

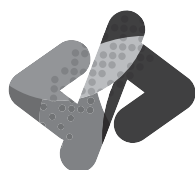


JAVA

**ПРОГРАММИРОВАНИЕ
ДЛЯ НАЧИНАЮЩИХ**

ПЕРВЫЙ ШАГ НА ПУТИ К УСПЕШНОЙ КАРЬЕРЕ

> Пятое издание



**ПРОГРАММИРОВАНИЕ
ДЛЯ НАЧИНАЮЩИХ**

Mike McGrath

JAVA PROGRAMMING EASY STEPS

5th Edition

Майк МакГрат

ПРОГРАММИРОВАНИЕ НА JAVA



Москва
2016

УДК 004.43
ББК 32.973.26-018.1
М15



Mike McGrath
JAVA IN EASY STEPS, 5TH EDITION

By Mike McGrath. Copyright ©2015 by In Easy Steps Limited.
Translated and reprinted under a licence agreement from the Publisher:
In Easy Steps, 16 Hamolton Terrace, Holly Walk, Leamington Spa, Warwickshire, U.K. CV32 4LY.

МакГрат, Майк.

М15 Программирование на Java для начинающих / Майк МакГрат ; [пер. с англ. М.А. Райтмана]. – Москва : Издательство «Э», 2016. – 192 с. – (Программирование для начинающих).

ISBN 978-5-699-85743-2

Книга «Программирование на Java для начинающих» является исчерпывающим руководством для того, чтобы научиться программировать на языке Java.

В этой книге с помощью примеров программ и иллюстраций, показывающих результаты работы кода, разбираются все ключевые аспекты языка. Установив свободно распространяемый Java Development Kit, вы с первого же дня сможете создавать свои собственные исполняемые программы!

УДК 004.43
ББК 32.973.26-018.1

ISBN 978-5-699-85743-2

© Райтман М.А., перевод на русский язык, 2016
© Оформление. ООО «Издательство «Э», 2016

Оглавление

Предисловие

8

1

Введение

9

Установка JDK	12
Создание первой программы на Java.	14
Компиляция и запуск программ	16
Создание переменных.	18
Распознавание типов данных	20
Создание констант	22
Добавление комментариев	23
Проблемы отладки	24
Заключение.	26

2

Выполнение операций

27

Выполнение арифметических операций	28
Присваивание значений	30
Сравнение величин	32
Оценочная логика	34
Проверка условий	36
Приоритет операций	38
Управляющие литералы	40
Работа с битами	42
Заключение.	44

3

Создание операторов

45

Ветвление с помощью условного оператора <code>if</code>	46
Альтернативное ветвление	48
Ветвление с помощью переключателей	50
Цикл <code>for</code>	52
Цикл <code>while</code>	54
Циклы <code>do-while</code>	56
Выход из циклов.	58
Возврат управления	60
Заключение.	62

Преобразование типов	64
Создание массивов переменных.	66
Передача аргументов	68
Передача множественных аргументов.	70
Обход элементов в цикле.	72
Изменение значений элемента	74
Добавление размеров массива.	76
Перехват исключений	78
Заключение.	80

Изучение классов Java.	82
Математические вычисления	84
Округление чисел.	86
Генерация случайных чисел	88
Управление строками	90
Сравнение строк	92
Поиск строк.	94
Обработка символов.	96
Заключение.	98

Программа как набор методов	100
Область видимости.	102
Использование множественных классов	104
Расширение существующего класса	106
Создание объектного класса	108
Создание экземпляра объекта	110
Инкапсуляция свойств	112
Создание объектных данных	114
Заключение.	116

Работа с файлами	118
Чтение консольного ввода.	120
Чтение файлов.	122
Запись файлов.	124
Сортировка элементов массива	126
Создание списочных массивов	128

Работа с датой	130
Форматирование чисел.	132
Заключение.	134

8 Построение интерфейсов 135

Создание окна	136
Добавление кнопок	138
Добавление меток	140
Добавление текстовых полей	142
Добавление элементов выбора.	144
Добавление переключателей	146
Изменение внешнего вида интерфейса	148
Размещение компонентов	150
Заключение.	152

9 Распознавание событий 153

«Прослушивание» событий	154
Генерация событий.	155
Обработка событий кнопок	156
Обработка событий элементов	158
Реагирование на события клавиатуры	160
Ответ на события мыши.	162
Вывод сообщений	164
Запрос пользовательского ввода	166
Воспроизведение звука.	168
Заключение.	170

10 Развертывание программ 171

Методы развертывания.	172
Распространение программ.	174
Построение архивов.	176
Развертывание приложений	178
Подписывание jar-файлов	179
Использование технологии Web Start	180
Создание апплетов.	182
Встраивание апплетов в код веб-страницы.	184
Развертывание апплетов	186
Заключение.	188

Предметный указатель 189

Предисловие

В этой книге мои предыдущие публикации по Java-программированию дополнены новинками данной технологии. Все приведенные здесь примеры демонстрируют возможности Java, которые поддерживаются современными компиляторами в операционных системах Windows и Linux, а представленные скриншоты отражают реальные результаты компиляции и исполнения приведенного кода.

Некоторые соглашения

Листинги кода, приведенные в книге, выглядят вот так:

```
// Примечание к коду
```

```
String message = "Добро пожаловать в язык Java!" ;
```

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

Кроме того, для идентификации каждого исходного файла, описанного в пошаговых инструкциях, на полях рядом с каждым пунктом будет появляться значок и имя соответствующего файла:



App.java



App.class



App.jar



App.jnlp

Получение исходных кодов

Для удобства я поместил файлы исходных кодов всех примеров, представленных в этой книге, в один ZIP-архив. Вы можете получить весь архив, выполнив следующие простые шаги.

1. Откройте браузер и загрузите архив по ссылке www.eksmo.ru.
2. Извлеките содержимое архива в любое удобное место на вашем компьютере.

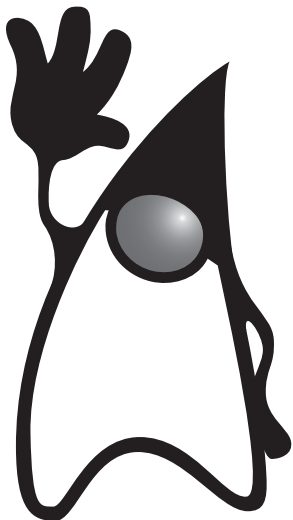
Я искренне надеюсь, что вам понравится открывать для себя интересные возможности языка программирования Java и вы получите при этом не меньшее удовольствие, чем я во время работы над этой книгой.

1

Введение

Добро пожаловать в удивительный мир программирования на Java. В этой главе демонстрируется, как создавать и запускать на выполнение простые Java-программы и как хранить в них данные.

- Установка JDK
- Создание первой программы на Java
- Компиляция и запуск программ
- Создание переменных
- Распознавание типов данных
- Создание констант
- Добавление комментариев
- Проблемы отладки
- Заключение



Язык программирования Java разработан в 1990 году Джеймсом Гослингом (James Gosling), инженером из компании Sun Microsystems. Он решил создать новый язык, так как был не очень доволен используемым языком программирования C++, и назвал его сначала Oak* — в честь дуба, который он мог наблюдать из окна своего офиса.

По мере роста популярности Всемирной паутины компания Sun предположила, что язык, разработанный Гослингом, может быть использован для интернет-разработок. Впоследствии языку дали название Java (просто потому что это звучит лучше), и в 1995 году он стал свободно доступным. Разработчики по всему миру быстро адаптировались к прекрасному новому языку и, благодаря его модульному дизайну, получили возможность создавать новую функциональность, обогащая тем самым языковое ядро. В последующих версиях языка было добавлено множество впечатляющих функций, превративших Java в очень мощный инструмент на сегодня.

Основную сущность языка Java составляют библиотеки файлов, называемые *классами*, каждый из которых содержит небольшие фрагменты проверенного, готового к выполнению кода. Подобно кирпичам в стене, любые из этих классов можно встраивать в новую программу, и таким образом, для окончательного завершения программы обычно остается написать небольшую часть кода. Такая методика экономит программистам много времени и является одной из основных причин широкой популярности программирования на Java. К тому же такая модульная организация упрощает процесс отладки: ведь найти ошибку в небольшом модуле гораздо проще, чем в одной большой программе.

Технология Java является одновременно и платформой, и языком программирования. Исходные коды программ языка Java написаны в человекочитаемом виде в обычном текстовом файле с расширением *.java*, который затем компилируется в файлы с расширением *.class* при помощи компилятора *javac*. После этого программа выполняется интерпретатором *java* при помощи виртуальной машины Java (Java VM):

Новинка

Значок, показанный выше, указывает на добавленные или улучшенные функции, представленные в последней версии Java.



Program.java



Program.class



Программа

* В пер. с англ. «Дуб». Прим. пер.

Поскольку виртуальная машина Java доступна на различных платформах, одни и те же файлы *.class* можно запускать как в среде Windows и Linux, так и на компьютере Mac. Это и есть основной принцип кросс-платформенности языка — «написано единожды, работает везде».

Чтобы создавать программы на Java, на вашем компьютере должны быть установлены библиотеки классов и компилятор *javac*, а чтобы их запускать, нужно установить среду Java Runtime Environment (JRE), которая поддерживает интерпретатор *java*. Все вышеперечисленные компоненты содержатся в пакете Java Platform, Standard Edition Development Kit (JDK), находящемся в свободном доступе.

Программы, представленные в этой книге, используют версию JDK 8, включающую как инструмент разработчика, так и среду исполнения. Загрузить данный пакет вы можете с сайта Oracle по адресу www.oracle.com/technetwork/java/javase/downloads.



Пакет JDK 8 доступен в версиях для 32-битной и 64-битной операционных систем Linux, OS X, Solaris и Windows — примите лицензионное соглашение Oracle, затем выберите подходящую для вас версию и загрузите Java Development Kit.

Java SE Development Kit 8		
Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.		
Product / File Description	File Size	Download
Linux x86	162.47 MB	jdk-8-linux-i586.tar.gz
Linux x64	133.85 MB	jdk-8-linux-x64.rpm
Mac OS X x64	207.72 MB	jdk-8-macosx-x64.dmg
Solaris x64	93.15 MB	jdk-8-solaris-x64.tar.gz
Windows x86	161.68 MB	jdk-8-windows-i586.exe
Windows x64	155.14 MB	jdk-8-windows-x64.exe

На заметку



Страница загрузки сайта Oracle содержит также другие пакеты, но для того чтобы начать программирование на Java, требуется только JDK 8.

Совет



Слухи о том, что JAVA означает Just Another Vague Acronym (еще одна непонятная аббревиатура) не подтверждены.

Внимание

Возможно, что предыдущие версии JRE уже установлены у вас в системе и веб-браузер может выполнять Java-апплеты, но во избежание конфликтов с новой версией JDK8 лучше всего переустановить JRE.

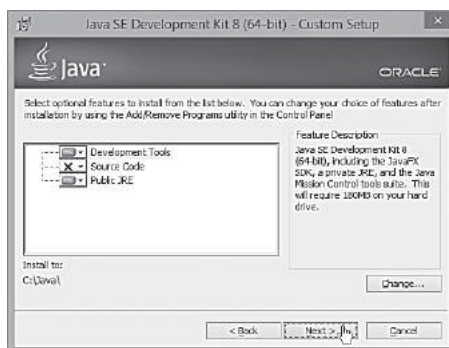
Совет

Для упрощения вы можете установить только минимальный набор функций.

Установка JDK

Для того чтобы установить Java на ваш компьютер, выберите подходящий пакет Java Development Kit для вашей операционной системы со страницы загрузки Oracle и выполните следующие шаги.

1. Удалите все предыдущие версии JDK и/или Java Runtime Environment, установленные на вашей системе.
2. Запустите процесс установки и примите лицензионное соглашение.
3. При появлении диалогового окна настроек либо подтвердите предлагаемое расположение установки, либо укажите предпочитаемое расположение, например *C:\Java* в операционных системах Windows или */usr/java* — в Linux.



4. Убедитесь, что в списке выбраны пункты **Development Tools** и **Public JRE**. Вы можете также отменить пометку для других опций установки, поскольку они необязательны для работы с примерами этой книги.
5. Нажмите кнопку **Далее** (Next), чтобы установить все необходимые инструменты и библиотеки классов Java в выбранное место.

Инструменты для компиляции и запуска Java-программ обычно выполняются из командной строки и размещаются в подкаталоге *bin* каталога *Java*. Их можно сделать доступными из любого места вашего компьютера, если добавить этот каталог в системный путь.

- В системе Windows 8, 7, Windows Vista или Windows XP нажмите кнопку **Пуск** (Start) и откройте окно **Панель управления** ⇒ **Система** ⇒ **Дополнительные параметры системы** ⇒ **Переменные среды** (Control Panel ⇒ System ⇒ Advanced System Settings ⇒ Environment Variables). Выберите системную переменную с именем **Path** и нажмите кнопку **Изменить** (Edit). Добавьте в конец значение переменной имя подкаталога **bin** (например, **C:\Java\bin**) и нажмите кнопку **ОК**.
- В системе Linux добавьте путь к каталогу **bin** в файл **.bashrc** из вашего домашнего каталога. Например, добавьте строку **PATH=\$PATH:/usr/Java/bin** и сохраните файл.

Чтобы проверить установленное окружение, откройте командную строку, введите в ней команду **java -version** и нажмите клавишу **Enter**, чтобы увидеть номер версии интерпретатора. Теперь наберите команду **javac -version**, нажмите клавишу **Enter**, и вы увидите номер версии компилятора. Номера должны совпадать — в нашем случае это версия 1.8.0. Теперь вы готовы начать программировать на Java.

```

Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\>java -version
java version "1.8.0"
Java(TM) SE Runtime Environment
Java HotSpot(TM) 64-Bit Server VM (mixed mode)

C:\>javac -version
javac 1.8.0

C:\>

```

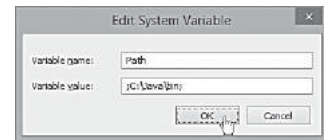
```

Linux Terminal

user ► java -version
java version "1.8.0"
Java(TM) SE Runtime Environment
Java HotSpot(TM) 64-Bit Server VM (mixed mode)

user ► javac -version
javac 1.8.0

```



Имена каталога, содержащие пробелы, должны быть заключены в двойные кавычки, например **"C:\Program Files\Java\bin"**.

На заметку



Если файл **.bashrc** не виден в списке файлов каталога, выберите команду меню **Вид** ⇒ **Показать скрытые файлы** (View ⇒ Show Hidden Files).

Внимание



В ранних версиях Windows инструменты JDK можно сделать доступными с помощью редактирования файла **autoexec.bat**, добавив имя подкаталога **bin** в конец строки **SET PATH**.

Создание первой программы на Java

Все программы на Java обычно начинаются как текстовые файлы, которые впоследствии используются для создания файлов «классов», которые, в свою очередь, являются в действительности исполняемыми программами. Это означает, что программы на Java могут быть написаны в любом простейшем текстовом редакторе, таком как, например, Блокнот (Notepad).



Hello.java

Выполните данные шаги, чтобы создать простую Java-программу, которая будет выводить традиционное приветствие.

1. Откройте простейший текстовый редактор, например, Блокнот (Notepad) и наберите в нем следующий код в точности как здесь — вы создадите класс с именем **Hello**.

```
class Hello
```

```
{
```

```
}
```

2. Между двумя фигурными скобками класса **Hello** вставьте следующий код для создания метода **main** класса **Hello**.

```
public static void main ( String[] args )
```

```
{
```

```
}
```

3. Между фигурными скобками метода **main** добавьте следующую строку кода, который определяет то, что будет делать программа.

```
System.out.println( "Hello World!" ) ;
```

4. Сохраните файл в любом удобном месте и назовите его точно так: *Hello.java* — готовая программа выглядит теперь следующим образом:



Внимание



Язык Java чувствителен к регистру, поэтому **Hello** и **hello** — совершенно разные имена. По традиции, имена программ принято всегда начинать с прописной буквы.

На заметку



Программы на Java всегда сохраняются в файле с точно таким же именем с добавлением расширения *.java*.

Для более четкого понимания каждой отдельной части программы рассмотрим их индивидуально.

Программный контейнер

```
class Hello { }
```

Имя программы объявляется после ключевого слова **class**, а после него следует пара фигурных скобок. Весь код программы, который станет определять класс **Hello**, будет помещен внутри этих фигурных скобок.

Метод main

```
public static void main ( String[] args ) {}
```

Эта выглядящая устрашающе строка является стандартным кодом для определения начальной точки, фактически всех программ на Java. В таком виде она будет использована почти во всех примерах данной книги, поэтому стоит ее запомнить.

В коде объявляется метод с именем **main**, который будет содержать внутри фигурных скобок все инструкции программы.

Ключевые слова **public static void**, предворяющие имя метода, определяют, как метод должен использоваться, и объясняются подробнее позже.

Строка кода (**String[] args**) используется при передаче значений методу и также будет подробнее объясняться позже.

Оператор

```
System.out.println( "Hello World!" ) ;
```

Операторы представляют собой команды, которые должна выполнить программа и которые всегда должны заканчиваться точкой с запятой. Метод может содержать многочисленные операторы внутри своих фигурных скобок, формируя тем самым «блок операторов», определяющий набор задач для выполнения. В данном случае одиночный оператор дает программе команду вывести строку текста.

Далее вы узнаете, как скомпилировать и запустить эту программу.

На заметку



Все самостоятельные программы Java должны содержать метод **main** (в отличие от java-апплетов, формат которых объясняется позднее).

Совет



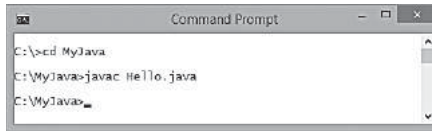
Создайте каталог *MyJava*, чтобы сохранять все ваши программные файлы в нем.

Компиляция и запуск программ

Перед тем как запускать Java-программу на выполнение, ее нужно скомпилировать в файл класса при помощи компилятора Java, который располагается в подкаталоге *bin* и имеет имя **javac**. Ранее было описано, каким образом нужно добавлять подкаталог *bin* в системный путь, чтобы компилятор **javac** мог быть запущен из любого места системы.

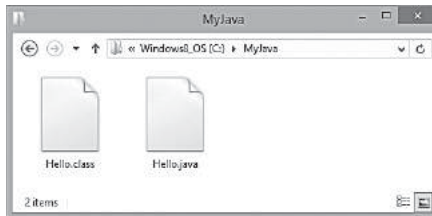
Выполните следующие шаги, чтобы скомпилировать программу с предыдущей страницы.

1. Откройте командную строку (терминальное окно) и перейдите в каталог, в котором вы сохранили файл с исходным кодом *Hello.java*.
2. В строке-подсказке наберите **javac**, затем пробел и имя файла *Hello.java* с исходным кодом и нажмите клавишу **Enter**.



Если компилятор **javac** находит ошибки в тексте программы, он останавливается и отображает сообщение, указывающее природу ошибки, — о проблемах отладки смотрите далее.

В случае, если компилятор **javac** не находит никаких ошибок, он создает новый файл с именем программы и расширением *.class*.



Когда процесс компиляции завершается, фокус возвращается в командную строку без какого-либо предупреждающего сообщения и программа готова к запуску.

Программный интерпретатор Java — это приложение с именем **java**, которое также размещается в подкаталоге *bin* вместе с компилятором **javac**. Поскольку данный каталог уже добавлен в системный путь, то интерпретатор **java** также может быть запущен из любого места.

Совет



Если в командной строке (терминальном окне) просто набрать команду **javac** и нажать клавишу **Enter**, то можно вывести параметры Java-компилятора.

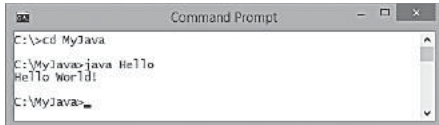
Совет



Вы можете также запускать компиляцию исходного кода из любого места, в том случае если вы укажете компилятору **javac** полный путь до файла — в данном случае это **C:\MyJava\Hello.java**.

Выполните следующие, шаги для того чтобы запустить программу, которая была скомпилирована, используя процедуру, описанную на предыдущей странице.

1. Откройте командную строку (терминальное окно) и перейдите в каталог, где расположен файл программы *Hello.class*.
2. Наберите **java** и имя программы **Hello**, а затем нажмите клавишу **Enter**.



```
C:\>cd MyJava
C:\MyJava>java Hello
Hello World!
C:\MyJava>
```

Программа **Hello** запускается и исполняет команды, описанные в операторах основного метода, в данном случае выводит строку Hello, world! После завершения фокус снова возвращается к строке-подсказке.

Процесс компиляции и запуска программы на Java обычно объединяется в последовательные шаги, и эти шаги одинаковы и независимы от платформы, на которой они выполняются. Снимок экрана, представленный ниже, демонстрирует последовательную компиляцию и запуск программы **Hello** в операционной системе Linux:



```
Linux Terminal
user > cd MyJava
user > javac Hello.java
user > java Hello
Hello World!
user >
```

Внимание



При запуске программ используйте только имя программы без расширения *.class*.



Внимание

Так же, как и другие операторы, каждое объявление переменной должно заканчиваться точкой с запятой.

Создание переменных

В Java-программировании «переменная» — это некоторый контейнер, в котором может храниться значение для дальнейшего использования в программе. Сохраненное значение может изменяться по мере исполнения программы — отсюда и название «переменная».

Переменная создается с помощью «объявления», в котором указывается тип данных, содержащихся в переменной, и задается для нее имя. Например, чтобы объявить переменную строкового типа **String** с именем **message**, которая будет содержать обычный текст, мы пишем:

```
String message ;
```

При назначении имен переменным программисты должны следовать некоторым договоренностям. Имя переменной в Java может начинаться только с латинской буквы, знака \$, либо знака _. Последующие символы могут быть латинскими буквами, цифрами, знаками \$ и знаками _. Все имена чувствительны к регистру, так что **var** и **Var** являются совершенно разными переменными. Знаки пробелов в именах не допускаются.

Также запрещается использовать в качестве имен переменных представленные в таблице ниже ключевые слова Java, которые имеют особое значение в языке:

abstract	default	goto	package	synchronized
assert	do	if	private	this
boolean	double	implements	protected	throw
break	else	import	public	throws
byte	enum	instanceof	return	transient
case	extends	int	short	true
catch	false	interface	static	try
char	final	long	strictfp	void
class	finally	native	String	volatile
const	float	new	super	while
continue	for	null	switch	

Рекомендуется именование переменной словами либо распознаваемыми аббревиатурами, описывающими назначение переменных. Например, для описания кнопки номер 1 можно использовать имена **button1** или **btn1**. Для однословных имен предпочтительно выбирать буквы нижнего регистра, а для имен многословных обычно используется так называемый «горбатый регистр», когда первая буква второго слова в верхнем регистре заглавная, например, **gearRatio**.

После того как переменная объявлена, ей можно присвоить начальное значение соответствующего типа, используя знак равенства, причем это можно сделать либо во время объявления, либо позднее, в программе. После этого к переменной можно обратиться по ее имени в любое время.

Выполните следующие шаги для создания программы, в которой переменная объявляется, тут же инициализируется, а затем изменяется..

1. Создайте новую программу с именем **FirstVariable**, содержащую метод **main**.

```
class FirstVariable
{
    public static void main ( String[] args ) { }
}
```

2. Внутри фигурных скобок главного метода добавьте следующий код для создания, инициализации и последующего вывода переменной:

```
String message = "Начальное значение" ;


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

3. Добавьте следующие строки, в которых значение переменной изменяется и снова выводится на экран:

```
message = "Измененное значение" ;

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

4. Сохраните программу под именем *FirstVariable.java*, затем скомпилируйте и запустите.



```

C:\MyJava>javac FirstVariable.java
C:\MyJava>java FirstVariable
Начальное значение
Измененное значение
C:\MyJava>

```

Совет



Строго говоря, некоторые слова, представленные в таблице, на самом деле не являются ключевыми словами языка: **true**, **false** и **null** — это литералы, **String** — это специальное имя класса, **const** и **goto** — являются зарезервированными словами (в текущей версии не используются). Они добавлены в таблицу, поскольку при именовании переменных нужно избегать давать имена, совпадающие с этими словами.



FirstVariable.java

На заметку



Если вы столкнулись с проблемой при компиляции или запуске программы, можете обратиться к концу этой главы, где описаны вопросы отладки.

Распознавание типов данных

Типы данных, наиболее часто используемые при объявлении переменных в Java, перечислены в таблице ниже с кратким описанием.

Внимание



Из-за погрешностей вычислений, вызванных использованием типа данных с плавающей точкой (`float`), не следует использовать этот тип данных там, где нужна высокая точность (например, при финансовых расчетах) — подробнее об этом см. в главе 7.

Тип данных	Описание	Пример
<code>char</code>	Одиночный символ в Юникоде	'a'
<code>String</code>	Любое количество символов в Юникоде	"my String"
<code>int</code>	Целое число в диапазоне от -2147483648 до 2147483647	1000
<code>float</code>	Вещественное число с плавающей точкой	3.14159265f
<code>boolean</code>	Логическое значение true или false	true

Обратите внимание, что переменные типа `char` выделяются одинарными кавычками, а типа `String` — двойными. Также следует запомнить, что значение типа `float` всегда имеет суффикс `f`.

В дополнение к наиболее распространенным типам данных, представленным выше, Java предоставляет возможность при некоторых обстоятельствах работать со специальными типами.

На заметку



Все ключевые слова, обозначающие типы данных, начинаются с нижнего регистра, за исключением `String`, который является специальным классом.

Тип данных	Описание
<code>byte</code>	Целое число в диапазоне от -128 до 127
<code>short</code>	Целое число в диапазоне от -32768 до 32767
<code>long</code>	Целое число в диапазоне от -9223372036854775808 до 9223372036854775807
<code>double</code>	Вещественное число в диапазоне от 1.7e-308 до 1.7e+308

Специальные типы данных применяются в более сложных Java-программах, а примеры данной книги используют в основном общие типы, представленные в верхней таблице.

Чтобы написать программу на Java, в которой создаются, инициализируются и выводятся переменные всех пяти основных типов программы, выполните следующие шаги.

1. Создайте новую программу с именем **DataTypes**, которая содержит стандартный метод **main**.

```
class DataTypes
{
    public static void main ( String[] args ) { }
```



DataTypes.java

2. Внутри фигурных скобок главного метода добавьте объявление пяти переменных, следующее ниже.

```
char letter = 'M' ;
String title = "Java in easy steps" ;
int number = 365 ;
float decimal = 98.6f ;
boolean result = true ;
```

3. Добавьте следующие строки для вывода соответствующих переменных с объединением строк.

```
System.out.println( "Буква " + letter ) ;
System.out.println( "Название " + title ) ;
System.out.println( "Количество дней " + number ) ;
System.out.println( "Температура " + decimal ) ;
System.out.println( "Ответ " + result ) ;
```

4. Сохраните программу под именем *DataTypes.java*, затем скомпилируйте и запустите.

```
Администратор: Command Prompt
C:\МуJava>javac DataTypes.java
C:\МуJava>java DataTypes
Буква М
Название Java in easy steps
Количество дней 365
Температура 98.6
Ответ true
C:\МуJava>
```

Совет



Обратите внимание, что символ `+` здесь используется для объединения (конкатенации) текстовых строк и значений сохраненных переменных.

Совет



Компилятор Java будет сообщать об ошибке, если программа попытается присвоить переменной значение не того типа — попробуйте изменить значение в примере, а затем скомпилируйте, чтобы увидеть результат.

Создание констант

Часто в программах нужно использовать «константы» — фиксированные значения, которые не должны изменяться по мере выполнения программы. Для объявления таких переменных существует ключевое слово **final**.

Константы в программе принято писать символами верхнего регистра, чтобы отличать их от обычных переменных. Если в программе происходит попытка изменить константу, то компилятор **javac** выдает сообщение об ошибке.

Выполните следующие шаги для создания Java-программы, использующей константы:



Constants.java

1. Создайте новую программу с именем **Constants**, содержащую стандартный метод **main**.

```
class Constants
{
    public static void main ( String[] args ) { }
```

2. Внутри фигурных скобок главного метода добавьте следующий код для создания и инициализации трех целочисленных констант.

```
final int TOUCHDOWN = 6 ;
final int CONVERSION = 1 ;
final int FIELDGOAL = 3 ;
```

3. Теперь объявите четыре целочисленные переменные.

```
int td , pat , fg , total ;
```

4. Проинициализируйте обычные переменные, используя операции умножения и константы.

```
td = 4 * TOUCHDOWN ;
pat = 3 * CONVERSION ;
fg = 2 * FIELDGOAL ;
total = ( td + pat + fg ) ;
```

5. Добавьте следующую строку для вывода общего количества очков.

```
System.out.println( "Очков всего: " + total ) ;
```

6. Сохраните программу под именем *Constans.java*, затем скомпилируйте и запустите, чтобы увидеть вывод общего количества очков: Очков всего: 33.

(4 x 6 = 24, 3 x 1 = 3, 2 x 3 = 6, таким образом, 24 + 3 + 6 = 33)

Совет



Символ ***** используется здесь для перемножения значений констант, а скобки, в которых заключены три слагаемых, особого значения не имеют.

Добавление комментариев

При программировании на любом языке рекомендуется добавление комментариев в код программы для объяснения каждого определенного раздела. Комментарии делают код более легким для понимания как другими программистами, так и самим разработчиком, когда он возвращается к частям программы спустя некоторый период времени.

При программировании на Java комментарии могут добавляться двух видов: многострочные с использованием символов `/*` и `*/` и однострочные с использованием символов `//`. Все, что находится между `/*` и `*/`, а также на строке после `//`, полностью игнорируется компилятором `javac`.

Если добавить комментарии в программу `Constants.java`, описанную в предыдущем разделе, исходный код может выглядеть примерно так:

```
/*
    Программа, демонстрирующая использование констант.
*/

class Constants
{
    public static void main( String args[] )
    {
        // Константы для подсчета очков.
        final int TOUCHDOWN = 6 ;
        final int CONVERSION = 1 ;
        final int FIELDGOAL = 3 ;

        // Подсчет очков.
        int td , pat , fg , total ;
        td = 4 * TOUCHDOWN ; // 4x6=24
        pat = 3 * CONVERSION ; // 3x1= 3
        fg = 2 * FIELDGOAL ; // 2x3= 6
        total = ( td + pat + fg ) ;      // 24+3+6=33

        // Вывод вычисленной суммы.
        System.out.println( "Очков всего: " + total ) ;
    }
}
```

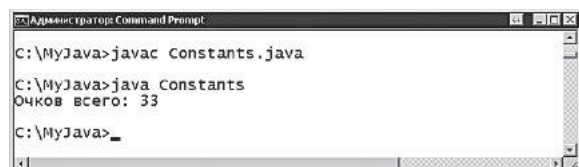
Сохранив программу с комментариями, откомпилировав и запустив ее, вы увидите, что она выполняется так же, как и раньше, без ошибок.



Constants.java
(с комментариями)

На заметку

Можете добавить операторы, которые попытаются изменить значение констант, а затем попробовать заново скомпилировать программу: вы увидите сообщение об ошибке.



Проблемы отладки

Иногда компилятор `javac` или интерпретатор `java` сообщают об ошибках. Поэтому полезно понимать причину этих ошибок и знать, как быстро решить данную проблему. Для демонстрации некоторых общих сообщений об ошибках приведем следующий пример:



Test.java

```
class test
{
    public static void main ( String[] args )
    {
        String text ;
        System.out.println( "Тест " + text )

    }
}
```

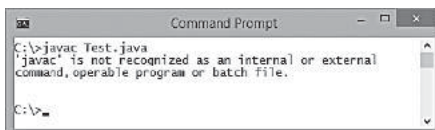
Первая попытка скомпилировать *Test.java* вызовет следующее сообщение об ошибке.

- Причина — компилятор `javac` не найден.
- Решение — отредактируйте системную переменную `PATH`, как описано ранее в этой главе, либо используйте полный путь до компилятора, чтобы запустить его.

Совет



Полный путь к файлу нужно включать в кавычки, если он содержит пробелы, например, "`C:\Program Files\Java`".



- Причина — файл *Test.java* не найден.
- Решение — либо перейдите в каталог, где размещен файл, либо используйте полный путь к файлу в самой команде.

```
Администратор Command Prompt
C:\>javac c:\MyJava\Test.java
c:\MyJava\Test.java:6: error: ';' expected
    System.out.println( "Test " + text )
                                   ^
1 error
C:\>
```

- Причина — оператор завершен некорректно.
- Решение — в исходном коде добавьте точку с запятой в конец оператора, затем сохраните файл, чтобы принять изменения.

```
Администратор Command Prompt
C:\MyJava>javac Test.java
Test.java:1: error: class test is public, should be
declared in a file named test.java
public class test
       ^
1 error
C:\MyJava>
```

- Причина — имя программы и имя класса не совпадают.
- Решение — измените в исходном коде имя класса с `test` на `Test`, затем сохраните файл для применения изменений.

```
Администратор Command Prompt
C:\MyJava>javac Test.java
Test.java:6: error: variable text might not have been initialized
    System.out.println( "Test " + text );
                                   ^
1 error
C:\MyJava>
```

- Причина — переменная `text` не имеет значения.
- Решение — в объявлении переменной присвойте переменной `text` корректное строковое значение, например, `text = "Successful"`, затем сохраните файл.

```
Администратор Command Prompt
C:\MyJava>javac Test.java
C:\MyJava>java Test
Test Successful
C:\MyJava>
```



Внимание

Вы должны запускать программу из того каталога, где она находится, — нельзя использовать полный путь к файлу, поскольку интерпретатор `java` требует не имя файла, а имя программы.

Заключение

- Java является как языком программирования, так и исполняемой платформой.
- Программы на Java можно писать в виде обычного текста и сохранять в файлах с расширением *.java*.
- Задача компилятора **javac** заключается в компиляции файлов программ *.class* из первоначальных файлов исходного кода *.java*.
- Интерпретатор **java** исполняет скомпилированные программы, используя экземпляр виртуальной машины Java (Java VM).
- Виртуальная машина Java доступна на многих операционных системах.
- Добавление подкаталога *bin* в системную переменную **PATH** позволяет запускать компилятор **javac** из любого расположения.
- Язык Java чувствителен к регистру.
- Стандартный метод **main** является точкой входа в программы на Java.
- Оператор **System.out.println()** предназначен для вывода текста.
- Имя файла программы на Java должно полностью совпадать с именем класса.
- Переменные в Java должны быть названы в соответствии с определенными соглашениями по именованию, и при этом не должны использоваться ключевые слова языка.
- Каждый оператор в Java должен быть завершен символом точки с запятой.
- Наиболее распространенные типы данных в Java — это **String** (строковые), **int** (целочисленные), **char** (символьные), **float** (с плавающей точкой) и **boolean** (логические).
- Значения строкового типа должны быть заключены в двойные кавычки, символьные — в одинарные кавычки, а значения с плавающей точкой должны содержать суффикс **f**.
- Для создания переменных, являющихся константами, используется ключевое слово **final**.
- В любую Java-программу можно добавлять многострочные комментарии, находящиеся между символами */** и **/*, либо однострочные, указываемыми после символов *//*.

Проблемы компиляции и исполнения идентифицируются с помощью сообщения об ошибках.

2

Выполнение операций

Данная глава демонстрирует применение различных операторов, используемых для создания выражений в Java-программах.

- **Выполнение арифметических операций**
- **Присваивание значений**
- **Сравнение величин**
- **Оценочная логика**
- **Проверка условий**
- **Расстановка акцентов**
- **Управляющие литералы**
- **Работа с битами**
- **Заключение**

Выполнение арифметических операций

Арифметические операции, перечисленные в таблице ниже, используются для создания в Java-программах выражений, которые возвращают одно результирующее значение. Например, выражение `4 * 2` возвращает значение 8:

Оператор	Операция
<code>+</code>	Сложение (и конкатенация строк)
<code>-</code>	Вычитание
<code>*</code>	Умножение
<code>/</code>	Деление
<code>%</code>	Деление по модулю (Остаток от деления)
<code>++</code>	Инкремент
<code>--</code>	Декремент

Операторы инкремент (`++`) и декремент (`--`) возвращают результат изменения единственного операнда на единицу. Например, `4++` возвращает значение 5, а `4--` возвращает значение 3.

Все другие арифметические операторы возвращают результат операции над двумя заданными операндами и работают обычным образом. Например, выражение `5 + 2` возвращает 7.

Оператор деления по модулю возвращает остаток от операции деления первого операнда на второй. Например, `32 % 5` возвращает 2 — так как 32 делится на 5: получается 6 и в остатке 2.

Результат операции, выполняемой оператором сложения (`+`), зависит от типа его операндов. Если операндами являются числовые значения, то возвратится значение суммы этих чисел, но когда в качестве операндов выступают строки, то в результате получится одна объединенная строка, включающая текст из обеих строк. Например, выражение `"Java" + "Arithmetic"` возвратит `"Java Arithmetic"`.

Выполните следующие шаги для создания Java-программы, выполняющей некоторые арифметические операции.

Совет



Операторы «инкремент» и «декремент» обычно используются для подсчета итераций в конструкциях с циклами, которые рассматриваются в следующей главе.

1. Создайте новую программу с именем **Arithmetic**, содержащую стандартный метод **main**.

```
class Arithmetic
{
    public static void main( String[] args ) { }
}
```



Arithmetic.java

2. Внутри фигурных скобок главного метода объявите и проинициализируйте три целочисленные переменные.

```
int num = 100 ;
int factor = 20 ;
int sum = 0 ;
```

3. Добавьте следующие строки для выполнения операций сложения и вычитания, а также вывода соответствующих результатов.

```
sum = num + factor ;// 100 + 20
System.out.println( "Результат сложения: " + sum ) ;
sum = num - factor ;// 100 - 20
System.out.println( "Результат вычитания: " + sum ) ;
```

4. Теперь добавьте строки, выполняющие операции умножения и деления, а также выводящие результаты.

```
sum = num * factor ;// 100 x 20
System.out.println( "Результат умножения: " + sum ) ;
sum = num / factor ;// 100 ÷ 20
System.out.println( "Результат деления: " + sum ) ;
```

5. Сохраните программу под именем *Arithmetic.java*, затем скомпилируйте и запустите.

```
C:\MyJava>javac Arithmetic.java
C:\MyJava>java Arithmetic
Результат сложения: 120
Результат вычитания: 80
Результат умножения: 2000
Результат деления: 5
C:\MyJava>
```

На заметку



В операторах, выводящих результат, используется оператор **+** для конкатенации в одну строку текстовой строки и целого числа, составляющего результат.

Присваивание значений

Операторы присваивания, перечисленные в таблице ниже, используются, чтобы занести результат выражения в переменную. Все из них, за исключением оператора `=`, являются сокращенной формой от более длинного эквивалентного выражения.

Оператор	Пример	Эквивалент
<code>=</code>	<code>a = b</code>	<code>a = b</code>
<code>+=</code>	<code>a += b</code>	<code>a = a + b</code>
<code>-=</code>	<code>a -= b</code>	<code>a = a - b</code>
<code>*=</code>	<code>a *= b</code>	<code>a = a * b</code>
<code>/=</code>	<code>a /= b</code>	<code>a = a / b</code>
<code>%=</code>	<code>a %= b</code>	<code>a = a % b</code>

Следует отличать оператор `=`, означающий присваивание, от оператора равенства `==`.

