Java Object Oriented Programming (OOP)

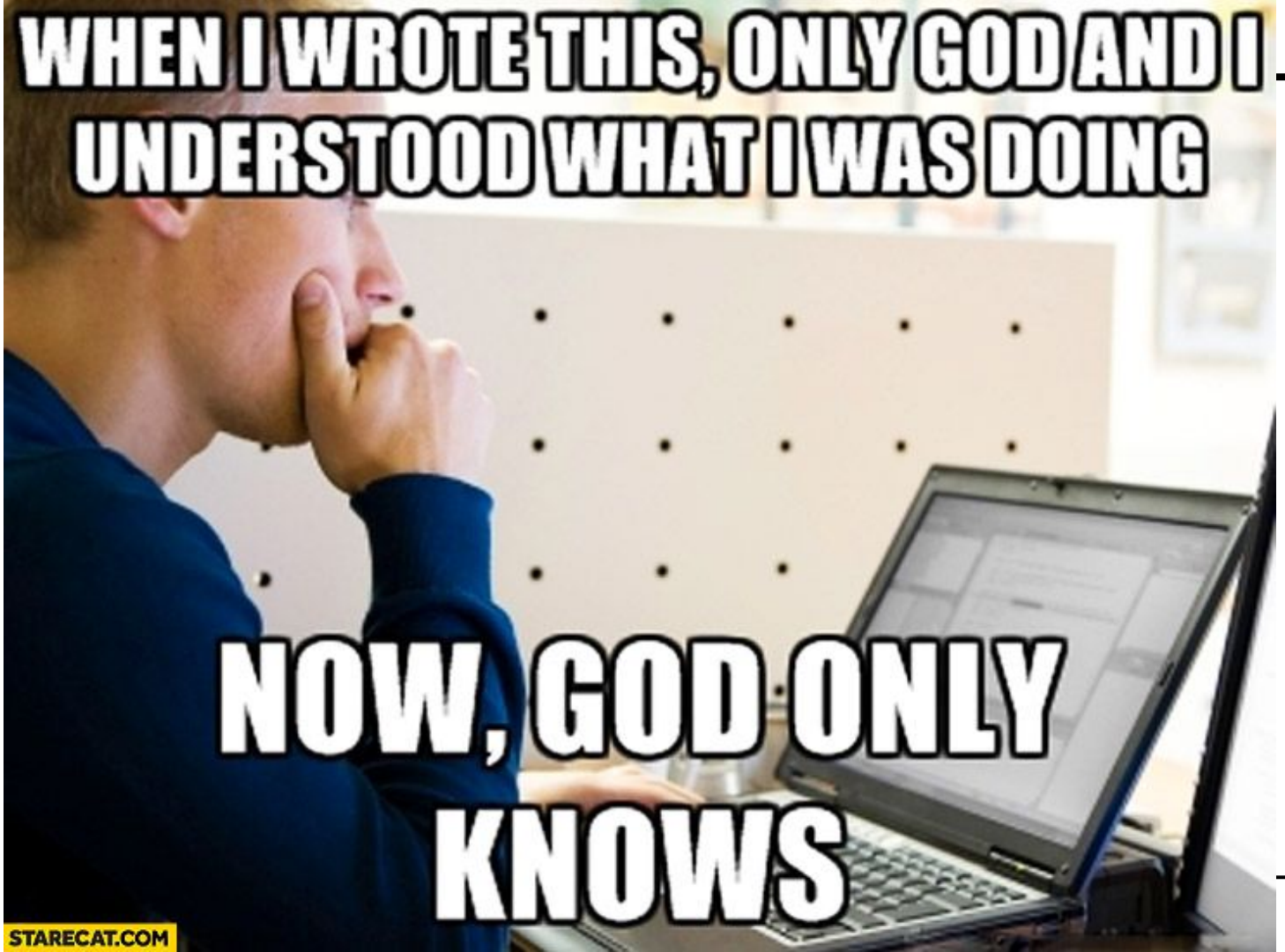
Wojciech Frącz
AGH IEiT
fracz@agh.edu.pl

