Ильдар Хабибуллин

bhv

Java 7

- Принципы объектно-ориентированного программирования
- Пакеты классов и интерфейсы Java SE 7
- Библиотеки Swing и Java 2D, NIO2
- Апплеты, графика, звук и печать
- Технологии Servlet, JSP, JSTL, JSF, XML
- Около 200 законченных программ

Наиболее полное руководство

В ПОДЛИННИКЕ®

УДК 681.3.06
ББК 32.973.26-018.2
Х12

Хабибуллин И. Ш.

Х12 Java 7. — СПб.: БХВ-Петербург, 2012. — 768 с.: ил. — (В подлиннике)

ISBN 978-5-9775-0735-6

Рассмотрено все необходимое для разработки, компиляции, отладки и запуска приложений Java. Изложены практические приемы использования как традиционных, так и новейших конструкций объектно-ориентированного языка Java, графической библиотеки классов Swing, расширенной библиотеки Java 2D, работа со звуком, печать, способы русификации программ. Приведено полное описание нововведений Java SE 7: двоичная запись чисел, строковые варианты разветвлений, "ромбовидный оператор", NIO2, новые средства многопоточности и др. Дано подробное изложение последней версии сервлетов, технологии JSP и библиотек тегов JSTL. Около двухсот законченных программ иллюстрируют рассмотренные приемы программирования. Приведена подробная справочная информация о классах и методах Core Java API.

Для программистов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Екатерина Капалыгина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 31.08.11.
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 61,92.
Тираж 1800 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0735-6

Оглавление

ГЛАВА 1

Встроенные типы данных, операции над ними

Приступая к изучению нового языка, полезно поинтересоваться, какие исходные данные могут обрабатываться средствами этого языка, в каком виде их можно задавать и какие стандартные средства обработки данных заложены в язык. Это довольно скучное занятие, поскольку в каждом развитом языке программирования множество типов данных и еще больше правил их использования. Однако несоблюдение этих правил приводит к появлению скрытых ошибок, обнаружить которые иногда бывает очень трудно. Ну что же, в каждом ремесле приходится сначала "играть гаммы", не можем от этого уйти и мы.

Все правила языка Java исчерпывающе изложены в его спецификации, сокращенно называемой JLS (Java Language Specification), местоположение которой указано во *введении*. Иногда, чтобы понять, как выполняется та или иная конструкция языка Java, приходится обращаться к спецификации, но, к счастью, это бывает редко: правила языка Java достаточно просты и естественны.

В этой главе перечислены примитивные типы данных, операции над ними, операторы управления и показаны "подводные камни", которых следует избегать при их использовании. Но начнем, по традиции, с простейшей программы.

Первая программа на Java

По давней традиции, восходящей к языку C, учебники по языкам программирования начинаются с программы "Hello, World!". Не будем нарушать эту традицию. В листинге 1.1 приведена подобная программа. Она написана в самом простом виде, какой только возможен на языке Java.

Листинг 1.1. Первая программа на языке Java

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

Вот и все, только пять строчек! Но даже на этом простом примере можно заметить целый ряд существенных особенностей языка Java.

- Всякая программа, написанная на языке Java, представляет собой один или несколько классов, в этом простейшем примере только один *класс* (class).
- Начало класса отмечается служебным словом `class`, за которым следует имя класса, выбираемое произвольно, в данном случае это имя `HelloWorld`. Все, что содержится в классе, записывается в фигурных скобках и составляет *тело класса* (class body).
- Все действия в программе производятся с помощью методов обработки информации, коротко говорят просто *метод* (method). Методы используются в объектно-ориентированных языках вместо функций, применяемых в процедурных языках.
- Методы различаются по именам и параметрам. Один из методов обязательно должен называться `main`, с него начинается выполнение программы. В нашей простейшей программе только один метод, а значит, имя его `main`.
- Как и положено функции, метод всегда выдает в результате (чаще говорят *возвращает* (returns)) только одно значение, тип которого обязательно указывается перед именем метода. Метод может и не возвращать никакого значения, играя роль процедуры. Так и есть в нашем случае. Тогда вместо типа возвращаемого значения записывается слово `void`, как это и сделано в примере.
- После имени метода в скобках через запятую перечисляются *параметры* (parameters) метода. Для каждого параметра указывается его тип и, через пробел, имя. У метода `main()` только один параметр, его тип — массив, состоящий из строк символов. Строка символов — это встроенный в Java API тип `String`, а квадратные скобки — признак массива. Имя параметра может быть произвольным, в примере выбрано имя `args`.
- Перед типом возвращаемого методом значения могут быть записаны *модификаторы* (modifiers). В примере их два: слово `public` означает, что этот метод доступен отовсюду; слово `static` обеспечивает возможность вызова метода `main()` в самом начале выполнения программы. Модификаторы, вообще говоря, необязательны, но для метода `main()` они необходимы.

