Николай Прохоренок

Основы Java

- Базовый синтаксис языка Java
- Объектно-ориентированное программирование
- Работа с файлами и каталогами
- Stream API
- Функциональные интерфейсы
- Лямбда-выражения
- Работа с базой данных MySQL
- Получение данных из Интернета
- Примеры и советы из практики

Материалы на www.bhv.ru

bhv®

Николай Прохоренок

Основы Java

Санкт-Петербург
«БХВ-Петербург»
2017

УДК 004.438 Java
ББК 32.973.26-018.1
П84

Прохоренок Н. А.

П84 Основы Java. — СПб.: БХВ-Петербург, 2017. — 704 с.: ил.

ISBN 978-5-9775-3785-8

Описан базовый синтаксис языка Java: типы данных, операторы, условия, циклы, регулярные выражения, объектно-ориентированное программирование. Рассмотрены основные классы стандартной библиотеки, получение данных из Интернета, работа с базой данных MySQL. Приводится описание большинства нововведений: Date API, Stream API, лямбда-выражения, ссылки на методы, функциональные интерфейсы и др. Книга содержит большое количество практических примеров, помогающих начать программировать на языке Java самостоятельно. Весь материал тщательно подобран, хорошо структурирован и компактно изложен, что позволяет использовать книгу как удобный справочник. Электронный архив с примерами находится на сайте издательства.

Для программистов

УДК 004.438 Java
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Екатерина Капалыгина*
Редактор	*Григорий Добин*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн обложки	*Марины Дамбиевой*

Подписано в печать 31.01.17.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 56,76.
Тираж 1500 экз. Заказ №
"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

Первая Академическая типография "Наука"
199034, Санкт-Петербург, 9 линия, 12/28

ISBN 978-5-9775-3785-8

Оглавление

Введение

Добро пожаловать в мир Java!

Java — это объектно-ориентированный язык программирования высокого уровня, предназначенный для самого широкого круга задач. С его помощью можно обрабатывать различные данные, создавать изображения, работать с базами данных, разрабатывать Web-сайты, мобильные приложения и приложения с графическим интерфейсом. Java также язык кроссплатформенный, позволяющий создавать программы, которые будут работать во всех операционных системах. В этой книге мы рассмотрим основы языка Java SE 8 (*SE* — Standard Edition) применительно к операционной системе Windows.

Язык Java часто путают с языком JavaScript. Помните — это абсолютно разные языки и по синтаксису, и по назначению. JavaScript работает в основном в Web-браузере, тогда как Java является универсальным языком, хотя он также может работать в Web-браузере (в виде апплета или приложения JavaFX).

Синтаксис языка Java очень похож на синтаксис языков C++ и C#. Если вы знакомы с этими языками, то изучить язык Java не составит вам особого труда. Однако если вы этих языков не знаете или вообще никогда не программировали, то у вас могут возникнуть некоторые сложности. Дело в том, что язык Java, как уже отмечалось, представляет собой язык объектно-ориентированный, и, в отличие от C++, объектно-ориентированный стиль программирования (ООП-стиль) является для него обязательным. Это означает, что с самой первой программы вы погрузитесь в мир ООП, а мне, как автору книги, придется много раз решать проблему, что было раньше: курица или яйцо. Ведь объяснить начинающему программисту преимущества ООП-стиля программирования, когда он не знает, что такое «переменная», очень сложно. И с самой первой программы нам необходимо будет разъяснить, что такое «класс» и «модификатор доступа», что такое «метод» и чем отличается обычный метод от статического, что такое «переменная» и «массив», что такое «поток вывода» и др. Вот всему этому и посвящена моя книга. Поэтому если вы не знакомы с языками C++ и C#, то в обязательном порядке вам следует прочитать книгу несколько раз, — только в этом случае сможете изучить язык Java.

Если вам ранее доводилось читать мои книги по PHP, JavaScript, Perl, Python или C++, то изучить язык Java вам будет гораздо проще, так как вы уже знаете структу-

ру этих книг. Их структура в большинстве случаев совпадает, но не на все 100 процентов, так как языки эти все-таки разные и обеспечивают выполнение различных задач. Если же структура моих книг вам пока не знакома, то ничего страшного, сейчас я ее вкратце представлю.

- *Глава 1* является вводной. Мы установим необходимое программное обеспечение, настроим среду разработки, скомпилируем и запустим первую программу — как из командной строки, так и из редактора Eclipse. Кроме того, мы вкратце разберемся со структурой программы, а также научимся выводить результат работы программы и получать данные от пользователя.
- В *главе 2* мы познакомимся с переменными и типами данных в языке Java, а в *главе 3* рассмотрим различные операторы, предназначенные для выполнения определенных действий с данными. Кроме того, в этой главе мы изучим операторы ветвления и циклы, позволяющие изменить порядок выполнения программы.
- *Глава 4* полностью посвящена работе с числами. Вы узнаете, какие числовые типы данных поддерживает язык Java, научитесь применять различные методы, генерировать случайные числа и др.
- *Глава 5* познакомит вас с массивами в языке Java. Вы научитесь создавать как одномерные массивы, так и многомерные, перебирать элементы массива, сортировать, выполнять поиск значений, преобразовывать массив в строку и др.
- *Глава 6* полностью посвящена работе с символами и строками. В этой главе вы также научитесь работать с различными кодировками, настраивать локаль, выполнять форматирование строки и осуществлять поиск внутри строки. А в *главе 7* мы рассмотрим регулярные выражения, которые позволят осуществить сложный поиск в строке в соответствии с шаблоном.
- *Главы 8* и *9* познакомят вас с двумя способами работы с датой и временем. Первый способ доступен с самой первой версии Java, а второй был добавлен в 8-й версии.
- В *главе 10* вы научитесь создавать пользовательские методы, а в *главе 11* познакомитесь с объектно-ориентированным программированием, позволяющим использовать код многократно.
- *Глава 12* познакомит вас с интерфейсами, а *глава 13* — с обобщенными типами данных.
- *Главы 14* и *15* посвящены описанию каркаса коллекций. В этот каркас входят обобщенные классы, реализующие различные структуры данных: динамические списки, очереди, множества и словари.
- В *главе 16* вы познакомитесь со способом организации больших программ в виде пакетов, а также научитесь создавать JAR-архивы. Если вас очень интересует способ запуска Java-программ двойным щелчком на значке файла, то именно в этой главе вы найдете ответ на свой вопрос.
- В *главе 17* мы рассмотрим способы поиска ошибок в программе и научимся отлаживать программу в редакторе Eclipse.

- *Главы 18*, *19* и *20* научат вас работать с файлами и каталогами, читать и писать файлы в различных форматах.
- *Глава 21* познакомит с потоками данных (Stream API), введенных в 8-й версии языка.
- В *главе 22* мы рассмотрим способы получения данных из сети Интернет, а в *главе 23* научимся работать с базой данных MySQL.
- И, наконец, *глава 24* познакомит вас с многопоточными приложениями, позволяющими значительно повысить производительность программы за счет параллельного выполнения задач несколькими потоками управления.

Все листинги из этой книги вы найдете в файле Listings.doc, электронный архив с которым можно загрузить с FTP-сервера издательства «БХВ-Петербург» по ссылке: **ftp://ftp.bhv.ru/9785977537858.zip** или со страницы книги на сайте **www.bhv.ru** (см. *приложение*).

Желаю приятного прочтения и надеюсь, что эта книга выведет вас на верный путь в мире профессионального программирования.

ГЛАВА 1

Первые шаги

Прежде чем мы начнем рассматривать синтаксис языка, необходимо сделать два замечания. Во-первых, не забывайте, что книги по программированию нужно не только читать, но и выполнять все приведенные в них примеры, а также экспериментировать, что-либо в этих примерах изменяя. Поэтому, если вы удобно устроились на диване и настроились просто читать, у вас практически нет шансов изучить язык! Во-вторых, помните, что прочитать эту книгу один раз недостаточно — ее вы должны выучить наизусть! Это же основы языка! Сколько на это уйдет времени, зависит от ваших способностей и желания. Как минимум, вы должны знать структуру книги.

Чем больше вы будете делать самостоятельно, тем большему научитесь. Обычно после первого прочтения многое непонятно, после второго прочтения — некоторые вещи становятся понятнее, после третьего — еще лучше, ну, а после *N*-го прочтения — не понимаешь, как могло быть что-то непонятно после первого прочтения. Повторение — мать учения. Наберитесь терпения, и вперед!

Итак, приступим к изучению языка Java и начнем с установки необходимых программ.

1.1. Установка Java SE Development Kit (JDK)

Для изучения языка Java необходимо установить на компьютер комплект разработчика Java Development Kit (сокращенно JDK), который включает компилятор, стандартные библиотеки классов, различные утилиты и исполнительную среду Java Runtime Environment (сокращенно JRE). Для этого переходим на страницу:

http://www.oracle.com/technetwork/java/javase/downloads/index.html

и скачиваем дистрибутив, соответствующий операционной системе вашего компьютера. Поскольку в этой книге мы станем рассматривать основы языка применительно к 64-битной операционной системе Windows 8, скачать вам следует файл jdk-8u77-windows-x64.exe. Соглашаемся с лицензионным соглашением, скачиваем файл и запускаем его. Процесс установки полностью автоматизирован и не нуждается в особых комментариях. Во всех случаях соглашаемся с настройками по умолчанию.

После установки в каталоге C:\Program Files\Java будут созданы две папки:

- jre1.8.0_77 — содержит файлы исполнительной среды Java Runtime Environment (JRE). Соответственно, в папке C:\Program Files\Java\jre1.8.0_77\bin расположены исполняемые файлы и библиотеки, которых достаточно для выполнения программ, написанных на языке Java. Если вы ранее пользовались такими программами, как OpenOffice.org, то JRE должен быть знаком;
- jdk1.8.0_77 — содержит файлы комплекта разработчика Java Development Kit (JDK). Соответственно, в папке C:\Program Files\Java\jdk1.8.0_77\bin расположен компилятор (javac) и различные утилиты.

Чтобы проверить правильность установки, запускаем приложение **Командная строка** (рис. 1.1) и выполняем следующую команду:

```
java -version
```

Результат должен быть примерно следующим:

```
C:\Users\Unicross>java -version
java version "1.8.0_77"
Java(TM) SE Runtime Environment (build 1.8.0_77-b03)
Java HotSpot(TM) 64-Bit Server VM (build 25.77-b03, mixed mode)
```

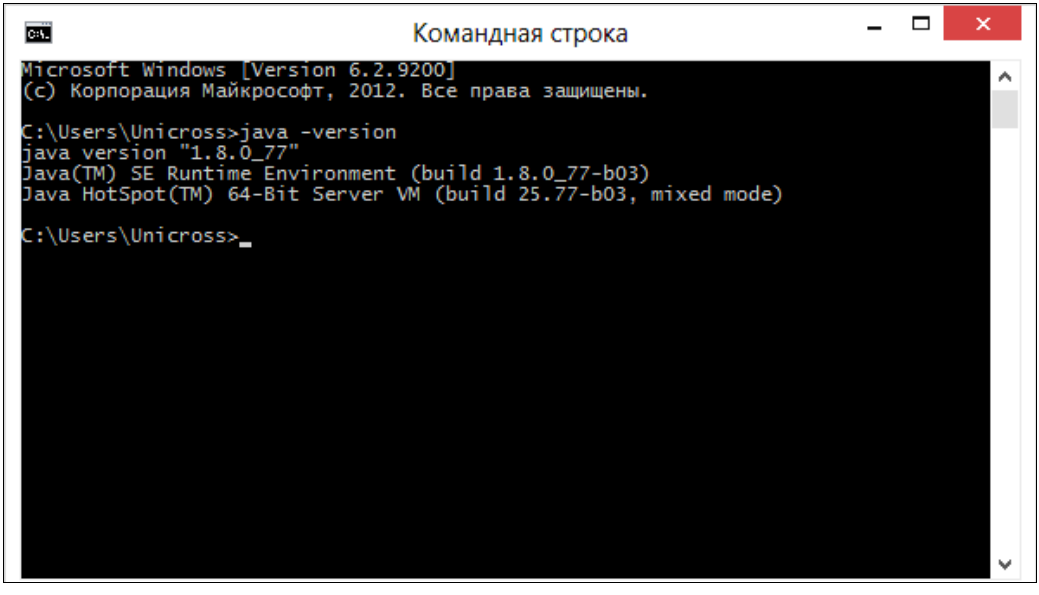

Рис. 1.1. Приложение **Командная строка**

Вполне возможно, что вы никогда не пользовались командной строкой и не знаете, как запустить это приложение. Давайте рассмотрим некоторые способы его запуска в Windows 8:

- через поиск находим приложение **Командная строка**;
- нажимаем комбинацию клавиш <Windows>+<R>. В открывшемся окне вводим cmd и нажимаем кнопку **OK**;

- находим файл cmd.exe в папке C:\Windows\System32;
- в Проводнике щелкаем правой кнопкой мыши на свободном месте списка файлов, удерживая при этом нажатой клавишу <Shift>, и из контекстного меню выбираем пункт **Открыть окно команд**.