Rules

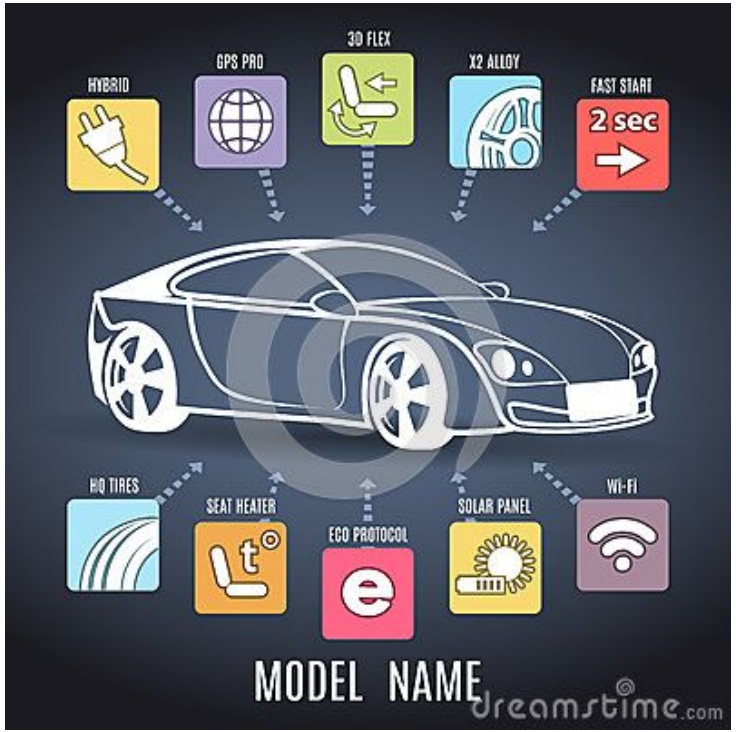
- test at the beginning of every class (0-100)
- some classes end up with a homework (0-100)
- final grade = 80% * average from tests + 20% * average from homework
- you need to have 50 to pass
- we will adjust it to your scale 0-20 at the end of the course
- for those who don't make it: final test (strongly not suggested)
- class attendance is required

OOP

- allows to represent "things" from real life (the problem domain) in your source code
- helps with the problem understanding
- facilitates the implementation
- enables decomposition of the problem ("divide and conquer")
- spaghetti code
  - https://pl.wikipedia.org/wiki/Spaghetti_code

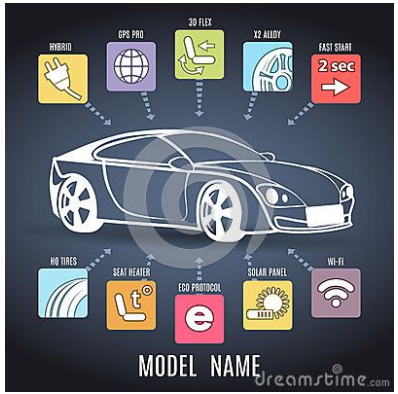

WHEN I WROTE THIS, ONLY GOD AND I UNDERSTOOD WHAT I WAS DOING
NOW, GOD ONLY KNOWS
STARECAT.COM

Class



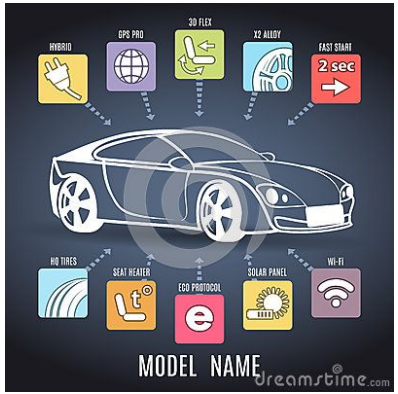

```
class Car {

}
```

Class



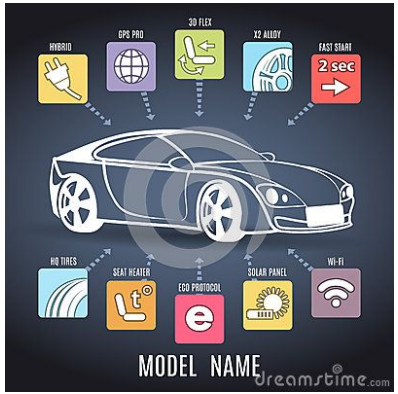

```
class Car {
  void turnOn() { System.out.println("Wrr"); }
}
```