> ***Замечание***
>
> В тексте этой книги после имени метода ставятся скобки, чтобы подчеркнуть, что это имя метода, а не простой переменной.

- Все, что содержит метод, *тело метода* (method body), записывается в фигурных скобках.

Единственное действие, которое выполняет метод `main()` в нашем примере, заключается в вызове другого метода со сложным именем `System.out.println` и передаче ему на обработку одного аргумента — текстовой константы `"Hello, XXI Century World!"`. Текстовые константы записываются в кавычках, которые являются только ограничителями и не входят в текст.

Составное имя `System.out.println` означает, что в классе `System`, входящем в Java API, определяется переменная с именем `out`, содержащая экземпляр одного из классов Java API, класса `PrintStream`, в котором есть метод `println()`. Все это станет ясно позднее, а пока просто будем писать это длинное имя.

Действие метода `println()` заключается в выводе заданного ему аргумента в выходной поток, связанный обычно с выводом на экран текстового терминала, в окно **MS-DOS Prompt**, **Command Prompt** или **Xterm** в зависимости от вашей системы. После вывода курсор переходит на начало следующей строки экрана, на что указывает окончание `ln`, само слово `println` — сокращение слов print line. В составе Java API есть и метод `print()`, оставляющий курсор в конце выведенной строки. Разумеется, это прямое влияние языка Pascal.

Сильное влияние языка C привело к появлению в Java SE 5 (Java Standard Edition) метода `System.out.printf()`, очень похожего на одноименную функцию языка C. Мы подробно опишем этот метод в *главе 23*, но желающие могут ознакомиться с ним прямо сейчас.

Сделаем сразу важное замечание. Язык Java различает строчные и прописные буквы, имена `main`, `Main`, `MAIN` различны с "точки зрения" компилятора Java. В примере важно писать `String`, `System` с заглавной буквы, а `main` — со строчной. Но внутри текстовой константы неважно, писать `Century` или `century`, компилятор вообще не "смотрит" на текст в кавычках, разница будет видна только на экране.

> ***Замечание***
>
> Язык Java различает прописные и строчные буквы.

В именах нельзя оставлять пробелы. Свои имена можно записывать как угодно, можно было бы дать классу имя `helloworld` или `helloWorld`, но между Java-программистами заключено соглашение, называемое "Code Conventions for the Java Programming Language", хранящееся по адресу **http://www.oracle.com/technetwork/java/codeconv-138413.html**. Вот несколько пунктов этого соглашения:

- имена классов начинаются с прописной (заглавной) буквы; если имя содержит несколько слов, то каждое слово начинается с прописной буквы;
- имена методов и переменных начинаются со строчной буквы; если имя содержит несколько слов, то каждое следующее слово начинается с прописной буквы;
- имена констант записываются полностью прописными буквами; если имя состоит из нескольких слов, то между ними ставится знак подчеркивания.

Конечно, эти правила необязательны, хотя они и входят в JLS, п. 6.8, но сильно облегчают понимание кода и придают программе характерный для Java стиль.