Если результат выглядит так:

```
C:\Users\Unicross>java -version
"java" не является внутренней или внешней
командой, исполняемой программой или пакетным файлом.
```

то необходимо добавить путь к папке C:\Program Files\Java\jre1.8.0_77\bin в системную переменную PATH. Программа установки обычно прописывает путь к файлу java.exe автоматически, но вполне возможно, что вы изменили настройки в процессе установки и в результате получили эту ошибку.

Чтобы изменить системную переменную в Windows 8, переходим в **Параметры | Панель управления | Система и безопасность | Система | Дополнительные параметры системы**. В результате откроется окно **Свойства системы** (рис. 1.2). На вкладке **Дополнительно** нажимаем кнопку **Переменные среды**. В открывшемся окне (рис. 1.3) в списке **Системные переменные** выделяем строку с переменной **Path** и нажимаем кнопку **Изменить**. В открывшемся окне (рис. 1.4) изменяем значение в поле **Значение переменной** — для этого переходим в конец существующей

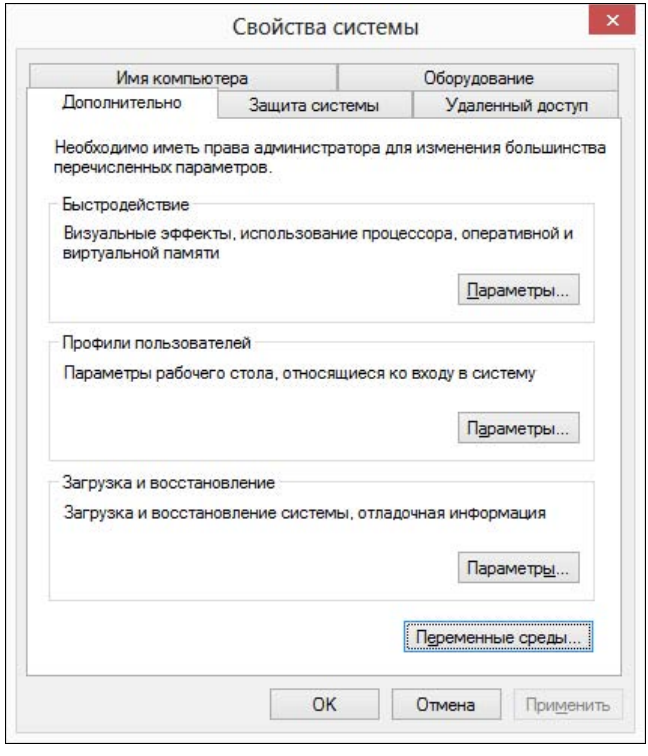

Рис. 1.2. Окно **Свойства системы**

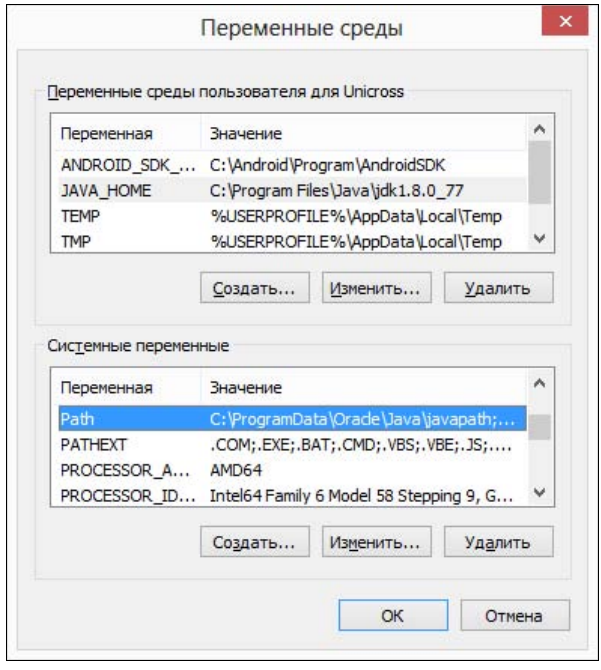

Рис. 1.3. Окно **Переменные среды**

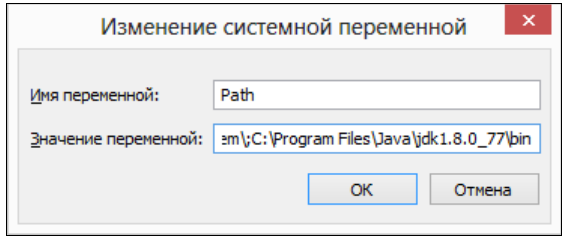

Рис. 1.4. Окно **Изменение системной переменной**

строки, ставим точку с запятой, а затем вводим путь к папке C:\Program Files\Java\jre1.8.0_77\bin:

```
<Текущее значение>;C:\Program Files\Java\jre1.8.0_77\bin
```

Сохраняем изменения и повторно проверяем установку.

> ***Внимание!***
>
> Случайно не удалите существующее значение переменной `PATH`, иначе другие приложения перестанут запускаться.

Помимо добавления пути к папке C:\Program Files\Java\jre1.8.0_77\bin в переменную `PATH`, во избежание проблем с настройками различных редакторов необходимо прописать переменную окружения `JAVA_HOME` и присвоить ей путь к папке с установленным JDK. Делается это также в окне **Переменные среды** (см. рис. 1.3), но в списке переменных для пользователя, хотя вы можете добавить ее и в список системных

переменных. Нажимаем в этом окне кнопку **Создать**, в поле **Имя переменной** вводим значение JAVA_HOME, а в поле **Значение переменной** — C:\Program Files\Java\jdk1.8.0_77. Нажимаем кнопку **OK**. Запускаем командную строку и проверяем установку переменной JAVA_HOME:

```
C:\Users\Unicross>set JAVA_HOME
JAVA_HOME=C:\Program Files\Java\jdk1.8.0_77
```

1.2. Первая программа

При изучении языков программирования принято начинать с программы, выводящей надпись **Hello, world!** в окно консоли. Не будем нарушать традицию и продемонстрируем, как это выглядит на языке Java (листинг 1.1).

Листинг 1.1. Первая программа

```
public class HelloWorld {
   public static void main(String[] args) {
      System.out.println("Hello, world!");
   }
}
```

> ***Электронный архив***
>
> Напомним, что все листинги из этой книги вы найдете в файле Listings.doc, электронный архив с которым можно загрузить с FTP-сервера издательства «БХВ-Петербург» по ссылке: **ftp://ftp.bhv.ru/9785977537858.zip** или со страницы книги на сайте **www.bhv.ru** (см. *приложение*).

На этом этапе мы пока не станем рассматривать структуру приведенной программы. Просто поверьте мне на слово, что эта программа выводит надпись **Hello, world!**. Сейчас мы попробуем скомпилировать программу и запустить ее на исполнение. Но для начала создадим следующую структуру каталогов:

```
book
Android
   Program
   Project
```

Папки book и Android лучше разместить в корне какого-либо диска. В моем случае это будет диск C:, следовательно, пути к содержимому папок: C:\book и C:\Android. При выборе диска следует учитывать размер свободного пространства. Android SDK и другие программы потребуют десятки гигабайтов, поэтому выберите подходящий диск с самого начала (если, конечно, будете программировать под Android, хотя в этой книге мы рассмотрим только основы языка Java). Можно создать папку и в любом другом месте, но в пути не должно быть русских букв и пробелов — только английские буквы, цифры, тире и подчеркивание. Остальных символов лучше избегать, если не хотите проблем с компиляцией и запуском программ.

В папке C:\book мы станем размещать наши тестовые программы и тренироваться при изучении работы с файлами и каталогами. Некоторые методы без проблем могут при неправильном действии удалить все дерево каталогов, поэтому для экспериментов мы воспользуемся отдельной папкой, а не папкой C:\Android, в которой у нас будет находиться все сокровенное, — обидно, если по случайности мы удалим все установленные программы и проекты.

Кстати, рекомендация не использовать русские буквы относится и к имени пользователя. Многие программы сохраняют настройки в каталоге C:\Users\<Имя пользователя>. В частности, по умолчанию настройки виртуальных устройств в Android SDK сохраняются в папке C:\Users\<Имя пользователя>\.android. В результате в пути окажутся русские буквы, которые могут стать причиной множества проблем.

Внутри папки Android у нас созданы две вложенные папки:

- Program — в эту папку мы будем устанавливать Android SDK, Android Studio, Eclipse и другие программы;
- Project — в этой папке мы станем сохранять проекты из различных редакторов, например, из Eclipse или Android Studio.

Для создания файла с программой можно воспользоваться любым текстовым редактором. Нам больше всего нравится редактор Notepad++. Итак, создаем текстовый файл с названием HelloWorld и расширением java. Обратите внимание на регистр символов в названии файла. Только так! Кроме того, это название в точности совпадает с именем класса, указанным после ключевого слова `class` внутри программы. Теперь несколько слов о расширении. Буквы в расширении набираются строчными. При сохранении файла обратите внимание, чтобы в конец не добавилось расширение txt. Программа Блокнот по умолчанию делает именно так, что приводит к ошибкам. Название файла должно быть HelloWorld.java, а не — HelloWorld.java.txt.

Набираем текст программы из листинга 1.1 и сохраняем файл в папке C:\book. Теперь необходимо скомпилировать программу. Для этого открываем командную строку, делаем текущей папку C:\book:

```
C:\Users\Unicross>cd C:\book
```

и запускаем процесс компиляции с помощью следующего кода:

```
C:\book>javac HelloWorld.java
```

Скорее всего, вы получите сообщение:

```
"javac" не является внутренней или внешней
командой, исполняемой программой или пакетным файлом.
```

Чтобы избежать этой проблемы, необходимо добавить путь к папке C:\Program Files\Java\jdk1.8.0_77\bin в конец текущего значения системной переменной `PATH` (через точку с запятой). Порядок изменения значения системной переменной `PATH` мы уже рассматривали в предыдущем разделе. Добавляем путь и заново запускаем компиляцию.

Вполне возможно, что вы получите следующее сообщение об ошибке:

```
C:\book>javac helloworld.java
helloworld.java:1: error: class HelloWorld is public, should be
declared in a file named HelloWorld.java
public class HelloWorld {
       ^
1 error
```

В этом случае регистр в имени файла не совпадает с регистром названия класса. Примерно такое же сообщение вы получите, если имя файла не будет совпадать с названием класса.

Если имя файла будет иметь расширение txt (например, HelloWorld.java.txt), то компилятор не сможет найти файл HelloWorld.java и выведет соответствующее сообщение:

```
C:\book>javac HelloWorld.java
javac: file not found: HelloWorld.java
Usage: javac <options> <source files>
use -help for a list of possible options
```

Возможны и другие сообщения об ошибках — например, если при наборе текста программы была допущена опечатка или вы забыли добавить какой-либо символ. Так, если вместо слова `public` набрать `publik`, то мы получим следующее сообщение об ошибке:

```
C:\book>javac HelloWorld.java
HelloWorld.java:1: error: class, interface, or enum expected
publik class HelloWorld {
^
1 error
```

В любом случае ошибку нужно найти и исправить.

Если компиляция прошла успешно, то никаких сообщений не появится, а отобразится приглашение для ввода следующей команды:

```
C:\book>javac HelloWorld.java


C:\book>
```

При этом в папке C:\book создастся одноименный файл с расширением class, который содержит специальный байт-код для виртуальной машины Java. Для запуска программы в командной строке набираем команду:

```
C:\book>java HelloWorld
```

Если все выполнено правильно, то получим приветствие:

```
Hello, world!
```

Обратите внимание, программе java.exe мы передаем имя файла с байт-кодом без расширения class. Если расширение указать, то получим следующее сообщение об ошибке:

```
C:\book>java HelloWorld.class
Error: Could not find or load main class HelloWorld.class
```

Итак, вначале мы создаем файл с расширением java, затем запускаем процесс компиляции с помощью программы javac.exe и получаем файл с расширением class, который можно запустить на выполнение с помощью программы java.exe. Вот так упрощенно выглядит процесс создания программ на языке Java. После компиляции мы получаем не EXE-файл, а всего лишь файл с байт-кодом. Зато этот байт-код может быть запущен на любой виртуальной машине Java вне зависимости от архитектуры компьютера — тем самым и обеспечивается кроссплатформенность программ на языке Java.

