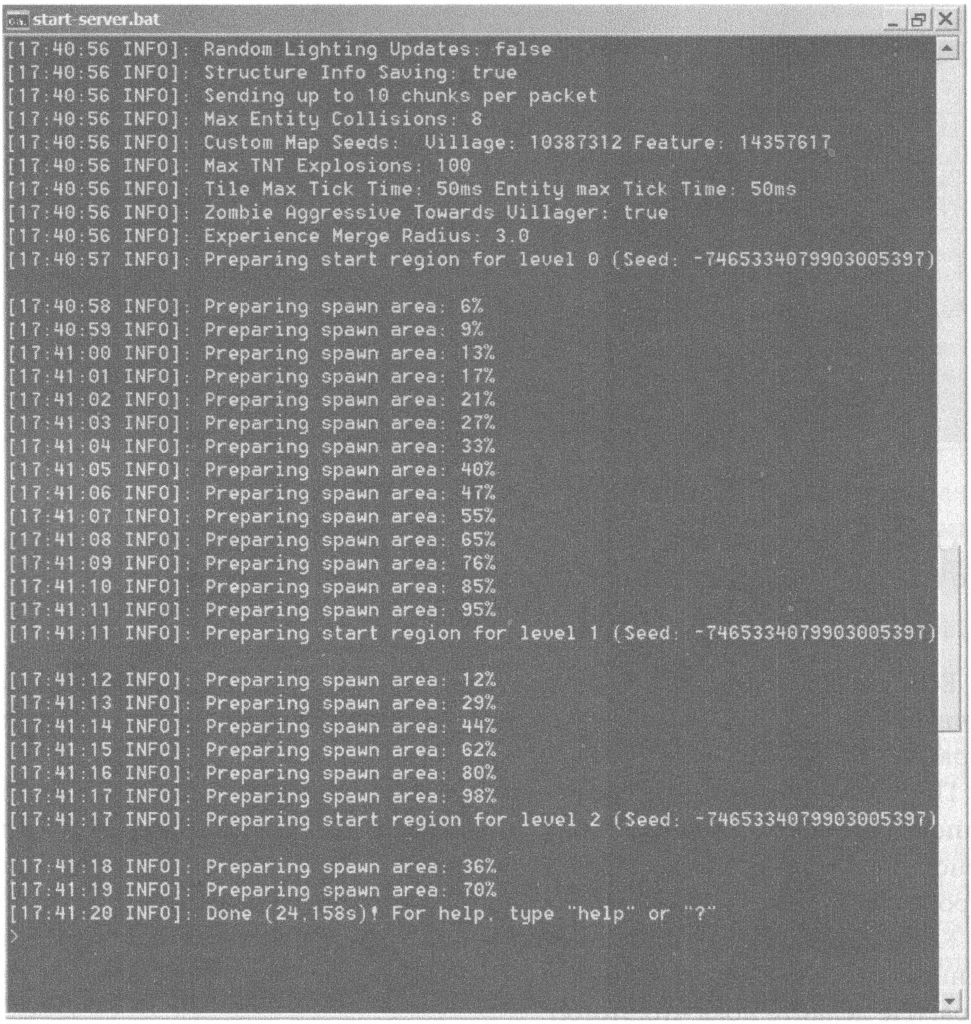


РИС. 23.2. Первый запуск сервера Minecraft

Чтобы устранить эту проблему, убедитесь в том, что файл `spigotserver.jar` находится в той же папке, что и сценарий запуска сервера (`start-server.bat`). Затем попытайтесь запустить сервер снова.

Во втором случае (рис. 23.4) отображается сообщение `"'java' is not recognized as an internal or external command"` ('java' не распознана в качестве внутренней или внешней команды).

Виртуальная машина Java находится в файле `java.exe`, который содержится в подпапке `bin` папки Java. Если команда `java` не распознана, это означает, что компьютер не может найти файл `java.exe`.

Чтобы устранить эту проблему в Windows, нужно добавить имя папки, в которой находится файл `java.exe`, в переменную среды PATH.

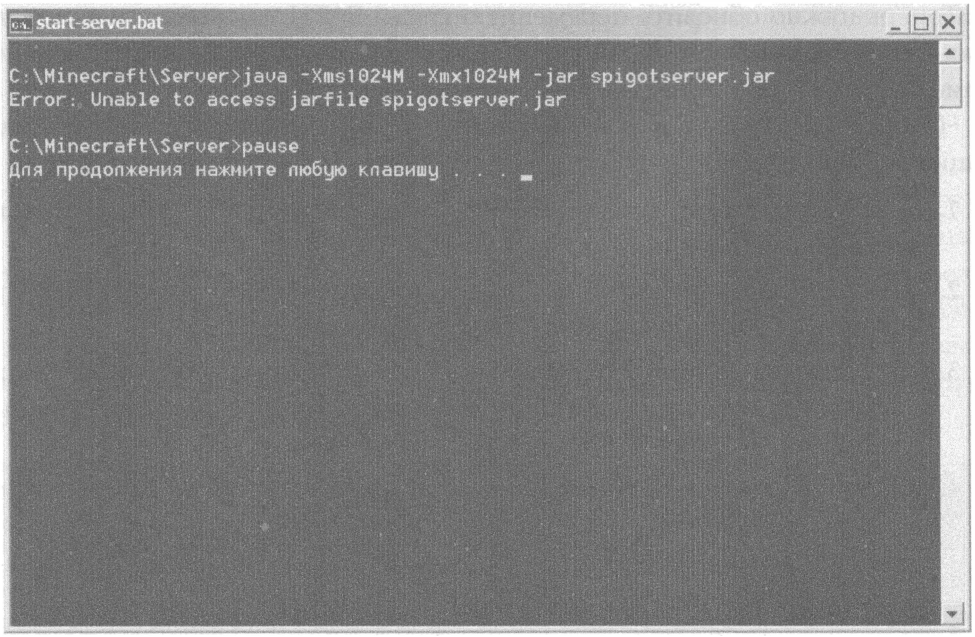


РИС. 23.3. JAR-файл Spigot не найден

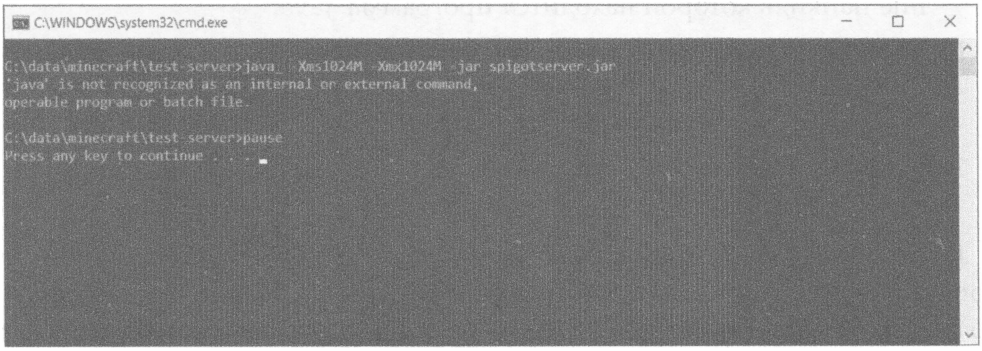


РИС. 23.4. Виртуальная машина Java (JVM) не найдена

Сначала найдите папку Java. Перейдите в папку верхнего уровня на основном жестком диске, после чего откройте папку `Program Files` или `Program Files (x86)` и проверьте, нет ли в ней подпапки `Java`. Найдя эту подпапку, перейдите в нее.

В папке `Java` могут находиться несколько разных версий JDK (Java Development Kit), каждая из которых имеет свой номер версии. Например, это могут быть папки `jdk1.8.0`, `jdk-9` или `jdk-10`. Больший номер означает более современную версию JDK. Откройте эту папку, а затем находящуюся в ней подпапку `bin`, в которой находятся несколько десятков приложений, включая `java`.

Теперь нужно обновить переменную среды PATH, включив в нее полное имя папки, например `C:\Program Files\jdk-10\bin` (используйте текущий номер JDK).

Чтобы добавить путь к JVM в переменную среды PATH, выполните следующие действия.

1. Откройте панель управления через меню Пуск либо введите **панель управления** в поле поиска.
2. Выберите элемент Система⇨Дополнительные параметры системы, чтобы открыть диалоговое окно Свойства системы.
3. Щелкните на кнопке Переменные среды. В открывшемся диалоговом окне будут перечислены пользовательские и системные переменные среды.
4. Прокручивайте панель Системные переменные вниз до тех пор, пока не найдете переменную `Path`. Выберите эту переменную и щелкните на кнопке Изменить в нижней части панели. После этого откроется диалоговое окно Изменение системной переменной.
5. Найдите поле Значение переменной и поместите курсор в конце текста, находящегося в этом поле. Добавьте точку с запятой, а затем название папки, в которой находится программа `java`.
6. Если поле Значение переменной не отображается, щелкните на кнопке Создать. В результате курсор переместится в поле под списком папок. Введите название папки, в которой находится программа `java`.
7. Последовательно щелкните на кнопках OK, чтобы закрыть каждое из трех диалоговых окон, а затем закройте панель управления.

Попытайтесь снова запустить сервер. Если он по-прежнему не работает, попробуйте устранить проблему, переустановив JDK. Посетите сайт Oracle по адресу `http://jdk.java.net/10/` и перейдите в раздел Downloads (Загрузки), в котором находятся ссылки, позволяющие загрузить JDK для разных операционных систем.

После переустановки JDK перезагрузите компьютер и снова запустите сервер Minecraft.

Подключение к серверу

Поскольку вы собрались создавать моды Minecraft, наверняка на вашем компьютере уже установлен клиент Minecraft, который используется для игры. Если же клиент не установлен, купите и загрузите игру на сайте `www.minecraft.net`. Имеются версии игры для Windows, Mac OS, игровых

приставок и мобильных устройств, но для программирования модов подходит лишь версия для Windows или Mac.

Запустите Minecraft и щелкните на кнопке **Сетевая игра**. Ваш новый сервер Minecraft должен появиться в списке доступных опций.

Сервер называется "Сервер Minecraft" (это название в дальнейшем можно изменить). Клиент отсылает ping-сообщение серверу и получает отчет о скорости соединения, который отображается в виде набора вертикальных зеленых полосок (рис. 23.5), означающих готовность к подключению. Если они появляются на экране, выберите сервер щелчком на соответствующем значке и щелкните на кнопке **Подключиться**. У вас должен появиться новый мир Minecraft, поддерживаемый вашим собственным сервером. Переходите к разделу "Создание первого мода".

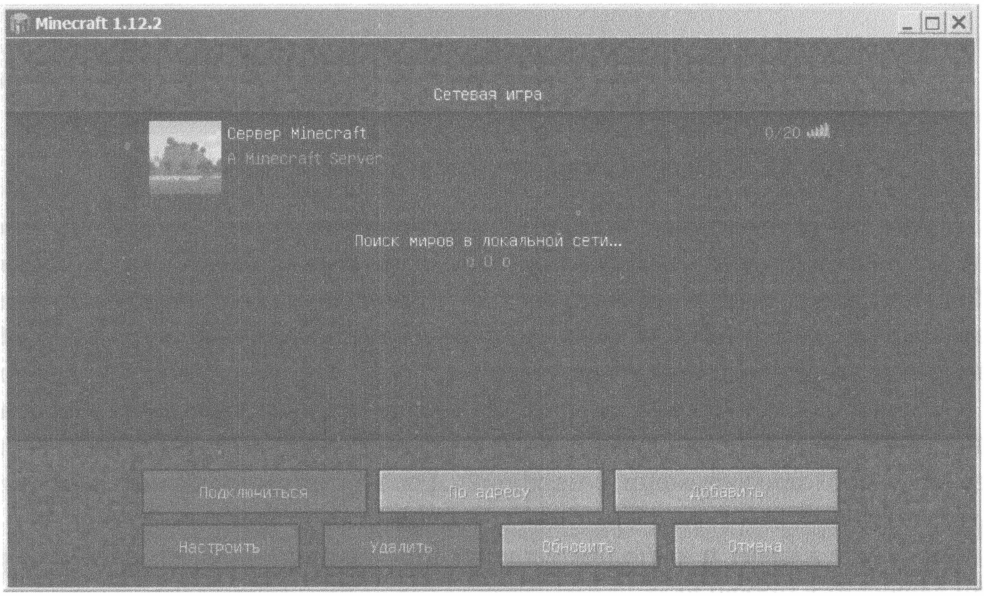


РИС. 23.5. Ваш новый сервер Minecraft готов к подключению

Если клиент не может подключиться к серверу, вместо зеленых полосок подключения появится красный значок ×. В следующем разделе будет описано, как устранить проблемы, возникающие при подключении к серверу.

Устранение проблем с подключением к серверу

Иногда возникают проблемы со связью между клиентом и сервером Minecraft. В таком случае вместо зеленых полосок появляется красный значок × (рис. 23.6).

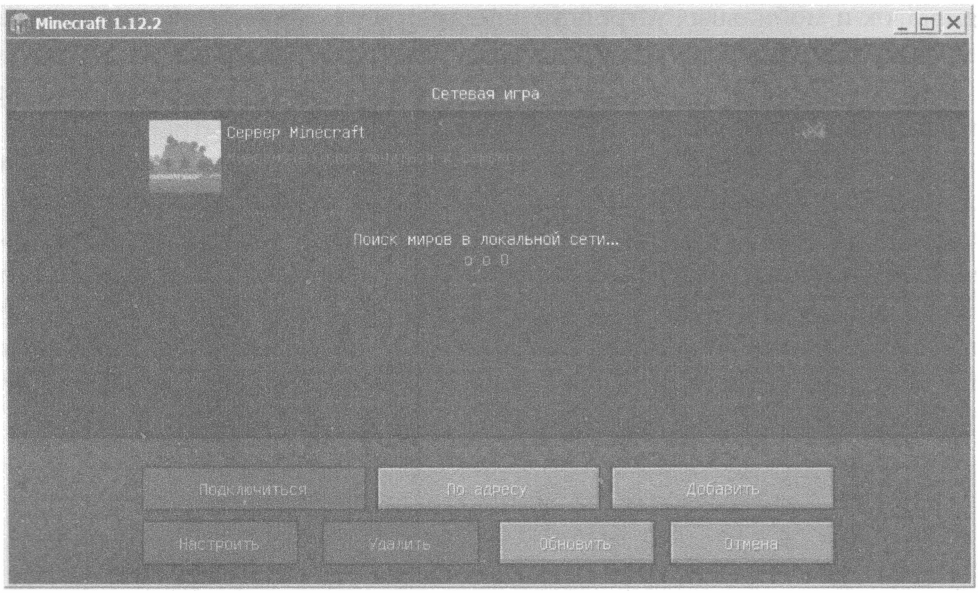


РИС. 23.6. Клиент Minecraft не может подключиться к серверу

Наиболее распространенная причина возникновения подобной проблемы заключается в том, что на сервере и клиенте выполняются разные версии Minecraft. Чтобы узнать версию Minecraft, выполняемую на сервере, обратите внимание на первое сообщение в его окне. Например, это может быть сообщение `"Starting Minecraft version 1.8.7"`.

Обычно в клиентской системе установлена самая последняя версия Minecraft (как в данном случае — 1.12.2). Чтобы использовать более старую версию, необходимо изменить профиль. Для этого нужно сначала выйти из клиента Minecraft, а затем запустить его повторно.

Запустите клиент и выполните следующие действия для изменения версии Minecraft в профиле.

1. В диалоговом окне запуска Minecraft щелкните на кнопке Launch Options (Параметры запуска), а затем — на кнопке Add New (Добавить новый).
2. В раскрывающемся списке Version (Версия) выберите пункт Release 1.8.7 (Выпуск 1.8.7).
3. Щелкните на кнопке Save (Сохранить).
4. Щелкните на кнопке Settings (Настройки).
5. Щелкните на стрелочке зеленой кнопки Воспроизведение и выберите пункт меню Unnamed Configuration 1.8.7 (Конфигурация без имени 1.8.7).

Щелкните на кнопке Воспроизведение и выберите пункт Сетевая игра. На экране появится сервер с зелеными вертикальными полосками, которые свидетельствуют об успешном подключении.

ПРЕДУПРЕЖДЕНИЕ

Изменение профиля игрока оказывает влияние на все используемые серверы Minecraft, включая те, которые используются для игры, а не для создания и тестирования модов. Не забудьте вернуться к текущей версии, когда отключитесь от собственного сервера Spigot (или создайте второй профиль игрока, что позволит использовать один мод для работы, а второй — для игры).

Создание первого мода

Теперь, когда установлен и выполняется сервер Spigot для Minecraft, вы готовы приступить к разработке модов для него.

Моды упаковываются в виде архивных файлов Java, которые также называются файлами JAR. В NetBeans, интегрированной среде разработки от Oracle, распространяемой на бесплатной основе, файлы JAR генерируются автоматически при каждом создании проекта. Готовый мод можно добавить в подпапку `plugins` папки сервера.

Первый мод, создаваемый на этом занятии, служит демонстрацией типичного каркаса, применяемого в каждом моде Spigot. Этот мод добавляет в игру команду `/siamesecat`, которая создает моба сиамской кошки, добавляет его в мир и делает вас (игрока) хозяином этой кошки.

Каждому моду соответствует собственный проект в NetBeans. Чтобы приступить к созданию проекта, выполните следующие действия.

1. В среде NetBeans выполните команду Файл⇨Создать проект. На экране появится окно мастера Создать проект.
2. На панели Категории выберите пункт Java, а на панели Проекты — пункт Приложение Java и щелкните на кнопке Далее.
3. В поле Имя проекта введите **`SiameseCat`**.
4. Щелкните на кнопке Просмотр справа от поля Расположение проекта, чтобы открыть диалоговое окно Выберите расположение проекта.
5. Найдите и выберите папку, в которой будет установлен сервер Minecraft, и щелкните на кнопке Открыть. Название выбранной папки появится в поле Расположение проекта.
6. Сбросьте флажок Создать главный класс.
7. Щелкните на кнопке Готово.

По завершении создания проекта SiameseCat на панели Проекты появятся две новые папки: Пакеты исходных кодов и Библиотеки (рис. 23.7).

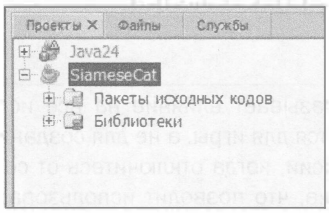


РИС. 23.7. Добавление библиотек в проект Java

1. Прежде чем создавать класс Java, для каждого проекта мода нужно добавить библиотеку классов Java. На панели Проекты щелкните правой кнопкой на папке Библиотеки и выберите команду Добавить библиотеку. Откроется диалоговое окно Добавить библиотеку.
2. Щелкните на кнопке Создать, после чего появится диалоговое окно Создание новой библиотеки.
3. В поле Имя библиотеки введите **Spigot** и щелкните на кнопке OK. На экране появится диалоговое окно Настроить библиотеку.
4. Щелкните на кнопке Добавить файл JAR/папку.
5. В открывшемся диалоговом окне Выбор архива JAR или папки найдите и откройте папку с соответствующим архивом JAR.
6. Щелкните на файле spigotserver.jar, чтобы выбрать его.
7. Щелкните на кнопке Добавить файл JAR/папку.
8. Щелкните на кнопке OK.
9. На панели Доступные библиотеки появится значок Spigot. Выберите его и щелкните на кнопке Добавить библиотеку.

В папке Библиотеки на панели Проекты теперь находится JAR-файл для сервера Spigot (чтобы увидеть этот файл, щелкните на значке + рядом с названием папки). Теперь можно приступать к созданию мода.

Чтобы создать мод, выполните следующие действия.

1. Выполните команду Файл⇨Создать файл, чтобы открыть диалоговое окно Создать файл.
2. На панели Категории выберите пункт Java.
3. На панели Типы файлов выберите пункт Пустой файл Java и щелкните на кнопке Далее.
4. В поле Имя класса введите **SiameseCat**.

5. В поле Пакет введите **org.cadenhead.minecraft**.
6. Щелкните на кнопке Готово.

Файл SiameseCat.java откроется в редакторе исходного кода NetBeans.

Каждый мод, создаваемый в Spigot, начинается с простого каркаса, который содержит стандартный код. После инструкции package и нескольких инструкций import следуют инструкции, выполняющие запуск мода.

```
public class SiameseCat extends JavaPlugin {
   public static final Logger log = Logger.getLogger("Minecraft");

   public boolean onCommand(CommandSender sender, Command command,
      String label, String[] arguments) {

      if (label.equalsIgnoreCase("siamesecat")) {
         if (sender instanceof Player) {
            // Пользовательские инструкции
            log.info("[SiameseCat] Meow!");
            return true;
         }
      }
      return false;
   }
}
```

Программы модов являются подклассами класса JavaPlugin, входящего в пакет org.bukkit.plugin.java. Каркас, являющийся основой мода, можно использовать для создания других модов. Единственное, что нужно будет поменять, — это три ссылки с упоминанием сиамского кота, потому что они специфичны для данного проекта.

Название программы в нашем случае — SiameseCat.

Аргумент, указанный в вызове метода label.equalsIgnoreCase ("siamesecat"), соответствует команде, которую пользователь будет вводить в игре для вызова мода. В данном случае это команда /siamesecat (командам предшествует символ /). Объект label, который в качестве аргумента передается методу onCommand(), представляет собой строку, в которой находится текст команды, введенной пользователем.

Метод log.info() отсылает журнальное сообщение "[SiameseCat] Meow!", которое отображается в окне сервера Minecraft. Соответствующий объект Logger, созданный в виде переменной экземпляра, служит для отправки сообщений серверу.