Class



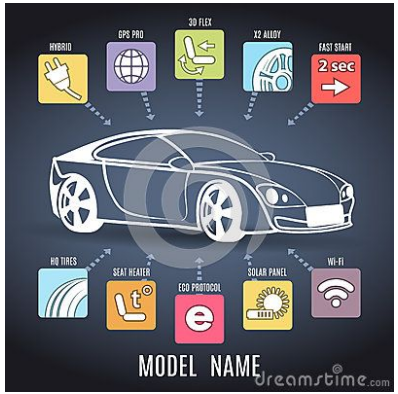

```
class Car {
  void turnOn() { System.out.println("Wrr"); }
  void turnOff() { System.out.println("Psss"); }
}
```

Class



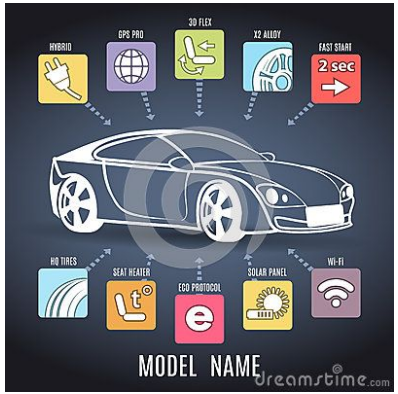

```
class Car {
  void turnOn() { System.out.println("Wrr"); }
  void turnOff() { System.out.println("Psss"); }
  void drive() { System.out.println("Brum"); }
}
```

Class



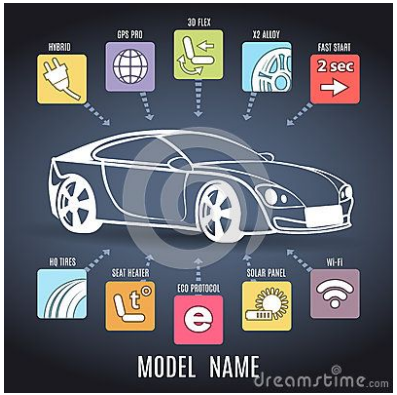

```
class Car {
  void turnOn() { System.out.println("Wrr"); }
  void turnOff() { System.out.println("Psss"); }
  void drive() { System.out.println("Brum"); }
  void stop() { System.out.println("Piii"); }
}
```

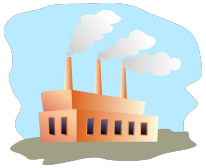
Class




```
class Car {
  void turnOn() { System.out.println("Wrr"); }
  void turnOff() { System.out.println("Psss"); }
  void drive() { System.out.println("Brum"); }
  void stop() { System.out.println("Piii"); }
  void turnLeft() {System.out.println("Left!");}
  void turnRight() {System.out.println("Right!");}
}
```

Objects





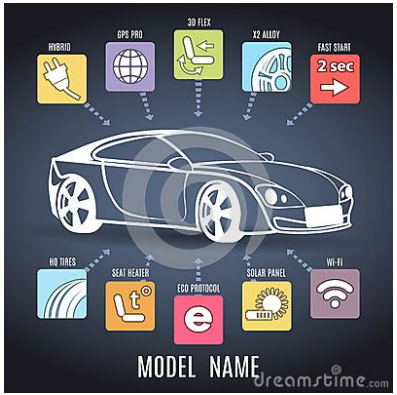
```
class Car {
}
```

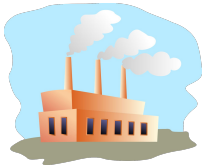
a class

```
Car myCar = new Car();
```

an object (instance)