Прежде чем мы продолжим, необходимо сказать несколько слов о кодировке файла с программой. По умолчанию предполагается, что файл с программой сохранен в кодировке, используемой в системе по умолчанию. В русской версии Windows этой кодировкой является Windows-1251. Не следует полагаться на кодировку по умолчанию. Вместо этого при компиляции нужно указать кодировку явным образом с помощью флага `-encoding`:

```
C:\book>javac -encoding utf-8 HelloWorld.java
```

В этой инструкции мы указали компилятору, что файл с программой был сохранен в кодировке UTF-8. Мы будем пользоваться именно этой кодировкой, т. к. она позволяет хранить любые символы. При использовании редактора Notepad++ убедитесь, что в меню **Кодировки** установлен флажок у пункта **Кодировать в UTF-8 (без BOM)**. Если это не так, то в меню **Кодировки** следует выбрать пункт **Преобразовать в UTF-8 без BOM**.

1.3. Установка и настройка редактора Eclipse

При описании большинства языков программирования (таких как PHP, Python и др.) я обычно рекомендовал читателям использовать редактор Notepad++. Он очень удобен и хорошо подсвечивает код, написанный практически на всех языках, включая Java. Тем не менее, для создания программ на языке Java возможностей этого редактора недостаточно. Посмотрите на код листинга 1.1. Такое большое количество кода выводит в окно консоли всего лишь одну надпись. Теперь представьте, сколько придется написать текста, чтобы сделать что-то более сложное. Инструкции и названия методов в языке Java весьма длинные, и набирать их вручную очень долго. Если вы не владеете методом слепого десятипальцевого набора текста, а еще хуже — не владеете английским языком, то программирование на языке Java при использовании редактора Notepad++ может стать настоящим мучением.

К счастью, существуют специализированные редакторы, которые не только подсвечивают код программы, но и позволяют вывести список всех методов и свойств объекта, автоматически закончить слово, подчеркнуть код с ошибкой, а также скомпилировать проект всего лишь нажатием одной кнопки без необходимости использования командной строки. В этой книге мы для создания программ воспользуемся редактором Eclipse.

Для загрузки редактора Eclipse переходим на страницу:

http://www.eclipse.org/downloads/eclipse-packages/

и скачиваем архив с программой из раздела **Eclipse IDE for Java Developers**. В нашем случае файл архива носит название eclipse-java-neon-R-win32-x86_64.zip. Распаковываем скачанный архив в папку C:\Android\Program — в результате в этой папке будет создана папка eclipse, содержащая файлы редактора. Редактор не нуждается в установке, поэтому просто переходим в папку C:\Android\Program\eclipse и запускаем файл eclipse.exe.

При запуске редактор попросит указать папку с рабочим пространством (рис. 1.5). Указываем **C:\Android\Project** и нажимаем кнопку **OK**. Для создания проекта в меню **File** редактора выбираем пункт **New | Java Project**. В открывшемся окне (рис. 1.6) в поле **Project name** вводим `HelloWorld`, выбираем версию JRE и нажимаем кнопку **Finish**. Теперь добавим в проект файл с классом. Для этого в меню **File** выбираем пункт **New | Class**. В открывшемся окне (рис. 1.7) в поле **Name** вводим `HelloWorld` и нажимаем кнопку **Finish**. Редактор создаст файл HelloWorld.java и откроет его для редактирования. Причем внутри файла уже будет вставлен код. Вводим сюда код из листинга 1.1 и сохраняем проект. Для этого в меню **File** выбираем пункт **Save**.

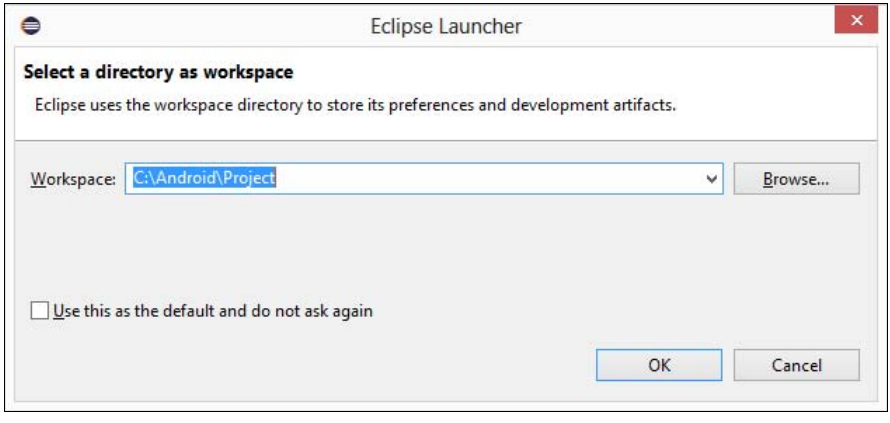

Рис. 1.5. Окно **Eclipse Launcher**

Давайте теперь откроем папку C:\Android\Project в Проводнике и посмотрим ее содержимое. В результате наших действий редактор создал папку HelloWorld и две папки внутри нее: src и bin. Внутри папки C:\Android\Project\HelloWorld\src мы можем найти файл HelloWorld.java, который содержит код из листинга 1.1. Помимо этого редактор создал вспомогательные папки и файлы, названия которых начинаются с точки. В этих папках и файлах редактор сохраняет различные настройки. Не изменяйте и не удаляйте эти файлы.

Теперь попробуем скомпилировать программу и запустить ее на выполнение. Для этого командная строка нам больше не понадобится. Переходим в редактор и в меню **Run** выбираем пункт **Run** или нажимаем комбинацию клавиш <Ctrl>+<F11>. В окне **Run As** (рис. 1.8) выбираем пункт **Java Application** и нажимаем кнопку **OK**.

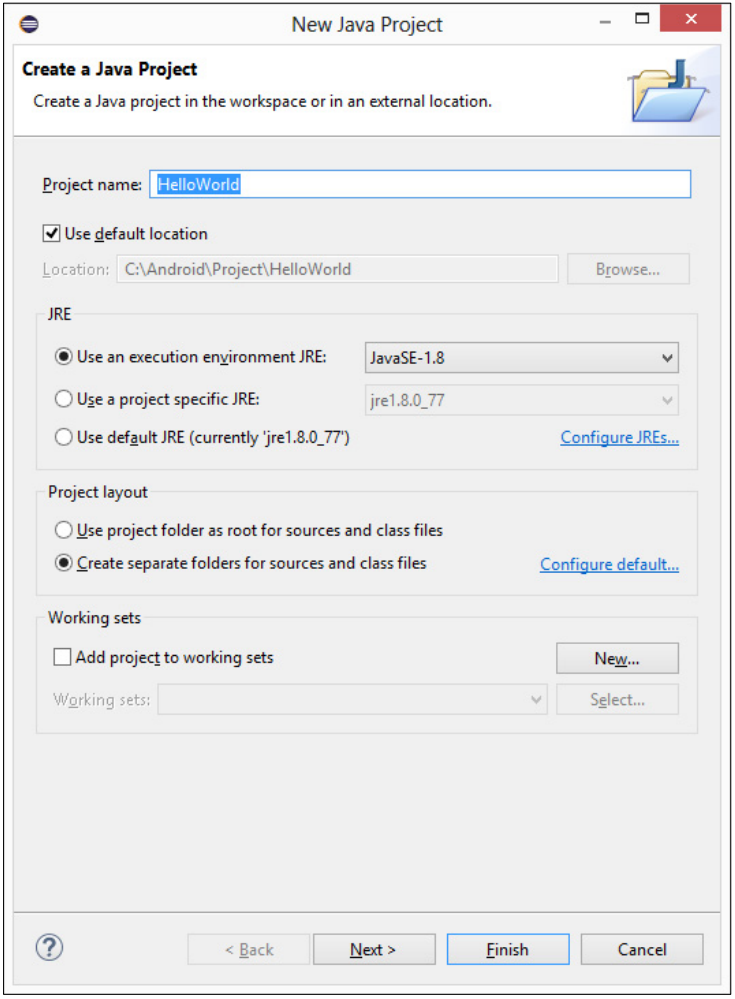

Рис. 1.6. Создание проекта

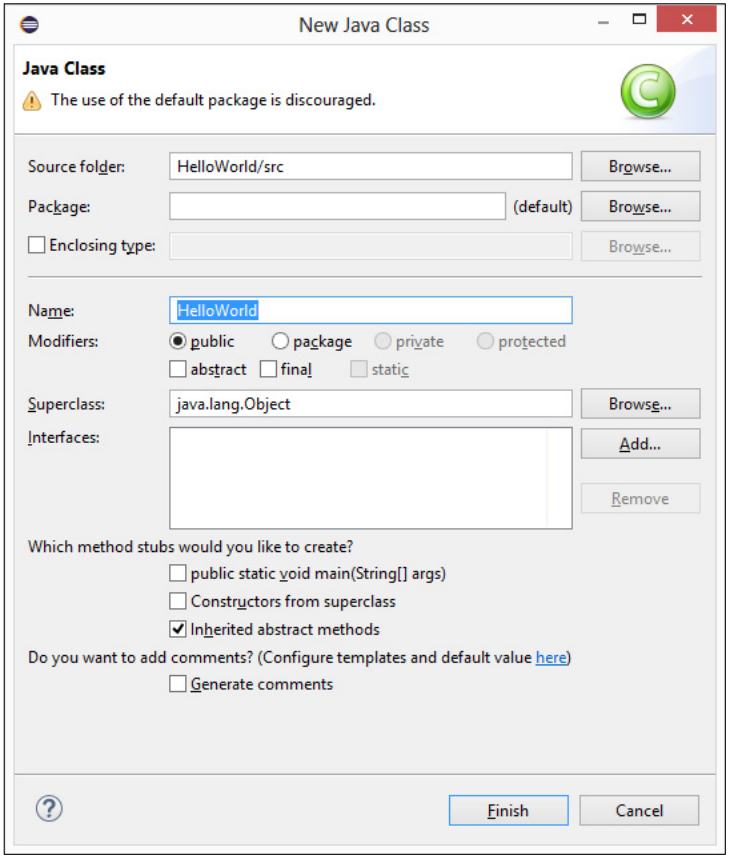

Рис. 1.7. Создание класса

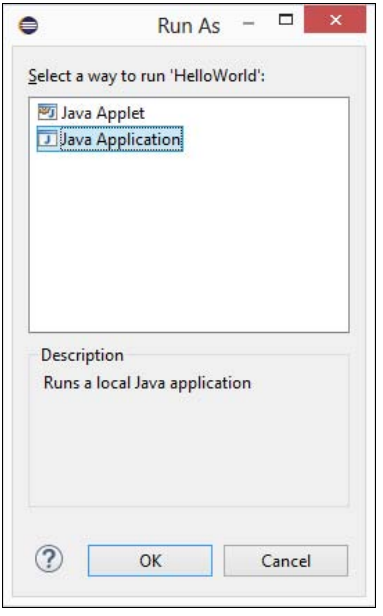

Рис. 1.8. Окно **Run As**

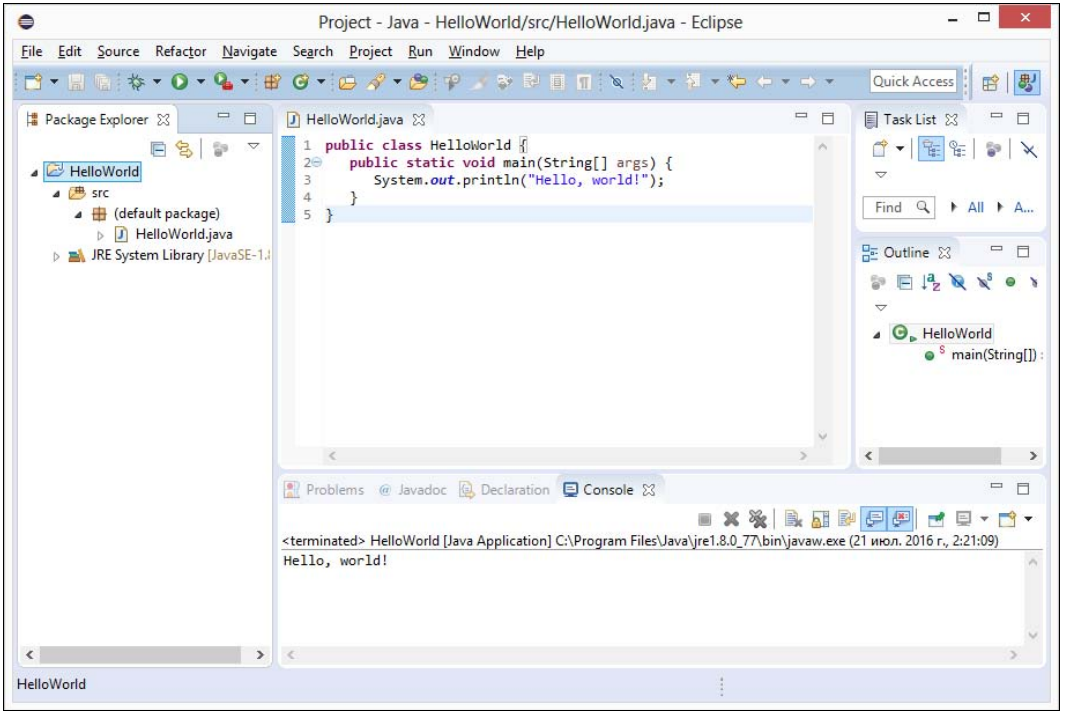

Рис. 1.9. Главное окно редактора Eclipse

В результате в папке C:\Android\Project\HelloWorld\bin будет создан файл HelloWorld.class с байт-кодом, а результат выполнения программы отобразится в окне **Console** редактора Eclipse (рис. 1.9). Все очень просто, быстро и удобно.