Объект sender — это еще один аргумент, передаваемый методу onCommand(). Он представляет персонажа Minecraft, который отправил команду. Проверка того, что отправитель является экземпляром класса Player, позволяет гарантировать, что команду отдал игрок.

Все действия мода, выполняемые после ввода команды, задаются в разделе, который помечен комментарием `// Пользовательские инструкции`.

Первым делом мод `SiameseCat` должен получше изучить игровой мир. Для этого используются следующие три инструкции.

```
Player me = (Player) sender;
Location spot = me.getLocation();
World world = me.getWorld();
```

Объект `me`, создаваемый путем приведения объекта `sender` к типу `Player`, представляет собой персонажа, управляемого игроком.

Для объекта `Player` можно вызвать метод `getLocation()` без аргументов, чтобы узнать точное местонахождение игрока, которое представлено классом `Location` и задается с помощью трех координат *(x,y,z)* на трехмерной карте игры.

Объект `Player` также включает метод `getWorld()`, который возвращает объект `World`, представляющий весь игровой мир.

Большинство создаваемых модов нуждается в объектах `Player`, `Location` и `World`.

Следующий мод создает новый моб, который является оцелотом. При этом используется метод `spawn()` класса `World`.

```
Ocelot cat = world.spawn(spot, Ocelot.class);
```

Для каждого типа моба в игре есть свой класс. Два аргумента метода `spawn()` задают местоположение, где должен находиться кот, и класс, применяемый для создания моба.

Приведенная выше инструкция создает объект `Ocelot` с именем `cat`, который находится в том же месте, где и игрок.

На данный момент мы создали оцелота, но его можно превратить в сиамского кота, выполнив следующую инструкцию:

```
cat.setCatType(Ocelot.Type.SIAMESE_CAT);
```

Метод `setCatType()` позволяет выбрать разновидность моба: оцелот, сиамский, черный или рыжий кот.

Коты в Minecraft являются мобами, которые могут быть приручены владельцем. Эта связь устанавливается путем вызова метода `setOwner()`, в качестве аргумента которого выступает игрок:

```
cat.setOwner(me);
```

Объединим рассмотренные выше фрагменты кода в листинг 23.2. После ввода кода листинга в редакторе исходного кода щелкните на кнопке **Сохранить все** на панели инструментов NetBeans (или выполните команду **Файл⇨Сохранить**).

ЛИСТИНГ 23.2. Исходный код программы `SiameseCat.java`

```
 1: package org.cadenhead.minecraft;
 2:
 3: import java.util.logging.*;
 4: import org.bukkit.*;
 5: import org.bukkit.command.*;
 6: import org.bukkit.entity.*;
 7: import org.bukkit.plugin.java.*;
 8:
 9: public class SiameseCat extends JavaPlugin {
10:    public static final Logger log = Logger.getLogger("Minecraft");
11:
12:    public boolean onCommand(CommandSender sender,
13:       Command command, String label, String[] arguments) {
14:
15:       if (label.equalsIgnoreCase("siamesecat")) {
16:          if (sender instanceof Player) {
17:             // Определить игрока
18:             Player me = (Player) sender;
19:             // Определить текущее местоположение игрока
20:             Location spot = me.getLocation();
21:             // Получить игровой мир
22:             World world = me.getWorld();
23:
24:             // Создать оцелота
25:             Ocelot cat = world.spawn(spot, Ocelot.class);
26:             // Превратить его в сиамского кота
27:             cat.setCatType(Ocelot.Type.SIAMESE_CAT);
28:             // Сделать игрока хозяином кота
29:             cat.setOwner(me);
30:             log.info("[SiameseCat] Meow!");
31:             return true;
32:          }
33:       }
34:       return false;
35:    }
36: }
```

Инструкции `import` в строках 3–7 подключают к программе пять пакетов: один на основе библиотеки классов Java и четыре на основе Spigot.

ПРИМЕЧАНИЕ

В программе используются следующие классы: `Logger` из пакета `java.util.logging`, `Location` и `World` из пакета `org.bukkit`, `Command` из пакета `org.bukkit.command`, `Ocelot` и `Player` из пакета `org.bukkit.entity` и `JavaPlugin` из пакета `org.bukkit.plugin.java`. Эти пакеты будут подробно рассмотрены далее.

Инструкции `return` в строках 31 и 34 являются частью стандартного каркаса мода. Метод `onCommand()` должен возвращать значение `true`, если мод обрабатывает команду пользователя, и `false` в противном случае.

Итак, вы только что создали свой первый мод, но его пока еще нельзя запустить под управлением Spigot. Для этого дополнительно понадобится файл `plugin.yml`, который сообщает серверу Minecraft о наличии мода.

Этот файл является текстовым и может быть создан в среде NetBeans следующим образом.

1. Выполните команду **Файл⇨Создать файл**, чтобы открыть диалоговое окно **Создать файл**.
2. На панели **Категории** выберите пункт **Прочее**.
3. На панели **Типы файлов** выберите пункт **Пустой файл** и щелкните на кнопке **Далее**.
4. В поле **Имя файла** введите **`plugin.yml`**.
5. В поле **Папка** введите **`src`**.
6. Щелкните на кнопке **Готово**.

После этого пустой файл `plugin.yml` откроется в редакторе исходного кода. Введите код из листинга 23.3 и проверьте, чтобы в каждой строке было одинаковое число пробелов. Не используйте символы табуляции вместо пробелов!

ЛИСТИНГ 23.3. Файл `plugin.yml` для текущего проекта

```
 1: name: SiameseCat
 2:
 3: author: Ваше имя
 4:
 5: main: org.cadenhead.minecraft.SiameseCat
 6:
 7: commands:
 8:    siamesecat:
 9:        description: создание сиамского кода.
10:
11: version: 1.0
```

Замените фразу *Ваше имя* своим именем.

Тщательно проверьте количество пробелов. В частности, должно быть четыре пробела в строке 8 перед словом `siamesecat` и восемь пробелов в строке 9 перед словом `description`.

В файл `plugin.yml` включено название мода, имя автора, название файла класса Java b номер версии. Также включена команда и приведено ее краткое описание.

Название этого файла отображается под заголовком **<пакет по умолчанию>** в разделе **Пакеты исходных кодов** на панели **Проекты** (рис. 23.8).

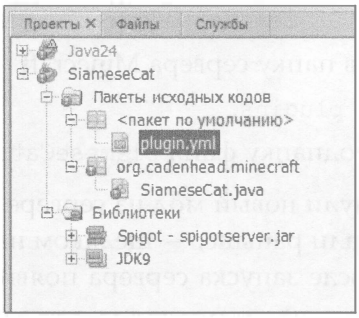


РИС. 23.8. Проверка местоположения файла `plugin.yml`

Если файл находится не на своем месте, например под заголовком `org.cadenhead.minecraft`, перетащите его в нужное место. Щелкните на имени файла и перетащите его на значок раздела **Пакеты исходных кодов**.

Теперь вы готовы к созданию мода. Выберите команду **Выполнить⇨ Очистить и собрать проект**.

Если все прошло успешно, в левом нижнем углу интерфейса пользователя NetBeans появится сообщение `"Finished Building SiameseCat (clean jar)"`.

Мод упакован в виде файла `SiameseCat.jar`, находящегося в подпапке проекта. Чтобы найти этот файл, щелкните на вкладке **Файлы** панели **Проекты** и откройте подпапку `dist`. На вкладке **Файлы** перечислены все файлы, которые образуют проект (рис. 23.9).

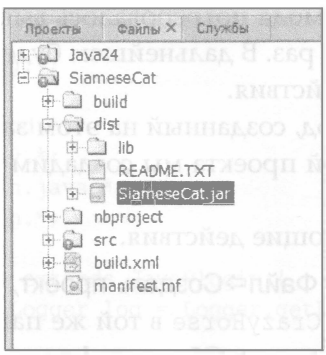


РИС. 23.9. В поисках JAR-файла мода SiameseCat

Файл `SiameseCat.jar` нужно скопировать из папки проекта на сервер Minecraft. Выполните следующие действия.

1. Если сервер Minecraft выполняется, остановите его путем перехода в окно сервера. Выйдите из среды NetBeans и откройте папку, в которой установлен сервер Minecraft.
2. Откройте подпапку `SiameseCat`, а затем подпапку `dist`.
3. Вернитесь обратно в папку сервера Minecraft.
4. Откройте подпапку `plugins`.
5. Скопируйте в эту подпапку файл `SiameseCat.jar`.

Вы только что развернули новый мод на сервере Minecraft. Запустите сервер точно так же, как делали раньше, — щелчком на значке командного файла `start-server.bat`. После запуска сервера появится следующее новое сообщение:

```
[SiameseCat] Enabling SiameseCat v1.0
```

Если же вместо этого сообщения появится целый ряд длинных и сложных сообщений об ошибках, тщательно проверьте код в файлах `SiameseCat.java` и `plugin.yml`, чтобы убедиться в его корректности, а затем повторно разверните мод.

Запустите сервер Minecraft, подключитесь к нему и введите команду **`/siamesecat`**. После этого в игровом мире появится новый сиамский кот, который будет следовать за вами повсюду. Каждый последующий ввод этой команды приводит к появлению нового кота.

На рис. 23.10 показаны результаты выполнения этой команды.

Учим зомби объезжать лошадей

При создании первого мода потребовалось выполнить много работы, поскольку все было в первый раз. В дальнейшем будет проще, так как мы будем повторять одни и те же действия.

Каждый следующий мод, созданный на этом занятии, будет сложнее предыдущего. В рамках второй проекта мы создадим кавалерийский отряд зомби, оседлавших лошадей.

Итак, выполните следующие действия.

1. Выполните команду **Файл⇨Создать проект**, чтобы создать новый проект Java под именем `CrazyHorse` в той же папке, что и сервер Minecraft, как в предыдущем проекте. Сбросьте флажок **Создать главный класс**.

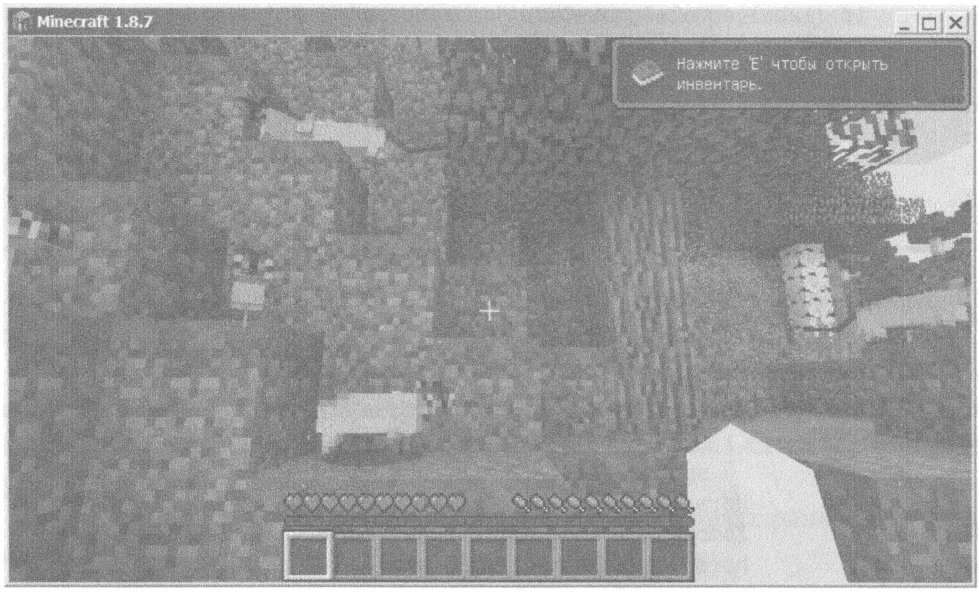


РИС. 23.10. Ко мне, мои верные коты!

2. Щелкните правой кнопкой мыши на записи **Библиотеки** панели **Проекты**, выберите пункт **Добавить библиотеку** и добавьте библиотеку Spigot в проект.
3. Выполните команду **Файл⇨Создать файл**, чтобы создать пустой файл класса `CrazyHorse` в пакете `org.cadenhead.minecraft`.

Введите код листинга 23.4 в редакторе исходного кода. Новые приемы, использованные в программе, будут подробно рассмотрены далее.

ЛИСТИНГ 23.4. Исходный код программы `CrazyHorse.java`

```
 1: package org.cadenhead.minecraft;
 2:
 3: import java.util.logging.*;
 4: import org.bukkit.*;
 5: import org.bukkit.command.*;
 6: import org.bukkit.entity.*;
 7: import org.bukkit.plugin.java.*;
 8: import org.bukkit.potion.*;
 9:
10: public class CrazyHorse extends JavaPlugin {
11:    public static final Logger log = Logger.getLogger("Minecraft");
12:
13:    public boolean onCommand(CommandSender sender,
14:       Command command, String label, String[] arguments) {
15:
```

```
16:       if (label.equalsIgnoreCase("crazyhorse")) {
17:          if (sender instanceof Player) {
18:             Player me = (Player) sender;
19:             Location spot = me.getLocation();
20:             World world = me.getWorld();
21:
22:             // Создание зомби, скачущих на лошадях
23:             int quantity = (int) (Math.random() * 10) + 1;
24:             log.info("[CrazyHorse] Creating " + quantity +
25:                      " zombies and horses");
26:             for (int i = 0; i < quantity; i++) {
27:                // Настроить местоположение лошадей и зомби
28:                Location horseSpot = new Location(world,
29:                   spot.getX() + (Math.random() * 15),
30:                   spot.getY() + 20,
31:                   spot.getZ() + (Math.random() * 15));
32:                Horse horse = world.spawn(horseSpot, Horse.class);
33:                Zombie zombie = world.spawn(horseSpot, Zombie.class);
34:                horse.setPassenger(zombie);
35:                horse.setOwner(me);
36:                // Выбор окраса лошади
37:                int coat = (int) (Math.random() * 7);
38:                switch (coat) {
39:                   case 0:
40:                      horse.setColor(Horse.Color.WHITE);
41:                      break;
42:                   case 1:
43:                      horse.setColor(Horse.Color.GRAY);
44:                      break;
45:                   case 2:
46:                      horse.setColor(Horse.Color.CREAMY);
47:                      break;
48:                   case 3:
49:                      horse.setColor(Horse.Color.CHESTNUT);
50:                      break;
51:                   case 4:
52:                      horse.setColor(Horse.Color.BROWN);
53:                      break;
54:                   case 5:
55:                      horse.setColor(Horse.Color.DARK_BROWN);
56:                      break;
57:                   case 6:
58:                      horse.setColor(Horse.Color.BLACK);
59:                }
60:                // Увеличение скорости лошади
61:                PotionEffect potion = new PotionEffect(
62:                   PotionEffectType.SPEED,
63:                   Integer.MAX_VALUE,
64:                   10 + (coat * 10));
```

```
65:                horse.addPotionEffect(potion);
66:             }
67:             return true;
68:          }
69:       }
70:       return false;
71:    }
72: }
```

После ввода кода листинга сохраните файл `CrazyHorse.java`, выполнив команду **Файл⇨Сохранить** или щелкнув на кнопке **Сохранить все**.

Этот мод добавляет команду `/crazyhorse` в Minecraft, в результате выполнения которой с небес спускается случайное количество зомби верхом на лошадях. Данная команда обрабатывается в строках 18–65. Остальная часть кода представляет собой стандартный каркас мода.

После выбора игрока, местоположения и мира мод сохраняет случайное число (от 1 до 11) в переменной.

```
int quantity = (int) (Math.random() * 10) + 1;
```

Метод `random()` из класса `Math` генерирует случайное число с плавающей точкой, находящееся в диапазоне от 0,0 до 1,0. После умножения этого числа на целый множитель создается случайное целое число, которое изменяется в диапазоне от нуля до множителя. Данное число представляет собой количество создаваемых зомби и лошадей.

Цикл `for`, который начинается в строке 26, обходит по одному разу каждую пару, состоящую из лошади и зомби.

Для указания местоположения зомби и лошадей вызывается конструктор `Location()` с четырьмя аргументами: игровым миром, координатами *x*, *y* и *z*.

```
Location horseSpot = new Location(world,
   spot.getX() + (Math.random() * 15),
   spot.getY() + 20,
   spot.getZ() + (Math.random() * 15));
```

Местоположение игрока, хранящееся в переменной `spot`, используется в качестве привязки для определения местоположения зомби и лошади. Для определения координаты *x* выбирается значение `getX()` координаты игрока, к которому добавляется случайное число из диапазона от 0 до 14. Для определения координаты *z* выбирается значение `getZ()`, к которому тоже добавляется случайное число из диапазона от 0 до 14. Для определения координаты *y* к значению `getY()` прибавляется число 20. Это означает, что мобы появятся на 20 блоков выше игрока (в небе) и в пределах 14 блоков от игрока в других двух направлениях.

Лошади и зомби создаются так же, как и сиамские коты. Вызовите метод игрового мира `spawn()` с указанием местоположения моба и его класса. При этом класс `Horse` представляет лошадей, а класс `Zombie` — зомби.

Каждый моб, способный перевозить кого-то в Minecraft, включает метод `setPassenger()`, который получает объект всадника в качестве единственного аргумента.

```
horse.setPassenger(zombie);
```

Если остановиться на этом этапе, зомби оседлает лошадь, но она быстро сбросит его. Чтобы предотвратить это, нужно приручить лошадь.

```
horse.setOwner(me);
```

Приручённая лошадь не сбросит всадника, даже если это зомби, жаждущий человеческой плоти. После создания моба наследуется стандартное поведение этого типа моба из игры.

Как и в случае с сиамскими котами, лошадей можно настраивать несколькими способами. Один из них заключается в изменении окраса лошади с помощью метода `setColor()`. Всего доступно семь возможных цветов.

После генерирования случайного числа в диапазоне от 0 до 6 оно будет использовано в инструкции `switch-case` (строки 38–59) для выбора окраса лошади.

Чтобы придать лошадям больше резвости, можно настроить также скорость их перемещения. К мобу можно применить эффект игрового зелья. Доступны 23 разных эффектов зелья, которые выбираются с помощью класса `PotionEffectType`.

Для создания эффекта зелья вызывается конструктор `PotionEffect` с тремя аргументами: типом эффекта, продолжительностью действия зелья и значением этого эффекта.

Следующая инструкция создает быстродействующее зелье с максимально возможной длительностью, задаваемой наибольшим целым числом, которое может храниться в Java.

```
PotionEffect potion = new PotionEffect(
   PotionEffectType.SPEED,
   Integer.MAX_VALUE,
   10 + (coat * 10));
```

Третий аргумент конструктора `PotionEffect()` настраивает скорость перемещения лошади. Этот аргумент имеет базовое значение 10, к которому добавляется случайное число, применяемое для определения окраса лошади, умноженное на 10. Поскольку цвета в инструкции `switch-case` упорядочены от светлого к темному, применение формулы `10 + (coat * 10)` приводит к ускорению перемещения лошадей по мере потемнения их окраса.

После создания класса Java для мода нужно создать файл описания подключаемого модуля. Выполните команду **Создать файл**, а затем на панели **Категории** выберите параметр **Прочее** и создайте пустой файл `plugin.yml` в папке `src`.

Введите код листинга 23.5 в файл, отформатировав его точно так же, как в листинге. При форматировании используйте пробелы, а не символы табуляции.

ЛИСТИНГ 23.5. Файл `plugin.yml` для текущего проекта

```
 1: name: CrazyHorse
 2:
 3: author: Ваше имя
 4:
 5: main: org.cadenhead.minecraft.CrazyHorse
 6:
 7: commands:
 8:     crazyhorse:
 9:         description: Порождает 1-11 зомби верхом на лошадях.
10:
11: version: 1.0
```

После ввода файла выполните команду **Выполнить⇨Очистить и собрать проект**, чтобы создать JAR-файл мода.

На панели **Проекты** под заголовком **Пакеты исходного кода** появятся два файла: `CrazyHorse.java`, относящийся к пакету `org.cadenhead.minecraft`, и файл `plugin.yml`, относящийся к разделу **<пакет, заданный по умолчанию>**.

За пределами среды NetBeans перейдите к папкам файла и скопируйте файл `CrazyHorse.jar` из папки проекта `dist` в папку сервера `plugins`. Перезапустите сервер (остановив и повторного запустив его), и вы сможете ввести команду `/crazyhorse`.

После ввода этой команды вокруг вас появится от 1 до 11 зомби верхом на лошадях. Соответствующий снимок экрана показан на рис. 23.11.

Разработчики Spigot отлично поработали над тем, чтобы классы Java соответствовали игровым объектам. После создания двух первых модов вы, вероятно, начнете думать о том, что можно изменить, приложив небольшие усилия.

Вот небольшое изменение в строке 33 листинга 23.4, которое приведет к изменению мода `CrazyHorse`:

```
Creeper creeper = world.spawn(horseSpot, Creeper.class);
```

В результате подобного изменения вместо зомби создается группа криперов на лошадях.

РИС. 23.11. Зомби, оседлавшие коней

Для всех классов, относящихся к Spigot, в Интернете можно найти обширную справочную информацию (`http://www.javaminecraft.com/bukkitapi`). Используйте эту информацию для изучения классов и интерфейсов Spigot, их переменных экземпляров, а также методов экземпляров и классов.

Изучая документацию и перестраивая мод `CrazyHorse`, вы сможете поиграть с разными мобами, а также испытать различные эффекты от действия зелья.

Следующий код реализует модификацию, которая делает лошадь невидимой, создавая эффект парения зомби в воздухе.

```
PotionEffect potion2 = new PotionEffect(
   PotionEffectType.INVISIBILITY,
   Integer.MAX_VALUE,
   1);
horse.addPotionEffect(potion2);
```

Поиск и уничтожение всех мобов

Первые два мода, которых мы создали в этой главе, населили игровой мир мобами. Следующий проект позволяет исследовать существующие мобы, найти каждого из них и поразить его ударом молнии.

