Лекция 1

Введение в ООП
Основы Java

Кириленко М. С.

История ООП

- **1950-1960** – зарождение идеи «объектно-» «ориентированного»
- **1960**е – появление объектно-ориентированного языка Simula.
- **1970**е – появление полностью объектно-ориентированного языка Smalltalk. В этом языке объектами является ВСЁ:
  - Строки
  - Числа
  - Логические значения
  - Блоки кода
  - Память

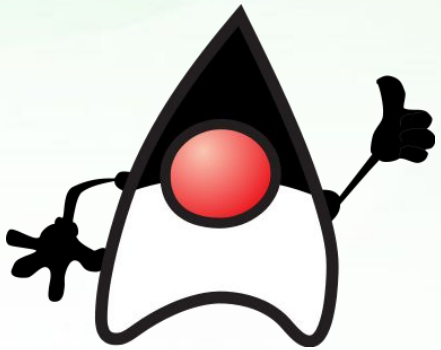
История ООП

- **1990**е – рост популярности графических интерфейсов пользователя (GUI), которые основывались на техниках ООП

- В это время методология ООП поддерживается многими языками:
  - C++
  - Delphi (Object Pascal)
  - Java
  - Visual FoxPro
  - И т.д.

История версий Java

- **1990** – Oak
- **1996** – JDK 1.0
- **1997** – JDK 1.1
- **1998** – J2SE 1.2
- **2000** – J2SE 1.3
- **2002** – J2SE 1.4
- **2004** – J2SE 5.0
- **2006** – Java SE 6
- **2011** – Java SE 7
- **2014** – Java SE 8
- **2017** – Java SE 9
- **20 марта 2018** – Java SE 10
- **25 сентября 2018** – Java SE 11
- **19 марта 2019** – Java SE 12
- **17 сентября 2019** – Java SE 13



Sun microsystems

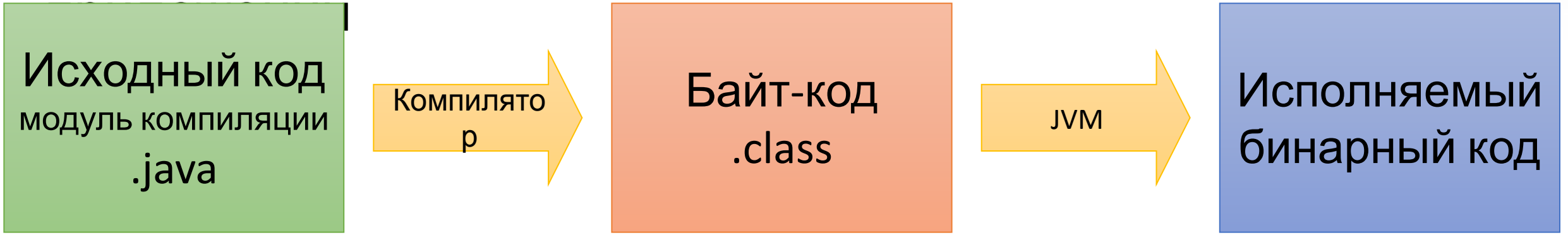
ORACLE

Особенности Java

- Кроссплатформенность (переносимость)
- Объектная ориентированность
- Привычный синтаксис, схожий с C и C++
- Язык сильно типизированный; компилируемый в байт-код
- Сборщик мусора
- Встроенная модель безопасности
- Ориентация на сетевые распределённые приложения
- Лёгкость развития
- Лёгкость освоения
- JavaScript – не Java!

Java Virtual Machine (JVM)

- Создаётся для каждой платформы, которая должна поддерживать запуск и функционирование Java-приложений
- Скрывает особенности платформы
- Обеспечивает единую среду исполнения для Java-



- Java Runtime Environment (JRE) – минимальная реализация виртуальной машины, состоящая из JVM и библиотеки Java-классов. Не содержит компилятора.