Objects





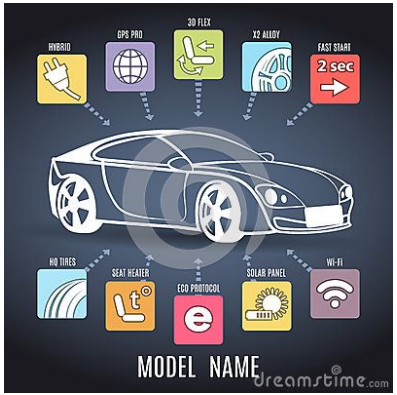
```
class Car {
}
```

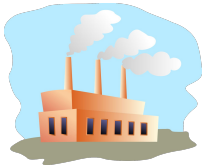
a class

```
Car car1 = new Car();
Car car2 = new Car();
Car car3 = new Car();
```

objects (instances)

Objects





```
class Car {
}
```

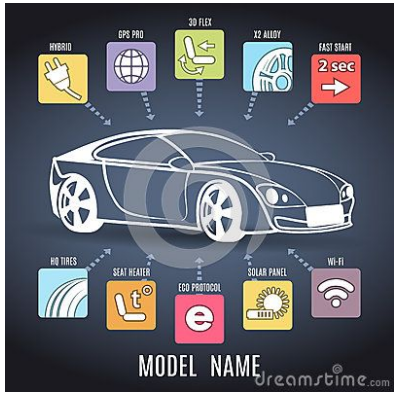
```
Car car1 = new Car();
Car car2 = new Car();
Car car3 = new Car();
car2.drive();
```

Quiz!

```
class Car {
  void turnOn() { System.out.println("Wrr"); }
  void turnOff() { System.out.println("Psss"); }
  void drive() { System.out.println("Brum"); }
  void stop() { System.out.println("Piii"); }
  void turnLeft() {System.out.println("Left!");}
  void turnRight() {System.out.println("Right!");}
}
```

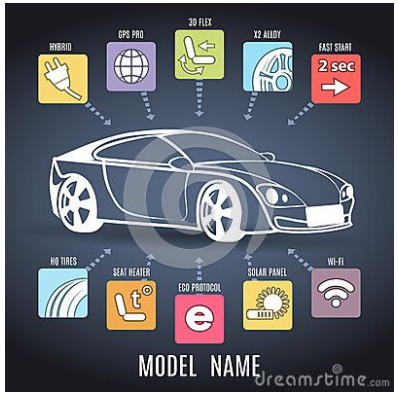
```
public class MyGarage {
    public static void main(String[] args) {
        Car myCar = new Car();
        myCar.turnOn();
        myCar.drive();
        myCar.turnRight();
        myCar.stop();
        myCar.turnOff();
    }
}
```

Decomposition



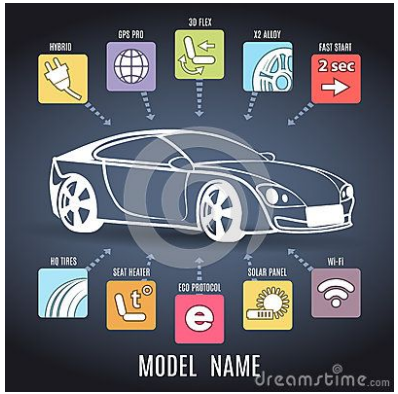
- a car does not move itself
  - it has an engine
- it does not steer itself
  - it has a steering wheel
- different parts of the car can be manufactured by different producents
- the same parts can be used in different cars

Decomposition




```
class Car {

  Engine engine;

  SteeringWheel steeringWheel;

}
```

Decomposition




```
class Car {
  Engine engine;
  SteeringWheel steeringWheel;
  void turnOn() { engine.turnOn(); }
  void turnOff() { engine.cutOffTheFuel(); }
  void drive() { engine.morePower(); }
  void stop() { engine.lessPower() }
  void turnLeft() { steeringWheel.left(); }
  void turnRight() { steeringWheel.right(); }
}
```