Начните с выполнения следующих действий.

1. Выполните команду **Файл⇨Создать проект**, чтобы создать новый проект `LightningStorm` в папке сервера Minecraft. Сбросьте флажок **Создать главный класс**.
2. На панели **Проекты** щелкните правой кнопкой мыши на разделе **Библиотеки**, в контекстном меню выберите пункт **Добавить библиотеку**, после чего добавьте библиотеку Spigot.
3. Выполните команду **Файл⇨Создать файл**, чтобы создать пустой класс `LightningStorm` в пакете `org.cadenhead.minecraft`.

С помощью редактора исходного кода NetBeans введите код из листинга 23.6 в только что созданный файл и сохраните его под именем `LightningStorm.java`.

ЛИСТИНГ 23.6. Исходный код программы `LightningStorm.java`

```
 1: package org.cadenhead.minecraft;
 2:
 3: import java.util.*;
 4: import java.util.logging.*;
 5: import org.bukkit.*;
 6: import org.bukkit.command.*;
 7: import org.bukkit.entity.*;
 8: import org.bukkit.plugin.java.*;
 9:
10: public class LightningStorm extends JavaPlugin {
11:    public static final Logger log = Logger.getLogger("Minecraft");
12:
13:    public boolean onCommand(CommandSender sender,
14:       Command command, String label, String[] arguments) {
15:
16:       Player me = (Player) sender;
17:       World world = me.getWorld();
18:       // Поиск всех живых мобов в игре
19:       List<LivingEntity> mobs = world.getLivingEntities();
20:
21:       // Выполнение команды lightningstorm
22:       if (label.equalsIgnoreCase("lightningstorm")) {
23:          if (sender instanceof Player) {
24:             int myId = me.getEntityId();
25:             // Поочередный обход всех мобов
26:             for (LivingEntity mob : mobs) {
27:                log.info("[LightningStorm]" + mob.getType());
28:                // Является ли этот моб игроком?
29:                if (mob.getEntityId() == myId) {
30:                   // Да, не будем убивать его
31:                   continue;
32:                }
```

```
33:                 // Определение местоположения моба
34:                 Location spot = mob.getLocation();
35:                 // Удар молнии!
36:                 world.strikeLightning(spot);
37:                 // Обнуление здоровья моба (убит)
38:                 mob.setHealth(0);
39:              }
40:           }
41:           return true;
42:        }
43:
44:        // Выполнение команды mobcount
45:        if (label.equalsIgnoreCase("mobcount")) {
46:           if (sender instanceof Player) {
47:              me.sendMessage("There are " + mobs.size() +
48:                             " living mobs.");
49:              return true;
50:           }
51:        }
52:        return false;
53:     }
54: }
```

Мод `LightningStorm` имеет иной каркас: он отслеживает две команды вместо одной. Команда `/lightningstorm` обрушивает молнии на мобов, а команда `/mobcount` подсчитывает количество живых мобов в игровом мире.

Мод Minecraft для сервера Spigot может реализовывать произвольное количество команд. Код команды `/lightningstorm` находится в строках 22–42 листинга 23.6, а код команды `/mobcount` — в строках 45–51.

Для выполнения этих команд требуются объекты `player` и `world`. Также нужен список всех живых мобов (строка 19):

```
List<LivingEntity> mobs = world.getLivingEntities();
```

Эта инструкция вызывает метод `getLivingEntitities()` объекта `world`, который возвращает объект `List`, включающий каждого моба. Мобы представлены в виде объектов, которые реализуют интерфейс `LivingEntity`.

Класс `List` — это тоже интерфейс, который реализуется с помощью списков массивов, стеков и других структур данных. Методы и приемы, используемые при работе со списками массивов, можно применять и здесь.

После формирования списка можно создать код команды `/mobcount` с помощью единственной инструкции (строки 47 и 48):

```
me.sendMessage("There are " + mobs.size() + " living mobs.");
```

Команда списка `size()` возвращает количество элементов в списке. Эта информация отсылается игроку в виде сообщения, которое отображается во время игры. Для отправки сообщения вызывается метод `sendMessage()` объекта `player`, имеющий единственный аргумент.

Реализовать команду `/lightningstorm` значительно сложнее.

Для работы со структурами данных в Java предусмотрена специальная разновидность цикла `for`, которая облегчает циклический обход элементов структуры. В строке 26 начинается цикл обхода каждого моба в списке `mobs`.

```
for (LivingEntity mob : mobs) {
```

В результате выполнения этой инструкции текущий моб сохраняется в объекте `mob` класса `LivingEntity`.

Если вы вознамерились поразить ударом молнии всех мобов на карте, то не забывайте о том, что сам игрок тоже является мобом, и вряд ли вам понравится идея испытать на себе разящий удар молнии.

Каждому мобу соответствует уникальный идентификатор (ID). Идентификатор игрока сохраняется перед началом цикла, в строке 24.

```
int myId = me.getEntityId();
```

Целое число, полученное в результате выполнения этой инструкции, в цикле сравнивается с идентификатором текущего моба. Для выполнения сравнения в строках 29–32 вызывается соответствующий метод `getEntityID()`.

```
if (mob.getEntityId() == myId) {
   continue;
}
```

Выполнение инструкции `continue` приведет к тому, что цикл возвращается обратно к строке 26 и начинает обрабатывать следующего моба.

Теперь, когда игрок в безопасности, переходим к определению местонахождения мобов путем вызова соответствующего метода `getLocation()`.

```
Location spot = mob.getLocation();
```

Затем для этого местоположения вызывается метод `strikeLightning()` объекта `world`.

```
world.strikeLightning(spot);
```

Далеко не всегда удар молнии будет фатальным для моба, некоторые из них выживают после этого. Если же вызвать метод моба `setHealth()`, аргумент которого равен нулю, то это приведет к обнулению здоровья моба, т.е. убьет его.

```
mob.setHealth(0);
```

Для этого мода в папке проекта `src` создается файл `plugin.yml`, код которого приведен в листинге 23.7.

ЛИСТИНГ 23.7. Файл `plugin.yml` для текущего проекта

```
 1: name: LightningStorm
 2:
 3: author: Ваше имя
 4:
 5: main: org.cadenhead.minecraft.LightningStorm
 6:
 7: commands:
 8:     lightningstorm:
 9:         description: Поражение каждого живого моба ударом молнии.
10:     mobcount:
11:         description: Подсчет живых мобов.
12:
13: version: 1.0
```

Соберите проект с помощью команды Выполнить⇨Очистить и собрать проект и разверните мод на сервере Minecraft. Для развертывания мода скопируйте файл `LightningStorm.jar` из папки проекта `dist` в папку сервера `plugins`. В следующий раз, когда вы запустите сервер и сыграете в игру, вы увидите, как на мобов обрушиваются молнии. В результате удара молнии корова, показанная на рис. 23.12, превратится в обугленный шашлык.

РИС. 23.12. Этой корове осталось жить недолго

Создание мода-строителя

Первые три мода, созданные в этой главе, были связаны с мобами, т.е. подвижными сущностями, которые бродили по миру Minecraft. Мод, который будет создан в этом разделе, демонстрирует использование Java для выполнения базовых действий в игре: строительства и разрушения блоков.

В проекте будет создана новая команда, `/icecreamscoop`, которая применяется для рытья круглой ямы вокруг игрока. Размер ямы определяется игроком как аргумент команды.

Чтобы создать мода-строителя, выполните следующие действия.

1. В папке проектов NetBeans создайте новый проект `IceCreamScoop` и бросьте флажок **Создать главный класс**.
2. На панели **Проекты** щелкните правой кнопкой мыши на пункте **Библиотеки** и добавьте библиотеку Spigot.
3. Создайте пустой класс `IceCreamScoop` в пакете `org.cadenhead.minecraft`.

Введите код из листинга 23.8 и сохраните полученный файл `IceCreamScoop.java`.

ЛИСТИНГ 23.8. Исходный код программы `IceCreamScoop.java`

```
 1: package org.cadenhead.minecraft;
 2:
 3: import java.util.logging.*;
 4: import org.bukkit.*;
 5: import org.bukkit.block.*;
 6: import org.bukkit.command.*;
 7: import org.bukkit.entity.*;
 8: import org.bukkit.plugin.java.*;
 9:
10: public class IceCreamScoop extends JavaPlugin {
11:    public static final Logger log = Logger.getLogger("Minecraft");
12:
13:    public boolean onCommand(CommandSender sender,
14:       Command command, String label, String[] arguments) {
15:
16:       // Радиус ямы, заданный по умолчанию
17:       double radius = 15;
18:       if (arguments.length > 0) {
19:          try {
20:             // Считывание радиуса, заданного пользователем
21:             radius = Double.parseDouble(arguments[0]);
22:             // Проверка допустимости радиуса (5-25)
23:             if ((radius < 5) | (radius > 25)) {
```

```
24:                radius = 15;
25:             }
26:          } catch (NumberFormatException exception) {
27:             // Использовать значение по умолчанию
28:          }
29:       }
30:
31:       if (label.equalsIgnoreCase("icecreamscoop")) {
32:          if (sender instanceof Player) {
33:             scoopTerrain(sender, radius);
34:          }
35:          return true;
36:       }
37:       return false;
38:    }
39:
40:    // Выкопать круглую яму в мире Minecraft
41:    private void scoopTerrain(CommandSender sender, double rad) {
42:       Player me = (Player) sender;
43:       Location spot = me.getLocation();
44:       World world = me.getWorld();
45:
46:       // Циклический обход куба, стороны которого
47:       // вдвое больше радиуса
48:       for (double x = spot.getX() - rad; x < spot.getX() + rad;
49:            x++) {
50:          for (double y = spot.getY() - rad; y < spot.getY() + rad;
51:               y++) {
52:             for (double z = spot.getZ() - rad; z < spot.getZ() + rad;
53:                  z++) {
54:                // Вычисление местоположения внутри куба
55:                Location loc = new Location(world, x, y, z);
56:                // Проверка удаленности от игрока
57:                double xd = x - spot.getX();
58:                double yd = y - spot.getY();
59:                double zd = z - spot.getZ();
60:                double distance = Math.sqrt(xd * xd + yd * yd +
61:                                            zd * zd);
62:                // Находится в пределах радиуса?
63:                if (distance < rad) {
64:                   // Да, разрушить этот блок
65:                   Block current = world.getBlockAt(loc);
66:                   current.setType(Material.AIR);
67:                }
68:             }
69:          }
70:       }
71:
72:       // Воспроизведение звука при копании ямы
```

```
73:        world.playSound(spot, Sound.BURP, 30, 5);
74:        log.info("[IceCreamScoop] Scooped at ("
75:           + spot.getX() + ","
76:           + spot.getY() + ","
77:           + spot.getZ() + ")");
78:     }
79: }
```

Этот мод выкапывает круглую яму, радиус которой изменяется от 5 до 25 блоков. Размер ямы определяет игрок путем вызова команды `/icecreamscoop`, за которой следует пробел и число:

```
/icecreamscoop 10
```

Подобно любым приложениям Java, моды могут принимать аргументы. Значения аргументов хранятся в четвертом аргументе метода `onCommand()`, в массиве типа `String`.

После создания переменной `radius` типа `double` (строка 17), которой присваивается значение 15, следующая инструкция присваивает переменной `radius` значение первого аргумента:

```
radius = Double.parseDouble(arguments[0]);
```

Метод `parseDouble()` встроенного класса `Double` преобразует строку в значение типа `double` в случае, когда подобное преобразование возможно.

Чтобы предотвратить ввод некорректных данных пользователем, метод `parseDouble()` нужно поместить в блок `try-catch`. Если пользователь ввел нечисловое значение, будет сгенерировано исключение `NumberFormatException`.

Введенное пользователем числовое значение присваивается радиусу. В строках 23–25 выполняется еще одна проверка, реализованная на основе условного выражения `if`, цель которой — удостовериться в том, что значение радиуса попадает в допустимый диапазон от 5 до 25.

Класс `IceCreamScoop` включает метод `scoopTerrain()`, который применяется для копания круглых ям. Подобно другим модам, данному моду требуются объекты, соответствующие игроку, его местоположению и игровому миру.

Существуют разные методики копания круглых ям вокруг игрока в трехмерной сетке, подобной миру Minecraft. Данный мод просматривает каждый блок, находящийся в кубе вокруг игрока, причем стороны этого куба вдвое больше радиуса ямы. Центр куба находится в точке, где стоит игрок.

Три вложенных цикла `for`, которые начинаются в строках 48–53, применяются для просмотра всех блоков. Первый цикл `for` выполняет итерации по оси X, второй цикл `for` — по оси Y, третий цикл `for` — по оси Z.

Для представления точки, которая просматривается на каждом шаге цикла, создается объект `Location`.

```
loc = new Location(world, x, y, z);
```

Информация о местоположении игрока была предварительно сохранена в объекте `spot` (строка 43).

```
Location spot = me.getLocation();
```

Для измерения расстояния между текущей точкой и игроком по всем трем осям применяются следующие три инструкции.

```
double xd = x - spot.getX();
double yd = y - spot.getY();
double zd = z - spot.getZ();
```

Расстояние определяется по теореме Пифагора. Сначала вычисляется сумма квадратов `xd`, `yd` и `zd`, после чего с помощью метода `sqrt()` класса `Math` извлекается квадратный корень из этой суммы:

```
double distance = Math.sqrt(xd * xd + yd * yd + zd * zd);
```

Если расстояние от текущего блока до игрока находится в пределах заданного игроком радиуса, блок взлетает на воздух с помощью следующих двух инструкций.

```
Block current = world.getBlockAt(loc);
current.setType(Material.AIR);
```

Сначала создается объект `Block`, который представляет блок. Затем вызывается метод блока `setType()` со значением `Material.AIR`. Это одно из нескольких десятков значений, представляющих материал блока в Minecraft.

Методику разрушения блока путем его превращения в воздух можно применить и в созидательных целях. Достаточно лишь изменить тип материала. Например, чтобы превратить выкопанную яму в бриллиант, воспользуйтесь следующей инструкцией:

```
current.setType(Material.DIAMOND);
```

После того как яма выкопана, происходят два события: воспроизводится звук и в журнал сервера записывается сообщение.

Для воспроизведения звука вызывается метод `playSound()` объекта `world`. Этому методу передаются следующие четыре аргумента: местоположение источника звука, тип, громкость и высота звука.

```
world.playSound(spot, Sound.BURP, 30, 5);
```

Используемый в данном случае звук напоминает отрыжку. Существуют десятки других возможных звуков.

Сообщение записывается в журнал сервера путем вызова команды `info()` объекта `Logger` с текстом сообщения в качестве аргумента.

```
log.info("[IceCreamScoop] Scooped at " + spot);
```

Для этого мода в папке проекта `src` создается файл `plugin.yml`, код которого приведен в листинге 23.9.

ЛИСТИНГ 23.9. Файл `plugin.yml` для текущего проекта

```
 1: name: IceCreamScoop
 2:
 3: author: Ваше имя
 4:
 5: main: org.cadenhead.minecraft.IceCreamScoop
 6:
 7: commands:
 8:     icecreamscoop:
 9:         description: Копание ямы вокруг игрока.
10:
11: version: 1.0
```

После создания мода и развертывания соответствующего JAR-файла на сервере Minecraft с помощью тех же действий, что и в предыдущих проектах, вы получите возможность перемещаться по игровому миру и копать огромные ямы. На рис. 23.13 показано, что случится, если вы встанете на плоской поверхности и воспользуетесь следующей командой:

```
/icecreamscoop 10
```

В процессе тестирования этого мода обнаружилось, что он вполне способен прокопать яму до самого основания игрового мира. Поэтому не увлекайтесь копанием в одной точке. Это чревато падением с большой высоты, что повлечет за собой смерть вашего мода-строителя.

Путем выбора другого материала (вместо исходного материала `AIR`) можно создавать шары в игровом мире. Полный список материалов приводится в документации Spigot Java (`www.javaminecraft.com/bukkitapi`). Ниже приведены наиболее популярные материалы:

- `Material.DIRT`;
- `Material.COBBLESTONE`;
- `Material.WOOD`;
- `Material.GRAVEL`.

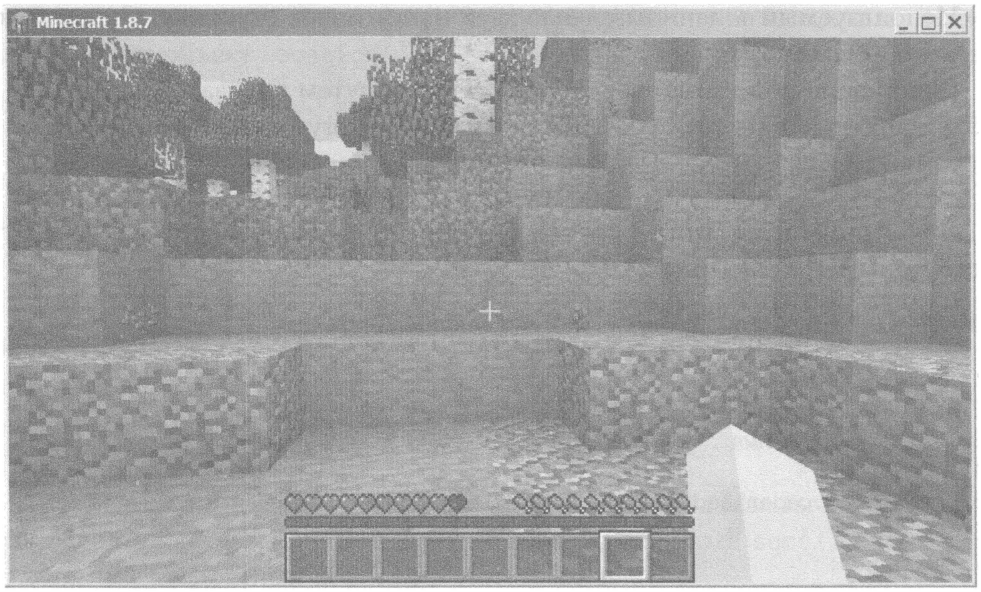


РИС. 23.13. Копаем котлован в мире Minecraft

Если превращать блоки в твердую субстанцию, а не в воздух, то это может привести к неприятным последствиям для игрока. Чтобы избежать неприятностей, выполните телепортацию с помощью следующих двух инструкций.

```
Location newSpot = new Location(world,
   spot.getX(),
   spot.getY() + rad + 1,
   spot.getZ());
me.teleport(newSpot);
```

Поскольку ось Y в Minecraft соответствует вертикальной позиции, в результате выполнения этих инструкций игрок переместится в точку, находящуюся прямо над сферой.

Резюме

За все годы, когда мне приходилось писать программы на Java и учить пользователей этому языку, я никогда не думал, что буду использовать эти знания для:

- обучения зомби езде на лошадях;
- обучения лошадей сбрасыванию зомби;
- активизации зомби после наступления ночи;
- рытья огромных котлованов.

Это самый забавный вид программирования на Java. Вам размещаете собственноручно созданные объекты Java в трехмерном мире, где вам предстоит сражаться с монстрами, рыть туннели, строить дома и убегать от толп живых мертвецов.

Благодаря Minecraft обеспечивается великолепная возможность обучения объектно-ориентированному программированию — одному из наиболее сложных аспектов Java. Все операции в Java реализуются посредством объектов. Вы создаете объекты с помощью конструкторов, записываете в них данные с помощью переменных экземпляра и заставляете их выполнять нужные вам действия с помощью методов. Объекты обозначают задачи, которые они могут выполнять, путем реализации интерфейсов.

Все эти концепции напрямую связаны с программированием модов. Вам нужен зомби? Создайте объект `Zombie`. Хотите, чтобы игрок переместился в новую точку? Вызовите метод `teleport()`. Потеряли лошадь? Вызовите ее метод `getLocation()`.

Не стесняйтесь как можно глубже погрузиться в мир программирования модов Minecraft. Эти знания пригодятся вам при создании серьезных приложений Java.

Вопросы и ответы

В. **Я внес полную сумятицу в мир моего сервера после создания мода `IceCreamScoop`. Могу ли я начать все с начала?**

О. Сервер Minecraft облегчает создание полностью нового мира или переключение между мирами.

Чтобы удалить старый мир и начать все заново, остановите сервер, откройте папку сервера Minecraft и удалите подпапку `world`. После повторного запуска сервера программа Minecraft обнаружит, что мир был удален, и создаст новый мир. Этот процесс происходит довольно медленно. Журнал сервера содержит сообщения, описывающие происходящее.

Чтобы переключиться к новому миру, сохранив старый, остановите сервер и переименуйте подпапку `world`. После этого сервер создаст новый мир. Чтобы вернуться к прежнему миру, необходимо отредактировать файл `server.properties`, находящийся в главной папке сервера. Если вы назвали папку исходного мира `world2`, переключиться к этому миру можно следующим образом: в любом текстовом редакторе откройте файл `server.properties` и измените строку `level-name=world` на строку `levelname= world2`. Сохраните файл и перезапустите сервер.

Коллоквиум

Проверьте знание материала, рассмотренного на этом занятии, ответив на следующие вопросы.

Контрольные вопросы

1. Как называется суперкласс всех модов, созданных для сервера Minecraft на основе библиотеки Spigot?
   - **А.** `Player`.
   - **Б.** `JavaPlugin`.
   - **В.** `Mod`.
2. Какой класс служит для создания спецэффекта, применяемого к мобу?
   - **А.** `Type`.
   - **Б.** `Effect`.
   - **В.** `Potion`.
3. Какой класс представляет каждый блок в игре, заполненный воздухом или другим материалом?
   - **А.** `Location`.
   - **Б.** `Block`.
   - **В.** `Spot`.

