**INTRODUCING TO OBJECTED ORIENTED PROGRAMMING**

**part II**

Imagine that you are the programmer, and some code is repeated again in another piece of program.  
The same logic, the same behaviour etc. What you will try to do to avoid this kind of situation?

There is no simple solution, but you can use power of polimorphism in Java and think about like follows.

I have some „core” application logic, which is in 95% the same but for 5% it might be different.   
Maybe is time see what **extends** keyword means in Java.

Example:

**public** **class** Parent {

public String **description(**String name, String type**)** **{**

**return** "I am " + this.getClass**()**.getName**()** + " my name is " + name + " and my type is " + type;

**}**

public void **sayHello()** **{**

System.***out***.println**(**"Hi all from Parent class"**)**;

**}**

}

**public** **class** Child **extends** Parent {

@Override

public String **description(**String name, String type**)** **{**

**return** "I am " + this.getClass**()**.getName**()** + " my name is " + name.toUpperCase**()** + " and my type is "+ type.toUpperCase**()**;

**}**

}

Keyword „extends” means for Java – that some clase is subclass (child of another class), because every class at the end has his own parent which is class Object (so everything in Java is an Object), but we do not declare explicitly **public class Myclass extends Object.** For Java it's obvious that every class extends parent class Object. But let's go back to our example. When we try to launch 2 instances of this classes we could do something like follows:

Parent parent = **new** Parent();

Child child = **new** Child();

So far so god, but imagine something like follows:

Parent p2 = **new** Child();

Now instance of p2 is pointing to the Parent type, not to child. For Java this is a sign, that p2 will be used API (methods) from Child class.

Now the description lie @Override – it means the same method signature exists in the super class (Parent class), but to discover which method is called we must sign this by using annotation   
@Override.

But what will happen when we try to execute the following code?  
Child child = **new** Child();

child.sayHello();

There is no method sayHello in Child class! What's going on? When we use an extension (extends)

it means, the every subclass (in this case Child) has an access to all public method from his super class (in this case Parent) , in that case we will execute the sayHello method from Parent class!!!  
This might be very usefull, think about it!

This what we done just a second before is called polimorphism. (Many meanings, many forms, states).

To add something more, if class extends another class, then extended class is type of his super class (Parent class).

This is the reason why sometime is good to make method private, and encapsulate the access to it.

To avoid the situation like mentioned before you may use keyword **final** in signature of method.

**public** **final** String description

Try to execute code now and see why the Java compiler is complaining?

Sometimes you may want to use method like in your Child class but with behaviour like method from parent class, what you can do?  
You can use keyword super.

**public** **class** Child **extends** Parent {

@Override

**public** String description(String name, String type) {

**return** **super**.description(name, type);

}

}

**Warning!!** In Java, you can extends only one class!!! The syntax like follow is forbidden

public class A extends B extends C …..

**Interfaces**

Interface – is a somekind of contract between class which implements an interface and interface it self.  
It's a kindly request to fullfill the contract but without say how.  
It's like an order from Parents like „You must buy today 1kg of carrot” - it's an order, but there is no said where you gonna to buy it or how, is it will be behind the corner, in shop, in supermarket or in hipermarket – it's not the point, you must just buy 1kg of carrot, that's it.

In normal attitude to the interfaces we push some common behaviour which will be „shared” with plenty of classes, but with diffrerences how single class which implement the interface will do it.  
  
Example:  
  
  
to declare interface you need coule lines of code  
  
  
**public** **interface** ICalculatorBehaviour {

**public** **int** addTwoInts(**int** x, **int** y);

}

and that's it! Now we must create a class which will be fullfil the contract of this interface (implements – keyword).  
  
Example:

**public** **class** CalculatorImpl **implements** ICalculator {

@Override

**public** **int** addTwoInts(**int** x, **int** y) {

// **TODO** Auto-generated method stub

**return** 0;

}

}

There is a very common pattern, to name the class which implements some interface with suffix like „Impl”.

In this example you see that IDE generates a stub, but there is no code in the method. Now it's your time to implement this and return a result.

You may declare this like follows:

@Override

**public** **int** addTwoInts(**int** x, **int** y) {

int result = x+y;

**return** result;

}

Class can implements many interfaces!! Class which implements interface is type of interface.  
  
The syntax if (CalculatorImpl instanceof Icalculator) will allways return true.  
  
  
Based on this knowledge try to model a calculator with all operations on the digits using interfaces.