Язык Java

Осенний семестр 2023, бакалавриат, 3 курс

Лекция 2: Базовые конструкции

Содержание

1. Базовые операторы
    - Объявление и присваивание переменных
    - Условные операторы
    - Циклы
    - Методы (функции)
2. Переменные, литералы, выражения
3. Примитивные типы: `boolean`, `int`, `double`, ...
4. Ссылочные типы:
    - массивы
    - строковый тип `String``
    - классы-обертки над примитивными типами: `Integer`, `Double`, ...

Объявление и присваивание переменных

- Объявление переменной

```
int number;    // целочисленная переменная
String string; // строковая переменная
boolean flag;  // логическая переменная
```

- Оператор присваивания `<имя переменной> = <выражение>`

```
number = 1;
string = "Привет";
flag = false;
```

- Сочетание определения и присваивания

```
int number = 1;
```

Условный оператор if

```
if (number < 0) {
    string = "отрицательное число";
} else if (number > 0) {
    string = "положительное число";
} else {
    string = "ноль"
}
```

- Блоки `else if` и `else` необязательны.
- Блок `else if` может встречаться несколько раз (между блоками `if` и `else`)

Условный оператор switch

```
switch (number) {
    case 0:
        string = "ноль";
        break;
    case 1:
        string = "один";
        break;
    default:
        string = "не ноль и не один";
}
```

Условный оператор ?

```
string = (number == 0) ? "ноль" : "не ноль";
```

Может быть переписан как:

```
if (number == 0) {
    string = "ноль";
} else {
    string = "не ноль";
}
```

Многократное использование:

```
string = (number == 0) ? "ноль" : (number == 1) ? "один" : "не ноль и не один"
```

Цикл for

Пример: вычисление суммы 1 + 2 + ... + `n`

```
int sum = 0;
for (int i = 1; i <= n; i = i++) { // i++ -> i = i + 1
    sum += i;                      // sum = sum + i;
}
```

Вопрос: что вычислит эта программа при отрицательных `n`?

Другой вариант цикла `for` ("for-each") будет рассмотрен позже (см. массивы)

Цикл while

Пример: вычисление суммы чисел от 1 + 2 + ... + `n`

```
int sum = 0;
int i = 1;
while (i <= n) {
    sum += i;
    i++;
}
```

Вопрос: что вычислит эта программа при отрицательных `n`?

Цикл "do while"

Пример: вычисление суммы чисел от 1 + 2 + ... + `n`

```
int sum = 0;
int i = 1;
do {
    sum += i;
    i++;
} while (i <= n);
```

Вопрос: что вычислит эта программа при отрицательных `n`?

Методы (функции)

```
public class Main {

    public static void main(String[] args) { // метод, не возвращающий значение
        System.out.println(factorial(n));
    }

    public static int factorial(int n) { // метод, возвращающий значение типа int
        int result = 1;
        for (int i = 2; i <= n; i++) {
            result *= i;
        }
        return result;
    }
}
```

О ключевых словах `public`, `static` мы поговорим в разделе ООП.

Рекурсия

- $n! = n * (n - 1)!$
- $1! = 1$

```
int factorial(int n) {
    if (number == 1) {
        return 1;
    }
    return n * factorial(n - 1);
}
```

Вопрос: корректно ли реализован метод `factorial`?

Методы: упражнение

```
void swap(int i, int j) {
    int tmp = i;
    i = j;
    j = tmp;
}
```

Верно ли, что после выполнения нижеследующей программы переменная `x` принимает значение 2, а `y` принимает значение 1?

```
int x = 1;
int y = 2;
swap(x, y);
```

Соглашения об именах переменных

Имя переменной:

- состоит из символов `A`,..., `Z`, `a`, ..., `z`, `0`, ..., `9`, `_`
- зависит от регистра (`number` и `NUMBER` разные переменные)
- не может быть ключевым словом (`for`, `int`, ...)
- рекомендуется использовать "Camel case"-запись:
  - корректно: `name`, `firstName`, `dateOfBirth`
  - некорректно: `FirstName`, `first_name`

Литералы (константы)

- Целочисленные:
  - десятичные: `0`, `1`, ..., `42`, ...
  - двоичные: `0b101001`
  - шестнадцатеричные: `0x1a`, `0x1ABCDEF`
- Вещественные: `0.012`, `1e-10`, ...
- Логические: `true`, `false`
- Символы: `'a'`, `'\n'`
- Строки: `"привет"`
- `null`

Выражения

- Выражения строятся из переменных, литералов и операторов.
- Операторы в выражениях:
  - арифметические: `+`, `-`, `*`, `/`, `%`,...
  - сравнения: `==`, `!=`, `<`, `>`, `<=`, `>=`
  - логические:
    - `&&` (логическое И), `||` (логическое ИЛИ), `!` (логическое отрицание)
  - вызов метода: `"привет".length()`
  - побитовые: `&`, `|`, `^`, ...
- Примеры выражений:

```
int number = (1 + 2 * x) % 10; // арифметическое выражение
boolean condition = !((x == 1) && (y <= 2)) || (z > 0)
```