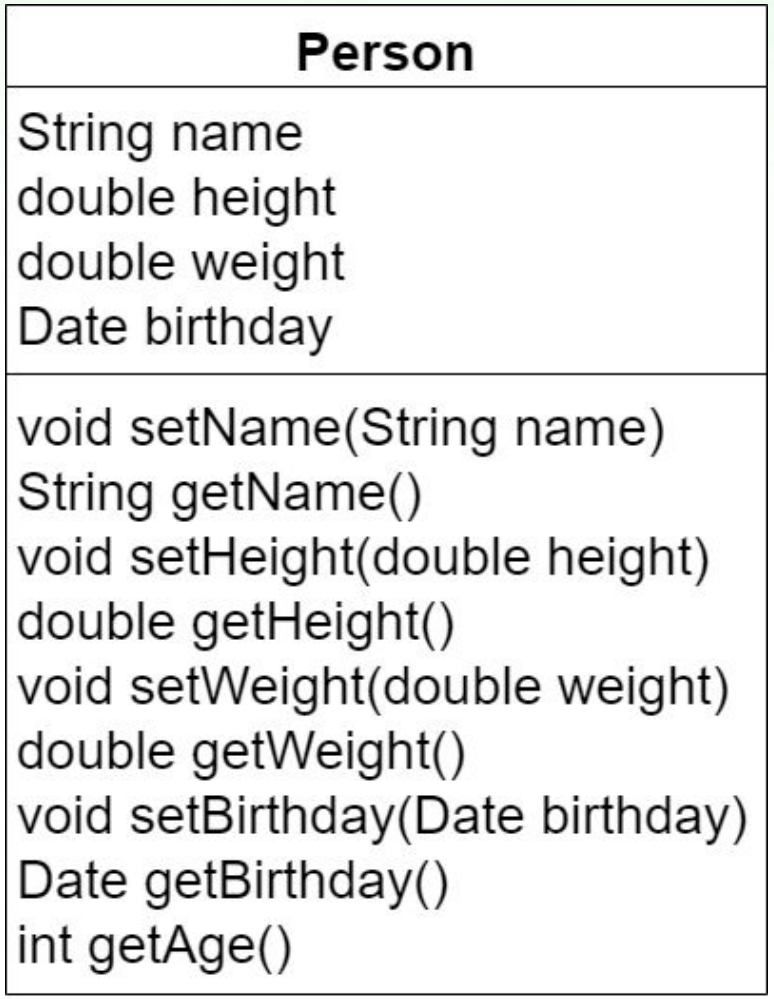
Java Development Kit (JDK)

- Комплект инструментов для разработки приложений на Java
- Включает в себя:
  - Компилятор (javac)
  - Стандартные библиотеки Java
  - Документацию
  - Примеры
  - Утилиты
  - JRE

- Начиная с 11 версии Java, Oracle JDK не является бесплатным для коммерческого использования
- OpenJDK – проект JDK, состоящий из свободного и открытого исходного кода

Объекты и классы

- В Java любой объект представляет собой экземпляр определённого класса
- Класс – шаблон поведения объектов определённого типа с заданными параметрами, определяющими состояние
- Все экземпляры одного и того же класса имеют один и тот же набор свойств и общее поведение



Название класса

Поля класса

Методы класса

Классы

- Код программы на Java представляет собой набор классов
- Классы могут иметь *статические* поля и методы
- Поле или метод называется статическим, если оно принадлежит контексту класса, а не контексту экземпляра этого класса (объекта)

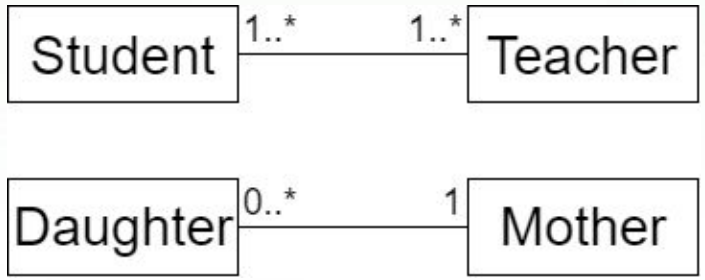
Person
String name double height double weight Date birthday
void setName(String name) String getName() void setHeight(double height) double getHeight() void setWeight(double weight) double getWeight() void setBirthday(Date birthday) Date getBirthday() static int getAge(Date birthday)

