Язык программирования JAVA

Введение

Базовый синтаксис и типы данных

Управление выполнением программы

Что такое JAVA?

- Объектно-ориентированный язык программирования
- Изначально был разработан для управления бытовой электроникой
- Поставляется с большой библиотекой классов
- Использует виртуальную машину (JVM) для выполнения программ

Ключевые особенности JAVA

- Объектно-ориентированный
- Интерпретируемый и платформонезависимый
- Динамическая загрузка библиотек
- Мультипоточность
- Надежность и безопасность

Объектно-ориентированный подход

- Объекты и классы
  - Объект-представление «вещи» в реальном мире
  - Класс – «шаблон», определяющий «вещи»
- Модель классов объединяет
  - Существующие классы и объекты
  - Поведение, цели и структуру
  - Отношения между классами
  - Отношения между объектами
- Модель используется во всем проекте

Независимость от платформы

- Исходные тексты хранятся в текстовом виде в файле .java
- Файл .java компилируется в файл .class
- Этот файл содержит байт-код (инструкции для выполнения интерпретатором)
- Байт-код интерпретируется во время выполнения

Just-In-Time (JIT) компилятор

- Компилирует байт-код в исполняемый код для конкретной платформы
- Увеличивает производительность
- Оптимизирует повторяющийся код, например, циклы

Java - приложения

- Клиентские
  - JVM выполняет отдельное приложение из командной строки
  - Классы загружаются с локального диска
- Серверные
  - Обслуживают несколько клиентов
  - Применяются для многозвенных приложений

Java SDK (JDK)

- Sun Java SDK включает в себя
  - Компилятор (javac)
  - Библиотеку классов
  - Отладчик (jdb)
  - Интерпретатор (java)
  - Генератор документации (javadoc)
  - Архиватор (jar)
  - Другое…

Варианты поставки

- J2SE (Standard Edition) – разработка обычных приложений
- J2EE (Enterprise Edition) – разработка приложений многозвенной архитектуры

Инструменты, используемые в данном курсе

- J2SE версии 1.8 (Java8)
- В качестве IDE будет использоваться IntelliJ IDEA 2018

Ключевые компоненты SDK

- Компилятор (javac) – создает из исходного кода байт-код
- Интерпретатор (java) – выполняет байт-код

Пакеты

- Классы объединяются в специальные структуры, называемые пакетами
- Стандартные пакеты для
  - Поддержки базовых конструкций языка (java.lang)
  - Создания оконного интерфейса (javax.swing)
  - Управления вводом/выводом (java.io)
  - Коллекции (java.collections)
  - Время-дата (java.time)

Структура исходного файла класса Java

- Исходный файл состоит из следующих частей
  - Необязательное слово **`package`**, за которым следует наименование пакета, в котором содержится класс
  - Необязательный оператор **`import`** (может быть несколько), который указывает, какие классы из сторонних пакетов используются создаваемым классом
  - Одно или более определение **`class`** или **`interface`**, за которым следует программный блок
  - Файл должен иметь ТО ЖЕ имя, что и создаваемый класс
- Ключевые слова языка Java ЧУВСТВИТЕЛЬНЫ К РЕГИСТРУ
- В файле может быть ТОЛЬКО ОДИН public класс

Пример класса Java

```
package ru.vsu.test;
import java.util.Date;
public class FirstProgram {
    private Date today;
    public Date getToday(){ return today; }
    public void setToday(Date aToday){
        today = aToday;
    }
    public static void main (String[] args){
        FirstProgram fp = new FirstProgram();
        fp.setToday(new Date());
        System.out.println (fp.getToday());
    }
}
```

Соглашения об именовании

- **Имена файлов**
  - **Customer.java**
  - **Person.class**
- **Имена пакетов**
  - **java.util**
  - **javax.swing**
- **Имена классов**
  - **Customer**
  - **Person**
- **Имена свойств класса**
  - **firstName**
  - **id**
- **Имена методов**
  - **getName**
  - **isAlive**
- **Имена констант**
  - **SQUARE_SIZE**

Также могут использоваться цифры 1..9, _, $

Определение класса

- Определение класса включает:
  - Модификатор доступа
  - Ключевое слово class
  - Статические поля
  - Экземплярные поля
  - Конструкторы
  - Статические методы
  - Экземплярные методы

