Типы данных и операторы

Лекция 3

УНЦ «Инфоком»
Самара
2013

NetCracker®

План лекции

- Типы данных в Java
- Операторы для работы с примитивными и ссылочными типами
- Работа со строками
- Массивы
- Инструкции, управляющие ходом выполнения программы

Типы данных в Java

- Java – строго типизированный язык
  - тип известен на момент компиляции
  - выявление многих ошибок до выполнения
- Две группы типов данных
  - Примитивные или простые (`primitive`)
  - Ссылочные или объектные (`reference`)

Характеристики типов данных

- **Множество значений**
  - для примитивных типов – значения из диапазона этого типа
  - для ссылочных типов – ссылки на объекты, контракт которых включает в себя контракт, определяемый типом ссылки
- **Возможные операции со значениями**
  - для примитивных типов – операторы
  - для ссылочных типов – действия, входящие в контракт типа (вызов методов и обращение к полям), и операторы
- **Форма хранения и представления**
  - форма хранения определяется реализацией JVM
  - JVM гарантирует одинаковое представление, не зависящее от реализации

Примитивные типы

- Булевый (логический) тип
  - `boolean` – допускает значения `true` или `false`
- Целочисленные типы
  - `char` – 16-битовый символ Unicode
  - `byte` – 8-битовое целое число со знаком
  - `short` – 16-битовое целое число со знаком
  - `int` – 32-битовое целое число со знаком
  - `long` – 64-битовое целое число со знаком
- Вещественные типы
  - `float` – 32-битовое число с плавающей точкой (IEEE 754-1985)
  - `double` – 64-битовое число с плавающей точкой (IEEE 754-1985)

Переменные

- Используются для хранения данных
- Имеет 3 базовые характеристики
  - имя
  - тип данных
  - значение
- Переменная объявляется
- Переменная инициализируется

Переменные

- Примеры объявления переменных примитивного типа

```
int a;
int b = 1, c = 2 + 3;
int d = b + c;
int e = a = 7;
final double pi = 3.1415;
```

- При объявлении нужно указать тип и имя
- Инициализация при объявлении или позже

Переменные

- Именованные участки памяти, способные содержать значения определенного типа
- Могут быть объявлены в различных частях кода
  - поля объектов и классов, параметры методов и др.
- Область видимости переменной определяется местом ее объявления
- Локальные переменные должны быть инициализированы перед их использованием

Примитивные и ссылочные типы данных

- Переменные простого типа хранят непосредственно свои значения
- При присваиваниях происходит копирование значений

```
int a = 100;
int b = a;

a = 101;
System.out.println(b); //чему равно b?
```

Целочисленные типы

Название типа	Длина, биты	Область значений
byte	8	-128 .. 127
short	16	-32.768 .. 32.767
int	32	-2.147.483.648 .. 2.147.483.647
long	64	-9.223.372.036.854.775.808 .. 9.223.372.036.854.775.807, (10^{19})
char	16	'\u0000' .. '\uffff', или 0 .. 65.535, беззнаковый, сравнения, операции

Арифметические операторы примитивных числовых типов

- Арифметические операции
  - `+` – сложение двух значений
  - `-` – вычитание второго значения из первого
  - `*` – умножение двух значений
  - `/` – деление первого значения на второе
  - `%` – остаток от деления первого значения на второе

- Результат имеет тип, совпадающий с «наиболее широким» типом из типов операндов, но не меньше, чем `int`

Арифметические операторы примитивных числовых типов

- Инкременты и декременты – соответственно, увеличивают и уменьшают значение на 1
  - Постфиксная форма: `i++`, `i--`
    результат – прежнее значение

```
idNum = nextID++;
```

  - Префиксная форма: `++i`, `--i`
    результат – новое значение

```
idNum = ++nextID;
```

- Унарные `+` и `-`
  - Аналогичны случаю, когда первый операнд равен 0
  - Если знак + или - находится перед литералом, он может трактоваться как часть литерала

Префиксная и постфиксная формы инкремента

Префиксная форма

```
int n = 1;
int i = 2;
System.out.println(n + ++i); // инкремент, сложение
System.out.println(i);
```

Постфиксная форма

```
int n = 1;
int i = 2;
System.out.println(n + i++); // сложение, инкремент
System.out.println(i);
```

