Object Oriented Programming in JAVA

As you know before everything in Java is an object, and must have his type. You allready know some primitive types like String, char, dobule, int etc.  
  
Now what will happen when you will create your own „type” in Java?  
You need just a write simple class. Let’s have an example, we would like a type, which keep information about name and surname the Employee/Person?

*public class Person {*

*private String name;*

*private String surname;*

*public String getName() {*

*return name;*

*}*

*public void setName(String empName) {*

*this.name = empName;*

*}*

*public String getSurname() {*

*return surname;*

*}*

*public void setSurname(String empSurname) {*

*this.surname = empSurname;*

*}*

*}*

This class is somekind of structure. The structure decides what kind of types you have inside.

Inside class Person we have two members – name and surname as type of String, which are declared with magic keyword „private”.

**Access Modifiers**

**private –** modifier specifies that the member can only be accessed its own class.

Example:

*pubilc class Person {*

***private*** *String name;*

*}*

**protected** – modifier specifies that the member can only be accessed within its own package and in addition by a subclass of its class in another package.

*pubilc class Person {*

***protected*** *String name;*

*}*

**public** – modifier means that all code can access the class, field, constructor or method, regardless of hwere the accessing code is located

*pubilc class Person {*

**public** *String name;*

*}*

**package (default)** – modifier means that code inside the class itself + code inside classes in the same package as this class can access the class, filed, constructor or method.

*pubilc class Person {*

*String name;*

*}*

Members in the class shouldn’t be marked as public, because this break the **encapsulation.**This means, we should declare private members, and public methods which access to those members, otherwise our code will be very easliy to break, or overwrite by someone else.  
  
Example (declare all members as a public).

*public class Person {*

***public*** *String name;*

***public*** *String surname;*

*public String getName() {*

*return name;}*

*}*

*public void setName(String empName) {*

*this.name = empName;*

*}*

*public String getSurname() {*

*return surname;*

*}*

*public void setSurname(String empSurname) {*

*this.surname = empSurname;*

*}*

*}*

Now when we will create some Launcher class with main method which will create our **new instance** of the class Person in static method, like follows.

*public class Launcher {*

*public static void main(String[] args) {*

*Person p = new Person();*

*p.setName("Kate");*

*System.out.println(p.getName());*

*p.name = "ABC";*

*System.out.println(p.name);*

*}*

*}*

Now imagine that you are not change the name of the Person by method but using simple assigmnets, you don’t know who is change it and when. Imagine this is very similar to have profile on society portal without password – anyone who knows your name and surname can login…. because member variable is public – is visible to everyone, and accessible for everyone at any time!

**public static void main(String[] args)**

It’s time to explain what does mean *public static void main(String[] args)* – imagine that something is static is ready to use, you don’t have to create an „object” to use.  
This method is very often described like universal „entry point” to your application.

String[] args – it’s just an array of String arguments which you provide to run. If you not provide any argument method just will execute code between clamps ({ …}).

Example:

*package pl.info;*

*public class Launcher {*

*public static void main(String[] args) {*

*String[] info = new String[3];*

*info[0] = args[0];*

*info[1] = args[1];*

*System.out.println("Your name is " + info[0] + " and you are " + info [1] + " years old ");*

*}*

*}*

You can run this program in IDE in some specific way, or from command line (very old manner, still usefull and used on Linux community).

**Methods in JAVA**

As you can see on code snippets there are some methods like public void, public String etc.  
Key concept is that in Java are two types of method (sometimes called function)

Syntax: *[access\_modifier]* ***void*** *[name\_of\_method]*

**Example**

*public void setName(String empName){*

*this.name=empName;*

*}*

This mean the method named „setName” takes an argument with type of String (empName) and assign the value empName to variable this.name and finish exectution of the method.

You must remember that void – **returns completly nothing** - **it’s void**! It’s used for modify some values, or state of object/variable etc.

Another type of method are method with do some calculations/operations on object/variable and return something with specified type. Key rule is that we must define what kind of type result we will would like to return.

Syntax: *[access\_modifier]* ***int*** *[name\_of\_method]*

Example:

*public* ***int*** *addTwoNumbers(int a, int b){*

***return*** *a+b;*

*}*

Now pay attention, we define a method with name „addTwoNumbers” which takes two parameters a and b, both with type of int. Method adds those two numbers and return a sum like an output.

WARNING!!! Method can also be declared, that she can return a type Person.  
The following code is also legal and acceptable. (It’s very usefull in design patterns).

*public Person createNewPerson(){*

***return*** *new Person();*

*}*

Nowi f you know this, try to rewrite your previous excercises to methods, which „do” something and return a result. Please may notice, that in Java, methods use camel case convention, for example for member of class declared private String name, access method are

getName, setName.