Стиль определяют не только имена, но и размещение текста программы по строкам, например расположение фигурных скобок: оставлять ли открывающую фигурную скобку в конце строки с заголовком класса или метода или переносить на следующую строку? Почему-то этот пустячный вопрос вызывает ожесточенные споры, некоторые средства разработки даже предлагают выбрать определенный стиль расстановки фигурных скобок. Многие фирмы устанавливают свой внутрифирменный стиль. В книге мы постараемся следовать стилю "Code Conventions" и в том, что касается разбиения текста программы на строки (компилятор же рассматривает всю программу как одну длинную строку, для него программа — это просто последовательность символов), и в том, что касается отступов (indent) в тексте.

Итак, программа написана в каком-либо текстовом редакторе, например в Блокноте (Notepad), emacs или vi. Теперь ее надо сохранить в файле в текстовом, но не в графи-

ческом формате. Имя файла должно в точности совпадать с именем класса, содержащего метод `main()`. Данное правило очень желательно выполнять. При этом система исполнения Java будет быстро находить метод `main()` для начала работы, просто отыскивая класс, совпадающий с именем файла. Расширение имени файла должно быть java.

> ***Совет***
>
> Называйте файл с программой именем класса, содержащего метод `main()`, соблюдая регистр букв.

В нашем примере сохраним программу в файле с именем HelloWorld.java в текущем каталоге. Затем вызовем компилятор, передавая ему имя файла в качестве аргумента:

```
javac HelloWorld.java
```

Компилятор создаст файл с байт-кодами, даст ему имя HelloWorld.class и запишет этот файл в текущий каталог.

Осталось вызвать интерпретатор байт-кодов, передав ему в качестве аргумента имя класса (а не файла!):

```
java HelloWorld
```

На экране появится строка:

```
Hello, XXI Century World!
```

> ***Замечание***
>
> Не указывайте расширение class при вызове интерпретатора.

На рис. 1.1 показано, как все это выглядит в окне **Command Prompt** операционной системы MS Windows 2003.

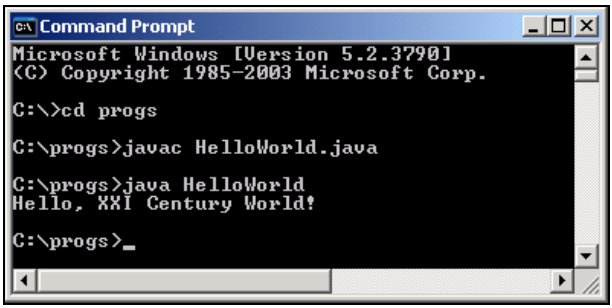

Рис. 1.1. Окно **Command Prompt**

При работе в какой-либо интегрированной среде, например Eclipse или NetBeans, все эти действия вызываются выбором соответствующих пунктов меню или "горячими" клавишами — единых правил здесь нет.

Комментарии

В текст программы можно вставить комментарии, которые компилятор не будет учитывать. Они очень полезны для пояснений по ходу программы. В период отладки можно выключать из действий один или несколько операторов, пометив их символами

комментария, как говорят программисты, "закомментировав" их. Кроме того, некоторые программы, работающие с Java, извлекают из комментариев полезные для себя сведения.

Комментарии вводятся таким образом:

- за двумя наклонными чертами, написанными подряд `//`, без пробела между ними, начинается комментарий, продолжающийся до конца строки;
- за наклонной чертой и звездочкой `/*` начинается комментарий, который может занимать несколько строк, до звездочки и наклонной черты `*/` (без пробелов между этими знаками);
- за наклонной чертой и двумя звездочками `/**` начинается комментарий, который может занимать несколько строк, до звездочки и наклонной черты `*/`. Из таких комментариев формируется документация.

Комментарии очень удобны для чтения и понимания кода, они превращают программу в документ, описывающий ее действия. Программу с хорошими комментариями называют *самодокументированной*. Поэтому в Java и введены комментарии третьего типа, а в состав JDK включена утилита — программа `javadoc`, извлекающая эти комментарии в отдельные файлы формата HTML и создающая гиперссылки между ними. В такой комментарий кроме собственно комментария можно вставить указания программе `javadoc`, которые начинаются с символа `@`.

Именно так создается документация к JDK.

Добавим комментарии к нашему примеру (листинг 1.2).