Пример

```
public class FirstProgram {
    private Date today;
    public Date getToday(){
      return today;
    }
    public static final PROGRAM_SIZE=560;
    public static void main (String[] args){
    ...
    }
}
```

Блоки кода

- Блоки кода обрамляются в фигурные скобки “{“ “}”
- Охватывают определение класса
- Определения методов
- Логически связанные разделы кода

```
import java.util.Date;
public class FirstProgram {
    public static void main (String[] args){
        System.out.println (new Date());
    }
}
```

Пример

```
import java.util.Date;
public class FirstProgram {
    private Date today;
    public Date getToday(){
        return today;
    }
    public void setToday(Date aToday){
        int i = 0;
        i++
        today = aToday;
    }
}
```

Переменные

- Основное место для хранения данных
- Должны быть явно объявлены
- Каждая переменная имеет тип, идентификатор и область видимости
- Определяются для класса, для экземпляра класса и внутри метода

Объявление переменных

- Может быть объявлена в любом месте блока кода
- Должна быть объявлена перед использованием
- Область видимости определяется блоком
- Иногда необходимо инициализировать переменные перед использованием

Объявление переменных

- Основная форма объявления
  - тип идентификатор [= значение];

```
public class FirstProgram {
    public static void main (String[] args){
        int itemsSold = 10;
        float itemCost = 11.0f;
        int i, j, k;
        double interestRate;
    }
}
```

- При объявлении переменные могут быть проинициализированы

Именование переменных

- Имя переменной должно начинаться с буквы, знака подчеркивания или со знака “$”
- Имя переменной может включать цифры
- Давайте переменным осмысленные имена

Типы данных

- Восемь простых типов данных
  - Шесть числовых
  - Символьный
  - Логический
- Определяемые пользователем типы
  - Классы
  - Интерфейсы
  - Массивы
  - Перечисления
  - Исключения

Простые типы данных

Целые	С плавающей точкой	Символьный	Логический
byte short int long	float double	char	boolean
1, 2, 3, -2 012 0x23f 2553L	3.0F .9937F 3.455E8 1.0D	‘s’ ‘\141’ ‘\u0061’ ‘\n’	true false

Оператор присваивания

- Оператор присваивания – выражение и может использоваться там, где допустимы выражения
- Сначала вычисляется правая часть, а затем полученное значение присваивается левой части

```
int itemsSold = 10;
double itemCost = 11.0F+12.0D;
int i = i+7;
i = j = k = 100;
```

Арифметические операторы

- Сложение (+)
- Умножение (*)
- Вычитание (-)
- Деление (/)
- Остаток от деления (%)

