# Программирование на языке Java. Введение в язык Java

Картузов А.В.

Исходный файл на языке Java—это текстовый файл, содержащий в себе одно или несколько описаний классов. Транслятор Java предполагает, что исходный текст программ хранится в файлах с расширениями Java. Получаемый в процессе трансляции код для каждого класса записывается в отдельном выходном файле, с именем совпадающем с именем класса, и расширением class.

Прежде всего, в этой главе мы напишем, оттранслируем, и запустим каноническую программу "Hello World". После этого мы рассмотрим все существенные лексические элементы, воспринимаемые Java-транслятором: пробелы, комментарии, ключевые слова, идентификаторы, литералы, операторы и разделители. К концу главы вы получите достаточно информации для того чтобы самостоятельно ориентироваться в хорошей Java-программе.

## 1. Hello World

Итак, вот ваша первая Java-программа:

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

Для того, чтобы поработать с приведенными в книге примерами вам нужно получить по сети из Sun Microsystems и установить Java Developers Kit—пакет для разработки Java-приложений (http://java.sun.com/products/jdk).

Язык Java требует, чтобы весь программный код был заключен внутри поименованных классов. Приведенный выше текст примера надо записать в файл HelloWorld.java. Обязательно проверьте соответствие прописных букв в имени файла тому же в названии содержащегося в нем класса. Для того, чтобы оттранслировать этот пример необходимо запустить транслятор Java—javac, указав в качестве параметра имя файла с исходным текстом:

```
C: \> javac HelloWorld.Java
```

Транслятор создаст файл HelloWorld.class с независимым от процессора байт-кодом нашего примера. Для того, чтобы исполнить полученный код, необходимо иметь среду времени выполнения языка Java (в нашем случае это программа java), в которую надо загрузить новый класс для исполнения. Подчеркнем, что указывается имя класса, а не имя файла, в котором этот класс содержится.

```
C: > java HelloWorld
Hello World
```

Полезного сделано мало, однако мы убедились, что установленный Java-транслятор и среда времени выполнения работают.

# 2. Шаг за шагом

Конечно, HelloWorld—это тривиальный пример. Однако даже такая простая программа новичку в языке Java может показаться пугающе сложной, поскольку она знакомит вас с массой новых понятий и деталей синтаксиса языка Давайте внимательно пройдемся по каждой строке нашего первого примера, анализируя те элементы, из которых состоит Java-программа.

## 2.1. Строка 1

```
class HelloWorld {
```

В этой строке использовано зарезервированное слово class. Оно говорит транслятору, что мы собираемся описать новый класс. Полное описание класса располагается между открывающей фигурной скобкой в первой строке и парной ей закрывающей фигурной

скобкой в строке 5. Фигурные скобки в Java используются точно так же, как в языках C и C++.

## 2.2. Строка 2

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

Такая, на первый взгляд, чрезмерно сложная строка примера является следствием важного требования, заложенного при разработке языка Java. Дело в том, что в Java отсутствуют глобальные функции. Поскольку подобные строки будут встречаться в большинстве примеров первой части книги, давайте пристальнее рассмотрим каждый элемент второй строки.

### 2.2.1. public

Разбивая эту строку на отдельные лексемы, мы сразу сталкиваемся с ключевым словом public. Это—модификатор доступа, который позволяет программисту управлять видимостью любого метода и любой переменной. В данном случае модификатор доступа public означает, что метод main виден и доступен любому классу. Существуют еще 2 указателя уровня доступа—private и protected, с которыми мы более детально познакомимся в главе 8.

### 2.2.2. static

Следующее ключевое слово—static. С помощью этого слова объявляются методы и переменные класса, используемые для работы с классом в целом. Методы, в объявлении которых использовано ключевое слово static, могут непосредственно работать только с локальными и статическими переменными.

### 2.2.3. void

У вас нередко будет возникать потребность в методах, которые возвращают значение того или иного типа: например, int для целых значений, float—для вещественных или имя класса для типов данных, определенных программистом. В нашем случае нужно просто вывести на экран строку, а возвращать значение из метода main не требуется. Именно поэтому и был использован модификатор void. Более детально этот вопрос

обсуждается в главе 4.