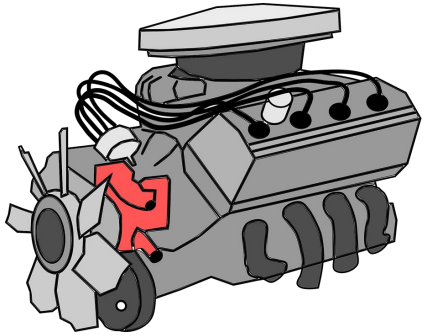
Decomposition

```
class SteeringWheel {
  void left() { System.out.println("Left!"); }
  void right() { System.out.println("Right");}
}
```

Decomposition

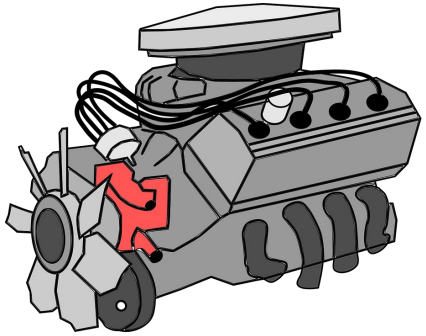
```
class SteeringWheel {
  void left() { System.out.println("Left!"); }
  void right() { System.out.println("Right!");}
  void adjust() { /* ... */ };
}
```

Decomposition



```
class Engine {
  void turnOn() { System.out.println("Wrrr"); }
  void cutOffTheFuel() { System.out.println("Psss"); }
  void morePower() { System.out.println("Brum"); }
  void lessPower() { System.out.println("Piii"); }
}
```

Decomposition

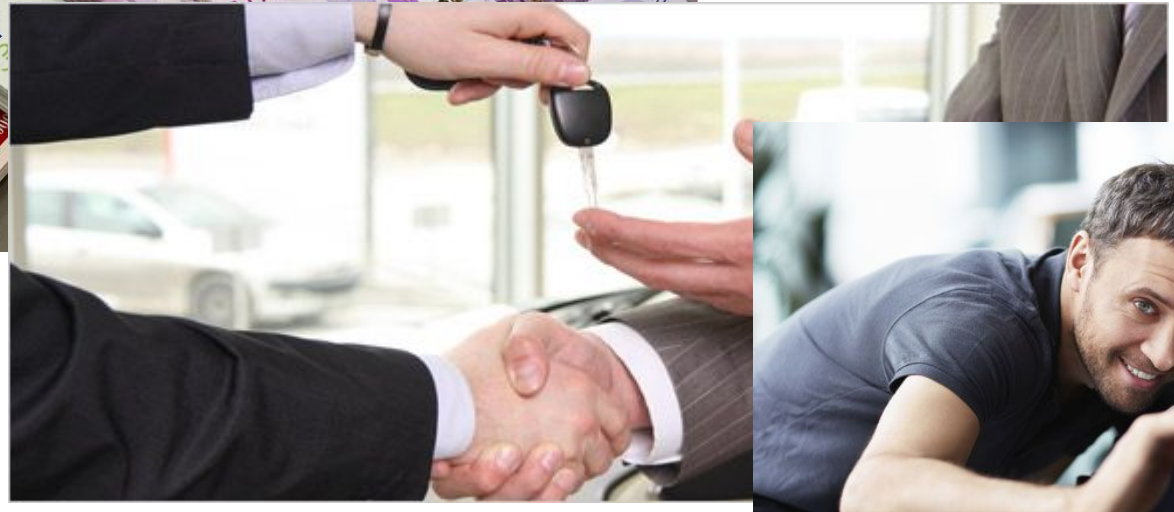


```
class Engine {
  SparkPlug plug;
  void turnOn() {
    plug.heat();
    plug.spark();
  }
  void cutOffTheFuel() { System.out.println("Psss"); }
  void morePower() { System.out.println("Brum"); }
  void lessPower() { System.out.println("Piii"); }
}
```