Листинг 1.2. Первая программа с комментариями

```
/**
* Разъяснение содержания и особенностей программы...
* @author Имя Фамилия (автора)
* @version 1.0 (это версия программы)
*/
class HelloWorld{   // HelloWorld — это только имя
  // Следующий метод начинает выполнение программы
  public static void main(String[] args){    // args не используются
    /* Следующий метод просто выводит свой аргумент
     * на экран дисплея */
    System.out.println("Hello, XXI Century World!");
    // Следующий вызов закомментирован,
    // метод не будет выполняться
    // System.out.println("Farewell, XX Century!");
  }
}
```

Звездочки в начале строк не имеют никакого значения, они написаны просто для выделения комментария. Пример, конечно, перегружен пояснениями (это плохой стиль), здесь просто показаны разные формы комментариев.

Аннотации

Обратите внимание на комментарий, приведенный в начале листинга 1.2. В него вставлены указания-теги `@author` и `@version` утилите `javadoc`. Просматривая текст этого комментария и встретив какой-либо из тегов, утилита `javadoc` выполнит предписанные тегом действия. Например, тег `@see` предписывает сформировать гиперссылку на другой документ HTML, а тег `@deprecated`, записанный в комментарий перед методом, вызовет пометку этого метода в документации как устаревшего.

Идея давать утилите предписания с помощью тегов оказалась весьма плодотворной. Кроме `javadoc` были написаны другие утилиты и целые программные продукты, которые вводят новые теги и используют их для своих целей. Например, программа XDoclet может автоматически создавать различные конфигурационные файлы, необходимые для работы сложных приложений. Разработчику достаточно вставить в свою программу комментарии вида `/**...*/` с тегами специального вида и запустить утилиту Xdoclet, которая сгенерирует все необходимые файлы.

Использование таких утилит стало общепризнанной практикой, и, начиная с пятой версии Java SE, было решено ввести прямо в компилятор возможность обрабатывать теги, которые получили название *аннотаций*. Аннотации записываются не внутри комментариев вида `/**...*/`, а непосредственно в том месте, где они нужны. Например, после того как мы запишем непосредственно перед заголовком какого-либо метода аннотацию `@Deprecated`, компилятор будет выводить на консоль предупреждение о том, что этот метод устарел и следует воспользоваться другим методом. Обычно замена указывается тут же, в этом же комментарии.

Несколько аннотаций, количество которых увеличивается с каждой новой версией JDK, объявлено прямо в компиляторе. Ими можно пользоваться без дополнительных усилий. Мы будем вводить их по мере надобности. Кроме них разработчик может объявить и использовать в своем приложении свои аннотации. Как это делается, рассказано в *главе 3*.

Константы

В языке Java можно записывать константы различных типов в разных видах. Форма записи констант почти полностью заимствована из языка C. Перечислим все разновидности констант.

Целые

Целые константы можно записывать в четырех системах счисления:

- в привычной для нас десятичной форме: `+5`, `-7`, `12345678`;
- в двоичной форме, начиная с нуля и латинской буквы `b` или `B`: `0b1001`, `0B11011`;
- в восьмеричной форме, начиная с нуля: `027`, `-0326`, `0777` (в записи таких констант недопустимы цифры 8 и 9);

> ***Замечание***
>
> Целое число, начинающееся с нуля, трактуется как записанное в восьмеричной форме, а не в десятичной.

- в шестнадцатеричной форме, начиная с нуля и латинской буквы `x` или `X`: `0xff0a`, `0xFC2D`, `0X45a8`, `0X77FF` (здесь строчные и прописные буквы не различаются).

Для улучшения читаемости группы цифр в числе можно разделять знаком подчеркивания: `1_001_234`, `0xFC_2D`.

Целые константы хранятся в оперативной памяти в формате типа `int` (см. далее).

В конце целой константы можно записать латинскую букву "L" (прописную `L` или строчную `l`), тогда константа будет сохраняться в длинном формате типа `long` (см. далее): `+25L`, `-037l`, `0xffL`, `0XDFDFl`.

> ***Совет***
>
> Не используйте при записи длинных целых констант строчную латинскую букву `l`, ее легко спутать с единицей.