Операторы примитивных целочисленных типов

- **операторы сравнения** (возврат булева значения)
  - `<`, `<=`, `>`, `>=`
  - `==`, `!=`
- **числовые операторы** (возвращают число)
  - унарные операции `+` и `-`
  - арифметические операции `+`, `-`, `*`, `/`, `%`
  - инкремента и декремента `++` и `--`
  - операции битового сдвига `<<`, `>>`, `>>>`
  - битовые операции `~`, `&`, `|`, `^`
- **оператор с условием** `? :`
- **оператор приведения типов**
- **оператор конкатенации со строкой** `+`

Операторы примитивных целочисленных типов

```
int a = -2147483648; // наименьшее значение типа int
int b = -a;          // -2147483648
```

```
double c = 3 / 10;
System.out.println(c);  // 0.0
```

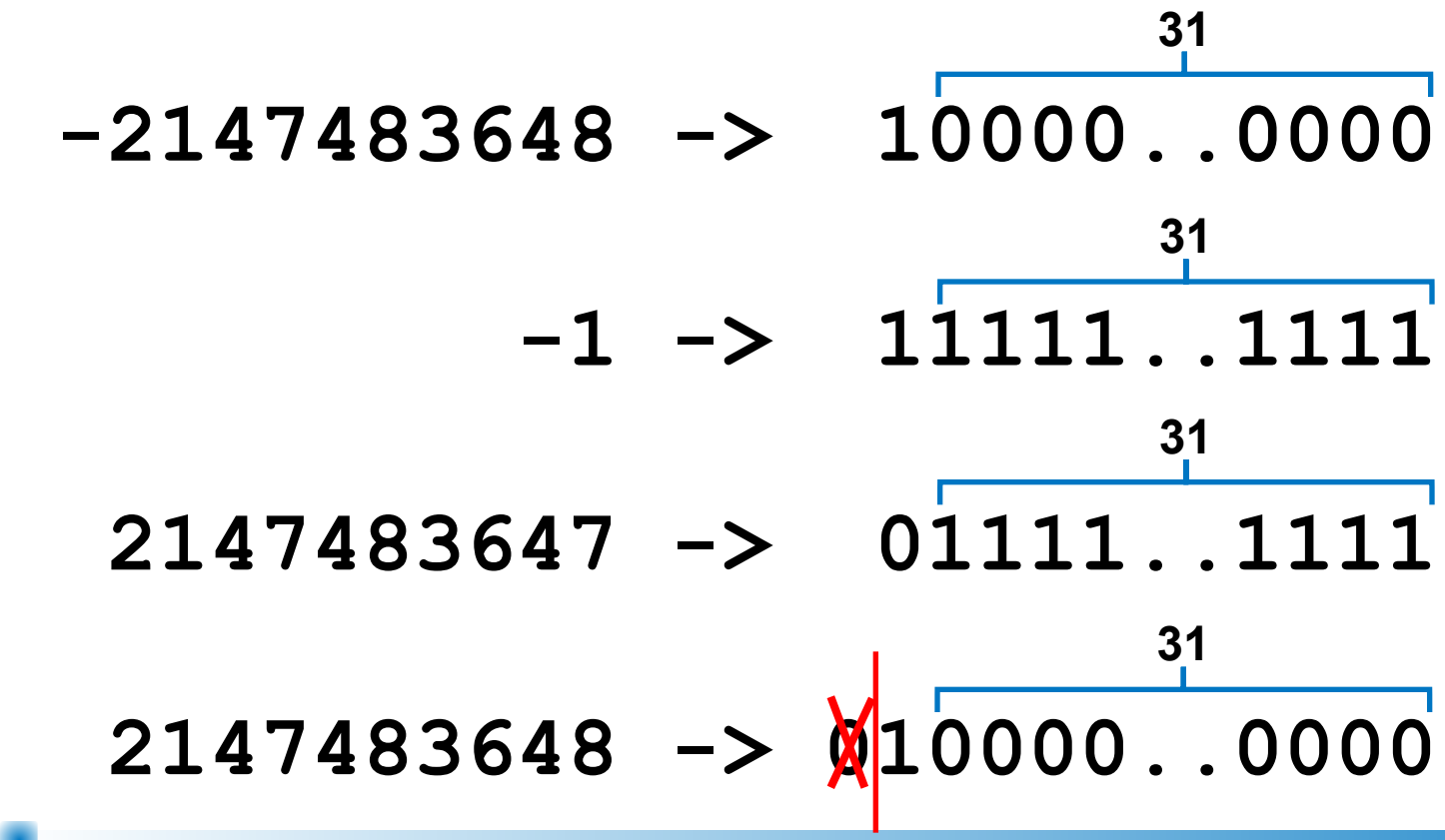
```
System.out.println(1000*60*60*24*7);   // для недели
System.out.println(1000*60*60*24*30L); // для месяца
```

```
int x = 100;
byte b = x; // ошибка компиляции!
```

```
byte b = 100;
byte c = b + 1; // ошибка компиляции!
```