В выражении `a = b` значение, хранящееся в переменной с именем `b`, присваивается переменной с именем `a`. Таким образом, в `a` заносится новое значение, замещая любое, которое там содержалось раньше.

Оператор `+=` используется для добавления какого-либо значения к значению, хранящемуся в переменной, сохраняя в этой переменной «текущую сумму».

В выражении `a += b` сначала вычисляется общая сумма значений, хранящихся в переменных `a` и `b`, затем результат присваивается переменной `a`. В программе могут после этого содержаться и дальнейшие присваивания, например `a += c`, которые вычисляют сумму значений, хранящихся в `a` и `c`, а затем присваивают эту сумму переменной `a`, добавляя значение переменной `c` к предыдущему значению, хранящемуся в `a`.

Все остальные операторы присваивания работают по тому же принципу, выполняя сначала арифметическое вычисление над двумя хранящимися значениями, а затем, присваивая результат первой переменной, заносят в нее новое значение.

Выполните следующие шаги для создания Java-программы, в которой выполняются различные операторы присваивания.

На заметку



Оператор равенства `==` сравнивает значения и подробно описан далее в этой главе.

1. Создайте новую программу с именем **Assignment**, содержащую стандартный метод **main**.

```
class Assignment
{
    public static void main( String[] args ) { }
}
```



Assignment.java

2. Внутри фигурных скобок главного метода добавьте следующие строки кода, в которых складывается и присваивается строковое значение.

```
String txt = "Fantastic " ;
String lang = "Java" ;
txt += lang ; // Присваивание с объединением строк
System.out.println( "Складываем и присваиваем строки: " + txt ) ;
```

3. Добавьте эти строки для сложения и присваивания целых чисел.

```
int sum = 10 ;
int num = 20 ;
sum += num ; // Присваиваем результат ( 10 + 20 = 30 )
System.out.println( " Складываем и присваиваем целые числа: " + sum ) ;
```

4. Теперь добавьте строки, в которых производится умножение и присваивание целых чисел.

```
int factor = 5 ;
sum *= factor ; // Присваиваем результат ( 30 x 5 = 150 )
System.out.println( "Результат умножения " + sum ) ;
```

5. Добавьте строки, выполняющие деление и присваивание целых чисел.

```
sum /= factor ; // Присваиваем результат ( 150 ÷ 5 = 30 )
System.out.println( " Результат деления: " + sum ) ;
```

6. Сохраните программу под именем *Assignment.java*, затем скомпилируйте и запустите.

```
Администратор: Command Prompt
C:\MyJava>javac Assignment.java
C:\MyJava>java Assignment
Складываем и присваиваем строки: Fantastic Java
Складываем и присваиваем целые числа: 30
Результат умножения: 150
Результат деления: 30
C:\MyJava>
```



Внимание

Присваивание переменной значения неверного типа приводит к сообщению об ошибке.

Сравнение величин

Операторы сравнения, перечисленные в таблице ниже, используются для оценки двух значений в выражении и возвращают логическое значение **true** или **false**, описывающее результат этого сравнения.

Оператор	Операция сравнения
<code>==</code>	Равно
<code>!=</code>	Не равно
<code>></code>	Больше
<code>>=</code>	Больше или равно
<code><</code>	Меньше
<code><=</code>	Меньше или равно

Оператор равенства `==` оценивает два операнда и возвращает **true**, если значения операндов равны. Если операнды являются одними и теми же числами, то они равны; если они являются строками, содержащими одни и те же символы в одном и том же порядке, то они тоже равны. Операнды логического типа равны, если оба содержат **true** или оба содержат **false**.

Оператор неравенства `!=` наоборот, возвращает **true**, если значения операндов не равны, используя те же правила, что и оператор равенства.

Операторы равенства и неравенства часто применяются для выполнения так называемого условного ветвления в программе, когда в зависимости от условия выбирается дальнейшее направление работы программы.

Оператор «больше» (`>`) сравнивает два операнда и возвращает **true**, если значение первого больше, чем значение второго.

Оператор «меньше» (`<`) делает то же самое сравнение, но возвращает **true** в случае, если значение первого операнда меньше.

Операторы «больше либо равно» и «меньше либо равно» возвращают также значение **true**, если значения обоих операндов равны.

Выполните следующие шаги для создания Java-программы, выполняющей различные операции сравнения.

Совет



Оператор меньше `<` обычно используется для проверки значения счетчика в цикле – пример вы можете найти в главе 3.

1. Создайте новую программу с именем `Comparison`, содержащую стандартный метод `main`.

```
class Comparison
{
    public static void main( String[] args ) { }
}
```



Comparison.java

2. Внутри фигурных скобок главного метода добавьте следующие строки кода для сравнения двух строковых переменных на равенство.

```
String txt = "Fantastic " ;
String lang = "Java" ;
boolean state = ( txt == lang ) ; // Присваиваем результат проверки
System.out.println( "Проверка строк на равенство: " + state ) ;
```

3. Добавьте следующие строки для проверки на неравенство.

```
state = ( txt != lang ) ; // Присваиваем результат
System.out.println( " Проверка строк на неравенство: " + state ) ;
```

4. Теперь добавьте строки для сравнения двух целочисленных переменных.

```
int dozen = 12 ;
int score = 20 ;
state = ( dozen > score ) ; // Присваиваем результат
System.out.println( "Проверка на больше: " + state ) ;
```

5. Добавьте еще две строки для сравнения целочисленных переменных еще раз.

```
state = ( dozen < score ) ; // Присваиваем результат
System.out.println( " Проверка на меньше: " + state ) ;
```

6. Сохраните программу под именем *Comparison.java*, затем скомпилируйте и запустите.

Совет



Обратите внимание, что проверяемое выражение можно заключить в скобки для лучшей читаемости.

33

На заметку



В данном случае неверно (`false`), что строковые значения равны, но верно (`true`), что они не равны.

```
Администратор Command Prompt
C:\MyJava>javac Comparison.java
C:\MyJava>java Comparison
Проверка строк на равенство: false
Проверка строк на неравенство: true
Проверка на больше : false
Проверка на меньше: true
C:\MyJava>
```

Оценочная логика

Логические операторы, перечисленные в таблице ниже, используются чтобы объединить несколько выражений, каждое из которых возвращает логическое значение, в одно сложное выражение, которое будет возвращать единственное логическое значение.

Оператор	Операция
&&	Логическое И
	Логическое ИЛИ
!	Логическое НЕ (отрицание)

Логические операторы работают с операндами, имеющими значение логического (булевого) типа, то есть **true** или **false**, либо со значениями, которые преобразуются в **true** или **false**.

Оператор «логическое И» (&&) оценивает два операнда и возвращает значение **true**, только если оба операнда сами имеют значение **true**, в противном случае оператор возвращает значение **false**. Этот оператор обычно используется при условном ветвлении, когда направление работы программы определяется проверкой двух условий: если оба они верны, программа идет в определенном направлении, в противном случае — в другом.

В отличие от оператора **and**, которому необходимо, чтобы оба операнда имели значение **true**, оператор «логическое ИЛИ» (||) оценивает два операнда и возвращает **true**, если хотя бы один из них сам возвращает значение **true**. В противном случае оператор || возвратит значение **false**. Это полезно использовать при программировании определенных действий в случае выполнения одного из двух проверяемых условий.

Оператор «логическое НЕ» (!) является унарным и используется с одним операндом. Он возвращает противоположное значение от того, какое имел операнд. Так, если переменная **a** имела значение **true**, то **!a** возвратит значение **false**. Он может использоваться, например, для переключения значения переменной в последовательных итерациях цикла при помощи выражения **a=!a**. Это значит, что на каждой итерации цикла логическое значение меняется на противоположное, подобно выключению и включению лампочки.

Выполните следующие шаги для создания Java-программы, использующей три логических оператора.

1. Создайте новую программу с именем **Logic**, содержащую стандартный метод **main**.

```
class Logic
{
    public static void main( String[] args ) { }
}
```



Logic.java

- Внутри фигурных скобок главного метода добавьте следующие строки с объявлением и инициализацией двух переменных логического типа.

```
boolean yes = true ;
boolean no = false ;
```

- Добавьте строки, содержащие проверки двух условий на их истинность.

```
System.out.println( "Результат выражения yes И yes: " + ( yes && yes ) ) ;
System.out.println( " Результат выражения yes И no: " + ( yes && no ) ) ;
```

- Добавьте строки для проверки, что одно из двух условий истинно.

```
System.out.println( " Результат выражения yes ИЛИ yes: " + ( yes || yes ) ) ;
System.out.println( " Результат выражения yes ИЛИ no: " + ( yes || no ) ) ;
System.out.println( "Результат выражения no ИЛИ no: " + ( no || no ) ) ;
```

- Добавьте строки, показывающие первоначальное и противоположное значения логической переменной.

```
System.out.println( "Первоначальное значение переменной yes: " + yes ) ;
System.out.println( "Инвертированная переменная yes: " + !yes ) ;
```

- Сохраните программу под именем *Logic.java*, затем скомпилируйте и запустите.

```
Администратор Command Prompt
C:\MyJava>javac Logic.java
C:\MyJava>java Logic
Результат выражения yes И yes: true
Результат выражения yes И no: false
Результат выражения yes ИЛИ yes: true
Результат выражения yes ИЛИ no: true
Результат выражения no ИЛИ no: false
Первоначальное значение переменной yes: true
Инвертированная переменная yes: false
C:\MyJava>
```

Совет



Логический тип данных (**boolean**) назван в честь математика Джорджа Буля — создателя логической алгебры.

Проверка условий



Совет

Условный оператор также называют «тернарным».

Наверное, одним из самых любимых Java-программистами операторов является условный оператор, который делает объемную конструкцию очень краткой. Хотя его необычный синтаксис кажется на первый взгляд слишком запутанным, все равно стоит познакомиться с этим полезным оператором.

Условный оператор вначале оценивает выражение на предмет значений **true** или **false**, а затем возвращает один из двух своих операндов в зависимости от результатов оценки. Его синтаксис выглядит следующим образом:

```
( логическое-выражение ) ? если-истина-возвращаем-это : если-ложь-возвращаем-это ;
```

Каждый из указанных операндов является альтернативной веткой исполнения программы в зависимости от логической величины, возвращаемой проверочным выражением. Например, в качестве вариантов могут возвращаться строковые значения:

```
status = ( quit == true ) ? "Готово!" : "В процессе..." ;
```

В данном случае, когда переменная **quit** содержит значение **true**, условный оператор присваивает значение своего первого операнда (**Готово!**) переменной **status**, а в противном случае оператор присваивает ей значение второго операнда (**В процессе...**).

Когда проверяется простая логическая величина, то доступна краткая запись проверочного выражения, в этом случае можно опустить символы **== true**. Таким образом, вышеприведенный пример может быть записан в виде:

```
status = ( quit ) ? " Готово!" : " В процессе..." ;
```

Условный оператор может возвращать значение любого типа данных и использовать любое действительное проверочное выражение. Например, для оценки двух числовых значений может использоваться оператор сравнения **>** и возвращаться логическое значение в зависимости от результата:

```
busted = ( speed > speedLimit ) ? true : false ;
```

Аналогично в проверочном выражении может использоваться оператор неравенства **!=** для оценки строковой величины и возвращать, например, числовое значение в зависимости от результата:

```
bodyTemperature = ( scale != "Celsius" ) ? 98.6 : 37.0 ;
```

Выполните следующие шаги для создания Java-программы, использующей условный оператор.

1. Создайте новую программу с именем **Condition**, содержащую стандартный метод **main**.

```
class Condition
{
    public static void main( String[] args ) { }
}
```



Condition.java

2. Внутри фигурных скобок главного метода добавьте следующие строки, объявляющие и инициализирующие две целочисленные переменные.

```
int num1 = 1357 ;
int num2 = 2468 ;
```

3. Далее объявите строковую переменную для хранения результата проверки.

```
String result ;
```

4. Добавьте строки для определения, является ли значение первой переменной четным или нечетным числом, также добавьте вывод результата.

```
result = ( num1 % 2 != 0 ) ? "Нечетное" : "Четное" ;
System.out.println( num1 + " - " + result ) ;
```

5. Добавьте следующие строки для аналогичной проверки значения второй целочисленной переменной.

```
result = ( num2 % 2 != 0 ) ? " Нечетное " : " Четное " ;
System.out.println( num2 + " - " + result ) ;
```

6. Сохраните программу под именем *Condition.java*, затем скомпилируйте и запустите.

```
Администратор: Command Prompt
C:\MyJava>javac Condition.java
C:\MyJava>java Condition
1357 - Нечетное
2468 - Четное
C:\MyJava>
```

На заметку



В данном случае выражение будет истинным, когда есть какой-либо остаток.

Приоритет операций

Сложные составные выражения, которые содержат несколько операторов и операндов, могут быть двусмысленными до тех пор, пока не определить, в каком порядке выполнять операции. Такая неясность может привести к тому, что одно и то же выражение даст разные результаты. Например, выражение:

`num = 8 + 4 * 2 ;`

Если выполнять операции слева направо, то $8 + 4 = 12$ и $12 * 2 = 24$. Таким образом, `num = 24`. Если действовать справа налево, то $2 * 4 = 8$, и $8 + 8 = 16$. Таким образом, `num = 16`.

Java-программисты могут с помощью скобок явно указывать, какая из операций будет выполняться первой, заключая в скобки оператор, который имеет более высокий приоритет. В таком случае $(8 + 4) * 2$ означает, что сложение будет выполняться перед умножением, и в результате будет получаться 24, а не 16. И наоборот, $8 + (4 * 2)$ выполняет вначале умножение и результатом будет 16.

В случае, когда порядок операторов явно не указан, используется приоритет операторов, заданный по умолчанию, который представлен в таблице ниже. Здесь представлены операторы в порядке убывания приоритетов.

Совет



Операторы, имеющие одинаковый приоритет, обрабатываются в порядке их появления в выражении — слева направо.

Оператор	Описание
<code>++ -- !</code>	Инкремент, декремент, логическое НЕ
<code>* / %</code>	Умножение, деление, деление по модулю
<code>+ -</code>	Сложение, вычитание
<code>> >= < <=</code>	Больше, больше или равно, меньше, меньше или равно
<code>== !=</code>	Равно, не равно
<code>&&</code>	Логическое И
<code> </code>	Логическое ИЛИ
<code>? :</code>	Тернарная условная операция
<code>= += -= *= /= %=</code>	Присваивание

Выполните следующие шаги для создания Java-программы, работающей с различным приоритетом операторов.

1. Создайте новую программу с именем **Precedence**, содержащую стандартный метод **main**.

```
class Precedence
{
    public static void main( String[] args ) { }
}
```



Precedence.java

2. Внутри фигурных скобок главного метода добавьте следующий код для объявления и инициализации целочисленной переменной, содержащей результат выражения, используя приоритет операторов по умолчанию.

```
int sum = 32 - 8 + 16 * 2 ; // 16 x 2 = 32, + 24 = 56

System.out.println( "Порядок действий по умолчанию: " + sum ) ;
```

3. Добавьте строки для присваивания переменной результата того же самого выражения, но с заданием приоритета для операций вычитания и сложения.

```
sum = ( 32 - 8 + 16 ) * 2 ; // 24 + 16 = 40, x 2 = 80

System.out.println( " Указанный порядок действий: " + sum ) ;
```

4. Добавьте следующие строки, в которых в переменную заносится результат выражения, где задан приоритет операций по убыванию: сначала сложение, затем вычитание, потом умножение.

```
sum = ( 32 - ( 8 + 16 ) ) * 2 ; // 32 - 24 = 8, * 2 = 16

System.out.println( " Специфичный порядок действий: " + sum ) ;
```

5. Сохраните программу под именем *Precedence.java*, затем скомпилируйте и запустите.

```
Администратор Command Prompt
C:\MyJava>javac Precedence.java
C:\MyJava>java Precedence
Порядок действий по умолчанию: 56
Указанный порядок действий: 80
Специфичный порядок действий: 16
C:\MyJava>_
```

На заметку



При использовании вложенных скобок в выражениях высший приоритет имеют внутренние скобки.

Управляющие литералы

Числовые и текстовые значения в Java-программах известны как «литералы» — на самом деле они ничего собой не представляют, это просто символы, которые вы видите.

Литералы обычно отделяются от ключевых слов Java-языка, но там, где требуются двойные или одинарные кавычки внутри строковой переменной, необходимо указать, что символ кавычек должен как-то отделяться, чтобы избежать аварийного завершения строки. Это достигается путем добавления перед каждым символом кавычек управляющей последовательности (или управляющего оператора) `\`. Например, чтобы включить кавычки в строковую переменную, пишем следующее:

```
String quote = " \"Удача благоволит храбрым.\" сказал Вергилий ";
```

Для форматирования простейшего вывода можно использовать различные управляющие последовательности:

Управляющий символ	Описание
<code>\n</code>	Перевод строки
<code>\t</code>	Табуляция
<code>\b</code>	Шаг назад (забой)
<code>\r</code>	Возврат каретки
<code>\f</code>	Перевод страницы
<code>\\</code>	Обратный слеш
<code>\'</code>	Одиночная кавычка (апостроф)
<code>\"</code>	Двойная кавычка

Совет



В качестве альтернативы управляющей последовательности символов внутри двойных кавычек можно использовать одинарные кавычки.

Управляющая последовательность для новой строки (`\n`) часто применяется для вывода нескольких строк. Аналогично, управляющая последовательность табуляции (`\t`) используется для вывода содержимого в столбцах. Использование комбинаций новой строки и табуляции позволяет форматировать вывод как в строках, так и в столбцах — в виде некоторой таблицы.

Выполните следующие шаги для создания Java-программы, использующей управляющие последовательности для форматирования вывода.

1. Создайте новую программу с именем **Escape**, содержащую стандартный метод **main**.

```
class Escape
{
    public static void main( String[] args ) { }
}
```



Escape.java

2. Внутри фигурных скобок главного метода добавьте следующие строки кода для создания строковой переменной, содержащей форматированные названия таблицы и заголовки столбцов.

```
String header = "\n\tНЬЮ-ЙОРК ПРОГНОЗ НА 3 ДНЯ:\n" ;
header += "\n\tДень\t\tМакс\tМин\tОсадки\n" ;
header += "\t---\t\t\t---\t---\t-----\n" ;
```

3. Добавьте следующие строки для создания строковой переменной, содержащей форматированные данные и ячейки таблицы.

```
String forecast = "\tВоскресенье\t68F\t48F\tЯсно\n" ;
forecast += "\tПонедельник\t69F\t57F\tЯсно\n" ;
forecast += "\tВторник\t\t71F\t50F\tОблачность\n" ;
```

4. Добавьте строку для вывода обоих значений форматированных строк.

```
System.out.print( header + forecast ) ;
```

5. Сохраните программу под именем *Escape.java*, затем скомпилируйте и запустите.

```
Администратор: Command Prompt
c:\MyJava>javac Escape.java
c:\MyJava>java Escape

НЬЮ-ЙОРК ПРОГНОЗ НА 3 ДНЯ:

    День      Макс   Мин   Осадки
    ---      ---   ---   ---
    Воскресенье  68F   48F   Ясно
    Понедельник  69F   57F   Ясно
    Вторник      71F   50F   Облачность

c:\MyJava>
```

На заметку



В данном случае управляющая последовательность добавляет символ новой строки, поэтому здесь используется метод `print()`, а не `println()`, который добавляет символ новой строки после вывода.

Работа с битами

В дополнение к обычным операторам, описанным в этой главе, язык Java предлагает специальные операторы для работы с бинарными значениями. Они, конечно, используются, в отличие от обычных операторов, гораздо реже, но для общего знакомства приведем краткий их обзор.

«Побитовые» операторы в языке Java можно использовать с данными целочисленного типа для того, чтобы манипулировать битами двоичного значения. Для этого нужно понимать, как десятичные числа от 0 до 255 представляются в виде 8 бит.

Например, десятичное число 53 представляется двоичным 00110101 (0×128,0×64,1×32,1×16,0×8,1×4,0×2,1×1).

Операции двоичного сложения выполняются подобно арифметическим операциям над десятичными числами.

$$\begin{array}{r} 53 = 00110101 \\ + \quad 7 = 00000111 \\ \hline 60 = 00111100 \end{array}$$

Кроме того, побитовые операторы предлагают специальные операции над двоичными числами.

**Внимание**

Не путайте логический оператор И (&) с побитовым оператором (&), а также логический оператор ИЛИ (||) с побитовым оператором (|).

Оператор	Операция	Пример	Результат
&	И	a & b	1, если оба бита 1
	ИЛИ	a b	1, если хотя бы один бит 1
^	Исключающее ИЛИ	a ^ b	1, если биты отличаются
~	НЕ (отрицание)	~a	Изменяет значение бита на противоположное
<<	Сдвиг влево	n << p	Сдвигает биты значения n на p позиций влево
>>	Сдвиг вправо	n >> p	Сдвигает биты значения n на p позиций вправо

Например, используя побитовый оператор И в двоичной арифметике, мы запишем:

$$\begin{array}{r} 53 = 00110101 \\ \& \quad 7 = 00000111 \\ \hline 5 = 00000101 \end{array}$$

Очень распространенное применение побитовых операторов заключается в возможности использовать в одной переменной несколько значений. Например, программа с несколькими целочисленными переменными (флагами), имеющими значение 1 или 0 (представляющие состояние «включено» и «выключено»), обычно требует 8 бит памяти. Эти значения требуют одного бита, однако в одной переменной можно скомбинировать до 8 флагов, используя 1 бит для каждого флага. Состояние каждого флага можно вызвать с помощью побитовой операции:

1. Создайте новую программу с именем **Bitwise**, содержащую стандартный метод **main**.

```
class Bitwise
{
    public static void main( String[] args ) { }
```



Bitwise.java

2. Внутри фигурных скобок главного метода добавьте следующий код, объявляющий и инициализирующий целочисленную переменную со значением, представляющим общее состояние восьми флагов.

```
int fs = 53 ; // Двоичное представление 00110101
```

3. Добавьте следующие строки для вызова состояния каждого флага.

```
System.out.println("Флаг 1: "+(( fs&1)>0) ? "ВКЛ" : "ВЫКЛ"));
System.out.println("Флаг 2: "+(( fs&2)>0) ? "ВКЛ" : "ВЫКЛ"));
System.out.println("Флаг 3: "+(( fs&4)>0) ? "ВКЛ" : "ВЫКЛ"));
System.out.println("Флаг 4: "+(( fs&8)>0) ? "ВКЛ" : "ВЫКЛ"));
System.out.println("Флаг 5: "+(( fs&16)>0) ? "ВКЛ" : "ВЫКЛ"));
System.out.println("Флаг 6: "+(( fs&32)>0) ? "ВКЛ" : "ВЫКЛ"));
System.out.println("Флаг 7: "+(( fs&64)>0) ? "ВКЛ" : "ВЫКЛ"));
System.out.println("Флаг 8: "+(( fs&128)>0) ? "ВКЛ" : "ВЫКЛ"));
```

4. Сохраните программу под именем *Bitwise.java*, затем скомпилируйте и запустите.

```
Администратор Command Prompt
C:\MyJava>javac Bitwise.java
C:\MyJava>java Bitwise
Флаг 1: ВЫКЛ
Флаг 2: ВЫКЛ
Флаг 3: ВЫКЛ
Флаг 4: ВЫКЛ
Флаг 5: ВЫКЛ
Флаг 6: ВЫКЛ
Флаг 7: ВЫКЛ
Флаг 8: ВЫКЛ
C:\MyJava>
```

На заметку



В данном случае побитовая операция возвращает значение 1 или 0, определяющее состояние каждого флага.

Заключение

- Арифметические операторы можно использовать для формирования выражений с двумя операндами для сложения (+), вычитания (-), умножения (*), деления (/) и деления по модулю (%).
- Операторы инкремента (++) и декремента (--) изменяют единственный операнд на единицу.
- Оператор присваивания (=) можно комбинировать с арифметическими операторами, чтобы выполнять арифметические вычисления, а затем присваивать результат.
- Операторы сравнения можно использовать для формирования выражений, оценивающих два операнда на равенство (==), неравенство (!=), больше (>) или меньше (<).
- Оператор присваивания может быть скомбинирован с операторами > и <, и он будет возвращать значение **true** при равенстве операндов.
- Логические операторы И (&&) и ИЛИ (||) формируют выражения, оценивающие два операнда и возвращающие логические значения **true** или **false**.
- Логический оператор ! возвращает обратную логическую величину от значения единственного операнда.
- Условный оператор ? : оценивает заданное логическое выражение и возвращает один из двух операндов в зависимости от результата.
- При оценке логических выражений на предмет истинности можно опускать знак == **true**.
- В сложных арифметических выражениях важно явно указывать порядок операторов с помощью добавления скобок.
- Во избежание синтаксических ошибок кавычки в строковых переменных нужно предварять управляющим оператором \.
- Управляющие последовательности \n (новая строка) и \t (табуляция) организуют простое форматирование вывода.
- В особых ситуациях для выполнения двоичных арифметических операций полезно использовать побитовые операторы.

3

Создание операторов

В данной главе демонстрируется использование различных ключевых слов для создания ветвлений в Java-программах.

- Ветвление с помощью условного оператора `if`
- Альтернативное ветвление
- Ветвление с помощью переключателей
- Цикл `for`
- Цикл `while`
- Циклы `do-while`
- Выход из циклов
- Возврат управления
- Заключение

Ветвление с помощью условного оператора `if`

Ключевое слово `if` выполняет условную проверку, оценивая логическое значение выражения. Оператор, следующий за этим выражением, будет выполняться, только если данное значение равно `true` (истина). В противном случае программа переходит на последующие строки кода, выполняя альтернативное ветвление. Синтаксис оператора `if` выглядит следующим образом:

`if (проверочное-выражение) код-для-исполнения-если-результат-истина ;`

Исполняемый код может содержать несколько операторов, при этом они образуют блок операторов и заключаются в фигурные скобки.



If.java

1. Создайте новую программу с именем `If`, содержащую стандартный метод `main`.

```
class If
{
    public static void main ( String[] args) { }
```

2. Внутри фигурных скобок главного метода добавьте следующую простую проверку, в которой, при условии, что одно число больше другого, выполняется один оператор (в данном случае выполняется вывод на печать).

```
if ( 5 > 1 ) System.out.println( "Пять больше чем один." ) ;
```

3. Добавьте вторую проверку, которая, при условии, что одно число меньше другого, будет выполнять блок операторов.

```
if ( 2 < 4 )
{
    System.out.println( "Два меньше четырех." ) ;
    System.out.println( "Проверка выполнена успешно." ) ;
}
```

4. Сохраните программу под именем `If.java`, затем скомпилируйте и запустите. Поскольку обе проверки дадут в результате значение `true`, то в обоих случаях все операторы выполнятся.

В проверяемом выражении может находиться сложное составное выражение. При этом проверяется несколько условий на предмет логической величины. Можно отделять скобками отдельные выражения для установления порядка вычислений — выражения в скобках будут оцениваться первыми.

Совет



В данном случае ключевые слова `true` и `false` опущены, поскольку проверочное выражение `(2 < 4)` — это краткая запись для `(2 < 4 == true)`.

```
Администратор Command Prompt
C:\MyJava>javac If.java
C:\MyJava>java If
Пять больше чем один.
Два меньше четырех.
Проверка выполнена успешно.
C:\MyJava>_
```

Логический оператор И (&&) будет возвращать **true**, только если оба проверяемых выражения имеют значение **true**:

```
if ( ( условие-1 ) && ( условие-2 ) ) исполнить-этот-код ;
```

Логический оператор ИЛИ (||) будет возвращать значение **true**, если хотя бы одно из проверочных выражений имеет значение **true**:

```
if ( ( условие-1 ) || ( условие-2 ) ) исполнить-этот-код ;
```

Комбинации вышеуказанных выражений могут составлять более сложные и длинные проверочные выражения.

5. Добавьте следующую строку внутрь главного метода, для того чтобы объявить и проинициализировать целочисленную переменную с именем **num**.

```
int num = 8 ;
```

6. Добавьте третье проверочное выражение, которое будет выполнять оператор вывода в том случае, если значение переменной **num** находится внутри указанного диапазона или в точности равно указанному значению.

```
if ( ( ( num > 5 ) && ( num < 10 ) ) || ( num == 12 ) )
System.out.println( "Число в диапазоне от 6 до 9 включительно или равно 12" ) ;
```

7. Заново скомпилируйте программу и запустите ее еще раз на выполнение, чтобы увидеть, что оператор выполняется после прохождения сложной проверки.

```
Администратор Command Prompt
C:\MyJava>javac If.java
C:\MyJava>java If
Пять больше чем один.
Два меньше четырех.
Проверка выполнена успешно.
Число в диапазоне от 6 до 9 включительно или равно 12
C:\MyJava>_
```

8. Измените значение, присвоенное переменной **num**, таким образом, чтобы оно не входило в диапазон 6–9 и не было равно 12. Скомпилируйте заново программу и запустите ее снова, чтобы увидеть, что теперь оператор после сложной проверки не выполняется.

Совет



Диапазон может быть расширен, если включить верхние и нижние границы, используя операторы **>=** и **<=**.

На заметку



В проверочном выражении используется оператор равенства **==**, а не оператор присваивания **=**.

Альтернативное ветвление

В дополнение к ключевому слову **if** можно использовать ключевое слово **else**, которое вместе с **if** образует оператор **if else**, обеспечивающий альтернативные ветви для продолжения программы в соответствии с результатом оценки проверочного выражения. В простейшем случае он просто предлагает альтернативный оператор для исполнения, а когда проверка неуспешна — выдает значение **false**:

Внимание



Обратите внимание, что первый оператор завершается точкой с запятой перед ключевым словом **else**.

if (проверочное-выражение)

код-для-исполнения-если-результат-истина;

else

код-для-исполнения-если-результат-ложь ;

Каждая альтернативная ветвь может являться либо отдельным оператором, либо блоком операторов, заключенным внутри фигурных скобок.

При помощи оператора **if else** можно строить более сложные конструкции для осуществления дополнительных проверок внутри каждой из ветвей **if** и **else**. При этом получают вложенные операторы **if**. Когда программа обнаруживает выражение, оцененное как **true**, она исполняет операторы, связанные с этой ветвью, а затем выходит из оператора **if else**.

1. Создайте новую программу с именем **Else**, содержащую стандартный метод **main**.

```
class Else
{
    public static void main ( String[] args ) { }
```

2. Внутри главного метода добавьте следующую строку с объявлением и инициализацией целочисленной переменной **hrs**.

```
int hrs = 11 ;
```

3. Добавьте простое проверочное выражение, которое будет исполнять один оператор, в случае если значение переменной **hrs** меньше 13.

```
if ( hrs < 13 )
{
    System.out.println( "Доброе утро " + hrs ) ;
}
```



Else.java

- Сохраните программу под именем *Else.java*, затем скомпилируйте и запустите для просмотра исполняемого оператора.



```

C:\MyJava>javac Else.java
C:\MyJava>java Else
Доброе утро: 11
C:\MyJava>

```

- Присвойте переменной **hrs** значение **15**, а затем добавьте альтернативную ветвь сразу после оператора **if**.

```

else if ( hrs < 18 )
{
    System.out.println( "Добрый день " + hrs ) ;
}

```

- Сохраните изменения, скомпилируйте и запустите заново программу, чтобы увидеть, что выполняется альтернативный оператор.



```

C:\MyJava>javac Else.java
C:\MyJava>java Else
Добрый день: 15
C:\MyJava>

```

Иногда полезно ставить последнюю ветвь **else** без вложенного оператора **if**, чтобы определить оператор по умолчанию, выполняемый в случае, если никакое проверочное выражение не будет истинно.

- Присвойте переменной **hrs** значение **21**, затем добавьте следующую ветвь, заданную по умолчанию, в конец оператора **if else**.

```

else System.out.println( "Добрый вечер: " + hrs ) ;

```

- Сохраните изменения, перекомпилируйте и запустите программу еще раз, чтобы увидеть, что выполняется оператор, заданный по умолчанию.



```

C:\MyJava>javac Else.java
C:\MyJava>java Else
Добрый вечер: 21
C:\MyJava>

```

На заметку



Условное ветвление — это фундаментальный процесс, с помощью которого компьютер выполняет программы.

Ветвление с помощью переключателей

Конструкции с операторами **if else**, предлагающими большое количество условных ветвлений программы, могут стать довольно громоздкими. В тех случаях, где нужно повторять проверку одного и того же значения переменной, более элегантное решение предлагает оператор **switch** (переключатель). Типичный синтаксис блока оператора **switch** выглядит следующим образом:

```
switch ( проверяемая-переменная )
{
    case значение-1 : код-для-исполнения-если-истина ; break ;
    case значение-2 : код-для-исполнения-если-истина ; break ;
    case значение-3 : код-для-исполнения-если-истина ; break ;
    default : код-для-исполнения-если-ложь ;
}
```

Оператор **switch** работает довольно необычно. Он проверяет значения, указанные переменной, на соответствие значениям своих опций **case**, а затем выполняет оператор, связанный со значением определенной опции.

Последняя опция **default** является необязательной и может добавляться в оператор **switch** для указания операторов, которые будут исполняться в том случае, если ни одно из указанных значений не соответствует проверяемой переменной.

Каждая опция оператора начинается с ключевого слова **case** и значения для проверки, для которого стоит символ двоеточия и операторы, исполняемые в случае соответствия значению.

Важно помнить, что оператор и блок операторов, связанный с определенной опцией **case**, должны завершаться ключевым словом **break**. В противном случае программа будет продолжать исполнять операторы другой опцией **case** после той, которая прошла проверку. Иногда это и полезно, чтобы указать несколько опций **case**, для которых нужно исполнить один и тот же блок операторов. Например, один оператор для каждой из трех опций выглядит следующим образом:

```
switch ( проверяемая-переменная )
{
    case значение-1 : case значение-2 : case значение-3 :
        код-А-для-исполнения-если-истина ; break ;
}
```

Внимание



Отсутствие ключевого слова **break** не является синтаксической ошибкой, поэтому, во избежание неожиданных для вас результатов, вы должны убедиться, что все операторы **break** проставлены там, где это необходимо.

```

case значение-4 : case значение-5 : case значение-6 :
    код-В-для-исполнения-если-истина ; break ;
}

```

1. Создайте новую программу с именем **Switch**, содержащую стандартный метод **main**.

```

class Switch
{
    public static void main ( String[] args ) { }
}

```

2. Внутри главного метода добавьте объявление и инициализацию трех целочисленных переменных.

```

int month = 2, year = 2016, num = 31 ;

```

3. Добавьте блок оператора **switch** для проверки значений, присвоенных переменной **month**.

```

switch ( month )
{
}

```

4. Внутри блока **switch** добавьте опции **case**, присваивая новые значения переменной **num** для месяцев 4, 6, 9 и 11.

```

case 4 : case 6 : case 9 : case 11 : num = 30 ; break ;

```

5. Добавьте опцию **case** с присваиванием нового значения переменной **num** для месяца 2 в соответствии со значением переменной **year**.

```

case 2 : num = ( year % 4 == 0 ) ? 29 : 28 ; break ;

```

6. После блока **switch** в конце главного метода добавьте следующую строку для вывода всех трех значений переменных.

```

System.out.println( month+"/"+year+": "+num+"дней" ) ;

```

7. Сохраните программу под именем *Switch.java*, затем скомпилируйте и запустите.

```

C:\МуJava>javac Switch.java
C:\МуJava>java Switch
2/2016: 29дней
C:\МуJava>

```



Switch.java

Совет



Обратите внимание, что все три целочисленные переменные объявлены и проинициализированы в одной строке, причем использовано удобное сокращение.

На заметку



На шаге 5 используется условный оператор для эффективной проверки. Чтобы вспомнить, как он работает, можете вернуться к главе 2.

Цикл for

Циклом называется блок кода, в котором с определенной периодичностью исполняются содержащиеся в нем операторы до тех пор, пока не выполнится определенное условие; затем цикл завершается, и программа переходит к своей следующей задаче.

Наиболее часто используемая структура циклов в программировании на Java использует ключевое слово **for**, и синтаксис данной структуры следующий:

```
for ( инициализация ; проверочное-выражение ; итерация )  
{  
    операторы-для-выполнения-на-каждой-итерации ;  
}
```

Скобки после ключевого слова **for** должны содержать три управляющих выражения, которые и определяют действие цикла.

- **Инициализация** — присваивает начальное значение переменной-счетчику, который будет подсчитывать число итераций цикла. Переменная для этих целей может быть объявлена прямо здесь, и обычно это самая простая целочисленная переменная с именем **i**.
- **Проверочное выражение** — данное выражение оценивается в начале каждой итерации цикла на предмет логического значения **true**. Когда оценка возвращает значение **true**, итерация продолжается, а при возвращении значения **false** цикл немедленно прекращает свою работу, не завершая текущую итерацию.
- **Итерация** — изменяет текущее значение переменной-счетчика, храня в себе общее число итераций, сделанных циклом. Обычно здесь используется выражение **i++** для увеличения либо **i--** для уменьшения значения счетчика.

На заметку



Итерацию часто называют счетчиком приращений, поскольку чаще всего она используется для увеличения значения, а не для уменьшения.

Исполняемый на каждой итерации цикла код может быть как одиночным оператором, так и блоком операторов и даже вложенным циклом.

Каждый цикл в какой-то определенной точке должен привести значение проверочного выражения в значение **false**, иначе будет создан так называемый бесконечный цикл. В общем случае проверочное выражение обычно оценивает текущее значение переменной-счетчика для выполнения определенного количества итераций. Например, если проинициализировать счетчик **i** значением 1 и увеличивать его на единицу с каждой итерацией, то после 10 итераций значение выражения **i < 11** станет ложным. Таким образом, цикл, исполнившись 10 раз, завершится.

1. Создайте новую программу с именем **For**, содержащую стандартный метод **main**.

```
class For
{
    public static void main ( String[] args ) { }
```



For.java

2. Внутри главного метода объявите и проинициализируйте целочисленную переменную для подсчета общего числа итераций.

```
int num = 0 ;
```

3. Добавьте цикл **for**, выполняющий три итерации и отображающий текущее значение своего счетчика-переменной **i** на каждой итерации.

```
for ( int i = 1 ; i < 4 ; i++ )
{
    System.out.println( "Внешний цикл i=" + i ) ;
}
```

4. Внутри блока цикла **for** добавьте вложенный цикл **for**, который также выполняет три итерации, отображая текущее значение счетчика-переменной **j** и общее число итераций.

```
for ( int j = 1 ; j < 4 ; j++ )
{
    System.out.print( "\tВнутренний цикл j=" + j ) ;
    System.out.println( "\t\tВсего num="+ (++num) ) ;
}
```

5. Сохраните программу под именем *For.java*, затем скомпилируйте и запустите, чтобы увидеть вывод.

```
Администратор: Command Prompt
C:\MyJava>javac For.java
C:\MyJava>java For
Внешний цикл i=1
    Внутренний цикл j=1      Всего num=1
    Внутренний цикл j=2      Всего num=2
    Внутренний цикл j=3      Всего num=3
Внешний цикл i=2
    Внутренний цикл j=1      Всего num=4
    Внутренний цикл j=2      Всего num=5
    Внутренний цикл j=3      Всего num=6
Внешний цикл i=3
    Внутренний цикл j=1      Всего num=7
    Внутренний цикл j=2      Всего num=8
    Внутренний цикл j=3      Всего num=9
C:\MyJava>
```



Внимание

Операторы инкремента и декремента могут быть как префиксными, так и постфиксными. В первом случае они вначале увеличивают значение переменной, а затем происходит обращение к ней, во втором случае — наоборот.