ВНИМАНИЕ ПЕРЕПОЛНЕНИЕ:

```
byte a = 100;

byte b = 100;

byte c = a+b;
```

c = -56

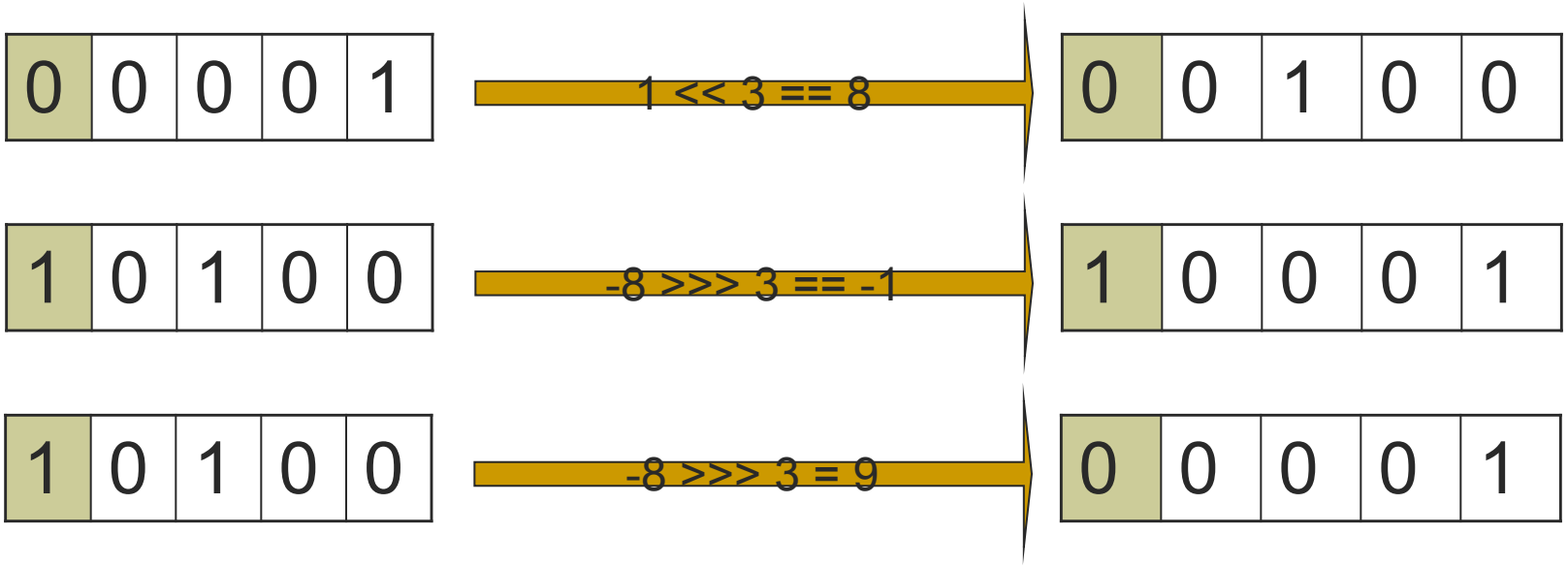
Операции инкремента и декремента

- Увеличение на 1 (++)
- Уменьшение на 1 (--)

```
int var1 = 3;
int var2 = 0;
var2 = ++var1; //сначала увеличивается var1, а затем
  //присваивается var2
var2 = var1++; //сначала присваивается var2, а затем
  // увеличивается var1
```

Побитовый сдвиг

- << - сдвиг влево
- >> - сдвиг вправо
- >>> - сдвиг вправо с заполнением нулями
- Правая часть сокращается до остатка от деления на длину числа, т.е. 1 << 35 == 1 << 3



Операторы сравнения

- < - меньше
- > - больше
- >= - больше или равно
- <= - меньше или равно
- == - равно
- != - не равно

Логические операторы

- && - and
- || - or
- ^ - xor
- ! – not

Управление выполнением программы

- Типы выполнения
  - Последовательность
  - Выбор
  - Итерация/цикл
  - Переход

Последовательность

- Каждый оператор завершается точкой с запятой
- Группы операторов обрамляются фигурными скобками
- Каждая группа выполняется как единый оператор внутри последовательности операторов

Оператор `if`

```
if (логическое выражение)
    оператор1;
[else
    оператор2];
```

```
if (i % 2 == 0)
    System.out.println("Even");
else
    System.out.println("Odd");
```

```
if (i % 2 == 0){
    System.out.print(i);
    System.out.println(" is even");
}
```

Оператор **switch**

```
switch (выражение целого типа){
  case const1:
    statement1;
    break;
  case const2:
    statement2;
    break;
  default:
    statement3;
}
```

- Используется для выбора из счетного количества вариантов
- Выражения const должны быть типа byte, int, char, short или enum

Циклы

- Типы
  - while
  - do..while
  - for
  - “foreach” aka for
- Все циклы имеют две части
  - Условие выполнения
  - Тело

Цикл **while**

```
while (логическое выражение)
    оператор;
```

```
int i = 0;
while (i < 100){
    System.out.println("i = "+i);
    i++;
}
```

Цикл do..while

```
do
    оператор;
while (условие выхода);
```

```
int i = 0;
do{
    System.out.println("i = "+i);
    i++;
}
while (i < 10);
```

Цикл for

```
for (инициализация; условие выхода; условие итерации)
    оператор;
```

```
for (int i = 0; i < 10; i++) {
    System.out.println("i = "+i);
}
```

```
for (int i = 0, j = 10; i < j; i++, j--) {
    System.out.println("i = "+i);
    System.out.println("j = "+j);
}
```

