Лекция 3

Введение в язык Java

Простейшая программа на Java

Простейшая программа на языке программирования Java выглядит примерно так:

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

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++.

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.

`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`.

`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.

`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`.

`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.

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`.

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

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.

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.

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
* ©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) {
```

Лексемы

Итак, были рассмотрены пробелы (в широком смысле этого слова, т.е. все символы, отвечающие за форматирование текста программы) и комментарии, применяемые для ввода пояснений к коду. С точки зрения программиста они

применяются для того, чтобы сделать программу более читаемой и понятной для дальнейшего развития.

С точки зрения компилятора, а точнее его части, отвечающей за лексический разбор, основная роль пробелов и комментариев - служить разделителями между лексемами, причем сами разделители далее отбрасываются и не влияют на компилированный код.

Виды лексем

Ниже перечислены все виды лексем в Java:

• идентификаторы (identifiers);
• ключевые слова (key words);
• литералы (literals);
• разделители (separators);
• операторы (operators).

Рассмотрим их по отдельности.

Идентификаторы

Идентификаторы - это имена, которые даются различным элементам языка для упрощения доступа к ним. Имена имеют пакеты, классы, интерфейсы, поля, методы, аргументы и локальные переменные (все эти понятия подробно рассматриваются в дальнейших главах).

Идентификаторы можно записывать символами Unicode, то есть, на любом удобном языке.

Длина имени не ограничена.

Идентификатор состоит из букв и цифр. Имя не может начинаться с цифры. Java-буквы, используемые в идентификаторах, включают в себя ASCII-символы A-Z (\u0041-\u005a), a-z (\u0061-\u007a), а также знаки подчеркивания _ (ASCII underscore, \u005f) и доллара $ (\u0024). Знак доллара используется только при автоматической генерации кода (чтобы исключить случайное совпадение имен), либо при использовании каких-либо старых библиотек, в которых допускались имена с этим символом. Java-цифры включают в себя обычные ASCII-цифры 0-9 (\u0030-\u0039). Для идентификаторов не допускаются совпадения с зарезервированными словами (это ключевые слова, булевские литералы true и false и null-литерал null). Конечно, если 2 идентификатора включают в себя разные буквы, которые одинаково выглядят (например, латинская и русская буквы A), то они считаются различными.

Ключевые слова

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

Ключевые слова goto и const зарезервированы, но не используются. Это сделано для того, чтобы компилятор мог правильно отреагировать на использование этих распространенных в других языках слов. Напротив, оба

булевских литерала true, false и null-литерал null часто считают ключевыми словами (возможно потому, что многие средства разработки подсвечивают их таким же образом), однако это именно литералы.

Литералы (константы)

Арифметические

Примеры арифметических констант

- 10
- 010 — это 8
- 0123 — это 83 (1*64 + 2*8 + 3)
- 0x10 — это 16
- 0x123 — это 291 (1*256 + 2*16 +3)
- 1e5 — это 100000
- 1.23e-3 — это 0.00123

Для указания типа константы применяются суффиксы: l (или L) — long, f (или F) — float, d (или D) — double. Например, 1L — единица, но типа long.

```
// float-литералы:
1f, 3.14F, 0f, 1e+5F
// double-литералы:
0., 3.14d, 1e-4, 31.34E45D
```

Логические литералы

Логические литералы — это true (истина) и false (ложь)

Строковые литералы

Записываются в двойных кавычках, например

"это строка"

Символьные литералы

Записываются в апострофах, например 'F', 'ш'.

В строковых и символьных литералах есть правила для записи специальных символов. Во-первых, есть набор предопределенных специальных символов. Это

- '\n' — конец строки (перевод строки)
- '\r' — возврат каретки
- '\t' — табуляция

и ряд других.

Java класс Math

Разработчику на Java доступно множество готовых (или библиотечных) классов и методов, полезных для использования в собственных программах. Наличие библиотечных решений позволяет изящно решать множество типовых задач.

Далее рассмотрим класс `Math`, содержащий различные математически функции. Рассмотрим некоторые из них:

`Math.abs(n)` — возвращает модуль числа n.

`Math.round(n)` — возвращает целое число, ближайшее к вещественному числу n (округляет n).

`Math.ceil(n)` — возвращает ближайшее к числу n справа число с нулевой дробной частью (например, `Math.ceil`(3.4) в результате вернёт 4.0).

`Math.cos(n)`, `Math.sin(n)`, `Math.tan(n)` — тригонометрические функции `sin`, `cos` и `tg` от аргумента *n*, указанного в радианах.

`Math.acos(n)`, `Math.asin(n)`, `Math.atan(n)` — обратные тригонометрические функции, возвращают угол в радианах.

`Math.toDegrees(n)` — возвращает градусную меру угла в *n* радианов.

`Math.toRadians(n)` — возвращает радианную меру угла в *n* градусов.

`Math.sqrt(n)` — возвращает квадратный корень из *n*.

`Math.pow(n, b)` — возвращает значение степенной функции *n* в степени *b*, основание и показатель степени могут быть вещественными.

`Math.log(n)` — возвращает значение натурального логарифма числа *n*.

`Math.log10(n)` — возвращает значение десятичного логарифма числа *n*.

Все перечисленные функции принимают вещественные аргументы, а тип возвращаемого значения зависит от типа аргумента и от самой функции.

Кроме функций в рассматриваемом классе имеются две часто используемых константы:

`Math.PI` — число «пи», с точностью в 15 десятичных знаков.

`Math.E` — число Неппера (основание экспоненциальной функции), с точностью в 15 десятичных знаков.

Примеры:

```
System.out.println(Math.abs(-2.33)); //выведет 2.33
System.out.println(Math.round(Math.PI)); //выведет 3
System.out.println(Math.round(9.5)); //выведет 10
System.out.println(Math.round(9.5-0.001)); //выведет 9
System.out.println(Math.ceil(9.4)); //выведет 10.0
double c = Math.sqrt(3*3 + 4*4);
System.out.println(c); //выведет гипотенузу треугольника с катетами 3 и 4
double s1 = Math.cos(Math.toRadians(60));
System.out.println(s1); //выведет косинус угла в 60 градусов
```

Псевдослучайные числа

В классе `Math` есть полезная функция без аргументов, которая позволяет генерировать псевдослучайные значения, т.е. при каждом вызове этой функции она будет возвращать новое значение, предсказать которое очень сложно (не вдаваясь в подробности можно сказать, что теоретически это всё-таки возможно, именно поэтому генерируемые функцией числа называются не случайными, а псевдослучайными).

Итак, `Math.random()` возвращает псевдослучайное вещественное число из промежутка [0;1).

Если требуется получить число из другого диапазона, то значение функции можно умножать на что-то, сдвигать и, при необходимости, приводить к целым числам.

Примеры:

```
System.out.println(Math.random()); //вещественное число из [0;1)
System.out.println(Math.random()+3); //вещественное число из [3;4)
```

```
System.out.println(Math.random()*5); //вещественное число из [0;5)
System.out.println((int)(Math.random()*5)); //целое число из [0;4]
System.out.println(Math.random()*5+3); //вещественное число из [3;8)
System.out.println((int)(Math.random()*5+3)); //целое число из [3;7]
System.out.println((int)(Math.random()*11) - 5); //целое число из [-5;5]
```

Псевдослучайные числа имеют серьёзнейшие практические приложения и используются, например, в криптографии.

Задания для самостоятельной работы

1. Создайте программу, которая вычислит выражение $20 \cdot \frac{1}{3} + 158^2$ и выведет результат на экран.
2. Создайте программу, которая вычислит выражение $\frac{14}{209} + 14 \cdot (29 - 13^2 + \frac{14}{3})$ и выведет результат на экран. При этом число 14 обязательно сохраните в отдельной переменной, выбрав для неё подходящий тип.
3. В переменной *n* хранится двузначное число. Создайте программу, вычисляющую и выводящую на экран сумму цифр *n*.
4. В переменной *n* хранится вещественное число с ненулевой дробной частью. Создайте программу, округляющую число *n* до ближайшего целого и выводящую результат на экран.
5. В переменной *n* хранится трёхзначное число. Создайте программу, вычисляющую и выводящую на экран сумму цифр *n*.
6. Вычислить и вывести на экран косинусы углов в 60, 45 и 40 градусов без использования функции `Math.toDegrees(n)`.
7. В переменных *a* и *b* лежат положительные длины катетов прямоугольного треугольника. Вычислить и вывести на экран площадь треугольника и его периметр.
8. Натуральное положительное число записано в переменную *n*. Определить и вывести на экран, сколько цифр в числе *n*.
9. В переменной *n* лежит некоторое вещественное число. Вычислить и вывести на экран значение функции «сигнум» от этого числа (-1, если число отрицательное; 0, если нулевое; 1 если, положительное).
10. Создайте программу, которая будет генерировать и выводить на экран вещественное псевдослучайное число из промежутка [-3;3).
11. Натуральное положительное число записано в переменную *n*. Создайте программу, которая будет генерировать и выводить на экран целое псевдослучайное число из отрезка [-*n*;*n*].
12. В переменные *a* и *b* записаны целые числа, при этом *b* больше *a*. Создайте программу, которая будет генерировать и выводить на экран целое псевдослучайное число из отрезка [*a*;*b*].