Цикл while

Альтернативной структурой для цикла **for** является структура цикла, использующая ключевое слово **while** и имеющая следующий синтаксис:

```
while ( проверочное-выражение )
{
    операторы-для-выполнения-на-каждой-итерации ;
}
```

Подобно циклу **for** цикл **while** тоже с периодичностью выполняет содержащиеся в нем операторы до тех пор, пока проверочное условие не будет иметь значение **true**. В этом случае цикл завершает свою работу, и программа переходит к следующей задаче.

В отличие от цикла **for**, в скобках после ключевого слова **while** не содержится ни инициализатора, ни модификатора, изменяющего переменную-счетчик. Это означает, что в проверочном выражении должна находиться какая-то величина, которая будет изменяться по мере выполнения итераций цикла, иначе будет создан бесконечный цикл, выполняющий свои операторы.

Проверочное выражение оценивается в начале каждой итерации цикла на предмет логического значения **true**. Если результат проверки возвращает **true**, итерация продолжается, в противном случае цикл немедленно завершается, не выполняя итерацию.

Обратите внимание, что если проверочное выражение возвращает значение **false** при самой первой оценке, то операторы цикла никогда не исполнятся.

Внимание



Бесконечный цикл заблокирует программу, поскольку итерации продолжают выполняться — в системе Windows, чтобы выйти из этой ситуации, нажмите сочетание клавиш **Ctrl+C**.

При помощи цикла **while** можно смоделировать структуру цикла **for**, при этом в проверочном выражении будет оцениваться переменная-счетчик, которая станет создаваться вне цикла, а изменять свое значение внутри цикла на каждой его итерации. Например, внешний цикл **for** из предыдущего примера можно воссоздать при помощи цикла **while** следующим образом:

```
int i = 1 ;
while ( i < 4 )
{
    System.out.println( "Внешний цикл i=" + i ) ;
    i++ ;
}
```

Таким образом, мы переместили инициализатор снаружи цикла перед структурой **while**, а модификатор — внутри блока операторов самого цикла.

1. Создайте новую программу с именем **While**, содержащую стандартный метод **main**.

```
class While
{
    public static void main ( String[] args ) { }
```



While.java

2. Внутри главного метода объявите и проинициализируйте целочисленную переменную с именем **num**.

```
int num = 100 ;
```

3. Добавьте цикл **while**, который будет отображать текущее значение переменной **num**, пока оно остается больше нуля.

```
while ( num > 0 )
{
    System.out.println( "Обратный отсчет с использованием While: " + num ) ;
}
```

4. В конец блока **while** добавьте модификатор, чтобы уменьшать значение переменной **num** на 10 на каждой итерации, тем самым избегая бесконечного цикла.

```
num -= 10 ;
```

5. Сохраните программу под именем *While.java*, затем скомпилируйте и запустите.

```
C:\MyJava>javac while.java
C:\MyJava>java while
Обратный отсчет с использованием while: 100
Обратный отсчет с использованием while: 90
Обратный отсчет с использованием while: 80
Обратный отсчет с использованием while: 70
Обратный отсчет с использованием while: 60
Обратный отсчет с использованием while: 50
Обратный отсчет с использованием while: 40
Обратный отсчет с использованием while: 30
Обратный отсчет с использованием while: 20
Обратный отсчет с использованием while: 10
C:\MyJava>
```

На заметку



Присваивание в данном модификаторе — это сокращенная запись для `num = (num - 10);`.

Циклы do-while

Одной из вариаций структуры цикла **while**, описанной на предыдущей странице, является цикл, использующий ключевое слово **do** и имеющий следующий синтаксис:

```
do
{
    операторы-для-выполнения-на-каждой-итерации ;
}
while ( проверочное-выражение ) ;
```

Подобно циклам **for** и **while**, цикл **do-while** с периодичностью выполняет содержащиеся в нем операторы до тех пор, пока проверочное выражение не примет значение **true** — затем цикл завершает свою работу и программа переходит к следующей задаче.

В отличие от циклов **for** и **while**, в цикле **do-while** проверочное выражение стоит после блока, содержащего выполняющиеся в цикле операторы. Проверочное выражение оценивается в конце каждой итерации цикла на предмет логического значения. Когда проверочное выражение в результате возвращает значение **true**, продолжается следующая итерация, в противном случае цикл немедленно завершается. Это означает, что операторы внутри цикла **do-while** исполняются, по крайней мере, один раз. Обратите внимание, что если проверочное выражение возвратит значение **false** при первой его оценке, то операторы цикла при этом уже один раз выполнены.

С помощью цикла **do-while** можно также смоделировать структуру цикла **for**, оценивая переменную-счетчик в его проверочном выражении, при этом инициализатор будет находиться вне цикла, а модификатор — внутри блока операторов, точно так же, как в цикле **while**.

Для всех видов циклов **for**, **while** и **do-while**, если они содержат только один оператор для исполнения, можно опускать фигурные скобки вокруг этого оператора. Но если вы захотите добавить дополнительные операторы, то обойтись без скобок не получится.

Совет



Операторы, исполняемые в цикле, всегда заключайте в скобки — это внесет ясность и улучшит обслуживаемость кода.

Какой из циклов — **for**, **while** или **do-while** — выбрать при программировании, зависит от персональных предпочтений и от того, для чего используется цикл. Структура цикла **for** удобно размещает инициализатор, проверочное выражение и модификатор (после ключевого слова **for**). Структура цикла **while** может быть более лаконичной, но вы не должны забывать включить модификатор в операторы цикла во избежание его закливания. Цикл **do-while** просто добавляет возможность исполнения своего блока операторов хотя бы один раз перед

оценкой проверочного выражения — как демонстрируется в примере ниже.

1. Создайте новую программу с именем **DoWhile**, содержащую стандартный метод **main**.

```
class DoWhile
{
    public static void main ( String[] args ) { }
```



DoWhile.java

2. Внутри главного метода добавьте объявление и инициализацию целочисленной переменной с именем **num**.

```
int num = 100 ;
```

3. Добавьте цикл **do-while** для вывода текущего значения переменной **num**, пока оно меньше нуля.

```
do
{
    System.out.println( "Используем DoWhile: " + num ) ;
}
while ( num < 0 )
```

4. Добавьте в конец блока **do-while** модификатор для изменения значения переменной **num** на каждой итерации, тем самым избежав бесконечного цикла.

```
num += 10 ;
```

5. Сохраните программу под именем *DoWhile.java*, затем скомпилируйте и запустите — вы увидите, что проверочное условие не выполняется, но оператор в цикле выполняется один раз.

```
Администратор Command Prompt
C:\МуJava>javac DoWhile.java
C:\МуJava>java DoWhile
Используем DoWhile: 100
C:\МуJava>
```

На заметку



Присваивание в этом модификаторе является сокращенной записью для `num = (num + 10)`.

Выход из циклов

Ключевое слово **break** может применяться для преждевременного прерывания цикла при определенном условии. Оператор **break** располагается внутри блока операторов цикла и предваряется проверочным выражением. Когда проверка возвращает значение **true**, цикл немедленно завершается и программа переходит к следующей задаче. Например, в случае вложенного цикла происходит переход к следующей итерации внешнего цикла.



Break.java

1. Создайте новую программу с именем **Break**, содержащую стандартный метод **main**.

```
class Break
{
    public static void main ( String[] args ) { }
```

2. Внутри главного метода создайте два вложенных цикла **for**. Эти циклы будут отображать значение своих счетчиков на каждой из трех итераций.

```
for ( int i = 1 ; i < 4 ; i++ )
{
    for ( int j = 1 ; j < 4 ; j++ )
    {
        System.out.println( "Итерация i="+i+" j="+j ) ;
    }
}
```

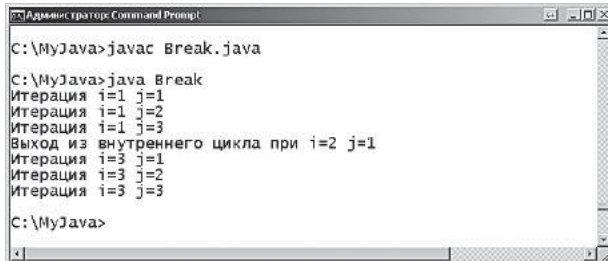
3. Сохраните программу под именем *Break.java*, затем скомпилируйте и запустите, чтобы увидеть вывод.

```
Администратор: Command Prompt
C:\MyJava>javac Break.java
C:\MyJava>java Break
Итерация i=1 j=1
Итерация i=1 j=2
Итерация i=1 j=3
Итерация i=2 j=1
Итерация i=2 j=2
Итерация i=2 j=3
Итерация i=3 j=1
Итерация i=3 j=2
Итерация i=3 j=3
C:\MyJava>
```

Здесь выполняются три итерации внешнего цикла, и на каждой выполняется внутренний цикл. Чтобы остановить второе исполнение внутреннего цикла, можно добавить оператор **break**.

4. Добавьте оператор **break** в начало блока операторов внутреннего цикла, чтобы выполнить выход из этого цикла, затем перекомпилируйте и запустите заново программу.

```
if ( i == 2 && j == 1 )
{
    System.out.println( "Выход из внутреннего цикла при i=" +i+ " j=" +j ) ;
    break ;
}
```

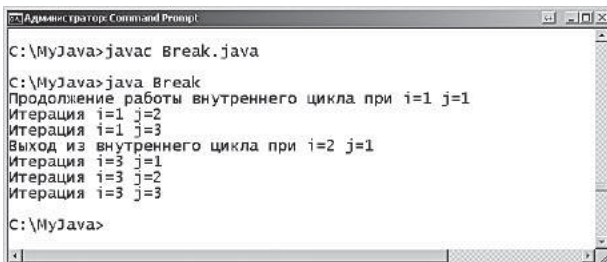


```
Администратор: Command Prompt
C:\MyJava>javac Break.java
C:\MyJava>java Break
Итерация i=1 j=1
Итерация i=1 j=2
Итерация i=1 j=3
Выход из внутреннего цикла при i=2 j=1
Итерация i=3 j=1
Итерация i=3 j=2
Итерация i=3 j=3
C:\MyJava>
```

Существует возможность пропустить одну итерацию цикла при определенном условии — для этого используется ключевое слово **continue**. Оператор **continue** располагается внутри блока операторов цикла и также предваряется проверочным выражением. Если проверка выдает значение **true**, то данная итерация завершается.

5. Добавьте оператор **continue** в начало блока операторов внутреннего цикла, чтобы пропустить первую итерацию внутреннего цикла, затем перекомпилируйте и запустите заново программу.

```
if ( i == 1 && j == 1 )
{
    System.out.println( "Продолжение работы внутреннего цикла при i=" +i+ " j=" +j ) ;
    continue;
}
```



```
Администратор: Command Prompt
C:\MyJava>javac Break.java
C:\MyJava>java Break
Продолжение работы внутреннего цикла при i=1 j=1
Итерация i=1 j=2
Итерация i=1 j=3
Выход из внутреннего цикла при i=2 j=1
Итерация i=3 j=1
Итерация i=3 j=2
Итерация i=3 j=3
C:\MyJava>
```

На заметку



В данном случае оператор **break** пропускает все три итерации внутреннего цикла на второй итерации внешнего цикла.

На заметку



В данном случае оператор **continue** позволяет пропустить первую итерацию внутреннего цикла на первой итерации внешнего цикла.

Возврат управления

Стандартное поведение операторов **break** и **continue** может быть изменено, если указать явно, что управление следует передать помеченному оператору цикла, указав имя его метки.



Label.java

1. Создайте новую программу с именем **Label**, содержащую стандартный метод **main**.

```
class Label
{
    public static void main ( String[] args ) { }
```

2. Внутри главного метода создайте два вложенных цикла **for**, которые выводят значения своих счетчиков на каждой из трех итераций.

```
for ( int i = 1 ; i < 4 ; i++ )
{
    for ( int j = 1 ; j < 4 ; j++ )
    {
        System.out.println( "Итерация i="+i+ " j="+j ) ;
    }
}
```

3. Сохраните программу под именем **Label.java**, затем скомпилируйте и запустите.

```
Администратор: Command Prompt
c:\MyJava>javac Label.java
c:\MyJava>java Label
Running i=1 j=1
Running i=1 j=2
Running i=1 j=3
Running i=2 j=1
Running i=2 j=2
Running i=2 j=3
Running i=3 j=1
Running i=3 j=2
Running i=3 j=3
c:\MyJava>
```

Для того чтобы пометить цикл, перед ним ставится имя метки и символ двоеточия.

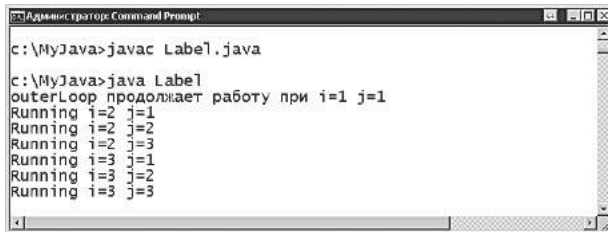
4. Отредактируйте начало внешнего цикла, пометив его меткой **outerLoop**.

```
outerLoop : for ( int i = 1 ; i < 4 ; i++ )
```

Чтобы явно указать программе, что нужно перейти к этому внешнему циклу, поставьте имя данной метки после ключевого слова **continue**.

5. Добавьте оператор **continue** в начало блока операторов внутреннего цикла, чтобы перейти к следующей итерации внешнего цикла, затем перекомпилируйте и запустите заново программу.

```
if ( i == 1 && j == 1 )
{
    System.out.println( "outerLoop продолжает работу при i=" +i+ " j=" +j ) ;
    continue outerLoop ;
}
```

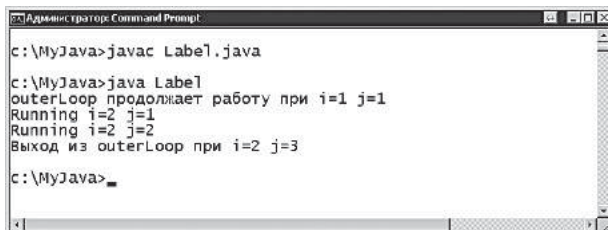


```
Администратор: Command Prompt
c:\MyJava>javac Label.java
c:\MyJava>java Label
outerLoop продолжает работу при i=1 j=1
Running i=2 j=1
Running i=2 j=2
Running i=2 j=3
Running i=3 j=1
Running i=3 j=2
Running i=3 j=3
```

Для явного указания того, что программа должна выйти из внешнего цикла, введите после ключевого слова **break** имя этой метки.

6. Добавьте оператор **break** в начало блока операторов внутреннего цикла для выхода из внешнего цикла, затем снова скомпилируйте и запустите программу.

```
if ( i == 2 && j == 3 )
{
    System.out.println( "Выход из outerLoop при i=" +i+ " j=" +j ) ;
    break outerLoop ;
}
```



```
Администратор: Command Prompt
c:\MyJava>javac Label.java
c:\MyJava>java Label
outerLoop продолжает работу при i=1 j=1
Running i=2 j=1
Running i=2 j=2
Выход из outerLoop при i=2 j=3
c:\MyJava>
```

На заметку



В данном случае оператор **continue** пропускает все три итерации первого прохода внутреннего цикла, возвращая управление внешнему циклу.

На заметку



В данном случае оператор **break** пропускает все дальнейшие итерации всей структуры цикла, совершая выход из внешнего цикла.

Заключение

- Ключевое слово **if** используется для оценки проверочного выражения на предмет логических значений **true** или **false**.
- Блок операторов **if** может содержать один или более операторов, которые выполняются только тогда, когда проверочное выражение возвращает значение **true**.
- Ключевое слово **else** определяет альтернативные операторы для выполнения программы в случае, когда проверка, выполняемая ключевым словом **if**, возвращает значение **false**.
- Комбинация операторов **if else** позволяет программе осуществлять условное ветвление.
- Оператор **switch** в большинстве случаев предлагает хорошую альтернативу операторам **if else**, предоставляя различные опции для выбора.
- Каждая опция **case** может быть завершена ключевым словом **break**, и, таким образом, выполняться будут только операторы, связанные с этой опцией.
- Ключевое слово **default** может определять операторы, исполняемые в случае, когда все опции **case** возвращают значение **false**.
- Цикл с периодичностью исполняет операторы, содержащиеся в нем, до тех пор, пока проверочное выражение не возвратит значение **false**.
- После ключевого слова **for** в скобках указываются инициализатор, проверочное выражение и модификатор счетчика.
- В циклах **while** и **do-while** во избежание входа в бесконечный цикл нужно изменять значение переменной, используемой в проверочном выражении.
- В циклах **for** и **while** проверочное выражение оценивается в самом начале — перед первой итерацией цикла.
- В циклах **do-while** проверочное выражение оценивается в конце цикла — после первой итерации.
- Итерацию цикла можно пропустить с помощью ключевого слова **continue**.
- Используя ключевое слово **break**, можно завершать работу цикла.
- Вложенные внутренние циклы могут использовать метки вместе с ключевыми словами **break** и **continue**, чтобы передавать управление внешнему циклу.

4

Работа с данными

В данной главе описывается, каким образом работать со значениями данных, используя различные программные конструкции языка Java.

- Преобразование типов
- Создание массивов переменных
- Передача аргументов
- Передача множественных аргументов
- Обход элементов в цикле
- Изменение значений элемента
- Добавление размеров массива
- Перехват исключений
- Заключение

Преобразование типов

В программировании на Java при работе со значениями следует внимательно учитывать типы данных во избежание ошибок компиляции. Например, если методу, который требует значение типа **int**, передать переменную типа **float**, то это приведет к ошибке компилятора. Это означает, что зачастую для того, чтобы обрабатывать данные, их нужно конвертировать из одного типа в другой.

Числовые значения можно легко преобразовывать (приводить) из одного числового типа данных в другой, используя синтаксис:

(тип-данных) значение

Необходимо понимать, что при приведении типов данных может происходить некоторая потеря точности. Так случается, например, при преобразовании числа с плавающей точкой (**float**) в целое число (**int**), поскольку значение после десятичной точки будет отбрасываться. Например, приведение значения 9.9 к целочисленному типу выдаст в результате значение 9.

Значения символьного типа данных (**char**) могут быть автоматически использованы в качестве значений целочисленного типа (**int**), потому что каждый из них имеет уникальное целочисленное представление, а именно числовой код ASCII, поддерживаемый компилятором Java. Например, латинская буква A имеет числовой код 65.

Числовые значения могут быть преобразованы к строковому типу (**String**) при помощи метода **toString()**. Он в качестве аргумента в своих скобках принимает числовое значение. Например, преобразование целочисленной переменной **num** к строковому типу имеет вид **Integer.toString(num)**. Аналогично преобразование переменной плавающего типа **num** в строковую будет иметь вид **Float.toString(num)**. На самом деле такие преобразования не всегда требуются, поскольку, например, объединение двух переменных в строку, если одна из них имеет строковый тип, происходит автоматически.

С большей вероятностью вам будет нужно преобразовывать величины строкового типа в числа, чтобы потом использовать их в арифметических вычислениях. Значение строкового типа может быть преобразовано в целочисленное при помощи метода **Integer.parseInt()**. Данный метод принимает строковую переменную в качестве аргумента в своих скобках. Например, преобразование строковой переменной **msg** в целое число будет иметь вид **Integer.parseInt(msg)**. Аналогично для переменной с плавающей точкой — **Float.parseFloat(msg)**. При преобразовании строковой величины в числовой тип данных строка должна содержать действительное числовое значение, иначе компилятор сообщит об ошибке.

На заметку



Все числовые классы имеют метод **parse...** и **toString**, позволяющие преобразовывать строковые и числовые значения.

1. Создайте новую программу с именем **Convert**, содержащую стандартный метод **main**.

```
class Convert
{
    public static void main ( String[] args ) { }
```



Convert.java

2. Внутри главного метода объявите и проинициализируйте одну строковую переменную и одну переменную с плавающей точкой.

```
float daysFloat = 365.25f ;
String weeksString = "52" ;
```

3. Преобразуйте переменную с плавающей точкой в целочисленную.

```
int daysInt = (int) daysFloat ;
```

4. Преобразуйте строковую переменную в целочисленную.

```
int weeksInt = Integer.parseInt( weeksString ) ;
```

5. Произведите арифметические действия над преобразованными значениями и выведите результат.

```
int week = ( daysInt / weeksInt ) ;
System.out.println( "Дней в неделе: " + week ) ;
```

6. Сохраните программу под именем **Convert.java**, затем скомпилируйте и запустите.

A screenshot of a Windows Command Prompt window. The title bar reads "Администратор: Command Prompt". The command history shows: "C:\MyJava>javac Convert.java", "C:\MyJava>java Convert", the output "Дней в неделе: 7", and "C:\MyJava>".

```
Администратор: Command Prompt
C:\MyJava>javac Convert.java
C:\MyJava>java Convert
Дней в неделе: 7
C:\MyJava>
```

Создание массивов переменных

Массив — это простая переменная, которая может содержать несколько значений, в отличие от обычной переменной, содержащей единственное значение.

Объявление массива включает в себя тип данных с использованием ключевых слов, обозначающих типы данных, после которого следуют квадратные скобки, обозначающие, что это будет массив. Затем в объявлении стоит имя массива в соответствии с договоренностями именования.

Массив можно проинициализировать при объявлении, присвоив значение соответствующего типа данных, записанное в фигурные скобки и разделенное запятыми. Например, объявление целочисленного массива, проинициализированного тремя значениями, может выглядеть следующим образом:

```
int[ ] numbersArray = { 1, 2, 3 } ;
```

При этом создается массив длиной, равной количеству элементов, указанных в списке. В данном случае массив состоит из трех элементов. Значения элементов сохраняются в массиве под порядковыми номерами (индексами), по которым можно обратиться к этим элементам. Для обращения к элементам следует написать имя массива и соответствующий индекс элемента в квадратных скобках. Элементы индексируются, начиная с нуля. Например, чтобы обратиться к первому элементу из массива, указанного выше, мы запишем `numbersArray[0]`.

Несмотря на то, что значения, хранящиеся в каждом элементе массива, могут быть изменены так же, как обычная переменная, размер массива определяется при объявлении и не может быть изменен впоследствии. Общее число элементов в массиве можно узнать, если обратиться к свойству массива `length`. Для этого нужно записать через точечную запись «имя массива.length». Например, `numbersArray.length` возвратит размер массива, приведенного выше, в данном случае целое число 3.

Массивы могут быть объявлены также и без присвоения начальных значений элементам. Это можно сделать при помощи ключевого слова `new`, создавая пустой «объект» массива указанного размера. Число требуемых элементов указывается в присваивании внутри квадратных скобок после типа данных. Например, объявление пустого целочисленного массива с тремя элементами может выглядеть следующим образом:

```
int[ ] numbersArray = new int[3] ;
```

Во время создания пустого массива элементам присваиваются значения в зависимости от типов данных: для типов `int` и `float` — 0, для

Внимание



Не забывайте, что массив индексируется, начиная с нуля. Это означает, что `index[2]` обращается к третьему элементу массива, а не ко второму.

типа **String** — значение **null**, для типа **char** — **\0**, а для типа **boolean** — значение **false**.

1. Создайте новую программу с именем **Array**, содержащую стандартный метод **main**.

```
class Array
{
    public static void main ( String[] args ) { }
```



Array.java

2. Внутри главного метода объявите и проинициализируйте строковый массив из трех элементов.

```
String[] str = { "Java ", "Хороший", " Язык" } ;
```

3. Объявите пустой целочисленный массив с тремя элементами.

```
int[] num = new int[3] ;
```

4. Присвойте значения первым двум элементам целочисленного массива.

```
num[0] = 100 ;
```

```
num[1] = 200 ;
```

5. Присвойте новые значения второму элементу строкового массива.

```
str[1] = "Лучший" ;
```

6. Выведите длину каждого массива и содержимое всех элементов.

```
System.out.println( "Размер строкового массива " + str.length ) ;
```

```
System.out.println( "Размер целочисленного массива " + num.length ) ;
```

```
System.out.println( num[0] + "," + num[1] + "," + num[2] ) ;
```

```
System.out.println( str[0] + str[1] + str[2] ) ;
```

7. Сохраните программу под именем *Array.java*, затем скомпилируйте и запустите.

```
Администратор: Command Prompt
c:\MyJava>javac Array.java
c:\MyJava>java Array
Размер строкового массива 3
Размер целочисленного массива 3
100,200,0
Java Лучший Язык
c:\MyJava>
```

На заметку



Строковые переменные должны быть заключены в кавычки.

Передача аргументов

Код на Java, объявляющий стандартный метод **main**, включает внутри своих скобок аргумент, который создает строковый массив с именем **args**:

```
public static void main( String[] args ) { }
```

Основное назначение массива **args[]** — передавать значения (аргументы) в программу при ее вызове. Значения, которые нужно передать в программу, добавляются в командную строку при вызове программы после ее имени. Например, команда, которая передаст строку **Java** команде с именем **Run**, будет выглядеть так: **Run Java**.

Одно значение, переданное в программу, автоматически помещается в первый элемент массива **args[]**. Тем самым, к нему можно обратиться в виде **args[0]**.

Важно знать, что массив **args[]** содержит строковый тип данных, поэтому любое числовое значение, переданное в программу, будет сохраняться как строковое представление этого числа. Это означает, что программа не сможет использовать данное значение в арифметической операции до тех пор, пока его не преобразовать к числовому типу данных, например **int**. Например, **Run 4** передает число 4 в программу, которая сохраняется как строковое значение "4", а не как целочисленное 4. Следовательно, результат выполнения выражения **args[0]+3** выдаст объединенную строку "43", а не сумму 7. Но с помощью преобразования, используя метод **Integer.parseInt()**, можно получить требуемый результат, если вам нужна сумма.

Строку, содержащую пробелы, можно передавать в программу как одно строковое значение, заключив всю эту строку в двойные кавычки. Например, **Run "Java In Easy Steps"**.

Передача аргументов в программу может, например, использоваться для того, чтобы указывать опции запуска конкретной программы. Опция передается в программу в виде строкового значения **args[0]** и может быть впоследствии оценена с помощью метода **String.equals()**. Например, **args[0].equals("b")** сравнивает аргумент со строковым значением "b".

1. Создайте новую программу с именем **Option**, содержащую стандартный метод **main**.

```
class Option
{
    public static void main ( String[] args ) { }
```



Option.java

2. Внутри главного метода напишите оператор **if**, в котором произведите поиск аргумента с именем **"-en"**.

```
if ( args[0].equals( "-en" ) )
{
    System.out.println( "Опция для Английского языка" ) ;
}
```

3. Добавьте альтернативную ветку **else** к оператору **if** для поиска аргумента **"-es"**.

```
else if ( args[0].equals( "-es" ) )
{
    System.out.println( "Опция для Испанского языка" ) ;
}
```

4. Добавьте еще одну альтернативную ветку **else** для вывода ответа по умолчанию.

```
else System.out.println( "Неизвестная опция" ) ;
```

5. Сохраните программу под именем *Option.java*, затем скомпилируйте и запустите.

```
Администратор: Command Prompt
C:\MyJava>javac Option.java
C:\MyJava>java Option -en
Опция для Английского языка
C:\MyJava>java Option -es
Опция для Испанского языка
C:\MyJava>java Option -ez
Неизвестная опция
C:\MyJava>
```

Передача множественных аргументов

С помощью командной строки в программу можно передать несколько аргументов, которые следуют после имени программы и пробела. Аргументы должны отделяться, по крайней мере, одним пробелом, а их значения помещаются в элементы массива **args[]**. После этого к каждому из значений можно обращаться по индексу, равно как и к любому другому элементу массива, — **args[0]** для первого аргумента, **args[1]** для второго аргумента и так далее.

Чтобы убедиться, что пользователь ввел необходимое количество аргументов, программе нужно проверить свойство **length** массива **args[]**. Если проверка не дала необходимого результата, можно использовать ключевое слово **return** для выхода из главного метода, тем самым произведя выход из программы.



Args.java

1. Создайте новую программу с именем **Args**, содержащую стандартный метод **main**.

```
class Args
{
    public static void main ( String[] args ) { }
```

2. Внутри главного метода введите условный оператор **if** для проверки количества введенных аргументов и вывода соответствующего сообщения, а также выхода из программы в случае, если количество аргументов не соответствует требуемому — в данном случае трем.

```
if ( args.length != 3 )
{
    System.out.println( "Неверное число аргументов" ) ;
    return ;
}
```

3. После оператора **if** создайте две целочисленные переменные и проинициализируйте их значениями первого и третьего аргументов соответственно.

```
int num1 = Integer.parseInt( args[0] ) ;
int num2 = Integer.parseInt( args[2] ) ;
```

- Добавьте строковую переменную, проинициализировав ее значением объединения всех трех аргументов.

```
String msg = args[0] + args[1] + args[2] + "=" ;
```

- Добавьте оператор **if else**, выполняющий арифметические действия над аргументами и объединяющий результат в переменной строкового типа.

```
if ( args[1].equals("+") ) msg += (num1 + num2);
else if ( args[1].equals("-") ) msg += (num1 - num2);
else if ( args[1].equals("x") ) msg += (num1 * num2);
else if ( args[1].equals("/") ) msg += (num1 / num2);
else msg = "Неправильный оператор" ;
```

- Добавьте следующую строку в конец главного метода для вывода получившейся строки.

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

- Сохраните программу под именем *Args.java*, затем скомпилируйте и запустите с тремя аргументами — целым числом, любым символом арифметической операции (+, -, x и /) и еще одним целым числом.

```

C:\MyJava>javac Args.java
C:\MyJava>java Args 16 + 4
16+4=20
C:\MyJava>java Args 16 - 4
16-4=12
C:\MyJava>java Args 16 x 4
16x4=64
C:\MyJava>java Args 16 / 4
16/4=4
C:\MyJava>

```

- Перезапустите программу с некорректным вторым аргументом и с неверным числом аргументов.

```

C:\MyJava>java Args 16 % 4
Неправильный оператор
C:\MyJava>java Args 16 +
Неверное число аргументов
C:\MyJava>

```

Совет



Ключевое слово **return** используется для выхода из текущего метода. Оно также может возвращать значение в точку, где метод был вызван (см. главу 6).

Совет



Программа сообщит об ошибке, если введены не числовые значения. Подробнее об обработке ошибок см. в конце этой главы.

Обход элементов в цикле

Для того чтобы прочитать все значения, хранящиеся в элементах массива, можно использовать любые типы циклов. Счетчик цикла должен начинаться с индекса первого элемента массива и возрастать до индекса последнего элемента. Заметьте, что индекс последнего элемента в массиве всегда будет на единицу меньше, чем длина массива, поскольку индексы начинаются с нуля.

Весьма полезным считается использованием свойства массива **length** (длина) для условной проверки, определяющей, когда цикл должен завершить свою работу. Это значит, что цикл будет продолжаться до тех пор, пока счетчик не превысит значение индекса последнего элемента.



Loops.java

1. Создайте новую программу с именем **Loops**, содержащую стандартный метод **main**.

```
class Loops
{
    public static void main ( String[] args ) { }
```

2. Внутри главного метода напишите условный оператор, выполняющий проверку того, введены ли какие-либо аргументы из командной строки в массив **args[]**.

```
if ( args.length > 0 ) { }
```

3. Добавьте внутри фигурных скобок условного оператора цикл **for** для вывода значений, хранящихся в каждом элементе.

```
for ( int i = 0 ; i < args.length ; i++ )
{
    System.out.println( "args[" + i + "] = | " + args[i] ) ;
}
```

4. Сохраните программу под именем **Loops.java**, затем скомпилируйте и запустите с аргументами **Java In Easy Steps**.

```
Command Prompt
C:\MyJava>javac Loops.java
C:\MyJava>java Loops Java In easy steps
args[0] is | Java
args[1] is | In
args[2] is | easy
args[3] is | steps
C:\MyJava>
```

5. Отредактируйте файл *Loops.java*, добавив строковый массив, а также цикл **while** для вывода значений, хранящихся в каждом элементе.

```
String[] htm = { "HTML5", "in", "easy", "steps" } ;

int j = 0 ;

while ( j < htm.length )

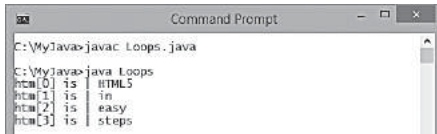
{

    System.out.println( "htm[" +j+ "] - | " + htm[j] ) ;

    j++ ;

}
```

6. Сохраните изменения, затем заново скомпилируйте и запустите программу.



7. Отредактируйте файл *Loops.java*, добавив еще один строковый массив, а также цикл **do while** для вывода значений, хранящихся в каждом элементе.

```
String[] xml = { "XML", "in", "easy", "steps" } ;

int k = 0 ;

if ( xml.length > 0 ) do

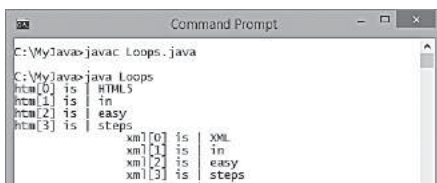
{

    System.out.println( "\t\t+xml["+k+"] - | "+xml[k] ) ;

    k++ ;

} while ( k < xml.length ) ;
```

8. Сохраните изменения, затем заново скомпилируйте и запустите программу.



Совет



Обратите внимание, что оператор **do** предваряется условной проверкой, чтобы перед тем, как попытаться вывести значение первого элемента, убедиться, что массив не пустой.

Изменение значений элемента

Значение любого элемента массива можно изменить, присвоив новое значение конкретному элементу, используя его индекс. К тому же, для того чтобы эффективно разместить элементы, хранящиеся в другом массиве, можно использовать любой тип цикла. Особенно это полезно при комбинировании данных из множественных массивов в один массив объединенных данных.



Elements.java

1. Создайте новую программу с именем **Elements**, содержащую стандартный метод **main**.

```
class Elements
{
    public static void main ( String[] args ) { }
```

2. Добавьте в главный метод проинициализированный целочисленный массив, представляющий собой ежемесячные объемы продаж киоска по кварталам года.

```
int[] kiosk_q1 = { 42000 , 48000 , 50000 } ;
int[] kiosk_q2 = { 52000 , 58000 , 60000 } ;
int[] kiosk_q3 = { 46000 , 49000 , 58000 } ;
int[] kiosk_q4 = { 50000 , 51000 , 61000 } ;
```

3. Добавьте проинициализированные целочисленные массивы, представляющие собой ежемесячные продажи за четыре квартала года.

```
int[] outlet_q1 = { 57000 , 63000 , 60000 } ;
int[] outlet_q2 = { 70000 , 67000 , 73000 } ;
int[] outlet_q3 = { 67000 , 65000 , 62000 } ;
int[] outlet_q4 = { 72000 , 69000 , 75000 } ;
```

4. Теперь создайте пустой целочисленный массив из 12 элементов, в котором будут объединены все ежемесячные продажи, а также создайте целочисленную переменную, в которой будет записана общая величина объема продаж.

```
int[] sum = new int[ 12 ] ;
int total = 0 ;
```

5. Добавьте цикл **for**, для того чтобы занести в каждый элемент пустого массива объединенное значение из других массивов.

```
for ( int i = 0 ; i < kiosk_q1.length ; i++ )
{
    sum[ i ] = kiosk_q1[i] + outlet_q1[i] ;
    sum[i+3] = kiosk_q2[i] + outlet_q2[i] ;
    sum[i+6] = kiosk_q3[i] + outlet_q3[i] ;
    sum[i+9] = kiosk_q4[i] + outlet_q4[i] ;
}
```

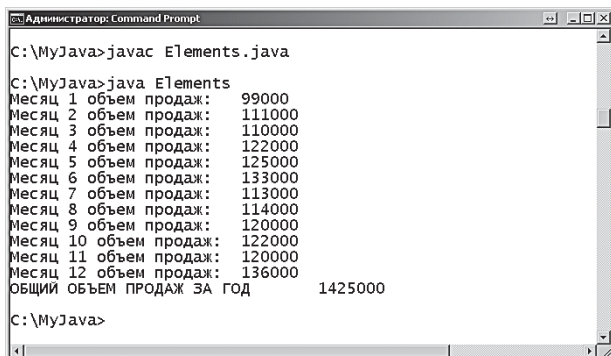
6. Теперь добавьте еще один цикл **for** для вывода каждого из объединенных ежемесячных объемов продаж, а также вычисления общей суммы.

```
for ( int i = 0 ; i < sum.length ; i++ )
{
    System.out.println( "Месяц "+ ( i+1 ) + " объем продаж:\t" + sum[i] ) ;
    total += sum[i] ;
}
```

7. Добавьте последний оператор в конец главного метода для вывода общей суммы (годового объема).

```
System.out.println( "ОБЩИЙ ОБЪЕМ ПРОДАЖ ЗА ГОД\t" + total ) ;
```

8. Сохраните программу под именем *Elements.java*, затем скомпилируйте и запустите.



```
Администратор: Command Prompt
C:\MyJava>javac Elements.java
C:\MyJava>java Elements
Месяц 1 объем продаж: 99000
Месяц 2 объем продаж: 111000
Месяц 3 объем продаж: 110000
Месяц 4 объем продаж: 122000
Месяц 5 объем продаж: 125000
Месяц 6 объем продаж: 133000
Месяц 7 объем продаж: 113000
Месяц 8 объем продаж: 114000
Месяц 9 объем продаж: 120000
Месяц 10 объем продаж: 122000
Месяц 11 объем продаж: 120000
Месяц 12 объем продаж: 136000
ОБЩИЙ ОБЪЕМ ПРОДАЖ ЗА ГОД 1425000
C:\MyJava>
```

На заметку



Счетчик увеличивается на единицу для того, чтобы произвести числа месяца от 1 до 12.

Добавление размеров массива

Внимание



Старайтесь избегать размерности больше, чем три, — это может вас запутать.

В массивах можно хранить множественные наборы элементов, причем каждый из таких наборов станет иметь собственную размерность. К отдельным значениям можно будет обращаться с помощью многомерного массива, используя соответствующие индексы для каждой размерности, например `num [1] [3]`.

Например, чтобы организовать бизнес-ежедневник с записями для каждого дня, потребуется массив из 52 элементов (один на неделю), каждый из которых содержит еще один массив из семи элементов (для каждого дня). Объявление такого массива будет выглядеть следующим образом:

```
int[][] dailyRecord = new int [52] [7] ;
```

Такой массив массивов обеспечивает элемент для каждого рабочего дня. Значения для каждого дня заносятся в многомерный массив с указанием соответствующего индекса для каждого размера. Например, чтобы задать значение для первого дня шестой недели, мы запишем:

```
dailyRecord [5] [0] = 5000 ;
```

Каждый массив содержит собственное свойство длины (`length`), к которому можно обратиться, указав требуемую размерность. Например, для вышеуказанного массива выражение `dailyRecord.length` возвратит значение 52 — размер первого измерения. Чтобы найти размер второго измерения, выражение `dailyRecord[0].length` возвратит значение 7.

Двухмерные массивы часто используются для хранения координат, где первая размерность представляет координату по оси X, а вторая координата — по оси Y, например `point[3] [5]`.

Трехмерные массивы могут использоваться для хранения координат XYZ аналогичным способом, но, конечно, запись `point[4] [8] [2]` сложнее для восприятия.

При организации многомерных массивов очень удобно пользоваться вложенными циклами — каждый уровень цикла может обращаться к элементам соответствующей размерности.