Примитивные типы данных

Примитивные типы:

- целочисленные: `byte`, `short`, `int`, `long`
- вещественные: `float`, `double`
- логический: `boolean`
- символьный: `char`

Свойства:

- Базовые элементы для построения более сложных типов (например, классов)
- Принимают значения из ограниченного диапазона
- Требуют для хранения в памяти ограниченное количество байт (до восьми)
- Значения хранятся в **стековой памяти**

Целочисленные типы

Тип	Размер	Диапазон значений
`byte`	1 байт	[-128, 127]
`short`	2 байта	[-32768, 32767]
`int`	4 байта	$\approx$ [-2.14 · 10^9, 2.14 * 10^9]
`long`	8 байтов	$\approx$ [-9.22 · 10^{18}, 9.22 * 10^{18}]

- Значение по умолчанию (неинициализированных переменных): `0`
- Операции: арифметические, побитовые, операции сравнения

Вещественные типы

Тип	Размер	Точность
`float`	4 байта	до 7 знаков после запятой
`double`	8 байтов	до 16 знаков после запятой

- Значение по умолчанию: `0.0`
- Операции: арифметические, операции сравнения

Операция деления

Для целочисленных типов `/` - частное при делении с остатком

```
int x = 1, y = 2;
double z = x / y; // 0.0
```

Исправление:

```
double z = (double)x / y; // 0.5
```

Логический тип

Тип	Размер	Значения
`boolean`	1 байт	`true`, `false`

- Значение по умолчанию: `false`
- Операции: логические, операции сравнения

Вопрос: Все ли "логично" на этом слайде?

Символьный тип

Тип	Размер	Значения	Смысл значения
`char`	2 байта	0...65535	Номер символа в Unicode

- Значение по умолчанию: `'\u0000'`
- Операции: как для целочисленных переменных

Ссылочные типы данных: примеры

- Классы:
  - Строковый класс `String`
  - Классы-обертки над примитивными типами `Integer`, `Boolean`, ...
  - Пользовательские классы
- Массивы
  - Одномерные массивы: `int[]`, `String[]`, ...
  - Двумерные массивы: `int[][]`, `String[][]`, ...

Ссылочные типы данных: свойства

- Количество памяти для хранения данных может быть неограничено (например, сколь угодно длинная строка)
- Данные хранятся в **динамической памяти (heap)**, а не в стеке
- Значение переменной: ссылка на участок памяти в heap'е (размером 4 байта)
- Значение по умолчанию: `null` (пустая ссылка)

Одномерные массивы: объявление и инициализация

- с помощью оператора `new`:

```
int[] a = new int[3];      // массив [0, 0, 0]
String[] b = new int[3];   // массив [null, null, null]
```

- перечислением элементов массива

```
int[] a = new int[] {1, 2, 3}; // массив [1, 2, 3]
int[] b = {1, 2, 3};           // более короткая запись
```

Одномерные массивы: чтение и запись элементов

```
int[] array = {10, 20, 30}
```

- Чтение элемента по индексу (нумеруются с 0)

```
int first = array[0]; // 10
int last = array[2];  // 30
int error = array[3]; // ArrayIndexOutOfBoundsException
```

- Запись элемента по индексу

```
array[1] = 1000; // array = [10, 1000, 30]
array[3] = 1000; // ArrayIndexOutOfBoundsException
```

Вопрос: Как записать четвертый элемент в массив `array`?

Обход одномерного массива

```
int[] array = {10, 20, 30}
```

- **Вариант 1**: цикл `for` по *индексам* элементов массива

```
for (int i = 0; i < array.length; i++) { // array.length – длина массива
    System.out.println(array[i]);
}
```

- **Вариант 2**: цикл `for` по *элементам* массива

```
for (int element : array) {
    System.out.println(element);
}
```

Сортировка одномерного массива

с помощью метода `Arrays.sort` (пакет `java.util`)

- по возрастанию:

```
int[] array = {20, 30, 10};
Arrays.sort(array); // сортировка "на месте": array = [10, 20, 30]
```

- по убыванию:

```
Integer[] array = {20, 30, 10}; // важно: с int[] будет ошибка компиляции
Arrays.sort(array, Collections.reverseOrder()); // array = [30, 20, 10]
```

Двумерные массивы: объявление и инициализация

- с помощью оператора `new`

```
int rows = 2; // число строк
int cols = 3; // число столбцов
int[][] matrix = new int[rows][cols];
```

- перечислением элементов

```
int[][] matrix = {{10, 20, 30},   // 1-я строка
                  {40, 50, 60}};  // 2-я строка
```

Двумерные массивы: чтение и запись элементов

```
int[][] matrix = {{10, 20, 30}, {40, 50, 60}};
```

- чтение элемента по индексам строки и столбца

```
int leftUp    = matrix[0][0]; // 10
int leftDown  = matrix[1][0]; // 40
int rightUp   = matrix[0][2]; // 30
int rightDown = matrix[1][2]; // 60
```

