Object-oriented programming

Session 3

27th of February 2019
Ljubov Jaanuska
How many hours did you spend on the course last week?
How do you succeed in this course?
I would like to start with
• **Primitive**
  - Whole numbers
    • `byte`
    • `short`
    • `int`
    • `long`
  - Fractions
    • `float`
    • `double`
  - Logical
    • `boolean`
  - Symbols
    • `char`

• **Object-oriented (reference)**
  - ...
  - Class as a new data type!
Paradigms

Process-oriented model
Withdraw, deposit

Data-oriented model
Client, money, account
Object-oriented languages

• The main focus on data
  – secondary focus on *what to do with the data*

• Assume we have an object DOG

• What a DOG can DO?
  – related activities: eat, play, sleep...
  – Not FEED, but EAT
  – The dog does not feed itself

• Object-oriented languages
  – Java, C++, Smalltalk, Ruby
  – also Python
OOP concepts

• In OOP languages, the focus is on objects
• A program is a collection of related objects
  – not a collection of procedures
• Each object has:
  – **Identity** – unique for each object
  – **State** – data of an object (different terms: attribute, property, instance) field, instance variable
  – Object **behavior** is defined by methods (procedures than an object can do e.g. change object state
Object, class, instance

- **An object** is an instance of a class with its **state** and **behavior**.
- **A class** - a template from which objects are created; basically, a user defined new data type.
- **An instance** - a reference to an individual object.
Attention!

• In some examples, we are going to use some tricks which do not meet OOP conventions
• This is done on purpose to introduce OOP concepts step by step
Identity

class BoxId {
}

BoxId box1 = new BoxId();

BoxId box2 = new BoxId();
class Box{
    double length;
    double width;
    double height;
}

Box box1 = new Box();

box1.width = 3.6;

System.out.println(box1.width);
class Box{
    double length;
    double width;
    double height;

    double volume() {
        return length * width * height;
    }

    void greet() {
        System.out.println("Hi!");
    }
}

Methods
Why do we need classes and where do we use them?

- Class is like a template which describes an object (its state and behavior)

```java
Box box1 = new Box("Paul", 2, 3, 4);
Box box2 = new Box("John", 3, 5, 3.4);

String box1Name = "Paul";
double box1Length = 2;
double box1Width = 3;
double box1Height = 4;
String box2Name = "John";
double box2Length = 3;
double box2Width = 5;
double box2Height = 3.4;
```
Constructor

• a class member with the same name as the class
• a special type of **methods** that is invoked by operator *new* to create and initialize objects.

• the name of a method is the **class name**
• there is **no return type** because a constructor **always returns an object of its own class**
• overloading is possible
  – a suitable constructor is chosen by the number of arguments and/or their data types
• every class a constructor
  – if a class does not have an explicit constructor, JRM builds a default constructor - a **zero argument constructor**
class Box{
    double length;
    double width;
    double height;
}

public Box(String name, double length, double width, double height) {
    this.name = name;
    this.length = length;
    this.width = width;
    this.height = height;
}

Instance fields (instance variables)

Constructor

Methods
Multiple constructors

```java
public Box(double side, int factor) {
    name = "";
    double newBox = side*factor;
    length = newBox;
    width = newBox;
    height = newBox;
    System.out.println("New box!");
}
```

this???
Keyword **this**

- Reference to the class members
  - reference to the instance field when the parameter name is the same as the instance field name
    - ```
    public Box(String name, double length) {
        this.nimi = name;
        this.length = length;
    }
    ```
  - call one constructor from other constructors
    - ```
    public Box(){
        this("",0.0,0.0,0.0,0.0);
    }
    ```
public Box(String name, double length, double width, double height) {
    this.nimi = name;
    this.length = length;
    this.width = width;
    this.height = height;
}

public Box(String name) {
    this.nimi = name;
    this.length = 1;
    this.width = 1;
    this.height = 1;
}

public Box(){
    this("",0.0,0.0,0.0);
}
Multiple constructors

```java
public Box(String name, double length, double width, double height) {
    this.nimi = name;
    this.length = length;
    this.width = width;
    this.height = height;
}

public Box(String name) {
    this.nimi = name;
    this.length = 1;
    this.width = 1;
    this.height = 1;
}

public Box() {
    this("", 0.0, 0.0, 0.0);
}
```

Box box1 = new Box("Paul", 4.8, 2, 3);
Box box3 = new Box("John");
Box box5 = new Box();
A constructor name is the same as