### 2.2.4. main

Наконец, мы добрались до имени метода main. Здесь нет ничего необычного, просто все существующие реализации Java-интерпретаторов, получив команду интерпретировать класс, начинают свою работу с вызова метода main. Java-транслятор может оттранслировать класс, в котором нет метода main. А вот Java-интерпретатор запускать классы без метода main не умеет.

Все параметры, которые нужно передать методу, указываются внутри пары круглых скобок в виде списка элементов, разделенных символами ";" (точка с запятой). Каждый элемент списка параметров состоит из разделенных пробелом типа и идентификатора. Даже если у метода нет параметров, после его имени все равно нужно поставить пару круглых скобок. В примере, который мы сейчас обсуждаем, у метода main только один параметр, правда довольно сложного типа.

Элемент String args[] объявляет параметр с именем args, который является массивом объектов—представителей класса String. Обратите внимание на квадратные скобки, стоящие после идентификатора args. Они говорят о том, что мы имеем дело с массивом, а не с одиночным элементом указанного типа. Мы вернемся к обсуждению массивов в следующей главе, а пока отметим, что тип String—это класс. Более детально о строках мы поговорим в главе 9.

## 2.3. Строка 3

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

В этой строке выполняется метод println объекта out. Объект out объявлен в классе OutputStream и статически инициализируется в классе System. В главах 9 и 13 у вас будет шанс познакомиться с нюансами работы классов String и OutputStream.

Закрывающей фигурной скобкой в строке 4 заканчивается объявление метода main, а такая же скобка в строке 5 завершает объявление класса HelloWorld.

# 3. Лексические основы

Теперь, когда мы подробно рассмотрели минимальный Java-класс, давайте вернемся назад и рассмотрим общие аспекты синтаксиса этого языка. Программы на Java—это набор пробелов, комментариев, ключевых слов, идентификаторов, литеральных констант, операторов и разделителей.

## 3.1. Пробелы

Java—язык, который допускает произвольное форматирование текста программ. Для того, чтобы программа работала нормально, нет никакой необходимости выравнивать ее текст специальным образом. Например, класс HelloWorld можно было записать в двух строках или любым другим способом, который придется вам по душе. И он будет работать точно так же при условии, что между отдельными лексемами (между которыми нет операторов или разделителей) имеется по крайней мере по одному пробелу, символу табуляции или символу перевода строки.

## 3.2. Комментарии

Хотя комментарии никак не влияют на исполняемый код программы, при правильном использовании они оказываются весьма существенной частью исходного текста. Существует три разновидности комментариев: комментарии в одной строке, комментарии в нескольких строках и, наконец, комментарии для документирования. Комментарии, занимающие одну строку, начинаются с символов // и заканчиваются в конце строки. Такой стиль комментирования полезен для размещения кратких пояснений к отдельным строкам кода:

```
a = 42; // если 42—ответ, то каков же был вопрос?
```

Для более подробных пояснений вы можете воспользоваться комментариями, размещенными на нескольких строках, начав текст комментариев символами /* и закончив символами */ При этом весь текст между этими парами символов будет расценен как комментарий и транслятор его проигнорирует.

```
/*
* Этот код несколько замысловат...
* Попробую объяснить:
* ....
```

```
        */
```

Третья, особая форма комментариев, предназначена для сервисной программы javadoc, которая использует компоненты Java-транслятора для автоматической генерации документации по интерфейсам классов. Соглашение, используемое для комментариев этого вида, таково: для того, чтобы разместить перед объявлением открытого (public) класса, метода или переменной документирующий комментарий, нужно начать его с символов /** (косая черта и две звездочки). Заканчивается такой комментарий точно так же, как и обычный комментарий—символами */. Программа javadoc умеет различать в документирующих комментариях некоторые специальные переменные, имена которых начинаются с символа @. Вот пример такого комментария:

```
        /**
        * Этот класс умеет делать замечательные вещи. Советуем всякому, кто
        * захочет написать еще более совершенный класс, взять его в качестве
        * базового.
        * @see Java. applet. Applet
        * (c)author Patrick Naughton
        * @version 1. 2
        */
        class CoolApplet extends Applet { /**
        * У этого метода два параметра:
        * @param key—это имя параметра.
        * @param value—это значение параметра с именем key.
        */ void put (String key, Object value) {
```