Как можно видеть, редактор не только подсвечивает код программы и выделяет фрагменты с ошибками, но и позволяет получить справку при наведении указателя мыши на какое-либо название. Например, наведите указатель на метод `println()`, и вы получите краткую справку по этому методу. Но сейчас это будет работать только при активном подключении к Интернету. Если хотите, чтобы справка отображалась без подключения к Интернету, то нужно выполнить следующие действия:

1. Переходим на страницу:

   http://www.oracle.com/technetwork/java/javase/downloads/index.html

   и скачиваем архив с документацией. В нашем случае архив называется jdk-8u77-docs-all.zip.

2. Распаковываем архив и копируем папку docs в каталог C:\Program Files\Java\jdk1.8.0_77.

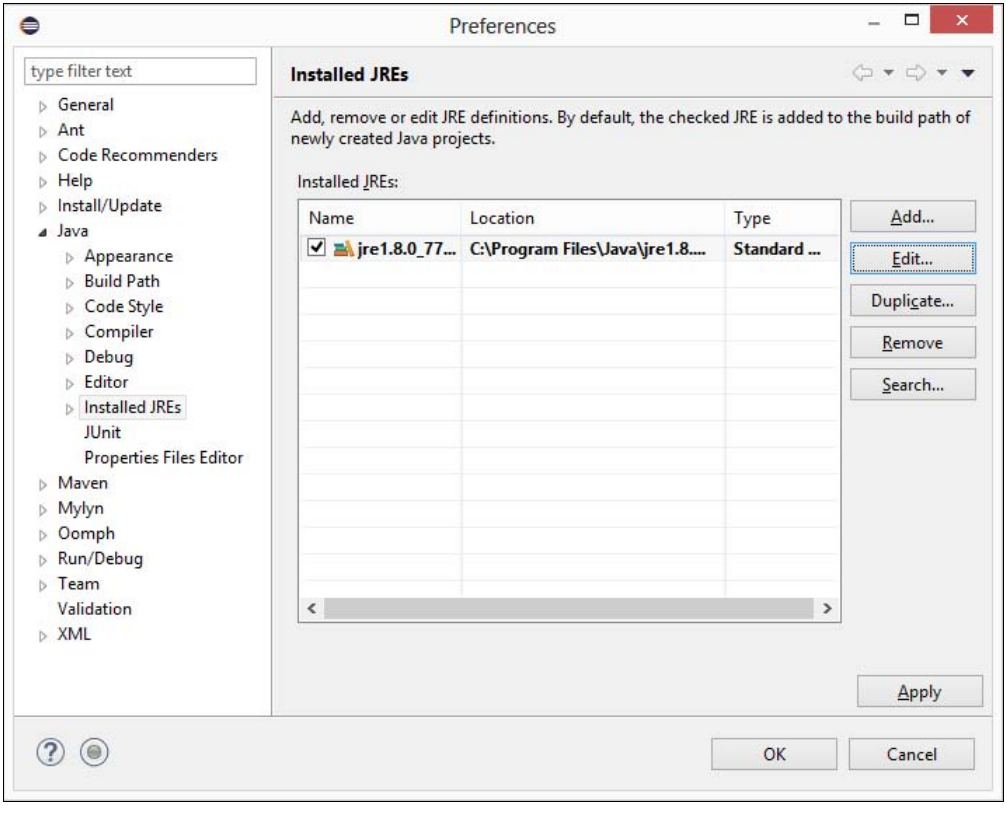

Рис. 1.10. Окно **Preferences**: вкладка **Installed JREs**

3. В меню **Window** редактора Eclipse выбираем пункт **Preferences**. В открывшемся окне переходим на вкладку **Java | Installed JREs** (рис. 1.10). Выделяем строку с JRE и нажимаем кнопку **Edit**. В списке библиотек (рис. 1.11) находим строку **C:\Program Files\Java\jre1.8.0_77\lib\rt.jar**, выделяем ее и нажимаем кнопку **Javadoc Location**. В поле **Javadoc location path** (рис. 1.12) вводим значение

`file:/C:/Program%20Files/Java/jdk1.8.0_77/docs/api/`. Если редактор не признает введенное значение, то находим папку docs/api/, нажав кнопку **Browse**. Сохраняем все изменения.

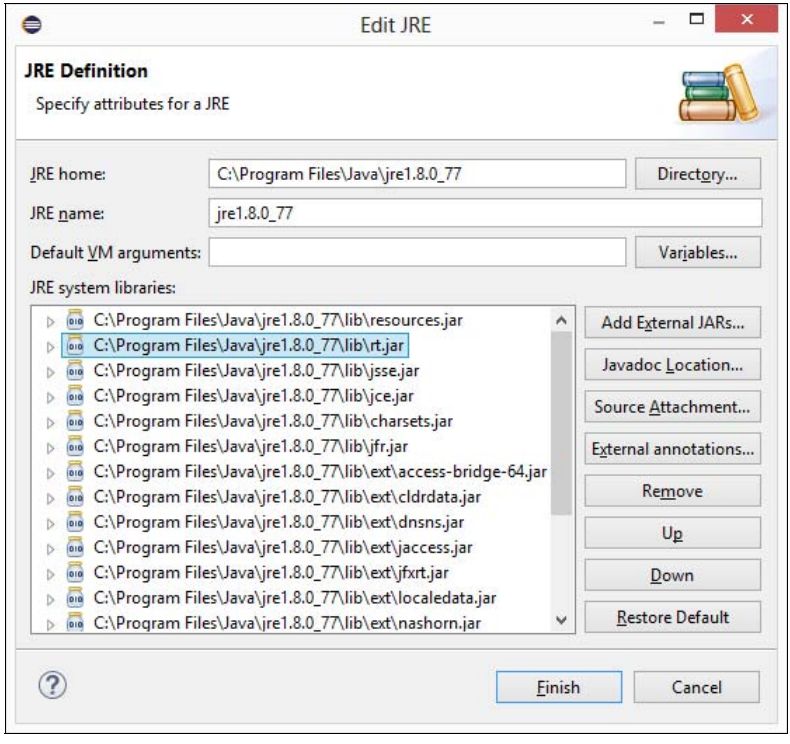

Рис. 1.11. Окно **Edit JRE**

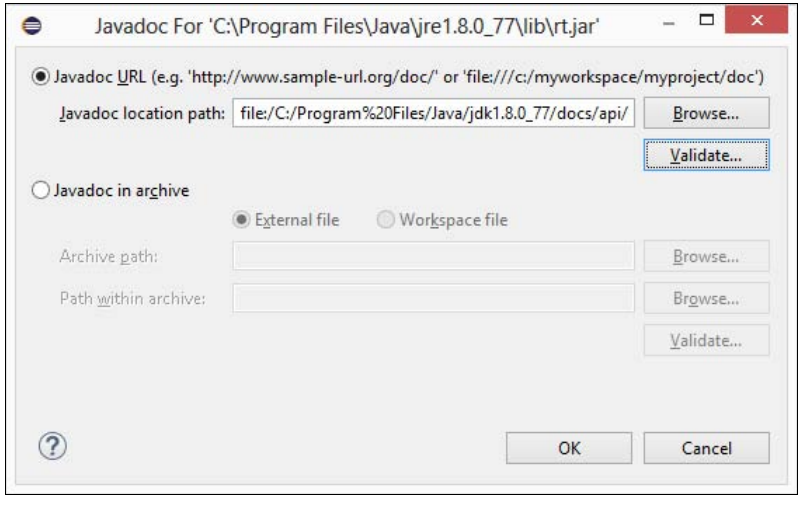

Рис. 1.12. Указание пути к документации

Если после названия объекта вставить точку, то редактор отобразит список методов и свойств. Например, попробуйте поставить точку после объекта `System.out`, и вы получите список методов этого объекта (рис. 1.13). Если при этом набирать первые буквы, то содержимое списка сократится. Достаточно выбрать метод из списка, и его название будет вставлено в код. Каким бы длинным ни было название метода, мы можем его вставить в код без ошибок в его названии, и очень быстро.

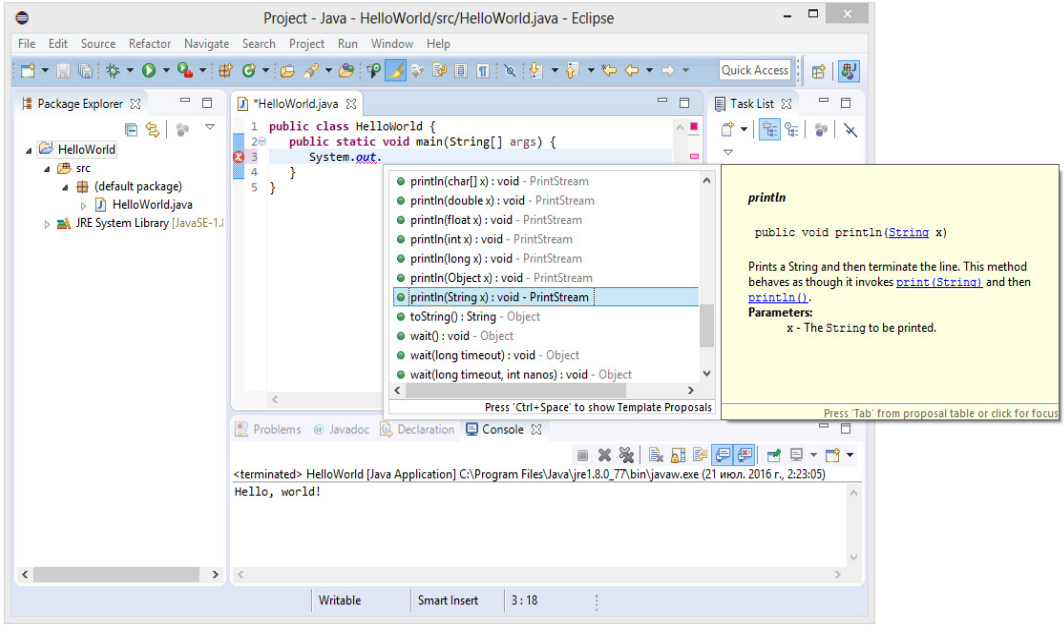

Рис. 1.13. Отображение списка методов и документации к методу

Редактор позволяет также закончить частично набранное слово. Для этого достаточно нажать комбинацию клавиш <Ctrl>+<Пробел>. В результате редактор автоматически закончит слово или предложит список с возможными вариантами. Причем этот способ отобразит еще и названия *шаблонов кода*. При выборе шаблона будет вставлено не просто слово, а целый фрагмент кода. Например, введите слово `sysout` и нажмите комбинацию клавиш <Ctrl>+<Пробел>. В результате будет вставлен следующий код:

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

Это очень удобно и экономит много времени. Чтобы увидеть все доступные шаблоны, в меню **Window** выбираем пункт **Preferences**. В открывшемся окне переходим на вкладку **Java | Editor | Templates** (рис. 1.14). С помощью этой вкладки можно не только посмотреть все шаблоны, но и отредактировать их, а также создать свой собственный шаблон.