Цикл “foreach”

```
for (Iterator<String> i = someIterable.iterator(); i.hasNext();) {
    String item = i.next();
    System.out.println(item);
}
```

```
for (String line: stringList) {
    System.out.println(line);
}
```

Переменная среды CLASSPATH

- Определяется в операционной системе
- Указывает JVM, где необходимо искать файлы .class
- Может ссылаться на каталоги и файлы .jar и .zip
- Интерпретатор загружает встроенные классы перед тем, как загрузить пользовательские
- Используется с командами **`java`** и **`javac`**

Выполнение JAVA программ

- Для того, чтобы класс можно было запустить, в нем должен быть определен метод **`main`**

```
public class FirstProgram {
    public static void main (String[] args){
        int itemsSold = 10;
        float itemCost = 11.0f;
        int i, j, k;
        double interestRate;
    }
}
```

Компиляция и запуск JAVA программ

- Компиляция .java файла

```
> javac [-classpath <path>] FirstProgram.java
```

- Запуск .class файла

```
> java [-classpath <path>] FirstProgram
```

Язык программирования JAVA

Создание собственных классов

Работа с памятью

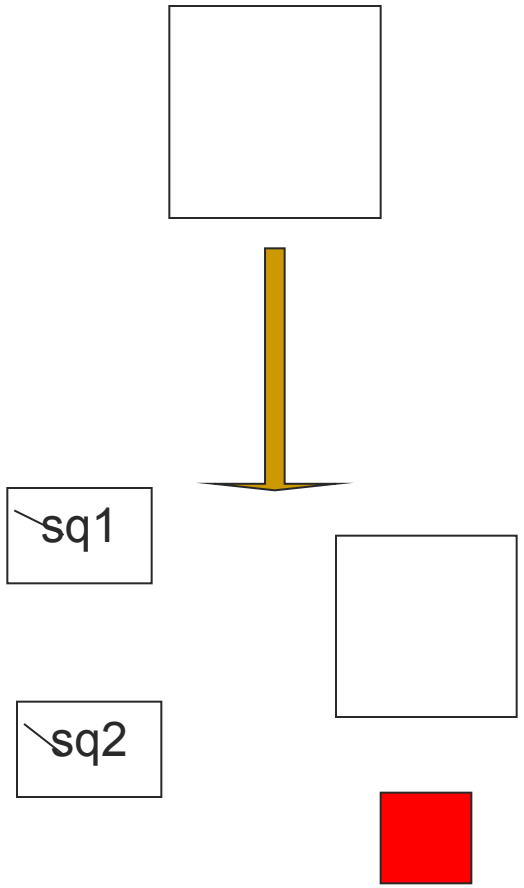
Класс

- Класс – шаблон для создания объекта
  - Поля – для хранения данных (простые и ссылочные)
  - Методы – для определения поведения
- Логически связанные классы объединяются в пакеты
- Пакеты также применяются для управления доступом к классам

Классы и объекты

```
public class Square{
   private double sideLength = 6.0;
   private Color color;
   . . .
}
```

```
{
  Square sq1 = new Square(5.0);
  sq1.setColor (new Color(Color.WHITE));
  Square sq2 = new Square(2.7);
  sq2.setColor(new Color(Color.RED));
}
```



Создание объектов

- Экземпляры класса (объекты) создаются при помощи оператора **`new`**

```
ClassName objectRef = new ClassName();
```

- Пример: создание двух квадратов

```
Square sq1 = new Square();

Square sq2 = new Square();
```

Оператор **new**

- Оператор выполняет следующие действия:
  - Выделяет участок памяти для нового экземпляра
  - Вызывает специальный метод класса, называемый конструктором
  - Возвращает ссылку на созданный объект

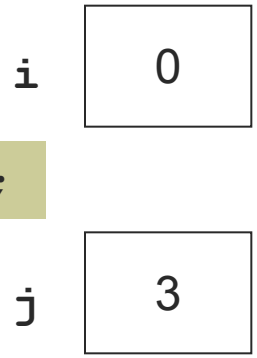
```
Square sq1 = new Square();

Square sq2; //sq2 равно null

sq2 = new Square();
```