Операторы примитивных целочисленных типов

```
int a = -2147483648; // наименьшее значение типа int
int b = -a;          // -2147483648
```



Операторы примитивных целочисленных типов

Десятичное целое в бинарную запись

```
int val = 2147483647;
String str = Integer.toBinaryString(val);
System.out.println(str); // 11111111111111111111111111111111
```

Бинарная запись в десятичное целое

```
String str = "11111111111111111111111111111111";
int val = Integer.parseInt(str, 2);
System.out.println(val); // 2147483647
```

Операторы примитивных целочисленных типов

```
int x = 100; long y = 200;
int z = x + y; // ошибка компиляции!
```

```
byte b = 127;
byte c = (byte)-b; // преобразование типа, иначе ошибка
```

```
byte x = 10; byte y = 20;
byte z = (x > y) ? x : y;  // верно, x и y одного типа
byte n = (x > 0) ? x : -x; // неверно, -x типа int
```

```
print(1 + 2 + "text"); // 3text
print("text" + 1 + 2); // text12
```

```
char c1 = 10; char c2 = 'A';  // для A код 65, \u0041
int i = c1 + c2 - 'B';        // результат 9
```

Побитовые операторы примитивных целых типов

- Логические операторы
  - `&` – побитовое «и» (and)

```
        1 &         3 ->         1
00000001 & 00000011 -> 00000001
```

  - `|` – побитовое «или» (or)

```
        1 |         3 ->         3
00000001 | 00000011 -> 00000011
```

  - `^` – побитовое «исключающее или» (xor)

```
        1 ^         3 ->         2
00000001 ^ 00000011 -> 00000010
```

  - `~` – побитовое отрицание

```
~       1 ->        -2
~00000001 -> 11111110
```

- Вычисления производятся в типе `int` либо `long`

Побитовые операторы примитивных целых типов

- Операторы сдвига
  - `<<` – сдвиг влево

```
        1 << 2 ->          4
00000001 << 2 -> 00000100
```

  - `>>` – арифметический сдвиг вправо

```
        4 >> 2 ->          1
00000100 >> 2 -> 00000001
       -1 >> 2 ->         -1
11111111 >> 2 -> 11111111
```

  - `>>>` – логический сдвиг вправо

```
        4 >>> 2 ->          1
00000100 >>> 2 -> 00000001
       -1 >>> 2 -> 1073741823
11111111 >>> 2 -> 00111111 11111111 11111111
11111111
```

- Вычисления производятся в типе `int` либо `long`

Дробные типы

Название типа	Длина биты	Область значений
float	4	-3.40282347E+38F..3.40282347e+38f 1.40239846e-45f
double	8	-1.79769313486231570E+308 .. 1.79769313486231570e+308; 4.94065645841246544e-324

- Знаковые типы
- Определены границы значений
- Определены границы точности

Операторы примитивных вещественных типов

- **операторы сравнения** (возврат булева значения)
  - `<`, `<=`, `>`, `>=`
  - `==`, `!=`
- **числовые операторы** (возвращают число)
  - унарные операции `+` и `-`
  - арифметические операции `+`, `-`, `*`, `/`, `%`
  - инкремента и декремента `++` и `--`
- **оператор с условием** `?` `:`
- **оператор приведения типов**
- **оператор конкатенации со строкой** `+`