Конструкторы и деструкторы

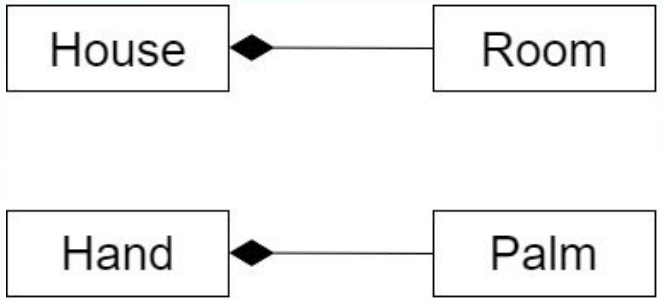
- Конструкторы – особые методы, которые вызываются при создании объекта
  - Необходимы для задания корректного состояния нового объекта
  - Возвращаемый тип конструктора – сам класс
  - Возвращаемый объект – новый экземпляр этого класса
- Деструктор – особый метод, который вызывается при уничтожении объекта
  - Необходим для освобождения ресурсов, использующихся объектом
  - В Java деструкторов нет
  - Для освобождения ресурсов рекомендуется использовать собственные методы с «говорящим» названием (close(), dispose(), clear() и т.п.)

Отношения

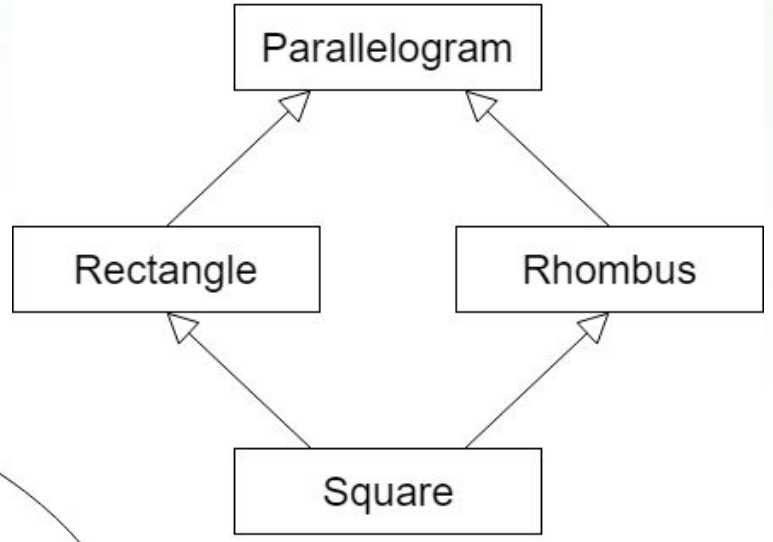
- Ассоциация



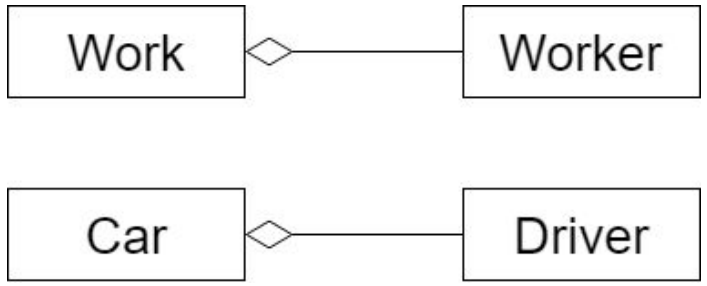
- Композиция



- Наследовани

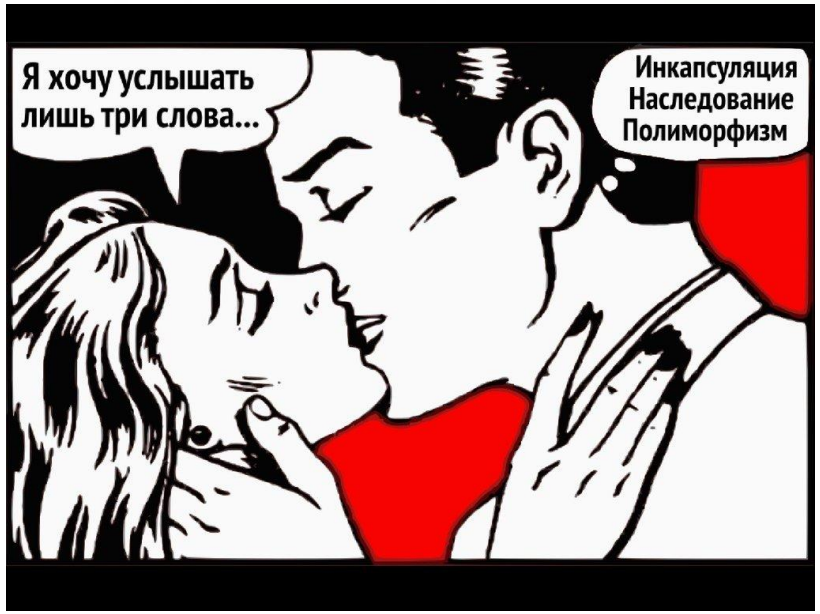


- Агрегация



Основные принципы ООП

- **Инкапсуляция** – сокрытие реализации класса и отделение его внутреннего представления (реализации) от внешнего (контракта/интерфейса)