Простые и объектные типы

- Простые переменные содержат значения
- Объектные переменные содержат ссылки

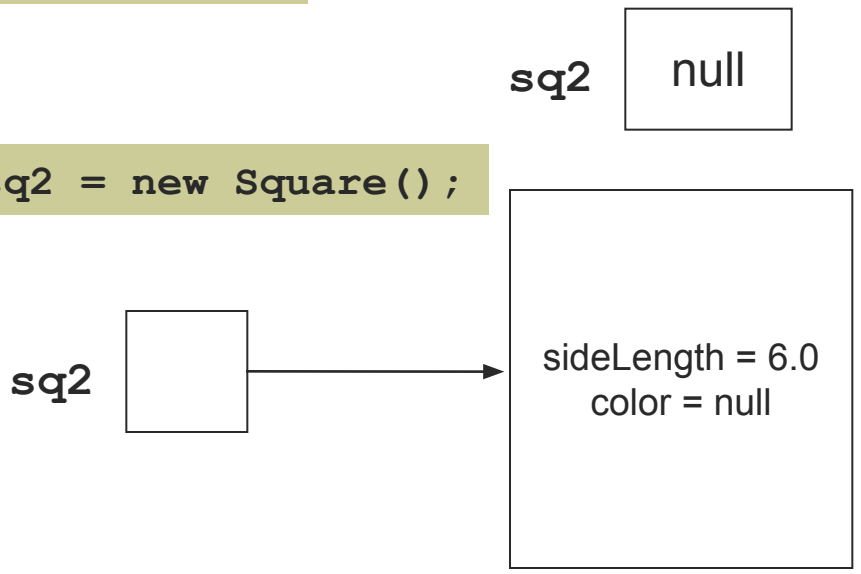
```
int i;
```

```
int j = 3;
```




```
Square sq2;
```

```
sq2 = new Square();
```



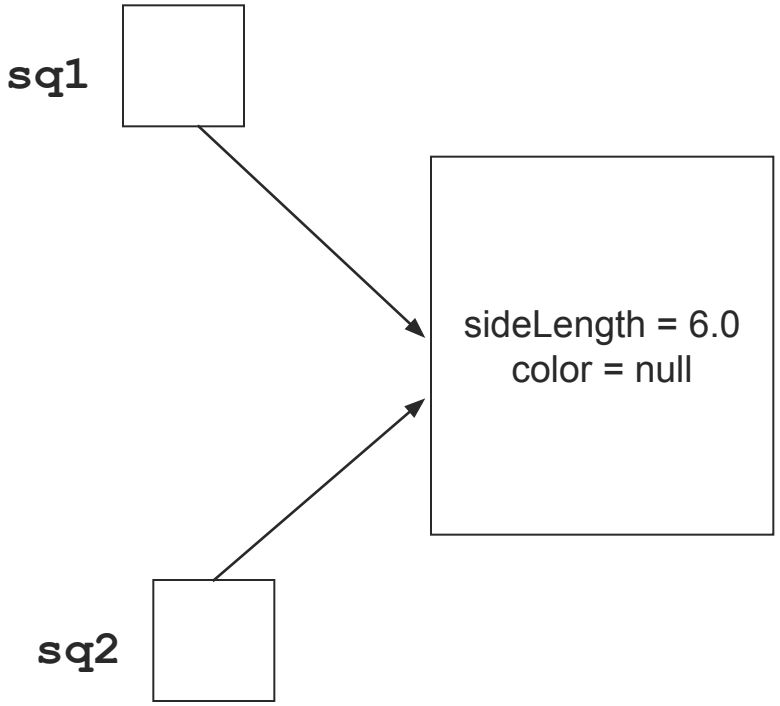
Что такое **`null`**?

