Самарский государственный аэрокосмический университет
имени академика С.П. Королёва

Объектно-ориентированное программирование

Лексика языка Java

Занятие 2

Самара
2015

План лекции

- Основы лексики Java
- Типы данных и литералы
- Операторы
- Работа со строками и массивами
- Инструкции

Кодировка

- Java ориентирован на Unicode

- Первые 128 символов почти идентичны набору ASCII
- Символы Unicode задаются с помощью escape-последовательностей
  `\u262f`, `\uu2042`, `\uuu203d`
- **Java чувствителен к регистру!**

Исходный код

Исходный код разделяется на:

- Пробелы
  - ASCII-символ SP, \u0020, дес. код 32
  - ASCII-символ HT, \u0009, дес. код 9
  - ASCII-символ FF, \u000c, дес. код 12
  - ASCII-символ LF, символ новой строки
  - ASCII-символ CR, возврат каретки
  - символ CR, за которым сразу следует символ LF
- Комментарии
- Лексемы

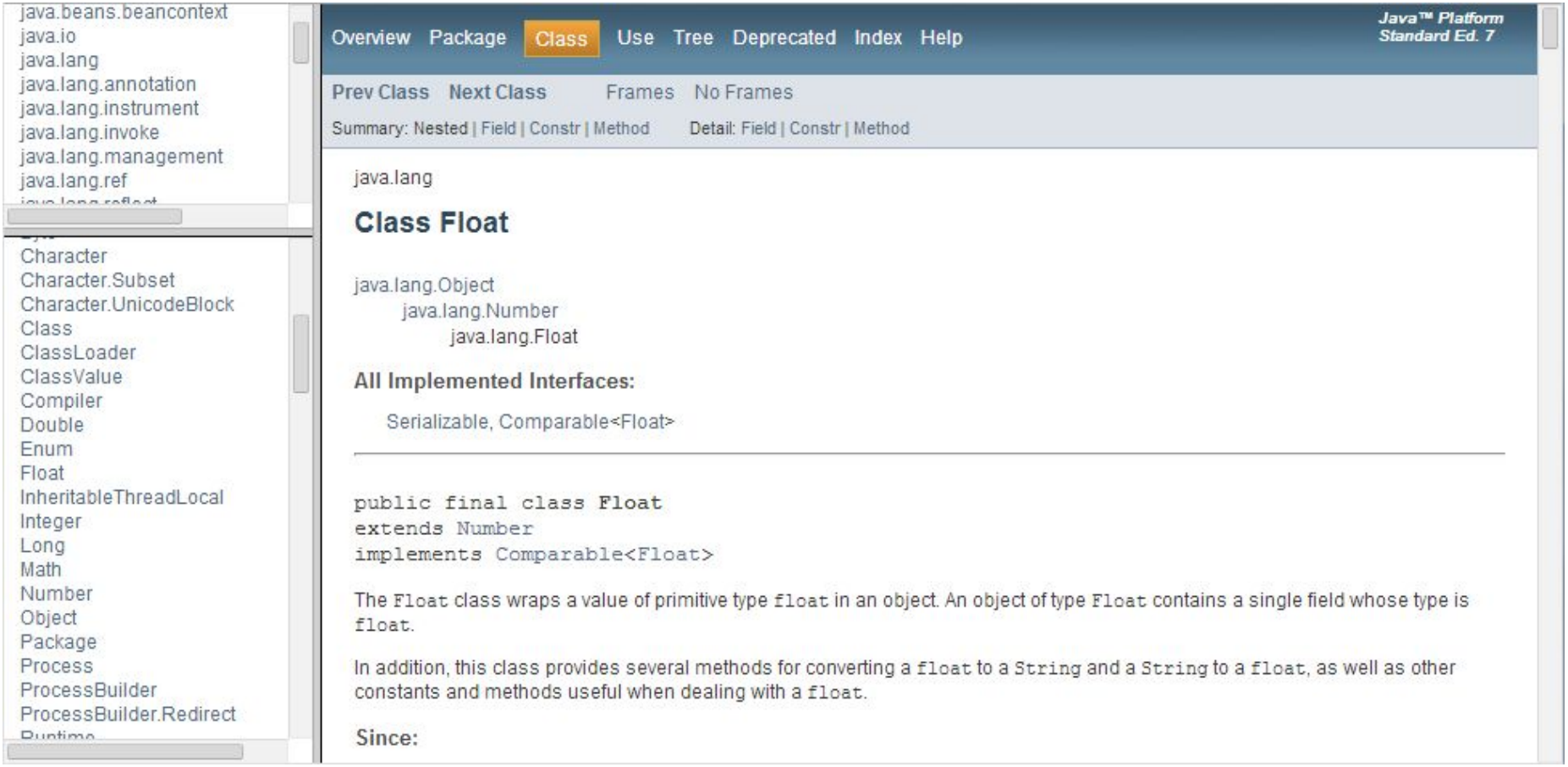
Комментарии

- `// Комментарий`
  Символы после `//` и до конца текущей строки игнорируются
- `/* Комментарий */`
  Все символы, заключенные между `/*` и `*/`, игнорируются
- `/** Комментарий */`
  Комментарии документирования

Комментарии документирования (javadoc)

- Начинаются с `/**`, заканчиваются `*/`
- В строках начальные символы `*` и пробелы перед ними игнорируются
- Допускают использование HTML-тэгов, кроме заголовков
- Специальные тэги `@see`, `@param`, `@deprecated`

Комментарии документирования (javadoc)



Лексемы

- Идентификаторы
- Служебные слова
  `class`, `public`, `const`, `goto`, и т.д.
- Литералы
- Разделители
  `{ } [ ] ( ) ; . ,`
- Операторы
  `= > < ! ? : == && ||`

Типы данных

- **Ссылочные**
  - Предназначены для работы с объектами
  - Переменные содержат ссылки на объекты
  - Ссылка – это не указатель!
  - Тип переменной определяет контракт доступа к объекту