1. Создайте новую программу с именем **Dimensions**, содержащую стандартный метод **main**.

```
class Dimensions
{
    public static void main ( String[] args ) { }
}
```



Dimensions.java

2. В главном методе создайте двухмерный массив для хранения координаты XY.

```
boolean[][] points = new boolean[5][20] ;
```

3. Определите одну точку Y на каждой оси X.

```
points[0][5] = true ;
```

```
points[1][6] = true ;
```

```
points[2][7] = true ;
```

```
points[3][8] = true ;
```

```
points[4][9] = true ;
```

4. Для прохода по первому индексу массива добавьте цикл **for**, вставляя на каждой итерации символ новой строки.

```
for ( int i = 0 ; i < points.length ; i++ )  
{  
    System.out.print( "\n" ) ;  
}
```

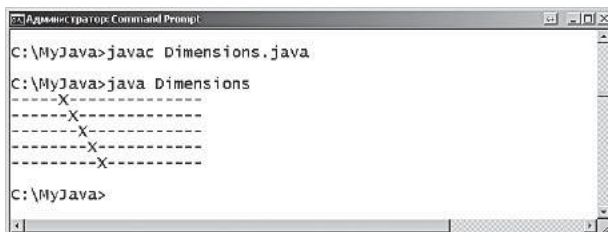
5. Внутри фигурных скобок цикла **for** добавьте еще один цикл **for** для итерации по второму индексу массива.

```
for ( int j = 0 ; j < points[0].length ; j++ ) { }
```

6. Внутри фигурных скобок второго цикла **for** добавьте выражение для вывода символов в соответствии с логическим значением элемента.

```
char mark = ( points[i][j] ) ? 'X' : '-' ;  
System.out.print( mark ) ;
```

7. Сохраните программу под именем *Dimensions.java*, затем скомпилируйте и запустите.



```
Администратор: Command Prompt  
C:\MyJava>javac Dimensions.java  
C:\MyJava>java Dimensions  
-----X-----  
-----X-----  
-----X-----  
-----X-----  
C:\MyJava>
```

На заметку



Логические переменные имеют по умолчанию значение **false**.

Перехват исключений

В программах на этапе выполнения могут возникать проблемы, приводящие к ошибкам (исключениям), которые, в свою очередь, вызывают остановку выполнения программы. Очень часто причиной таких исключений является некорректный пользовательский ввод. Следовательно, хорошо продуманная программа должна попытаться учесть все возможные варианты, при которых пользователь может вызвать исключение на этапе выполнения.

Части кода, в которых возможно возникновение исключений, заключаются в специальные блоки операторов **try catch**. Такие блоки позволяют программе обрабатывать исключения, не приводя к остановкам, а выглядят они следующим образом:

```
try
{
    операторы, где может возникнуть исключение
}
catch( Exception e )
{
    операторы, обрабатывающие исключение
}
```

В скобках, следующих после ключевого слова **catch**, определяется класс исключения, которое нужно перехватить. Класс **Exception** является исключением верхнего уровня, он перехватывает все исключения. Используя многочисленные операторы **catch**, можно перехватывать различные классы исключений более низкого уровня.

Наиболее распространенные исключения — это **NumberFormatException**, которое возникает, когда программа находит значение некорректного типа, а также **ArrayIndexOutOfBoundsException**, которое возникает при попытке обращения к элементу массива с индексом, выходящим за пределы разрешенного диапазона. Для каждого из этих исключений полезно создавать отдельный ответ для объяснения пользователю природы возникшей ошибки.

Блок операторов **try catch** может быть расширен дополнительным необязательным блоком **finally**, содержащий код, который будет выполняться в любом случае, независимо от того, возникло исключение в программе или нет.

Совет



Метод **e.getMessage()** возвращает дополнительную информацию о перехваченных исключениях.

1. Создайте новую программу с именем **Exceptions**, содержащую стандартный метод **main**.

```
class Exceptions
```

```
{
```

```
    public static void main ( String[] args ) { }
```

```
}
```

2. Внутри главного метода напишите оператор **try**, в котором будет выводиться единственный целочисленный аргумент.

```
try
```

```
{
```

```
    int num = Integer.parseInt( args[0] ) ;
```

```
    System.out.println( "Вы ввели: "+ num ) ;
```

```
}
```

3. Добавьте оператор **catch**, чтобы обработать исключение, которое возникнет в том случае, если программа запускается без аргумента.

```
catch( ArrayIndexOutOfBoundsException e )
```

```
{ System.out.println( "Требуется целочисленный аргумент." ) ; }
```

4. Добавьте оператор **catch** для обработки исключения, которое возникнет в том случае, когда программа запускается с нечисловым аргументом.

```
catch( NumberFormatException e )
```

```
{ System.out.println( "Неверный формат аргумента." ) ; }
```

5. Добавьте оператор **finally** в конец программы.

```
finally { System.out.println( "Программа завершила работу." ) ; }
```

6. Сохраните программу под именем *Exceptions.java*, затем скомпилируйте и запустите, пытаясь вызвать исключения.



Exceptions.java

```
Администратор: Command Prompt
c:\МуJava>javac Exceptions.java
c:\МуJava>java Exceptions
Требуется целочисленный аргумент.
Программа завершила работу.
c:\МуJava>java Exceptions TWENTY
Неверный формат аргумента.
Программа завершила работу.
c:\МуJava>java Exceptions 20
Вы ввели: 20
Программа завершила работу.
```


Заключение

- Любые числовые значения можно конвертировать в другие типы числовых данных с помощью приведения типов, а также к строковому типу, используя метод `toString()`.
- Значение строкового типа может быть сконвертировано в целочисленное (`int`) с помощью метода `Integer.parseInt()`, а также в значение плавающего типа (`float`), используя метод `Float.parseFloat()`.
- Массив — это переменная, которая может содержать множественные значения, которые инициализируются списком внутри фигурных скобок.
- Пустой массив может быть создан при помощи ключевого слова `new`.
- Свойство массива `length` содержит целочисленную величину, равную числу элементов этого массива.
- К любому элементу массива можно обращаться по его индексу.
- Главный метод программы создает строковый массив с именем `args`, в котором сохраняются аргументы командной строки.
- Первый аргумент командной строки автоматически сохраняется в элементе `args[0]`.
- Множественные аргументы, передаваемые в программу из командной строки, должны быть разделены пробелом.
- Циклы являются идеальным способом для того, чтобы прочитать значение, хранящееся в элементах массива.
- Данные из нескольких массивов можно комбинировать и формировать из них новый массив объединенных данных.
- Многомерные массивы могут содержать множественные наборы значений элементов, каждый из которых имеет свою размерность.
- Блок операторов `try catch` используется для обработки исключений, возникающих на этапе исполнения.
- Класс `Exception` перехватывает все исключения, включая `NumberFormatException` и `ArrayIndexOutOfBoundsException`.
- Блок `try catch` может быть расширен оператором `finally`, содержащим код, который будет выполняться в любом случае.

5

Работа с данными

В этой главе демонстрируется, как управлять программными данными, используя различные методы из библиотеки языка Java.

- Изучение классов Java
- Математические вычисления
- Округление чисел
- Генерация случайных чисел
- Управление строками
- Сравнение строк
- Поиск строк
- Обработка символов
- Заключение

Изучение классов Java

Язык Java содержит обширную библиотеку протестированного кода, который размещен в так называемых пакетах («packages»). Основным пакетом языка Java является пакет **java.lang**, который по умолчанию имеет доступ к прикладному программному интерфейсу Java (Java API). Это означает, что при создании программ всегда доступны свойства и методы, предоставляемые пакетом **java.lang**. Например, функциональность стандартного вывода, предоставляемая методом **System.out.println()**, на самом деле вызывает метод класса **System**, который является частью пакета **java.lang**.

Содержимое пакета организовано в иерархическом порядке, который позволяет обращаться к любому его элементу, используя точечную запись. Например, класс **System** содержит свойство **out** (поле), которое, в свою очередь, включает метод **println()** — к нему можно обратиться как **System.out.println()**.

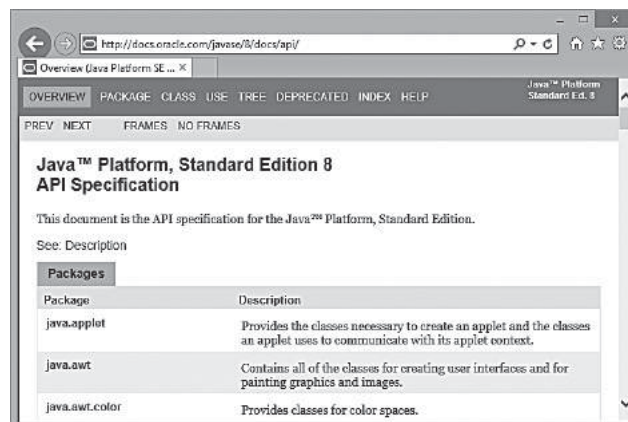
Для того чтобы изучить подробнее классы в Java, можно обратиться к документации, которая предоставляет информацию о любом доступном элементе. Документация доступна во Всемирной паутине по адресу **docs.oracle.com/javase/8/docs/api**. Также вы можете ее скачать для офлайн-изучения. Объем документации достаточно большой, но познакомиться с ней все равно стоит. Для начала полезно ознакомиться со страницей API Overview (Обзор API-функций), которая содержит список всех пакетов с их кратким описанием.

На заметку



Вы можете щелкнуть мышью по ссылке **Frames**, чтобы просматривать документацию в мультиоконном режиме.

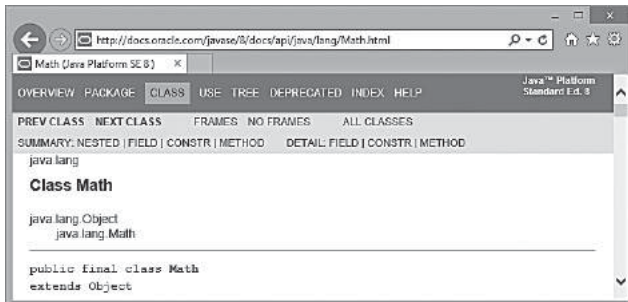
1. Запустите веб-браузер и откройте страницу обзора API-функций по адресу **docs.oracle.com/javase/8/docs/api**.



2. Просмотрите перечисленные в алфавитном порядке пакеты на вкладке **Overview**, затем пролистайте вниз, найдите пакет **java.lang** и щелкните по его ссылке.



3. Просмотрите перечисленные в алфавитном порядке классы в разделе **Class Summary** на соответствующей странице, прокрутите страницу вниз и, найдя класс **Math**, щелкните по его гиперссылке.



4. Просмотрите все методы данного класса, перечисленные в алфавитном порядке в разделе **Method Summary**, затем щелкните на любой из гиперссылок, чтобы ознакомиться с синтаксисом конкретного метода и его назначением.
5. Щелкните на элементе **Package** в меню страницы, чтобы вернуться к странице пакета **java.lang** и изучить другие классы.

Совет



Изучите информацию, доступную по другим ссылкам в меню, чтобы ознакомиться с документацией ближе.

Математические вычисления

В пакете `java.lang` существует класс `Math`, который предоставляет две константы, используемые для математических вычислений. Первая из них — `Math.PI` — хранит значение числа π , а вторая — `Math.E` — хранит основание натурального логарифма. Обе константы имеют тип `double` (двойной точности, с 15 знаками после запятой).



Pi.java

1. Создайте новую программу с именем *Pi*, содержащую стандартный метод `main`.

```
class Pi
{
    public static void main ( String[] args ) { }
```

2. Внутри главного метода объявите и проинициализируйте переменную типа `float` из аргумента командной строки, а также вторую переменную типа `float`, которую нужно получить с помощью приведения типов из константы `Math.PI`.

```
float radius = Float.parseFloat( args[0] ) ;
float shortPi = (float) Math.PI ;
```

3. Проведите математические вычисления, используя приведенные значения и присваивая результаты еще двум переменным типа `float`.

```
float circ = shortPi * ( radius + radius ) ;
float area = shortPi * ( radius * radius ) ;
```

4. Выведите значение константы `Math.PI`, а также ее эквивалента, приведенного к типу `float`, затем выведите результаты вычислений.

```
System.out.print( "Если число Пи рассчитано в диапазоне от " + Math.PI ) ;
System.out.println( " до " + shortPi + "... " ) ;
System.out.println( "Окружность с радиусом " + radius + " см" );
System.out.print( "имеет длину " + circ + " см" ) ;
System.out.println( " и площадь " + area + " кв.см" ) ;
```

5. Сохраните программу под именем *Pi.java*, затем скомпилируйте и запустите.

Вышеуказанный класс `Math` предоставляет множество различных методов, полезных для выполнения математических вычислений. Например, используя метод `Math.pow()`, можно любое заданное число возвести в указанную степень. Первым аргументом является число, заданное в скобках, а вторым — показатель степени, в которую нужно возвести.

Совет



Вычисленное значение π обычно имеет достаточную точность.

```
Администратор Command Prompt
C:\МуJava>javac Pi.java
C:\МуJava>java Pi 5
Если число Пи рассчитано в диапазоне от 3.141592653589793 до 3.1415927...
Окружность с радиусом 5.0 см
имеет длину 31.415928 см
и площадь 78.53982 кв.см
C:\МуJava>_
```

Метод **Math.sqrt()** возвращает квадратный корень числа, указываемый в качестве единственного аргумента.

Оба этих метода возвращают значение типа **double**.

1. Создайте новую программу с именем **Power**, содержащую стандартный метод **main**.

```
class Power
{
    public static void main ( String[] args ) { }
```



Power.java

2. Внутри главного метода объявите и проинициализируйте целочисленную переменную, используя аргумент командной строки.

```
int num = Integer.parseInt( args[0] ) ;
```

3. Произведите математические вычисления, приведя результаты к целочисленному типу.

```
int square = (int) Math.pow( num , 2 ) ;
```

```
int cube = (int) Math.pow( num , 3 ) ;
```

```
int sqrt = (int) Math.sqrt( num ) ;
```

4. Выведите результаты вычислений.

```
System.out.println( num + " в квадрате равно " + square ) ;
```

```
System.out.println( num + " в кубе равно " + cube ) ;
System.out.println( "Квадратный корень из " + num + " равен " + sqrt ) ;
```

5. Сохраните программу под именем *Power.java*, затем скомпилируйте и запустите.

```
Администратор Command Prompt
C:\МуJava>javac Power.java
C:\МуJava>java Power 9
9 в квадрате равно 81
9 в кубе равно 729
Квадратный корень из 9 равен 3
C:\МуJava>_
```

На заметку



Оба этих примера можно улучшить, если добавить блок операторов **try catch**, чтобы отловить пользовательские ошибки, — см. главу 4.

Округление чисел

Класс **Math** внутри пакета **java.lang** предлагает три метода для округления чисел с плавающей точкой до ближайшего целого. Простейший из методов **Math.round()** округляет число, являющееся его аргументом, вверх или вниз до ближайшего целого.

Метод **Math.floor()** производит округление до ближайшего целого вниз, а метод **Math.ceil()** округляет до ближайшего целого вверх.

В то время как метод **Math.round()** возвращает значение целочисленного типа (**int**), методы **Math.floor()** и **Math.ceil()** возвращают значение типа **double**.



Round.java

1. Создайте новую программу с именем **Round**, содержащую стандартный метод **main**.

```
class Round
{
    public static void main ( String[] args ) { }
```

2. Внутри главного метода объявите и проинициализируйте переменную типа **float**.

```
float num = 7.25f ;
```

3. Выведите округленное значение, которое приобрело тип **int** в результате округления.

```
System.out.println(num+"округленное равно "+Math.round(num) );
```

4. Выведите округленное значение в виде значений типа **double**.

```
System.out.println( num+" округленное вниз равно " +Math.floor( num ) );
```

```
System.out.println( num+" округленное вверх равно " + Math.ceil( num ) );
```

5. Сохраните программу под именем *Round.java*, затем скомпилируйте и запустите.

Совет



По умолчанию метод **Math.round()** будет округлять в сторону увеличения — таким образом, 7,5 округлится до 8.

```
Администратор: Command Prompt
C:\МуJava>javac Round.java
C:\МуJava>java Round
7.25 округленное равно 7
7.25 округленное вниз равно 7.0
7.25 округленное вверх равно 8.0
C:\МуJava>
```

Класс **Math** пакета **java.lang** предлагает два метода для сравнения двух числовых значений — это методы **Math.max()** и **Math.min()**, принимающие в качестве двух своих аргументов числа. **Math.max()** возвращает большее из двух чисел, а **Math.min()** — меньшее из них.

Сравниваемые числа, используемые этими двумя методами, могут иметь любой из числовых типов данных, но возвращаемый результат будет всегда иметь тип **double**.

1. Создайте новую программу с именем **Compare**, содержащую стандартный метод **main**.

```
class Compare
{
    public static void main ( String[] args ) { }
```



Compare.java

2. Внутри главного метода объявите и проинициализируйте две переменных типа **float** и типа **int**.

```
float num1 = 24.75f ;
int num2 = 25 ;
```

3. Выведите большее из двух значений.

```
System.out.println( "Наибольшее: " + Math.max( num1, num2 ) ) ;
```

4. Выведите меньшее из двух значений.

```
System.out.println( "Наименьшее: " + Math.min( num1, num2 ) ) ;
```

5. Сохраните программу под именем *Compare.java*, затем скомпилируйте и запустите.

```
Администратор: Command Prompt
C:\MyJava>javac Compare.java
C:\MyJava>java Compare
Наибольшее 25.0
Наименьшее 24.75
C:\MyJava>
```


Генерация случайных чисел

Класс **Math** пакета **java.lang** предоставляет возможность для генерации случайных чисел при помощи метода **Math.random()**, который возвращает случайное число двойной точности (типа **double**) в диапазоне от 0,0 до 0,999. Желаемый диапазон можно расширить при помощи умножения на случайное число. Например, умножив на 10, можно создать случайное число от 0,0 до 9,999. После этого, если округлить полученное число при помощи метода **Math.ceil()**, то полученное целое число попадает в диапазон от 1 до 10.



Random.java

1. Создайте новую программу с именем **Random**, содержащую стандартный метод **main**.

```
class Random
{
    public static void main ( String[] args ) { }
```

2. Внутри главного метода присвойте переменной типа **float** значение случайного числа и выведите это значение.

```
float random = (float) Math.random() ;
System.out.println( "Случайное число: " + random ) ;
```

3. Второй переменной типа **float** присвойте значение случайного числа, умноженное на 10, и также выведите ее значение.

```
float multiplied = random * 10 ;
System.out.println( "Умноженное на 10: " + multiplied ) ;
```

4. Теперь занесите округленное целое значение, полученное в предыдущем шаге случайного числа, в переменную типа **int**, затем выведите ее значение.

```
int randomInt = (int) Math.ceil( multiplied ) ;
System.out.println( "Случайное целое: " + randomInt ) ;
```

5. Сохраните программу под именем *Random.java*, затем скомпилируйте и запустите.

Совет



Программа, моделирующая лотерею и описанная далее, комбинирует все эти три шага в одном операторе.

```
Администратор Command Prompt
C:\MyJava>javac Random.java
C:\MyJava>java Random
Случайное число: 0.4534767
Умноженное на 10: 4.534767
Случайное целое: 5
C:\MyJava>
```

При помощи метода `Math.random()` можно сгенерировать последовательность из шести неповторяющихся целых чисел в диапазоне от 1 до 49 включительно, чтобы смоделировать выпадение номеров в лотерее.

1. Создайте новую программу с именем **Lottery**, содержащую стандартный метод **main**.

```
class Lottery
{
    public static void main ( String[] args ) { }
```



Lottery.java

2. Внутри главного метода создайте целочисленный массив из 50 элементов, а затем заполните элементы с 1 по 49 целыми числами от 1 до 49.

```
int[] nums = new int[50] ;
for( int i = 1 ; i < 50 ; i++ ) { nums[i] = i ; }
```

3. Перемешайте значения в элементах массива от 1 до 49.

```
for( int i = 1 ; i < 50 ; i++ )
{
    int r = (int) Math.ceil( Math.random() * 49 ) ;
    int temp = nums[i] ;
    nums[i] = nums[r] ;
    nums[r] = temp ;
}
```

4. Выведите только те значения, которые содержатся в элементах от 1 до 6.

```
for ( int i = 1 ; i < 7 ; i++ )
{
    System.out.print( Integer.toString( nums[i] ) + " " );
}
```

5. Сохраните программу под именем *Lottery.java*, затем скомпилируйте и запустите три раза, чтобы произвести три различных последовательности чисел.

```
Администратор Command Prompt
C:\MyJava>javac Lottery.java
C:\MyJava>java Lottery
40 16 8 13 2 26
C:\MyJava>java Lottery
35 19 24 45 27 8
C:\MyJava>java Lottery
39 1 22 10 26 37
C:\MyJava>
```

Совет



К данной программе мы вернемся при изучении графического пользовательского интерфейса в главе 10.

**Внимание**

Array.length — это свойство, но **String.length()** — это метод, поэтому он содержит скобки.

Управление строками

В языке Java строка (элемент типа **String**) представляет собой 0 или более символов, заключенных в кавычки. Корректными строками являются, например: **String txt1 = "Моя первая строка" ;**

```
String txt2 = " " ;
```

```
String txt3 = "2" ;
```

```
String txt4 = "null" ;
```

Пустые кавычки в примере с переменной **txt2** инициализируют переменную, содержащую пустую строку. Числовое значение, присвоенное переменной **txt3**, — это строковое представление числа. Ключевое слово языка Java **null** обычно представляет собой отсутствие какого-либо значения, но оно является простым строковым литералом, если заключено в кавычки.

Строка в языке Java, по существу, является набором символов, каждый из которых содержит собственные данные, подобно элементам массива. Поэтому логично представлять строку как массив символов и, работая с ней, применять характеристики массива.

Класс **String** является частью базового пакета **java.lang** и предоставляет метод **length()**, который возвращает размер строки подобно свойству массива **length**. Каждая строковая переменная создается как экземпляр класса **String**, поэтому методы данного класса можно использовать, применив точечную запись к имени переменной. Например, чтобы получить размер строки, сохраненной в переменной **txt**, мы запишем **txt.length()**.

В классе **String** пакета **java.lang** существует альтернатива оператору конкатенации **+** для объединения строковых значений друг с другом — это метод **concat()**, который требует единственного аргумента, определяющего вторую строку, добавляемую к основной. Например, чтобы добавить строку **txt1**, хранящуюся в переменной **txt2**, к строке, хранящейся в **txt1**, мы напишем **txt1.concat(txt2)**.

1. Создайте новую программу с именем **StringLength**, содержащую стандартный метод **main**.

```
class StringLength
{
    public static void main ( String[] args ) { }
```



StringLength.java

2. Внутри главного метода создайте и проинициализируйте две переменные строкового типа.

```
String lang = "Java" ;
String series = " in easy steps" ;
```

3. Добавьте еще одну строковую переменную и присвойте ей значение двух объединенных выше созданных строк.

```
String title = lang.concat( series ) ;
```

4. Выведите объединенную строку, используя знаки кавычек, и ее размер.

```
System.out.print( "\"" + title + "\" содержит " ) ;
System.out.println( title.length() + " символов" ) ;
```

5. Сохраните программу под именем *StringLength.java*, затем скомпилируйте и запустите.

```
Администратор: Command Prompt
C:\MyJava>javac StringLength.java
C:\MyJava>java StringLength
"Java in easy steps" содержит 18 символов
C:\MyJava>
```

На заметку



Знаки кавычек не являются частью строки, поэтому не включаются в подсчет символов, в отличие от пробелов.

Сравнение строк

Класс **String** пакета **java.lang** предоставляет полезный метод **equals()**, он был показан в главе 4, где оценивался аргумент командной строки, представленный аргументом **args[0]**. Данный метод также можно использовать для сравнения любых двух строковых переменных, используя точечную запись — имя метода ставится после имени переменной, а вторая переменная является аргументом. Например, для сравнения переменных **txt2** и **txt1** запишем **txt1.equals(txt2)**. В случае если обе переменные содержат одни и те же символы в одинаковом порядке, то метод возвращает значение **true**, в противном же случае возвращает значение **false**.

Строки, которые содержат одинаковые буквы, но в разных регистрах, не являются равными (например, **Java** и **JAVA**), поскольку у символов различаются коды ASCII. Например, значение символа верхнего регистра **A** равно 65, в то время как символ нижнего регистра **a** имеет значение 97.

В программах очень часто используется сравнение введенной пользователем строки с нужной строкой, чтобы введенная строка соответствовала определенному регистру. Для этих целей класс **String** предлагает два метода: **toUpperCase()** и **toLowerCase()**. Входная строка указывается этим методам в качестве аргумента, а они, в свою очередь, возвращают трансформированную строку. Типичным примером может являться ситуация, когда введенный пользователем строковый пароль преобразуется к нижнему регистру, перед тем как будет сравниваться с верным паролем, хранящимся в программе в нижнем регистре. Это позволит пользователю вводить свой пароль не глядя на регистр, в том случае, если регистры, независимые от паролей, разрешены.

Точечная запись позволяет использовать методы друг за другом, выполняя операции последовательно. Это означает, что запись **toLowerCase().equals()** может быть использована для преобразования строкового значения к нижнему регистру, а затем сравнения полученного результата с указанным аргументом.

1. Создайте новую программу с именем **StringComparison**, содержащую стандартный метод **main**.

```
class StringComparison
{
    public static void main ( String[] args ) { }
```

Внимание



Будьте внимательны при написании заглавных букв **C** в названии методов **toUpperCase** и **toLowerCase**.

- Внутри главного метода создайте и проинициализируйте строковую переменную, содержащую корректный пароль из символов нижнего регистра.

```
String password = "bingo" ;
```

- Добавьте блок операторов **try catch**, для того чтобы перехватить исключение, которое возникнет в случае, если аргумент, представляющий пароль, не будет введен.

```
try {}  
  
catch( Exception e )  
{  
    System.out.println( "Требуется ввод пароля." ) ;  
}
```

- Добавьте блок **if else** в фигурные скобки оператора **try** для оценки введенного пользователем аргумента.

```
if ( args[0].toLowerCase().equals( password ) )  
{  
    System.out.println( "Пароль подтвержден." ) ;  
}  
  
else  
{  
    System.out.println( "Неверный пароль." ) ;  
}
```

- Сохраните программу под именем *StringComparison.java*, затем скомпилируйте и запустите.



StringComparison.java

```
Администратор: Command Prompt  
c:\MyJava>javac StringComparison.java  
c:\MyJava>java StringComparison BINGO  
Пароль подтвержден.  
c:\MyJava>java StringComparison bimbo  
Неверный пароль.  
c:\MyJava>java StringComparison  
Требуется ввод пароля.  
c:\MyJava>
```

Поиск строк

Класс `String` пакета `java.lang` предлагает методы `startsWith()` и `endsWith()` для сравнения частей строк. Данные метода весьма полезны, например, при нахождении строк, имеющих одинаковое начало либо одинаковое окончание. Когда часть строки соответствует указанному аргументу, метод возвращает значение `true`, в противном случае возвращает значение `false`.

Существует возможность скопировать часть какой-то строки, указав позицию первого символа, с которого нужно начинать копировать, в качестве аргумента для метода `substring()`. Данный метод возвратит так называемую подстроку к оригинальной строке, начиная с указанной стартовой позиции и заканчивая последним символом первоначальной строки.

Метод `substring()` может принимать необязательный второй аргумент, который будет указывать позицию последнего копируемого символа. В этом случае метод возвратит подстроку первоначальной строки, начиная с указанной стартовой позиции и заканчивая указанной конечной позицией.

Для поиска символа или подстроки в строке используется метод `indexOf()`. Данный метод возвращает числовую позицию первого вхождения указанного символа или подстроки внутри обрабатываемой строки. Если соответствие не найдено, то метод возвращает значение `-1`.



StringSearch.java

1. Создайте новую программу с именем `StringSearch`, содержащую стандартный метод `main`.

```
class StringSearch
{
    public static void main ( String[] args ) { }
```

2. Внутри главного метода создайте и проинициализируйте строковый массив, содержащий названия книг.

```
String[] books =
{ "Java in easy steps", "XML in easy steps" ,
  "HTML in easy steps" , "CSS in easy steps" ,
  "Gone With the Wind" , "Drop the Defense" } ;
```

3. Создайте и проинициализируйте три целочисленных переменных-счетчика.

```
int counter1 = 0 , counter2 = 0 , counter3 = 0 ;
```

4. Добавьте цикл **for** для обхода строкового массива, выводя первые четыре символа каждого названия.

```
for ( int i = 0 ; i < books.length ; i++ )
{
    System.out.print( books[i].substring( 0,4 ) + " | " ) ;
}
```

5. Добавьте в блок цикла **for** оператор для подсчета названий, содержащих указанное окончание.

```
if ( books[i].endsWith( "in easy steps" ) ) counter1++ ;
```

6. Добавьте еще один оператор в блок цикла **for** для подсчета названий, содержащих указанное начало.

```
if ( books[i].startsWith( "Java" ) ) counter2++ ;
```

7. Добавьте еще один оператор для подсчета названий, не содержащих указанную подстроку.

```
if ( books[i].indexOf( "easy" ) == -1 ) counter3++ ;
```

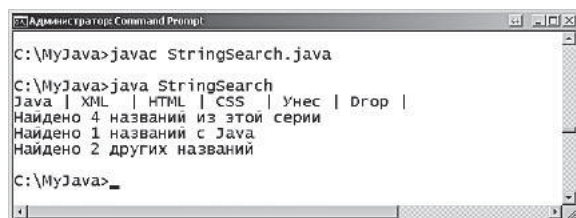
8. В конец главного метода добавьте следующие операторы для вывода результатов каждого поиска.

```
System.out.println( "\nНайдено " + counter1 + " названий из этой серии" ) ;
```

```
System.out.println( "Найдено " + counter2 + " названий с Java" ) ;
```

```
System.out.println( "Найдено " + counter3 + " других названий" ) ;
```

9. Сохраните программу под именем *StringSearch.java*, затем скомпилируйте и запустите.



```

C:\MyJava>javac StringSearch.java
C:\MyJava>java StringSearch
Java | XML | HTML | CSS | Унес | Drop |
Найдено 4 названий из этой серии
Найдено 1 названий с Java
Найдено 2 других названий
C:\MyJava>

```

На заметку



Оператор ! NOT не может использоваться для проверки того, что метод `indexOf()` выдал ошибку, поскольку он возвращает не логическое значение, а целое.

Обработка символов

Класс **String** пакета **java.lang** предоставляет метод **trim()**, который используется для удаления любых пустых символов в начале и в конце строки, указанной в качестве аргумента этому методу. При этом удаляются все лишние пробелы, знаки новой строки и табуляция, и возвращается укороченная версия новой строки.

К отдельному символу строки можно обратиться, указав номер, соответствующий его позиции в строке, в качестве аргумента методу **charAt()**. Данный метод обращается со строкой как с массивом символов, в котором первый символ находится в нулевой позиции – подобно массивам, чьи элементы индексируются, начиная с нуля. Таким образом, к первому символу строки можно обратиться при помощи записи **charAt(0)**, ко второму символу — **charAt(1)** и так далее.

Поскольку индексы начинаются с нуля, то последний символ в строке будет всегда на единицу меньше, чем общее число символов в этой строке. Это означает, что последний символ будет иметь индекс, эквивалентный числу **length() - 1**. Следовательно, чтобы обратиться к последнему символу строки с именем **str**, нужно записать **str.charAt(str.length() -1)**.

Класс **String** предлагает специальный метод **replace()** для замены всех вхождений определенного символа другим символом. Данный метод принимает два аргумента, которые определяют заменяемый символ и символ, который встает на его место. Например, для замены в строке всех символов **a** на символы **z**, мы запишем **replace('a' , 'z')**.

Чтобы проверить, содержит ли строка какие-либо символы, существует специальный метод **isEmpty()**. Данный метод возвратит значение **true**, если строка совершенно пустая, в противном случае он будет возвращать значение **false**.



CharacterSwap.java

1. Создайте новую программу с именем **CharacterSwap**, содержащую стандартный метод **main**.

```
class CharacterSwap
{
    public static void main ( String[] args ) { }
}
```

2. Внутри главного метода объявите и проинициализируйте пустую строковую переменную.

```
String txt = "" ;
```

3. Присвойте переменной определенное значение из каких-либо символов, добавив в начало и конец несколько пробелов.

```
if ( txt.isEmpty() ) txt = " Боррокудо " ;
```

4. Выведите значение строковой переменной и количество символов, которые она содержит.

```
System.out.println( "Строка: " + txt ) ;
```

```
System.out.println( "Длина первоначальной строки: " + txt.length() ) ;
```

5. Удалите пробелы в начале и конце, а затем выведите значение переменной и ее размер заново.

```
txt = txt.trim() ;
```

```
System.out.println( "Строка: " + txt ) ;
```

```
System.out.println( "Длина строки: " + txt.length() ) ;
```

6. Выведите первый символ строки.

```
char initial = txt.charAt(0) ;
```

```
System.out.println( "Первая буква: " + initial ) ;
```

7. Теперь выведите последний символ строки.

```
initial = txt.charAt( ( txt.length() -1 ) ) ;
```

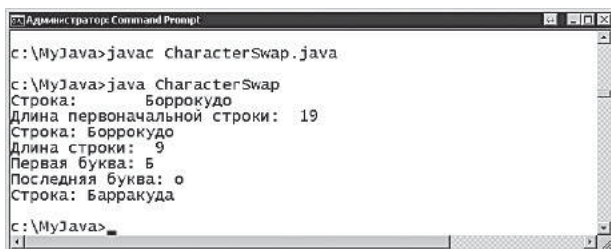
```
System.out.println( "Последняя буква: " + initial ) ;
```

8. Замените все символы "o", встречающиеся в строке, символами "a".

```
txt = txt.replace( 'o' , 'a' ) ;
```

```
System.out.println( "Строка: " + txt ) ;
```

9. Сохраните программу под именем *CharacterSwap.java*, затем скомпилируйте и запустите.



```

c:\MyJava>javac CharacterSwap.java
c:\MyJava>java CharacterSwap
Строка:   Боррокудо
Длина первоначальной строки:  19
Строка: Боррокудо
Длина строки:  9
Первая буква: Б
Последняя буква: о
Строка: Барракуда
c:\MyJava>

```

Заключение

- В документации по языку Java можно найти информацию о методах и свойствах для каждого класса.
- Базовые классы языка Java содержатся в пакете `java.lang`.
- Класс `Math` предоставляет константы `Math.PI` и `Math.E`.
- Метод `Math.pow()` осуществляет возведение в указанную степень, а метод `Math.sqrt()` возвращает квадратный корень указанного числа.
- Числа можно округлять до целых значений, используя методы `Math.round()`, `Math.floor()` и `Math.ceil()`.
- При помощи методов `Math.max()` и `Math.min()` можно сравнивать числа.
- Метод `Math.random()` возвращает случайное число двойной точности в диапазоне от 0,0 до 0,9999999999999999.
- Строка представляет собой 0 или более символов, заключенных в знаки кавычек.
- Подобно свойству массива `length` метод `length()` возвращает размер строки.
- Метод `concat()` используется для добавления одной строки к другой.
- Метод `equals()` возвращает значение `true` только в том случае, когда две строки содержат идентичные символы в одинаковом порядке.
- Регистр символа строки можно изменять, используя методы `toUpperCase()` и `toLowerCase()`.
- Используя методы `startsWith()` и `endsWith()`, можно сравнивать строки.
- При помощи методов `indexOf()` и `substring()` можно производить поиск подстроки в указанной строке.
- Метод `isEmpty()` возвращает значение `true` только в том случае, когда строка не содержит ни одного символа.
- Для работы с символами строки используются методы `trim()`, `charAt()` и `replace()`.

6

Создание классов

*В этой главе
демонстрируется, как
создавать Java-программы,
применяющие
множественные методы
и классы.*

- Программа как набор методов
- Область видимости
- Использование множественных классов
- Расширение существующего класса
- Создание объектного класса
- Создание экземпляра объекта
- Инкапсуляция свойств
- Создание объектных данных
- Заключение

Программа как набор методов

Программы обычно разбиваются на отдельные методы — таким образом создаются модули кода, каждый из которых выполняет определенную задачу и может быть вызван какое угодно количество раз по мере необходимости. Такая разбивка программы на множественные методы позволяет также легче отслеживать ошибки, поскольку каждый из методов может быть проверен отдельно. Методы можно объявлять внутри фигурных скобок, которые следуют за объявлением класса, при этом используются те же самые ключевые слова, что и при объявлении главного метода. Любому новому методу нужно давать имя, следуя стандартным договоренностям именования, а также указывать по необходимости аргументы в скобках после имени метода.



Methods.java

1. Создайте новую программу с именем **Methods**, содержащую стандартный метод **main**.

```
class Methods
{
    public static void main ( String[] args ) { }
```

2. Внутри фигурных скобок главного метода добавьте операторы для вывода сообщения и вызова второго метода с именем **sub**.

```
System.out.println( "Сообщение из главного метода." ) ;
sub() ;
```

3. После главного метода перед последней фигурной скобкой класса добавьте второй метод для вывода сообщения.

```
public static void sub()
{
    System.out.println( "Сообщение из метода sub." ) ;
}
```

4. Сохраните программу под именем *Methods.java*, затем скомпилируйте и запустите.

На заметку



Если метод вызывается без аргументов, то достаточно написать только его имя и пустые скобки.

```
Администратор Command Prompt
C:\MyJava>javac Methods.java
C:\MyJava>java Methods
Сообщение из главного метода.
Сообщение из метода sub.
C:\MyJava>
```

В классе могут содержаться несколько методов с одним и тем же именем, только если они имеют либо различное число аргументов, либо аргументы различного типа. Это называется «перегрузкой» методов.

1. Создайте новую программу с именем **Overload**, содержащую стандартный метод **main**.

```
class Overload
{
    public static void main ( String[] args ) { }
```



Overload.java

2. Внутри фигурных скобок главного метода добавьте три оператора, вызывающих различные перегруженные методы и передающие им значения аргументов.

```
System.out.println( write( 12 ) ) ;
System.out.println( write( "Двенадцать" ) ) ;
System.out.println( write( 4 , 16 ) ) ;
```

3. После главного метода перед последней фигурной скобкой класса добавьте три перегруженных метода, каждый из которых возвращает строку вызвавшему его оператору.

```
public static String write( int num )
{ return ( "Переданное целое " + num ) ; }

    public static String write( String num )
{ return ( "Переданное строковое " + num ) ; }

    public static String write( int num1 , int num2 )
{ return ( "Результат равен " + ( num1 * num2 ) ) ; }
```

4. Сохраните программу под именем *Overload.java*, затем скомпилируйте и запустите.

```
Администратор Command Prompt
c:\MyJava>javac Overload.java
c:\MyJava>java Overload
Переданное целое 12
Переданное строковое двенадцать
Результат равен 64
c:\MyJava>
```



Внимание

В объявлении каждого из перегруженных методов должно быть указано, что метод возвращает строковую величину (используется **String**, а не **void**).

Область видимости

Переменная, которая объявляется внутри метода, доступна только в пределах этого метода — ее так называемая область видимости ограничена данным методом, в котором она объявлена. Это означает, что в другом методе можно объявить переменную с точно таким же именем без какого-либо конфликта.