Особенность примитивных вещественных типов

```
int a = 5, b = 0;
int c = a / b;
System.out.println(c);
```

```
Exception in thread "main"
java.lang.ArithmeticException
```

```
float a = 5, b = 0;
float c = a / b;
System.out.println(c);
```

```
Infinity
```

- Константы классов Float и Double
  - Positive Infinity (`Infinity`)
  - Negative Infinity (`-Infinity`)
  - Not a Number (`NaN`)
- Различаются значения `0`, `+0` и `-0`

Операторы примитивных вещественных типов

```
float f = 1e40f; // значение слишком велико, overflow
double d = 1e-350; // за границами точности, underflow
```

```
System.out.println(1e-40f/1e10f);  //  0.0
System.out.println(-1e-300/1e100); // -0.0
```

```
System.out.println(1f/0f);  // Infinity
System.out.println(-1d/0d); // -Infinity
```

```
System.out.println(0.0/0.0);      // NAN
System.out.println((1.0/0.0)*0.0);  // NAN
```

Операторы примитивных вещественных типов

```
System.out.println(0.0==-0.0);// true
System.out.println(0.0>-0.0); // false
```

```
System.out.println(1/2);// 0
System.out.println(1/2.);  //0.5
```

```
int x = 1;
int y = 2;
print (x / y);     // 0
print((double) x / y);  // 0.5
print(1.0 * x / y);  // 0.5
```

Операторы примитивных вещественных типов

Еще небольшой пример

```
double d = 1e-305 * Math.PI;
System.out.println(d);
for (int i = 0; i < 4; i++)
System.out.println(d /= 100000);
```

И его результат

```
3.141592653589793E-305
3.1415926535898E-310
3.141592653E-315
3.142E-320
0.0
```

Операторы

Постфиксные	`[] . (params) expr++ expr--`
Унарные	`++expr --expr +expr -expr ~ !`
Создание и приведение	`new (type)expr`
Арифметика	`* / %`
Арифметика	`+ -`
Побитовый сдвиг	`<< >> >>>`
Сравнение	`< > >= <= instanceof`
Равенство	`== !=`
И (and)	`&`
Исключающее ИЛИ (xor)	`^`
Включающее ИЛИ (or)	`|`
Условное И (and)	`&&`
Условное ИЛИ (or)	`||`
Условный оператор	`? :`
Операторы присваивания	`= += -= *= /= %= >>= <<= >>>= &= ^= |=`

Операторы сравнения примитивных числовых типов

- `>` и `<` – строгое сравнение
- `>=` и `<=` – нестрогое сравнение
- `==` – определение равенства
- `!=` – определение неравенства
- Результат – логическое значение: `true` или `false`
- Сравнение проводится в наиболее широком типе из типов операндов

Операторы присваивания примитивных типов

- `=` – простое присваивание
  - Тип выражения справа должен допускать присваивание в переменную слева

- `+=`, `-=`, `*=`, `/=`, `%=`, `>>=`, `<<=`, `>>>=`, `&=`, `^=`, `|=`
  - Присваивание с действием
  - Тернарный оператор, `x = x > 1 ? 2 : 1;`
  - Типы операндов должны позволять совершить операцию

Преобразование примитивных числовых типов

- **Неявное преобразование типов**
  Преобразование к более широкому типу
- **Явное преобразование типов**
  Преобразование к указанному типу с помощью оператора `(type)expr`

```
short s1 = 29;
int i1 = s1;
float f1 = i1;

int i2 = 14;
short s2 = (short) i2;

short s = -134;
byte b = (byte) s; // b = 122;
```

Особенности преобразования примитивных числовых типов

- Более широкий тип тот, переменные которого могут принимать большее количество значений
- Вещественные типы считаются шире целочисленных
- Это, естественно, не так

```
int big = 1234567890;
float approx = big;              // 1.23456794E9
System.out.println(big - (int)approx); // -46
```

Операторы примитивного логического типа

- `==` – определение равенства
- `!=` – определение неравенства
- `!` – отрицание
- `&` – логическое «и» (and)
- `|` – логическое «или» (or)
- `^` – логическое «исключающее или» (xor)
- `&&` – условное «и»
  (может не вычислять второй операнд)
- `||` – условное «или»
  (может не вычислять второй операнд)
- `? :` – оператор с условием
- `+` – конкатенация со строкой