Как уже было отмечено ранее, работать мы будем с кодировкой UTF-8, поэтому давайте настроим кодировку файлов по умолчанию. Для этого в меню **Window** выбираем пункт **Preferences**. В открывшемся окне переходим на вкладку **General |**

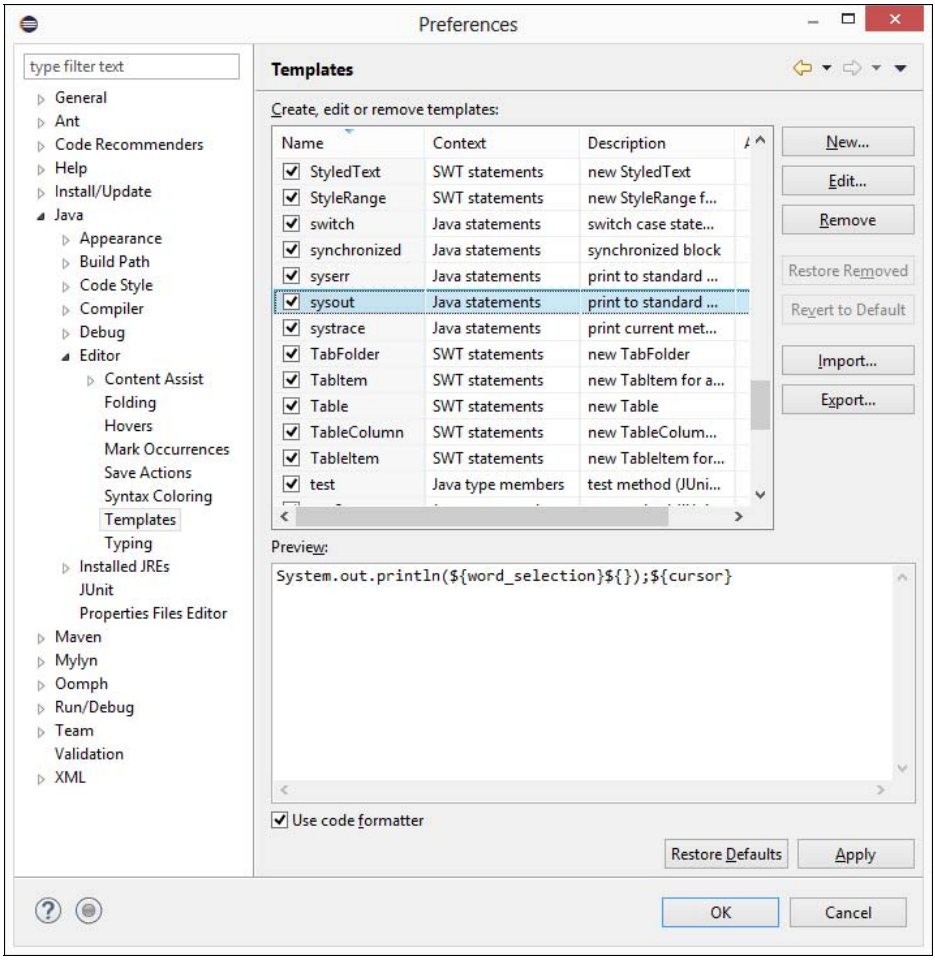

Рис. 1.14. Окно **Preferences**: вкладка **Templates**

Workspace (рис. 1.15). В группе **Text file encoding** устанавливаем флажок **Other** и из списка выбираем кодировку **UTF-8**. Сохраняем изменения. Если необходимо изменить кодировку уже открытого файла, в меню **Edit** выбираем пункт **Set Encoding**.

По умолчанию редактор вместо пробелов вставляет символы табуляции. Нас это не устраивает. Давайте изменим настройку форматирования кода. Для этого в меню **Window** выбираем пункт **Preferences**. В открывшемся окне переходим на вкладку **Java | Code Style | Formatter** (рис. 1.16). Нажимаем кнопку **New**. В открывшемся окне (рис. 1.17) в поле **Profile name** вводим название стиля, например: `MyStyle`, а из списка выбираем пункт **Eclipse [built-in]**. Нажимаем кнопку **OK**. Откроется окно, в котором можно изменить настройки нашего стиля. На вкладке **Indentation** из списка **Tab policy** выбираем пункт **Spaces only**, а в поля **Indentation size** и **Tab size** вводим число `3`. Сохраняем все изменения.

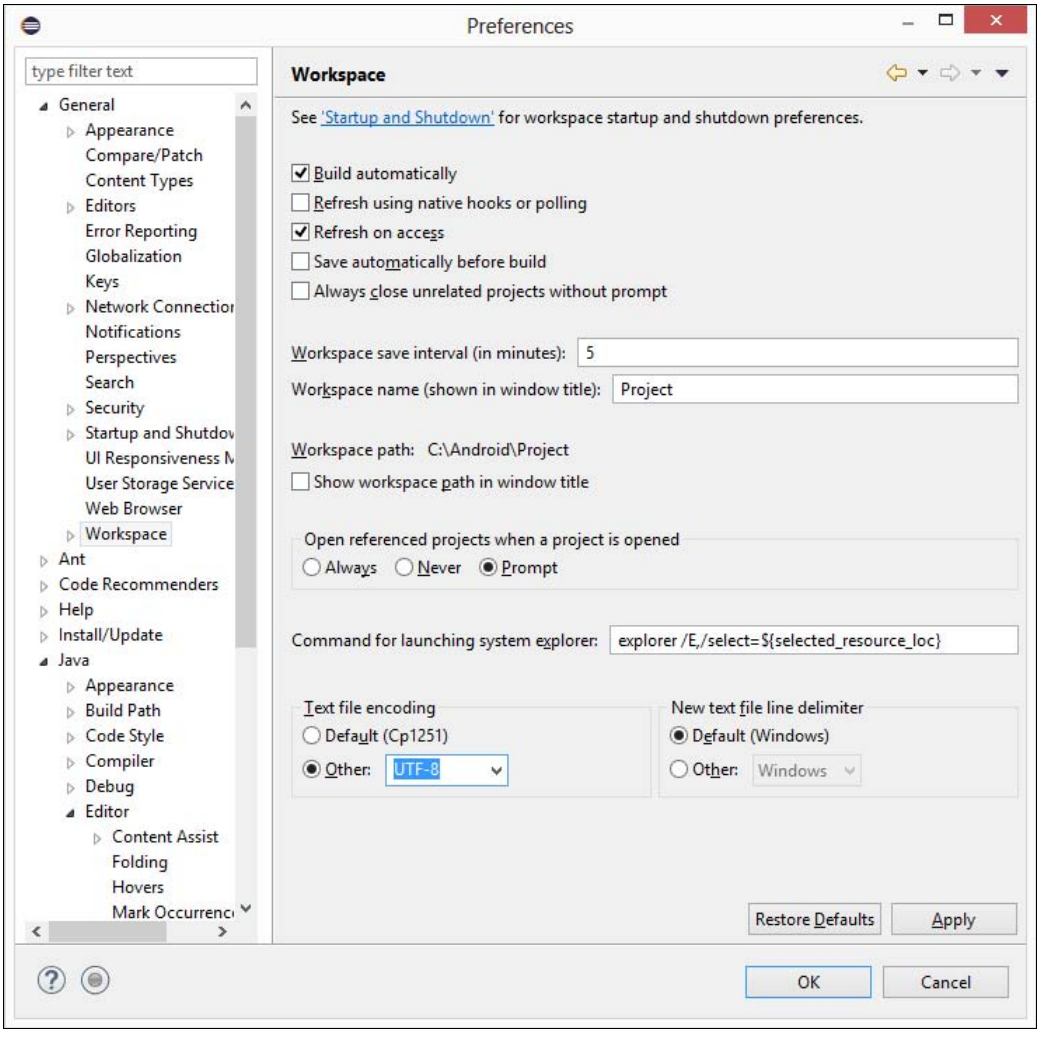

Рис. 1.15. Указание кодировки файлов

Если необходимо изменить размер шрифта, то в меню **Window** выбираем пункт **Preferences**. В открывшемся окне переходим на вкладку **General | Appearance | Colors and Fonts** (рис. 1.18). Из списка выбираем пункт **Java | Java Editor Text Font** и нажимаем кнопку **Edit**.