## 3.3. Зарезервированные ключевые слова

Зарезервированные ключевые слова—это специальные идентификаторы, которые в языке Java используются для того, чтобы идентифицировать встроенные типы, модификаторы и средства управления выполнением программы. На сегодняшний день в языке Java имеется 59 зарезервированных слов (см. таблицу 2). Эти ключевые слова совместно с синтаксисом операторов и разделителей входят в описание языка Java. Они могут применяться только по назначению, их нельзя использовать в качестве идентификаторов для имен переменных, классов или методов.

| | | | | |
|---|---|---|---|---|
| abstract | boolean | break | byte | byvalue |
| case | cast | catch | char | class |

| const | continue | default | do | double |
|---|---|---|---|---|
| else | extends | false | final | finally |
| float | for | future | generic | goto |
| if | implements | import | inner | instanceof |
| int | interface | long | native | new |
| null | operator | outer | package | private |
| protected | public | rest | return | short |
| static | super | switch | syncronized | this |
| throw | throws | transient | true | try |
| var | void | volatile | while | |

**Таблица 1. Зарезервированные слова Java**

Отметим, что слова byvalue, cast, const, future, generic, goto, inner, operator, outer, rest, var зарезервированы в Java, но пока не используются Кроме этого, в Java есть зарезервированные имена методов (эти методы наследуются каждым классом, их нельзя использовать, за исключением случаев явного переопределения методов класса Object).

| clone | equals | finalize | getClass | hashCode |
|---|---|---|---|---|
| notify | notifyAll | toString | wait | |

**Таблица 2. Зарезервированные имена методов Java**

## 3.4. Идентификаторы

Идентификаторы используются для именования классов, методов и переменных. В качестве идентификатора может использоваться любая последовательность строчных и прописных букв, цифр и символов _ (подчеркивание) и $ (доллар). Идентификаторы не должны начинаться с цифры, чтобы транслятор не перепутал их с числовыми литеральными константами, которые будут описаны ниже. Java—язык, чувствительный к регистру букв. Это означает, что, к примеру, Value и VALUE—различные идентификаторы.

## 3.5. Литералы

Константы в Java задаются их литеральным представлением. Целые числа, числа с плавающей точкой, логические значения, символы и строки можно располагать в любом месте исходного кода. Типы будут рассмотрены в главе 4.

## 3.6. Целые литералы

Целые числа—это тип, используемый в обычных программах наиболее часто. Любое целочисленное значение, например, 1, 2, 3, 42—это целый литерал. В данном примере приведены десятичные числа, то есть числа с основанием 10—именно те, которые мы повседневно используем вне мира компьютеров. Кроме десятичных, в качестве целых литералов могут использоваться также числа с основанием 8 и 16—восьмеричные и шестнадцатиричные. Java распознает восьмеричные числа по стоящему впереди нулю. Нормальные десятичные числа не могут начинаться с нуля, так что использование в программе внешне допустимого числа 09 приведет к сообщению об ошибке при трансляции, поскольку 9 не входит в диапазон 0.. 7, допустимый для знаков восьмеричного числа. Шестнадцатиричная константа различается по стоящим впереди символам нуль-х (0х или 0X). Диапазон значений шестнадцатиричной цифры—0.. 15, причем в качестве цифр для значений 10.. 15 используются буквы от A до F (или от a до f). С помощью шестнадцатиричных чисел вы можете в краткой и ясной форме представить значения, ориентированные на использование в компьютере, например, написав Oxffff вместо 65535.