- **Наследование** – отношение между классами, при котором класс использует структуру или поведение другого класса (одиночное наследование) или других классов (множественное наследование)
- **Полиморфизм** – свойство, позволяющее объектам с одинаковой спецификацией (контрактом/интерфейсом) иметь



Достоинства ООП

- Абстрагирование от деталей реализации
- Локализация кода и разделение функциональности
- Возможность создания расширяемых систем
- Единообразная обработка разнородных данных
- Изменение поведения во время выполнения
- Меньше затрат времени на разработку
- Уменьшение дублирования кода
- И т.д.

Недостатки ООП

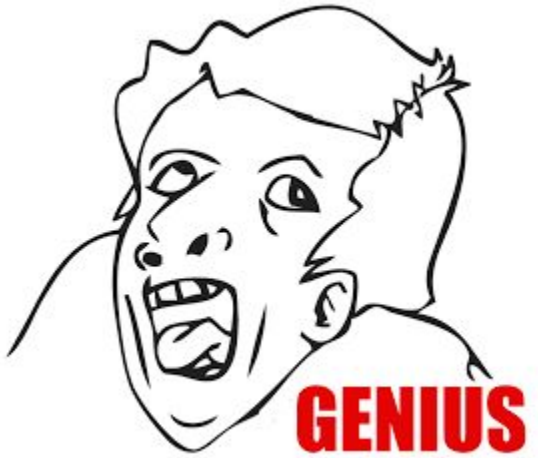
- Усложнённое документирование
- Сложность проектирования
- Затруднённая навигация внутри и между классами в сложных иерархиях классов
- Возможность злоупотребления наследованием
- Неэффективность на этапе выполнения
- Неэффективность в смысле распределения памяти
- Излишняя универсальность
- Ещё один зачёт в семестре
- И т.д.