Ответы

1. **Б.** Все создаваемые моды включают спецификацию `extends JavaPlugin` в определении класса.
2. **В.** Объект `Potion` создается с указанием значения `PotionEffect`, такого как `SPEED` или `INVISIBILITY`.
3. **Б.** Каждый блок доступен моду с помощью класса `Block`. Ответ **А** практически верен, поскольку класс `Location` представляет местоположение любого блока.

Упражнения

Выполните следующие упражнения, чтобы закрепить изученный материал.

- Создайте мод, созывающий всех цыплят игрового мира к игроку.
- Изучите документацию библиотеки Spigot и создайте мод, который порождает моба, не упомянутого на этом занятии.

ЗАНЯТИЕ 24
Создание приложений для Android

На этом занятии вы узнаете...

- зачем была создана платформа Android;
- как создать приложение для Android;
- как структурировать приложение Android;
- как запустить приложение в среде эмулятора;
- как выполнить приложение на смартфоне Android.

Язык программирования Java является универсальным и может выполняться на множестве платформ.

Одна из новых платформ, появившихся в последние годы, стала чрезвычайно популярной среди программистов на Java. Речь идет об Android, которая начинала свое триумфальное шествие как операционная система для смартфонов, а затем была перенесена на другие устройства. Данная система предназначена для выполнения программ, написанных на Java. Эти программы, называемые *мобильными приложениями*, пишутся на мобильной платформе с открытым исходным кодом, которая полностью бесплатна для разработчиков. Любой желающий может создавать, развертывать и продавать приложения Android.

На этом занятии вы узнаете о том, как появилась операционная система Android, почему она занимает особое положение и почему множество программистов разрабатывают приложения для этой платформы. Вы также напишете приложение и запустите его на смартфоне или эмуляторе Android.

Знакомство с Android

Операционная система Android была создана в 2003 году. В 2007 году она была куплена компанией Google и стала отраслевой инициативой по созданию новой платформы для мобильных телефонов, которая была бы

открытой, не проприетарной (в отличие от технологии, применяемой в RIM BlackBerry и Apple iPhone). Ряд крупнейших компаний в мобильной индустрии (Google, Intel, Motorola, Nvidia, Samsung и др.) сформировали консорциум Open Handset Alliance для продвижения новой платформы на взаимовыгодной основе.

Компания Google выпустила Android Software Development Kit (SDK) — свободно распространяемый набор инструментов, предназначенный для разработки приложений Android. Первый смартфон, работающий под управлением Android, T-Mobile G1, появился в июне 2008 года.

Изначально платформа развивалась медленно, но в 2010 году начался ее взрывообразный рост. Это привело к тому, что платформа Android стала главным конкурентом для iPhone и других мобильных платформ. В настоящее время все основные производители мобильных телефонов предлагают смартфоны Android. Также растет рынок планшетов и электронных книг.

До появления Android для разработки мобильных приложений применялись дорогостоящие инструменты программирования. Производители телефонов имели полное право решать, кому разрешено разрабатывать приложения для их платформы и можно ли продавать приложения пользователям.

С появлением Android статус-кво оказался нарушен. Благодаря открытому и непатентованному характеру Android любой разработчик может создавать, выпускать и продавать мобильные приложения. Следует лишь внести номинальную плату за передачу приложений на рынок Google. Все остальное предоставляется на бесплатной основе.

На сайте Android Developer, доступном по адресу `https://developer.android.com`, можно загрузить Android SDK и найти дополнительные сведения о создании программ для платформы Android. Вы будете часто обращаться к этому сайту при разработке приложений, поскольку там задокументированы все классы из библиотеки классов Java для Android, а также приведено множество справочной информации.

Создание приложений Android упрощается при использовании интегрированной среды разработки (IDE), которая поддерживает Android SDK. Для разработки приложений Android рекомендуется среда Android Studio, которая является бесплатной и имеет открытый исходный код. С помощью Android Studio можно создавать приложения Android, тестировать их на эмуляторе, который ведет себя подобно смартфону Android, и даже развертывать их на физическом устройстве.

До недавних пор язык Java использовался для написания программ, которые могут выполняться на настольном компьютере, на веб-сервере или в браузере. С появлением Android язык Java стал повсеместным. Приложения Java могут развертываться на смартфонах, планшетах и других мобильных устройствах.

Все это соответствует замыслу автора Java, Джеймса Гослинга, который создал этот язык в середине 1990 годов. Ему было поручено написать язык программирования, который мог бы выполняться повсеместно на таких устройствах, как мобильные телефоны, смарт-карты и бытовые электроприборы.

Сначала Java приобрел популярность в качестве средства выполнения интерактивных программ браузером, а затем — в качестве универсального языка программирования.

По оценкам отраслевых экспертов, два десятилетия спустя на платформе Android уже выполняется около 3 млрд приложений Java во всем мире.

Платформа Android останется наиболее популярной и прибыльной областью программирования на Java на долгие годы вперед.

Создание приложения для Android

Приложения Android — это обычные программы на Java, которые используют каркас приложения, представляющий собой базовый набор классов и файлов, присущих всем мобильным приложениям. Каркас строится по определенному набору правил структурирования, обеспечивающих корректное выполнение приложений на устройствах Android.

Для создания приложений нужно установить среду Android Studio. Перейдите на сайт `https://developer.android.com`, загрузите установочный файл, запустите его и следуйте инструкциям, чтобы установить программу на компьютере.

Чтобы начать работу в Android Studio, выполните следующие действия.

1. Запустите Android Studio.
2. В появившемся диалоговом окне выберите параметр Start a New Android Project (Создать новый проект Android). На экране появится окно мастера Create New Project (Создание нового проекта), показанное на рис. 24.1.
3. В поле Application name (Имя приложения) введите **`SalutonMondo`**.
4. В поле Company domain (Домен компании) введите **`java24hours.com`**. Полю Package name автоматически присваивается значение, сформированное путем комбинирования имени пакета и приложения, в данном случае это `com.java24hours.salutonmondo`.
5. Можно принять заданное по умолчанию значение поля Project location (Местоположение проекта) (подпапка папки `AndroidStudioProjects` в вашей пользовательской папке) либо выбрать другую папку с помощью кнопки ..., находящейся справа от поля.
6. Щелкните на кнопке Next (Далее).

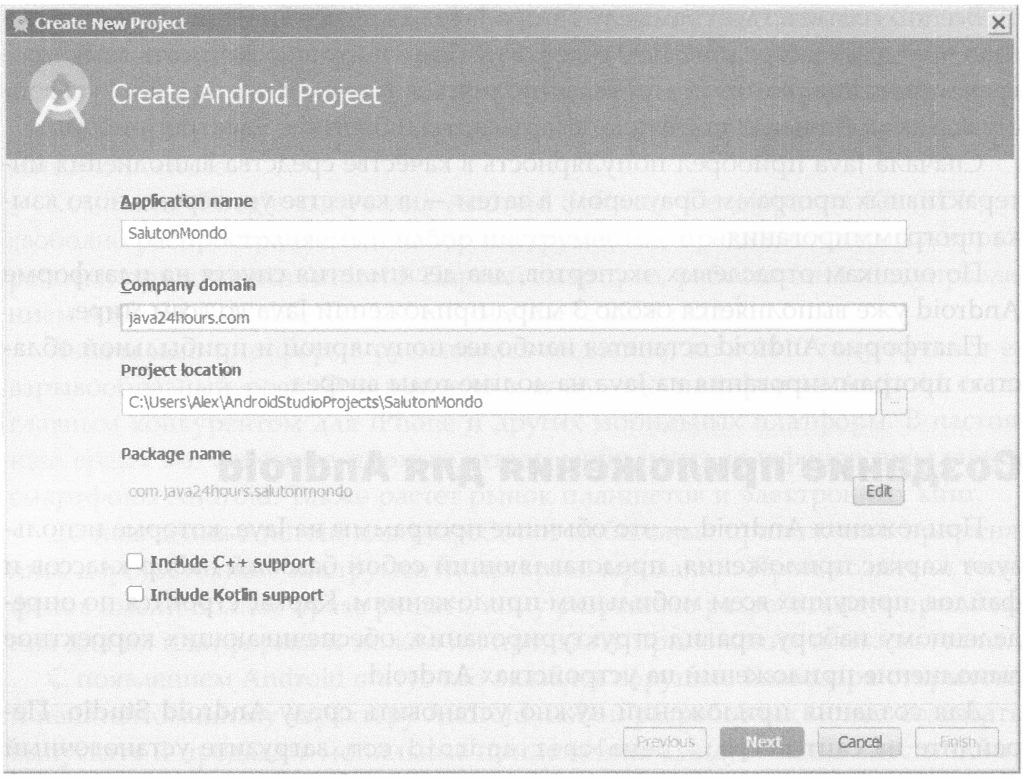


РИС. 24.1. Создание нового проекта в среде Android Studio

7. Каждому проекту Android назначается целевая платформа. В качестве такой платформы выбирается старейшая версия Android, на которой может выполняться приложение. Поскольку каждая новая версия Android включает расширенные средства, результат выбора целевой платформы определяет доступные средства. В качестве цели выберите Phone and Tablet (Телефон и планшет), а в качестве минимальной версии SDK — API 15 (*либо более раннюю версию, если эта недоступна*).
8. Щелкните на кнопке Next. На следующем этапе нужно задать операцию, выполняемую приложением. Выберите Fullscreen Activity (Полноэкранная операция) и щелкните на кнопке Next.
9. В поле Activity Name (Имя операции) введите **`SalutonActivity`**. Это приведет к изменению значений полей Layout Name (Имя компоновки) и Title (Название). Сохраните эти изменения, щелкнув на кнопке Next.
10. Щелкните на кнопке Finish (Готово). В результате будет создано новое приложение, а на панели Package Explorer (Браузер проектов) появится элемент `SalutonMondo`.

Структура нового проекта Android

Новый проект Android состоит примерно из двадцати файлов и папок, которые имеют одинаковую структуру для любого приложения Android. В зависимости от возможностей приложения количество файлов может быть большим, но структура файлов и папок остается неизменной.

На рис. 24.2 показана панель Projects (Проекты) Android Studio сразу же после создания нового проекта. Если эта панель не отображается, щелкните на вкладке Projects, находящейся вдоль левого края окна IDE.

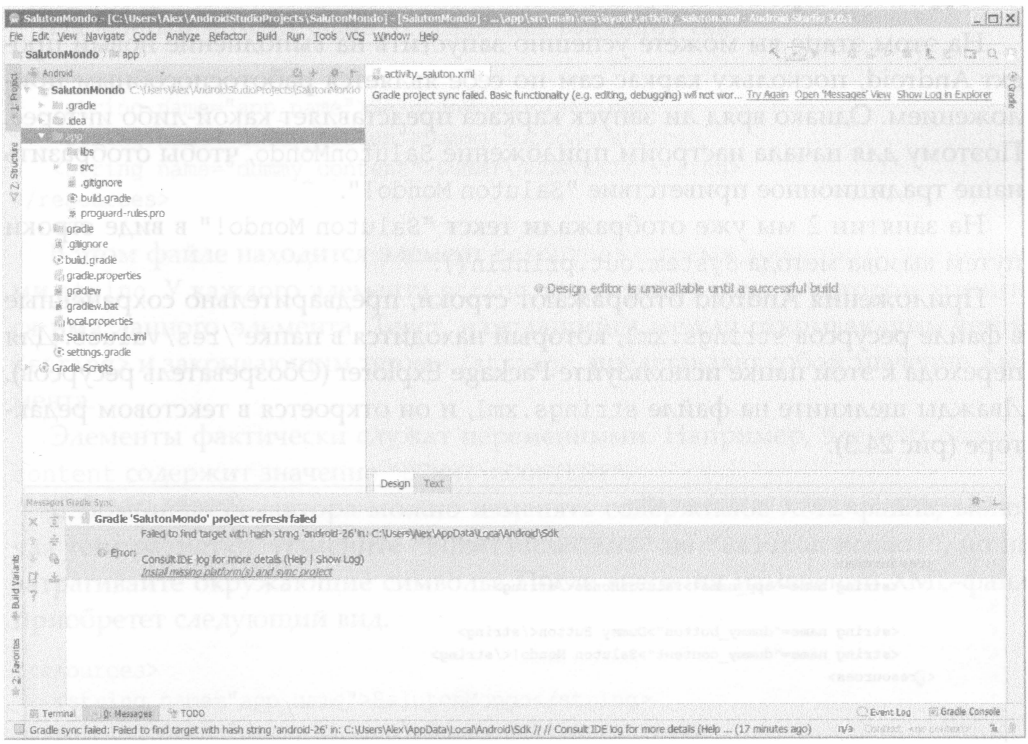


РИС. 24.2. Просмотр составных частей проекта Android

Исследуем структуру файлов и папок проекта. Новое приложение `SalutonMondo` включает следующие компоненты.

- `/javar`. Корневая папка, в которой находится исходный код приложения Java.
- `/src/com.java24hours.salutonmondo/SalutonActivity.java`. Класс, предназначенный для операции, которая выполняется по умолчанию при запуске приложения.
- `/res`. В этой папке находятся ресурсы приложения, такие как файлы данных, файлы компоновки, графика и анимация. Здесь же находятся

подпапки для определенных типов ресурсов: `layout`, `values`, `drawable` и `mipmap`.

- `AndroidManifest.xml`. Основной файл конфигурации приложения.

Эти файлы и папки образуют каркас приложения. Первое, что должен освоить разработчик приложений Android, — научиться изменять каркас. Благодаря этому можно лучше узнать о назначении каждого компонента. Для изменения каркаса нужно включить в проект дополнительные файлы.

Создание приложения

На этом этапе вы можете успешно запустить на выполнение новый проект Android, поскольку каркас сам по себе является работоспособным приложением. Однако вряд ли запуск каркаса представляет какой-либо интерес. Поэтому для начала настроим приложение `SalutonMondo`, чтобы отобразить наше традиционное приветствие `"Saluton Mondo!"`.

На занятии 2 мы уже отображали текст `"Saluton Mondo!"` в виде строки путем вызова метода `System.out.println()`.

Приложения Android отображают строки, предварительно сохраненные в файле ресурсов `strings.xml`, который находится в папке `/res/values`. Для перехода к этой папке используйте Package Explorer (Обозреватель ресурсов). Дважды щелкните на файле `strings.xml`, и он откроется в текстовом редакторе (рис 24.3).

РИС. 24.3. Редактирование строковых ресурсов приложения Android

Подобно любым переменным в Java, строкам и другим ресурсам присваиваются имена и значения. На панели редактирования находятся три строковых ресурса, представленных в виде XML-данных: `app_name`, `dummy_button` и `dummy_content`.

При именовании ресурсов соблюдайте следующие правила:

- все символы дожны быть нижнего регистра;
- пробелы должны отсутствовать;
- в качестве знаков пунктуации можно использовать только символ подчеркивания (_).

Ресурсы находятся в XML-файлах, а в качестве редактора ресурсов используется простой XML-редактор. Можно также непосредственно редактировать саму XML-разметку.

На данный момент файл `strings.xml` выглядит так.

```
<resources>
   <string name="app_name">SalutonMondo</string>
   <string name="dummy_button">Dummy Button</string>
   <string name="dummy_content">DUMMY\nCONTENT</string>
</resources>
```

В этом файле находится элемент `resources` с тремя дочерними элементами `string`. У каждого элемента `string` есть атрибут `name`, в котором хранится имя данного элемента. Текст, находящийся между открывающим тегом `<string>` и закрывающим тегом `</string>`, представляет собой значение элемента.

Элементы фактически служат переменными. Например, элемент `dummy_content` содержит значение `"DUMMY\nCONTENT"`.

С помощью редактора можно изменять содержимое XML-файла, включая теги разметки. Измените `"DUMMY\nCONTENT"` на `"Saluton Mondo!"`, но не затрагивайте окружающие символы. После внесения изменений XML-файл приобретет следующий вид.

```
<resources>
   <string name="app_name">SalutonMondo</string>
   <string name="dummy_button">Dummy Button</string>
   <string name="dummy_content">Saluton Mondo!</string>
</resources>
```

Щелкните на кнопке Save (Сохранить) панели инструментов Android Studio или выполните команду File⇨Save All (Файл⇨Сохранить все), чтобы сохранить изменения в файле `strings.xml`.

После внесения этого изменения вы практически готовы к запуску приложения.

Настройка эмулятора Android

Чтобы получить возможность создавать приложения Android, нужно сконфигурировать среду отладки. Подобную среду можно настроить в Android

Studio. Необходимо установить виртуальное устройство Android (Android Virtual Device — AVD), которое будет запускать приложение на рабочем столе в режиме эмулятора, а также создать конфигурацию отладки проекта. По завершении этих действий можно собрать приложение и запустить его с помощью эмулятора.

Чтобы приступить к конфигурированию виртуального устройства Android, щелкните на зеленом значке смартфона Android на панели инструментов Android Studio (рис. 24.4).

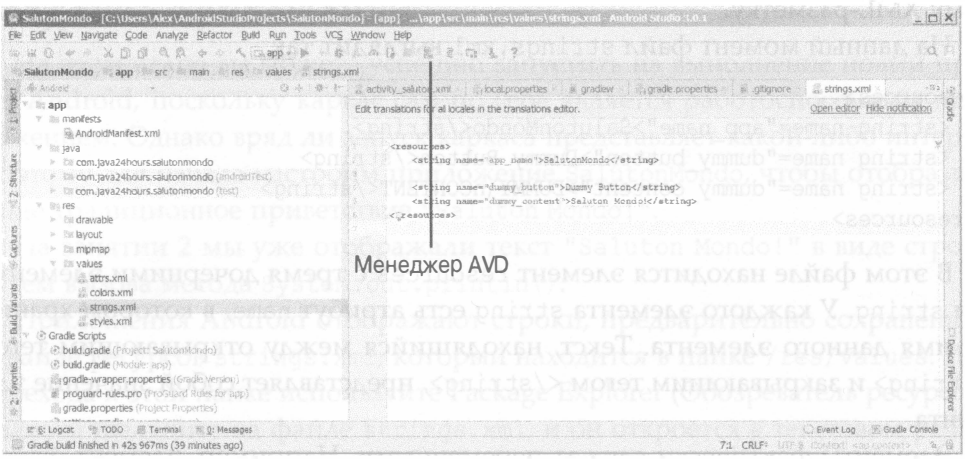


РИС. 24.4. Конфигурирование виртуального устройства Android

На экране появится окно диспетчера виртуальных устройств Android (Android Virtual Device Manager), одного из инструментов, входящих в Android SDK (рис. 24.5).

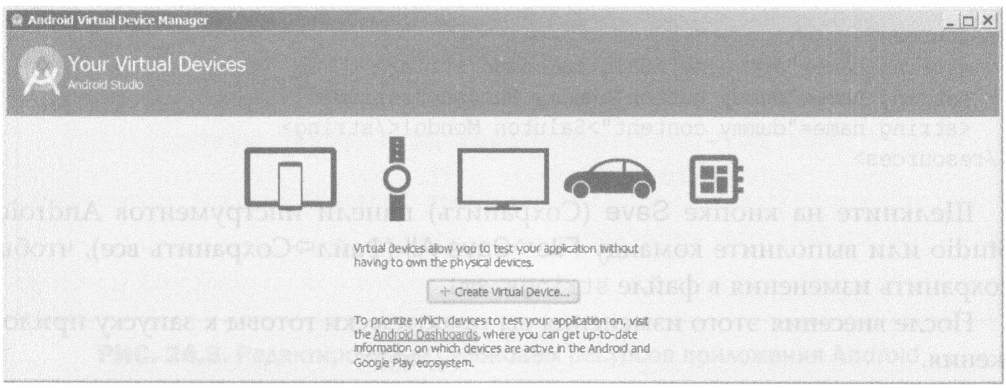


РИС. 24.5. Создание эмулятора Android

Чтобы создать эмулятор, щелкните на кнопке Create Virtual Device (Создать виртуальное устройство) и выполните следующие действия.

1. Выберите устройство в списке, например Galaxy Nexus, и щелкните на кнопке Next (*Далее*).
2. Выберите версию Android в списке, например Nougat, и щелкните на кнопке Next.
3. В поле AVD Name (Имя виртуального устройства Android) укажите имя **`SimpleAVD`**.
4. Щелкните на кнопке Finish (Готово).
5. Для создания эмулятора может понадобиться немного времени, но обычно это не займет больше минуты.

Можно создать произвольное число эмуляторов и настроить их с учетом использования разных версий Android и различных типов экранов.

Закройте окно Android Virtual Device Manager, чтобы вернуться к основному интерфейсу Android Studio.

Запуск приложения

Завершив создание эмулятора Android, запустите первое приложение. Выберите команду Run⇨Run App (Выполнить⇨Выполнить приложение), чтобы открыть диалоговое окно Select Deployment Target (Выбрать цель развертывания). В разделе Available Virtual Devices (*Доступные виртуальные устройства*) выберите `SimpleAVD` и щелкните на кнопке OK. Возможно, понадобится установить платформу, для чего нужно щелкнуть на кнопке Install and Continue (Установить и продолжить).

Эмулятор Android загружается в своем собственном окне. Эта процедура может занять несколько минут, поэтому проявите терпение.

Когда появится установленное приложение вместе со значком Android и подписью SalutonMondo, щелкните на значке для запуска приложения.

Эмулятор отобразит сообщение `"Saluton Mondo!"`, выводимое приложением (рис. 24.6). Работать с эмулятором можно почти так же, как и с обычным смартфоном, только вместо пальцев используется мышь. Щелкните на кнопке возврата, чтобы закрыть приложение и посмотреть, как эмулируется устройство Android.

