Лекция 4. Простые типы данных и операторы.

NetCracker®

Переменные

Любая переменная имеет три характеристики:

- имя
- тип
- значение

Переменная может иметь инициализаторы:

```
int a;
int b = 3 + 2;
int c = b + 2;
int d = a = c;
```

Ключевое слово *final*

```
final double g = 9.81;
```

Целочисленные типы

Тип	Длинна (байты)	Диапазон значений
`byte`	1	-128 ... 127
`short`	2	-32,768 ... 32,767
`int`	4	-2,147,483,648 ... 2,147,483,647
`long`	8	-9,223,372,036,854,775,808 ... 9,223,372,036,854,775,807
`char`	2	'\u0000' ... '\uffff', или 0 ... 65,535

Операции числовыми аргументами

- **операции сравнения** (возвращают булево значение)
  - `<, <=, >, >=`
  - `==, !=`
- **числовые операции** (возвращают числовое значение)
  - унарные операции + и –
  - арифметические операции `+, -, *, %`
  - деление /
  - операции инкремента и декремента (в префиксной и постфиксной форме): ++ и `--`
  - операции битового сдвига `<<, >>, >>>`
  - битовые операции `~, &, |, ^`
- **оператор с условием** `condition ? true_value : false_value`
- **оператор приведения типов** `(type) value`
- **оператор конкатенации со строкой** +

Особенности целых чисел

```
int x = -2147483648; // Наименьшее возможное типа int
int y = -x;
System.out.println(y); // Результат: -2147483648

x = 300000;
System.out.println(x * x); // -194.313.216
```

```
          300.000
*         300.000
= 90.000.000.000

b 1.0100.1111.0100.0110.1011.0000.0100.0000.0000
i         0000.1011.1001.0100.1111.1011.1111.1111
+         0000.0000.0000.0000.0000.0000.0000.0001
=         0000.1011.1001.0100.1111.1100.0000.0000

= -194.313.216
```

Числа с плавающей точкой

Название	Длина (байты)	Диапазон значений
`float`	4	$3.402,823,47 \times 10^{38}$ $1.402,398,46 \times 10^{-45}$
`double`	8	$1.797,693,134,862,315,70 \times 10^{308}$ $4.940,656,458,412,465,44 \times 10^{-324}$

Операция	Результат	
`1.0/0.0`	`Double.POSITIVE_INFINITY`	+∞
`-1.0/0.0`	`Double.NEGATIVE_INFINITY`	-∞
`0.0/0.0`	`Double.NaN`	Не-число
`(1.0/0.0)*0.0`	`Double.NaN`	Не-число
`-0.0`		Отрицательный ноль

Overflow & underflow

Пример overflow:

```
 1e20f * 1e20f =  Infinity
-1e200 * 1e200 = -Infinity
```

Пример underflow:

```
System.out.println( 1e-40f / 1e10f);
System.out.println(-1e-300 / 1e100);
float f = 1e-6f;
System.out.println(f);
f += 0.002f;
System.out.println(f);
f += 3;
System.out.println(f);
f += 4000;
System.out.println(f);
```

```
 0.0
-0.0

0000.000001

0000.002001

0003.002001

4003.002000
```