Объекты и Java

- В Java все сущности являются объектами…

  либо классами или интерфейсами…

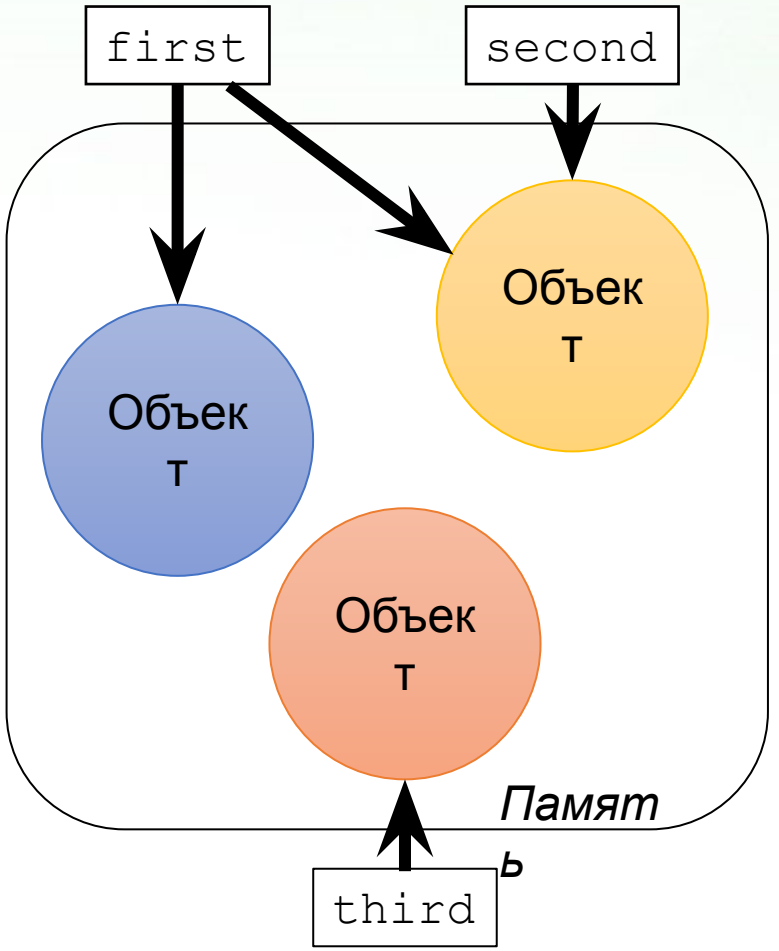
  и те, и другие на самом деле тоже являются объектами
- Все классы являются наследниками класса **Object**
- Все классы и интерфейсы являются объектами класса **Class**
- Класс **Class** всё ещё является наследником класса **Object**
- Реализовано одиночное наследование от классов
- Реализовано множественное наследование от интерфейсов



Виртуальная память Java

- Классы и интерфейсы являются ссылочными типами
- Во время выполнения программы ссылка может поменять своё значение и начать ссылаться на другой объект
- Ссылка – это не указатель!

```
Object first =
new Object();
Object second =
new Object();
first = second;
Object third =
new Object();
third = null;
```



Пакеты

- Программа на Java состоит из набора пакетов
- Пакеты предназначены для логической группировки типов
- Пакет может содержать вложенные пакеты
- Пакет может содержать классы и интерфейсы (типы)
- Пакет имеет своё пространство имён, что позволяет создавать одноимённые классы в различных пакетах
- Способы организации пакета:
  - Как обычная файловая структура
  - Как JAR-файл
- Переменная окружения CLASSPATH позволяет указать путь к используемым пакетам
  - Но вместо неё можно указать путь при запуске JVM

Имена

- Имена используются для доступа к конструкциям языка
- Имена классов и интерфейсов:
  - Простые: **Object**
  - Составные: java.lang.**Object** – включают в себя имя пакета
- Именованные конструкции

Java:

- Пакеты
- Классы
- Интерфейсы
- Поля
- Методы
- Внутренние/вложенные классы/интерфейсы
- Аргументы методов
- Аргументы конструкторов
- Аргументы обработчиков ошибок
- Локальные переменные
- Аннотации
- ~~Метки~~

HelloWorld.java

```
public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }

}
```

- Метод `public static void main(String[] args)` является точкой входа программы

Модуль компиляции

```
package ru.ssau.tk;

import java.util.Date;

public class Person {
    private Date birthday;

    public Date getBirthday() {
        return birthday;
    }

    public void setBirthday(Date birthday) {
        this.birthday = birthday;
    }
}
```

Объявление пакета

Выражение импорта

Объявление верхнего уровня

- **Модуль компиляции** хранится в .java-файле
- Является единичной порцией входных данных для компилятора
- Состоит из трёх частей

Модуль компиляции

- **Объявление пакета** указывает, к какому пакету будут принадлежать объявляемые ниже типы
  - Если отсутствует, то типы располагаются в безымянном пакете (пакете по умолчанию)
- **Выражения импорта** позволяют импортировать типы (и статические элементы) в модуль компиляции для обращения к ним по простым именам
  - Могут отсутствовать
- **Объявления верхнего уровня** могут содержать одно или несколько объявлений типов
  - А могут и не содержать
  - Не рекомендуется делать более одного объявления