Целые литералы являются значениями типа int, которое в Java хранится в 32-битовом слове. Если вам требуется значение, которое по модулю больше, чем приблизительно 2 миллиарда, необходимо воспользоваться константой типа long. При этом число будет храниться в 64-битовом слове. К числам с любым из названных выше оснований вы можете приписать справа строчную или прописную букву L, указав таким образом, что данное число относится к типу long. Например, Ox7fffffffffffffL или 92233720368547758O7L—это значение, наибольшее для числа типа long.

## 3.7. Литералы с плавающей точкой

Числа с плавающей точкой представляют десятичные значения, у которых есть

дробная часть. Их можно записывать либо в обычном, либо экспоненциальном форматах. В обычном формате число состоит из некоторого количества десятичных цифр, стоящей после них десятичной точки, и следующих за ней десятичных цифр дробной части. Например, 2.0, 3.14159 и .6667—это допустимые значения чисел с плавающей точкой, записанных в стандартном формате. В экспоненциальном формате после перечисленных элементов дополнительно указывается десятичный порядок. Порядок определяется положительным или отрицательным десятичным числом, следующим за символом E или e. Примеры чисел в экспоненциальном формате: 6.022e23, 314159E-05, 2e+100. В Java числа с плавающей точкой по умолчанию рассматриваются, как значения типа double. Если вам требуется константа типа float, справа к литералу надо приписать символ F или f. Если вы любитель избыточных определений—можете добавлять к литералам типа double символ D или d. Значения используемого по умолчанию типа double хранятся в 64-битовом слове, менее точные значения типа float—в 32-битовых.

## 3.8. Логические литералы

У логической переменной может быть лишь два значения—true (истина) и false (ложь). Логические значения true и false не преобразуются ни в какое числовое представление. Ключевое слово true в Java не равно 1, а false не равно 0. В Java эти значения могут присваиваться только переменным типа boolean либо использоваться в выражениях с логическими операторами.

## 3.9. Символьные литералы

Символы в Java—это индексы в таблице символов UNICODE. Они представляют собой 16-битовые значения, которые можно преобразовать в целые числа и к которым можно применять операторы целочисленной арифметики, например, операторы сложения и вычитания. Символьные литералы помещаются внутри пары апострофов (' '). Все видимые символы таблицы ASCII можно прямо вставлять внутрь пары апострофов:—'a', 'z', '@'. Для символов, которые невозможно ввести непосредственно, предусмотрено несколько управляющих последовательностей.

| Управляющая последовательность | Описание |
|---|---|
| \ddd | Восьмеричный символ (ddd) |

| | |
|---|---|
| \uxxxx | Шестнадцатиричный символ UNICODE (xxxx) |
| \' | Апостроф |
| \" | Кавычка |
| \\ | Обратная косая черта |
| \r | Возврат каретки (carriage return) |
| \n | Перевод строки (line feed, new line) |
| \f | Перевод страницы (form feed) |
| \t | Горизонтальная табуляция (tab) |
| \b | Возврат на шаг (backspace) |

**Таблица 3. Управляющие последовательности символов**

## 3.10. Строчные литералы

Строчные литералы в Java выглядят точно также, как и во многих других языках—это произвольный текст, заключенный в пару двойных кавычек (""). Хотя строчные литералы в Java реализованы весьма своеобразно (Java создает объект для каждой строки), внешне это никак не проявляется. Примеры строчных литералов: "Hello World!"; "две\строки; \ А это в кавычках\"". Все управляющие последовательности и восьмеричные / шестнадцатиричные формы записи, которые определены для символьных литералов, работают точно так же и в строках. Строчные литералы в Java должны начинаться и заканчиваться в одной и той же строке исходного кода. В этом языке, в отличие от многих других, нет управляющей последовательности для продолжения строкового литерала на новой строке.