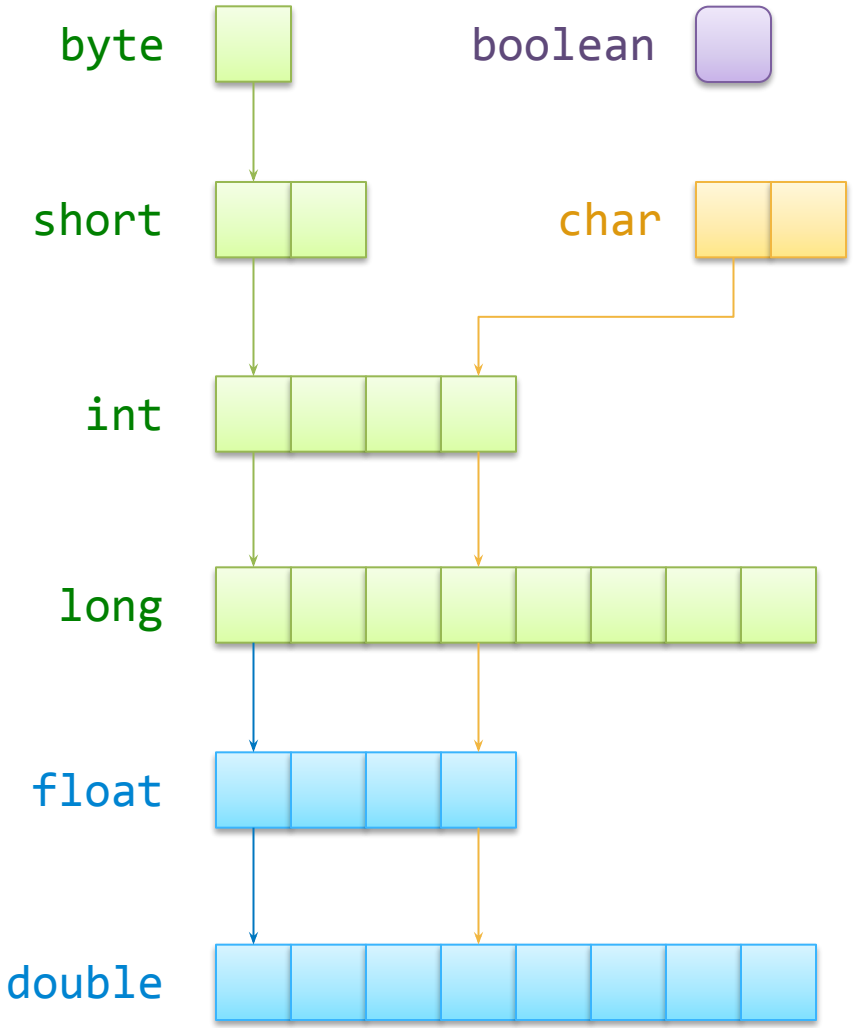
Операции над ссылочными значениями

- обращение к **полям и методам** объекта (оператор точка)
- оператор **instanceof** (возвращает булевское значение)
- операции **сравнения** == и != (возвращают булевское значение)
- оператор **приведения типов**
- оператор с **условием** ? :
- оператор **конкатенации** со строкой +

Приведение типов

- **тождественное** (identity);
- **расширение** примитивного типа (widening primitive);
- **сужение** примитивного типа (narrowing primitive);
- **расширение** объектного типа (widening reference);
- **сужение** объектного типа (narrowing reference);
- преобразование к **строке** (String);
- **запрещенные** преобразования (forbidden).

Расширение примитивных типов

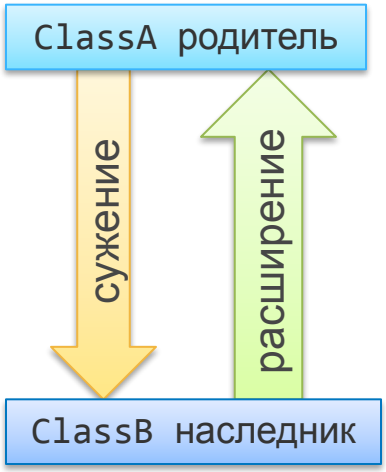


Приведение ссылочных типов

Расширяющие преобразования:

- **от класса B к классу A, если B наследуется от A** (частный случай – преобразование от любого ссылочного типа к Object);
- **от null-типа к любому объектному типу.**

```
ClassA a = classB
```
если **`classB instanceof ClassA`**



наследует ≡ является

Сужающие преобразования:

- **от класса A к классу B, если B наследуется от A** (важным частным случаем является сужение типа Object до любого другого ссылочного типа).

```
ClassB b = (ClassB) classA
```
если **`classA instanceof ClassB`**