- Показывает, что ссылка на объект пуста
- Может быть присвоено только объектной переменной
- Можно сравнивать объектные переменные с **`null`**
- Объект может быть удален сборщиком мусора после присвоения **`null`** его переменной

```
Square sq1;

. . .

if (sq1 == null){

 sq1 = new Square();

}

...

sq1 = null;
```

Присвоение объектных переменных

```
Square sq1 = new Square();
```




```
Square sq2 = sq1;
```

```
sq1 = null;
```

```
sq2 = null;
```

Объявление полей

- Поля класса объявляются внутри класса, но вне методов или инициализационных частей

```
public class Square{
   private double sideLength = 6.0;
   private Color color;
   public double getArea(){
      double result = sideLength * sideLength;
           return result;
   }
}
```

Инициализация полей класса

- Поля могут быть явно инициализированы при объявлении
- Присваивание значений происходит в момент создания экземпляра класса

```
public class Square{
   private double sideLength = 6.0;
   private Color color;
   . . .
}
```

- Все поля инициализируются неявно значениями по умолчанию
- Сложные инициализации нужно **помещать в конструктор**

Доступ к полям класса

- Public поля могут быть прочитаны или изменены при помощи оператора “.” (точка)

```
public class Square{
   public double sideLength = 6.0;
. . .
}
```

```
Square sq1 = new Square();
sq1.sideLength = 5.22;
if (sq1.sideLength >6){
. . .
}
```

- Применять такой подход НЕ РЕКОМЕНДУЕТСЯ
- Как правило, используется только для объявления констант

Объявление методов

- Метод – эквивалент процедуры или функции в других языках программирования

```
[access-modifier] [modifier] returnType methodName
([argumentList]){
//тело метода
. . .
}
public class Square{
. . .
   public double getArea(){
      double result = sideLength * sideLength;
           return result;
   }
}
```

Вызов методов

- Все методы определяются только внутри классов
- Вызов метода всегда происходит в контексте объекта или класса
- В скобках передаются аргументы метода
- Если у метода нет аргументов, то скобки оставляются пустыми

```
Square sq = new Square();
sq.setColor(new Color(Color.RED));
System.out.println("Area is "+sq.getArea());
```

Задание аргументов для методов

- Количество и типы аргументов задаются при объявлении метода

```
public class Square{
   public double getArea(){
     . . .
   }
   public void setLenAndColor(double len, Color cl){
     . . .
   }
}
```

Передача аргументов в метод

- Все аргументы в Java передаются ПО ЗНАЧЕНИЮ, т.е. для аргумента делается локальная копия и метод работает с ней
- Для параметров простых типов передаются значения
- Для параметров объектных типов передается КОПИЯ ССЫЛКИ

Возврат значения из метода

- Для возврата значения используется оператор

```
public class Square{
. . .
   public double getArea(){
      double result = sideLength * sideLength;
           return result;
   }
}
```

- Если возвращаемый тип метода **`void`**, то оператор **`return`** обычно не пишется
- Оператор **`return`** без значения пишется для прерывания методов типа **`void`**

Вызов методов

- Для вызова методов используется оператор “.” (точка)
- В скобках перечисляются фактические аргументы
- Если у метода нет аргументов, пустые скобки все равно нужно ставить

```
public class Square{
   private double sideLength = 6.0;
   private Color color;
   public double getArea(){
      double result = sideLength * sideLength;
           return result;
   }
      sq.setColor(new Color(Color.RED));
}     System.out.println(“Area is ”+sq.getArea());
```

Перегрузка методов

- Методы в классе могут иметь одинаковое имя
- У методов ДОЛЖЕН БЫТЬ разный набор аргументов (разные сигнатуры)
- Компилятор различает вызовы методов по типам и количеству параметров
- Методы не могут различаться по типу возвращаемого значения

Инкапсуляция

- Правила
  - Поля класса должны быть объявлены private (исключения - константы)
  - Доступ к полям должен осуществляться только через методы
- Преимущества
  - Легкость модификации (сохраняется интерфейс класса)
  - Простота контроля за присваиваемыми значениями

Getter’ы и Setter’ы

- Для работы с полями создаются методы следующего вида:

```
public PropertyClass getPropertyName (){
    return propertyName;
}
```

```
public void setPropertyName (PropertyClass value){
    propertyName = value;
}
```

Ссылка `this`

- В методах можно использовать переменную this, которая представляет из себя ссылку на текущий объект (экземпляр класса)

```
public class Square{
private double sideLength;
public void setSideLength(double newLength){
    this.sideLength = newLength;
}
}
```