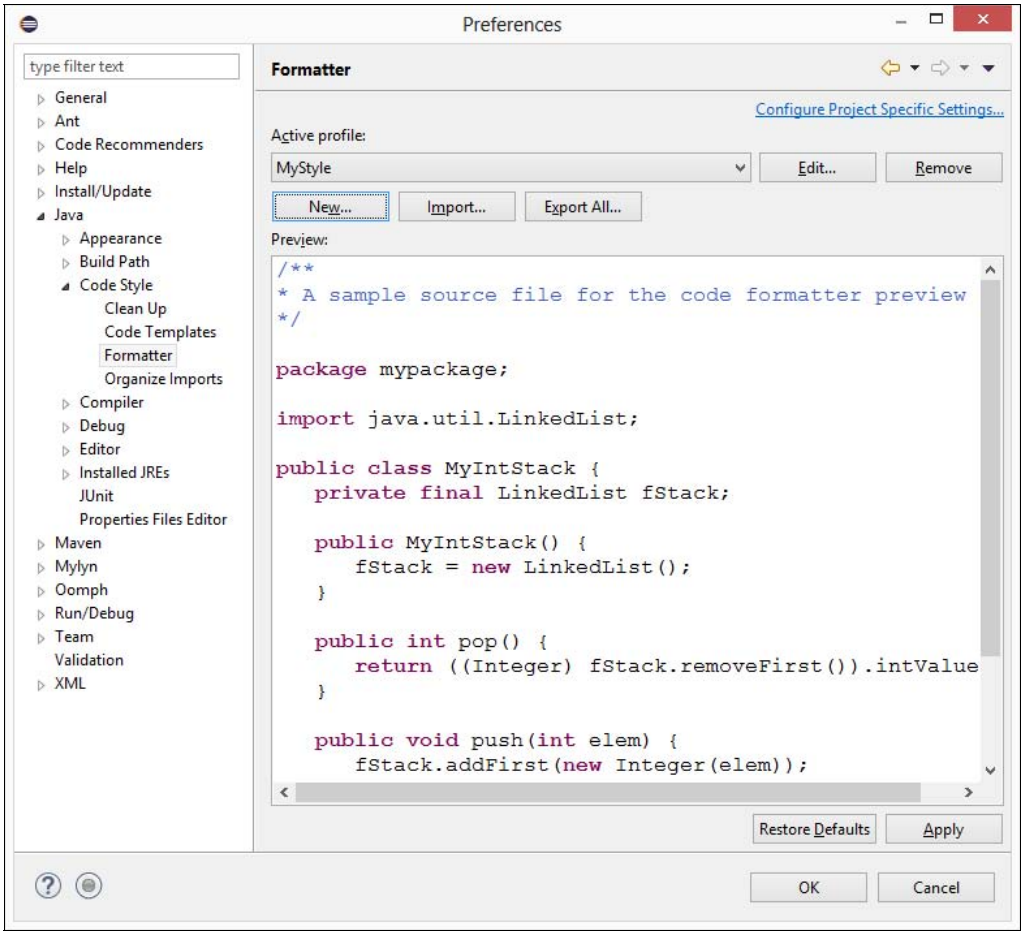

Рис. 1.16. Окно **Preferences**: вкладка **Formatter**

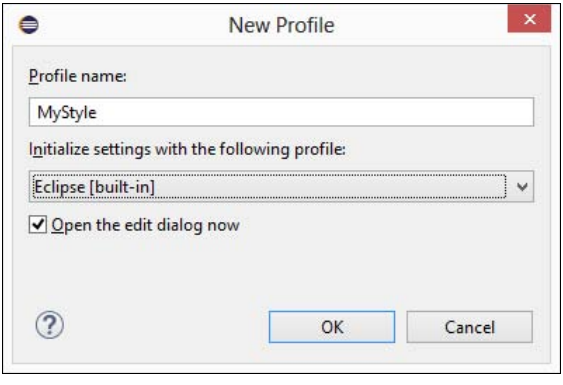

Рис. 1.17. Окно **New Profile**

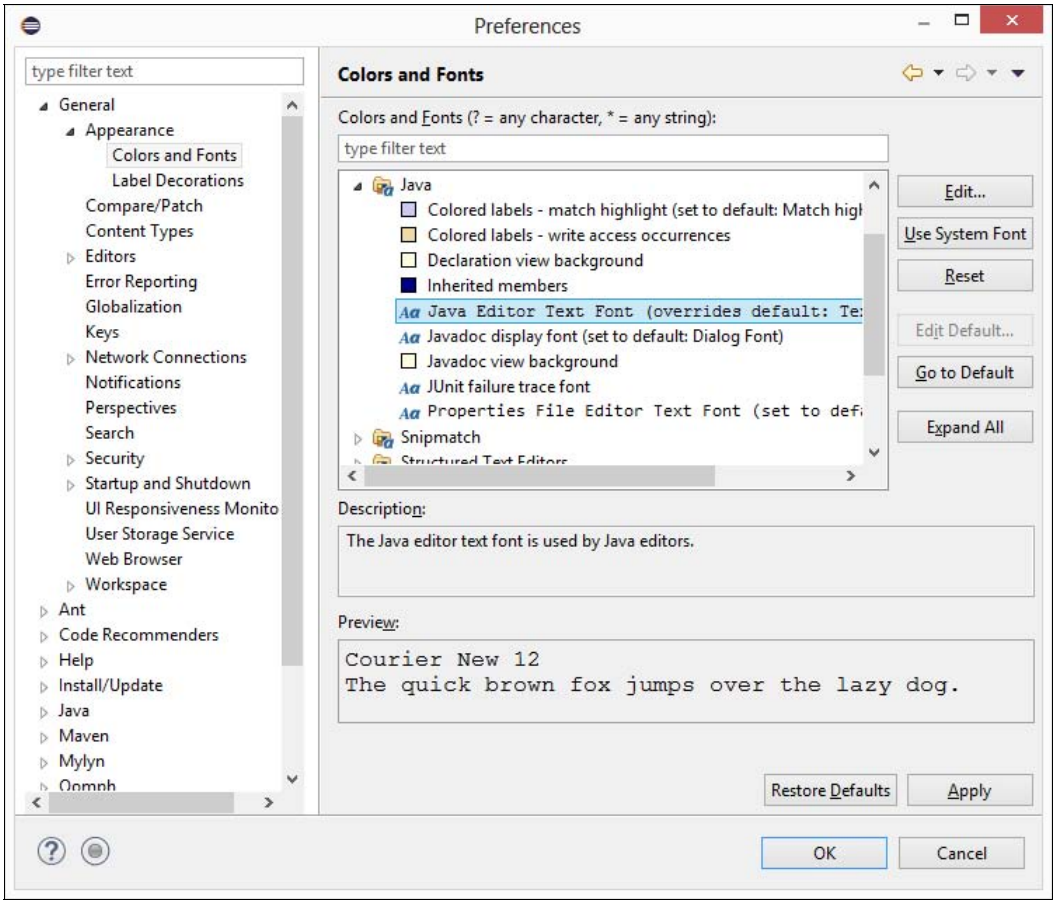

Рис. 1.18. Окно **Preferences**: вкладка **Colors and Fonts**

1.4. Структура программы

Что ж, программная среда установлена, и редактор настроен. Теперь можно приступать к изучению языка. Начнем с рассмотрения кода из листинга 1.1, который выводит приветствие в окно консоли. Весь код состоит из трех инструкций:

- описание класса `HelloWorld`;
- описание метода `main()`;
- вывод приветствия с помощью метода `println()`.

Поскольку язык Java является объектно-ориентированным языком программирования, текст программы всегда находится внутри описания какого-либо класса. Причем название файла, содержащего описание класса, в точности совпадает с названием класса. Вплоть до регистра символов! В нашем случае класс `HelloWorld` находится внутри файла с названием HelloWorld.java.

Описание класса является составной инструкцией:

```
public class HelloWorld {
}
```

Модификатор доступа `public` означает, что класс `HelloWorld` является общедоступным, и любой другой объект может обратиться к этому классу, создать экземпляр этого класса, получить доступ к общедоступным методам и полям. Ключевое слово `class` означает, что описывается класс с названием, указанным после слова `class`. Далее между фигурными скобками вставляется описание методов и полей класса. В нашем случае описание метода `main()`.

Фигурные скобки заключают блок, который ограничивает область видимости идентификаторов. Все идентификаторы, описанные внутри блока, видны только внутри этого блока. Извне блока обратиться к идентификаторам можно только через точку после имени класса. В нашем случае к методу `main()` можно обратиться так:

```
HelloWorld.main(<Значение>)
```

Причем не ко всем идентификаторам можно получить доступ таким образом. Давайте взглянем на описание метода `main()`:

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

Модификатор доступа `public` означает, что метод `main()` является общедоступным. Если метод был бы описан с помощью ключевого слова `private`, то обратиться к методу с помощью оператора «точка» было бы нельзя.

Ключевое слово `static` означает, что метод является *статическим*. В этом случае мы можем обратиться к методу через оператор «точка» без необходимости создания экземпляра класса. Что же такое класс, а что *экземпляр класса*?

В языке Java все — либо класс, либо объект (экземпляр класса). Предположим у нас есть класс `Телевизор`. Внутри этого класса описан чертеж, электрическая схема и т. д. То есть, приведено всего лишь описание телевизора и принципа его работы. Далее на основании этого класса мы производим несколько экземпляров телевизоров (создаем несколько объектов класса `Телевизор`). Все они построены на основе одного класса, но могут иметь разные свойства. Один черного цвета, второй серого, третий белого. Один включен, второй в режиме ожидания, третий вообще выключен. Таким образом, говоря простым языком, класс — это схема, а экземпляр класса — объект, созданный по этой схеме. Класс один, а экземпляров может быть множество, и каждый экземпляр может обладать своими собственными свойствами, не зависящими от свойств других экземпляров.

Класс может быть не только схемой, но и пространством имен, внутри которого описываются какие-либо методы. В этом случае методы помечаются с помощью ключевого слова `static`. Доступ к статическому методу осуществляется без создания экземпляра класса с помощью оператора «точка» по следующей схеме:

```
<Название класса>.<Метод>([<Параметры>])
```

В описании метода `main()` после ключевого слова `static` указан тип возвращаемого методом значения. Ключевое слово `void` означает, что метод вообще ничего не возвращает. Что такое метод? *Метод* — это фрагмент кода внутри блока класса, который может быть вызван из какого-либо места программы сколько угодно раз. При этом код может возвращать какое-либо значение в место вызова или вообще ничего не возвращать, а просто выполнять какую-либо операцию. В нашем случае метод `main()` выводит приветствие в окно консоли и ничего не возвращает. Если вы программировали на других языках, то знакомы с понятием *функция*. Можно сказать, что метод — это функция, объявленная внутри класса. В языке Java нет процедурного стиля программирования, поэтому просто функций самих по себе не существует. Только методы внутри класса.

Название метода `main()` является предопределенным. Именно этот метод выполняется, когда мы запускаем программу из командной строки. Регистр в названии метода имеет значение. Если программа не содержит метода `main()`, то при запуске из командной строки мы получим следующее сообщение об ошибке:

```
C:\book>java HelloWorld
Error: Main method not found in class HelloWorld, please define
the main method
as:
   public static void main(String[] args)
or a JavaFX application class must extend javafx.application.Application
```

Метод может принимать какие-либо параметры, указываемые внутри круглых скобок через запятую. Язык Java является строго типизированным языком, поэтому в описании метода указывается тип значения принимаемого параметра. В нашем случае внутри круглых скобок указано следующее выражение:

```
String[] args
```

Ключевое слово `String` означает, что метод принимает объект класса `String` (проще говоря, строку из символов). Квадратные скобки являются признаком массива (проще говоря, строк может быть много). Идентификатор `args` — всего лишь имя переменной, через которое мы можем получить доступ к массиву строк внутри метода. При запуске программы в метод передаются параметры, указанные после названия программы в командной строке. Например, передадим два параметра:

```
C:\book>java HelloWorld string1 string2
```

Описание метода также является блочной инструкцией, поэтому после описания параметров указывается открывающая фигурная скобка. Закрывающая фигурная скобка является признаком конца блока. Описание метода `main()` вложено в блок класса, поэтому метод принадлежит классу. Чтобы наглядно видеть вложенность блоков, в коде перед описанием метода `main()` указывается одинаковое количество пробелов. Обычно три или четыре. Вместо пробелов можно использовать символ табуляции. Выберите для себя какой-либо один способ и используйте его постоянно. В этой книге мы будем использовать три пробела. Компилятору не важно, сколько пробелов вы поставите. Можно вообще написать так:

```
public class HelloWorld {
public static void main(String[] args) {
System.out.println("Hello, world!");
}
}
```

Такой код становится трудно читать, прежде всего, программисту. Ведь непонятно где начало блока, а где конец. Поэтому возьмите за правило выделять блоки в программе одинаковым количеством пробелов. Для вложенных блоков количество пробелов умножают на уровень вложенности. Если для блока первого уровня вложенности использовалось три пробела, то для блока второго уровня вложенности должно использоваться шесть пробелов, для третьего уровня — девять пробелов и т. д. В одной программе в качестве отступа не следует использовать и пробелы, и табуляцию, — необходимо выбрать что-то одно и пользоваться этим во всей программе.

Фигурные скобки также можно расставлять по-разному. Опять-таки компилятору это не важно. Обычно применяются два стиля. Первый стиль мы использовали в листинге 1.1. Открывающая фигурная скобка на одной строке с описанием класса или метода, а закрывающая — на отдельной строке. Второй стиль выглядит следующим образом:

```
public class HelloWorld
{
   public static void main(String[] args)
   {
      System.out.println("Hello, world!");
   }
}
```

Во втором стиле обе фигурные скобки расположены на отдельных строках. Многие программисты считают такой стиль наиболее приемлемым, т. к. открывающая и закрывающая скобки расположены друг под другом. На мой же взгляд здесь образуются лишние пустые строки. А так как размеры экрана ограничены, при наличии пустой строки на экране помещается меньше кода, и приходится чаще пользоваться полосой прокрутки. Если размещать инструкции с равным отступом, то блок кода выделяется визуально и следить за положением фигурных скобок просто излишне. Тем более, что редактор позволяет подсветить парные скобки. Какой стиль использовать, зависит от личного предпочтения программиста или от правил оформления кода, принятых в той или иной фирме. Главное, чтобы стиль оформления внутри одной программы был одинаковым. В этой книге мы будем использовать первый стиль.

Внутри метода `main()` с помощью инструкции

```
System.out.println("Hello, world!");
```

выводится приветствие **Hello, world!** в окно консоли. Идентификатор `System` является системным классом, входящим в состав библиотеки языка Java. Внутри этого класса находится свойство `out`, содержащее ссылку на объект класса `PrintStream`,

который связан с окном консоли. Так как `out` находится внутри класса `System`, то получить к нему доступ можно через оператор «точка». Теперь у нас есть объект класса `PrintStream`, который имеет несколько методов, позволяющих вывести данные. Чтобы получить доступ к этим методам, опять применяется оператор «точка». В нашем случае используется метод `println()` из этого класса. Внутри круглых скобок мы передаем строку, которая и будет выведена в окне консоли.

После закрывающей круглой скобки ставится точка с запятой, которая говорит компилятору о конце инструкции. Это все равно, что поставить точку в конце предложения. Если точку с запятой не указать, то компилятор выведет сообщение об ошибке. Обратите внимание на то, что в конце составных инструкций после закрывающей фигурной скобки точка с запятой не ставится. Именно скобки указывают компилятору начало и конец блока. На самом деле, если поставить точку с запятой после закрывающей фигурной скобки, ошибки не будет. Просто компилятор посчитает пустое пространство между скобкой и точкой с запятой как пустую инструкцию.

Начинающему программисту сложно понять все, что мы рассмотрели в этом разделе. Слишком много информации... Тут и классы, и методы, и инструкции. Да, *объектно-ориентированный стиль программирования* (сокращенно *ООП-стиль*) даже опытные программисты не могут сразу понять. Но ничего не поделаешь, в языке Java возможен только ООП-стиль. Не беспокойтесь, в дальнейшем классы и методы мы рассмотрим подробно и последовательно. Чтобы сократить количество кода, в дальнейших примерах код создания класса и метода `main()` мы будем опускать. В результате шаблон для тестирования примеров должен выглядеть так:

```
public class MyClass {
   public static void main(String[] args) {
<Здесь набираем код из примеров>
   }
}
```

1.5. Комментарии в программе

Комментарии предназначены для вставки пояснений в текст программы, и компилятор полностью их игнорирует. Внутри комментария может располагаться любой текст, включая инструкции, которые выполнять не следует. Помните, комментарии нужны программисту, а не компилятору. Вставка комментариев в код позволит через некоторое время быстро вспомнить предназначение фрагмента кода.