Объявление массивов

Одномерный массив, основанный на примитивном типе *int*:

```
int a[];
int[] a;
```

Многомерный массив:

```
int a[][];
int[] a[];
int[][] a;

int[] a, b[]; == int a[], b[][];
```

Создание экземпляра массива

```
int[] array = new int[5];

int[] predefined = new int[] { 1, 2, 3 };

Object[] objects = { null, "Hello" };

int[][] matrix = {{1,2},{3,4}};

char[] string;

string = new char[5];

// Каждый элемент многомерного массива - массив

int[][] nonsquare = new int[3][];

nonsquare[0] = new int[3];

nonsquare[1] = new int[] { 1, 2 };

nonsquare[2] = new int[1];
```

Поле length

Использование поля *length* при работе с массивом:

```
Point p[] = new Point[5];
for (int i = 0; i < p.length; i++) {
        p[i] = new Point(i, i);
}
```

Значение индекса массива всегда имеет тип **int**.

Допустимые типы при обращении к элементу: **byte, short, char**.

Попытка задействовать **long** приведет к ошибке компиляции.

Максимальное количество элементов в массиве: *2,147,483,647*

Блоки и локальные переменные

Объявление нескольких локальных переменных с одинаковыми именами в пределах видимости блока недопустимо:

```
public class Test {
    public Test() {}
    public static void main(String[] args) {
        Test t = new Test();
        int x;
        {
            int x = 0;
            System.out.println("x = " + x);
        }
        for (int i = 0; i < 2; i++);
        for (int i = 0; i < 2; i++);
    }
}
```

Блоки и локальные переменные

Локальные переменные и параметры методов перекрывают видимость полей класса:

```
public class Test {
    static int x = 5;
    static int args = 0;
    public static void main(String[] args) {
        Test t = new Test();
        int x = 1;
        System.out.println("x = " + x + ", " + args[0]);
    }
}
```

Оператор if

Общая конструкция оператора:

```
if (логическое выражение)
    выражение или блок 1
else
    выражение или блок 2
```

Пример:

```
int x = 5;

if (x < 4)
    System.out.println("Меньше 4");
else if (x > 4) {
    System.out.println("Больше 4");
}
else if (x == 5)
    System.out.println("Равно 5");
else
    System.out.println("Другое значение");
```

Оператор switch

Общая конструкция оператора:

```
switch(int value) {
   case const1:
      выражение или блок
   case const2:
      выражение или блок
   case constn:
      выражение или блок
   default:
      выражение или блок
}
```

Работает с целыми типами меньше чем long, их объектными обертками и перечисляемыми типами.

Циклы

```
while (логическое условие продолжения)
    повторяющееся выражение или блок

do
   повторяющееся выражение или блок
While (логическое условие продолжения);

for (выполнить до цикла; условие продолжения;
        выполнить после каждого повторения)
    повторяющееся выражение или блок

for (Тип_элемента имя_переменной : массив или коллекция)
    повторяющееся выражение или блок
```

Операторы break, continue

```
public class Test2 {
  public static void main(String[] args) {
    int i = 0;
    outer: for (;;) { // бесконечный цикл
      for (; i < 10; i++) {
        System.out.println("i = " + i);
        if (i == 0) {
          System.out.println("continue");
          continue;
        }
        if (i == 1) {
          System.out.println("break");
          i++; // В противном случае i не будет увеличено.
          break;
        }
        if (i == 2) {
          System.out.println("continue outer");
          i++; // В противном случае i не будет увеличено.
          continue outer;
        }
        if (i == 3) {
          System.out.println("break outer");
          break outer;
```

Результат:

```
i = 0
continue
i = 1
break
i = 2
continue outer
i = 3
break outer
```

Литература

- http://www.intuit.ru/department/pl/javapl/
- Bruce Eckel, Thinking in Java, 3rd Edition

Thank you!