Scope.java

1. Создайте новую программу с именем **Scope**, содержащую стандартный метод **main**.

```
class Scope
{
    public static void main ( String[] args ) { }
```

2. Внутри фигурных скобок главного метода объявите и проинициализируйте локальную строковую переменную, а затем выведите ее значение.

```
String txt = "Это локальная переменная метода main" ;

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

3. После главного метода перед последней фигурной скобкой класса добавьте еще один метод с именем **sub**.

```
public static void sub( ) { }
```

4. Внутри фигурных скобок метода **sub** объявите и проинициализируйте локальную строковую переменную с тем же именем, что и переменная в главном методе.

```
String txt = "Это локальная переменная метода sub" ;

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

5. Добавьте вызов метода **sub** в конец метода **main**.

```
sub() ;
```

6. Сохраните программу под именем **Scope.java**, затем скомпилируйте и запустите.

На заметку



Переменная-счетчик, объявленная в цикле **for**, недоступна снаружи цикла — ее область видимости ограничена блоком операторов цикла.

```
Администратор: Command Prompt
C:\МуJava>javac Scope.java
C:\МуJava>java Scope
Это локальная переменная метода main
Это локальная переменная метода sub
C:\МуJava>
```

Ключевое слово **static**, которое используется в объявлениях метода, указывает, что данный метод является «методом класса» — он доступен из любого другого метода данного класса.

Аналогично, при помощи ключевого слова **static** может быть объявлена «переменная класса», которая доступна глобально для всего класса. Объявление этой переменной должно быть выполнено перед объявлением главного метода сразу после фигурных скобок, следующих за объявлением класса.

В программе могут встретиться одноименные глобальная переменная класса и локальная переменная метода. Локальная переменная метода имеет приоритет до тех пор, пока глобальная переменная класса не будет вызвана при помощи точечной записи с использованием имени класса, либо не объявлена локальная переменная с тем же именем.

7. Отредактируйте код в файле *Scope.java*, добавив глобальную переменную класса с тем же именем, что и локальная переменная метода.

```
final static String txt =
    "Это глобальная переменная класса Scope" ;
```

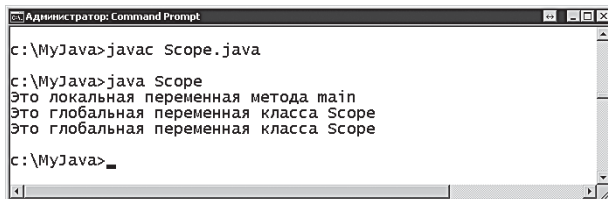
8. Добавьте оператор в конец главного метода для вывода значений глобальной переменной класса.

```
System.out.println( Scope.txt ) ;
```

9. Закомментируйте строку, которая объявляет локальную переменную в методе **sub** — таким образом, оператор вывода будет теперь обращаться к глобальной переменной с тем же именем.

```
//String txt = "Это локальная переменная метода sub";
```

10. Сохраните изменения, затем снова скомпилируйте и запустите программу на выполнение, чтобы увидеть изменения.



```
Администратор: Command Prompt
c:\MyJava>javac Scope.java
c:\MyJava>java Scope
Это локальная переменная метода main
Это глобальная переменная класса Scope
Это глобальная переменная класса Scope
c:\MyJava>
```

Совет



По возможности используйте локальные переменные метода, чтобы избежать конфликтов, — глобальные переменные класса используются в основном для констант.

Использование множественных классов

Подобно тому, как программа может содержать несколько методов, большие программы могут состоять из нескольких классов, каждый из которых представляет свою функциональность. Такая модульность предпочтительнее варианта, когда вся программа пишется при помощи одного класса, и это делает процесс отладки более простым.

Ключевое слово **public**, которое добавляется при объявлении, является так называемым модификатором доступа, который определяет видимость элемента для других классов. Его можно использовать в объявлении класса, чтобы явно указать, что класс будет доступен (виден) для других классов. Если ключевое слово **public** опущено, по умолчанию разрешается доступ из других локальных классов. Однако для главного метода всегда нужно использовать ключевое слово **public**, для того чтобы этот метод был виден компилятору.



Multi.java

1. Создайте новую программу под именем **Multi**, содержащую стандартный метод **main** (включая ключевое слово **public**, как обычно).

```
class Multi
{
    public static void main ( String[] args ) { }
```

2. Внутри фигурных скобок главного метода объявите и проинициализируйте переменную строкового типа, а затем выведите ее содержимое.

```
String msg = "Это локальная переменная класса Multi" ;
System.out.println( msg ) ;
```

3. Выведите содержимое константы с именем **txt** из класса **Data**.

```
System.out.println( Data.txt ) ;
```

4. Вызовите метод с именем **greeting** из класса **Data**.

```
Data.greeting() ;
```

5. Вызовите метод **line** из класса **Draw**.

```
Draw.line() ;
```

6. Сохраните программу под именем *Multi.java*.

Совет



Компилятор автоматически находит классы в файлах с расширением *.java* и создает скомпилированные файлы с расширением *.class* для каждого из них.

7. Начните новый файл с создания класса **Data**.

```
class Data
{
}
```



Data.java

8. Объявите и проинициализируйте константу класса.

```
public final static String txt =
    "Это глобальная переменная класса Data" ;
```

9. Добавьте метод класса.

```
public static void greeting
{
    System.out.print( "Это глобальный метод " ) ;
    System.out.println( "класса Data" ) ;
}
```

10. Сохраните файл под именем *Data.java* в том же каталоге, что и программа *Multi.java*.

11. Создайте новый файл, создав класс **Draw** и метод **line** для доступа по умолчанию — без ключевого слова **public**.

```
class Draw
{
    static void line()
    {
        System.out.println( " _____ " );
    }
}
```



Draw.java

12. Сохраните файл под именем *Draw.java* в том же каталоге, что и программа *Multi.java*, затем скомпилируйте и запустите.

```

C:\МуJava>javac Multi.java
C:\МуJava>java Multi
Это локальная переменная класса Multi
Это глобальная переменная класса Data
Это глобальный метод класса Data
C:\МуJava>

```

На заметку



Ключевое слово **public** разрешает доступ из любого другого класса, а доступ по умолчанию разрешает открывать доступ только для классов одного пакета.

Расширение существующего класса

Класс может наследовать свойства любого другого класса, если его объявить при помощи ключевого слова **extends** с указанием имени класса, из которого следует наследовать свойства. Например, следующее объявление `class Extra extends Base` наследует из класса **Base**.

Класс, который наследует свойство, называется подклассом, а класс, от которого наследуется свойство, называется суперклассом. В объявлении, представленном выше, класс **Base** является суперклассом, а класс **Extra** — подклассом.

Методы и переменные, созданные в суперклассе, обычно могут быть использованы таким образом, как будто они были созданы в подклассе, при условии, если они не объявлены с ключевым словом **private**, которое запрещает доступ извне оригинального класса.

Метод в подклассе будет перегружать метод с тем же именем из суперкласса, только если их аргументы не различаются. К методу суперкласса можно явно обратиться, используя имя класса и точечную запись. Например, `SuperClass.run()`.

Следует отметить, что оператор **try catch** в любом методе суперкласса не перехватывает исключения, которые появляются в подклассе — вызываемый оператор нужно заключать в свой собственный блок операторов **try catch** для перехвата таких исключений.



SuperClass.java

1. Начните новый класс с именем **SuperClass**.

```
class SuperClass { }
```

2. Внутри фигурных скобок класса добавьте метод, который выводит строку для идентификации.

```
public static void hello( )  
{  
    System.out.println( "Привет из Суперкласса" ) ;  
}
```

3. Добавьте второй метод, который пытается вывести переданный аргумент, затем сохраните файл под именем *SuperClass.java*.

```
public static void echo( String arg )  
{  
    try  
    { System.out.println( "Вы ввели: " + arg ) ; }  
    catch( Exception e )  
    { System.out.println( "Требуется аргумент" ) ; }  
}
```

4. Создайте новую программу с именем **SubClass**, которая расширяет класс **SuperClass**.

```
class SubClass extends SuperClass
{
    public static void main ( String[] args ) { }
```



SubClass.java

5. После главного метода добавьте еще один метод, который выводит строку для идентификации, перегружая наследуемый метод с тем же именем.

```
public static void hello()
{
    System.out.println( "Привет из Подкласса" ) ;
}
```

6. Внутри фигурных скобок главного метода добавьте вызов перегружаемого метода, а затем явный вызов метода с тем же именем в суперклассе.

```
hello() ;
SuperClass.hello() ;
```

7. Добавьте вызов другого наследуемого метода.

```
echo( args[0] ) ;
```

8. Сохраните программу под именем *SubClass.java*, затем скомпилируйте и запустите, не вводя аргумент в командной строке.

```
Администратор Command Prompt
C:\MyJava>javac SubClass.java
C:\MyJava>java SubClass
Привет из Подкласса
Привет из Суперкласса
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
    at SubClass.main(SubClass.java:7)
C:\MyJava>
```

На заметку

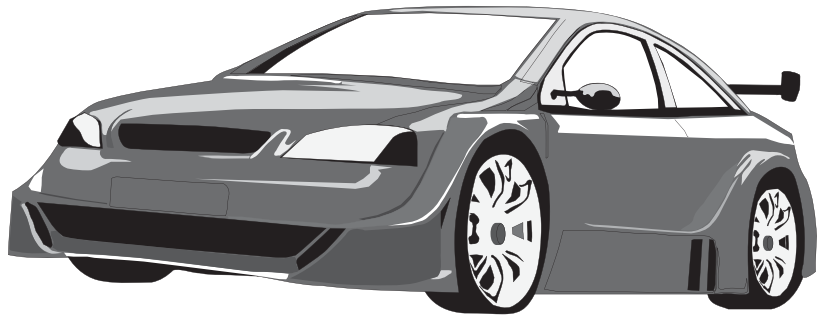


Более подробную информацию по перехвату исключений см. в главе 4.

9. Отредактируйте файл *SubClass.java*, добавив для вызова метода свой собственный блок операторов **try catch**, перехватывающий исключения, а затем скомпилируйте и заново запустите программу, чтобы увидеть, что проблема решена.

Создание объектного класса

Объекты реального мира, окружающие нас, можно описать при помощи атрибутов (характеристик или свойств), а также действий, которые они выполняют. Например, автомобиль можно описать при помощи атрибутов «красный», «купе», а также при помощи действия «ускоряется». Применительно к Java-программированию все это можно представить при помощи класса **Car**, содержащего свойство **color** и **bodyType**, а также метод **accelerate()**.



Поскольку в Java-программировании широко применяются атрибуты и методы объекта, говорят, что Java является объектно-ориентированным языком программирования.

Объекты в Java создаются при помощи определения класса в качестве шаблона, из которого могут быть произведены копии (экземпляры). Каждому экземпляру класса можно присвоить атрибуты и методы, описывающие объект. Далее мы создаем класс **Car** в качестве шаблона класса с атрибутами и методами по умолчанию, описанными выше. После этого создается экземпляр класса **Car**, который наследует те же самые атрибуты и методы по умолчанию.



FirstObject.java

1. Начните новый шаблон класса с именем **Car**.

```
class Car
{
}
```

2. Внутри фигурных скобок класса **Car** объявите и проинициализируйте две глобальные строковые константы, описывающие атрибуты.

```
public final static String color = "Красный" ;  
public final static String bodyType = "Купе" ;
```

3. Добавьте глобальный метод, описывающий действие.

```
public static String accelerate()  
{  
    String motion = "Ускоряется..." ;  
    return motion ;  
}
```

4. После класса **Car** начните новый класс с именем **FirstObject**, содержащий стандартный метод **main**.

```
class FirstObject  
{  
    public static void main ( String[] args ) { }  
}
```

5. Внутри фигурных скобок главного метода добавьте операторы для вывода каждого из значений атрибутов класса **Car**, а также вызова его метода.

```
System.out.println( "Цвет " + Car.color ) ;  
System.out.println( "Тип кузова " + Car.bodyType ) ;  
System.out.println( Car.accelerate() ) ;
```

6. Сохраните программу под именем *FirstObject.java*, затем скомпилируйте и запустите.



```
Администратор: Command Prompt  
C:\МуJava>javac FirstObject.java  
C:\МуJava>java FirstObject  
Цвет Красный  
Тип кузова Купе  
Ускоряется...  
C:\МуJava>
```

На заметку



Ключевое слово **static** объявляет переменные и методы класса в данном случае как члены класса **Car**.

Совет



Классы объектов обычно создаются перед классом программы, содержащим метод **main**.

Создание экземпляра объекта

Каждый класс содержит встроенный метод, который можно использовать для создания нового экземпляра этого класса. Этот метод называется «конструктор». Он имеет то же самое имя, что и класс, и запускается с ключевым словом **new**.

Каждый экземпляр класса наследует атрибуты и действия объекта. Принцип наследования является одним из основных в Java-программировании. Благодаря ему программы могут использовать «готовые» свойства.

Для большей гибкости шаблоны класса можно определять в отдельном файле, который затем можно использовать в разных программах.



Car.java

1. Начните новый файл, скопировав шаблон класса **Car** из предыдущего примера.

```
class Car
{
    public final static String color = "Красный" ;
    public final static String bodyType = "Купе" ;
    public static String accelerate()
    {
        String motion = "Ускоряется..." ;
        return motion ;
    }
}
```

2. Сохраните файл под именем *Car.java*.



FirstInstance.java

3. Создайте новую программу с именем **FirstInstance**, содержащую стандартный метод **main**.

```
class FirstInstance
{
    public static void main ( String[] args ) { }
```

- Внутри фигурных скобок главного метода добавьте операторы для вывода каждого из значений атрибутов класса **Car**, а затем вызовите его метод.

```
System.out.println( "Цвет автомобиля " + Car.color ) ;  
System.out.println( "Тип кузова "+ Car.bodyType ) ;  
System.out.println( Car.accelerate() ) ;
```

- Теперь добавьте операторы для создания экземпляра **Porsche** класса **Car**.

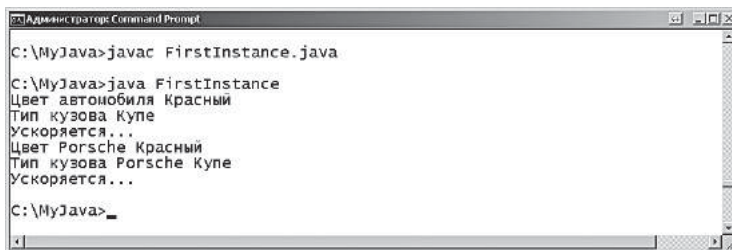
Car class

```
Car Porsche = new Car() ;
```

- Добавьте операторы для вывода унаследованных значений каждого атрибута **Porsche**, а также вызова его метода.

```
System.out.println( "Цвет Porsche " + Porsche.color ) ;  
System.out.println( "Тип кузова Porsche "+ Porsche.bodyType ) ;  
System.out.println( Porsche.accelerate() ) ;
```

- Сохраните программу под именем *FirstInstance.java* рядом с файлом шаблона *Car.java*, затем скомпилируйте и запустите.



```
Администратор: Command Prompt  
C:\MyJava>javac FirstInstance.java  
C:\MyJava>java FirstInstance  
Цвет автомобиля Красный  
Тип кузова Купе  
Ускоряется...  
Цвет Porsche Красный  
Тип кузова Porsche Купе  
Ускоряется...  
C:\MyJava>
```

Для нового объекта **Porsche** создается виртуальный класс, который копирует исходный класс **Car**. Оба этих объекта содержат статические переменные класса и метод класса, к которым можно обратиться, используя точечную запись и имя класса, но поскольку они доступны глобально, то это является не очень хорошей практикой программирования.

В данном примере демонстрируется, как экземпляр объекта наследует свойства исходного класса. В следующем примере используются нестатические члены — переменная экземпляра и метод экземпляра, к которым нельзя обратиться извне класса, поскольку они глобально не доступны, и это является хорошей практикой программирования.

На заметку



Вы не можете обратиться к переменной *motion* напрямую — она находится за пределами видимости внутри объявления метода.

Совет



Компилятор автоматически находит класс **Car** во внешнем файле *.java* и создает скомпилированный файл *.class* для него.

Инкапсуляция свойств

Когда нужно описать переменные и методы объекта и защитить их от обработки внешним программным кодом, используется ключевое слово **private** (закрытый). Для вызова значений переменных, а также вызова методов, в объект можно включать общедоступные методы. Данный прием инкапсулирует переменные и методы внутри структуры объекта. Это демонстрируется в следующих шагах, которые воспроизводят предыдущий пример с инкапсуляцией метода и атрибутов.



SafeInstance.java

1. Начните новый класс с именем **Car**.

```
class Car
{
}
```

2. Внутри фигурных скобок класса объявите три закрытые строковые переменные для хранения атрибутов объекта.

```
private String maker ;
private String color ;
private String bodyType ;
```

3. Добавьте закрытый метод, описывающий действие.

```
private String accelerate()
{
    String motion = "Ускоряется..." ;
    return motion ;
}
```

4. Добавьте общедоступный метод для присваивания значения переданного аргумента каждой из закрытых переменных.

```
public void setCar( String brand , String paint , String style )
{
    maker = brand ;
    color = paint ;
    bodyType = style ;
}
```

5. Добавьте еще один общедоступный метод, обращающийся к значениям закрытых переменных и вызывающий закрытый метод.

```

{
System.out.println( maker + " цвет " + color ) ;
System.out.println( maker + " тип кузова " + bodyType ) ;
System.out.println( maker + " " + accelerate() + "\n" ) ;
}

```

6. После класса **Car** начните еще один класс с именем **SafeInstance**, содержащий стандартный метод **main**.

```

class SafeInstance
{
    public static void main ( String[] args) { }
}

```

7. Внутри фигурных скобок главного метода добавьте оператор, создающий экземпляр класса **Car**.

```
Car Porsche = new Car() ;
```

8. Добавьте оператор, который вызывает общедоступный метод класса **Car** для присваивания значений его закрытым переменным.

```
Porsche.setCar( "Porsche" , "Красный" , "Купе" ) ;
```

9. Теперь добавьте оператор для вызова еще одного общедоступного метода класса **Car**, вызывающего сохраненные значения атрибутов, а также закрытый метод, определяющий действие.

```
Porsche.getCar() ;
```

10. Создайте еще один экземпляр, сначала присвоив, а потом вызвав значение.

```
Car Bentley = new Car() ;
```

```
Bentley.setCar( "Bentley" , "Зеленый" , "Седан" ) ; Bentley.getCar() ;
```

11. Сохраните программу под именем *SafeInstance.java*, затем компилируйте и запустите.

На заметку



Проинициализированная строковая переменная хранит значение **null** — таким образом, вызов метода **getCar()** перед методом **setCar()** возвратит значение **null** каждой из переменных.

```

C:\MyJava>javac SafeInstance.java
C:\MyJava>java SafeInstance
Porsche цвет Красный
Porsche тип кузова Купе
Porsche Ускоряется...

Bentley цвет Зеленый
Bentley тип кузова Седан
Bentley Ускоряется...

C:\MyJava>

```

Создание объектных данных

Метод конструктора объекта можно вызывать напрямую в классе объекта, для того чтобы проинициализировать его переменные. Это позволит хранить отдельно объявление и присваивание, что является хорошим стилем программирования. Этот пример демонстрируется в следующих шагах, которые воспроизводят предыдущий пример с инкапсулируемыми методом и атрибутами вместе с инициализацией при помощи конструктора.



Constructor.java

1. Начните новый класс с именем **Car**.

```
class Car
{
}
```

2. Внутри фигурных скобок класса объявите три закрытые строковые переменные для хранения атрибутов объекта.

```
private String maker ;
private String color ;
private String bodyType ;
```

3. Добавьте метод constructor, в котором инициализируются все три переменные со значениями атрибутов.

```
public Car()
{
    maker = "Porsche" ;
    color = "Серебристый" ;
    bodyType = "Купе" ;
}
```

4. Добавьте закрытый метод, описывающий действие.

```
private String accelerate()
{
    String motion = "Ускоряется..." ;
    return motion ;
}
```

5. Добавьте общедоступный метод для присваивания значений переданного аргумента каждой из закрытых переменных.

Внимание



В объявлениях конструктора не указываются типы данных.

```

public void setCar( String brand , String paint , String style )
{
    maker = brand ;
    color = paint ;
    bodyType = style ;
}

```

6. Добавьте еще один общедоступный метод, который выводит значения закрытых переменных и вызывает закрытый метод.

```

public void getCar()
{
    System.out.println( maker + " цвет " + color ) ;
    System.out.println( maker + " тип кузова " + bodyType ) ;
    System.out.println( maker + " " + accelerate() + "\n" ) ;
}

```

7. После класса **Car** добавьте еще один класс с именем **Constructor**, содержащий стандартный метод **main**.

```

class Constructor
{
    public static void main ( String[] args ) { }
}

```

8. Внутри фигурных скобок главного метода добавьте операторы для создания экземпляра класса **Car** и вызова первоначальных значений по умолчанию.

```

Car Porsche = new Car() ;
Porsche.getCar() ;

```

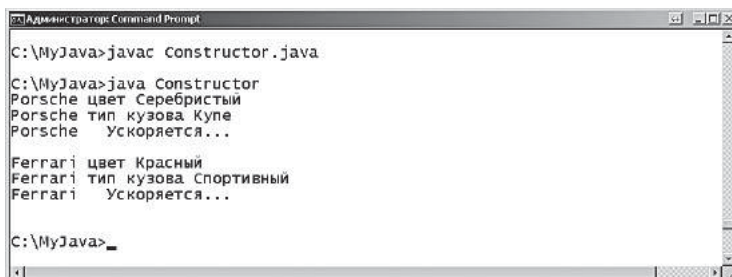
9. Создайте еще один экземпляр, присвоив и вызвав значения.

```

Car Ferrari = new Car() ;
Ferrari.setCar( "Ferrari" , "Красный" , "Спортивный" ) ;
Ferrari.getCar() ;

```

10. Сохраните программу под именем *Constructor.java*, затем скомпилируйте и запустите.



```

Администратор: Command Prompt
C:\MyJava>javac Constructor.java
C:\MyJava>java Constructor
Porsche цвет Серебристый
Porsche тип кузова Купе
Porsche Ускоряется...

Ferrari цвет Красный
Ferrari тип кузова Спортивный
Ferrari Ускоряется...

C:\MyJava>

```

Заключение

- Разбиение программ на многочисленные методы, которые можно вызывать при необходимости, увеличивает гибкость и упрощает отслеживание ошибок.
- Перегруженные методы имеют то же имя, но принимают другие аргументы.
- Переменные, объявленные внутри метода, имеют локальную область видимости, а переменные класса — глобальную область видимости в пределах этого класса.
- Ключевое слово **static** используется для объявления методов и переменных класса, которые имеют глобальную область видимости в пределах этого класса.
- Ключевое слово **public** явно разрешает доступ из любого класса.
- Объявление класса может содержать ключевое слово **extends**, в котором указывается суперкласс, откуда объявляемый класс будет наследовать свойства.
- Для того чтобы явно обращаться к определенному методу класса или к его переменной, можно использовать имя класса и точечную запись.
- Объекты реального мира имеют атрибуты и действия, которые можно представить в программах с помощью переменных и методов.
- Объекты в Java создаются как шаблон класса, из которого можно сделать копии (экземпляры).
- У каждого класса существует метод **constructor**, который можно запустить с помощью ключевого слова **new** для создания экземпляра этого класса.
- Экземпляры наследуют атрибуты и методы класса, из которого они произведены.
- Инкапсуляция защищает переменные экземпляра и методы экземпляра от доступа из внешних классов.
- Ключевое слово **private** запрещает доступ снаружи класса.
- Метод конструктора можно вызывать для инициализации атрибутов этого объекта.

7

Импортирование функций

В данной главе демонстрируется, как из специализированных Java-классов импортировать в программы дополнительную функциональность.

- Работа с файлами
- Чтение консольного ввода
- Чтение файлов
- Запись файлов
- Сортировка элементов массива
- Создание списочных массивов
- Работа с датой
- Форматирование чисел
- Заключение

Работа с файлами

Java содержит пакет с именем `java.io`, предназначенный для работы с процедурами ввода и вывода в файл. Такой пакет можно сделать доступным, если в самое начало файла `.java` включить оператор `import`. Для этого используется шаблон `*`, означающий включение всех классов пакета. Например, `import java.io.* ;`.

В пакете `java.io` содержится класс с именем `File`, который может быть использован для доступа к файлам или целым каталогам. Вначале нужно создать объект `File`, используя ключевое слово `new` и указывая имя файла или имя каталога в качестве аргумента для конструктора. Например, синтаксис, создающий объект `File` с именем `info`, который будет представлять файл в локальной системе под именем `info.dat`, выглядит следующим образом:

```
File info = new File( "info.dat" ) ;
```

Данный файл будет расположен в том же каталоге, что и программа, но аргумент может содержать полный путь к файлу, расположенному в любом месте. Обратите внимание, что при создании объекта `File` в действительности файл не создается, а происходит лишь представление конкретного элемента файловой системы.

Как только объект `File`, представляющий конкретный файл системы, создан, можно вызывать его методы для работы с файлом. Наиболее используемые и полезные методы объекта `File` с кратким их описанием перечислены в таблице ниже.

На заметку



Имя файла, указанного в качестве аргумента для конструктора, должно быть заключено в кавычки.

Метод	Возвращает
<code>exists()</code>	<code>true</code> , если файл существует, <code>false</code> — в противном случае
<code>getName()</code>	имя файла в виде строки
<code>length()</code>	размер файла в байтах (тип <code>long</code>)
<code>createNewFile()</code>	<code>true</code> , если удалось создать новый файл
<code>delete()</code>	<code>true</code> в случае успешного удаления файла
<code>renameTo(File)</code>	<code>true</code> в случае успешного переименования файла
<code>list()</code>	массив имен файлов или каталогов (строкового типа)

1. Создайте новую программу, которая импортирует функции всех классов пакета `java.io`.

```
import java.io.* ;
```

2. Добавьте класс с именем `ListFiles`, содержащий стандартный метод `main`.

```
class ListFiles
```

```
{  
  
    public static void main( String[] args ) { }  
  
}
```

3. Внутри фигурных скобок главного метода добавьте оператор, создающий объект `File` для каталога с именем `data`.

```
File dir = new File( "data" ) ;
```

4. Добавьте оператор `if` для вывода имен всех файлов в данном каталоге либо вывода сообщения в случае, если каталог пустой.

```
if ( dir.exists() )  
{  
  
    String[] files = dir.list() ;  
  
    System.out.println( files.length + " файлов найдено..." ) ;  
  
    for ( int i = 0 ; i < files.length ; i++ )  
    {  
  
        System.out.println( files[i] ) ;  
  
    }  
  
}  
  
else  
{ System.out.println( "Каталог не найден." ) ;
```

5. Сохраните программу под именем `ListFiles.java` внутри каталога `data`, содержащего несколько файлов, затем скомпилируйте и запустите; вы увидите перечисленные имена файлов в качестве вывода.



ListFiles.java

```
Администратор: Командная строка  
C:\MyJava>javac ListFiles.java  
C:\MyJava>java ListFiles  
3 файлов найдено...  
albums.ods  
oscar.txt  
xml.jpg  
C:\MyJava>
```


Чтение консольного ввода

Пакет `java.io` позволяет в программах на языке Java читать данные, вводимые пользователем в командной строке. Подобно объекту `System.out`, отправляющему вывод в командную строку, объект `System.in` можно использовать для чтения из командной строки при помощи объекта `InputStreamReader`. Вводимые данные читаются побайтно, затем конвертируются в целочисленные значения, представляющие значения символов в формате Unicode.

Для того чтобы прочитать целую строку вводимого текста, метод `readLine()` объекта `BufferedReader` читает символы, декодированные при помощи объекта `InputStreamReader`. Данный метод следует вызывать внутри блока операторов `try catch`, чтобы перехватывать исключения `IOException` (исключения, вызванные вводом-выводом).

Обычно метод `readLine()` заносит введенные данные в строковую переменную для следующей обработки в программе.



ReadString.java

1. Создайте новую программу, которая импортирует функциональность всех классов. `java.io`

```
import java.io.* ;
```

2. Добавьте класс с именем `ReadString`, содержащий стандартный метод `main`.

```
class ReadString
{
    public static void main( String[] args ) { }
```

3. Внутри фигурных скобок главного метода добавьте оператор для вывода сообщения, предлагающего пользователю ввести данные.

```
System.out.print( "Введите название данной книги: " ) ;
```

4. Добавьте оператор, создающий объект `InputStreamReader`, позволяющий читать вводимые с командной строки данные.

```
InputStreamReader isr =
    new InputStreamReader( System.in ) ;
```

5. Создайте объект `BufferedReader` для чтения декодированного ввода.

```
BufferedReader buffer = new BufferedReader( isr ) ;
```

- Объявите и проинициализируйте пустую строковую переменную, в которой будет сохраняться ввод.

```
String input = "" ;
```

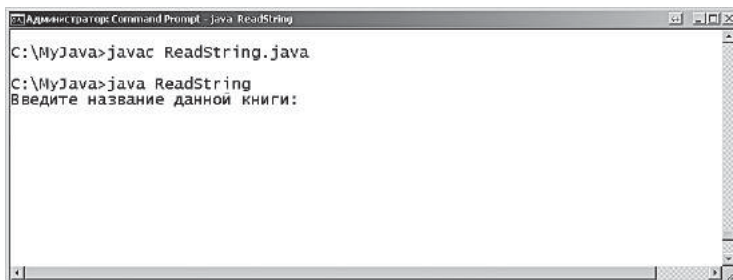
- Добавьте блок операторов **try catch**, в котором будет происходить чтение данных, вводимых из командной строки, и сохранение их в переменной.

```
try
{
    input = buffer.readLine() ;
    buffer.close() ;
}
catch ( IOException e )
{
    System.out.println( "Произошла ошибка ввода" ) ;
}
```

- Выведите сообщение, включающее сохраненное значение ввода.

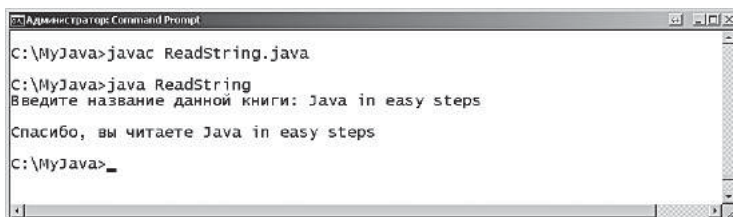
```
System.out.println( "\nСпасибо, вы читаете " + input ) ;
```

- Сохраните программу под именем *ReadString.java*, затем скомпилируйте и запустите.



```
Администратор: Command Prompt - java ReadString
C:\MyJava>javac ReadString.java
C:\MyJava>java ReadString
Введите название данной книги:
```

- Введите запрашиваемый текст, нажмите клавишу **Enter**, и вы увидите выводимое сообщение, содержащее этот текст.



```
Администратор: Command Prompt
C:\MyJava>javac ReadString.java
C:\MyJava>java ReadString
Введите название данной книги: Java in easy steps
Спасибо, вы читаете Java in easy steps
C:\MyJava>_
```

Совет



Хорошей практикой является вызов метода `close()` объекта `BufferedReader`, если он больше не нужен.

Чтение файлов

Пакет `java.io` содержит класс с именем **FileReader**, который специально предназначен для чтения текстовых файлов. Он является подклассом класса **InputStreamReader**, который используется для чтения консольного ввода и преобразует поток байтов в целые числа, представляющие символы в формате ЮНИКОД.

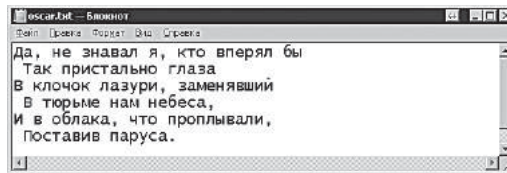
Объект **FileReader** создается при помощи ключевого слова **new**, а также имени файла, из которого нужно читать, в качестве аргумента. Аргумент может включать полный путь к файлу за пределами каталога, где расположена программа.

Для эффективного чтения текста строка за строкой используется метод **readLine()** объекта **BufferedReader**, который читает символы, декодированные объектом **FileReader**.

Данный метод следует вызывать внутри блока **try catch** для перехвата возможных исключений ввода-вывода.

Для того чтобы прочитать все строки текстового файла, в цикле вызывается метод **readLine()**. В конце файла он возвратит значение **null**, которое можно использовать для выхода из цикла.

1. Откройте простой текстовый редактор, скажем, Блокнот (Notepad), и введите несколько строк текста, например знаменитую фразу Оскара Уайльда из произведения «Баллада Редингской тюрьмы».



2. Сохраните текстовый файл под именем *oscar.txt*, затем создайте новую программу, которая импортирует функции всех классов пакета `java.io`.

```
import java.io.* ;
```

3. Добавьте класс с именем **ReadFile**, содержащий стандартный метод **main**.

```
class ReadFile
{ public static void main( String[] args ) { } }
```

4. Внутри фигурных скобок главного метода добавьте блок операторов **try catch**.

```
try {}  
catch ( IOException e )  
{  
    System.out.println( "Произошла ошибка чтения" ) ;  
}
```

5. Внутри фигурных скобок блока **try** добавьте оператор для создания объекта **FileReader**.

```
FileReader file = new FileReader( "oscar.txt" ) ;
```

6. Создайте объект **BufferedReader** для чтения файла.

```
BufferedReader buffer = new BufferedReader( file ) ;
```

7. Объявите и проинициализируйте пустую строковую переменную для хранения строки текста.

```
String line = "" ;
```

8. Добавьте цикл для чтения в переменную содержимого текстового файла, а также для вывода всех строк текста.

```
while ( ( line = buffer.readLine() ) != null )  
{ System.out.println( line ) ; }
```

9. Не забудьте закрыть объект **BufferedReader**, поскольку он больше не нужен.

```
buffer.close() ;
```

10. Сохраните программу под именем *ReadFile.java* рядом с текстовым файлом, затем скомпилируйте и запустите.

На заметку



Имя текстового файла, указанного в качестве аргумента для **FileReader**, должно быть заключено в кавычки.

```
Администратор: Command Prompt  
c:\MyJava>javac ReadFile.java  
c:\MyJava>java ReadFile  
Да, не знавал я, кто вперял бы  
Так пристально глаза  
В клочок лазури, заменявший  
В тюрне наш небеса,  
И в облака, что проплывали,  
Поставив паруса.  
c:\MyJava>
```

Запись файлов

В пакете `java.io` классы `FileReader` и `BufferedReader`, использующиеся для чтения текстовых файлов, имеют аналоги с именами `FileWriter` и `BufferedWriter`, которые можно использовать для записи текстовых файлов.

Объект `FileWriter` создается с использованием ключевого слова `new` и принимает имя файла, в который нужно производить запись, в качестве аргумента. Аргумент может включать полный путь к файлу за пределами каталога, в котором расположена программа.

Объект `BufferedWriter` создается с использованием ключевого слова `new` и принимает в качестве аргумента имя объекта `FileWriter`. После этого при помощи метода `write` объекта `BufferedWriter` можно записывать текст, в котором строки разделяются вызовом метода `newLine()`. Эти методы следует располагать внутри блока операторов `try catch` для перехвата возможных исключений ввода-вывода.

Если файл с указанным именем уже существует, то метод `write()` станет перезаписывать содержимое файла, в противном случае будет создаваться новый файл с этим именем, в который станет записываться содержимое.



WriteFile.java

1. Создайте новую программу, которая импортирует функции всех классов пакета `java.io`.

```
import java.io.* ;
```

2. Добавьте класс с именем `WriteFile`, содержащий стандартный метод `main`.

```
class WriteFile
{
    public static void main ( String[] args ) { }
```

3. Внутри фигурных скобок главного метода добавьте блок операторов `try catch`.

```
try { }
catch ( IOException e )
{
    System.out.println( "Произошла ошибка записи" ) ;
}
```

4. Внутри фигурных скобок блока **try** добавьте оператор для создания объекта **FileWriter** для текстового файла с именем *tam.txt*.

```
FileWriter file = new FileWriter( "tam.txt" ) ;
```

5. Создайте объект **BufferedWriter** для записи файла.

```
BufferedWriter buffer = new BufferedWriter( file ) ;
```

6. Добавьте операторы для записи строк текста, а также символов новой строки в текстовый файл. Например, перевод произведения «Тэм О’Шентер» Роберта Бернса.

```
buffer.write( "Дул ветер из последних сил," ) ;
```

```
    buffer.newLine() ;
```

```
buffer.write( "И град хлестал, и ливень лил," ) ;
```

```
    buffer.newLine() ;
```

```
buffer.write( "И вспышки молний тьма глотала," ) ;
```

```
    buffer.newLine() ;
```

```
buffer.write( "И небо долго грохотало..." ) ;
```

```
    buffer.newLine() ;
```

```
buffer.write( "В такую ночь, как эта ночь," ) ;
```

```
    buffer.newLine() ;
```

```
buffer.write( "Сам дьявол погулять не прочь." ) ;
```

7. Не забудьте закрыть объект **BufferedWriter**, поскольку он больше не нужен.

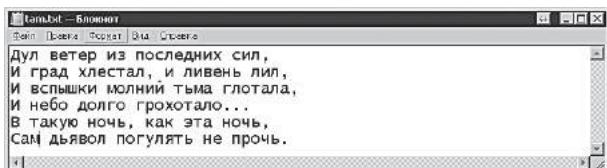
```
buffer.close() ;
```

8. Сохраните программу под именем *WriteFile.java*, затем скомпилируйте и запустите; она запишет текстовый файл в тот же каталог, где находится программа.



```

C:\МуJava>javac WriteFile.java
C:\МуJava>java WriteFile
C:\МуJava>
  
```



```

Дул ветер из последних сил,
И град хлестал, и ливень лил,
И вспышки молний тьма глотала,
И небо долго грохотало...
В такую ночь, как эта ночь,
Сам дьявол погулять не прочь.
  
```

Совет



Вместо того чтобы перезаписывать текст в файле при помощи метода **write()**, вы можете использовать метод **append()** объекта **BufferedWriter** для добавления текста в файл.

Сортировка элементов массива

В Java содержится пакет с именем `java.util`, который предлагает полезные утилиты для работы с набором данных. Данный пакет можно сделать доступным любой программе, включив в самое начало файла `.java` оператор `import`. Например, используя шаблон `*`, можно импортировать все классы данного пакета: `import java.util.* ;`.

В пакете `java.util` содержится класс с именем `Arrays`, предлагающий методы, которые можно использовать для работы с массивами. Данная функциональность может быть доступна в программах, если импортировать все классы пакета `java.util`, либо, если программа требует только одного класса, указать в операторе `import` определенное имя класса. Например, для импорта класса `Arrays` запишем `import java.util.Arrays ;`.

В классе `Arrays` содержится метод `sort()`, который может упорядочивать элементы массива либо в алфавитном порядке, либо численно.



Sort.java

1. Создайте новую программу, импортирующую функциональность класса `java.util.Arrays`.

```
import java.util.Arrays ;
```

2. Добавьте класс с именем `Sort`, содержащий стандартный метод `main`.

```
class Sort
```

```
{ public static void main( String[] args ) { }}
```

3. После метода `main` добавьте метод для отображения содержимого всех элементов переданного строкового массива.

```
public static void display( String[] elems )
```

```
{
```

```
    System.out.println( "\nСтроковый Массив:" ) ;
```

```
    for ( int i = 0 ; i < elems.length ; i++ )
```

```
        System.out.println( "Элемент "+i+ " - "+elems[i] ) ;
```

```
}
```

4. Добавьте перегруженную версию метода `display()` для отображения содержимого всех элементов переданного целочисленного массива.