## 3.11. Операторы

Оператор—это нечто, выполняющее некоторое действие над одним или двумя аргументами и выдающее результат. Синтаксически операторы чаще всего размещаются между идентификаторами и литералами. Детально операторы будут рассмотрены в главе 5, их перечень приведен в таблице 3. 3.

| | | | |
|---|---|---|---|
| + | += | - | -= |

| | | | |
|---|---|---|---|
| * | *= | / | /= |
| \| | \|= | ^ | ^= |
| & | &= | % | %= |
| > | >= | < | <= |
| ! | != | ++ | — |
| >> | >>= | << | <<= |
| >>> | >>>= | && | \|\| |
| == | = | ~ | ?: |
| | instanceof | [ ] | |

**Таблица 4. Операторы языка Java**

## 3.12. Разделители

Лишь несколько групп символов, которые могут появляться в синтаксически правильной Java-программе, все еще остались неназваннами. Это—простые разделители, которые влияют на внешний вид и функциональность программного кода.

| Символы | Название | Для чего применяются |
|---|---|---|
| ( ) | круглые скобки | Выделяют списки параметров в объявлении и вызове метода, также используются для задания приоритета операций в выражениях, выделения выражений в операторах управления выполнением программы, и в операторах приведения типов. |
| { } | фигурные скобки | Содержат значения автоматически инициализируемых массивов, также используются для ограничения блока кода в |

| | | |
|---|---|---|
| | | классах, методах и локальных областях видимости. |
| [ ] | квадратные скобки | Используются в объявлениях массивов и при доступе к отдельным элементам массива. |
| ; | точка с запятой | Разделяет операторы. |
| , | запятая | Разделяет идентификаторы в объявлениях переменных, также используется для связи операторов в заголовке цикла for. |
| . | точка | Отделяет имена пакетов от имен подпакетов и классов, также используется для отделения имени переменной или метода от имени переменной. |

## 3.13. Переменные

Переменная—это основной элемент хранения информации в Java-программе. Переменная характеризуется комбинацией идентификатора, типа и области действия. В зависимости от того, где вы объявили переменную, она может быть локальной, например, для кода внутри цикла for, либо это может быть переменная экземпляра класса, доступная всем методам данного класса. Локальные области действия объявляются с помощью фигурных скобок.

### 3.13.1. Объявление переменной

Основная форма объявления переменной такова:

```
тип идентификатор [ = значение] [, идентификатор [ = значение 7...];
```

Тип—это либо один из встроенных типов, то есть, byte, short, int, long, char, float, double, boolean, либо имя класса или интерфейса. Мы подробно обсудим все эти типы в следующей главе. Ниже приведено несколько примеров объявления переменных различных типов. Обратите внимание на то, что некоторые примеры включают в себя

инициализацию начального значения. Переменные, для которых начальные значения не указаны, автоматически инициализируются нулем.

```
int a, b, c;
```

Объявляет три целых переменных a, b, c.

```
int d = 3, e, f = 5;
```

Объявляет еще три целых переменных, инициализирует d и f.

```
byte z = 22;
```

Инициализирует z. double pi = 3. 14159;

Объявляет число пи (не очень точное, но все таки пи).

```
char x = 'x';
```

Переменная x получает значение 'x'.

В приведенном ниже примере создаются три переменные, соответствующие сторонам прямоугольного треугольника, а затем с помощью теоремы Пифагора вычисляется длина гипотенузы, в данном случае числа 5, величины гипотенузы классического прямоугольного треугольника со сторонами 3-4-5.

```
class Variables {
        public static void main (String args []) {
                double a = 3;
                double b = 4;
                double c;
                c = Math.sqrt (a* a + b* b);
                System.out.println ("c = "+ c);
                }
        }
```

## 4. Ваш первый шаг

Мы уже многого достигли: сначала написали небольшую программу на языке Java и подробно рассмотрели, из чего она состоит (блоки кода, комментарии). Мы

познакомились со списком ключевых слов и операторов, чье назначение будет подробно объяснено в дальнейших главах. Теперь вы в состоянии самостоятельно различать основные части любой Java-программы и готовы к тому, чтобы приступить к чтению главы 4, в которой подробно рассматриваются простые типы данных.