Quiz!

```
class Car {
  String color;
  /* … */
}
```

```
public class DontTrustYourWife {
    public static void main(String[] args) {
        Car myCar = new Car();
        myCar.color = "black";
        Car wifeCar = myCar;
        wifeCar.color = "pink";
        System.out.println(myCar.color);
    }
}
```



```
Car myCar = new Car();
myCar.color = "black";
```

```
Car wifeCar = myCar;
```


```
wifeCar.color = "pink";
```

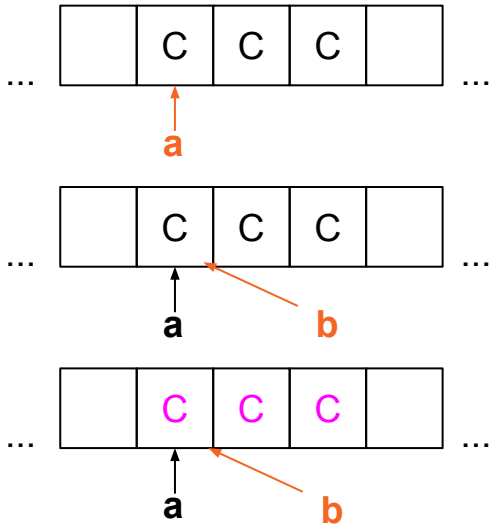
```
System.out.println(myCar.color);
```

```
Car myCar = new Car();
myCar.color = "black";
```

```
Car wifeCar = new Car();
wifeCar.color = "pink";
```

References

```
Car a = new Car();
a.color = "black";
Car b = a;

a.color = "pink";
```



In complex types the **references** are assigned to variables.

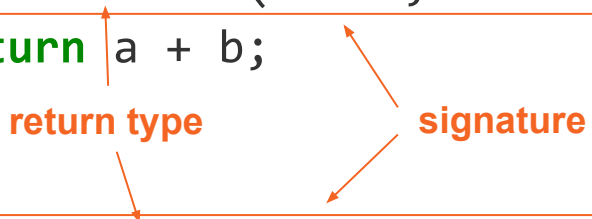
Methods

```
class Calculator {
  public int add(int a, int b) {
    return a + b;
  }

  public boolean isNegative(int a) {
     return a < 0;
  }
}
```

```
Calculator c = new Calculator();
int num1 = 10;
int num2 = -10;
int sum = c.add(num1, 19);
if (c.isNegative(num2)) {
  // ...
}
```

Methods

```
class Calculator {
  public int add(int a, int b) {
    return a + b;
  }

  public boolean isNegative(int a) {
    return a < 0;
  }
}
```




```
Calculator c = new Calculator();
int num1 = 10;
int num2 = -10;
int sum = c.add(num1, 19);
if (c.isNegative(num2)) {
  // ...
}
```

Methods

```
class Calculator {
  public int add(int a, int b) {
    return a + b;
  }

  public boolean isNegative(int a) {
     return a < 0;
  }
}
```

parameters

```
Calculator c = new Calculator();
int num1 = 10;
int num2 = -10;
int sum = c.add(num1, 19);
if (c.isNegative(num2)) {
  // ...
}
```

arguments

Constructor = initializes an object

```
Car myCar = new Car();
```

```
class Car {
}
```

==

```
class Car {
  public Car() {
  }
}
```

Class fields / attributes = object's state

```
class Car {
  private String color;

  public Car(String color) {
    this.color = color;
  }

  public String getColor() {
    return this.color;
  }
}
```

Access modifiers

```java
class Car {
  private String color;

  public Car(String color) {
    this.color = color;
  }

  public String getColor() {
    return this.color;
  }
}
```

```java
Car autoZony = new Car("pink");

String kolorAutaZony = autoZony.getColor();

String kolorAutaZony2 = autoZony.color;
autoZony.color = "black";
```