Совет



Более подробно о перегрузке методов см. в главе 6.

```

public static void display( int[] elems )
{
    System.out.println( "\nЦелочисленный Массив:" ) ;

    for ( int i = 0 ; i < elems.length ; i++ )

        System.out.println( "Элемент "+i+" - "+elems[i] ) ;
}

```

5. Внутри фигурных скобок главного метода объявите и проинициализируйте строковый и целочисленный массивы.

```

String[] names = { "Майк" , "Дэйв" , "Энди" } ;

int[] nums = { 200 , 300 , 100 } ;

```

6. Выведите содержимое всех элементов каждого массива.

```

display( names ) ;

display( nums ) ;

```

7. Отсортируйте содержимое элементов обоих массивов.

```

Arrays.sort( names ) ;

Arrays.sort( nums ) ;

```

8. Выведите содержимое всех элементов каждого массива еще раз.

```

display( names ) ;

display( nums ) ;

```

9. Сохраните программу под именем *Sort.java*, затем скомпилируйте и запустите.

На заметку



В данном примере в каждом из циклов **for** выполняется только один оператор, поэтому фигурных скобок не требуется, но для ясности иногда можно их добавлять.

```

C:\MyJava>javac Sort.java
C:\MyJava>java Sort

Строковый массив:
Элемент 0 Майк
Элемент 1 Дэйв
Элемент 2 Энди

Целочисленный массив:
Элемент 0 200
Элемент 1 300
Элемент 2 100

Строковый массив:
Элемент 0 Дэйв
Элемент 1 Майк
Элемент 2 Энди

Целочисленный массив:
Элемент 0 100
Элемент 1 200
Элемент 2 300

C:\MyJava>

```


Создание списочных массивов

В пакете `java.util` содержится класс с именем `ArrayList`, который хранит данные в упорядоченном «наборе», являющемся последовательностью с изменяемым размером. Этот класс делается доступным в программе при помощи команды импорта: `import java.util.ArrayList;`. Список может содержать повторяющиеся элементы, и объект `ArrayList` имеет полезные методы, позволяющие работать с элементами массива, используя их порядковые номера (индексы). Например, вызов `get(0)` позволяет получить значение, хранящееся в первом элементе, в то время как `remove(1)` удаляет второй элемент списка.

Совет



Найти количество элементов, содержащихся в списке, вы можете, вызвав метод `size()`.

При помощи метода списка `set()` можно изменять значения элементов, указав в качестве аргументов этому методу порядковый номер и новое значение. Метод `add()` позволяет в определенное место списка добавлять элементы, указав в качестве аргументов порядковый номер и значение. При этом список расширяется, в него включаются дополнительные элементы, а порядковые номера сдвигаются.

Объект `ArrayList` создается при помощи ключевого слова `new`, но, как и другие наборы Java, при создании нужно указывать тип элемента, в котором будет содержаться список. Обычно списки содержат строковые элементы, поэтому объект `ArrayList` должен иметь суффикс `<String>`.

Такие наборы, как `ArrayList` содержат метод `forEach()`, с помощью которого можно обходить элементы списка.

Элементы списка можно также обойти при помощи «лямбда-выражения», которое можно указать методу `forEach()` в качестве аргумента. Лямбда-выражения — это простые и короткие анонимные методы, которые определяются в том же месте, где и выполняются. Они начинаются со скобок, содержащих аргументы, затем ставится последовательность символов `->`, после которой идет блок операторов:

```
( аргумент/-ы ) -> { оператор/-ы }
```

Новинка

Лямбда-выражения появились в версии Java 8 для краткой записи анонимных методов.

Тип данных аргументов может быть объявлен либо явно, либо может подразумеваться из контекста — `(String x)` может быть заменено на `(x)`. Кроме того, фигурные скобки могут быть опущены, если блок операторов лямбда-выражений содержит только один оператор.

При помощи метода `forEach()` значение текущего элемента на каждой итерации может быть передано лямбда-выражению в качестве его аргумента, а затем отображено в выводе при помощи его оператора.

1. Создайте новую программу, которая импортирует все методы класса `java.util.ArrayList`.

```
import java.util.ArrayList ;
```

2. Добавьте класс с именем **Lists**, содержащий стандартный метод **main**.

```
class Lists
```

```
{ public static void main( String[] args ) { }}
```

3. Внутри фигурных скобок главного метода добавьте оператор для создания строкового списочного массива с именем **list**.

```
ArrayList<String> list = new ArrayList<String>() ;
```

4. Далее добавьте операторы, заполняющие элементы списка строковыми значениями, а затем выведите весь список.

```
list.add( "Альфа" ) ;
```

```
list.add( "Дельта" ) ; list.add( "Чарли" ) ; System.out.  
println( "Список: " + list ) ;
```

5. Теперь идентифицируйте текущее значение, хранящееся во втором элементе, а затем замените его новым строковым значением.

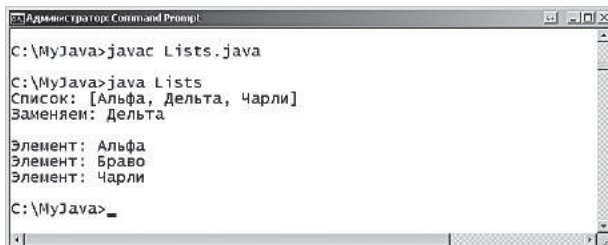
```
System.out.println( "Заменяем: " + list.get(1) + "\n" ) ;
```

```
list.set( 1, "Браво" ) ;
```

6. Наконец, сделайте обход списка и выведите строковое значение, хранящееся в каждом элементе.

```
list.forEach( ( x ) -> System.out.println( "Элемент: " + x )  
) ;
```

7. Сохраните программу под именем **Lists.java**, затем скомпилируйте и запустите.



```

C:\MyJava>javac Lists.java
C:\MyJava>java Lists
Список: [Альфа, Дельта, Чарли]
Заменяем: Дельта
Элемент: Альфа
Элемент: Браво
Элемент: Чарли
C:\MyJava>

```



Lists.java

На заметку



Так же, как и обычные массивы, элементы списочного массива нумеруются, начиная с нуля.

Совет



Графический компонент **JComboBox**, представленный в следующей главе, содержит раскрывающийся список вариантов и при его создании нужно также указывать тип данных.

Работа с датой

Новинка

Пакет `java.time` был представлен в версии Java 8, для того чтобы облегчить работу с датой и временем.



DateTime.java

Пакет `java.time` содержит класс с именем `LocalDateTime`, в котором есть полезные методы для извлечения определенных полей, описывающих конкретные метки времени. Для работы с этим классом нужно либо импортировать конкретно его командой `import java.time.LocalDateTime;`, либо импортировать все классы данного пакета, используя шаблон `import java.time.* ;`.

При помощи метода `now()` можно создать новый объект `LocalDateTime`, содержащий поля, описывающие текущее значение даты и времени. Данные поля инициализируются текущим значением локального системного времени.

Значения отдельных полей объекта `LocalDateTime` можно получить, используя соответствующие методы. Например, метод `getYear()` вызывает значение поля «Год». Точно так же значения всех полей можно и изменять, применяя соответствующие методы объекта `LocalDateTime` и указывая значение для установки. Например, новое значение года можно задать, указав его в качестве аргумента методу `withYear()`.

1. Создайте новую программу, импортирующую все методы класса `java.time.LocalDateTime`.

```
import java.time.LocalDateTime ;
```

2. Добавьте класс с именем `DateTime`, содержащий стандартный метод `main`.

```
class DateTime  
{  
    public static void main ( String [] args ) {}  
}
```

3. Внутри фигурных скобок главного метода добавьте оператор для создания объекта `LocalDateTime`, который будет содержать текущее значение времени.

```
LocalDateTime date = LocalDateTime.now() ;
```

4. Выведите текущее значение даты и времени.

```
System.out.println( "\nСейчас " + date ) ;
```

5. Увеличьте значение года и выведите измененное значение текущего времени.

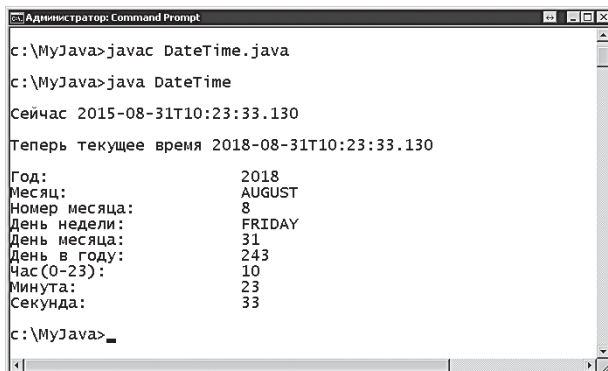
```
date = date.withYear( 2018 ) ;
```

```
System.out.println( "\nТеперь текущее время " + date ) ;
```

6. Выведите отдельные поля объекта `LocalDateTime`.

```
String fields = "\nГод:\t\t\t" + date.getYear() ;  
fields += "\nМесяц:\t\t\t" + date.getMonth() ;  
fields += "\nНомер месяца:\t\t\t" + date.getMonthValue() ;  
fields += "\nДень недели:\t\t\t" + date.getDayOfWeek() ;  
fields += "\nДень месяца:\t\t\t" + date.getDayOfMonth() ;  
fields += "\nДень в году:\t\t\t" + date.getDayOfYear() ;  
fields += "\nЧас (0-23):\t\t\t" + date.getHour() ;  
fields += "\nМинута:\t\t\t" + date.getMinute() ;  
fields += "\nСекунда:\t\t\t" + date.getSecond() ;  
  
System.out.println( fields ) ;
```

7. Сохраните программу под именем `DateTime.java`, затем скомпилируйте и запустите.



```
Администратор: Command Prompt  
c:\MyJava>javac DateTime.java  
c:\MyJava>java DateTime  
Сейчас 2015-08-31T10:23:33.130  
Теперь текущее время 2018-08-31T10:23:33.130  
Год: 2018  
Месяц: AUGUST  
Номер месяца: 8  
День недели: FRIDAY  
День месяца: 31  
День в году: 243  
Час (0-23): 10  
Минута: 23  
Секунда: 33  
c:\MyJava>
```

Совет



Использование конкатенации строк в данном примере является эффективным способом для вывода, поскольку функция `println()` вызывается только один раз, вместо того чтобы вызывать ее много раз для вывода каждого отдельного поля.

На заметку



Вместо `LocalDateTime` вы можете использовать класс `ZonedDateTime`, если вам требуется поле временной зоны.

Форматирование чисел

В Java содержится пакет с именем `java.text`, который предлагает полезные классы для форматирования чисел, в том числе значений валют. Пакет можно сделать доступным в любой программе, включив в начало файла `.java` оператор `import`. Это можно сделать, используя шаблон `*`, включающий все классы пакета: `import java.text.*`; , либо указывая определенный класс по имени.

В пакете `java.text` содержится класс с именем `NumberFormat`, который содержит методы, используемые для форматирования числовых значений при выводе: добавление разделителей, символов валют, а также знаков процента.

Новинка

Пакет `java.time.format` был введен в версии Java 8 для упрощения представления значения даты.

При создании объекта `NumberFormat` определяется тип его форматирования — метод `getNumberInstance()` для разделителей, `getCurrencyInstance()` — для символов валюты и `getPercentInstance()` — для знаков процента. Форматирование применяется указанием числового значения, которое нужно отформатировать, в качестве аргумента методу `format()` объекта `NumberFormat`.

Для форматирования объектов, содержащих дату и время, используется пакет `java.time.format`, в котором имеется класс `DateTimeFormat`. Объект `DateTimeFormatter` содержит шаблон форматирования, который указывается в качестве строкового аргумента методу `ofPattern()`. Шаблон форматирования включает буквы, определенные в документации, а также символы для разделителей, которые выбираете вы сами. Например, шаблон `M/d/y` определяет месяц, день и год, разделенные слешем. Формат применяется указанием шаблона форматирования в качестве аргумента методу `format()` объекта `java.time`.



Formats.java

1. Создайте новую программу, импортирующую функциональность всех методов класса `NumberFormat` пакета `java.text`, а также всех методов класса `DateTimeFormatter` пакета `java.time.format`.

```
import java.text.NumberFormat ;  
import java.time.format.DateTimeFormatter ;
```

2. Добавьте класс с именем `Formats`, содержащий стандартный метод `main`.

```
class Formats  
{  
    public static void main ( String [] args )  
    {  
    }  
}
```

3. Внутри фигурных скобок главного метода добавьте операторы для вывода числа с разделителями групп.

```
NumberFormat nf = NumberFormat.getNumberInstance() ;
System.out.println( "\nЧисло: " + nf.format(123456789) ) ;
```

4. Добавьте выражения для вывода числа с символом валюты.

```
NumberFormat cf = NumberFormat.getCurrencyInstance() ;
System.out.println( "\nВалюта: " + cf.format(1234.50f) ) ;
```

5. Добавьте выражения для вывода числа со знаком процента.

```
NumberFormat pf = NumberFormat.getPercentInstance()
System.out.println( "\nПроцент: " + pf.format( 0.75f ) ) ;
```

6. Добавьте оператор, создающий объект `LocalDateTime`, содержащий значение текущего времени.

```
java.time.LocalDateTime now =
    java.time.LocalDateTime.now() ;
```

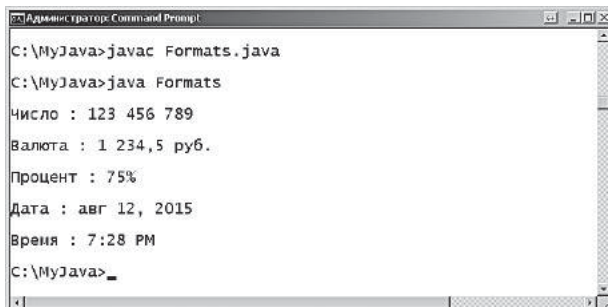
7. Добавьте операторы для вывода форматированного значения даты.

```
DateTimeFormatter df =
    DateTimeFormatter.ofPattern( "MMM d, yyy" ) ;
System.out.println( "\nДата: " + now.format( df ) ) ;
```

8. Добавьте операторы для вывода форматированного значения времени.

```
DateTimeFormatter tf =
    DateTimeFormatter.ofPattern( "h:m a" ) ;
System.out.println( "\nВремя: " + now.format( tf ) ) ;
```

9. Сохраните программу под именем *Formats.java*, затем скомпилируйте и запустите.



```

C:\МуJava>javac Formats.java
C:\МуJava>java Formats
Число : 123 456 789
Валюта : 1 234,5 руб.
Процент : 75%
Дата : авг 12, 2015
Время : 7:28 PM
C:\МуJava>

```

Совет



Операторы могут обращаться к классу, который не был импортирован, если указать полный путь к пакету, как показано здесь, в операторе, создающем объект `LocalDateTime`.

Внимание



Буквы, используемые в шаблонах, чувствительны к регистру — для подробного описания используемых шаблонов обратитесь к документации.

Заключение

- Для того чтобы в программе использовать функциональность других классов, применяется оператор **import**, который ставится в начале программы.
- Оператор **import** может импортировать как отдельные классы, определенные по имени, так и классы пакета при помощи символа шаблона *****.
- Для обработки процедур ввода и вывода существует пакет **java.io**.
- Объект **File** можно использовать для доступа к файлам и каталогам.
- Объект **InputStreamReader** декодирует введенные байты в символы, а объект **BufferedReader** читает эти декодированные символы.
- Для декодирования байтов текстового файла в символы используется объект **FileReader**.
- Объекты **FileWriter** и **BufferedWriter** можно использовать для создания и изменения текстового файлов.
- Пакет **java.util** содержит утилиты для обработки наборов данных, подобных массивам класса **Arrays**.
- В пакете **java.util** содержится также класс **ArrayList**, у которого есть методы для обработки элементов последовательностей.
- Объект **ArrayList** — это набор, который должен указывать тип элемента, содержащегося в списке, например **<String>**.
- Лямбда-выражение — это анонимный метод, который определяется сразу же на месте выполнения.
- Пакет **java.time** содержит класс **LocalDateTime**, представляющий поля для компонентов даты и времени.
- Пакет **java.text** содержит класс **NumberFormat**, который используется для форматирования чисел и значений валют.
- Пакет **java.time.format** содержит класс **DateTimeFormatter**, который определяет шаблоны для форматирования значений даты и времени.

8

Построение интерфейсов

В этой главе демонстрируется, как использовать компоненты Java Swing для создания графических программных интерфейсов.

- **Создание окна**
- **Добавление кнопок**
- **Добавление меток**
- **Добавление текстовых полей**
- **Добавление элементов выбора**
- **Добавление переключателей**
- **Изменение внешнего вида интерфейса**
- **Размещение компонентов**
- **Заключение**

Создание окна

На заметку



Буква **x** в названии **javax.swing** подразумевает **JAVAXtra** (дополнение к Java).

В программах на Java можно создавать графический пользовательский интерфейс (GUI), используя графический компонент библиотеки Java под названием Swing. Пакет **javax.swing** содержит классы, при помощи которых можно создать разнообразные компоненты, используя стиль операционной системы. Чтобы включить эту возможность, нужно добавить оператор импорта в программу: **import javax.swing.*;**

Для создания графического пользовательского интерфейса нужно организовать класс, к которому можно добавлять компоненты, используемые при построении интерфейса. Проще всего это сделать, объявив подкласс класса **JFrame** при помощи ключевого слова **extends** — тем самым наследуя атрибуты и методы, которые позволят пользователю работать с окном: перемещать, изменять размер и закрывать.

Конструктор класса должен включать операторы, удовлетворяющие следующим минимальным требованиям:

- заголовок окна — указывается в качестве строкового аргумента наследуемому методу **super()** класса **JFrame**;
- размер окна — указывается значение ширины и высоты в пикселях как аргументы для метода **setSize()**;
- действие при закрытии пользователем окна — определяется константой, являющейся аргументом для метода **setDefaultCloseOperation()**;
- параметр отображения окна — указывается в виде аргумента графического типа для метода **setVisible()**.

Дополнительно в конструкторе можно добавлять компонент окна — так называемый контейнер **JPanel**, к которому будут добавляться более мелкие компоненты при использовании наследуемого метода **add()** класса **JFrame**.

Контейнер **JPanel** (или панель) по умолчанию использует менеджер шаблона **FlowLayout**, который располагает компоненты в колонках слева направо, привязываясь к правой границе окна.

В шагах, представленных ниже, описывается создание базового окна, содержащего контейнер **JPanel** с менеджером шаблона **FlowLayout**. Это окно будет использоваться в дальнейших примерах книги для демонстрации добавления различных компонентов к контейнеру **JPanel**.

Совет



Менеджеры шаблонов описаны более подробно в конце этой главы.

1. Создайте новую программу, в которую импортируются все компоненты Swing.

```
import javax.swing.* ;
```

2. Создайте в классе **JFrame** подкласс с именем **Window**, содержащий стандартный метод **main**.

```
class Window extends JFrame  
{  
    public static void main ( String[] args ) { }  
}
```

3. Перед методом **main** создайте контейнер **JPanel**.

```
JPanel pnl = new JPanel( ) ;
```

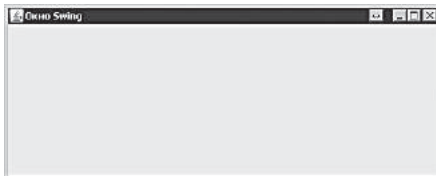
4. Затем добавьте конструктор с указанием параметров окна и добавлением объекта **JPanel** к классу **JFrame**.

```
public Window()  
{  
    super( "Окно Swing" ) ;  
    setSize( 500 , 200 ) ;  
    setDefaultCloseOperation( EXIT_ON_CLOSE ) ;  
    add( pnl ) ;  
    setVisible( true ) ;  
}
```

5. Создайте экземпляр класса **Window** с добавлением следующей строки в главный метод.

```
Window gui = new Window() ;
```

6. Сохраните программу под именем **Window.java**, затем скомпилируйте и запустите — вы увидите, что появится базовое окно.



Window.java

На заметку



В данном примере значение **EXIT_ON_CLOSE** — это константа, являющаяся членом класса **JFrame**. Она производит выход из программы при закрытии окна.

Совет



Обратите внимание, как используется метод **add()** для добавления объекта **JPanel** на окно **JFrame**.

Добавление кнопок

На графический интерфейс можно добавлять кнопки, которые создаются при помощи класса **JButton**. Этими кнопками пользователь будет взаимодействовать с программой, нажимая на них для выполнения определенного действия.

Объект **JButton** создается при помощи ключевого слова **new**, и его конструктор принимает строковый аргумент, в котором указывается текст, отображаемый на кнопке.

Совет



Подробнее о том, как создавать методы-обработчики событий, реагирующие на действия пользователя, например нажатие кнопки мыши, будут рассматриваться в следующей главе.



Buttons.java

На кнопках можно также располагать изображения. Для этого нужно сначала создать объект изображения — **ImageIcon**. Конструктору этого объекта в качестве аргумента указывается имя файла изображения. Обычно файл изображения размещается рядом с программой, но это необязательно, поэтому аргумент может включать путь к изображению за пределами каталога с программой.

Чтобы отобразить изображение на кнопке, нужно указать имя объекта **ImageIcon** в качестве аргумента конструктору **JButton** или указать два аргумента — строку и имя объекта **ImageIcon**. В этом случае на кнопке будет отображаться и текст, и изображение.

1. Отредактируйте копию файла *Window.java*, описанного ранее, заменив имя класса **Window** в объявлении, конструкторе и экземпляре на **Buttons**.
2. Перед конструктором **Buttons()** создайте два объекта изображений **ImageIcon**.

```
ImageIcon tick = new ImageIcon( "tick.png" ) ;
```

```
ImageIcon cross = new ImageIcon( "cross.png" ) ;
```

3. Затем создайте три объекта **JButton**: в первом из них будет отображаться текст, во втором — изображение, а в третьем — и текст, и изображение.

```
JButton btn = new JButton( "Нажми меня" ) ;
```

```
JButton tickBtn = new JButton( tick ) ;
```

```
JButton crossBtn = new JButton( "СТОП" , cross ) ;
```

4. Внутри конструктора **Buttons()** добавьте три оператора для добавления компонентов **JButton** на контейнер **JPanel**.

```
pnl.add( btn ) ;
```

```
pnl.add( tickBtn ) ;
```

```
pnl.add( crossBtn ) ;
```

На заметку



Для добавления компонентов на панель объект **JPanel** содержит метод **add()**.

5. Сохраните программу под именем *Buttons.java*, затем скомпилируйте и запустите — вы увидите, как в окне появятся три кнопки.



Кнопки реагируют графически при нажатии на них, но не выполняют никаких действий до тех пор, пока в программе не будет задан метод-обработчик для реагирования на каждое событие нажатия.

В случае если вы намерены распространять программу в виде одиночного Java-архива (JAR), то перед созданием объектов **ImageIcon** соответствующие графические ресурсы должны быть загружены при помощи объекта **ClassLoader**.

Имя файла или его путь указывается методу **getResource()** объекта **ClassLoader**, и в результате будет возвращаться адрес URL, который можно будет использовать в качестве аргумента для конструктора **ImageIcon**. Для этого в пакете **java.net** существует полезный класс **URL**, которому можно все это присвоить.

6. Перед конструктором **Buttons()** создайте объект **ClassLoader**.

```
ClassLoader ldr = this.getClass().getClassLoader() ;
```

7. Загрузите адреса ресурсов (файлов изображений).

```
java.net.URL tickURL = ldr.getResource( "tick.png" ) ;
```

```
java.net.URL crossURL = ldr.getResource( "cross.png" ) ;
```

8. Отредактируйте конструкторы **ImageIcon()**, описанные на шаге 2, чтобы использовать адреса URL.

```
ImageIcon tick = new ImageIcon( tickURL ) ;
```

```
ImageIcon cross = new ImageIcon( crossURL ) ;
```

9. Сохраните изменения, затем скомпилируйте и перезапустите программу на выполнение. Результат будет такой же, но теперь программу можно будет распространять в файле JAR.

Совет



Подробнее о том, как создавать Java-архивы (JAR) см. в главе 10.

Совет



Обратите внимание, что для ссылки на объект класса используется метод **getClass()** и ключевое слово **this**.

Добавление меток

На графический интерфейс можно добавлять элемент «метка» при помощи класса **JLabel**. Данный элемент используется для отображения неизменяемого пользователем текста или изображения или и того и другого.

Объект **JLabel** создается при помощи ключевого слова **new**, и его конструктор принимает строковый аргумент, указывающий текст для отображения на метке, или имя объекта **ImageIcon**, представляющего изображение для отображения. Он может также принимать три аргумента, определяющие текст, изображение и величину горизонтальной привязки в виде константы объекта **JLabel**. Например, **JLabel("text", img, JLabel.CENTER)** осуществляет привязку по центру.

В случае если объект **JLabel** содержит текст и изображение, то относительное положение текста можно определять при помощи указания константы **JLabel** в качестве аргумента методам **setVerticalPosition()** и **setHorizontalPosition()** объекта **JLabel**.

Существует дополнительный элемент **ToolTip** — всплывающая подсказка, которая появляется при наведении курсора мыши на объект. Ее можно создавать при помощи метода **setToolTipText()**, указав ему в качестве аргумента строку текста с подсказкой.



Labels.java

1. Отредактируйте копию файла *Window.java*, заменив имя класса **Window** в объявлении, конструкторе и экземпляре на **Labels**.
2. Перед конструктором **Labels()** создайте объект **ImageIcon** для изображения.

```
ImageIcon duke = new ImageIcon( "duke.png" );
```

3. Затем создайте три объекта **JLabel**: первый для вывода изображения, второй для вывода текста, а третий — и для того, и для другого.

```
JLabel lbl1 = new JLabel( duke );
```

```
JLabel lbl2 = new JLabel( "Дюк — талисман технологии Java." );
```

```
JLabel lbl3 = new JLabel( "Дюк" , duke , JLabel.CENTER );
```

4. Внутри конструктора **Labels()** добавьте следующий оператор, создающий подсказку для первой метки.

```
lbl1.setToolTipText( "Дюк — талисман Java" );
```

5. После третьей метки напишите два оператора для привязки текста по центру и внизу.

```
lbl3.setHorizontalTextPosition( JLabel.CENTER ) ;  
lbl3.setVerticalTextPosition( JLabel.BOTTOM ) ;
```

6. Теперь допишите три оператора для добавления компонентов **JLabel** на контейнер **JPanel**.

```
pnl.add( lbl1 ) ;  
pnl.add( lbl2 ) ;  
pnl.add( lbl3 ) ;
```

7. Сохраните программу под именем *Labels.java*, затем скомпилируйте и запустите, поместив указатель мыши над первой меткой.



В случае если вы намерены распространять программу в одиночном Java-архиве, перед созданием объекта **ImageIcon** графические ресурсы должны быть загружены при помощи объекта **ClassLoader**.

Методу **getResource()** объекта **ClassLoader** нужно указать имя файла или путь к нему, а метод вернет адрес URL, который можно использовать в качестве аргумента для конструктора **ImageIcon**.

8. Перед конструктором **Labels()** создайте объект **ClassLoader**.

```
ClassLoader ldr = this.getClass().getClassLoader() ;
```

9. Отредактируйте конструктор **ImageIcon()**, представленный на шаге 2, для того чтобы загрузить адрес URL файла ресурса, используя объект **ClassLoader**.

```
ImageIcon duke =  
    new ImageIcon( ldr.getResource( "duke.png" ) ) ;
```

10. Сохраните изменения, затем заново скомпилируйте и запустите программу. Теперь ее можно распространять в файле JAR.

На заметку



Константы для привязки к компонентам **JLabel** включают **LEFT**, **CENTER**, **RIGHT**, **TOP** и **BOTTOM**.

Совет



Подробнее о создании Java-архива (JAR) см. в главе 10.

Добавление текстовых полей

Совет



В тех случаях, когда вам нужно скрыть вводимые пользователем символы, вместо класса `JTextField` используйте класс `JPasswordField`.

В библиотеке Swing существует класс `JTextField`, который создает компонент графического интерфейса, представляющий собой однострочное текстовое поле. Данный компонент позволяет отобразить на экране редактируемый текст, с помощью которого пользователь может взаимодействовать с программой.

Объект `JTextField` создается при помощи ключевого слова `new`, и его конструктор может принимать строковый аргумент, задающий текст по умолчанию, который будет отображаться в этом поле. В этом случае размер компонента будет подогнан в соответствии с длиной указанной строки. В качестве альтернативного варианта аргументом может быть числовое значение, определяющее размер текстового поля. Конструктор может также принимать и оба аргумента одновременно — для текста по умолчанию и для размера текстового поля.

При помощи класса `JTextArea` можно создавать многострочные текстовые поля. Конструктор данного класса принимает два числовых аргумента, определяющих число строк и ширину поля. Альтернативным вариантом является указание трех аргументов, задающих текст по умолчанию, а также число строк и ширину. При помощи методов `setLineWrap()` и `setWrapStyleWord()` объекта `JTextArea` можно управлять переносом слов, подгоняя тем самым ширину вводимого текста под размер поля.

Если текст, введенный в компонент `JTextArea`, превышает его первоначальный размер, компонент будет растягиваться. Можно задать компоненту фиксированный размер с возможностью скроллинга. Для этого его нужно поместить в контейнер `JScrollPane`. Это можно сделать при помощи ключевого слова `new` с использованием имени объекта `JTextArea` как аргумента.

По умолчанию полоса прокрутки (`Scrollbar`) будет появляться, когда поле содержит текст, превышающий его размер. Но можно сделать так, чтобы полоса прокрутки включалась постоянно. Для этого нужно указать константу контейнера `JScrollPane` в качестве аргумента методам `setVerticalScrollBarPolicy()` или `setHorizontalScrollBarPolicy()`. Например, для того, чтобы всегда отображать вертикальную полосу прокрутки, используйте константу `JScrollPane.VERTICAL_SCROLLBAR_ALWAYS` в качестве аргумента.

1. Отредактируйте копию файла *Window.java*, заменив имя класса **Window** в объявлении, конструкторе и экземпляре на **TextFields**.
2. Перед конструктором **TextFields()** создайте два объекта **TextField**.



TextFields.java

```
TextField txt1 = new TextField( 38 ) ;

TextField txt2 = new TextField( "Текст по умолчанию" , 38 ) ;
```

3. Создайте объект **TextArea** высотой в пять строк.
4. Добавьте объект **ScrollPane**, который будет содержать объект **TextArea**, созданный на предыдущем шаге.

```
ScrollPane pane = new JScrollPane( txtArea ) ;
```

5. В методе конструктора **TextFields()** добавьте операторы для включения опции переноса слов.

```
txtArea.setLineWrap( true ) ;

txtArea.setWrapStyleWord( true ) ;
```

6. Добавьте оператор для постоянного отображения вертикальной полосы прокрутки для объекта **TextArea**.

```
pane.setVerticalScrollBarPolicy

( JScrollPane.VERTICAL_SCROLLBAR_ALWAYS ) ;
```

7. Допишите два оператора для добавления компонентов **TextField** на контейнер **JPanel**.

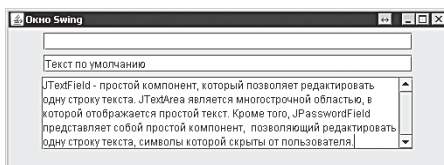
```
pnl.add( txt1 ) ;

pnl.add( txt2 ) ;
```

8. Вставьте еще один оператор для добавления контейнера **ScrollPane** (содержащего поле **TextArea**) в контейнер **JPanel**.

```
pnl.add( pane ) ;
```

9. Сохраните программу под именем *TextFields.java*, затем скомпилируйте и запустите, вводя текст в текстовые поля.



Внимание

Компонент **JTextArea** не имеет возможности прокрутки до тех пор, пока его не поместить внутрь компонента **ScrollPane**.

Добавление элементов выбора

В библиотеке Swing существует класс **JCheckBox**, при помощи которого создается компонент «флажок» (**CheckBox**). Его можно добавлять в графический интерфейс, и этот компонент дает возможность выбирать либо отменять выбор отдельного элемента. Объект **JCheckBox** создается при помощи ключевого слова **new**, и его конструктор принимает строковый аргумент, определяющий отображаемый рядом с флажком текст. Можно также указывать и второй аргумент, имеющий логическое значение **true** — он делает флажок выбранным по умолчанию.

Класс **JComboBox** предлагает возможность создания выпадающего списка элементов, из которого пользователь может выбрать единственный. Данный объект также создается при помощи ключевого слова **new**, а его конструктор принимает имя строкового массива в качестве аргумента. Каждый элемент этого массива соответствует элементу выбора выпадающего списка. Аналогичный список предлагается создать при помощи класса **JList**, который создает список фиксированного размера, и из него пользователь может выбирать один или более элементов. Объект **JList** создается при помощи ключевого слова **new**, а его конструктор принимает в качестве аргумента массив. Объекты **JList** и **JComboBox** являются наборами, поэтому они в своем объявлении должны содержать суффикс **<String>**, определяющий общий тип данных, который может содержать этот набор.



Items.java

1. Отредактируйте копию файла *Window.java*, заменив имя класса **Window** в объявлении, конструкторе и экземпляре на **Items**.
2. Перед конструктором **Items()** создайте строковый массив с элементами, которые будет выбирать пользователь.

```
String[] toppings =  
{ "Пеперони" , "Грибная" , "С ветчиной" , "Томатная" } ;
```

3. Теперь создайте четыре объекта **JCheckBox**, каждый из которых будет представлять элемент массива для выбора. Для одного из них установите значение выбора по умолчанию.

```
JCheckBox chk1 = new JCheckBox( toppings[0] ) ;  
JCheckBox chk2 = new JCheckBox( toppings[1] , true ) ;  
JCheckBox chk3 = new JCheckBox( toppings[2] ) ;  
JCheckBox chk4 = new JCheckBox( toppings[3] ) ;
```

4. Добавьте строковый массив с элементами для выбора.

```
String[] styles =
    { "В глубокой форме" , "Для гурманов" , "Тонкая" } ;
```

5. Создайте объект `JComboBox` для отображения элементов второго массива, предлагающихся к выбору.

```
JComboBox<String> box1 =
    new JComboBox<String>( styles ) ;
```

6. Добавьте объект `JList` для отображения элементов первого массива, предлагающихся к выбору.

```
JList<String> lst1 = new JList<String>( toppings ) ;
```

7. В методе конструктора `Items()` добавьте операторы, помещающие каждый из компонентов `JCheckBox` в контейнер `JPanel`.

```
pnl.add( chk1 ) ;
pnl.add( chk2 ) ;
pnl.add( chk3 ) ;
pnl.add( chk4 ) ;
```

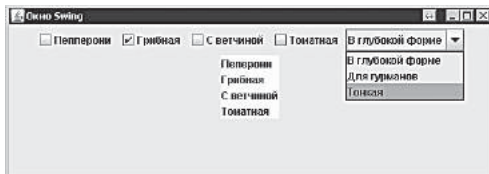
8. Добавьте операторы, которые устанавливают выбор элемента по умолчанию, а затем помещают компонент `JComboBox` в контейнер `JPanel`.

```
box1.setSelectedIndex( 0 ) ;
pnl.add( box1 ) ;
```

9. Теперь добавьте оператор для помещения компонента `JList` на контейнер `JPanel`.

```
pnl.add( lst1 ) ;
```

10. Сохраните программу под именем `Items.java`, затем скомпилируйте и запустите, выбирая элементы из списков.



На заметку



Если в компоненте `JComboBox` может быть выбран только один элемент, то в `JList` — несколько.

Совет



Подробнее о том, как создавать методы для обработки событий, реагирующие на действия пользователя, такие как выбор элементов списка, будет описано в следующей главе.

Добавление переключателей

В библиотеке Swing существует класс **JRadioButton**, который используется для создания такого компонента графического интерфейса, как «переключатель». Он может быть использован для выбора пользователем одного элемента из группы таких переключателей.

Объект **JRadioButton** создается при помощи ключевого слова **new**, а его конструктор принимает строковый аргумент, определяющий текст, отображаемый рядом с переключателем. Конструктор может также принимать и второй аргумент логического типа, имеющий значение **true**, который определяет, что соответствующий переключатель выбран по умолчанию.

Переключатели логически объединяются в группу при помощи объекта **ButtonGroup**, и из этой группы может быть выбран одновременно только один из них. Переключатель добавляется в объект **ButtonGroup** при помощи его метода **add()** указанием имени данного переключателя в качестве аргумента этому методу.



Radios.java

1. Отредактируйте копию файла **Window.java**, заменив имя класса **Window** в объявлении, конструкторе и экземпляре на **Radios**.
2. Перед конструктором **Radios()** создайте три объекта **JRadioButton**, один из которых сделайте выбранным по умолчанию.

```
JRadioButton rad1 = new JRadioButton( "Красное" , true ) ;
JRadioButton rad2 = new JRadioButton( "Розовое" ) ;
JRadioButton rad3 = new JRadioButton( "Белое" ) ;
```

3. Затем создайте объект **ButtonGroup**, при помощи которого сгруппируйте переключатели.

```
ButtonGroup wines = new ButtonGroup() ;
```

4. В методе конструктора **Radios()** вставьте операторы для добавления компонентов **JRadioButton** в группу **JButtonGroup**.

```
wines.add( rad1 ) ;
wines.add( rad2 ) ;
wines.add( rad3 ) ;
```

На заметку



Кнопки группируются в объект **ButtonGroup** не физически, а логически.

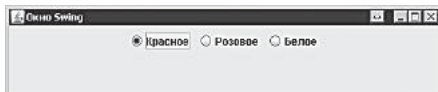
5. Вставьте операторы для добавления компонентов `JRadioButton` в контейнер `JPanel`.

```
pn1.add( rad1 ) ;
```

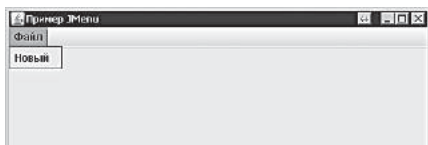
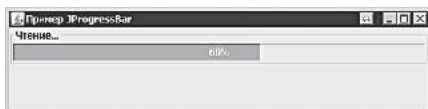
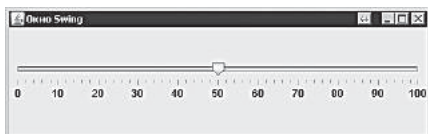
```
pn1.add( rad2 ) ;
```

```
pn1.add( rad3 ) ;
```

6. Сохраните программу под именем *Radios.java*, затем скомпилируйте и запустите, выбрав любую из кнопок.



Примеры выше демонстрировали наиболее общие компоненты библиотеки Swing — `JButton`, `JLabel`, `JTextField`, `JCheckBox`, `JComboBox`, `JList` и `JRadioButton`. В пакете `javax.swing` содержатся и другие, более специфичные компоненты, чье подробное описание можно найти в документации по Java. Например, компоненты `JSlider`, `JProgressBar` и `JMenuBar`, представленные на рисунках ниже.



Совет



Подробнее о создании методов обработки событий, реагирующих на действия пользователя — в следующей главе.

Совет



Попробуйте, используя документацию, добавить компонент `JSlider` в программу *Radios*. Как использовать документацию — см. главу 5.