1. one of the instance field names
2. class name
3. nothing
4. a method name from the same class
To use overloading, it is important that methods have different

1. names
2. length
3. return type
4. signature
5. nothing
public class Pair {
    int i;
    double d;
    Pair(int i, double d) {
        this.i = i * 10;
        this.d = d;
    }
}

Pair p1 = new Pair(2, 4.1);
System.out.println(p1.i);

1. 20
2. 20.0
3. something else
4. error
What is the output?

```java
public class Pair {
    int i;
    double d;
    Pair(int i, double d) {
        this.i = i * 10;
        this.d = d;
    }
}
```

Pair p1 = new Pair(2.0, 4.1);
System.out.println(p1.i);

1. 20
2. 20.0
3. something else
4. error
What is the output?

```
public class Pair {
    int i;
    double d;
    Pair(int i, double d) {
        this.i = i * 10;
        this.d = d;
    }
}

Pair p1 = new Pair(2, 4);
System.out.println(p1.i);
```

1. 20  
2. 20.0  
3. something else  
4. error
What is the output?

1. 20
2. 20.0
3. something else
4. error

```java
public class Pair {
    int i;
    double d;
    Pair(int i, double d) {
        this.i = i * 10;
        this.d = d;
    }
    Pair(double d, int i) {
        this.i = i;
        this.d = d * 10;
    }
}
```

```java
Pair p1 = new Pair(2.0, 4);
System.out.println(p1.d);
```
What is the output?

```
public class Pair {
    int i;
    double d;
    Pair(int i, double d) {
        this.i = i * 10;
        this.d = d;
    }
    Pair(double d, int i) {
        this.i = i;
        this.d = d * 10;
    }
}
```

1. 20
2. 20.0
3. something else
4. error
Reference semantics

Box box1;

null
Reference semantics

Box box1;
box1 = new Box();

Instance of class Box
Reference semantics

Box box1;
box1 = new Box();
Box box2 = new Box();
Box box1;
box1 = new Box();
Box box2 = new Box();
box2 = box1;
Box box1;
box1 = new Box();
Box box2 = new Box();
box2 = box1;

Garbage Collection
Java already has classes

• Java API
• https://docs.oracle.com/en/java/javase/11/docs/api/
Modifier

• a keyword at the beginning of a class, a field, a method and other members that sets accessibility

• multiple modifiers are separated by a space (order is not important)

• Examples
  - public, protected, private
  - static
  - final
  - abstract

- Access (or visibility labels)
  - Cannot be directly instantiated
  - Cannot be changed

- Class modifier (not object modifier)
Modifiers

• Most OOP languages support hiding the implementation details from the clients - encapsulation
• Fields and methods can be public or private
• Private fields and methods are visible inside the class; public fields and methods are visible everywhere
  – Common convention: fields are private, methods are public
• Easy to maintain large programs
Access modifiers

• public, protected, private, not specified

• If an access modifier is not specified, the class, method or field is accessible within a package
Package

• Grouped classes
• Each class belongs to a package
• Double check the package – some classes have the same name, but placed in different packages
  – java.util.List
  – java.awt.List

• Hierarchical structure
• Import classes from different packages
  – import java.util.Scanner;
  – import java.util.*;

• Since Java 9 packages are grouped into modules
package p1;
public class K1 {
    public int x;
    int y;
    private int z;

    public void m1(){
    }
    void m2(){
    }
    private void m3(){
    }
}

package p1;
public class K2 {
    public static void main(String[] args) {
        K1 k = new K1();
        int a = k.x;
        int b = k.y;
        //int c = k.z;
        k.m1();
        k.m2();
        //k.m3();
    }
}
package p1;
public class K1 {
    public int x;
    int y;
    private int z;

    public void m1(){
    }
    void m2(){
    }
    private void m3(){
    }
}

package p2;
import p1.*;
public class K3 {
    public static void main(String[] args) {
        K1 k = new K1();
        int a = k.x;
        //int b = k.y;
        //int c = k.z;
        k.m1();
        //k.m2();
        //k.m3();
    }
}
private double length;

public double getLength(){
    return length;
}

public void setLength(double length){
    this.length = length;
}
An instance field that cannot be changed afterwards

```java
public class Person{
    private long personID;

    public Isik(long personID) {
        this. personID = personID;
    }
    public long getPersonID() {
        return personID;
    }
}
```