Эмулятор может выполнять многие функции реального смартфона, включая подключение к Интернету. Эмулятор может также имитировать телефонные звонки и обмен SMS-сообщениями. Но поскольку эмулятор не является полностью функциональным устройством, разрабатываемые приложения следует тестировать на реальных смартфонах и планшетах Android.

Если смартфон (*или другое устройство*) Android можно подключить к компьютеру с помощью USB-кабеля, вы сможете запустить приложение

на смартфоне, находящемся в режиме отладки. Приложения, разработанные с помощью Android SDK, могут развертываться на смартфоне только в режиме отладки.

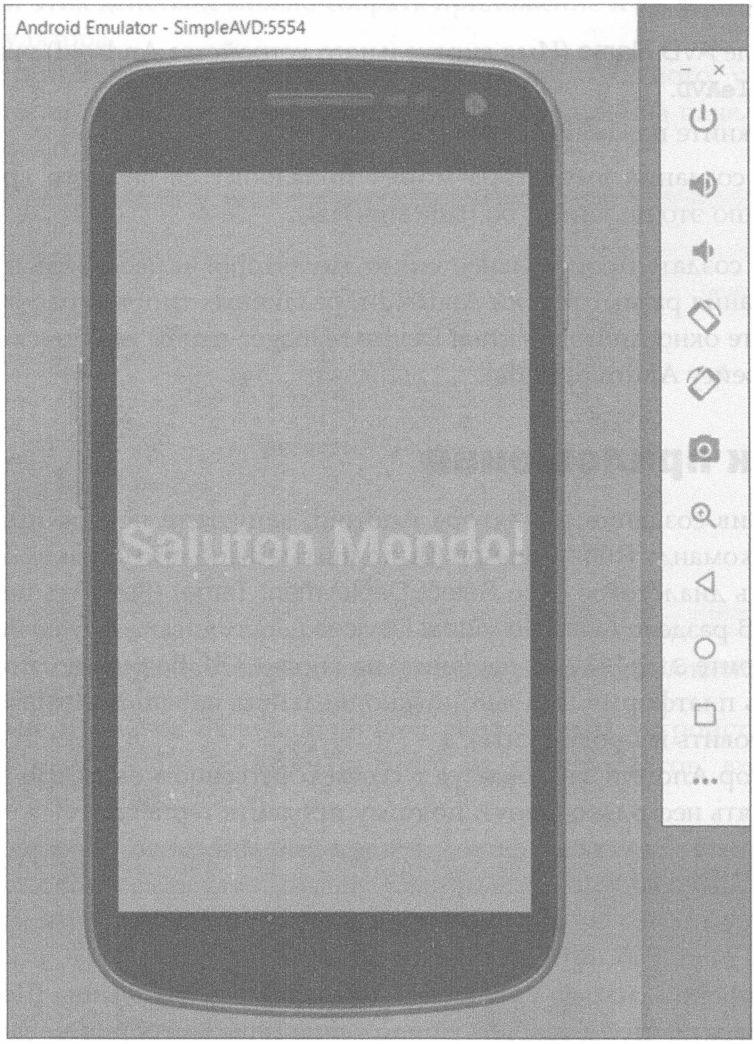


РИС. 24.6. Запуск приложения в среде эмулятора Android

Чтобы переключить смартфон в режим отладки, выполните команду Home⇨Settings⇨Applications⇨Development (Домой⇨Настройки⇨Приложения⇨Разработка). На некоторых смартфонах для переключения в режим отладки используется команда Home⇨Settings⇨Developer Options (Домой⇨Настройки⇨Параметры разработчика). В появившихся на экране настройках Development (Разработка) выберите параметр USB debugging (Отладка USB).

С помощью USB-кабеля подключите смартфон Android к компьютеру. После этого на панели, находящейся в верхней части экрана смартфона, появится значок Android. Если перетащить эту панель вниз, появится сообщение "USB Debugging Connected" (Подключена отладка USB). Затем в Android выполните следующие действия.

1. Выполните команду Run⇨Run App (Выполнить⇨Выполнить приложение).
2. В разделе Connected Devices (Подключенные устройства) выберите ваш смартфон.
3. Щелкните на кнопке OK.

Приложение будет выполняться на смартфоне точно так же, как и на эмуляторе.

Подобно первой программе, написанной на Java (см. занятие 2), ваше первое приложение не слишком полезное. А вот следующий проект будет более практичным.

Закройте текущий проект, выполнив команду File⇨Close Project (Файл⇨Закрыть проект).

Разработка реального приложения

Приложения Android могут использовать все функциональные возможности устройства, такие как обмен SMS-сообщениями, геолокацию и сенсорный ввод. В рамках заключительного проекта книги мы создадим приложение Take Me to Your Leader (Соедините меня с президентом).

Это приложение может использовать все функции смартфона Android, такие как телефонные звонки, просмотр сайтов или нахождение местоположения на картах Google. Оно позволит вам связаться по телефону с президентом США, найти сайт Белого дома и идентифицировать его местоположение на карте. (Если вы живете в другой стране, можно изменить настройки приложения, чтобы получить возможность пообщаться с руководителем своей страны.)

Начните с создания нового проекта в Android Studio.

1. Щелкните на ссылке Start a New Android Studio Project (Начать новый проект Android Studio). Если эта ссылка не отображается, закройте последний проект, выполнив команду File⇨Close Project.
2. В поле Application Name (Имя приложения) введите **Leader**. Это же имя автоматически появится в поле Project Location (Местоположение проекта).

3. В поле **Company Domain** (Домен компании) должно отображаться значение `java24hours.com`. Убедитесь в том, что это действительно так, и щелкните на кнопке **Next**.
4. В разделе **Phone and Tablet** (Телефон и планшет) выберите минимальную версию SDK `API 15`. Если эта версия недоступна, выберите минимальную среди доступных версий и щелкните на кнопке **Next**.
5. Выберите параметр **Fullscreen Activity** (Полноэкранная операция) и щелкните на кнопке **Next**.
6. В поле **Activity Name** (Имя операции) введите `LeaderActivity`.
7. Щелкните на кнопке **Finish**.

Только что созданный проект появится на панели проектов Android Studio.

СОВЕТ

Данный проект требует довольно большой подготовительной работы. По мере выполнения этой работы просматривайте в окне браузера справочный раздел сайта Android Developer, доступный по адресу `https://developer.android.com/reference`. Чтобы получить дополнительные сведения о классах Java в библиотеке классов Android, найдите соответствующие имена файлов в проекте и просмотрите доступную информацию по ним.

Организация ресурсов

Для создания приложения Android необходимо уметь программировать на Java, но большая часть работы выполняется средствами интерфейса Android Studio. Вам достаточно разобраться в возможностях Android SDK, и вы сможете многое сделать, не прибегая к программированию на Java.

Например, без помощи программирования можно создавать ресурсы, которые будут использоваться приложением. Каждый новый проект Android начинается с нескольких папок, в которых находятся ресурсы. Чтобы просмотреть эти папки, откройте папку `Leader` в окне обозревателя пакетов (Package Explorer), а затем папку `/res` и ее подпапки.

К ресурсам относятся изображения в формате PNG, JPG или GIF, строки, хранящиеся в файле `strings.xml`, файлы макета пользовательского интерфейса в формате XML и другие создаваемые вами файлы. Два других файла ресурсов — `colors.xml`, описывающий цвета, применяемые в приложении, и `styles.xml`, определяющий внешний вид компонентов пользовательского интерфейса.

В папке `/res` нового проекта находятся папки `ic_launcher.png` и `ic_launcher_round.png`, в которых хранятся разные версии значка приложения. Значок — это маленький рисунок, применяемый для запуска приложения.

Несколько версий изображения `ic_launcher.png` представляют собой одно и то же изображение, размеры которого различны для отображения на

мониторах с разным разрешением. Эти значки не предназначены для применения пользователями. Новый графический файл, `appicon.png`, будет добавлен в проект и определен в качестве значка в файле `AndroidManifest.xml`, представляющем собой главный файл конфигурации приложения.

На сайте книги, адрес которого был указан во введении, можно найти файл `appicon.png` и четыре других графических файла, используемых для этого приложения: `browser.png`, `maps.png`, `phone.png` и `whitehouse.png`. Откройте сайт книги, перейдите на страницу `Hour 24`, загрузите все пять файлов и сохраните их во временной папке.

Поддержка разных разрешений экрана со стороны Android удобна, но в данном случае не нужна.

Файлы можно добавлять к ресурсам с помощью операций вырезания и вставки. Откройте временную папку, в которой находятся пять файлов, выделите файл `appicon.png` и нажмите комбинацию клавиш <Ctrl+C>, чтобы скопировать его. На панели **Project** (Проект) в среде Android Studio выберите папку `mipmap`, щелкните правой кнопкой мыши и в контекстном меню выберите пункт **Paste** (Вставить). Вернитесь обратно во временную папку, выберите четыре других файла и скопируйте их. После этого в Android Studio выберите папку `drawable` и поместите в нее файлы.

ПРЕДУПРЕЖДЕНИЕ

Для идентификации ресурсов в приложении применяются идентификаторы, созданные на основе имени файла без расширения. Например, файлу `appicon.png` соответствует идентификатор `appicon`, файлу `browser.png` — `browser` и т.п. У двух ресурсов не может быть одинакового идентификатора (исключение составляет изображение, которое в версиях с разным разрешением хранится в папках `mipmap` и рассматривается как один ресурс).

Если два ресурса имеют одно и то же название, но разные расширения, например `appicon.png` и `appicon.gif`, Android Studio расценивает это как ошибку и отказывается компилировать приложение.

Имена ресурсов могут состоять из букв нижнего регистра, цифр, символов подчеркивания (_) и точек (.). Имена файлов для данного проекта выбирались с учетом этих правил.

Теперь ваш проект обзавелся новым значком, который можно использовать в качестве значка приложения. Чтобы изменить значок, следует модифицировать файл `AndroidManifest.xml`.

Конфигурирование файла манифеста приложения

Основной инструмент конфигурирования приложения Android — это файл `AndroidManifest.xml`, который находится в папке приложения `manifests`. Дважды щелкните на XML-файле `AndroidManifest.xml`, чтобы открыть его для редактирования.

Поле `Icon` идентифицирует значок приложения и в данный момент имеет некорректное значение `@mipmap/ic_launcher`.

Найдите следующую строку в файле:

```
android:icon="@mipmap/ic_launcher"
```

Она настраивает значок приложения и содержит имя папки и ресурса. Измените ее следующим образом:

```
android:icon="@mipmap/appicon"
```

Разработка пользовательского интерфейса

Графический интерфейс приложения состоит из компоновок, которые являются контейнерами для виджетов, таких как текстовые поля, кнопки, изображения и пользовательские компоненты. Каждый экран, отображаемый для пользователя, может включать одну или несколько компоновок, содержащихся одна в другой. Компоновки бывают самые разные: одни предназначаются для размещения компонентов по вертикали или по горизонтали, другие — для организации компонентов в виде таблиц или других структур.

Приложение может быть очень простым и состоять из одного экрана, а может включать несколько экранов. Например, игра может быть организована в виде следующих экранов:

- всплывающий экран, который отображается при загрузке игры;
- экран главного меню с кнопками, предназначенный для просмотра других экранов;
- экран справки, где объясняется процесс игры;
- экран достижений, где отображаются наивысшие достижения игроков;
- экран титров, где перечисляются имена разработчиков игры;
- собственно игровой экран.

Приложение `Leader` состоит из одного экрана с кнопками, предназначенными для связи с президентом США или руководителем, имя которого будет названо позже.

Все экраны приложения хранятся в папке `/res/layout`. Наш проект включает файл `activity_leader.xml`, который представляет экран, отображаемый при загрузке приложения.

Чтобы изменить компоновку этого экрана, дважды щелкните на файле `activity_leader.xml` в обозревателе пакетов (Package Explorer). На основной панели редактирования Android Studio откроется соответствующий экран (рис. 24.7).

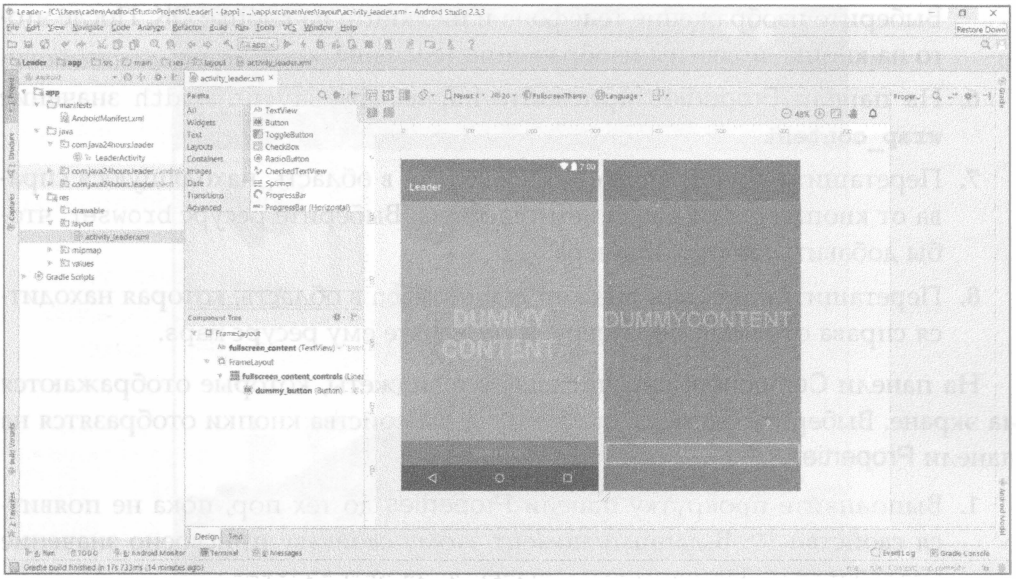


РИС. 24.7. Редактирование графического интерфейса приложения

В окне редактирования находится панель Palette, включающая несколько раскрываемых категорий компонентов пользовательского интерфейса. Эти компоненты можно перетаскивать на экран, находящийся справа.

Чтобы добавить три графические кнопки на экран, выполните следующие действия.

1. Удалите виджет, отображающий текст "Dummy Button". Для этого щелкните на виджете, находящемся в нижней части экрана на панели редактирования, и нажмите клавишу <Delete>.
2. Щелкните на папке Layouts, находящейся на панели Palette, чтобы отобразить подпанель.
3. Перетащите виджет LinearLayout (vertical) из панели Palette на экран, чтобы отобразить панель Properties (Свойства) справа от панели редактирования. Надпись Linear Layout (*Линейная компоновка*) находится над раскрывающимся списком Orientation (Ориентация). Выберите значение vertical (по вертикали), если оно еще не выбрано. Виджет компоновки определяет порядок размещения элементов пользовательского интерфейса, находящихся в этом виджете.
4. На панели Palette щелкните на папке Images, чтобы открыть подпанель.
5. Перетащите виджет ImageButton из панели Palette на экран приложения в панели редактирования. В открывшемся диалоговом окне Resource (Ресурс) отобразится запрос о выборе значка для кнопки.

Выберите изображение телефона и щелкните на кнопке OK. После этого на кнопке появится изображение телефона.

6. На панели Properties присвойте параметру `Layout_Width` значение **`wrap_content`**.
7. Перетащите другой виджет `ImageButton` в область, находящуюся справа от кнопки с изображением телефона. Выберите ресурс `browser`, чтобы добавить кнопку браузера.
8. Перетащите еще один виджет `ImageButton` в область, которая находится справа от кнопки браузера, и назначьте ему ресурс `maps`.

На панели Component перечислены все виджеты, которые отображаются на экране. Выберите элемент `imageButton1`. Свойства кнопки отобразятся на панели Properties.

1. Выполняйте прокрутку панели Properties до тех пор, пока не появится свойство `ID`. В данный момент этому свойству присвоено значение `imageButton1`. Измените это значение на **`phoneButton`**.
2. Свойству `On Click` присвойте значение **`processClicks`**. (Это значение подробно рассматривается в следующем разделе.)
3. Повторите пп. 1 и 2 для виджета `imageButton2`, присвоив свойству `ID` значение **`webButton`**, а свойству `On Click` — значение **`processClicks`**.
4. Повторите пп. 1 и 2 для виджета `imageButton3`, присвоив свойству `ID` значение **`mapButton`**, а свойству `On Click` — значение **`processClicks`**.
5. В дереве компонентов выберите элемент `LinearLayout`. Свойства экрана отобразятся на панели Properties.
6. На панели Properties щелкните на ссылке View All Properties (Просмотр всех свойств).
7. Выберите значение для фона и щелкните на кнопке с изображением многоточия (...), чтобы открыть диалоговое окно Resource.
8. Выберите ресурс `whitehouse` и щелкните на кнопке OK. В результате изображение Белого дома будет выбрано в качестве фона экрана.
9. Щелкните на кнопке Save (Сохранить).

Только что созданный нами экран показан на рис. 24.8.

Написание кода Java

На данный момент вы проделали большую работу по созданию нового приложения, но все еще не написали ни одной строки кода Java. Как видите, разработка приложений существенно упрощается, если использовать по максимуму возможности Android SDK, не прибегая к программированию.

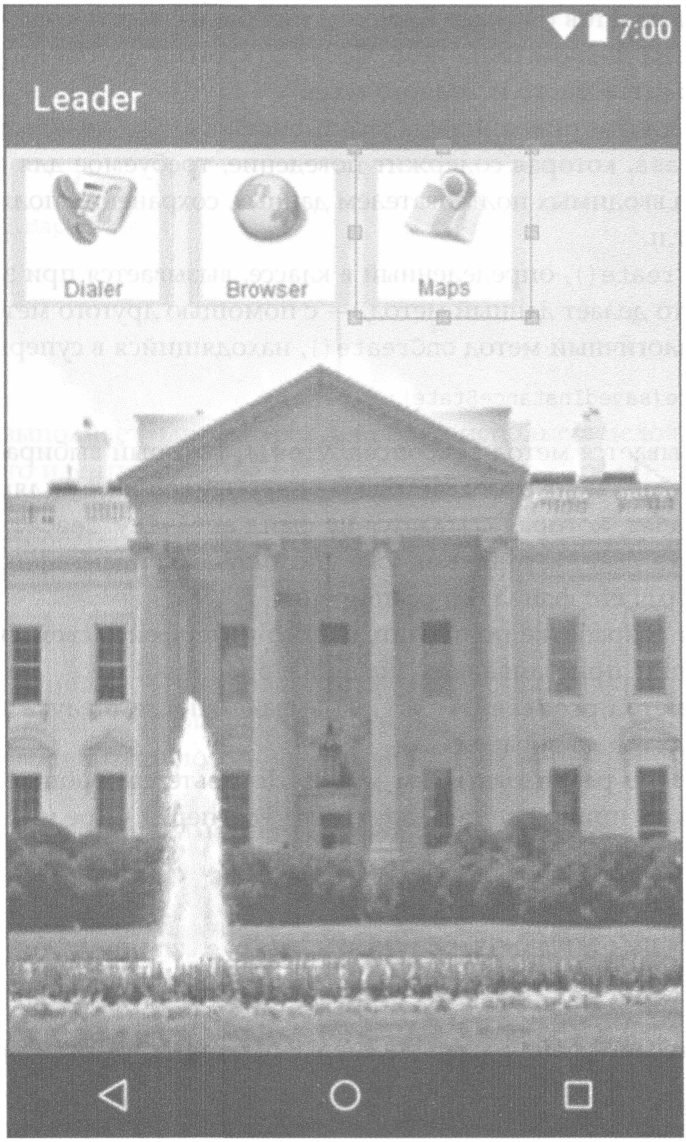


РИС. 24.8. Просмотр графического интерфейса приложения

Приложения представляют собой набор операций, каждая из которых реализует ту или иную функциональную возможность программы. Операция определяется собственным классом Java. В начале проекта вы указали, что должна быть создана операция `LeaderActivity`. При загрузке приложения автоматически запускается класс, соответствующий этому имени.

Исходный код программы `LeaderActivity.java` находится в папке `/java/com.java24hours.leader`, доступ к которой можно получить с помощью

обозревателя пакетов (Package Explorer). Чтобы изменить содержимое этого файла, дважды щелкните на нем. После открытия файла вы увидите, что в классе находится много исходного кода.

Подобно другим операциям, класс `LeaderActivity` является подклассом операции `class`, которая содержит поведение, требуемое для отображения экрана, сбора вводимых пользователем данных, сохранения пользовательских установок и т.п.

Метод `onCreate()`, определенный в классе, вызывается при загрузке класса. Первое, что делает данный метод, — с помощью другого метода, `super()`, вызывает аналогичный метод `onCreate()`, находящийся в суперклассе.

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

Затем вызывается метод `setContentView()`, который выбирает отображаемый экран. Аргумент этого метода — переменная экземпляра `R.layout.activity_leader`, которая ссылается на файл `activity_leader.xml`, находящийся в папке `/res/layout`. Как уже упоминалось, идентификатор (ID) ресурса — это имя его файла без расширения.

Ранее при разработке пользовательского интерфейса свойствам `On Click` каждой кнопки присваивались значения `processClicks`. Это приводило к тому, что метод `processClicks()` вызывался после щелчка пользователя в области виджета на экране.

Теперь нужно реализовать сам метод. Добавьте следующие инструкции в исходный код приложения `LeaderActivity` перед определением метода `onCreate()`.

```
public void processClicks(View display) {
   Intent action = null;
   int id = display.getId();
}
```

В верхнюю часть файла добавьте следующую инструкцию `import`:

```
import android.content.Intent;
```

Теперь вернемся к методу `processClicks()`. Он вызывается с одним аргументом — объектом `View` из пакета `android.view`. Объект `View` — это визуальный дисплей приложения. В нашем случае на этом экране находятся кнопки **Dialer**, **Browser** и **Maps**.