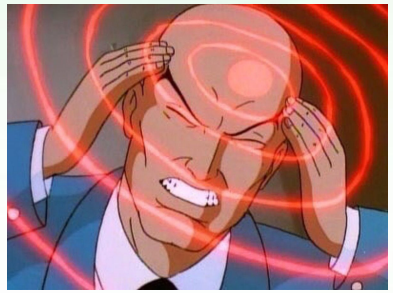
Правила именования

- Пакеты:
  - java.lang, com.google.android, java.sql, ru.ssau.tk
- Типы (классы и интерфейсы):
  - Object, Class, Date, TextStyle, Runnable, Serializable, NullPointerException
- Поля и локальные переменные:
  - birthday, name, visible, editedBy, totalDistanceKm, timeFromStartMs
- Методы:
  - getName(), setBirthday(), isVisible(), toString(), equals(), writeObject()
- Завершённые (константные) поля:
  - PI, MAX_VALUE, NEGATIVE_INFINITY, PARAMETER_NAME

Рекомендации именования

- Называйте переменную так, чтобы было понятно, за что она отвечает
- Не используйте сокращений (за исключением общепринятых)
- Старайтесь не использовать цифры в переменных
- Если ваш код работает, но никто не может в нём

Когда надо придумать имя переменной…



```
class x {
    x x;
    x x(x x) {
        x:
        for (;;) {
            while (x.x(x) != x) {
                x.x = x;
                break x;
            }
        }
        return x;
    }
}
```

Объявление класса

- Самое простое создание класса:

```
class MyClass {
}
```

- Такая запись эквивалентна наследованию (**расширению**) от класса **Object:**

```
class MyClass extends Object {
}
```

- Расширение от другого класса:

```
class Child extends Parent {
}
```

Наследование классов

- Транзитивность наследования: если класс **A** является наследником класса **B**, а класс **B** является наследником класса **C**, то класс **A** также является наследником класса **C**
- Циклическое наследование недопустимо: если класс **A** является наследником класса **B**, то класс **B** не может являться наследником класса **A**
- Отсутствует множественное наследование классов: если класс **A** является наследником класса **B**, а класс **B** не является наследником класса **C**, то класс **A** не может являться наследником класса **C**

Поля и методы

```
class Item {

    String name;
    int cost = 0;

    void increaseCost(int value) {
        cost += value;
    }

    String getDescription() {
        return "This is the " +
                name + ", $" + cost;
    }

}
```

Поля

Методы

- Поля хранят в себе данные
- Методы – поведение
- Вызов метода всегда записывается со скобками
- Поле и метод могут иметь одинаковые имена

Объявление полей

```
int x, y;
double weight = 0;
Point a, b = null, c = new Point();
String description = getDescription();
```

- Если переменная не инициализирована, то ей присваивается значение по умолчанию:
  - Для числовых полей примитивных типов – 0
  - Для логического типа – false
  - Для ссылочного типа – null
- Рекомендуется объявлять каждую переменную на отдельной строке
- Запрещено иметь два поля с одинаковым именем

Объявление методов

- В объявлении метода обязательно нужно указать тип возвращаемого значения (или void), имя метода и список аргументов
- Имя и аргументы метода (количество, тип и порядок следования) определяют **сигнатуру** метода
- Класс не может иметь два метода с одинаковыми сигнатурами
- После объявления метода следует его реализация, если только метод не является *абстрактным*
- Внутри методов можно создавать локальные переменные, обращаться к доступным полям и методам своего и других объектов (а также классов и интерфейсов)

Методы

- Пример метода и его реализации:

```
int getArea(int width, int height) {
    return width * height;
}
```

- Пример вызова метода:

```
int x = getArea(3, 5);                    // 15
x = getArea(2, 7);                        // 14
getArea(6, 3);                            // 18
System.out.println(getArea(10, 20)); // 200
```

- Аргументы передаются по значению

Переменное число аргументов

- Есть возможность создания метода с переменным числом аргументов (только в конце метода!):

```
void save(String path, File... files) { … }
double average(int... numbers) { … }
```

- Примеры вызовов:

```
save("D:/folder/", firstFile, secondFile);
save("D:/directory/", thirdFile);
writer.save("D:/empty/");
double x = average(1, 3, 4);
double y = calculator.average(-5, 5);
double z = average();
```

Переменное число аргументов?