Действительные

Действительные константы записываются только в десятичной системе счисления в двух формах:

- с фиксированной точкой: `37.25`, `-128.678967`, `+27.035`;
- с плавающей точкой: `2.5e34`, `-0.345e-25`, `37.2E+4`; можно писать строчную или прописную латинскую букву `E`; пробелы и скобки недопустимы.

В конце действительной константы можно поставить букву `F` или `f`, тогда константа будет сохраняться в оперативной памяти в формате типа `float` (см. далее): `3.5f`, `-45.67F`, `4.7e-5f`. Можно приписать и букву `D` (или `d`): `0.045D`, `-456.77889d`, означающую тип `double`, но это излишне, поскольку действительные константы и так хранятся в формате типа `double`.

Символы

Одиночные символы записываются в апострофах, чтобы отличить их от имен переменных. Для записи символов используются следующие формы:

- печатные символы, записанные на клавиатуре, просто записываются в апострофах (одинарных кавычках): `'a'`, `'N'`, `'?'`;
- управляющие и специальные символы записываются в апострофах с обратной наклонной чертой, чтобы отличить их от обычных символов:
  - `'\n'` — символ перевода строки LF (Line Feed) с кодом ASCII 10;
  - `'\r'` — символ возврата каретки CR (Carriage Return) с кодом 13;
  - `'\f'` — символ перевода страницы FF (Form Feed) с кодом 12;
  - `'\b'` — символ возврата на шаг BS (Backspace) с кодом 8;
  - `'\t'` — символ горизонтальной табуляции HT (Horizontal Tabulation) с кодом 9;
  - `'\\'` — обратная наклонная черта;
  - `'\"'` — кавычка;
  - `'\''` — апостроф;

- код любого символа с десятичной кодировкой от 0 до 255 можно задать, записав его не более чем тремя цифрами в восьмеричной системе счисления в апострофах после обратной наклонной черты: `'\123'` — буква `S`, `'\346'` — буква `ж` в кодировке CP1251. Нет смысла использовать эту форму записи для печатных и управляющих символов, перечисленных в предыдущем пункте, поскольку компилятор сразу же переведет восьмеричную запись в указанную ранее форму. Наибольший восьмеричный код `'\377'` — десятичное число 255;
- код любого символа в кодировке Unicode набирается в апострофах после обратной наклонной черты и латинской буквы `u` четырьмя шестнадцатеричными цифрами: `'\u0053'` — буква `S`, `'\u0416'` — буква `Ж`.

Символы хранятся в формате типа `char` (см. далее).

> ***Примечание***
>
> Прописные русские буквы в кодировке Unicode занимают диапазон от `'\u0410'` — заглавная буква `А`, до `'\u042F'` — заглавная `Я`, строчные буквы от `'\u0430'` — `а`, до `'\u044F'` — `я`.

В какой бы форме ни записывались символы, компилятор переводит их в Unicode, включая и исходный текст программы.

> ***Замечание***
>
> Компилятор и исполняющая система Java работают только с кодировкой Unicode.

Строки

Строки символов заключаются в кавычки. Управляющие символы и коды записываются в строках точно так же, с обратной наклонной чертой, но, разумеется, без апострофов, и оказывают то же действие. Строки могут располагаться только на одной строке исходного кода, нельзя открывающую кавычку поставить на одной строке, а закрывающую — на следующей.

Вот некоторые примеры:

```
"Это строка\nс переносом"
"\"Зубило\" — Чемпион!"
```

> ***Замечание***
>
> Строки символов нельзя начинать на одной строке исходного кода, а заканчивать на другой.

Для строковых констант определена операция сцепления, обозначаемая плюсом. Запись

```
"Сцепление " + "строк"
```

дает в результате строку `"Сцепление строк"`. Обратите внимание на то, что между сцепляемыми строками не вставлены никакие дополнительные символы. Пробел между ними принадлежал первой строке.

Чтобы записать длинную строку в виде одной строковой константы, надо после закрывающей кавычки на первой и следующих строках поставить плюс (+); тогда компилятор соберет две (или более) строки в одну строковую константу, например:

```
"Одна строковая константа, записанная " +
"на двух строках исходного текста"
```