Классы и объекты

- Класс – это описание объектов со схожей структурой и поведением (шаблон)
- Объект – экземпляр некоторого класса, или экземпляр массива
- Объектов имеющего класса может быть много, а может не быть вовсе
- Создание объектов происходит путем вызова конструктора класса (`new`)

Пример класса

```
public class Employee {
    private String name;
    private String position;
    private float salary;

    public Employee(String n) {
        name = n;
    }
    public Employee(String n, String p) {
        name = n;
        position = p;
    }
   ...
}
```

Примитивные и ссылочные типы данных

- Переменные ссылочного типа хранят ссылку на объект или `null`
- При присваиваниях происходит копирование значений ссылок

```
MyGadget m1 = new MyGadget("object
1");
MyGadget m2 = m1;

m1.name = "object 2";
System.out.println(m2.name); //а тут?
```

Примитивные и ссылочные типы данных

- Переменные ссылочного типа хранят ссылку на объект или `null`
- При присваиваниях происходит копирование значений ссылок

```
MyGadget m1 = new MyGadget("object
1");
MyGadget m2 = m1;

m1 = new MyGadget("object 2");
System.out.println(m2.name); //а тут?
```

Ссылки и объекты

- Доступ к объектам происходит по ссылкам-"безопасным указателям" на объект
- Допускается несколько ссылок на один и тот же объект
- Допускается отсутствие ссылок на объект
- Соответствие типа объекта по ссылке контролируется (на этапе компиляции)

Операторы ссылочных типов

- `new` – создание объекта класса (вызов конструктора)
- `=` – присвоение ссылки
  - Тип выражения справа должен допускать присвоение в тип переменной слева
- `==` и `!=` – сравнение ссылок
  - Сравниваются только ссылки, а не состояние объектов!

Операторы ссылочных типов

- `.` – разыменование ссылки
  - `reference.method()`
  - `reference.field`
- `()` – вызов метода
- У любого объекта можно вызвать методы, объявленные в классе `Object`

Сравнение объектов по ссылке

- Операторы == и != для ссылочных типов сравнивают не состояния объектов, а значения ссылок

```
Place p1 = new Place("Good Cafe");
Place p2 = p1;
Place p3 = new Place("Good Cafe");
System.out.println(p1 == p2); //true
System.out.println(p1 == p3); //false
```

Сравнение объектов по ссылке

- Если при сравнении один из аргументов равен **`null`**, то результат равен **`false`**
- Если оба аргумента сравнения равны **`null`**, то результат сравнения равен **`true`**

```
Place p1 = new Place("Good Cafe");
Place p2 = null;
Place p3 = null;
System.out.println(p1 == p2); // false
System.out.println(p2 == p3); // true
```

Сравнение объектов по значению

- Метод `equals(Object o)`, доступный для любого объекта
- Ссылочные величины можно складывать со строкой (вызов `toString()` объекта)
- Если ссылка равна `null`, то к строке добавляется текст `"null"`

```
String s = "a string";
s = s + " line"
System.out.println(s.equals("a string line"));
```

Преобразование ссылочных типов

- Преобразование типа возможно, только если контракт целевого типа является частью контракта приводимого типа
- Более широким считается тип, переменные которого могут принимать большее количество значений. Родительский тип считается более общим, чем дочерний.
- Неявное преобразование типов – преобразование от более узкого к более широкому
- Явное преобразование типов – преобразование от более широкого к более узкому с помощью оператора явного преобразования `(type)expr`

Преобразование и проверка ссылочных типов

```
Integer i = new Integer(5);
Object o = i;
i = (Integer) o;
```

- Если явное преобразование типов невозможно, возникает ошибка `java.lang.ClassCastException`

- Соответствие типа можно проверить с помощью оператора `instanceof`, возвращающего `true`, если тип применим к объекту и `false`, если нет

Преобразование и проверка ссылочных типов

```
Integer i = new Integer(5);
Object o = i;
if (o instanceof Integer) {
   i = (Integer) o;
   ...
}
else { ... }
```

- Оператор **`instanceof`** не позволяет определить реальный тип объекта, а лишь проверяет объект на совместимость с указанным типом

Оператор ветвления

- Формат:
  `<логическое выражение> ? <значение 1> : <значение 2>`

```
double factor = (a > b) ? 1 : 0.7;
```