Изменение внешнего вида интерфейса

Пакет `java.awt` (Abstract Window Toolkit) содержит классы для «рисования», которые вы можете использовать для «украшения» внешнего вида компонентов интерфейса. Этот пакет можно сделать доступным в программе, включив начальный оператор `import java.awt.*` ;.

Внутри пакета `java.awt` содержится класс `Color`, в котором имеются константы, представляющие несколько базовых цветов, такие, как, например, `Color.RED`. При помощи ключевого слова `new` можно создавать экземпляры класса `Color`, которые будут представлять пользовательские цвета. Конструктор может принимать три целочисленных аргумента, значения которых лежат в диапазоне от 0 до 255. Эти аргументы будут представлять компоненты RGB (красный, зеленый, синий), формирующие пользовательский цвет.

Любой компонент интерфейса содержит методы `setBackground()` и `setForeground()`, которые в качестве своего аргумента принимают объект `Color`, чтобы раскрасить компонент указанным цветом.

Обратите внимание, что фон заднего плана (background) компонентов `JLabel` по умолчанию всегда прозрачен, поэтому, перед тем как изменять цвета компонентов, нужно установить значение непрозрачности в `true` при помощи метода `setOpaque()`.

В пакете `java.awt` существует также класс `Font`, который можно использовать для изменения шрифта отображаемого текста. Конструктор объекта `Font` может принимать три аргумента, которые определяют имя, стиль и размер шрифта:

- имя должно быть одним из трех платформенезависимых имен: `Serif`, `SansSerif` или `Monospaced`;
- в качестве стиля должна выступать одна из этих трех констант: `Font.PLAIN`, `Font.BOLD` или `Font.ITALIC`;
- размер должен быть целым числом, указывающим количество точек.

Для того чтобы изменить цвет шрифта любого из компонентов интерфейса, существует метод `setFont()`, который принимает в качестве аргумента объект `Font`.



Custom.java

1. Отредактируйте копию файла *Window.java*, заменив имя класса **Window** в объявлении, в конструкторе и в экземпляре на **Custom**.
2. Добавьте оператор в самое начало программы для импортирования функциональности всех классов пакета `java.awt`.

```
import java.awt.* ;
```

3. Перед конструктором `Custom()` создайте три объекта `JLabel`.

```
JLabel lbl1 = new JLabel( "Пользовательский задний фон" ) ;  
JLabel lbl2 = new JLabel( "Пользовательский передний фон" ) ;  
JLabel lbl3 = new JLabel( "Пользовательский шрифт" ) ;
```

4. Создайте объект `Color`.

```
Color customColor = new Color( 255 , 0 , 0 ) ;
```

5. Создайте объект `Font`.

```
Font customFont = new Font( "Serif" , Font.PLAIN , 32 ) ;
```

6. В конструкторе `Custom()` добавьте операторы для изменения цвета заднего фона объекта `JLabel`, используя константу `Color`.

```
lbl1.setOpaque( true ) ;  
lbl1.setBackground( Color.YELLOW ) ;
```

7. Добавьте оператор для изменения цвета переднего плана объекта `JLabel`, используя пользовательский объект `Color`.

```
lbl2.setForeground( customColor ) ;
```

8. Добавьте оператор для изменения текста на компоненте `JLabel` с использованием пользовательского шрифта.

```
lbl3.setFont( customFont ) ;
```

9. Добавьте все метки на контейнер `JPanel`.

```
pnl.add( lbl1 ) ; pnl.add( lbl2 ) ; pnl.add( lbl3 ) ;
```

10. Сохраните программу под именем *Custom.java*, затем скомпилируйте и запустите для просмотра результата изменения компонентов.



Совет



В данном случае пользовательский цвет эквивалентен значению `Color.RED`, поскольку здесь указывается максимальное значение для красного, а зеленый и синий цвета полностью отсутствуют.

Размещение компонентов

В пакете `java.awt` содержится набор классов для менеджера компоновки, которые можно использовать для размещения компонентов в контейнере различными способами.

Менеджер компоновки создается с использованием ключевого слова **new** и впоследствии может быть использован в качестве аргумента для конструктора `JPanel`, который будет указывать, чтобы панель использовала именно этот шаблон. При добавлении компонентов на панель они будут размещаться в соответствии с правилами указанного менеджера шаблонов.

Менеджер шаблонов	Правила размещения компонентов
<code>BorderLayout</code>	Вверху, внизу, справа, слева и в центре (используется по умолчанию)
<code>BoxLayout</code>	В одну строку или столбец
<code>CardLayout</code>	Поочередно в указанной области
<code>FlowLayout</code>	В строке слева направо с возможностью переноса (используется по умолчанию для <code>JPanel</code>)
<code>GridBagLayout</code>	В сетке ячеек (могут их растягивать)
<code>GridLayout</code>	В таблице по строкам и столбцам
<code>GroupLayout</code>	Горизонтально и вертикально
<code>SpringLayout</code>	С привязкой к границам компонентов

На заметку



Каждый из менеджеров компоновки подробнее описывается в разделе документации `java.awt`.

Объект верхнего уровня `JFrame` содержит контейнер, который размещает компоненты, используя по умолчанию менеджер компоновки `BorderLayout`. На него можно помещать до пяти контейнеров `JPanel`, которые могут использовать свои менеджеры компоновки (по умолчанию `FlowLayout`), представленные в таблице выше. Используя разнообразие менеджеров компоновки, можно добиться различных вариантов размещения компонентов.

Контейнер content pane может быть представлен объектом `java.awt.Container`, чей метод `add()` может указывать позицию и имя помещаемого в данный контейнер компонента.

1. Отредактируйте копию файла *Window.java*, заменив объявление класса, конструктор и экземпляр **Window** на **Layout**, затем в начале программы добавьте оператор для импортирования функциональности пакета **java.awt**.

```
import java.awt.* ;
```



Layout.java

2. Перед конструктором **Layout()** создайте объект **Container**, представляющий собой контейнер **JFrame**.

```
Container contentPane = getContentPane() ;
```

3. Создайте еще один объект **JPanel**, используя менеджер компоновки **GridLayout** в виде сетки 2 x 2.

```
JPanel grid = new JPanel( new GridLayout( 2 , 2 ) ) ;
```

4. В конструкторе **Layout()** вставьте операторы, добавляющие компоненты **JButton** на оба объекта **JPanel**.

```
pnl.add( new JButton( "Да" ) ) ;
```

```
pnl.add( new JButton( "Нет" ) ) ;
```

```
pnl.add( new JButton( "Отмена" ) ) ;
```

```
grid.add( new JButton( "1" ) ) ;
```

```
grid.add( new JButton( "2" ) ) ;
```

```
grid.add( new JButton( "3" ) ) ;
```

```
grid.add( new JButton( "4" ) ) ;
```

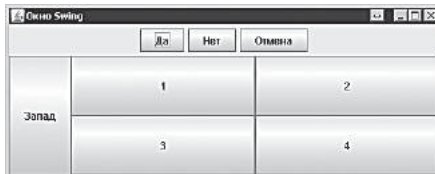
5. Теперь вставьте операторы, добавляющие обе панели и кнопку в контейнер **contentPane**.

```
contentPane.add( "North" , pnl ) ;
```

```
contentPane.add( "Center" , grid ) ;
```

```
contentPane.add( "West" , new JButton( "Запад" ) ) ;
```

6. Сохраните программу под именем *Layout.java*, затем скомпилируйте и запустите, чтобы увидеть размещение компонентов.



Внимание

Если **FlowLayout** использует размер объекта **JButton**, то другие менеджеры компоновки растягивают компоненты, чтобы заполнить пространство компоновки.

Заключение

- Для создания компонентов графического пользовательского интерфейса используются классы библиотеки Java Swing, содержащиеся в пакете `javax.swing`.
- Окно создается в виде контейнера верхнего уровня **JFrame**.
- В конструкторе **JFrame** указывается заголовок окна, размер, операция по умолчанию для закрытия и значение видимости.
- Контейнер **JPanel** отображает более мелкие компоненты, используя менеджер компоновки **FlowLayout**.
- Конструктор **JButton** может определять текст и изображения, которые должны быть отображены на компоненте «кнопка».
- Изображения, используемые в программе, представляются объектом **ImageIcon**.
- Если программы должны быть распространены в виде отдельного Java-архива (JAR), то они должны использовать объект **ClassLoader** для указания исходного файла изображения.
- Объект **JLabel** отображает неизменяемый текст и изображение.
- Текст, который можно редактировать, отображается в полях **TextField** и **TextArea**.
- Объект **JScrollPane** организует полосу прокрутки для поля **TextArea**.
- При помощи компонентов **CheckBox**, **ComboBox** и **List** можно отображать элементы для выбора.
- Компоненты **RadioButton** логически группируются при помощи компонента **ButtonGroup**.
- Класс **Color** пакета `java.awt` представляет компоненты RGB зеленого цвета.
- Для создания объектов, представляющих определенное имя, стиль и размер шрифта, используется класс **Font** пакета `java.awt`.
- При помощи класса **Container** пакета `java.awt` несколько контейнеров можно добавлять в контейнер **JFrame**.
- При создании контейнера **JPanel** можно указывать менеджер компоновки.

9

Распознавание событий

В данной главе демонстрируется, как в Java-программах создавать обработчики событий, которые реагируют на действия пользователя.

- «Прослушивание» событий
- Генерация событий
- Обработка событий кнопок
- Обработка событий элементов
- Реагирование на события клавиатуры
- Ответ на события мыши
- Вывод сообщений
- Запрос пользовательского ввода
- Проигрывание звука
- Заключение

«Прослушивание» событий

С программами, предлагающими графический интерфейс (GUI) пользователь может взаимодействовать, выполняя действия с мышью, клавиатурой либо другими устройствами ввода. Действия пользователя вызывают «события», которые заставляют программу отвечать на них. Этот процесс называется «обработкой событий».

Для того чтобы программа могла распознавать события, вызванные действиями пользователя, в нее нужно добавить интерфейс «слушателя событий» — **EventListener** из пакета **java.awt.event**. Чтобы сделать его доступным в программе, нужно добавить в первоначальный оператор строку **import java.awt.event.* ;**.

Для того чтобы включить интерфейс **EventListener** в объявление класса, нужно использовать ключевое слово **implements**. Например, объявление класса для прослушивания событий кнопки мыши может выглядеть следующим образом:

```
class Click extends JFrame implements ActionListener { }
```

Документация по Java описывает множество интерфейсов **EventListener**, которые могут «слушать» различные события, но наиболее распространенные представлены в таблице ниже вместе с кратким их описанием.

Совет



После ключевого слова **implements** можно добавлять несколько слушателей, разделяя их запятой.

Слушатель события	Описание
ActionListener	Распознает события действия, которые происходят при нажатии или освобождении кнопки
ItemListener	Распознает события элемента, которые происходят при выборе элемента или отмене выбора
KeyListener	Распознает события клавиатуры, которые происходят, когда пользователь нажимает или отпускает клавишу
MouseListener	Распознает события кнопок мыши, которые происходят, когда пользователь нажимает или отпускает кнопку мыши либо когда указатель мыши входит в область компонента или покидает ее
MouseMotionListener	Распознает события движения указателя мыши, которые происходят, когда пользователь двигает мышью

Генерация событий

Для того чтобы интерфейс **EventListener** мог распознавать события, компоненты интерфейса должны их каким-либо образом генерировать. Если в программу добавлен какой-либо **EventListener**, как описано выше, то в соответствующий компонент необходимо добавить генератор события.

Например, для того, чтобы программа реагировала на нажатие кнопки, в нее добавляется интерфейс **ActionListener** и должен быть вызван метод кнопки **addActionListener()**, в качестве аргумента которому указывается ключевое слово **this**. Таким образом генерируется событие при нажатии на данную кнопку, и это событие будет распознаваться интерфейсом **ActionListener**.

Операторы, создающие кнопку, которая генерирует события, могут выглядеть следующим образом:

```

JButton btn = new JButton( "Жми меня" ) ;
btn.addActionListener( this ) ;

```

Когда пользователь нажимает кнопку, которая генерирует событие, интерфейс **ActionListener** распознает данное событие и ищет метод обработчика событий в программе для выполнения ответных действий.

Каждый интерфейс **EventListener** имеет связанный с ним метод обработчика событий, который вызывается, когда событие распознается. Например, когда нажимается кнопка, интерфейс **ActionListener** вызывает связанный метод с именем **actionPerformed()** и передает в качестве его аргумента объект **ActionEvent**.

Объект **ActionEvent** содержит информацию о событии и компоненте-источнике, вызвавшем его. Существует полезный метод **getSource()**, который идентифицирует объект, вызвавший событие. Его можно использовать для создания соответствующего ответного действия для компонента.

Метод обработчика события, создающий ответ на нажатие кнопки, может выглядеть приблизительно так:

```

public void actionPerformed( ActionEvent event )
{
    if ( event.getSource() == btn )
    {
        Операторы, которые будут выполняться по нажатию кнопки
    }
}

```

Обработка событий кнопок

Компонент **JButton** в библиотеке **Swing** генерирует событие **ActionEvent**, которое распознается интерфейсом **ActionListener** и передается методу-обработчику **actionPerformed()**. У объекта **ActionEvent** существует метод **getSource()**, который идентифицирует исходный компонент, являющийся инициатором события, а также метод **getActionCommand()**, возвращающий строковое значение. Оно может быть либо текстовой меткой для кнопки, либо содержимым текстового поля компонента.

Одним из возможных ответов на событие нажатия кнопки может быть отключение компонента вызовом его метода **setEnabled()** с аргументом **false**, либо повторное его включение при помощи этого же метода **setEnabled()** с аргументом **true**.



Actions.java

1. Отредактируйте копию файла *Window.java*, заменив имя класса **Window** в объявлении, конструкторе и операторе экземпляра на **Actions**.
2. В самое начало программы добавьте оператор для импортирования функциональности всех классов пакета **java.awt.event**.

```
import java.awt.event.* ;
```
3. Отредактируйте объявление класса для добавления интерфейса **ActionListener** в программу.

```
class Actions extends JFrame implements ActionListener
```
4. Перед конструктором **Actions()** создайте две кнопки **JButton**, а также текстовое поле **JTextArea**.

```
JButton btn1 = new JButton( "Кнопка 1" ) ;  
JButton btn2 = new JButton( "Кнопка 2" ) ;  
JTextArea txtArea = new JTextArea( 5 , 38 ) ;
```
5. Добавьте кнопки и текстовую область на контейнер **JPanel**.

```
pnl.add( btn1 ) ;  
pnl.add( btn2 ) ;  
pnl.add( txtArea ) ;
```
6. Вставьте операторы для установки начального состояния двух компонентов.

```
btn2.setEnabled( false ) ;  
txtArea.setText( "Кнопка 2 деактивирована" ) ;
```

7. В конструкторе `Actions()` вставьте операторы, которые генерируют событие `ActionEvent` при нажатии кнопок.

```
btn1.addActionListener( this ) ;
```

```
btn2.addActionListener( this ) ;
```

8. После метода конструктора добавьте метод-обработчик события для интерфейса `ActionListener`, который будет отображать текст, идентифицирующий нажатую кнопку.

```
public void actionPerformed( ActionEvent event )
```

```
{
```

```
    txtArea.setText( event.getActionCommand()
```

```
        + " Нажата и деактивирована" ) ;
```

```
}
```

9. Вставьте операторы `if`, которые позволяют выполнить определенное ответное действие на нажатие каждой кнопки, в метод обработчика событий.

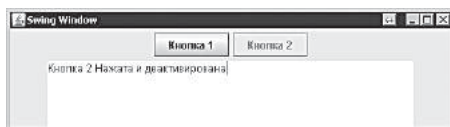
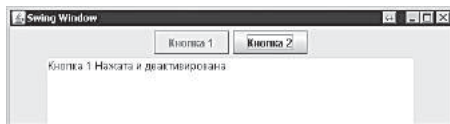
```
if ( event.getSource() == btn1 )
```

```
{ btn2.setEnabled( true ) ; btn1.setEnabled( false ) ; }
```

```
if ( event.getSource() == btn2 )
```

```
{ btn1.setEnabled( true ) ; btn2.setEnabled( false ) ; }
```

10. Сохраните программу под именем `Actions.java`, затем скомпилируйте и запустите, пробуя нажимать кнопки.



На заметку



Компоненты объявлены перед конструктором, поэтому они доступны методу-обработчику событий.

Совет



Иногда полезно деактивировать компонент до того, как пользователь выполнит требуемое действие.

Обработка событий элементов

Изменения состояния компонентов `JRadioButton`, `JCheckBox` и `JComboBox` библиотеки Swing могут распознаваться интерфейсом `ItemListener`, который, в свою очередь, передает событие `ItemEvent` методу-обработчику `itemStateChanged()`. У объекта `ItemEvent` существует метод `getItemSelectable()`, который идентифицирует компонент, вызвавший событие, а также метод `getStateChange()`, возвращающий состояние компонента. Данный метод позволяет определить, что явилось причиной изменения статуса — выбор или отмена выбора элемента. Возвращенный статус может быть сравнен с константой `ItemEvent.SELECTED`.



States.java

1. Отредактируйте копию файла *Window.java*, заменив имя класса **Window** в объявлении, конструкторе и операторе экземпляра на **States**. Затем добавьте в самое начало программы оператор для импортирования функциональности пакета `java.awt.event`.

```
import java.awt.event.* ;
```

2. Отредактируйте объявление класса, чтобы добавить интерфейс `ItemListener` в программу.

```
class States extends JFrame implements ItemListener
```

3. Перед конструктором `States()` создайте следующие компоненты.

```
String[] styles =
{ "В глубокой форме" , "Для гурманов" , "Тонкая" } ;
JComboBox<String> box = new JComboBox<String> ( styles ) ;
JRadioButton rad1 = new JRadioButton( "Белое" ) ;
JRadioButton rad2 = new JRadioButton( "Красное" ) ;
ButtonGroup wines = new ButtonGroup() ;
JCheckBox chk = new JCheckBox( "Пеперони" ) ;
JTextArea txtArea = new JTextArea( 5 , 38 ) ;
```

4. В конструкторе `States()` вставьте операторы для группировки двух компонентов `JRadioButton`.

```
wines.add( rad1 ) ;
wines.add( rad2 ) ;
```

5. Вставьте операторы для добавления компонентов на контейнер `JPanel`.

```
pnl.add( rad1 ) ;
pnl.add( rad2 ) ;
pnl.add( chk ) ;
```

Совет



Обратите внимание, как добавляется текст в текстовую область при помощи метода `append()`.

```
pnl.add( box ) ;
pnl.add( txtArea ) ;
```

- Добавьте операторы, чтобы выбираемые компоненты генерировали событие `ItemEvent` при выборе или отмене выбора элемента.

```
rad1.addItemListener( this ) ;
rad2.addItemListener( this ) ;
chk.addItemListener( this ) ;
box.addItemListener( this ) ;
```

- После метода конструктора добавьте метод-обработчик события для интерфейса `ItemListener`, который идентифицирует элементы, выбранные компонентами `JRadioButton`.

```
public void itemStateChanged( ItemEvent event )
{
    if ( event.getItemSelectable() == rad1 )
        txtArea.setText( "Выбрано белое вино" ) ;
    if ( event.getItemSelectable() == rad2 )
        txtArea.setText( "Выбрано красное вино" ) ;
}
```

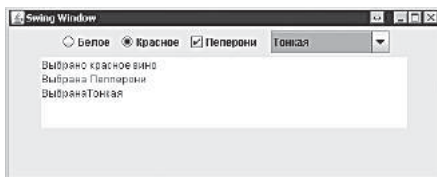
- Добавьте оператор `if` в обработчик события для определения статуса компонента `JCheckBox`.

```
if ( ( event.getItemSelectable() == chk ) &&
    ( event.getStateChange() == ItemEvent.SELECTED ) )
    txtArea.append( "\nВыбрана Пепперони\n" ) ;
```

- Добавьте оператор `if` в обработчик события для определения статуса компонента `JComboBox`.

```
if ( ( event.getItemSelectable() == box ) &&
    ( event.getStateChange() == ItemEvent.SELECTED ) )
    txtArea.append( "Выбрана" + event.getItem().toString() ) ;
```

- Сохраните программу под именем `States.java`, затем скомпилируйте и запустите, пробуя выбрать различные элементы слева направо.



Внимание

Компонент `JComboBox` порождает два события `ItemEvents` при выборе элемента: первое — это выбор элемента, а второе — отмена выбора предыдущего выбранного элемента. Вот почему в шагах 8 и 9 нужно идентифицировать оба порождающих компонента, а также тип события `ItemEvent`.



Совет

Обратите внимание, как метод `getItem()` возвращает измененный элемент.

Реагирование на события клавиатуры

Компоненты библиотеки Swing, которые позволяют пользователю вводить текст, могут распознавать нажатие клавиш при помощи интерфейса **KeyListener**, который передает событие **KeyEvent** следующим трем методам-обработчикам:

Обработчик события	Описание
<code>keyPressed(KeyEvent)</code>	Вызывается при нажатии клавиши на клавиатуре
<code>keyTyped(KeyEvent)</code>	Вызывается после нажатия клавиши на клавиатуре
<code>keyReleased(KeyEvent)</code>	Вызывается при освобождении клавиши на клавиатуре

Если в программе реализуется интерфейс **KeyListener**, эти три метода должны быть обязательно объявлены (даже если используются не все три).

У объекта **KeyEvent** существует метод `getKeyChar()`, который возвращает символ, соответствующий нажатой клавише, а также метод `getKeyCode()`, возвращающий целочисленное значение, представляющее данную клавишу в формате ЮНИКОД. Дополнительный метод `getKeyText()` принимает в качестве аргумента код клавиши и возвращает описание этой клавиши.



Keystrokes.java

1. Отредактируйте копию программы *Window.java*, заменив имя класса **Window** в объявлении, конструкторе и экземпляре на **Keystrokes**. Затем добавьте в начало импортирование функциональности пакета `java.awt.event`.

```
import java.awt.event.* ;
```

2. Отредактируйте объявление класса, чтобы добавить в программу интерфейс **KeyListener**.

```
class Keystrokes extends JFrame implements KeyListener
```

3. Перед конструктором **Keystrokes()** создайте компонент **JTextField** и компонент **JTextArea**.

```
JTextField field = new JTextField( 38 ) ;
```

```
JTextArea txtArea = new JTextArea( 5 , 38 ) ;
```

4. Вставьте операторы для добавления этих двух компонентов на контейнер `JPanel`.

```
pnl.add( field ) ; pnl.add( txtArea ) ;
```

5. В конструкторе `Keystrokes()` добавьте оператор, который заставляет компонент `JTextField` генерировать события `KeyEvent`.

```
field.addKeyListener( this ) ;
```

6. После метода конструктора добавьте обработчик событий, который распознает нажатие клавиш.

```
public void keyPressed( KeyEvent event )
{
    txtArea.setText( "Нажата клавиша" ) ;
}
```

7. Добавьте еще один обработчик события, который отображает символ клавиши после того, как она нажата.

```
public void keyTyped( KeyEvent event )
{
    txtArea.append( "\nСимвол : "
        + event.getKeyChar() ) ;
}
```

8. Добавьте третий обработчик события, который будет отображать код клавиши и текст, когда клавиша отпущена.

```
public void keyReleased( KeyEvent event )
{
    int keyCode = event.getKeyCode() ;
    txtArea.append( "\nКод клавиши : "
        + event.getKeyCode() ) ;
    txtArea.append( "\nТекст клавиши : "
        + event.getKeyText( keyCode ) ) ;
}
```

9. Сохраните программу под именем `Keystrokes.java`, затем скомпилируйте и запустите, пробуя набрать что-нибудь в текстовом поле сверху.



Внимание

Метод `getKeyCode()` возвращает код клавиши в случае, если вызван обработчик `keyPressed()` или `keyReleased()`, но не `keyTyped()`.



Совет

Запустите программу и нажмите несимвольную клавишу, например **Пробел**, чтобы увидеть ее текстовое название.

Ответ на события мыши

Компоненты библиотеки Swing могут распознавать действия пользователя с мышью при помощи интерфейса **MouseListener**, который передает событие **MouseEvent** следующим пяти методам-обработчикам.

Обработчик события	Описание
<code>mousePressed(MouseEvent)</code>	Нажата кнопка мыши
<code>mouseReleased(MouseEvent)</code>	Освобождена кнопка мыши
<code>mouseClicked(MouseEvent)</code>	Выполнен щелчок кнопкой мыши
<code>mouseEntered(MouseEvent)</code>	Указатель входит в область
<code>mouseExited(MouseEvent)</code>	Указатель выходит из области

Движение мыши можно распознавать при помощи интерфейса **MouseMotionListener**, который передает событие **MouseEvent** двум обработчикам:

Обработчик события	Описание
<code>mouseMoved(MouseEvent)</code>	Перемещение указателя
<code>mouseDragged(MouseEvent)</code>	Перетаскивание (с нажатой кнопкой)

Если в программе реализованы интерфейсы **MouseListener** или **MouseMotionListener**, то нужно объявлять все методы-обработчики (даже если они не используются).

У объекта **MouseEvent**, который передается интерфейсу **MouseMotionListener**, существуют методы `getX()` и `getY()`, которые возвращают текущие координаты указателя мыши относительно графического компонента, сгенерировавшего событие.



Mouse.java

1. Отредактируйте копию файла *Window.java*, заменив имя класса **Window** в объявлении, конструкторе и операторе экземпляра на **Mouse**. Затем добавьте начальный оператор для импортирования функциональности пакета `java.awt.event`.

```
import java.awt.event.* ;
```

2. Отредактируйте объявление класса, чтобы добавить интерфейсы **MouseListener** и **MouseMotionListener** в программу.

```
class Mouse extends JFrame
```

```
implements MouseListener , MouseMotionListener
```

3. Перед конструктором `Mouse()` создайте компонент `JTextArea`, а также две целочисленные переменные для хранения координат указателя.

```
JTextArea txtArea = new JTextArea( 8 , 38 ) ;

int x , y ;
```

4. В конструкторе `Mouse()` вставьте операторы для добавления компонентов `JTextArea` на контейнер `JPanel`, а также для генерации событий `MouseEvent`.

```
pnl.add( txtArea ) ;

txtArea.addMouseMotionListener( this ) ;

txtArea.addMouseListener( this ) ;
```

5. После метода конструктора добавьте два обработчика событий для интерфейса `MouseMotionListener`.

```
public void mouseMoved( MouseEvent event )

{ x = event.getX() ; y = event.getY() ; }

public void mouseDragged( MouseEvent event ) { }
```

6. Добавьте пять обработчиков событий для интерфейса `MouseListener`.

```
public void mouseEntered( MouseEvent event )

{ txtArea.setText( "\nНажата кнопка мыши" ) ; }

public void mousePressed( MouseEvent event )

{ txtArea.append( "\nКнопка нажата, когда указатель в позиции X: " +x+ "Y: " +y ) ; }

public void mouseReleased( MouseEvent event )

{ txtArea.append( "\nКнопка мыши отпущена" ) ; }

public void mouseClicked(MouseEvent event ) { }

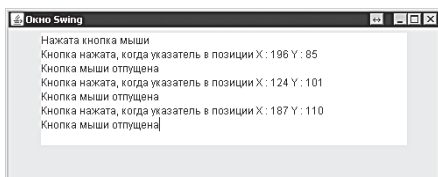
public void mouseExited(MouseEvent event ) { }
```

7. Сохраните программу под именем `Mouse.java`, затем скомпилируйте и запустите, щелкнув указателем мыши по компоненту `JTextArea`.

Совет



При помощи замены изображений в методах-обработчиках `mouseEntered()` и `mouseExited()` можно достичь эффекта перекачивания (ролловера).



Вывод сообщений

В библиотеке Swing для создания стандартного диалогового окна существует класс `JOptionPane`. Метод `showMessageDialog()` данного класса отображает сообщение пользователю; в сообщении размещается различная информация, предупреждение или описание ошибок. Отображаемое сообщение центрируется относительно родительского окна.

Метод `showMessageDialog()` может принимать четыре аргумента:

- родительский объект — обычно на него ссылается ключевое слово `this`;
- строковое сообщение для отображения;
- строковый заголовок диалога;
- одна из констант: `INFORMATION_MESSAGE`, `WARNING_MESSAGE` или `ERROR_MESSAGE`.

В зависимости от того, какая указана константа, в диалоговом окне будет отображаться соответствующая иконка.



Messages.java

1. Отредактируйте копию файла *Window.java*, заменив имя класса **Window** в объявлении, конструкторе и операторе экземпляра на **Messages**.
2. Добавьте начальный оператор для импортирования функциональности пакета `java.awt.event`.

```
import java.awt.event.* ;
```
3. Отредактируйте объявление класса, чтобы добавить интерфейс **ActionListener** в программу.

```
class Messages extends JFrame implements ActionListener
```
4. Перед конструктором **Messages()** создайте три компонента **JButton**.

```
JButton btn1=new JButton( "Показать информационное сообщение" ) ;  
JButton btn2= new JButton( "Показать предупреждение" ) ;  
JButton btn3= new JButton( "Показать сообщение об ошибке" ) ;
```
5. Вставьте операторы для добавления компонентов кнопок на контейнер **JPanel**.

```
pn1.add( btn1 ) ;  
pn1.add( btn2 ) ;  
pn1.add( btn3 ) ;
```

6. В конструкторе `Messages()` вставьте операторы, генерирующие события `ActionEvent` для каждой кнопки.

```
btn1.addActionListener( this ) ;  
btn2.addActionListener( this ) ;  
btn3.addActionListener( this ) ;
```

7. После метода конструктора добавьте метод-обработчик событий для интерфейса `ActionListener`.

```
public void actionPerformed( ActionEvent event ) { }
```

8. Внутри фигурных скобок метода-обработчика вставьте условные операторы, отображающие диалог при нажатии клавиш.

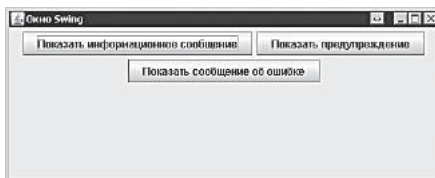
```
if ( event.getSource() == btn1 )  
    JOptionPane.showMessageDialog( this , "Информация..." ,  
    "Диалоговое сообщение" , JOptionPane.INFORMATION_MESSAGE ) ;  
if ( event.getSource() == btn2 )  
    JOptionPane.showMessageDialog( this , "Предупреждение..." ,  
    "Диалоговое сообщение" , JOptionPane.WARNING_MESSAGE ) ;  
if ( event.getSource() == btn3 )  
    JOptionPane.showMessageDialog( this , "Ошибка..." ,  
    "Диалоговое сообщение" , JOptionPane.ERROR_MESSAGE ) ;
```

9. Сохраните программу под именем `Messages.java`, затем скомпилируйте и запустите, щелкнув указателем по каждой из кнопок.

Совет



Вы можете указать только два аргумента — родительское окно и сообщение, тем самым создав диалог, в котором будут использоваться значок и заголовок по умолчанию.



Запрос пользовательского ввода

В библиотеке Swing существует класс `JOptionPane`, позволяющий запрашивать информацию у пользователя при помощи двух методов, открывающих диалоговое окно — метод `showConfirmationDialog()`, запрашивающий подтверждение, а также метод `showInputDialog()`, который предлагает ввести пользователю строку.

Оба эти метода могут принимать четыре аргумента:

- родительский объект — ссылка на него определяется ключевым словом `this`;
- строка запроса для отображения;
- строка, определяющая заголовок диалогового окна;
- одна из констант `JOptionPane`, таких как `PLAIN_MESSAGE` и определяющая кнопки для подтверждения, `YES_NO_CANCEL_OPTION`.

Диалоговое окно будет возвращать строку ввода в случае с диалогом ввода или целое число в результате нажатия на кнопки подтверждения — 0 для подтверждения, 1 для отказа или 2 для отмены.



Request.java

1. Отредактируйте копию файла *Window.java*, заменив имя класса `Window` в объявлении, конструкторе и операторе экземпляра на `Request`. После этого добавьте начальный оператор для импортирования функциональности пакета `java.awt.event`.

```
import java.awt.event.* ;
```

2. Отредактируйте объявление класса, чтобы добавить в программу интерфейс `ActionListener`.

```
class Request extends JFrame implements ActionListener
```

3. Перед конструктором `Request()` создайте компоненты `JTextField` и два компонента `JButton`.

```
JTextField field = new JTextField( 38 ) ;
```

```
JButton btn1 = new JButton( "Запрос подтверждения" ) ;
```

```
JButton btn2 = new JButton( "Запрос ввода" ) ;
```

4. Добавьте все компоненты на контейнер `JPanel`.

```
pnl.add( field ) ; pnl.add( btn1 ) ; pnl.add( btn2 ) ;
```

5. В конструкторе `Request()` вставьте операторы, которые будут генерировать событие `ActionEvent` для каждой кнопки.

```
btn1.addActionListener( this ) ;
```

```
btn2.addActionListener( this ) ;
```

6. После метода конструктора добавьте метод обработчика событий для интерфейса `ActionListener`.

```
public void actionPerformed((ActionEvent event) { }
```

7. Внутри фигурных скобок обработчика событий вставьте условный оператор, для того чтобы организовать ответ на нажатие кнопок.

```
if ( event.getSource() == btn1 )
{
    int n = JOptionPane.showConfirmDialog( this ,
        "Вы согласны?" , "Диалог подтверждения" ,
        JOptionPane.YES_NO_CANCEL_OPTION ) ;
    switch( n )
    {
        case 0 : field.setText( "Согласен" ) ; break ;
        case 1 : field.setText( "Не согласен" ) ; break ;
        case 2 : field.setText( "Отменено" ) ; break ;
    }
}
```

8. Вставьте условный оператор для обработки пользовательского ввода.

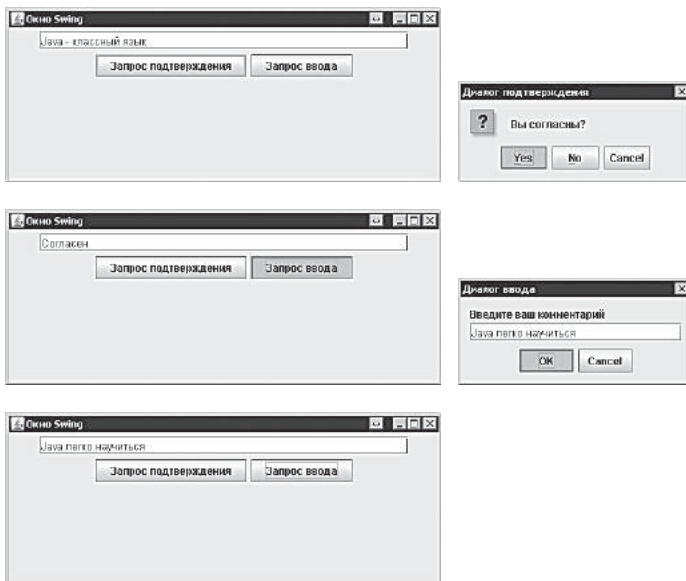
```
if ( event.getSource() == btn2 )
field.setText( JOptionPane.showInputDialog( this ,
    "Введите ваш комментарий" , "Диалог ввода" ,
    JOptionPane.PLAIN_MESSAGE ) ) ;
```

Сохраните программу под именем *Request.java*, затем скомпилируйте и запустите, выбрав указателем каждую из кнопок.

Совет



Константа `OK_CANCEL` предлагает две кнопки для подтверждения: `OK` возвращает 0, а `CANCEL` возвращает 2. Обратитесь к документации, чтобы узнать информацию о полном наборе констант.



Воспроизведение звука

Пакет `java.applet` содержит класс с именем `AudioClip`, в который встроены методы для проигрывания аудиофайлов форматов `.au`, `.aiff`, `.wav`, а также `.mid`. Совместимость с компонентами библиотеки Swing поддерживается при помощи класса `JApplet`, который предлагает метод `newAudioClip()`, возвращающий URL-адрес аудиофайла.

На заметку



Подробнее о загрузке файлов изображений при помощи `ClassLoader` см. в главе 8.



Sound.java



music.wav

Аудиофайл должен быть загружен в качестве ресурса в объект `AudioClip` при помощи метода `getResource()` объекта `ClassLoader` — точно так же, как и файл изображения.

В ответ на пользовательские действия воспроизведение аудиофайла может быть начато и прекращено при помощи методов `play()` и `stop()` объекта `AudioClip`. Кроме того, аудиофайл может быть воспроизведен непрерывно при помощи метода `loop()`.

1. Отредактируйте копию файла *Window.java*, заменив имя класса `Window` в объявлении, конструкторе и операторе экземпляра на `Sound`.

2. Добавьте начальный оператор для импортирования функциональности всех классов пакета `java.awt.event`.

```
import java.awt.event.* ;
```

3. Отредактируйте объявление класса для добавления интерфейса `ActionListener` в программу.

```
class Sound extends JFrame implements ActionListener
```

4. Перед конструктором `Sound()` создайте объект `ClassLoader`.

```
ClassLoader ldr = this.getClass().getClassLoader() ;
```

5. Создайте объект `AudioClip` и загрузите файл звукового ресурса при помощи объекта `ClassLoader`.

```
java.applet.AudioClip audio =
```

```
JApplet.newAudioClip( ldr.getResource( "music.wav" ) ) ;
```

6. Создайте два компонента `JButton` для управления воспроизведением звука.

```
JButton playBtn = new JButton( "Играть" ) ;
```

```
JButton stopBtn = new JButton( "Стоп" ) ;
```

7. Добавьте кнопки на контейнер **JPanel**.

```
pnl.add( playBtn ) ;  
pnl.add( stopBtn ) ;
```

8. В конструкторе **Sound()** вставьте операторы, чтобы генерировать событие **ActionEvent** для каждой кнопки при их нажатии.

```
playBtn.addActionListener( this ) ;  
stopBtn.addActionListener( this ) ;
```

9. После метода конструктора добавьте обработчик события для интерфейса **ActionListener**.

```
public void actionPerformed((ActionEvent event) { }
```

10. Внутри фигурных скобок обработчика событий вставьте оператор для проигрывания аудиофайла по нажатию кнопки воспроизведения.

```
if ( event.getSource() == playBtn ) audio.play() ;
```

11. Вставьте оператор для остановки воспроизведения аудиофайла по нажатию кнопки остановки.

```
if ( event.getSource() == stopBtn ) audio.stop() ;
```

12. Сохраните программу под именем *Sound.java*, затем скомпилируйте и запустите, используя кнопки для контроля над воспроизведением аудиофайла.

Совет

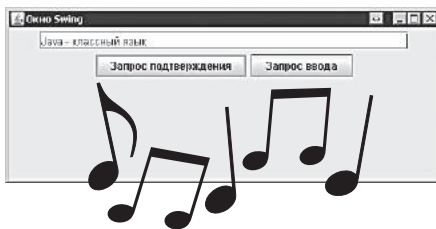


Можно воспроизводить несколько звуковых объектов одновременно.

На заметку



Каждый раз, когда вызывается метод `play()`, воспроизведение стартует с начала файла.