- запись элемента по индексам строки и столбца

```
matrix[0][0] = 1000;
```

Обход двумерного массива построчно сверху вниз

```
int[][] matrix = {{10, 20, 30}, {40, 50, 60}};
```

- **Вариант 1**: цикл `for` по *индексам* строк и столбцов

```
for (int rowIndex = 0; rowIndex < matrix.length; rowIndex++) {
    for (int colIndex = 0; colIndex < matrix[row].length; colIndex++) {
        System.out.println(matrix[rowIndex][colIndex]);
    }
}
```

- **Вариант 2**: цикл `for` по *строкам* и вложенный цикл по столбцам

```
for (int[] row : matrix) {
    for (int element : row) {
        System.out.println(element);
    }
}
```

Строки: объявление и инициализация

- с помощью строкового литерала

```
String greeting = "Привет";
```

- с помощью оператора `new`

```
char[] charArray = {'П', 'р', 'и', 'в', 'е', 'т'};
String greeting = new String(charArray);
```

Обход строк

```
String str = "Привет";
```

- длина строки

```
int length = str.length(); // 6
```

- чтение символа строки по индексу:

```
char firstLetter = str.chartAt(0); // 'П'
char lastLetter  = str.chartAt(5); // 'т'
```

- обход строки (по *индексам* символов)

```
for (int i = 0; i < str.length(); i++) {
  System.out.println(str.charAt(i));
}
```

Мутация (изменение содержимого) строки

- *невозможна*, так как строки неизменяемые (immutable)
- для изменения содержимого строки необходимо создать новую строку

Полезные методы для работы со строками

```
String str = "Привет";
```

- запись подстроки в новую строку

```
String prefix = str.substring(0, 2);  // "Пр"
String suffix = str.substring(2);     // "ивет"
String segment = str.substring(2, 4); // "ив"
```

- поиск подстроки

```
boolean startsWith = str.startsWith("Пр"); // true
boolean contains = str.contains("рив"); // true
boolean endsWith = str.contains("вет"); // true
```

- замена подстроки

```
String newString = string.replace("т", "тик"); // "Приветик"
```

Конвертация строк

- конвертация строки в массив символов

```
String str = "Привет";
char[] charArray = str.toCharArray(); // ['П', 'р', 'и', 'в', 'е', 'т']
```

- конвертация строки в целое число

```
int number = Integer.parseInt("200"); // 200
int error  = Integer.parseInt("abc");   // NumberFormatException
```

- конвертация целого числа в строку

```
String variant1 = String.valueOf(200); // "200"
String variant2 = 200 + "";            // "200"
```

Сравнение строк

- Напомним, что значение строковой переменной - это ссылка на содержимое строки в heap'e (грубо говоря, адрес)
- Оператор `==` сравнивает ссылки (адреса), а не содержимое строк
- Для посимвольного сравнения строк используется метод `equals`

```
String s1 = "ab";
String s2 = new String(new char[]{'a', 'b'}); // "ab"
boolean equalReferences = (s1 == s2);         // false
boolean equalContent = s1.equals(s2);         // true
```

Вопрос: чему будет равно значение выражения `"ab" == "ab"`?

Объединение (конкатенация) строк

- Оператор `+`

  ```
  String s = "При" + "вет"; // "Привет"
  ```

  Внимание: неэффективность при конкатенации большого количества строк

- Метод `String.join`
- Класс `StringBuilder` (пакет `java.lang`)

Упражнение: пусть дан длинный массив длинных строк. Реализуйте конкатенацию массива строк в одну строку: (a) с помощью оператора `+`; (b) используя метод `String.join`; (c) используя класс `StringBuilder`. Сравните быстродействие трех реализаций.

Классы-обертки над примитивными типами

Мотивация:

- Переменные могут принимать:
  - значения примитивного типа (например, `int`)
  - значение `null`, если неопределены
- Нужны для работы с шаблонными классами (списками, словарями)
- В отличие от примитивных типов, классы обертки содержат методы (например, `Integer.parse`)

Классы-обертки: перечисление

- `byte` -> `Byte`
- `short` -> `Short`
- `int` -> `Integer`
- `long` -> `Long`
- `float` -> `Float`
- `double` -> `Double`
- `boolean` -> `Boolean`
- `char` -> `Character`

Классы-обертки

Примеры:

```
Integer wrapperInt;
wrapperInt = 256;                  // autoboxing
int primitiveInt = wrapperInt; // unboxing
Integer sum = wrapperInt + primitiveInt;
```

Unboxing нулевой ссылки

```
Integer wrapperInt = null;
int primitiveInt = wrapperInt; // NullPointerException
```

Полезные методы:

```
byte maxValue = Byte.MAX_VALUE;                // 127
int bitCount = Integer.bitCount(4);            // 1, т.к. 4 = 0b100
boolean isLetter = Character.isLetter('a'); // true
```