В языке Java присутствуют два типа комментариев: *однострочный* и *многострочный*. Однострочный комментарий начинается с символов `//` и заканчивается в конце строки. Вставлять однострочный комментарий можно как в начале строки, так и после инструкции:

```
// Это комментарий
System.out.println("Hello, world!"); // Это комментарий
```

Если символ комментария разместить перед инструкцией, то она не будет выполнена:

```
// System.out.println("Hello, world!"); // Инструкция выполнена не будет
```

Если символы // расположены внутри кавычек, то они не являются признаком начала комментария:

```
System.out.println("// Это НЕ комментарий!!!");
```

Многострочный комментарий начинается с символов /* и заканчивается символами */. Комментарий может быть расположен как на одной строке, так и на нескольких. Кроме того, многострочный комментарий можно размещать внутри выражения, хотя это и нежелательно. Следует иметь в виду, что многострочные комментарии не могут быть вложенными, поэтому при комментировании больших блоков следует проверять, что в них не встречается закрывающая комментарий комбинация символов */. Вот примеры многострочных комментариев:

```
/*
Многострочный комментарий
*/

System.out.println("Hello, world!"); /* Это комментарий */

/* System.out.println("Hello, world!"); // Инструкция выполнена не будет
*/

int x;
x = 10 /* Комментарий */ + 50 /* внутри выражения */;

System.out.println("/* Это НЕ комментарий!!! */");
```

Редактор Eclipse позволяет быстро добавить символы комментариев. Чтобы вставить однострочный комментарий в начале строки, нужно сделать текущей строку с инструкцией и нажать комбинацию клавиш <Ctrl>+</>. Если предварительно выделить сразу несколько строк, то перед всеми выделенными строками будет вставлен однострочный комментарий. Если все выделенные строки были закомментированы ранее, то комбинация клавиш <Ctrl>+</> удалит все однострочные комментарии. Для вставки многострочного комментария необходимо выделить строки и нажать комбинацию клавиш <Shift>+<Ctrl>+</>. Для удаления многострочного комментария предназначена комбинация клавиш <Shift>+<Ctrl>+<\>.

У начинающих программистов может возникнуть вопрос, зачем может потребоваться комментировать инструкции? Проблема заключается в том, что часто в логике работы программы возникают проблемы. Именно по вине программиста. Например, программа выдает результат, который является неверным. Чтобы найти ошибку в алгоритме работы программы, приходится отключать часть кода с помощью комментариев, вставлять инструкции вывода промежуточных результатов и анализировать их. Как говорится: разделяй и властвуй. Таким «дедовским» способом мы обычно ищем ошибки в коде. А «дедовским» мы назвали способ потому,

что сейчас все редакторы предоставляют методы отладки, которые позволяют выполнять код построчно и сразу видеть промежуточные результаты. Раньше такого не было. Хотя способ и устарел, но все равно им часто пользуются.

Язык Java поддерживает также *комментарии документирования*. Комментарий документирования является многострочным и начинается с символов `/**` и заканчивается символами `*/`. Чтобы автоматически сгенерировать такой комментарий в редакторе Eclipse, необходимо предварительно установить курсор, например, на название класса, и в меню **Source** выбрать пункт **Generate Element Comment** или нажать комбинацию клавиш <Shift>+<Alt>+<J>. Попробуйте создать комментарии документирования для класса и для метода `main()`. В результате код будет выглядеть примерно следующим образом:

```
/**
 * @author Unicross
 *
 */
public class MyClass {
   /**
    * @param args
    */
   public static void main(String[] args) {
      System.out.println("Hello, world!");
   }
}
```

Внутри комментария документирования может присутствовать как обычный текст (допускается использование HTML-тегов для форматирования), так и специальные дескрипторы, начинающиеся с символа `@`. В нашем примере присутствуют два дескриптора: `@author` и `@param`. Перечислим основные дескрипторы и их предназначение:

- `@author` — задает имя автора;
- `@version` — номер версии;
- `@since` — начальная версия, с которой стал доступен код;
- `@param` — описание параметра метода;
- `@return` — описание возвращаемого методом значения;
- `@throws` и `@exception` — описывают генерируемое внутри метода исключение;
- `{@inheritDoc}` — вставляет комментарий из базового класса;
- `{@code}` — позволяет добавить код примера:

```
* <pre> {@code
* List<Integer> list = new ArrayList<Integer>();
* }</pre>
```

- `{@link}` — создает ссылку на дополнительную информацию (ссылка размещается внутри текста):

```
* {@link MyClass#MyClass() Конструктор класса}
```

- `@see` — создает ссылку на дополнительную информацию (ссылка размещается в блоке `See Also`):

```
@see MyClass#MyClass() Конструктор класса
```

Существуют и другие дескрипторы. Лучший способ изучать комментарии документирования — смотреть исходный код классов из стандартной библиотеки. Такой исходный код можно найти в архиве C:\Program Files\Java\jdk1.8.0_77\src.zip.

Теперь попробуем сгенерировать документацию в формате HTML на основе комментариев документирования. Для этого в редакторе Eclipse в меню **Project** выбираем пункт **Generate Javadoc**. В открывшемся окне (рис. 1.19) нажимаем кнопку **Configure** и находим программу **javadoc.exe** (**C:\Program Files\Java\jdk1.8.0_77\bin\javadoc.exe**). В результате путь к программе должен отображаться в поле **Javadoc command**. Выбираем наш класс и нажимаем кнопку **Next**. И еще раз нажимаем кнопку **Next**. Чтобы русские буквы нормально отображались в документации, необходимо дополнительно указать кодировку файла с программой, файла с документацией и кодировку для тега `<meta>`:

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
```

Для этого в поле **Extra Javadoc options** (рис. 1.20) вводим следующую команду:

```
-encoding utf-8 -docencoding utf-8 -charset utf-8
```

Нажимаем кнопку **Finish** для запуска генерации документации. Если все сделано правильно, то в папке C:\Android\Project\MyClass\doc будет создано множество различ-

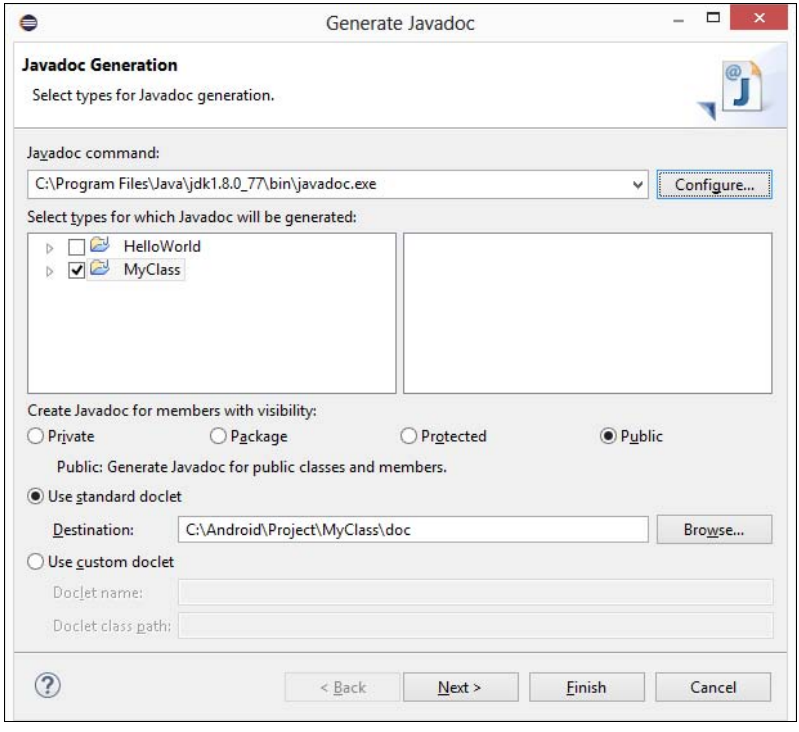

Рис. 1.19. Окно **Generate Javadoc**

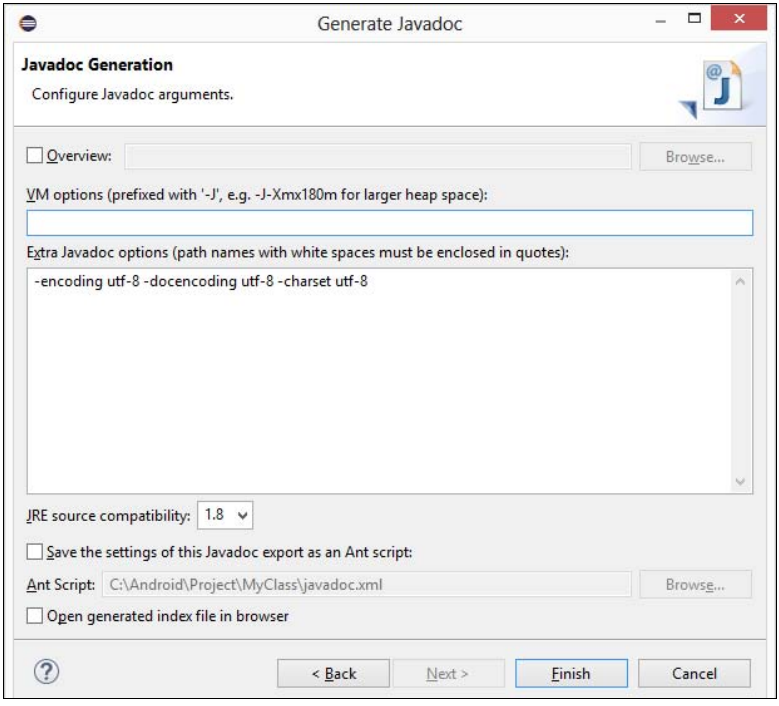

Рис. 1.20. Указание кодировки файлов документации

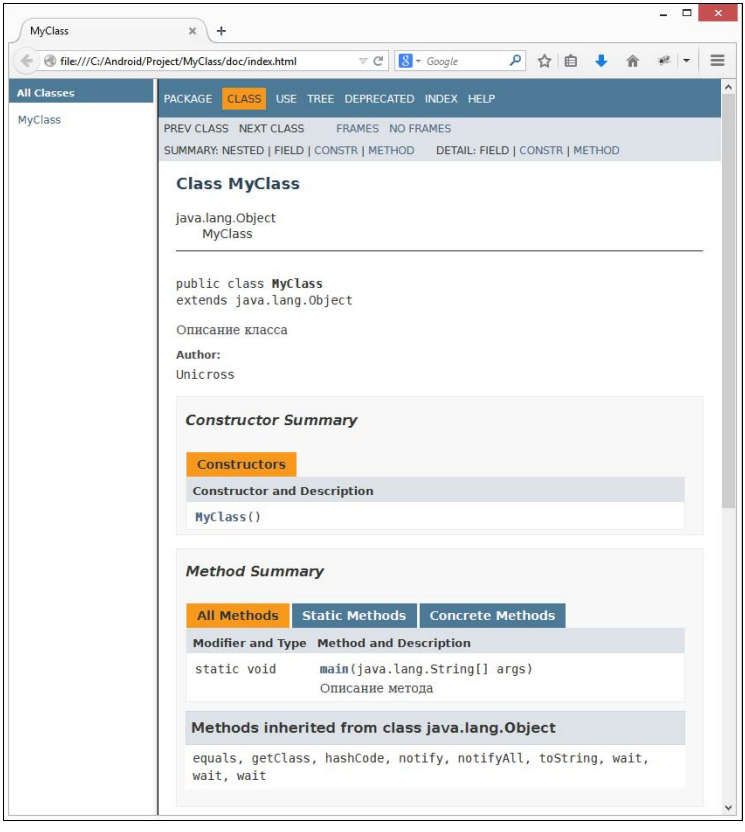

Рис. 1.21. Документация в окне Web-браузера

ных файлов. Для просмотра документации открываем файл index.html с помощью любого Web-браузера. Результат показан на рис. 1.21.

1.6. Вывод данных

Для вывода данных в языке Java предназначены объекты `System.out` и `System.err`. Объект `System.out` используется для вывода обычных сообщений в окно консоли, а объект `System.err` — для вывода сообщений об ошибках. Оба объекта первоначально связаны с окном консоли, однако возможно перенаправить поток на другое устройство, например, в файл.

Объекты `System.out` и `System.err` возвращают экземпляр класса `PrintStream`, через который доступны следующие основные методы:

- `print()` — отправляет данные в стандартный поток вывода. Формат метода:

  ```
  public void print(<Данные>)
  ```

  В параметре `<Данные>` можно указать данные типов `boolean`, `int`, `long`, `float`, `double`, `char`, `char[]`, `String` и `Object`. Пример:

  ```
  System.out.print(10);
  System.out.print("Hello, world!");
  // Результат: 10Hello, world!
  System.err.print("Сообщение об ошибке");
  // Результат: Сообщение об ошибке
  ```

- `println()` — отправляет данные в стандартный поток вывода и завершает текущую строку. Формат метода:

  ```
  public void println([<Данные>])
  ```

  В параметре `<Данные>` можно указать данные типов `boolean`, `int`, `long`, `float`, `double`, `char`, `char[]`, `String` и `Object`. Если параметр не указан, то просто завершает текущую строку. Пример:

  ```
  System.out.println(10);
  System.out.println("Hello, world!");
  /*
  Результат:
  10
  Hello, world!
  */
  System.err.println("Сообщение об ошибке");
  ```

- `printf()` — предназначен для форматированного вывода данных. Форматы метода:

  ```
  public PrintStream printf(String format, Object... args)
  public PrintStream printf(Locale locale, String format,
                                           Object... args)
  ```

В параметре `format` указывается строка специального формата, внутри которой с помощью спецификаторов задаются правила форматирования. Какие спецификаторы используются, мы рассмотрим при изучении форматирования строк. В параметре `args` через запятую указываются различные значения. Параметр `locale` позволяет задать *локаль*. Настройки локали для разных стран отличаются. Например, в одной стране принято десятичный разделитель вещественных чисел выводить в виде точки, в другой — в виде запятой. В первом формате метода используются настройки локали по умолчанию. Пример:

```
System.out.println(10.5125484);              // 10.5125484
System.out.printf("%.2f", 10.5125484);       // 10,51
```

- `flush()` — сбрасывает данные из буфера. Формат метода:

```
public void flush()
```

- `checkError()` — сбрасывает данные из буфера и возвращает значение `true`, если произошла ошибка, или `false`, если ошибка не возникла. Формат метода:

```
public boolean checkError()
```

С помощью стандартного вывода можно создать индикатор выполнения процесса в окне консоли. Чтобы реализовать такой индикатор, нужно вспомнить, что символ перевода строки в Windows состоит из двух символов `\r` (перевод каретки) и `\n` (перевод строки). Таким образом, используя только символ перевода каретки `\r`, можно перемещаться в начало строки и перезаписывать ранее выведенную информацию. Пример индикатора процесса показан в листинге 1.2.

Листинг 1.2. Индикатор выполнения процесса

```
public class MyClass {
   public static void main(String[] args) throws InterruptedException {
      System.out.print("... 0%");
      for (int i = 5; i < 101; i += 5) {
         Thread.sleep(1000); // Имитация процесса
         System.out.print("\r... " + i + "%");
      }
      System.out.println();
   }
}
```

Открываем командную строку (в редакторе Eclipse эффекта не будет видно), компилируем программу и запускаем:

```
C:\Users\Unicross>cd C:\book

C:\book>javac -encoding utf-8 MyClass.java

C:\book>java MyClass
```

Эта программа немного сложнее, чем простое приветствие из листинга 1.1. Здесь присутствует обработка исключений (выражение `throws InterruptedException`),

имитация процесса с помощью метода `sleep()` из класса `Thread` (при этом программа «засыпает» на указанное количество миллисекунд).

Кроме того, в программе использован цикл `for`, который позволяет изменить порядок обработки инструкций. Обычно программа выполняется сверху вниз и слева направо. Инструкция за инструкцией. Цикл `for` меняет эту последовательность выполнения. Инструкции, расположенные внутри блока, выполняются несколько раз. Количество повторений зависит от выражений внутри круглых скобок. Этих выражений три, и разделены они точками с запятой. Первое выражение объявляет целочисленную переменную `i` и присваивает ей значение `5`. Второе выражение является условием продолжения повторений. Пока значение переменной `i` меньше значения `101`, инструкции внутри блока будут повторяться. Это условие проверяется на каждой итерации цикла. Третье выражение на каждой итерации цикла прибавляет значение `5` к текущему значению переменной `i`.

1.7. Ввод данных

Для ввода данных в языке Java предназначен объект `System.in`. Работать с этим объектом напрямую не очень удобно, поэтому обычно используется вспомогательный класс `Scanner`, который связывают с входным потоком следующим образом:

```
Scanner in = new Scanner(System.in);
```

Класс `Scanner` объявлен в пакете `java.util`, поэтому, прежде чем использовать этот класс, необходимо подключить пакет, добавив в начале программы такую инструкцию:

```
import java.util.Scanner;
```

Для получения данных предназначены следующие основные методы из класса `Scanner` (полный список методов мы будем рассматривать при изучении потоков):

- `nextInt()` — используется для ввода целых чисел (тип `int`). Формат метода:

  ```
  public int nextInt()
  ```

  Пример:

  ```
  Scanner in = new Scanner(System.in);
  int x = 0;
  x = in.nextInt();                    // Вводим:  10
  System.out.println("Вы ввели " + x); // Выведет: Вы ввели 10
  ```

- `nextLong()` — предназначен для ввода целых чисел (тип `long`). Формат метода:

  ```
  public long nextLong()
  ```

  Пример:

  ```
  Scanner in = new Scanner(System.in);
  long x;
  x = in.nextLong();                   // Вводим:  20
  System.out.println("Вы ввели " + x); // Выведет: Вы ввели 20
  ```

- `nextShort()` — используется для ввода целых чисел (тип `short`). Формат метода:

```
public short nextShort()
```

Пример:

```
Scanner in = new Scanner(System.in);
short x;
x = in.nextShort();                    // Вводим:  30
System.out.println("Вы ввели " + x); // Выведет: Вы ввели 30
```

- `nextFloat()` — предназначен для ввода вещественных чисел (тип `float`). Формат метода:

```
public float nextFloat()
```

Пример:

```
Scanner in = new Scanner(System.in);
float x;
x = in.nextFloat();                    // Вводим:  10,5
System.out.println("Вы ввели " + x); // Выведет: Вы ввели 10.5
```

- `nextDouble()` — используется для ввода вещественных чисел (тип `double`). Формат метода:

```
public double nextDouble()
```

Пример:

```
Scanner in = new Scanner(System.in);
double x;
x = in.nextDouble();                   // Вводим:  10,5
System.out.println("Вы ввели " + x); // Выведет: Вы ввели 10.5
```

- `nextLine()` — получает строку (тип `String`). Формат метода:

```
public String nextLine()
```

Пример:

```
Scanner in = new Scanner(System.in);
String s;
s = in.nextLine();                     // Вводим:  Сергей
System.out.println("Вы ввели " + s); // Выведет: Сергей
```

В качестве примера произведем суммирование двух целых чисел, введенных пользователем в окне консоли (листинг 1.3).

Листинг 1.3. Суммирование двух введенных чисел

```
import java.util.Scanner;

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

```
        Scanner in = new Scanner(System.in);
        int x = 0, y = 0;
        System.out.print("x = ");
        x = in.nextInt();
        System.out.print("y = ");
        y = in.nextInt();
        System.out.println("Сумма = " + (x + y));
    }
}
```

Язык Java, как уже указывалось, является строго типизированным. Это означает, что переменная может содержать значения того типа, который указан при объявлении переменной. В нашем примере мы объявляем две целочисленные переменные `x` и `y`:

```
int x = 0, y = 0;
```

Эти переменные могут хранить только значения, имеющие тип `int`. То есть, целые числа в определенном диапазоне значений.

Говоря простым языком: *переменная* — это коробка, в которую мы можем что-то положить и из которой потом вытащить. Поскольку таких коробок может быть много, то каждая коробка подписывается (каждая переменная имеет уникальное имя внутри программы). Коробки могут быть разного размера. Например, необходимо хранить яблоко и арбуз. Согласитесь, что размеры яблока и арбуза отличаются. Чтобы поместить арбуз, мы должны взять соответствующего размера коробку. Таким образом, тип данных при объявлении переменной задает, какого размера коробку подготовить и что мы туда будем класть. Кроме того, в одну коробку мы можем положить только один предмет. Если нам нужно положить несколько яблок, то мы должны взять уже ящик (который в языке программирования называется *массивом*) и складывать туда коробки с яблоками.

Что будет, если вместо целого числа мы введем в окне консоли, например, строку, не содержащую число? В этом случае метод `nextInt()` не сможет преобразовать строку в число, и программа аварийно завершится. Обработку таких ошибок мы будем рассматривать далее в этой книге. А пока набирайте только целые числа!

1.8. Получение данных из командной строки

Передать данные можно в командной строке после названия файла. Получить эти данные в программе позволяет параметр `args` в методе `main()`. Квадратные скобки после типа `String` означают, что доступен массив строк (массив объектов типа `String`). Чтобы получить количество аргументов, переданных в командной строке, следует воспользоваться свойством `length`. Рассмотрим получение данных из командной строки на примере (листинг 1.4).

Листинг 1.4. Получение данных из командной строки. Вариант 1

```
public class MyClass {
   public static void main(String[] args) {
      for (int i = 0; i < args.length; i++) {
         System.out.println(args[i]);
      }
   }
}
```

Компилируем программу и запускаем из командной строки. Для запуска программы вводим команду:

```
C:\book>java MyClass string1 string2
```

В этой команде мы передаем программе `MyClass` некоторые данные (`string1 string2`). Результат выполнения программы будет выглядеть так:

```
string1
string2
```

Для перебора элементов массива можно также использовать другой синтаксис цикла `for` (листинг 1.5).

Листинг 1.5. Получение данных из командной строки. Вариант 2

```
public class MyClass {
   public static void main(String[] args) {
      for (String s: args) {
         System.out.println(s);
      }
   }
}
```

В этом случае внутри круглых скобок объявляется переменная `s`, имеющая тип `String` (текстовая строка), а после двоеточия указывается массив `args` (это переменная, которая объявлена в методе `main()`). На каждой итерации цикла из массива строк берется одна строка и присваивается переменной `s`. И так до тех пор, пока не будут перебраны все элементы массива.

1.9. Преждевременное завершение выполнения программы

В некоторых случаях может возникнуть условие, при котором дальнейшее выполнение программы лишено смысла. Тогда лучше вывести сообщение об ошибке и прервать выполнение программы досрочно. Для этого предназначен метод `exit()` из класса `System`. Формат метода:

```
public static void exit(int status)
```

В качестве параметра метод принимает число, которое является статусом завершения. Число `0` означает нормальное завершение программы, а любое другое число — некорректное завершение. Это число передается операционной системе.

В качестве примера произведем деление двух целых чисел, введенных пользователем, и выведем результат. При этом обработаем деление на нуль (листинг 1.6).

Листинг 1.6. Преждевременное завершение выполнения программы

```
import java.util.Scanner;

public class MyClass {
   public static void main(String[] args) {
      Scanner in = new Scanner(System.in);
      int x = 0, y = 0;
      System.out.print("x = ");
      x = in.nextInt();
      System.out.print("y = ");
      y = in.nextInt();
      if (y == 0) {
         System.out.println("Нельзя делить на 0");
         System.exit(1); // Завершаем выполнение программы
      }
      System.out.println("Результат деления = " + (x / y));
   }
}
```

В этом примере вместо метода `exit()` можно было воспользоваться инструкцией `return`, т. к. завершение программы выполнялось внутри метода `main()`. Однако в больших программах основная часть кода расположена вне метода `main()`, и в этом случае инструкцией `return` для завершения всей программы уже не обойтись. Попробуйте заменить инструкцию:

```
System.exit(1); // Завершаем выполнение программы
```

следующей:

```
return; // Завершаем выполнение программы
```

Код из листинга 1.6 содержит новую для нас инструкцию — условный оператор `if`. С помощью этого оператора мы можем проверить значение и, если оно соответствует условию, выполнить инструкции внутри блока (внутри фигурных скобок). Условие указывается внутри круглых скобок. Здесь мы проверяем значение переменной `y` равенству значению `0` с помощью оператора сравнения `==` (равно). Операторы сравнения возвращают либо значение `true` (истина), либо значение `false` (ложь). Если сравнение вернуло значение `true`, то будут выполнены инструкции, расположенные внутри фигурных скобок, а если `false` — то инструкции пропускаются.