- **Примитивные (простые)**
  - Предназначены для работы со значениями естественных, простых типов
  - Переменные содержат непосредственно значения

Типы данных

Тип данных определяется следующими характеристиками:

- Множество значений
  - для примитивных типов – числа, не выходящие за диапазон 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

Переменные

- Именованные участки памяти, способные содержать значения определенного типа
- Могут быть объявлены в различных частях кода
  - поля объектов
  - поля классов (статические поля)
  - параметры методов
  - локальные переменные методов и блоков инициализации
- Объявление переменной состоит из наименования типа, идентификатора и инициализации
- Область видимости переменной определяется местом ее объявления
- **Локальные переменные должны быть инициализированы перед их использованием**

Ссылочные типы

- К ссылочным типам относятся типы классов (в т.ч. массивов) и интерфейсов
- Переменная ссылочного типа способна содержать ссылку на объект, относящийся к этому типу
- Ссылочный литерал `null`

Примитивные типы

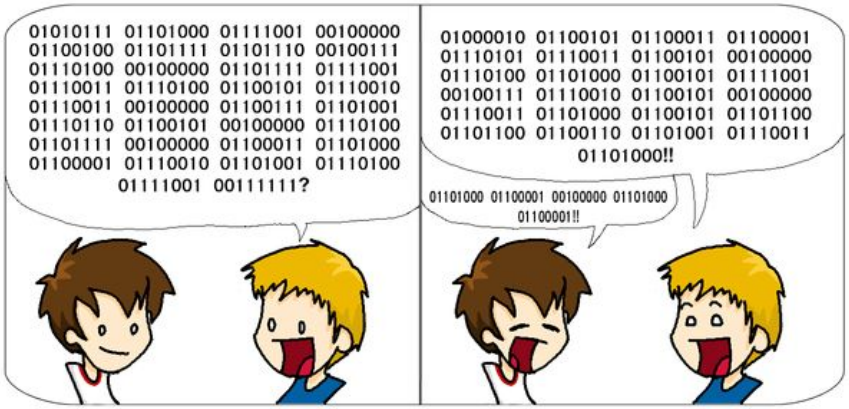
- Булевский (логический) тип
  - `boolean` – допускает хранение значений true или false
- Целочисленные типы
  - `char` – 16-битовый символ Unicode
  - `byte` – 8-битовое целое число со знаком
  - `short` – 16-битовое целое число со знаком
  - `int` – 32-битовое целое число со знаком
  - `long` – 64-битовое 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)

Литералы

- Булевы
  `true false`
- Символьные
  `'a' '\n' '\\' '\377' '\u0064'`
- Целочисленные
  `29  035  0x1D  0X1d 0xffffL`
  - По умолчанию имеют тип `int`
- Числовые с плавающей запятой
  `1.  .1  1e1 1e-4D 1e+5f`
  - По умолчанию имеют тип `double`
- Строковые
  `"Это строковый литерал"  ""`