Метод `getId()` объекта `View` возвращает идентификатор кнопки, на которой был выполнен щелчок мышью: `phoneButton`, `webButton` или `mapButton`.

Этот идентификатор хранится в переменной `id` и может использоваться в инструкции `switch` для выбора выполняемого действия. После строки, в которой вызывается метод `display.getId()`, добавьте следующие инструкции.

```
switch (id) {
   case (R.id.phoneButton):
      // ...
      break;
   case (R.id.webButton):
      // ...
      break;
   case (R.id.mapButton):
      // ...
      break;
   default:
      break;
}
```

Этот код выполняет одно из трех действий, используя целочисленное значение каждого идентификатора в качестве условного выражения инструкции `switch`.

Первая инструкция в методе `processClicks()` создает переменную, в которой хранится объект `Intent`, являющийся классом в пакете `android.content`.

```
Intent action;
```

Намерение (intent) — это объект Android, с помощью которого одна операция сообщает другой операции о том, что ей нужно сделать. Также намерения позволяют приложениям обмениваться данными с устройствами Android.

Ниже создаются три намерения, используемые в данном методе.

```
action = new Intent(Intent.ACTION_DIAL, Uri.parse(
   "tel:202-456-1111"));

action = new Intent(Intent.ACTION_VIEW, Uri.parse(
   "http://whitehouse.gov"));

action = new Intent(Intent.ACTION_VIEW, Uri.parse(
   "geo:0,0?q=White House, Washington, DC"));
```

Конструктор `Intent()` использует следующие два аргумента:

- предпринимаемое действие, представленное одной из соответствующих переменных класса;
- данные, связанные с действием.

Эти три намерения указывают устройству Android сделать звонок на телефонную линию Белого дома по номеру (202) 456-1111, перейти на сайт `http://whitehouse.gov` и загрузить карту Google с адресом "White House, Washington, DC" соответственно.

Три конструктора намерений вставляются вместо комментариев в инструкцию `switch`, и каждый из них включается в свой блок условий. Исходящий звонок включается в блок `phoneButton`, посещение сайта — в блок `webButton`, а загрузка карты — в блок `mapButton`.

После создания намерения воспользуйтесь следующей инструкцией для реализации намерения:

```
startActivity(action);
```

Поместите эту инструкцию после инструкции `switch`.

Поскольку фиктивная кнопка (Dummy Button) была удалена из операции, нужно удалить следующую строку кода, которая является последней в методе `onCreate()`:

```
findViewById(R.id.dummy_button).setOnTouchListener(
   mDelayHideTouchListener);
```

Простейший способ удалить эту строку — поместить слева от нее символы `//`. В результате строка превратится в комментарий, игнорируемый компилятором.

Чтобы обеспечить корректное выполнение кода, нужно добавить следующую инструкцию `import`:

```
import android.net.Uri;
```

Полный текст метода `processClicks()` из класса `LeaderActivity` приводится в листинге 24.1. (Учтите, что это не полный исходный код приложения `LeaderActivity.java`, а всего лишь исходный код метода `processClicks()`.)

ЛИСТИНГ 24.1. Фрагмент исходного кода приложения `LeaderActivity.java`

```
 1: public void processClicks(View display) {
 2:    Intent action = null;
 3:    int id = display.getId();
 4:    switch (id) {
 5:       case (R.id.phoneButton):
 6:          action = new Intent(Intent.ACTION_DIAL,
 7:             Uri.parse("tel:202-456-1111"));
 8:          break;
 9:       case (R.id.webButton):
10:          action = new Intent(Intent.ACTION_VIEW,
11:             Uri.parse("http://whitehouse.gov"));
12:          break;
13:       case (R.id.mapButton):
14:          action = new Intent(Intent.ACTION_VIEW,
15:            Uri.parse("geo:0,0?q=White House, Washington, DC"));
16:          break;
17:       default:
```

```
18:             break;
19:     }
20:     startActivity(action);
21: }
```

Чтобы скомпилировать файл, выполните команду Build⇨Make Project (Скомпилировать⇨Создать проект). Если компиляция будет неудачной, отобразятся сообщения об ошибках и красные линии, относящиеся к файлам на панели Project, в которых найдены ошибки.

Чтобы запустить приложение, выберите команду Run⇨Run App (Выполнить⇨Выполнить приложение), а в качестве виртуального устройства укажите SimpleAVD. Это приведет к тому, что эмулятор загрузит, а затем автоматически запустит приложение.

Эмулятор не может имитировать все функции устройства Android. Кнопки Dialer и Browser приложения `Leader` должны работать корректно, однако могут возникнуть проблемы при работе с Картами Google.

Приложение может также выполняться на смартфоне Android, если он поддерживает Android SDK и переключен в режим отладки. Выполните команду Run⇨Run App, в качестве устройства укажите не виртуальное устройство, а ваш смартфон, и щелкните на кнопке OK.

На рис. 24.9 показано приложение, выполняющееся на смартфоне автора. Обратите внимание на то, что, как только ориентация смартфона изменится с портретной на альбомную, соответствующим образом изменится и ориентация экрана приложения.

Приложение `Leader` добавляется в список приложений в виде значка Take Me to Your Leader, который останется на экране смартфона даже после отключения USB-кабеля.

Примите мои поздравления! Мир пополнился еще одним приложением Android.

ПРИМЕЧАНИЕ

Как вы понимаете, невозможно научиться программированию для Android всего лишь за одно занятие. Для более глубокого изучения этой темы обратитесь к специализированным книгам, например к книге *Android. Сборник рецептов: задачи и решения для разработчиков приложений, 2-е издание* (Ян Дарвин).

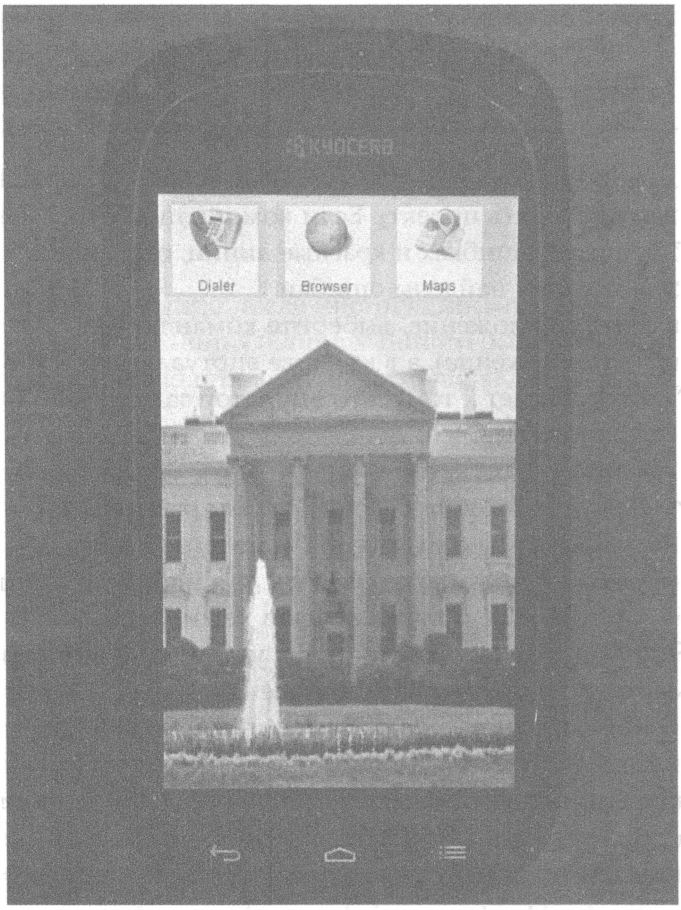


РИС. 24.9. Пришло время позвонить президенту

Резюме

Целью книги было научить читателя концепциям программирования, чтобы он мог создавать собственные программы, способные выполняться на настольном компьютере, веб-сервере и даже на смартфоне. Подход, применяемый в Java, непрост в освоении, но результаты вас приятно обрадуют.

По мере приобретения опыта программирования на Java вы будете повышать свой уровень мастерства в целом, поскольку такие концепции, как объектно-ориентированное программирование, виртуальные машины и эмуляторы, применяются во всех современных средствах разработки.

Если вам еще не надоела эта книга, прочитайте приложения, в которых содержится дополнительная полезная информация.

Дополнительные сведения о Java можно найти во множестве ресурсов. Для программистов имеются полезные блоги `www.journaldev.com` и `https://blogs.oracle.com/java/`. Также можете обратиться на сайт книги, доступный по следующему адресу:

`http://workbench.cadenhead.org/book/java-9-24-hours/`

На этом сайте можно отправить сообщение автору книги и скачать исходные коды примеров.

Если вы хотите расширить и углубить свои познания в Java, обратитесь к книгам, предназначенным для профессионалов, например к серии книг *Java. Библиотека профессионала*.

Итак, вы прочли и освоили материал всех 24 занятий. Теперь вы готовы применить этот материал на практике. Смелее приступайте к делу!

Вопросы и ответы

В. **Почему для разработки приложений Android вместо среды NetBeans используется Android Studio?**

О. Для разработки приложений можно использовать и NetBeans, но эта среда более громоздкая и менее приспособлена для программирования приложений Android. Android Studio позиционируется компанией Google в качестве предпочтительной среды разработки приложений Android. Официальная документация и учебники по Android Studio доступны на сайте `http://developer.android.com`.

В большинстве книг, посвященных программированию для Android, рекомендуется использовать среду Android Studio. Конечно, потребуется определенное время, чтобы перейти от NetBeans к Android Studio. Но как только вы освоите основы создания, отладки и развертывания приложений, вы поймете, что Android Studio легче в использовании, поскольку эта среда намного лучше поддерживается разработчиками и составителями технической документации.

Коллоквиум

Проверьте знание материала, рассмотренного на этом занятии, ответив на следующие вопросы.

Контрольные вопросы

1. Какие из следующих компаний не входят в альянс Open Handset Alliance, занимающийся разработкой и продвижением платформы Android?

А. Google.

Б. Apple.

В. Motorola.

2. Какой объект применяется для передачи инструкций между операциями?

А. `Intent`.

Б. `Action`.

В. `View`.

3. Какие из следующих задач не могут выполняться эмулятором Android?

А. Получение SMS-сообщения.

Б. Подключение к Интернету.

В. Осуществление телефонного звонка.

Ответы

1. **Б.** Apple, поскольку Android создавалась как платформа с открытым исходным кодом и представляла собой непатентованную альтернативу Apple iPhone.
2. **А.** Операции взаимодействуют с устройствами Android с помощью объектов `Intent`.
3. **В.** Эмуляторы не могут заменить реальные устройства, поскольку являются лишь частью инфраструктуры тестирования приложений.

Упражнения

Выполните следующие упражнения, чтобы закрепить изученный материал.

- Измените текст приложения `SalutonMondo` на `"Hello, Android"` и запустите приложение на эмуляторе и устройстве Android (по возможности).
- Создайте новую версию приложения `Take Me to Your Leader`, которая позволит вам связаться с руководителем вашей страны. Вам придется изменить номер телефона, сайт и местоположение на карте.

Часть VII

Приложения

ПРИЛОЖЕНИЕ А

Использование интегрированной среды разработки NetBeans

Программы на Java можно разрабатывать с помощью JDK (Java Development Kit) и текстового редактора, но лучше все же применять интегрированную среду разработки (IDE).

В этой книге преимущественно используется NetBeans — бесплатная среда, предлагаемая компанией Oracle для разработчиков приложений Java. Эта среда упрощает структурирование, написание, компиляцию и отладку программ на Java. Она включает менеджер проектов и файлов, дизайнер графического интерфейса и другие инструменты разработки. Редактор кода позволяет автоматически обнаруживать синтаксические ошибки Java по мере ввода кода.

Среда разработки NetBeans 8.2 пользуется популярностью у профессиональных разработчиков, поскольку предлагаемые ею функциональные возможности и производительность вне всякой конкуренции. Кроме того, она имеет репутацию одной из наиболее простых сред разработки для новичков.

В этом приложении приведены сведения, которые помогут вам установить среду NetBeans и настроить ее для выполнения примеров книги.

Установка NetBeans

Интегрированная среда разработки NetBeans за короткий период прошла путь от никому не известной программы до одного из наиболее популярных наборов инструментов, применяемых для разработки приложений Java. Джеймс Гослинг, автор Java, в своей книге *NetBeans Field Guide* пишет: "Я применяю NetBeans для разработки всех своих приложений Java". Перепробовав много разных сред разработки, появившихся за последние годы, я тоже остановился на NetBeans.

В NetBeans поддерживаются все аспекты программирования на Java для всех трех редакций этого языка: Java Standard Edition (JSE), Java Enterprise Edition (JEE) и Java Mobile Edition (JME). Эта среда также поддерживает

разработку веб-приложений, веб-служб и JavaBeans. В этой книге основное внимание уделяется JSE.

На сайте `https://netbeans.org` можно загрузить версии NetBeans для Windows, Mac OS и Linux. Эта программа доступна для загрузки вместе с JDK либо может быть загружена отдельно. На компьютере должны быть установлены NetBeans и JDK. Среда NetBeans доступна в виде нескольких пакетов — выберите пакет, который поддерживает Java SE.

ПРЕДУПРЕЖДЕНИЕ

Не можете загрузить программу с сайта NetBeans? Возможно, это связано с тем, что вы пытаетесь выполнить загрузку с сайта `www.netbeans.org` (он не работает) вместо `https://netbeans.org`.

Если хотите убедиться в том, что загружаете версии NetBeans и JDK, требуемые для выполнения примеров книги, перейдите на сайт книги по следующему адресу:

`http://workbench.cadenhead.org/book/java-9-24-hours/`

Там вы увидите ссылки Download JDK (Загрузить JDK) и Download NetBeans (Загрузить NetBeans), после щелчка на которых вы перейдете на нужные сайты.

Создание нового проекта

JDK и NetBeans загружаются в виде мастеров установки, которые устанавливают программное обеспечение в вашей системе. Пакет можно установить в произвольную папку или группу меню, но лучше принять настройки по умолчанию, если у вас нет веских оснований отказаться от них.

После установки и первого запуска NetBeans отобразится начальная страница, включающая ссылки на демонстрации и учебные руководства (рис. A.1). Ознакомиться с ними можно в среде разработки, используя встроенный браузер NetBeans.

Проект NetBeans состоит из набора связанных классов Java, файлов, используемых этими классами, и библиотек классов Java. Для каждого проекта выделена его собственная папка. Файлы проекта можно просматривать и изменять вне среды NetBeans с помощью текстовых редакторов и других инструментов программирования.

Чтобы создать новый проект, щелкните на кнопке Создать проект (рис. A.1) либо выполните команду Файл⇨Создать проект. Это приведет к открытию окна мастера Создать проект (рис. A.2).

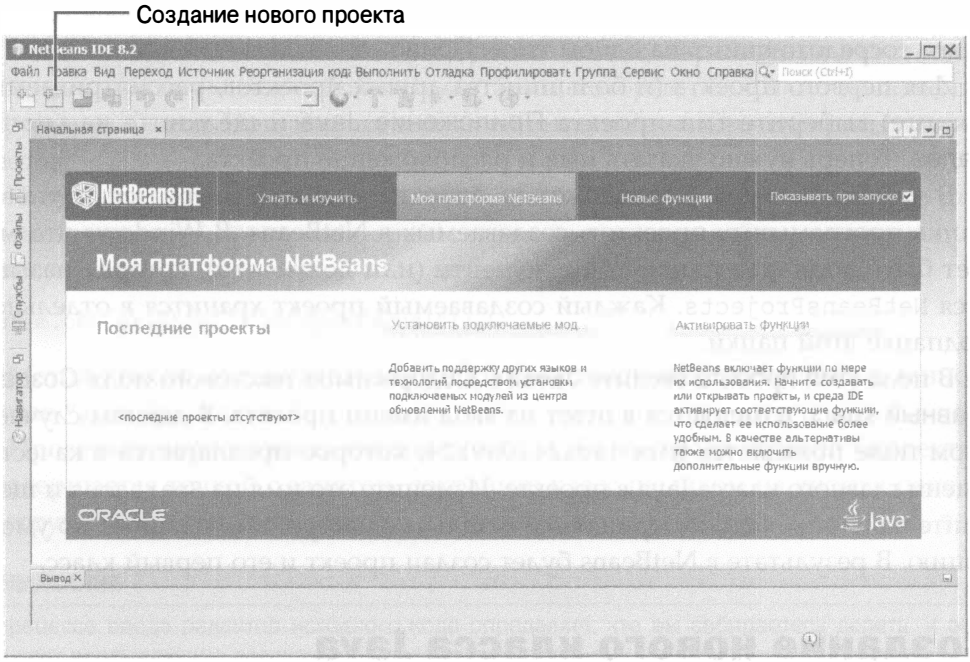


РИС. А.1. Пользовательский интерфейс NetBeans

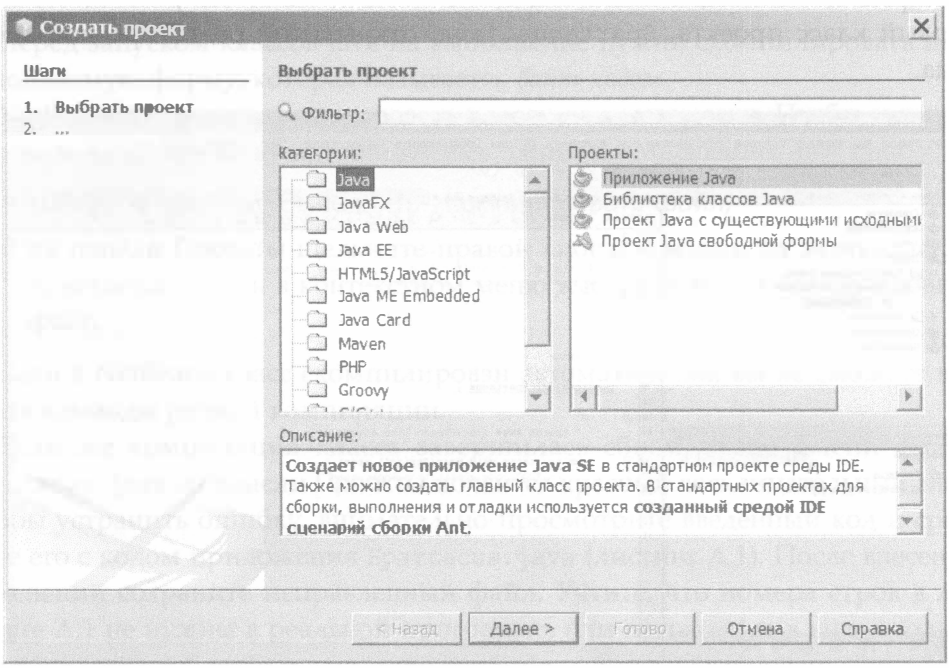


РИС. А.2. Окно мастера Создать проект

В NetBeans можно создавать различные типы проектов Java, но в этой книге мы сосредоточились на одном типе: **Приложение Java**.

Для первого проекта (и большинства других проектов, рассматриваемых в книге) выберите тип проекта **Приложение Java** и щелкните на кнопке **Далее**. Теперь нужно указать имя и расположение проекта.

В текстовом поле **Расположение проекта** идентифицируется корневая папка программных проектов, создаваемых в NetBeans. В Windows это может быть подпапка папки `Мои документы` (или `Документы`), которая называется `NetBeansProjects`. Каждый создаваемый проект хранится в отдельной подпапке этой папки.

В поле **Имя проекта** введите **`Java24`**. Содержимое текстового поля **Создать главный класс** изменяется в ответ на ввод имени проекта. В данном случае в этом поле появляется имя `java24.Java24`, которое предлагается в качестве имени главного класса Java в проекте. Измените это имя на `Spartacus` и щелкните на кнопке **Готово**, приняв все остальные настройки, заданные по умолчанию. В результате в NetBeans будет создан проект и его первый класс.

Создание нового класса Java

После создания в NetBeans нового проекта настраиваются все необходимые файлы и папки, а также создается главный класс. На рис. А.3 показан первый класс проекта, `Spartacus.java`, открытый в редакторе исходного кода.

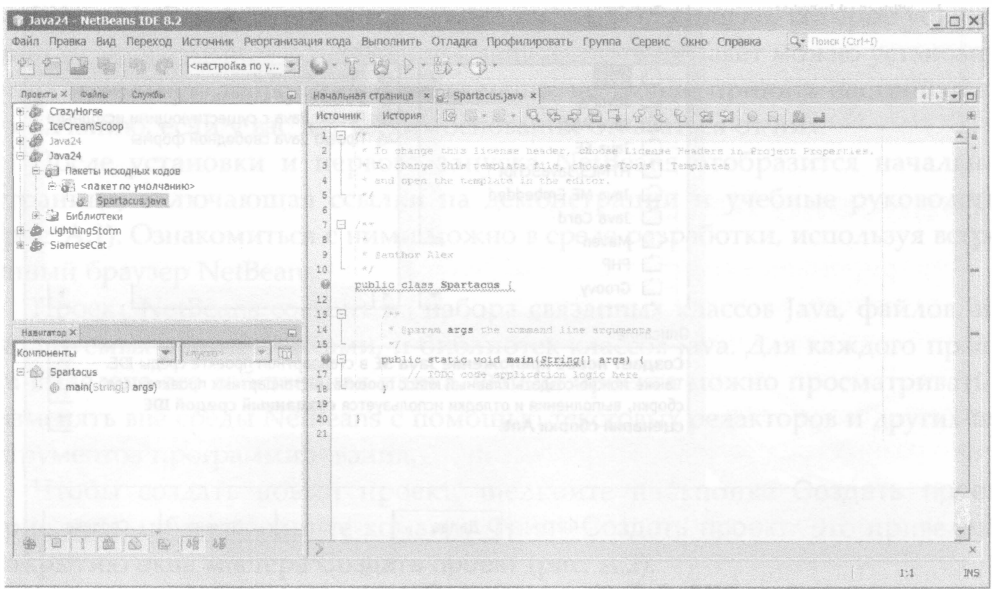


РИС. А.3. Редактор исходного кода NetBeans

На данный момент `Spartacus.java` является каркасом класса Java и состоит лишь из одного метода `main()`. Строки в классе, окрашенные в светло-серый цвет, представляют собой комментарии, которые объясняют назначение и функции класса. При запуске класса на выполнение комментарии игнорируются.