✓

✘

✘

this

```
class Car {
  private String color;

  public Car(String color) {
    this.color = color;
  }

  public String getColor() {
    return color;
  }

  public void setColor(String newColor) {
    color = newColor;
  }
}
```

- refers to the object, for which the code is being executed for
- can be omitted when accessing fields or methods, unless they are hidden by local variables or parameters

Sample class

```java
class Car {
  private String color;

  public Car(String color) {
    this.color = color;
  }

  public String getColor() {
    return color;
  }

  public void setColor(String color) {
    this.color = color;
  }
}
```

```java
Car ferrari = new Car("red");

String color = ferrari.getColor(); ✓
ferrari.setColor("black"); ✓
ferrari.getColor(); ✓
// void result = ferrari.setColor("pink");
```

✗

Mystery solved!

```
Scanner in = new Scanner(System.in);
```

Methods overloading

```
class Car {
  private double tankCapacity;
  private double fuel;

  public double fill(double howMuch) {
    fuel += howMuch;
    return tankCapacity - fuel;
  }

  public double fill() {
    return fill(tankCapacity - fuel);
  }

  //public boolean fill(double howMuch) {
  //  fuel += howMuch;
  //  return tankCapacity <= fuel;
  //}
}
```

- allows to define default parameter values
- overloaded methods have the same name, but must differ in parameters list (types or quantity)
- changing only the return type is not sufficient

Methods overloading

```
class Car {
  private double tankCapacity;
  private double fuel;

  public double fill(double howMuch) {
    fuel += howMuch;
    return tankCapacity - fuel;
  }

  public double fill() {
    return fill(tankCapacity - fuel);
  }

  //public boolean fill(double howMuch) {
  //  fuel += howMuch;
  //  return tankCapacity <= fuel;
  //}
}
```

- allows to define default parameter values
- overloaded methods have the same name, but must differ in parameters list (types or quantity)
- changing only the return type is not sufficient

QUIZ!: What is missing in the `fill(double howMuch)` method?

What can you create in Java?

applications	Java as desktop or mobile applications
applets	Java as programs that can be launched in web browser
servlets	Java as server applications

http://docs.oracle.com/cd/E19798-01/821-1770/6nmnum0kr/index.html

What can you create in Java?

applications	Java as desktop or mobile applications
~~applets~~	~~Java as programs that can be launched in web browser~~
servlets	Java as server applications

http://docs.oracle.com/cd/E19798-01/821-1770/6nmnum0kr/index.html

Java editions

Java SE (Standard Edition)	Applications and applets
Java EE (Enterprise Edition)	Servlets
Java ME (Micro Edition)	Applications for small devices, like mobile phones or embedded systems.

http://docs.oracle.com/cd/E19798-01/821-1770/6nmnum0kr/index.html

Java API

- Each Java version comes with JDK
- Besides tools (e.g. `java`, `javac`), JDK contains Application Programming Interface (API), aka *Standard library*
- Ready to use classes, which provide features for our programs

Java API - history

JDK	Date	Number of classes
Java SE 9 / JDK 1.9.0	2017	6005
Java SE 8 / JDK 1.8.0	2014	4240
Java SE 7 / JDK 1.7.0	2011	4024
Java SE 6 / JDK 1.6.0	2006	3793
Java 2 SE 5.0 / JDK 1.5.0	2004	3279
Java 2 SE / SDK 1.4.0	2002	2991
Java 2 SE / SDK 1.3	2000	1842
Development Kit 1.0	1996	212

Java API - examples

java.lang	Basic elements (e.g. System, String) - implicite imported
java.util	Complex data tpyes (e.g. collections), dates, random number generator
java.io	Input/Output operations (I/O) - e.g. file access
java.math	Math tools (e.g. `BigDecimal`)
java.net	Network communication support

```java
import java.util.Random;
import java.util.Arrays;
public class Loto {
    public static void main(String[] args) {
        int[] lottery = new int[6];
        Random random = new Random();
        for (int draw = 0; draw < 6; draw++) {
            lottery[draw] = random.nextInt(49) + 1;
        }
        Arrays.sort(lottery);
        System.out.println(Arrays.toString(lottery));
    }
}
```