Бинарные литералы (Java 7)

- Префиксы
  - `0b`
  - `0B`



- Полезны при работе с битовыми представлениями чисел, поскольку позволяют лучше видеть:
  - саму структуру числа
    `0b0000111111110000 // 2040, 0x07f8`
  - взаимосвязь чисел
    `0b0000111111111111 // 4095, 0x0fff`
    `0b1111000000000000 // 61440, 0xf000`

Подчеркивание в числовых литералах (Java7)

- Можно использовать
  - В литералах любых числовых типов
    `765_324_213_434L`
  - В литералах в любых системах счисления
    `0xFF_00_FF_00`
  - В нужных местах числа
    `1_23_456_7890`
  - В нужном количестве
    `6_____6`

Подчеркивание в числовых литералах (Java7)

- Нельзя использовать
  - В начале и в конце числа
    `_123  123_`
  - Рядом с разделителем целой и дробной части
    `10_.01  10._01`
  - Перед суффиксами `L`, `F` и `D`
    `1_L  1.1_F  1.1_D`
  - В строковых литералах с числами
    `"6_____6"`

Константы

- Констант как особого вида переменных и полей в Java нет
- Если необходима константа в методе, то переменная при объявлении снабжается модификатором `final`
  - Так можно сделать даже с параметром метода
  - Это не имеет особого смысла
  - Но иногда это явно требуется в многопоточных программах
- Если необходима общедоступная константа, то создаётся поле с модификаторами `public static final`
  - Такие поля и называют константами (условно)
  - Их имена записываются заглавными буквами
  - Обращение чаще всего происходит через имя класса, например `BigInteger.ZERO`

Операторы

- Постфиксные
- Унарные

- Создание и приведение
- Арифметика
- Арифметика
- Побитовый сдвиг
- Сравнение
- Равенство
- И (and)
- Исключающее ИЛИ (xor)
- Включающее ИЛИ (or)
- Условное И (and)
- Условное ИЛИ (or)
- Условный оператор
- Операторы присваивания

```
[] . (params) expr++ expr--
++expr --expr +expr -expr
   ~ !
new (type)expr
* / %
+ -
<<  >>  >>>
< > >= <= instanceof
== !=
&
^
|
&&
||
? :
= += -= *= /= %=
   >>= <<= >>>= &= ^= |=
```

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`

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
```

- Легальные значения
  - Positive Infinity (Infinity)
  - Negative Infinity (-Infinity)
  - Not a Number (NaN)
- Различаются значения 0, +0 и −0

$$\lim_{x \to \infty} x = \infty$$

$$\lim_{x \to \infty} x^2 = \infty\infty$$

$$\lim_{x \to \infty} x^3 = \infty\infty\infty$$

$$\lim_{x \to \infty} x^4 = \infty\infty\infty\infty$$

Арифметические операторы примитивных числовых типов

- Инкременты и декременты – соответственно, увеличивают и уменьшают значение на 1
  - Постфиксная форма: `i++`, `i--`
    результатом оператора является прежнее (неизмененное) значение
  - Префиксная форма: `++i`, `--i`
    результатом оператора является новое значение

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

Побитовые операторы примитивных целых типов

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

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

  - `|` – «или» (or)

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

  - `^` – «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`

Побитовые операторы примитивных целых типов

```
float Q_rsqrt( float number )
{
        long i;
        float x2, y;
        const float threehalfs = 1.5F;

        x2 = number * 0.5F;
        y  = number;
        i  = * ( long * ) &y;                       // evil floating point bit level hacking
        i  = 0x5f3759df - ( i >> 1 );               // what the fuck?
        y  = * ( float * ) &i;
        y  = y * ( threehalfs - ( x2 * y * y ) );   // 1st iteration
//      y  = y * ( threehalfs - ( x2 * y * y ) );   // 2nd iteration, this can be removed

        return y;
}
```

Это - оригинальный кусок кода игры Quake 3 с комментариями Джона Кармака без каких либо изменений.

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 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)

Операторы присваивания примитивных типов

- `=` – простое присваивание
  - Тип выражения справа должен допускать присваивание в переменную слева
- `+=`, `-=`, `*=`, `/=`, `%=`, `>>=`, `<<=`, `>>>=`, `&=`, `^=`, `|=`
  - Присваивание с действием
  - Выражение `a ?= b` эквивалентно `a = a ? b`, но выполняется быстрее
  - 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 широким считается тип, переменные которого могут принимать большее количество значений
- Вещественные типы считаются шире целочисленных
- Это, естественно, не так