Чтобы предоставить классу возможность что-либо делать, сразу же под комментарием `// TODO code application logic here` добавьте следующую строку кода:

```
System.out.println("I am Spartacus!");
```

Метод `System.out.println()` отображает текстовое сообщение, в данном случае `"I am Spartacus!"`.

Проверьте корректность введенного кода. Если строка введена корректно и завершается точкой с запятой, сохраните класс щелчком на кнопке панели инструментов **Сохранить все** или с помощью команды **Файл⇨Сохранить все**.

ПРИМЕЧАНИЕ

В процессе ввода редактор исходного кода определяет, что вы собираетесь делать, и отображает всплывающие подсказки с полезными сведениями, относящимися к классу `System`, переменной экземпляра `out` или к методу `println()`. Эти подсказки весьма информативны, но в данный момент лучше проигнорировать их.

Перед запуском классов Java на выполнение нужно скомпилировать их в исполняемую форму, которая называется *байт-кодом*.

NetBeans пытается компилировать классы автоматически. Чтобы скомпилировать класс вручную, воспользуйтесь одним из следующих двух способов:

- выберите команду **Выполнить⇨Компилировать файл**;
- на панели **Проекты** щелкните правой кнопкой мыши на значке файла `Spartacus.java` и в контекстном меню выберите пункт **Компилировать файл**.

Если в NetBeans класс скомпилирован автоматически, вы не сможете выбрать команды ручной компиляции.

Если же компиляция класса завершилась сбоем, возле имени файла `Spartacus.java` на панели **Проекты** появится красный восклицательный знак. Чтобы устранить ошибку, внимательно просмотрите введенный код и сравните его с кодом приложения `Spartacus.java` (листинг А.1). После внесения изменений сохраните исправленный файл. Учтите, что номера строк в листинге А.1 не нужны в реальной программе, они приводятся в книге только для того, чтобы облегчить описание кода. (Также в строке 9 вместо слова `User` укажите ваше имя пользователя.)

ЛИСТИНГ А.1. Класс `Spartacus.java`

```
 1: /*
 2:  * To change this license header, choose License Headers in
 3:  * Project Properties. To change this template file, choose
 4:  * Tools | Templates and open the template in the editor.
 5:  */
 6:
 7: /**
 8:  *
 9:  * @author User
10:  */
11: public class Spartacus {
12:
13:    /**
14:     * @param args the command line arguments
15:     */
16:    public static void main(String[] args) {
17:       // TODO code application logic here
18:       System.out.println("I am Spartacus!");
19:
20:    }
21:
22: }
```

Класс Java определен в строках 11–22. Все, что находится выше строки 11, — это комментарий, который добавляется в NetBeans для каждого нового класса в том случае, если в качестве типа проекта выбран вариант Приложение Java. Комментарии включены для удобства читателя, компилятор их игнорирует.

Запуск приложения

После создания класса `Spartacus.java` и успешной компиляции можно запустить программу на выполнение в среде NetBeans. Воспользуйтесь одним из следующих двух способов:

- выберите команду Выполнить⇨Выполнить файл;
- щелкните правой кнопкой мыши на значке файла `Spartacus.java`, находящемся на панели Проекты, и в контекстном меню выберите пункт Выполнить файл.

После запуска класса Java виртуальная машина Java вызывает метод `main()` этого класса, и на панели Вывод отображается строка `"I am Spartacus!"` (рис. А.4).

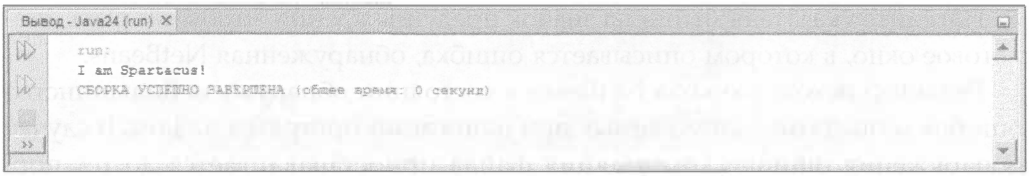


РИС. А.4. Результат выполнения приложения `Spartacus`

Для выполнения класса Java нужен метод `main()`. Если попытаться запустить класс, не имеющий этого метода, NetBeans выдаст сообщение об ошибке.

После просмотра результата выполнения программы закройте панель **Вывод** щелчком на значке ×, находящемся на вкладке панели (справа от названия `Вывод - Java24 (run)`). Это позволит увеличить окно редактора исходного кода, что весьма удобно при написании программ.

Устранение ошибок

Теперь, когда приложение `Spartacus` написано, скомпилировано и запущено на выполнение, рассмотрим, каким образом NetBeans реагирует на появление каких-либо ошибок.

В редакторе исходного кода откройте файл `Spartacus.java` и удалите точку с запятой в строке, в которой вызывается метод `System.out.println()` (строка 18 в листинге А.1). Даже если вы еще не сохранили файл, NetBeans обнаружит ошибку и отобразит красный значок предупреждения слева от строки с ошибкой (рис. А.5).

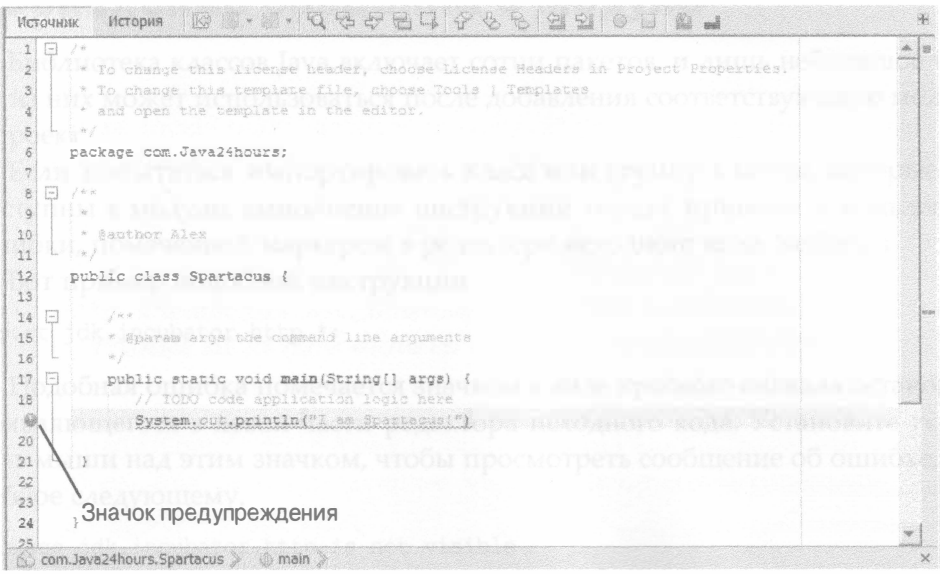


РИС. А.5. Выделение ошибки в редакторе исходного кода

Наведите указатель мыши на значок предупреждения, чтобы открыть диалоговое окно, в котором описывается ошибка, обнаруженная NetBeans.

Редактор исходного кода NetBeans в состоянии обнаружить большинство ошибок и опечаток, допускаемых при написании программ на Java. В случае обнаружения ошибки компиляция файла приостанавливается до тех пор, пока ошибка не будет устранена.

Верните точку с запятой в конец строки. Значок предупреждения исчезнет, и вы снова сможете сохранять и выполнять класс.

Базовых возможностей NetBeans, описанных в этом приложении, вполне достаточно для создания и компиляции программ, рассмотренных в этой книге. Конечно, NetBeans имеет гораздо больше возможностей, но прежде чем приступить к их освоению, следует хорошо изучить Java. Поэтому используйте NetBeans в качестве простого менеджера программных проектов и текстового редактора. Создавайте классы, устраняйте ошибки и проверяйте возможность компиляции и успешного выполнения каждого проекта.

Как только вы будете готовы к дальнейшему освоению NetBeans, воспользуйтесь списком ресурсов на начальной странице NetBeans. Также обратите внимание на список ресурсов, предлагаемый компанией Oracle на сайте `https://netbeans.org/kb`.

ПРИЛОЖЕНИЕ Б
Устранение ошибок, связанных с недоступностью пакетов в NetBeans

Одно из наиболее интересных средств, появившихся в Java 9, — это модули, с помощью которых Java-программы указывают, какие компоненты библиотеки классов Java им нужны и какие пакеты классов они экспортируют.

Модули могут также применяться для получения доступа к пакетам Java, которые обычно недоступны в библиотеке классов. Как было показано на занятии 21, экспериментальная поддержка нового компонента HTTP Client была недоступна в программном проекте до тех пор, пока в проект не был добавлен модуль `jdk.incubator.httpclient`.

В этом приложении описано, что делать в том случае, если при разработке и выполнении проекта Java в NetBeans появляется ошибка, сообщающая о недоступности модуля.

Добавление сведений о модуле

Библиотека классов Java включает сотни пакетов, и лишь небольшое число из них может использоваться после добавления соответствующего модуля в проект.

Если попытаться импортировать класс или группу классов, которые недоступны в модуле, выполнение инструкции `import` приведет к появлению ошибки, помеченной маркером в редакторе исходного кода NetBeans.

Вот пример подобной инструкции:

```
import jdk.incubator.http.*;
```

Подобная ошибка помечается значком в виде красного сигнала остановки, появляющегося в левой части редактора исходного кода. Установите указатель мыши над этим значком, чтобы просмотреть сообщение об ошибке, подобное следующему.

```
Package jdk.incubator.http is not visible
(package jdk.incubator.http is declared in module
jdk.incubator.httpclient, which is not in the module graph)
```

В этом сообщении указывается имя модуля, который нужно добавить в проект, чтобы устранить проблему. В данном случае это модуль `jdk.incubator.httpclient`.

Для настройки модулей применяется класс `module-info`, содержащийся в пакете по умолчанию (другими словами, в классе отсутствует инструкция `import`).

Чтобы добавить модуль в проект `Java24` в среде NetBeans, выполните следующие действия.

- Выполните команду **Файл⇨Создать файл** и на панели **Категории** выберите параметр **Java**.
- На панели **Типы файлов** выберите параметр **Информация о модуле Java**.
- Щелкните на кнопке **Далее**.
- Отобразится диалоговое окно, в котором поле **Имя класса** принимает значение `module-info`. Это значение не может быть изменено пользователем. Щелкните на кнопке **Готово**.

Файл `module-info.java` открывается для редактирования в редакторе исходного кода и включает следующие три инструкции.

```
module Java24 {
   requires jdk.incubator.httpclient;
}
```

За ключевым словом `module` следует слово `Java24`, представляющее собой имя проекта. Если модуль был добавлен в другой проект, используется имя этого проекта.

В блоке `module` в фигурных скобках находится ключевое слово `requires`, за которым следует имя модуля `jdk.incubator.httpclient`. Если же для проекта нужен другой модуль, измените эту инструкцию соответствующим образом.

Проект может включать несколько инструкций `requires`.

После сохранения файла `module-info.java` классы в модуле (или модулях) могут использоваться любой программой Java, входящей в проект. Инструкции `import`, добавляющие пакеты из этого модуля, не вызовут ошибок, и программа сможет быть скомпилирована и запущена.

ПРИЛОЖЕНИЕ В
Устранение проблем при использовании эмулятора Android Studio

Бесплатная интегрированная среда Android Studio приобрела статус официального инструмента разработки приложений Android с момента появления этой платформы в 2015 году.

Создание мобильных приложений Java в этой среде рассматривалось на занятии 24. Если вы уже освоили материал того занятия и успешно выполнили приложение на эмуляторе Android, вам не нужно читать данное приложение. Но если по какой-то причине вы не можете запустить эмулятор, то здесь вы найдете помощь в решении проблемы.

Проблемы, возникающие при выполнении приложения

Если вы работаете с проектом в Android Studio и хотите запустить приложение, выберите команду Run⇨Run App (Выполнить⇨Выполнить приложение).

В открывшемся диалоговом окне Choose Device (Выбрать устройство) запрашивается устройство, на котором будет выполняться приложение. В качестве такого устройства может использоваться эмулятор Android, а также физический телефон или планшет Android, если он подключен к компьютеру через порт USB и настроен для тестирования приложений.

Эмулятор Android в Android Studio ведет себя подобно настоящим смартфонам или планшетам, на которых выполняется мобильная операционная система. Для эмуляции доступны различные виртуальные устройства Android.

Некоторые пользователи сталкиваются с проблемами, когда впервые запускают приложение Android в эмуляторе в Android Studio. Эмулятор завершается со сбоем, отображая следующее довольно неприятное сообщение.

```
ERROR: x86 emulation currently requires hardware acceleration!
Please ensure Intel HAXM is properly installed and usable. CPU
acceleration status: HAX kernel module is not installed!
```

Эта ошибка возникает на компьютерах Windows и указывает на необходимость установки программы от Intel, которая называется HAXM (Hardware Accelerated Execution Manager). Именно она обеспечивает работоспособность эмулятора. Ее можно загрузить в Android Studio, но устанавливается она отдельно.

HAXM представляет собой механизм аппаратной виртуализации, предназначенный для компьютеров с процессорами Intel. Благодаря этому механизму ускоряется работа эмуляторов, что, в свою очередь, способствует ускорению разработки приложений Android. Одно из самых узких мест в программировании для Android заключается в медленной загрузке эмуляторов.

Прежде чем приступать к настройке HAXM, следует добавить ее в набор Android SDK, который входит в Android Studio.

ПРЕДУПРЕЖДЕНИЕ

Учтите, что HAXM должен устанавливаться только на компьютерах с процессором Intel. Это приложение посвящено устранению проблемы, проявляющейся в виде сбоя эмулятора Android из-за отсутствия программы HAXM в Android Studio. Если же эмулятор сбоит по причинам, не связанным с отсутствием HAXM, то это приложение вам не поможет.

Установка HAXM в Android Studio

Программу HAXM можно загрузить и установить в Android SDK в среде Android Studio. Щелкните на кнопке SDK Manager (Диспетчер SDK) панели инструментов Android Studio (рис B.1).

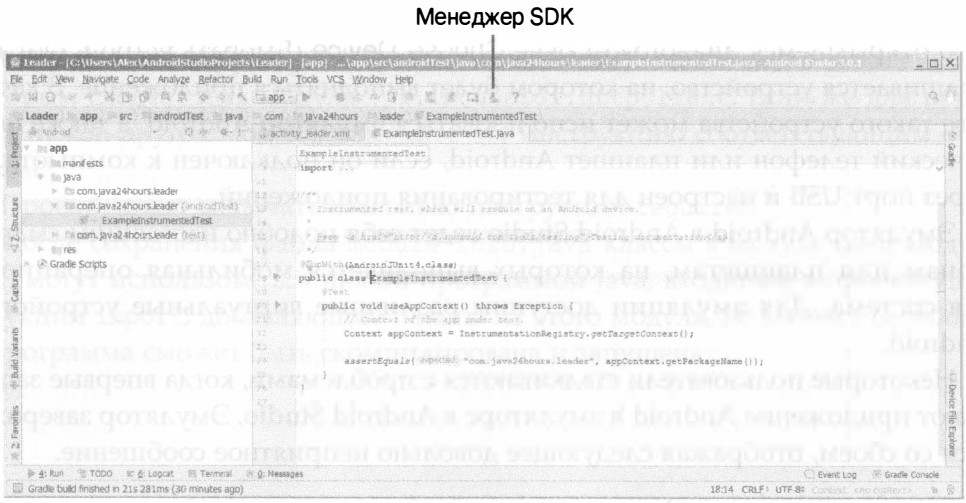


РИС. B.1. Запуск Android SDK Manager

Благодаря SDK Manager можно устанавливать в SDK дополнительные версии Android и полезные инструменты. Щелкните на вкладке SDK Tools (Инструменты SDK), чтобы активизировать ее.

Отобразится перечень инструментов, доступных для SDK. Возле установленных инструментов стоят галочки. Найдите элемент Intel x86 Emulator Accelerator (HAXM Installer). Если возле этого элемента отсутствует галочка, значит, он еще не установлен в Android SDK для вашей копии Android Studio. (Если же галочка имеется, то элемент уже установлен, в таком случае переходите к следующему разделу.)

Окно диспетчера Android SDK Manager показано на рис. В.2.

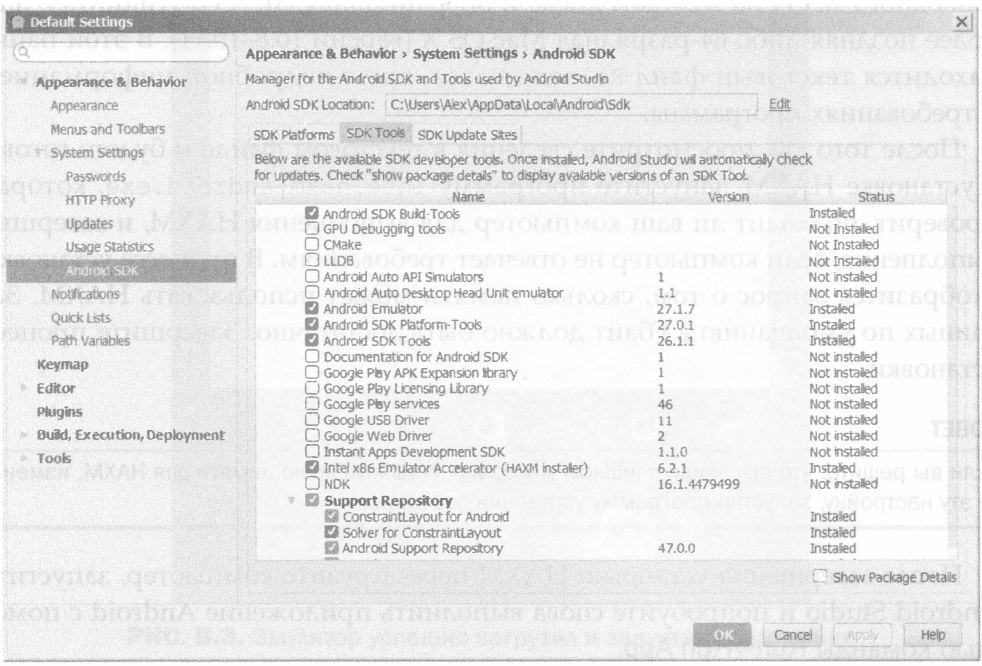


РИС. В.2. Установка инструментов SDK

Выберите элемент Intel x86 Emulator Accelerator (HAXM Installer) и щелкните на кнопке OK. После появления запроса о подтверждении изменения щелкните на кнопке OK.

После этого Android Studio начнет загружать HAXM, отображая отчет о ходе процесса. Если установка завершилась корректно, можете перейти к следующему этапу и установить эту программу на компьютере.

Установка HAXM на компьютере

Чтобы установить HAXM на компьютере, сначала закройте Android Studio. В файловой системе найдите папку, в которой мастер установки Android

Studio сохранил Android SDK. Если вы не помните, в какой папке находится этот файл, учтите, что по умолчанию Windows помещает файл SDK в подпапку `AppData\Local\Android\sdk` личной папки пользователя. И если имя пользователя Windows — `Catniss`, то, скорее всего, файл SDK находится в папке `\Users\Catniss\AppData\Local\Android\sdk`.

Как только вы найдете папку SDK на компьютере, откройте ее, а затем откройте подпапку `extras\intel\Hardware_Accelerated_Execution_Manager`, в которой находится программа `intelhaxm-android.exe` — установочный файл программы HAXM.

Для установки HAXM требуется компьютер с процессором Intel, 1 Гбайт свободного места на жестком диске и операционная система Windows 7 или более поздняя либо 64-разрядная Mac OS X (версии 10.8–10.11). В этой папке находится текстовый файл `Release Notes.txt` с подробной информацией о требованиях программы.

После того как просмотрите сведения в текстовом файле и будете готовы к установке HAXM, запустите программу `intelhaxm-android.exe`, которая проверит, подходит ли ваш компьютер для выполнения HAXM, и завершит выполнение, если компьютер не отвечает требованиям. В процессе установки отобразится запрос о том, сколько памяти может использовать HAXM. Заданных по умолчанию 2 Гбайт должно быть достаточно. Завершите процесс установки.

СОВЕТ

Если вы решите, что выделили слишком много или слишком мало памяти для HAXM, измените эту настройку, запустив программу установки повторно.

После завершения установки HAXM перезагрузите компьютер, запустите Android Studio и попробуйте снова выполнить приложение Android с помощью команды Run⇨Run App.

Приложение будет выполняться в эмуляторе Android, который напоминает смартфон. Он отображает слово `Android` в процессе загрузки, а затем выполняет приложение. На рис. B.3 показано, как выглядит приложение `SalutonMondo` в случае его успешного запуска в эмуляторе.

Если эмулятор успешно загрузил и запустил приложение, можете вернуться к занятию 24.