- Если логическое выражение истинно, возвращается значение второго операнда, а если ложно – третьего операнда
- Типы второго и третьего операндов должны быть «совместимы»
- Оператор можно применять в выражениях присваивания вместо инструкции ветвления

Оператор ветвления

```
boolean flag = ...;
...
factor = flag ? 1 : 0.7;
/*
if (flag)
   factor = 1;
else
   factor = 0.7;
*/
```

Работа со строками

- Для работы со строками существуют специальные классы `String` и `StringBuffer` (`StringBuilder` с Java5)
- Каждый строковый литерал порождает экземпляр класса `String`
- Значение любого типа может быть приведено к строке
- Если хотя бы один из операндов оператора `+` является ссылкой на строку, то остальные операнды также приводятся к строке, а оператор трактуется как конкатенация строк

Массивы

- Массив – упорядоченный набор элементов одного типа
- Элементами могут быть значения простых и ссылочных типов
- Массивы сами по себе являются объектами и наследуют от класса `Object`
- Доступ к элементам по целочисленному индексу с помощью оператора `[]`

Объявление одномерных массивов

- Объявление, инициализация, заполнение

```
int array1[], justIntVariable = 0;
int[] array2;
array2 = new int[20];
for (int i = 0; i < array2.length; i++)
  array2[i] = 1000;
```

- Способ «3 в 1»

```
byte[] someBytes = {0, 2, 4, 8, 16, 32};

someMethod(new long[] {1, 2, 3, 4, 5});
```

Работа с одномерными массивами

- Форма объявления ссылки на массив с квадратными скобками после типа элемента является более предпочтительной
- Объект массива создается с помощью оператора `new`
- Массив при этом заполняется значениями по умолчанию для типа его элементов (`0`, `false` или `null`)
- Нумерация в массивах начинается с 0
- Длина массива хранится в публичном неизменяемом поле `length`
- Изменить длину массива после создания его объекта нельзя

Многомерные массивы

- Состоят из одномерных массивов, элементами которых являются ссылки на массивы меньшей размерности
- При создании объекта необязательно указывать все размерности
- Массив необязательно должен быть «прямоугольным»

Многомерные массивы

```
// Автоматическая
int[][] twoDimArr = new int[10][5];

// Вручную
int[][] twoDimArr = new int[10][];
for (int i = 0; i < 10; i++)
  twoDimArr[i] = new int[i];

// Явно
int[][] arr3 = { {0}, {0, 1}, {0, 2, 4} };
```

Виды инструкций

- Выражения присваивания
- Префиксные и постфиксные формы выражений с операторами инкремента и декремента
- Конструкции вызова методов
- Выражения создания объектов
- Составные инструкции
- Управляющие порядком вычислений

Блок

- Составная инструкция
- Может использоваться в любом месте, где допускается инструкция
- Определяет область видимости локальных переменных: объявленная внутри блока переменная не видна за его пределами

```
int a = 5;
int b = 10;
{
  int c = a + b;
  int d = a - b;
}
```

Ветвление

- Полная форма

```
if (ЛогическоеВыражение)
  Инструкция1
else
  Инструкция2
```

- Неполная форма

```
if (ЛогическоеВыражение)
  Инструкция1
```

- `else` относится к ближайшему выражению `if`, поэтому настоятельно рекомендуется использование блоков инструкций

Блок переключателей

```
switch (ЦелочисленноеВыражение) {
   case n: Инструкции
   case m: Инструкции
   ...
   default: Инструкции
}
```

- Для типов `char`, `byte`, `short`, `int`
- Выполняются инструкции, расположенные за меткой `case`, предложение которой совпало со значением параметра блока переключателей

Блок переключателей

- Если ни одно из предложений не подошло, выполняются инструкции, расположенные за меткой `default`
- Метка `default` является необязательной
- Метка `case` или `default` не служит признаком завершения блока переключателей
- Команда `break` передает управление первой инструкции, следующей за блоком переключателей

Условные циклы while

- Форма с предусловием
  - Выполняется пока условие истинно
  - Если при входе в цикл условие ложно, цикл не выполняется

```
while (ЛогическоеВыражение)
  Инструкция
```