• Method `setPersonID` can help
If an access modifier is not set, a method can be called out

1. within the same class
2. within the same package
3. everywhere
Will the following line cause an error:
static public void main(String[] args)

1. yes
2. no
Will the following line cause an error:
public static void main(String[] a)

1. yes
2. no
Method `toString`

- returns a string which contains information about an object in a readable way
  - without method `toString`
    ```java
    Box@50cbc42f
    ```
  - with method `toString`
    ```java
    I am a box Paul: length=2.0, width=3.0, height=4.0
    ```

```
public String toString() {
    return "I am a box " + name + ": length=" + length + ", width=" + width + ", height=" + height;
}
```
An object as an argument

• In class Box

```java
void greet(Box anotherBox) {
    System.out.println("Hi, " + anotherBox.name);
}
```

• In class BoxTest

```java
box1.greet(box2);
```
Class method and instance method

• Class methods

  static double average(double num1, double num2)
  – Can be called out everywhere where the class is accessible (objects are not needed)
    • the main method
      public static void main (String[] args)
    • methods of class java.lang.Math

• Instance methods

  void greet()
  – can be called out by an object using the dot notation, e.g. box2.greet()
  – inside the class klass use keyword this.
Class field and instance field

• Class field

  static int a

  – Can be called out everywhere where the class is accessible (objects are not needed)

  – ei ole isendi osaks

  – Fields of class java.lang.Math

    • static double E, use Math.E

• Instance field

  int b

  – can be used only with an object using the dot notation

  – inside the class klass use keyword this.
Note

• One class can contain both class and instance fields and methods
What is the output?

```java
public class Pair {
    static int factor;
    int i;
    double d;
    Pair(int i, double d) {
        this.i = i * factor;
        this.d = d;
    }
}

Pair p = new Pair(2, 4.1);
System.out.println(p.i);
```

1. 2
2. 0
3. something else
4. error
Default values

• Local variables do not have default values
• Instance fields have default values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Default Value (for fields)</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>0</td>
</tr>
<tr>
<td>short</td>
<td>0</td>
</tr>
<tr>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>long</td>
<td>0L</td>
</tr>
<tr>
<td>float</td>
<td>0.0f</td>
</tr>
<tr>
<td>double</td>
<td>0.0d</td>
</tr>
<tr>
<td>char</td>
<td>\u0000'</td>
</tr>
<tr>
<td>String (or any object)</td>
<td>null</td>
</tr>
<tr>
<td>boolean</td>
<td>false</td>
</tr>
</tbody>
</table>

http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html
What is the output?

```java
public class Pair {
    static int factor;
    int i;
    double d;
    Pair(int i, double d) {
        this.i = i * factor;
        this.d = d;
    }
}
```

```java
Pair.factor = 10;
Pair p = new Pair(2, 4.1);
System.out.println(p.i);
```

1. 20
2. 0
3. something else
4. error
What is the output?

```java
class Salary{
    static int tax;
    static int bruto;
    int neto;
    String worker;

    public Salary(int bruto, String worker) {
        this.bruto = bruto;
        this.worker = worker;
        this.neto = bruto * (100-tax)/100;
    }
}

Salary.tax = 20;
Salary p1 = new Salary(1000,"John Smith");
System.out.println(p1.tax);
```

1. 1000
2. 20
3. 0
4. 800
5. smth else
6. error
What is the output?

class Salary{
    static int tax;
    static int bruto;
    int neto;
    String worker;

    public Salary(int bruto, 
                    String worker) {
        this.bruto = bruto;
        this.worker = worker;
        this.neto = bruto * 
                    (100-tax)/100;
    }
}

Salary t1 = new Salary(1000, "John Smith");
Salary t2 = new Salary(500, "Anna Oak");
System.out.println(t1.bruto);
class Salary{
    static int tax;
    int bruto;
    int neto;
    String worker;

    public Salary(int bruto, String worker) {
        this.bruto = bruto;
        this.worker = worker;
        this.neto = bruto * (100-tax)/100;
    }
}

Salary t1 = new Salary(1000, "John Smith");
Salary t2 = new Salary(500, "Anna Oak");
System.out.println(t1.bruto);
What is the order of the class parts?

1. instance fields, class fields, constructors, instance methods
2. class fields, instance fields, constructors, instance methods
3. class fields, constructors, instance methods, instance fields
4. instance fields, class fields, instance methods, constructors