```
long orig = 0x7effffff00000000L;
float fval = orig;
long lose = (long)fval;
```

```
orig = 9151314438521880576
fval = 9.1513144e18
lose = 9151314442816847872
```

$500 млн за строчку кода или стоимость ошибок ПО в космосе
http://geektimes.ru/post/252690/

Операторы ссылочных типов

- `new` – создание объекта класса
- `=` – присвоение ссылки
  - Тип выражения справа должен допускать присвоение в тип переменной слева
- `==` и `!=` – сравнение ссылок
  - Сравниваются только ссылки, а не состояние объектов!
- `.` – разыменование ссылки
  - `reference.method()`
  - `reference.field`
- `()` – вызов метода
- У любого объекта можно вызвать методы, объявленные в классе `Object`

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 (широким), 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
- Оператор `instanceof` не позволяет определить реальный тип объекта, а лишь проверяет его соответствие указанному типу

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

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

```
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

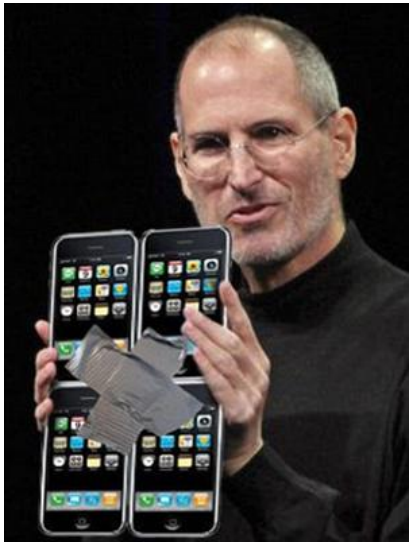
Многомерные массивы

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

```
// 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} };
```



Инструкции

- Инструкция
  - Описание одного действия
  - «Заканчивается» знаком `;`
  - Тела методов, конструкторов и блоков инициализации состоят из набора инструкций
- 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 или String) {
  case n: Инструкции
  case m: Инструкции
  ...
  default: Инструкции
}
```

- Для типов `char`, `byte`, `short`, `int`, `String (Java 7)`
- Выполняются инструкции, расположенные за меткой `case`, предложение которой совпало со значением параметра блока 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

Строки в предложениях switch

- В качестве проверяемого значения можно указывать ссылку на объект строки
- В качестве значений для сравнения можно указывать
  - строковые литералы
  - ссылки на строки, объявление которых снабжено модификатором `final`
- Сравнение значений производится также, как если бы использовался метод `String.equals()`
  - регистр имеет значение
  - начальные и концевые пробелы имеют значение

Строки в предложениях switch

```
public static Gender convert(String s) {
    Gender g;
    switch (s) {
        case "м": case "муж": case "муж.": case "мужской":
            g = Gender.Male; break;
        case "ж": case "жен": case "жен.": case "женский":
            g = Gender.Female; break;
        default:
            g = Gender.Unknown; break;
    }
    return g;
}
```

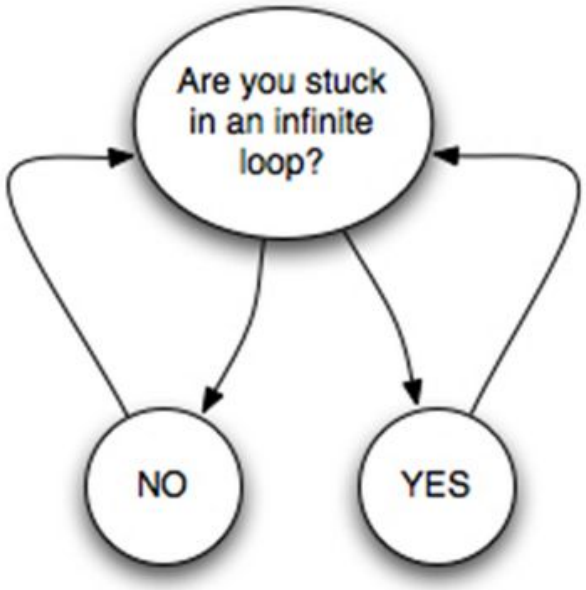
Условные циклы while

- 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!

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.