- Форма с постусловием
  - Выполняется пока условие истинно
  - При первом входе в цикл проверка условия не производится

```
do
  Инструкция
while (ЛогическоеВыражение);
```

Цикл с предусловием for

- Формально цикл for в Java не является циклом со счетчиком
- Общий синтаксис

```
for(СекцияИнициализации; ЛогическоеВыражение;
СекцияИзменения)
  Инструкция
```

- Все секции заголовка являются необязательными
- Тело также может быть пустым

```
for( ; ; );
```

Секции цикла for

- Секции инициализации и изменения могут быть представлены списком выражений, разделенных запятой

```
for (i = 0, j = 50; j >= 0; i++, j--) {
  //...
}
```

- Допустимо объявление переменных в секции инициализации

```
for (int i = 0, j = 50; j >= 0; i++, j--) {
  //...
}
```

Объявление переменных в цикле for

```
for (int i = 0, Cell node = head;
     i < MAX && node != null;
     i++, node = node.next) {
  //...
}
```

- При инициализации переменных различных типов они не должны объявляться внутри заголовка

```
int i; Cell node;
for (i = 0, node = head;
     i < MAX && node != null;
     i++, node = node.next) {
  //...
}
```

Работа с метками

- Метка
  ```
  метка: Инструкция
  ```
- Оператора `goto` в Java нет!!!
- Метками можно помечать блоки инструкций и циклы
- Обращаться к меткам разрешено только с помощью команд `break` и `continue`

break

- Применяется для завершения выполнения кода блока инструкций
- Завершение текущего блока (безымянная форма)
  ```
  break;
  ```
- Завершение указанного блока (именованная форма)
  ```
  break метка;
  ```
- Завершить блок, который сейчас не выполняется, нельзя!

break

```
private float[][] matrix;

public boolean workOnFlag(float flag) {
  int y, x;
  boolean found = false;
  search:
    for (y = 0; y < matrix.length; y++) {
      for (x = 0; x < matrix[y].length; x++) {
        if (matrix[y][x] == flag) {
          found = true;
          break search;
        }
      }
    }
  //...
}
```

continue

- Применяется только в контексте циклических конструкций
- Производит передачу управления в конец тела цикла
- Завершение витка текущего цикла (безымянная форма)
  ```
  continue;
  ```
- Завершение витка указанного цикла (именованная форма)
  ```
  continue метка;
  ```
- Завершить виток цикла, который сейчас не выполняется, нельзя!

continue

```
static void doubleUp(int[][] matrix) {
  int order = matrix.length;
  column:
    for (int i = 0; i < order; i++) {
      for (int j = 0; j < order; j++) {
        matrix[i][j] = matrix[j][i] =
          matrix[i][j] * 2;
        if (i == j)
          continue column;
      }
    }
}
```

Возврат из метода

- Инструкция `return` прекращает выполнение метода и возвращает его результат
- С возвращаемым значением
  `return value;`
  - Значение должно быть приводимо к типу, возвращаемому методом
- Без возвращаемого значения
  `return;`
  - методы void
  - конструкторы

Спасибо за внимание!

NetCracker®

Дополнительные источники

- Арнолд, К. Язык программирования Java [Текст] / Кен Арнолд, Джеймс Гослинг, Дэвид Холмс. – М. : Издательский дом «Вильямс», 2001. – 624 с.
- Вязовик, Н.А. Программирование на Java. Курс лекций [Текст] / Н.А. Вязовик. – М. : Интернет-университет информационных технологий, 2003. – 592 с.
- Хорстманн, К. Java 2. Библиотека профессионала. Том 1. Основы [Текст] / Кей Хорстманн, Гари Корнелл. – М. : Издательский дом «Вильямс», 2010 г. – 816 с.
- Эккель, Б. Философия Java [Текст] / Брюс Эккель. – СПб. : Питер, 2011. – 640 с.
- JavaSE at a Glance [Электронный ресурс]. – Режим доступа: http://www.oracle.com/technetwork/java/javase/overview/index.html, дата доступа: 21.10.2011.
- JavaSE APIs & Documentation [Электронный ресурс]. – Режим доступа: http://www.oracle.com/technetwork/java/javase/documentation/api-jsp-136079.html, дата доступа: 21.10.2011.