**Constructors in JAVA**

Let’s imagine that you are a house constructor, you need to have some plan how to build some houses, like where, what kind of dimension has example house etc. Very similar is in programming languages. Previously we create a new instance of class Person using a **new** operator. This means we called a constructor in Java.  
  
Rules about constructors in Java.

* If you don’t create any constructor in your class Java will create implicitly a default constructor like follows

*public Person(){*

*}*

* Constructor allways has the same name like java file and don’t have keyword „return” inside clamps
* Constructor from your class allways implicitly called a constructor of his parent (super class, this will be explained later).
* You can define constructor very similar like a method with parameter for example

*public Person(String name, String surname){*

*this.name=name;*

*this.surname=surname;*

*}*

**Excercises:**

**Ex. 1**

Try to write a class with 2-3 members, generate the access methods, and the default constructor.

**Ex. 2**

Based on Ex.1 try to write a constructor which initialize all 2-3 members in constructor.

**Ex. 3**

Try to create a class with 2-3 members, generate the access methods, and the default constructor.  
Then in some class with public static void main(String[] args) method try to create yours object, like on example

*Person p = new Person(„Adam”,”Smith”);*

**Conclusions?**

**Equals, hashCode, toString methods**

**EQUALS**

Imagine that you are comparing two integer values int x = 2 and int y=2 may use comparison like x==y (this works only for primitive types) and check the result as a boolean for example

**Example:**

*int x = 2;*

*int y = 2;*

*boolean result = x == y;*

*System.out.println(result);*

Displays a true, but those are primitives and we should use an OOP (Object Oriented Programming) approach, in this case thera are equivalent data types like int but in type of Object.   
Integer, Double, Long, Char, new String(); etc

This approch is used to have some better API from Object class, like equals.  
We have to provide the code which compare every member from class A with all members from class B. This is like comparison two cars. To discover that both are equal, you must compare for exmple: engine capacity, horse power, top speed, make, name, type etc. The same is in OOP world. In our example with class Person we have to compare name and surname with other name and surname.

The following code is legal and compares well two objects.

*public boolean equals(Object obj) {*

*if (this == obj) return true;*

*if (obj == null) return false;*

*if (getClass() != obj.getClass()) return false;*

*Person other = (Person) obj;*

*if (name == null) {*

*if (other.name != null) return false;*

*}*

*else if (!name.equals(other.name)) return false;*

*if (surname == null) {*

*if (other.surname != null) return false;*

*}*

*else if (!surname.equals(other.surname)) return false;*

*return true; }*

**WARNING!!!  
If you create (override) equals method, that will be works irrational and unproperly you may have false output.**

**EX 1.** Add the following snippet to your class (or you may generate it using IDE).

In main method create two instances of your class (for example p1, p2) and use equals to check that both classes (and their member values) are equals. Display the result in the console.

**hashCode**

HashCode – is some special „mixing” algorithm which display a int value representation of your object with whole his members. HashCode is used to some collections. The better hashCode alghorithm you provide the faster and better will be work whole program.  
  
Usually programmers override hashCode and equals methods to have proof that two object are „equal”.  
**Remember you shouldn’t return in hasCode method a simply value like 1.**

Example.

public int hashCode(){

return 1;

}

Above code is incorrect! It allways will be return 1, this is wrong implemented method! Do not the same in your code!

**toString**

**toString** –this is method interited from class Object (like 2 previous as well), to idea of String method is to display a „String” representation of the object, but If we do not create/override toString method in our class, then Java will return toString from class Object, which is usually a address in memory!

**Example:**

*Person p = new Person("Anna", "Nowak");*

*System.out.println(p);*

Will display output similar like *pl.info.Person@8470e61* – which means we didn’t provide a toString method in Person class.

Good way to generate / override toString method is generate it from your IDE.

The following toString() method is correct.

*public String toString() {*

*return "My name is=" + name + ", and my surname is=" + surname;*

*}*

When we execute the above code we will have an output like

My name is=Anna, and my surname is=Nowak

**REMEMBER**

If you corectly override hashCode and equals methods, both the same objects - should return that hashCode and equals are true.

Exercise

Try to write the code with toString, equals, hashCode methods and create two instances of some class and try to figure out that are equals.

**WARNING!!!**

If you will try to execute code like below, Java will don’t know which method to call (be declare to display object) then Java by default allways calls toString() method!!

*Person p = new Person("Anna", "Nowak");*

*System.out.println(p);*

**REMEMBER**

Creating your classes, you should allways override/generate **equals** and **hashCode** methods properly, and it’s good when you override toString method. This will protect you agains some bad responsible in the future.