- На самом деле число не переменное, а последний аргумент представляет собой массив:

```
void save(String path, File[] files) { … }
double average(int[] numbers) { … }
```

- Разница в том, что в этом случае, в отличие от предыдущего, в метод нужно обязательно передавать массив:

```
save("D:/folder/", new File[] {firstFile, secondFile});
save("D:/directory/", new File[] {thirdFile});
save("D:/empty/", new File[] {});
double x = average(new int[] {1, 3, 4});
double y = average(new int[] {-5, 5});
double z = average(new int[] {});
```

Переменное число аргументов

- Сигнатуры методов будут идентичными и, таким образом, в классе не могут существовать два таких метода:

```
double average(int... numbers) { … }   (1)
double average(int[] numbers) { … }    (2)
```

- Не зависимо от того, какой из методов объявлен, **(1)** или **(2)**, в реализации аргумент (`numbers`) является массивом
- Метод **(1)** можно вызывать двумя способами:

```
double x = average(1, 3, 4);
double y = average(new int[] {1, 3, 4});
```

- Метод **(2)** можно вызывать только через массив:

```
double x = average(new int[] {1, 3, 4});
```

Реализация метода

- Метод, возвращаемый тип которого не void, обязан вернуть значение этого типа с помощью ключевого слова return
- После возврата значения метод заканчивает своё выполнение, так что после return никаких других действий быть не должно
- Компилятор проводит анализ структуры метода, чтобы гарантировать, что при любых операторах ветвления возвращаемое значение будет сгенерировано

```
int printSum(int a, int b) {
    int sum = a + b;
    System.out.println("Result is " + sum);
    return sum;
}
```

Ветвление в методе

```
int get1(boolean condition) {
    if (condition) {
        return 4;
    } else {
        return 5;
    }
}

int get2(boolean condition) {
    if (condition) {
        return 4;
    }
    return 5;
}
```

- Эти методы будут работать одинаково
- Если метод ничего не возвращает, то return также можно использовать для досрочного выхода из метода:

```
void setAge(int age) {
    if (age < 0) {
        return;
    }
    this.age = age;
}
```

Ссылка объекта на себя

- Внутри нестатического контекста класса можно получить ссылку объекта на себя при помощи ключевого слова this

```
public class Detail {

    int version = 1;

    void setVersion(int version) {
        this.version = version;
    }

    void print() {
        System.out.println(this);
    }

    …
```

Поле

Аргумент

Ссылка объекта на себя

```
Detail returnNewestDetail(Detail otherDetail) {
    if (version > otherDetail.version) {
        return this;
    }
    return otherDetail;
}
```

Родительские классы

- Доступ к родительским полям и методам класса можно получить при помощи ключевого слова super (но это не ссылка!)

```
public class Rectangle {

    private double width;
    private double height;

    public void setSize(double width, double height) {
        this.width = width;
        this.height = height;
    }

}
```

Переопределение методов

```
public class Square extends Rectangle {

    @Override
    public void setSize(double width, double height) {
        if (width != height) {
            // TODO: incorrect data
            return;
        }
        super.setSize(width, height);
    }

}
```

Перегрузка методов

- Два или более методов класса называются перегруженными, если они имеют одинаковое имя, но разные сигнатуры

```
public void setSize(double width) {
    setSize(width, width);
}

public void setSize(double width, double height) {
    …
}

public void setSize(double height, double width) {
    …
}
```

Настоятельные рекомендации

- Называйте переменную так, чтобы было понятно, за что она отвечает
- Метод должен выполнять одну задачу, он должен выполнять её хорошо и ничего другого он делать не должен
- Пишите код так, как будто сопровождать его будет склонный к насилию психопат, который знает, где вы живёте
  - Или не сопровождать, а проверять
- Всегда держите под рукой:
  - google.com
  - translate.google.com
  - книгу: Роберт Мартин, «Чистый код. Создание, анализ и рефакторинг»
- Старайтесь не передавать в методы null и не возвращать из методов null
  - Но если очень хочется, используйте аннотацию @Nullable

Итоги

- Принципы ООП
  - Инкапсуляция, наследование, полиморфизм
- Отношения между объектами и классами
  - Ассоциация, агрегация, композиция, наследование
- Классы и объекты
  - Ссылочный тип
  - Пакеты
  - Статические и нестатические поля и методы
  - Конструкторы ~~и деструкторы~~
  - Переопределение методов и перегрузка методов