Если же опять появляется сообщение об ошибке `"ensure Intel HAXM is properly installed and usable"` (обеспечьте корректную установку и использование HAXM), проверьте настройки BIOS и в случае необходимости внесите соответствующие изменения.

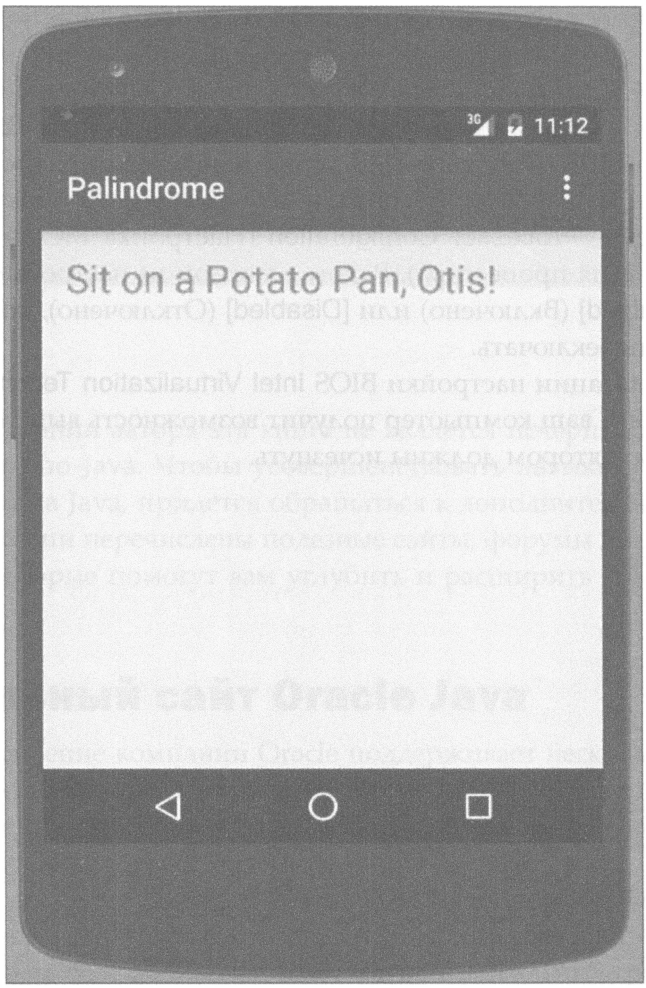


РИС. B.3. Эмулятор успешно загрузил и запустил приложение

Проверка настроек BIOS

Чтобы обеспечить выполнение HAXM, необходимо включить настройку BIOS Intel Virtualization Technology (Технология виртуализации Intel). Если вы имеете определенный опыт работы с BIOS, выбор этой настройки не составит особого труда.

Поскольку изменения в настройках BIOS оказывают влияние на процесс загрузки компьютера и могут вообще привести к тому, что компьютер будет заблокирован, выполняйте их только в том случае, если у вас есть соответствующий опыт. В противном случае обратитесь к услугам специалиста.

Аббревиатура BIOS обозначает утилиту, которая контролирует процесс включения и загрузки компьютера, а также настраивает параметры

компьютерного оборудования. При загрузке компьютера появляется сообщение о том, что можно нажать функциональную клавишу для перехода к настройкам BIOS. Если вы не нажмете указанную клавишу, начнется загрузка Windows. Если же вы нажмете эту клавишу, появится экран BIOS.

На настольном компьютере Dell автора книги для получения доступа к настройке Intel Virtualization Technology нужно выбрать пункты меню BIOS Setup⇨Advanced⇨Processor Configuration (Настройка BIOS⇨Дополнительно⇨Конфигурация процессора). Рядом с каждой из настроек отображаются варианты [Enabled] (Включено) или [Disabled] (Отключено), которые пользователь может переключать.

После активизации настройки BIOS Intel Virtualization Technology и сохранения изменений ваш компьютер получит возможность выполнять HAXM, а проблемы с эмулятором должны исчезнуть.

ПРИЛОЖЕНИЕ Г
Ресурсы, посвященные Java

При всем желании автора эта книга не является исчерпывающим источником сведений по Java. Чтобы усовершенствовать навыки в области программирования на Java, придется обращаться к дополнительным ресурсам. В этом приложении перечислены полезные сайты, форумы Интернета и другие ресурсы, которые помогут вам углубить и расширить знания по языку Java.

Официальный сайт Oracle Java

Java-подразделение компании Oracle поддерживает несколько сайтов, которые будут интересны программистам и пользователям этого языка.

Если вы ищете информацию, имеющую отношение к Java, начните с сайта Oracle Technology Network for Java Developers (`www.oracle.com/technetwork/java`). На нем доступны для скачивания JDK (Java Development Kit) и другие программистские ресурсы, а также документация по библиотеке классов Java. Здесь же есть база данных ошибок, каталог локальных групп пользователей и форумы технической поддержки.

На сайте Java.com размещены материалы, призванные подчеркивать преимущества языка Java для потребителей и рядовых пользователей, которые не являются программистами. На этом сайте можно загрузить среду выполнения Java, которая позволяет запускать программы на Java на компьютерах пользователей. Здесь также доступен раздел справки, посвященный проблемам, возникающим при выполнении приложений Java, и вопросам безопасности.

Документация по классам Java

Возможно, наиболее важным разделом сайта Oracle является раздел документации по всем классам, переменным и методам из библиотеки классов Java. На сайте доступны на бесплатной основе буквально тысячи страниц документации, в которой описывается использование классов в программах.

Документация, описывающая классы в Java 10, доступна по следующему адресу:

```
https://docs.oracle.com/javase/10/docs/api/overview-summary.html
```

Другие сайты, посвященные Java

Огромное количество сайтов в Интернете посвящено Java и программированию на Java.

Сайт книги

Сайт книги описан во введении и доступен по следующему адресу:

```
http://workbench.cadenhead.org/book/java-9-24-hours/
```

Workbench

Автор книги ведет блог Workbench, который посвящен Java, интернет-технологиям, компьютерным книгам и другим темам. Этот сайт доступен по следующему адресу:

```
http://workbench.cadenhead.org
```

Slashdot

Сайт технологических новостей Slashdot пользуется огромной популярностью среди программистов и других компьютерных специалистов, начиная с 1997 года. На первой странице публикуются лучшие истории от пользователей. Также пользователи могут оценивать комментарии, что помогает отсеивать спам. Этот сайт доступен по следующему адресу:

```
http://www.slashdot.org/tag/java
```

Другие блоги по Java

В Интернете есть сотни других блогов, посвященных программированию на Java. Например, на портале WordPress.com доступны тысячи блогов. Последние сообщения, связанные с Java и опубликованные в этих блогах, доступны по следующему адресу:

```
https://en.wordpress.com/tag/java
```

Чтобы найти блоги и новости Java в Твиттере, воспользуйтесь поиском по хеш-тегу `#java`.

```
https://twitter.com/search?q=%23java
```

Учтите, что некоторые результаты поиска будут относиться не к языку программирования, а к марке кофе или острову Ява.

Stack Overflow

Сетевое сообщество Stack Overflow представляет собой место, где программисты могут задавать вопросы и оценивать ответы других пользователей. На этом сайте используются теги, которые позволяют сузить поиск так, чтобы найти интересующие пользователя язык программирования или тему. Раздел этого сайта, посвященный Java, доступен по следующему адресу:

```
https://stackoverflow.com/questions/tagged/java
```

Журнал *JavaWorld*

Этот журнал издается с момента появления языка Java в 1990-х годах. В нем публикуются учебные статьи, новости, касающиеся Java, а также аудио- и видеоподкасты. Этот сайт доступен по следующему адресу:

```
http://www.javaworld.com
```

Каталог ресурсов Java на сайте Developer.com

Поскольку Java является объектно-ориентированным языком, вы сможете легко использовать ресурсы, созданные другими разработчиками, в своих собственных программах. Прежде чем начать проект Java любой сложности, просмотрите веб-ресурсы, которые вы могли бы использовать в своей программе.

Хорошее место для начала — каталог ресурсов Java на сайте Developer.com. На этом сайте собраны программы на Java, материалы для программистов и другая подобная информация. Этот сайт доступен по следующему адресу:

```
https://www.developer.com/java/
```

Java-тусовки

На сайте Meetup.com содержатся сведения о тысячах встреч, которые организуются формальными и неформальными группами, объединяющими программистов на Java.

На сайте анонсировано множество групп пользователей Java и других событий, связанных с Java. Чтобы найти встречу, которая будет проведена не далее чем в 100 км от вас, перейдите по следующей ссылке:

```
https://www.meetup.com/ru-RU/find/?keywords=Java&radius=100
```

Предметный указатель

П

Р

С

Т

JAVA SE 9
БАЗОВЫЙ КУРС
ВТОРОЕ ИЗДАНИЕ

Кей С. Хорстманн

В этом кратком руководстве рассматриваются основные языковые средства и нововведения в версии Java SE 9, главным из которых является новая модульная система на платформе Java. На практических примерах исходного кода поясняются основные языковые средства, новшества, усовершенствования и прочие незначительные изменения в версии Java SE 9. Книга рассчитана как на начинающих, так и на опытных программистов, стремящихся писать в недалекой перспективе надежный, эффективный и безопасный код на Java.

www.williamspublishing.com

ISBN 978-5-6040043-0-2 **в продаже**

JAVA
ЭФФЕКТИВНОЕ ПРОГРАММИРОВАНИЕ
ТРЕТЬЕ ИЗДАНИЕ

Джошуа Блох

www.dialektika.com

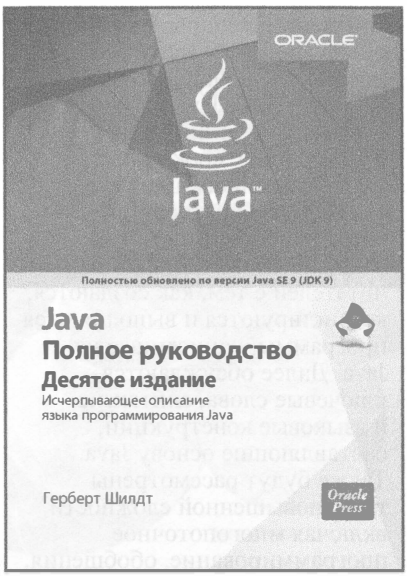
Говоря о третьем издании книги *Java: эффективное программирование*, достаточно упомянуть ее автора, Джошуа Блоха, и это будет наилучшей ее рекомендацией.
Книга представляет собой овеществленный опыт ее автора как программиста на Java. Новые возможности этого языка программирования, появившиеся в версиях, вышедших со времени предыдущего издания книги, по сути, знаменуют появление совершенно новых концепций, так что для их эффективного использования недостаточно просто узнать об их существовании и программировать на современном Java с использованием старых парадигм.
К программированию в полной мере относится фраза Евклида о том, что в геометрии нет царских путей. Но пройти путь изучения и освоения языка программирования вам может помочь проводник, показывающий наиболее интересные места и предупреждающий о ямах и ухабах. Таким проводником может послужить книга Джошуа Блоха. С ней вы не заблудитесь и не забредете в дебри, из которых будете долго и трудно выбираться с помощью отладчика.

ISBN 978-5-6041394-4-8

в продаже

JAVA
ПОЛНОЕ РУКОВОДСТВО
ДЕСЯТОЕ ИЗДАНИЕ

Герберт Шилдт



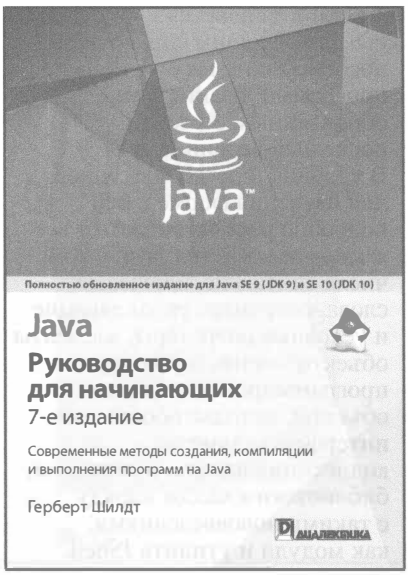
www.dialektika.com

Эта книга является исчерпывающим справочным пособием по языку программирования Java, обновленным с учетом последней версии Java SE 9. В удобной и легко доступной для изучения форме в ней подробно рассматриваются все языковые средства Java, в том числе синтаксис, ключевые слова, операции, управляющие и условные операторы, элементы объектно-ориентированного программирования (классы, объекты, методы, обобщения, интерфейсы, пакеты, коллекции), аплеты и сервлеты, библиотеки классов наряду с такими нововведениями, как модули и утилита JShell. Основные принципы и методики программирования на Java представлены на многочисленных и наглядных примерах написания программ. Книга рассчитана на широкий круг читателей, интересующихся программированием на Java.

ISBN 978-5-6040043-6-4 **в продаже**

JAVA: РУКОВОДСТВО ДЛЯ НАЧИНАЮЩИХ
7-Е ИЗДАНИЕ

Герберт Шилдт



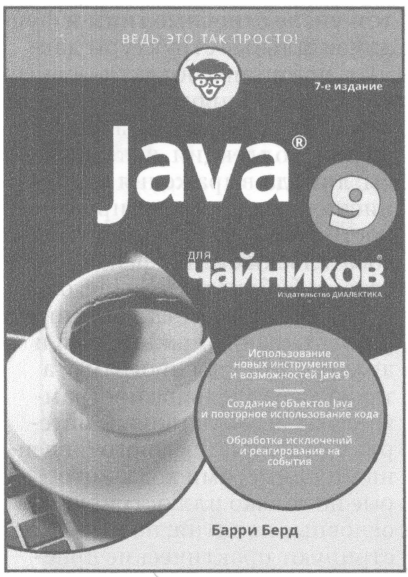
www.dialektika.com

Очередное издание бестселлера, обновленное с учетом всех новинок Java Platform, Standard Edition 9 и 10 (Java SE 9 и 10), позволит читателям в кратчайшие сроки приступить к программированию на языке Java. Опытнейший автор Герберт Шилдт уже в начале книги познакомит читателей с тем, как создаются, компилируются и выполняются программы, написанные на Java. Далее обсуждаются ключевые слова, синтаксис и языковые конструкции, составляющие основу Java. Также будут рассмотрены темы повышенной сложности, включая многопоточное программирование, обобщения, лямбда-выражения, Swing, JavaFX и ключевое нововведение Java SE 9 — модули. В качестве бонуса читателей ждет знакомство с JShell — новой интерактивной оболочкой Java.

ISBN 978-5-6041394-5-5 **в продаже**

JAVA ДЛЯ ЧАЙНИКОВ
7-Е ИЗДАНИЕ

Барри Берд



www.dialektika.com

Перед вами бестселлер для начинающих, посвященный Java 9 — новой версии самого мощного объектно-ориентированного языка программирования. Программа, написанная на Java, будет выполняться практически на любом компьютере, ноутбуке или портативном устройстве. Освоив Java, вы сможете создавать мультимедийные приложения, предназначенные для любой платформы. Теперь пришел ваш черед! Независимо от того, на каком языке вы программировали раньше (и даже если вы никогда прежде не программировали), вы быстро научитесь создавать современные кроссплатформенные приложения, используя возможности Java 9.

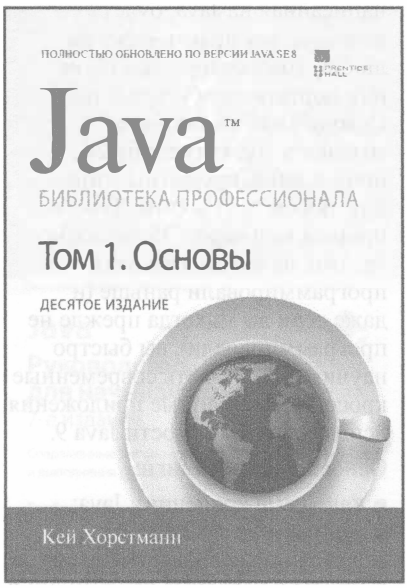
Основные темы книги:

- ключевые концепции Java;
- грамматика языка;
- повторное использование кода;
- циклы и условные конструкции;
- основы объектно-ориентированного программирования;
- обработка исключений;
- использование ссылочных типов данных.

ISBN 978-5-9500296-1-5 **в продаже**

JAVA®
БИБЛИОТЕКА ПРОФЕССИОНАЛА
Том 1. Основы
Десятое издание

Кей Хорстманн



www.williamspublishing.com

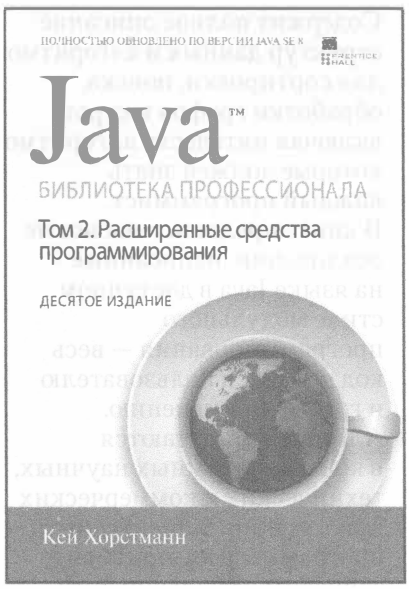
Это первый том обновленного, десятого издания исчерпывающего справочного руководства по программированию на Java с учетом всех нововведений в версии Java SE 8. В нем подробно рассматриваются основы программирования на Java, в том числе основные типы и фундаментальные структуры данных, принципы объектно-ориентированного программирования и его реализация в Java, обобщения, коллекции, интерфейсы, лямбда-выражения и функциональное программирование, построение графических пользовательских интерфейсов средствами библиотеки Swing, обработка событий и исключений, развертывание приложений и аплетов, отладка программ, а также параллельное программирование. Излагаемый материал дополняется многочисленными примерами кода, которые не только иллюстрируют основные понятия, но и демонстрируют практические приемы программирования на Java.

Книга рассчитана на программистов разной квалификации и будет также полезна студентам и преподавателям дисциплин, связанных с программированием на Java.

ISBN 978-5-8459-2084-3 **в продаже**

JAVA™
БИБЛИОТЕКА ПРОФЕССИОНАЛА
Том 2. Расширенные средства программирования
Десятое издание

Кей Хорстманн



www.williamspublishing.com

Это второй том обновленного, десятого издания исчерпывающего справочного руководства по программированию на Java с учетом всех нововведений в версии Java SE 8. В этом томе подробно рассматриваются расширенные средства программирования на Java, в том числе потоки данных, файловый ввод-вывод, XML, манипулирование датами и отметками времени, сетевое программирование и базы данных, интернационализация прикладных программ, расширенные функциональные возможности библиотек Swing и AWT, обеспечение безопасности, обработка аннотаций, оперирование платформенно-ориентированными методами. Излагаемый материал дополняется многочисленными примерами кода, которые не только иллюстрируют поясняемые понятия, но и демонстрируют практические приемы усовершенствованного программирования на Java.

Книга рассчитана на программистов разной квалификации и будет также полезна студентам и преподавателям дисциплин, связанных с программированием на Java.

ISBN 978-5-9909445-0-3

в продаже

АЛГОРИТМЫ НА JAVA
4-Е ИЗДАНИЕ

Роберт Седжвик,
Кевин Уэйн

www.williamspublishing.com

Последнее издание из серии бестселлеров Седжвика, содержащее самый важный объем знаний, наработанных за последние несколько десятилетий. Содержит полное описание структур данных и алгоритмов для сортировки, поиска, обработки графов и строк, включая пятьдесят алгоритмов, которые должен знать каждый программист.
В книге представлены новые реализации, написанные на языке Java в доступном стиле модульного программирования — весь код доступен пользователю и готов к применению.
Алгоритмы изучаются в контексте важных научных, технических и коммерческих приложений. Клиентские программы и алгоритмы записаны в реальном коде, а не на псевдокоде, как во многих других книгах.
Дается вывод точных оценок производительности на основе соответствующих математических моделей и эмпирических тестов, подтверждающих эти модели.

ISBN 978-5-8459-2049-2

в продаже

Java™

8-е издание

за 24 часа

Программировать на Java гораздо проще, чем вы думаете, особенно если у вас под рукой эта книга. Всего лишь за 24 занятия длительностью не более одного часа каждое вы научитесь создавать Java-приложения на весьма достойном уровне.

Выполняя понятные пошаговые инструкции, вы получите знания и опыт, необходимые для разработки компьютерных программ и веб-приложений на Java, научитесь создавать приложения для Android и даже моды для Minecraft.

Каждое занятие служит продолжением предыдущего, позволяя шаг за шагом приобретать необходимые навыки.

- Наглядные иллюстрации и пошаговые инструкции позволят четко понять, как работают программы на Java.
- Коллоквиумы и упражнения в конце каждого занятия помогут проверить уровень полученных знаний.
- Примечания, советы и предупреждения предоставят полезную информацию.

Об авторе

Роджерс Кейденхед — писатель, программист и разработчик веб-приложений. Автор более 25 книг, посвященных Java и Интернету. Активный блогер и администратор нескольких популярных сайтов.

Основные темы книги:

- настройка среды программирования Java
- создание первой рабочей программы за несколько минут
- управление поведением программы
- сохранение данных и работа с файлами
- создание простого пользовательского интерфейса
- создание интерактивных веб-приложений
- использование потоков для создания производительных программ
- чтение/запись XML-данных
- приемы объектно-ориентированного программирования
- использование HTTP-клиента
- создание приложений для Android

Категория: программирование
Предмет рассмотрения: Java
Уровень: начальный/средний

ISBN: 978-5-6041394-6-2

9 785604 139462 18065

Диалектика
www.williamspublishing.com

Pearson