Конструкторы

- Для правильной инициализации в классе может быть создан конструктор
- Конструктор – метод, вызываемый автоматически при создании объекта
  - Обычно объявляется как public
  - Должен иметь то же имя, что и класс
  - Для конструктора не указывается возвращаемый тип
- По умолчанию компилятор создает конструктор без аргументов, если конструкторы не указаны явно. В противном случае используются только конструкторы, которые указаны в классе
- Конструкторы могут быть перегружены

Пример

```
public class Square{
private double sideLength;
public void setSideLength(double newLength){
    this.sideLength = newLength;
}
public double getSideLength(){
    return sideLength;
}
public Square(double sideLength){
    this.sideLength = sideLength;
}
}
```

```
Square sq = new Square();
Square sq = new Square(3.0);
```

Разделение кода между конструкторами

- Конструктор может вызывать другой конструктор, используя **`this()`**

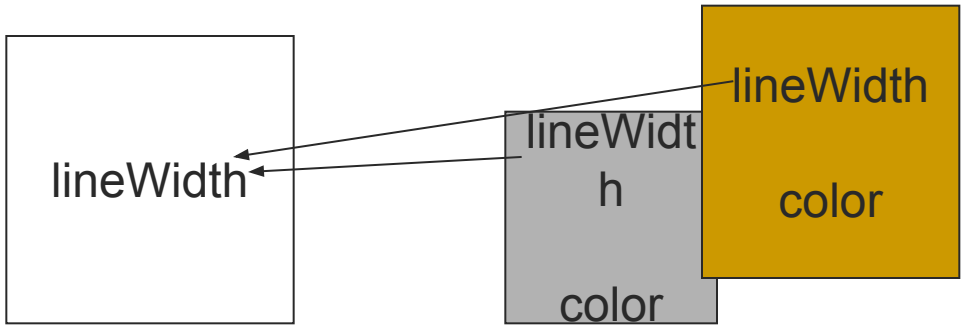
```
public class Square{

private double sideLength;

public Square(){

    this(3.5);

}

public Square(double sideLength){
            this.sideLength = sideLength;
            }

}
```

- Вызов **`this()`** должен быть первым оператором в конструкторе
- Аргументы **`this()`** должны совпадать с аргументами вызываемого конструктора

Статические поля

- Поля, принадлежащие только классу и общие для всех экземпляров класса
- Объявляются при помощи модификатора **`static`**

```
public class Square{
private Color color;
private static int lineWidth;
}
```



Инициализация статических полей

- Статические поля могут быть проинициализированы при объявлении
- Для сложной инициализации используется статический блок инициализации

```
public class Square{
private Color color;
private static int lineWidth;
static{
    lineWidth = 0.75;
}
}
```

Статические методы

- Статические методы разделяются всеми экземплярами класса
- Могут работать со статическими полями
- При объявлении используется модификатор **`static`**

```
public class Square{
private static int lineWidth;
public static int getLineWidth(){
    return lineWidth;
}
}
```

- Могут вызываться, используя имя класса или имя объекта

```
Square sq1 = new Square();
Square.getLineWidth();
sq1.getLineWidth();
```

Неизменяемые поля, методы и классы

- Поля с модификатором final – это константа и оно не может быть изменено
  - Но может быть проинициализировано
  - Обычно объявляется как public static для использования вне класса
- Метод с модификатором final не может быть перекрыт в наследнике
- Класс с модификатором final не может быть унаследован

Пример

```
public final class Color{
public final static Color black = new Color(0,0,0);
public final static Color black = new Color(0,0,255);
. . .
public static final int BLACK = 0;
public static final int RED = 255;
. . .
}
```

Метод `finalize()`

- Если объект работает с ресурсами (например, с файлом), то он должен освободить эти ресурсы перед тем, как он будет удален из памяти
- Для этого используется метод **`finalize()`**
- Метод вызывается перед тем, как объект будет убран из памяти
- Время, когда будет вызван **`finalize()`** не определено

Работа с памятью

- Когда все ссылки на объект пустые, он помечается для удаления
- Сборщик мусора периодически освобождает память, занимаемую объектами
- Сборка мусора происходит автоматически
- Для сигнала, что можно запустить сборщик мусора, можно воспользоваться методом `System.gc();`

Вопросы?