Packages

Class defined in the project's sources root (src) - default package

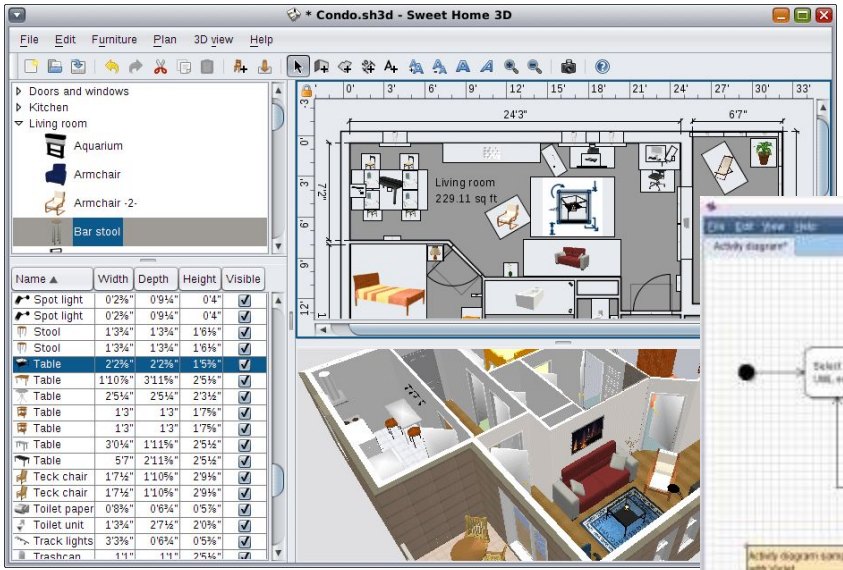
```
public class Lotto {

}
```

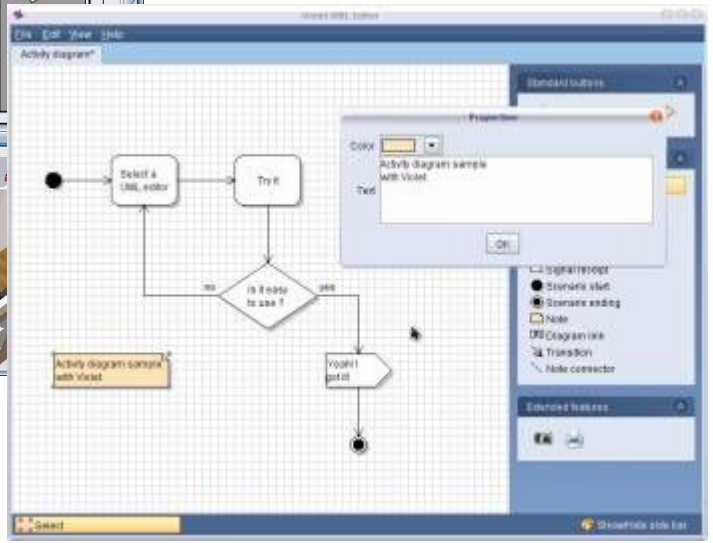
Packages

Class defined in src/pl/edu/agh/lottery
(package pl.edu.agh.lottery).

```
package pl.edu.agh.lottery;

public class Lotto {

}
```

Java applications - examples





Java Servlets - examples

Literature

- *Head First Java* - Kathy Sierra & Bert Bates
- *Effective Java* - Joshua Bloch
- *Java Puzzlers* - Joshua Bloch
- Java Concurrency in Practice - Brian Goetz
- *Thinking in Java* - Bruce Eckel
- Java Language Specification (JLS)
  https://docs.oracle.com/javase/specs/

Glossary

http://www.wordclouds.com/

Quiz!

```
public class GoodByeJava {
    public static void main(String[] args) {
        Integer a = 200;
        Integer b = 200;
        System.out.println(a == b);
    }
}
```