Заключение

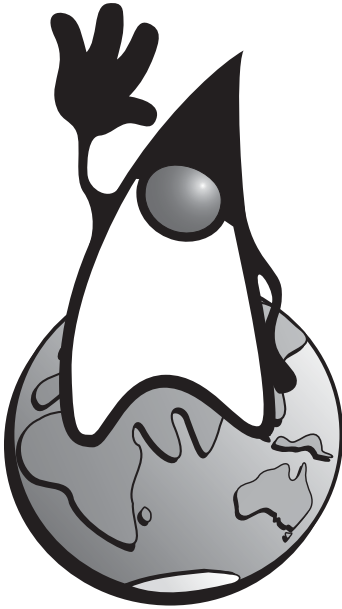
- Для добавления одного или нескольких интерфейсов **EventListener** в объявление класса используется ключевое слово **implements**.
- Чтобы заставить компонент сгенерировать событие **ActionEvent** при его активации, используется метод компонента **addActionListener()**, который принимает в качестве аргумента ключевое слово **this**.
- Интерфейс **ActionListener** передает сгенерированное событие **ActionEvent** в качестве аргумента методу-обработчику события **actionPerformed()**, который может реагировать на нажатие кнопки пользователем.
- Для того чтобы идентифицировать компонент, который сгенерировал событие, используется метод **getSource()**.
- Интерфейс **ItemListener** создает сгенерированное событие **ItemEvent** в качестве аргумента методу-обработчику **itemStateChanged()**, который может реагировать на выбор элемента пользователем.
- Для идентификации компонента, сгенерировавшего событие, используется метод **getItemSelectable()**.
- Интерфейс **KeyListener** передает сгенерированное событие **KeyEvent** в качестве аргумента трем обязательным методам-обработчикам событий, которые могут реагировать на нажатие кнопки пользователем и возвращать символ, соответствующий нажатой кнопке.
- Интерфейс **MouseListener** передает сгенерированное событие **MouseEvent** в качестве аргумента пяти обязательным методам-обработчикам, которые могут реагировать на действия пользователя с мышью.
- Интерфейс **MouseMotionListener** передает сгенерированное событие **MouseEvent** в качестве аргумента двум обязательным обработчикам событий, которые могут реагировать на движения указателя мыши.
- Метод **showMessageDialog()** класса **JOptionPane** создает диалоговое окно, отображающее пользователю сообщение, а для запроса пользовательского ввода используются методы **showInputDialog()** и **showConfirmationDialog()**.
- Звуковые ресурсы могут представляться при помощи класса **AudioClip** пакета **java.applet** и проигрываются при помощи метода этого класса **play()**.

10

Развертывание программ

В данной главе демонстрируется, как развертывать Java-программы при помощи настольных приложений и встроенных в веб-страницы апплетов.

- Методы развертывания
- Распространение программ
- Построение архивов
- Развертывание приложений
- Подписывание jar-файлов
- Использование технологии Web Start
- Изготовление апплетов
- Преобразование веб-страниц
- Развертывание апплетов
- Заключение



Внимание



Программы в данной книге скомпилированы при помощи Java 8 — для корректного выполнения приложения требуется установка соответствующего Java 8 JRE, а для выполнения апплетов — плагин Java 8.

Методы развертывания

Технология Java предлагает два различных способа для развертывания программ:

- настольные приложения, которые могут быть запущены на любой из платформ, где установлено соответствующее окружение Java Runtime Environment;
- апплеты для веб-страниц, которые выполняются на любом веб-браузере, где установлен соответствующий Java-плагин.

Принцип выбора метода развертывания обычно определяется назначением программ. Например, апплеты используются в легких программах, которые должны быть интегрированы в содержимое веб-страниц, в то время как настольные приложения предпочтительнее использовать для более серьезных программ.

Как апплеты, так и настольные приложения могут быть созданы изначально из общей программы, наподобие представленной ниже программы *Lotto.java*.

Конструктор `Lotto()` генерирует простой интерфейс Swing, представляющий собой панель, содержащую одну метку, одно текстовое поле и одну кнопку. Данная панель помещена на окно размерами 260×200 пикселей. Компонент «метка» просто отображает описательное графическое изображение:



Lotto.png — области с шахматным рисунком являются прозрачными.

Метод-обработчик для кнопки запускает алгоритм выбора последовательности из шести случайных чисел в диапазоне от 1 до 49, которые будут отображаться в компоненте «текстовое поле».

Программа *Lotto.java*, представленная здесь, используется на протяжении всей главы для создания, как приложения, так и апплета.

```
import javax.swing.* ;
import java.awt.event.* ;

public class Lotto extends JFrame implements ActionListener
{
    // Компоненты.
    ClassLoader ldr = this.getClass().getClassLoader() ;
    java.net.URL iconURL = ldr.getResource( "Lotto.png" ) ;
```

```

ImageIcon icon = new ImageIcon( iconURL ) ;
JLabel img = new JLabel( icon ) ;
JTextField txt = new JTextField( "" , 18 ) ;
JButton btn = new JButton( "Показать счастливые номера" ) ;
JPanel pnl = new JPanel() ;
// Конструктор.
public Lotto()
{
    super( "Приложение Lotto" ) ; setSize( 260 , 200 ) ;
    setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE ) ;
    pnl.add( img ) ; pnl.add( txt ) ; pnl.add( btn ) ;
    btn.addActionListener( this ) ;
    add( pnl ) ; setVisible( true ) ;
}
// Обработчик событий.
public void actionPerformed((ActionEvent event)
{
    if ( event.getSource() == btn )
    {
        int[] nums = new int[50] ; String str = "" ;
        for ( int i = 1 ; i < 50 ; i++ ) { nums[i] = i ; }
        for ( int i = 1 ; i < 50 ; i++ )
        { int r = (int) ( 49 * Math.random() ) + 1 ;
          int temp= nums[i] ; nums[i]= nums[r] ; nums[r]= temp ; }
        for ( int i = 1 ; i < 7 ; i++ )
        { str += " " + Integer.toString( nums[i] ) + " " ; }
        txt.setText( str ) ;
    }
}
// Точка входа.
public static void main ( String[] args )
{ Lotto lotto = new Lotto() ; }
}

```



Lotto.java

Совет



Данный обработчик событий перемешивает целые числа в массиве, а затем первые шесть чисел заносит в строковый массив.

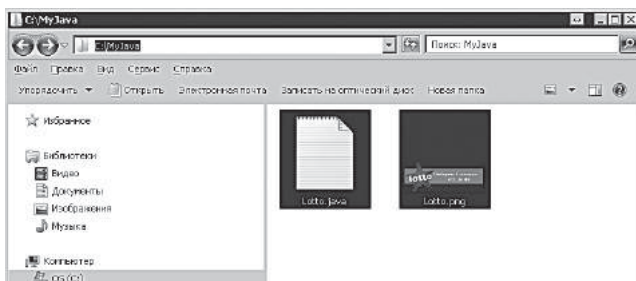
Распространение программ

Объект **ClassLoader** в программе *Lotto.java* с предыдущей страницы ожидает, что графический файл *Lotto.png* размещен в том же каталоге, что и файл программы, поэтому, прежде чем попытаться скомпилировать программу, нужно убедиться, что файлы размещены именно таким образом.

Внимание



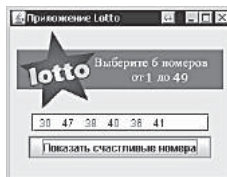
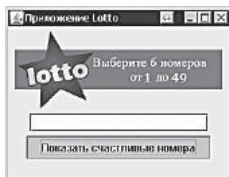
Файл изображения должен быть именован в точности так: *Lotto.png* (а не *lotto.png*).



Убедившись, что файлы расположены так, как это требуется, можно запускать компилятор **javac** для создания файла *Lotto.class*, а затем интерпретатор **java** для запуска программы.

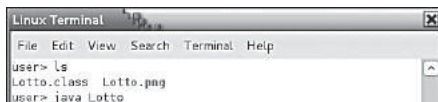


Программа **Lotto** открывает новое окно определенного размера, содержащее компоненты интерфейса Swing. Каждый раз, когда пользователь нажимает на кнопку, обработчик события отображает шесть новых случайных чисел в диапазоне от 1 до 49 в текстовом поле.



Программа **Lotto**, как и другие примеры данной книги, скомпилирована для Java 8. Ее можно распространять для исполнения на других компьютерах, где установлено Java 8 Runtime Environment независимо от операционной системы.

Например, на скриншотах ниже файлы *Lotto.class* и *Lotto.png* скопированы на рабочий стол компьютера, на котором установлена операционная система Linux и Java 8 Runtime Environment. Поэтому программа **Lotto** может быть запущена при помощи интерпретатора **java** точно так же, как и на исходной операционной системе Windows:



```
Linux Terminal
File Edit View Search Terminal Help
user> ls
Lotto.class  Lotto.png
user> java Lotto
```



При распространении таким образом Java-программ существует, однако, опасность, что программа не выполнится, если файлы ресурсов станут недоступны — в данном случае отсутствие файла *Lotto.png* приведет к следующей ошибке:



```
Linux Terminal
File Edit View Search Terminal Help
user> ls
Lotto.class
user> java Lotto
Exception in thread "main" java.lang.NullPointerException
  at javax.swing.ImageIcon.<init>(ImageIcon.java:295)
  at Lotto.<init>(Lotto.java:9)
  at Lotto.main(Lotto.java:61)
user>
```

Совет



Для распространения программы не требуется включать файл исходных кодов *.java*. Необходимы только файлы *.class*, а также ресурсные файлы.

На заметку



В больших программах может использоваться большое количество файлов ресурсов, корректное расположение которых может быть случайно нарушено пользователем. Решение, описанное ниже, заключается в упаковке программы и всех ее ресурсов в один исполняемый архивный файл.

Построение архивов

Пакет JDK содержит утилиту **jar**, которая позволяет упаковывать файлы программ, а также все ресурсные файлы в один файловый Java-архив (JAR). Данная утилита сжимает все файлы, используя популярный формат ZIP, и создает файл с расширением *.jar*. JAR-архивы являются эффективным методом и помогают избавиться от случайного удаления ресурсных файлов. Интерпретатор **java** полностью поддерживает JAR-архивы, так что приложения и апплеты могут быть запущены без необходимости извлекать отдельные файлы из архива. Инструмент **jar**, так же, как и интерпретатор **java** и компилятор **javac**, размещен в каталоге *bin*, и его можно запускать из командной строки для выполнения следующих основных операций с архивами.

Совет



В случае с большими программами для архивации множественных файлов можно использовать символ шаблона ***. Например, команда `jar cf Program.jar *.class` архивирует все файлы классов в текущем каталоге.

Синтаксис команды	Операция
<code>jar cf jar-файл входные файлы</code>	Создание JAR-архива
<code>jar cfe jar-файл точка_входа входные файлы</code>	Создание JAR-архива с указанием точки входа в приложение
<code>jar tf jar-файл</code>	Просмотр содержимого JAR-архива
<code>jar uf jar-файл</code>	Обновление содержимого JAR-архива
<code>jar ufm jar-файл файл_атрибутов</code>	Обновление содержимого манифеста JAR-архива с добавлением в него атрибутов
<code>jar xf jar-файл</code>	Извлечение содержимого JAR-архива
<code>jar xf jar-файл архивный_файл</code>	Извлечение определенного файла из содержимого JAR-архива

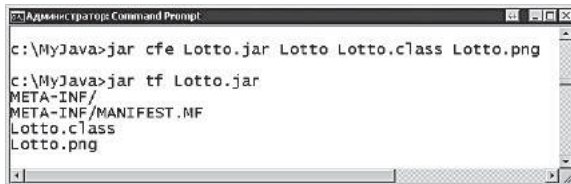
Выполните следующие шаги, чтобы создать JAR-архив для программы **Lotto**, описанной в начале главы.



Lotto.jar

1. Откройте командную строку (терминальное окно) и перейдите в каталог, в котором размещены файлы программы **Lotto** — *Lotto.class* и *Lotto.png*.
2. В командной строке наберите `jar cfe Lotto.jar Lotto Lotto.class Lotto.png`, затем нажмите клавишу **Enter** для создания архива *Lotto.jar*.

3. Теперь наберите `jar tf Lotto.jar` для просмотра содержимого JAR-архива.



```
Администратор: Command Prompt
c:\MyJava>jar cfe Lotto.jar Lotto Lotto.class Lotto.png
c:\MyJava>jar tf Lotto.jar
META-INF/
META-INF/MANIFEST.MF
Lotto.class
Lotto.png
```

Обратите внимание, что инструмент `jar` автоматически создает каталог `META-INF` рядом с архивными файлами. Он содержит текстовый файл манифеста с именем `MANIFEST.MF`. В данный файл манифеста добавляется атрибут `permissions`, который определяет доступ к ресурсам приложения.

4. Запустите простой текстовый редактор и наберите в нем следующее:

Application-Name: Lotto

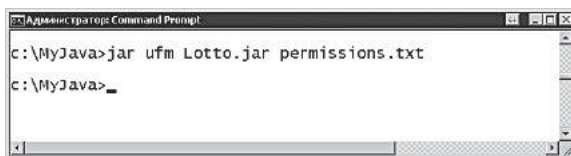
Permissions: all-permissions

Затем сохраните файл под именем `permissions.txt` рядом с JAR-архивом.

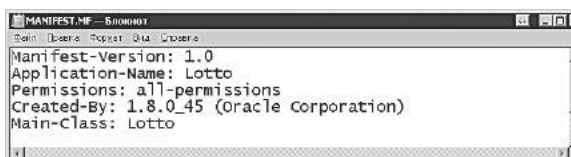


```
permissions.txt — Блокнот
Файл  Формат  Формат  Формат  Формат
Application-Name: Lotto
Permissions: all-permissions
```

5. Наберите команду `jar ufm Lotto.jar permissions.txt`, затем нажмите клавишу `Enter` для добавления атрибута `permissions` в манифест.



```
Администратор: Command Prompt
c:\MyJava>jar ufm Lotto.jar permissions.txt
c:\MyJava>
```



```
MANIFEST.MF — Блокнот
Файл  Формат  Формат  Формат  Формат
Manifest-Version: 1.0
Application-Name: Lotto
Permissions: all-permissions
Created-By: 1.8.0_45 (Oracle Corporation)
Main-Class: Lotto
```



Внимание

Файл `permissions.txt` должен оканчиваться символом новой строки — нажмите клавишу **Enter** после последней строки перед сохранением файла.



`permissions.txt`



Совет

Извлеките копию каталога `META-INF`, используя команду `jar xf Lotto.jar META-INF`, чтобы изучить файл `MANIFEST.MF` — он должен выглядеть точно так же, как показано на последнем рисунке.

Развертывание приложений

Файлы JAR-архивов являются исполняемыми на любой системе, на которой установлена соответствующая версия Java Runtime.

Внимание



При запуске JAR-файлов расширение *.jar* необходимо указывать.

1. В командной строке перейдите в каталог, в котором расположен файл *Lotto.jar*, затем наберите команду `java -jar Lotto.jar` и нажмите клавишу **Enter** для запуска приложения *Lotto*.

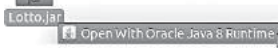
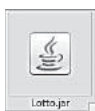


Совет



Установите для файлов JAR запускемым по умолчанию приложением JRE.

2. Альтернативным методом запуска является двойной щелчок мышью по файлу *Lotto.jar*, либо щелчок правой кнопкой мыши и выбор команды **Открыть с помощью** (Open with) и **Java Runtime**.



How do you want to open this file?

☒ Use this app for all jar files



Keep using Java(TM) Platform SE binary

Подписывание jar-файлов

В случае если Java-программа предназначена для широкого распространения, JAR-архивы подписывают цифровой подписью, для того чтобы пользователи, запускающие их, могли удостовериться, что программа выпущена доверенным источником. Попытка запустить неподписанное Java-приложение, распространяемое через Интернет при помощи технологии Java Web Start или через Java-апплет, встроенный в веб-страницу, будет выводить сообщение об ошибке:

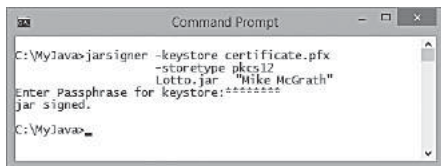


Сертификаты, предназначенные для подписи кода, выпускаются доверенными центрами сертификации (Certificate Authority или CA), такими как VeriSign, DigiCert или GlobalSign. Сертификаты выпускаются после проверки центрами авторства разработчиков. Таким образом, пользователи могут быть уверены, что программа, подписанная сертификатом, получена из надежного источника. Центры сертификации обеспечивают разработчикам персональный файл, содержащий закрытый ключ, с помощью которого разработчик подписывает JAR-файлы, а в пакете JDK для этих целей существует утилита **jarsigner**.

1. Откройте командную строку (терминальное окно) и перейдите в каталог, где размещен файл *Lotto.jar*.
2. Введите команду **jarsigner**, которая идентифицирует файл ключа, тип хранения, файл JAR-архива и имя разработчика; затем нажмите клавишу **Enter**.

```
jarsigner -keystore certificate.pfx
          -storetype pkcs12
          Lotto.jar "Mike McGrath"
```

3. Введите парольную фразу, которую вы отправите в центр сертификации для своего сертификата. Далее вы увидите ответ, подтверждающий подпись JAR-файла.



Внимание

В пакете JDK существует утилита **keytool**, при помощи которой можно создать самоподписываемые сертификаты, но они больше не поддерживаются для JAR-файлов.



certificate.pfx

Данный файл не включен в архив файлов примеров к этой книге, поскольку вам нужен будет собственный сертификат.

Совет

Проверить, подписан ли JAR-файл, вы можете с помощью команды **jarsigner -verify Lotto.jar**.

Использование технологии Web Start

Технология Java Web Start позволяет пользователям запускать Java-приложения простым нажатием на гиперссылку, находящуюся на веб-странице. Ссылка указывает на файл JNLP (протокол Java Network Launching Protocol), который предоставляет информацию о приложении в формате XML. Выполните следующие шаги, чтобы запустить приложение **Lotto** при помощи ссылки с веб-страницы.



Lotto.jnlp

1. Откройте простой текстовый редактор, например Блокнот (Notepad), и скопируйте следующее содержимое в формате XML для создания файла протокола JNLP.

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<jnlp spec = "1.0+" codebase = "file:///C:/MyJava/" href = "Lotto.jnlp" >
<information>
  <title>Lotto Application</title>
  <vendor>Java in easy steps</vendor>
  <homepage href = "http://www.ineasysteps.com" />
  <offline-allowed />
</information><security><all-permissions/></security>
<resources>
  <jar href = "Lotto.jar" />
  <j2se version = "1.6+" href = "http://java.sun.com/products/autodl/j2se" />
</resources>
<application-desc main-class = "Lotto" />
</jnlp>
```

2. Сохраните новый файл JNLP под именем *Lotto.jnlp* рядом с файлом *Lotto.jar*.
3. В тот же самый каталог добавьте файл *Lotto.html* и включите в его содержимое следующую гиперссылку.

```
<a href = "Lotto.jnlp" >Запустить приложение Lotto</a>
```

Совет



Помните, что в файле JNLP нужно установить параметр безопасности **all-permissions**, чтобы привести файл в соответствие с манифестом.

- Откройте файл *Lotto.html* в вашем веб-браузере и щелкните по ссылке для того чтобы запустить приложение **Lotto**, используя технологию Web Start.



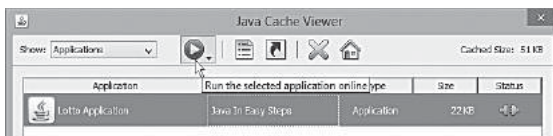
Lotto.html



Во время загрузки на короткое время появляется заставка Java Web Start, а потом загружается ваше приложение.

Для того чтобы задействовать технологию Web Start, необходимо сначала сконфигурировать веб-сервер таким образом, чтобы он поддерживал формат JNLP. Для этого нужно добавить в описание MIME-типа следующую строку: **application/x-java-jnlp-file JNLP**. Кроме того, нужно задать атрибут **codebase** с указанием расположения вашего приложения на сервере, например **codebase= "http://www.myserver/java-apps/"**. После загрузки необходимых файлов JNLP, JAR и HTML в это место, Java Web Start может загружать приложение по ссылке, как указывалось выше.

Во время первой загрузки приложения информация из JNLP-файла автоматически сохраняется в локальном кэше Java. Она может быть использована для последующей загрузки приложения, а также для создания ярлыков на рабочем столе для запуска приложений.

- Чтобы открыть кэш Java на компьютере под управлением операционной системы Windows, щелкните по иконке **Java** в окне панели управления, а затем по кнопке **View** (Показать) на вкладке **General** (Общие) в открывшемся диалоговом окне.



- Выберите программу **Lotto** и щелкните по иконке запуска , для того чтобы открыть приложение.
- Нажмите кнопку **Install shortcuts** (Установить ярлык)  для добавления на рабочий стол ярлыка. После этого попробуйте запустить приложение, используя созданный ярлык.



Внимание

В атрибуте **codebase** должен быть точно указан каталог с файлами на сервере, иначе Web Start не сможет найти их и программа не запустится.

На заметку

Кэш Java можно открыть из командной строки как в среде Windows, так и Linux с помощью команды **javaws -viewer**.

Создание апплетов

В качестве альтернативы Java-приложениям, которые выполняются при помощи среды Java Runtime Environment, установленной в системе, программу можно разрабатывать в виде встроенных в веб-страницы Java-апплетов, которые исполняются Java-плагином в браузере.

Java-апплеты отличаются от приложений Java двумя основными аспектами:

- апплетам не нужно отдельное окно — они располагаются внутри веб-страницы при помощи кода HTML;
- апплеты не содержат метод `main` — вместо него они используют метод `init()`.

Чтобы понять разницу между апплетом и приложением, выполните следующие шаги, преобразуя приложение *Lotto.java* в апплет.

1. Измените имя класса с **Lotto** на **LottoApplet**, а затем в объявлении класса замените **JFrame** на **JApplet**.
2. Поменяйте конструктор на метод апплета `init()` путем замены `public Lotto()` на `public void init()`.
3. Удалите все операторы, определяющие свойство окна в бывшем конструкторе, убрав следующие строки.

```
super( "Приложение Lotto" ) ;

setSize( 260 , 200 ) ;

setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE ) ;

setVisible( true ) ;
```

4. Добавьте возможность задания цвета фона апплета при помощи шестнадцатеричного параметра, устанавливаемого в коде HTML.

```
String bgStr = getParameter( "BgColor" ) ;
int bgHex = Integer.parseInt( bgStr , 16 ) ;
pnl.setBackground( new java.awt.Color( bgHex ) ) ;
```

5. Удалите метод `main`, убрав следующий блок кода.

```
public static void main( String[] args )
{ Lotto lotto = new Lotto() ; }
```

6. Сохраните полученный файл под именем *LottoApplet.java* — его содержимое должно выглядеть следующим образом.

```
import javax.swing.* ;
import java.awt.event.* ;
```

Внимание



Объявления `init` в апплете должны включать ключевое слово `void` — апплет не может возвращать значение.

```

public class LottoApplet extends JApplet implements ActionListener
{
    // Компоненты.
    ClassLoader ldr = this.getClass().getClassLoader() ;
    java.net.URL iconURL = ldr.getResource( "Lotto.png" ) ;
    ImageIcon icon = new ImageIcon( iconURL ) ;
    JLabel img = new JLabel( icon ) ;
    JTextField txt = new JTextField( "" , 18 ) ;
    JButton btn = new JButton( "Показать счастливые номера" ) ;
    JPanel pnl = new JPanel() ;
    // Точка входа апплета.
    public void init()
    {
        pnl.add( img ) ; pnl.add( txt ) ; pnl.add( btn ) ;
        btn.addActionListener( this ) ;
        String bgStr = getParameter( "BgColor" ) ;
        int bgHex = Integer.parseInt( bgStr , 16 ) ;
        pnl.setBackground( new java.awt.Color( bgHex ) ) ;
        add( pnl ) ;
    }
    // Обработчик событий.
    public void actionPerformed((ActionEvent event) )
    {
        if ( event.getSource() == btn )
        {
            int[] nums = new int[50] ; String str = "" ;
            for ( int i = 1 ; i < 50 ; i++ ) { nums[ i ] = i ; }
            for ( int i = 1 ; i < 50 ; i++ )
            {
                int r = (int) Math.ceil( 49 * Math.random() ) + 1 ;
                int temp=nums[i]; nums[i]=nums[r]; nums[r]=temp;
            }

            for ( int i = 1 ; i < 7 ; i++ )
            { str += " " + Integer.toString( nums[ i ] ) + " " ; }
            txt.setText( str ) ;
        }
    }
}

```



LottoApplet.java

На заметку



Компоненты `LottoApplet`, а также обработчик событий остались теми же самыми.

Встраивание апплетов в код веб-страницы

Для того чтобы встроить Java-апплет в содержимое веб-страницы, в исходный код этой страницы нужно добавить некоторый дополнительный код HTML, определяющий пространство, в котором будет запускаться апплет.

В начале при помощи HTML-тега `<object>` сам Java-апплет определяется как объект при помощи назначения атрибуту **type** значения **"application/x-java-applet"**. После этого размер апплета на веб-странице определяется заданием числовых значений (в пикселях) атрибутам **width** и **height**.

Затем параметры, используемые апплетом, указываются в виде атрибутов **name** и **value** тегов `<param>`, входящих внутрь тегов `<object>` `</object>`. Обязательным должен быть тег `<param>`, который присваивает атрибуту **name** значение **"code"**, а атрибуту **value** — имя файла Java-апплета. Элемент `<object>` может также содержать всплывающее сообщение, отображающееся на веб-странице в случае невозможности запустить апплет.

Каталог *bin* пакета JDK содержит инструмент **appletviewer**, который можно использовать для предварительного просмотра апплета из указанного документа HTML.

1. Откройте простой текстовый редактор, например Блокнот (Notepad), затем скопируйте следующее содержимое для создания файла апплета.



LottoApplet.html

```
<!DOCTYPE HTML>

<html>
  <head>
    <meta charset = "UTF-8" >
    <title>Lotto Applet Host</title>
  </head>
  <body>
    <object type = "application/x-java-applet"
      width = "260" height = "160" >
      <param name = "code" value = "LottoApplet.class" >
      <param name = "BgColor" value = "FFFF00" > [ Java Applet
        - Requires Java Plugin ]
    </object>
  </body>
</html>
```

2. Сохраните содержимое в файле с именем *LottoApplet.html* рядом с файлами программ.
3. В командной строке для создания файла *LottoApplet.class* скомпилируйте приведенную выше программу *LottoApplet.java*, а затем используйте инструмент **appletviewer** для предварительного просмотра апплета.

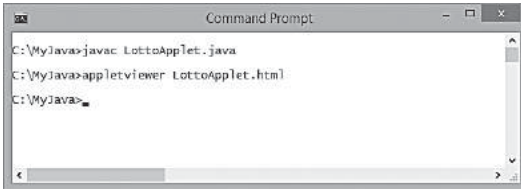


LottoApplet.class



Внимание

Параметр `code` по-прежнему необходим, чтобы определить класс, который содержит точку входа в программу.



Фон апплета *LottoApplet.class* установлен параметром **BgColor** (**FFFF00** — Желтый), а в остальном он выглядит так же, как и приложение **Lotto**. Так же, как и в случае с Java-приложениями, все файлы Java-апплетов рекомендуется собирать в один JAR-архив, чтобы, во-первых, случайно не потерять какой-нибудь ресурсный файл, а, во-вторых, уменьшить общий размер файлов. Для этого следует добавлять еще один HTML-элемент **<param>** с атрибутом **name**, имеющим значение **"archive"**, и атрибутом **value**, в котором указывается имя JAR-архива.

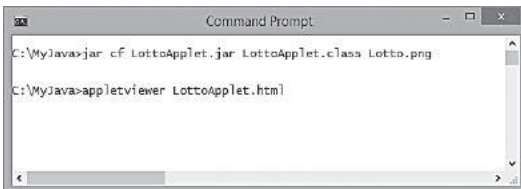
4. Добавьте следующий тег внутри элемента **<object>** файла *LottoApplet.html* и заново сохраните файл.

```
<param name = "archive" value = "LottoApplet.jar" >
```

5. В том же каталоге, что и файл HTML, создайте архив *LottoApplet.jar*, содержащий файлы *Lotto.png* и *LottoApplet.class*, а затем просмотрите апплет с помощью инструмента **appletviewer**, чтобы убедиться, что он запускается так же, как и раньше.



LottoApplet.jar

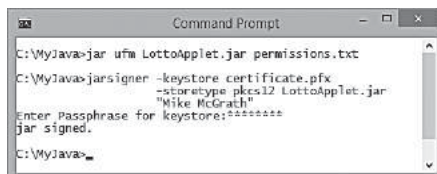


Развертывание апплетов

Поскольку Java-апплеты предназначены для широкого распространения через Интернет, для JAR-файлов требуется задавать определенные разрешения, а также дополнять их цифровой подписью.

1. Наберите команду `jar ufm LottoApplet.jar permissions.txt`, нажмите клавишу **Enter** для добавления атрибута `permissions` в манифест.
2. Введите команду `jarsigner`, задающую файл ключа, тип хранения, имя JAR-файла и имя разработчика; затем нажмите клавишу **Enter**.

```
jarsigner -keystore certificate.pfx
          -storetype pkcs12
          LottoApplet.jar "Mike McGrath"
```



При просмотре апплетов на различных веб-браузерах и платформах могут возникать различные неожиданные проблемы, поэтому хорошей практикой является тестирование апплета перед распространением на множестве различных окружений.

3. Скопируйте HTML-код апплета в желаемое место на веб-странице, которую нужно распространить.
4. Настройте все необходимые атрибуты HTML для размещения страницы — для **LottoApplet** установите шестнадцатеричный параметр **BgColor**, соответствующий цвету фона на странице.
5. Сохраните настроенную веб-страницу на компьютере и просмотрите изменения в вашем браузере.

На заметку



Процесс задания разрешений и подписи JAR-файлов ранее в этой главе.

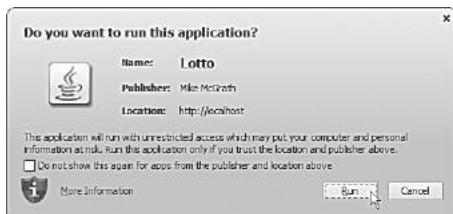


PlayLotto.html

6. Если вы удовлетворены видом апплета, загрузите веб-страницу и JAR-файл на ваш веб-сервер, затем протестируйте его производительность в различных окружениях.



Если в веб-браузере не установлен Java-плагин или Java не включена в настройках, то на странице вместо апплета может отображаться определенное всплывающее сообщение. В тех случаях, когда опция Java включена в браузере, апплет предоставляет на веб-странице ту же самую функциональность, что и Java-приложение:



На заметку



Текст, заключенный между HTML-тегами `<object>` и `</object>`, является сообщением, которое будет отображаться в браузере в случае неудачной загрузки апплета.

Совет



Цвет фона апплета `LottoApplet` настроен здесь таким образом, чтобы соответствовать фону таблицы, в которой он появляется на веб-странице.

Заключение

- Java-программы могут быть распространены в виде отдельно запускающихся настольных приложений, которые работают на соответствующих версиях Java Runtime Environment.
- Альтернативным способом распространения Java-программ являются встроенные в веб-страницы апплеты, которые работают с соответствующими версиями Java-плагинов, встроенных в браузер.
- Файлы приложений могут распространяться для дальнейшего исполнения на других системах с использованием подходящего интерпретатора **java**.
- Упаковка всех необходимых программ в один JAR-архив позволяет избегать случайной потери ресурсных файлов.
- JAR-архивы, которые должны быть распространены в виде настольных приложений, должны включать указание входной точки программы в файле манифеста.
- Файлы манифеста JAR-архивов должны содержать атрибут **Permissions**.
- JAR-приложения могут быть запущены из командной строки при помощи команды **java -jar** либо двойным нажатием по соответствующей иконке.
- Для широкого распространения Java-программ необходимо добавлять к ним цифровые подписи.
- Технология Java Web Start позволяет запускать приложения с помощью гиперссылки на веб-страницу.
- Информация о Java-приложении хранится в формате XML в файле JNLP.
- Перед тем как распространять Java-приложение через Интернет, веб-серверы должны быть сконфигурированы для поддержки технологии Web Start.
- Используя Java Cache Viewer, можно повторно запускать приложения при помощи технологии Web Start, а также создавать ярлыки на рабочем столе для последующего запуска.
- Java-апплеты не требуют компонентов окон и вместо метода **main** используют метод **init**.

Параметры Java-апплета можно установить в коде HTML, тем самым настраивая апплет таким образом, чтобы он подходящим образом вписывался в окружающую его веб-страницу.

Предметный указатель

Г

GUI, 136

Ж

JAR, 139, 176

Java API, 82

Java-плагин, 172

Л

Lambda-выражения, 128

А

апплет, 12, 172, 182

аргумент, 64, 84

архив, 176

атрибут, 108

Б

бесконечный цикл, 54

библиотека Java, 82

бит, 42

блок кода, 52

булев, 34

В

внешний цикл, 58

внутренний цикл, 58

всплывающая подсказка, 140

выпадающий список, 144

выход из цикла, 58, 59

Г

главный метод, 51

Д

декремент, 28, 53

деление по модулю, 28

длина массива, 76

И

индекс, 66

инициализатор, 55

инициализация, 35, 48

инкапсуляция, 112

инкремент, 28, 53

интерпретатор, 11

исключение, 78

итерация, 52

К

класс, 82

ключевое слово, 50

кнопка, 138

код, 31

командная строка, 68

комментарии, 23

компилятор, 10, 16, 64

компиляция, 16, 64

конкатенация, 29

константа, 22, 84

конструктор, 110

контейнер, 136

Л

литерал, 40, 90

логическое значение, 32, 46

локальный кэш Java, 181

М

массив, 66
менеджер шаблона, 136
метка, 60, 140
метод, 64
модификатор, 54

О

область видимости, 102
обработка событий, 154
общедоступный метод, 113
объявление, 35, 48, 66
окно, 136
округление, 86
операнд, 28
оператор, 15, 46
операция, 28
ошибка компиляции, 64

П

пакет, 82
перегрузка метода, 101
переключатель, 50
переменная, 18, 19
переменная-счетчик, 52
побитовый оператор, 42
подкласс, 106
подстрока, 94
полоса прокрутки, 142
постфиксный, 53
префиксный, 53
приложение, 172
приоритет, 38
присваивание, 57, 66
проверочное выражение, 56

Р

равенство, 30
развертывание программ, 172
размер массива, 66

С

свойство, 66
сертификат, 179
символ, 32
символьный, 64
синтаксис, 36
случайное число, 88
слушатель событий, 154
сообщение об ошибке, 31
список, 66
ссылка, 180
строка, 28
строковый тип, 64
суперкласс, 106
счетчик, 72
счетчик цикла, 32

Т

тег, 184
текстовое поле, 142
технология Java Web Start, 180
тип данных, 20, 64
точечная запись, 66, 82

У

унарный оператор, 34
условное ветвление, 32

Ф

файл манифеста, 177
флажок, 144

Ц

цикл, 32, 52, 72
цифровая подпись, 179

Ч

числовой код ASCII, 64
число с плавающей точкой, 64

Э

экземпляр класса, 90, 108
элемент массива, 66

Все права защищены. Книга или любая ее часть не может быть скопирована, воспроизведена в электронной или механической форме, в виде фотокопии, записи в память ЭВМ, репродукции или каким-либо иным способом, а также использована в любой информационной системе без получения разрешения от издателя. Копирование, воспроизведение и иное использование книги или ее части без согласия издателя является незаконным и влечет уголовную, административную и гражданскую ответственность.

Производственно-практическое издание
ПРОГРАММИРОВАНИЕ ДЛЯ НАЧИНАЮЩИХ

МакГрат Майк
ПРОГРАММИРОВАНИЕ НА JAVA ДЛЯ НАЧИНАЮЩИХ
(орыс тілінде)

Директор редакции *Е. Капъёв*
Ответственный редактор *В. Обручев*
Художественный редактор *В. Брагина*

В оформлении обложки использована иллюстрация:
Myimagine / Shutterstock.com
Используется по лицензии от Shutterstock.com

ООО «Издательство «Э»
123308, Москва, ул. Зорге, д. 1. Тел. 8 (495) 411-68-86.
Өндіруші: «Э» АҚБ Баспасы, 123308, Мәскеу, Ресей, Зорге көшесі, 1 үй.
Тел. 8 (495) 411-68-86.
Тауар белгісі: «Э»
Қазақстан Республикасында дистрибьютор және өнімі бойынша арыз-талаптарды қабылдаушының
өкілі «РДЦ-Алматы» ЖШС, Алматы қ., Домбровский көш., 3-а, литер Б, офис 1.
Тел.: 8 (727) 251-59-89/90/91/92, факс: 8 (727) 251 58 12 вн. 107.
Өнімнің жарамдылық мерзімі шектелмеген.
Сертификация туралы ақпарат сайтта Өндіруші «Э»
Сведения о подтверждении соответствия издания согласно законодательству РФ
о техническом регулировании можно получить на сайте Издательства «Э»
Өндірген мемлекет: Ресей
Сертификация қарастырылмаған

Подписано в печать 27.01.2016. Формат 84x108¹/₁₆.
Печать офсетная. Усл. печ. л. 20, 16.
Тираж экз. Заказ

ISBN 978-5-699-85743-2



9 785699 857432 >



В электронном виде книги издательства вы можете
купить на www.litres.ru

ЛитРес:
одна книга до книг



Начни программировать прямо сейчас!

САМОЕ ВАЖНОЕ:

- ПЕРЕМЕННЫЕ
- ОПЕРАТОРЫ
- ЦИКЛЫ
- МАССИВЫ
- ДАННЫЕ
- КЛАССЫ
- ОБЪЕКТЫ
- ИНТЕРФЕЙСЫ
- РАЗВЕРТЫВАНИЕ ПРОГРАММ
- АППЛЕТЫ

Книга **«ПРОГРАММИРОВАНИЕ НА JAVA ДЛЯ НАЧИНАЮЩИХ»** является исчерпывающим руководством для того, чтобы научиться программировать на языке Java.

В этой книге с помощью примеров программ и иллюстраций, показывающих результаты работы кода, разбираются все ключевые аспекты языка Java. Установив свободно распространяемый Java Development Kit, вы с первого же дня сможете создавать свои собственные исполняемые программы!

Познакомившись с основами языка, вы научитесь использовать основные возможности Java, необходимые для изучения операторов, операций обработки данных, импорта, создания интерфейсов и изготовления апплетов с использованием свободно распространяемого исходного кода. Вы сможете использовать лямбда-выражения, библиотеку `java.time` и другие ключевые новшества версии Java 8. В обучении вам помогут готовые примеры, которые можно скачать по адресу <http://eksmo.ru/upload/java5thsrc-rus.zip>

Книга **«ПРОГРАММИРОВАНИЕ НА JAVA ДЛЯ НАЧИНАЮЩИХ»** идеально подойдет начинающим программистам, стремящимся освоить язык Java.

ЧТО ВНУТРИ?



Эти значки сделают обучение еще проще. Каждый раз, когда при чтении книги вы встречаете один из этих значков, знайте — мы приготовили для вас какой-то полезный совет, придающий остроту процессу обучения, выделили нечто необходимое для запоминания или выносим предостережение держаться подальше от возможных проблем.



«Учебник предельно практичен и очень легко написан. Начинающие смогут изучать все конструкции языка экспериментальным путем на примерах, ввод которых займет не более пары минут. Ничего лишнего. Рекомендую для быстрого старта. Для опытного программиста в тексте выделены важные фрагменты, можно быстро просмотреть отличия Java от других языков программирования».

ISBN 978-5-699-85743-2



9 785699 857432



Вячеслав Рожков,
Группа компаний ICL