INTERFACE

There are a number of situations in software engineering when it is important for disparate groups of programmers to agree to a "contract" that spells out how their software interacts. Each group should be able to write their code without any knowledge of how the other group's code is written. Generally speaking, *interfaces* are such contracts.
In the Java programming language, an *interface* is a reference type, similar to a class, that can contain *only* constants and method signatures. There are no method bodies. Interfaces cannot be instantiated—they can only be *implemented* by classes or *extended* by other interfaces.

Defining an interface is similar to creating a new class:

```
public interface MyInterface {
  // constant declarations, if any
  // method signatures
}
```

For example the interface ***IFigure*** (Listing 1) contains only two methods (without body) ***show, calculateField.*** Note that the method signatures have no braces and are terminated with a semicolon.

Listing 1.

```
public interface IFigure {
  // method signatures:
  void show();
  double calculateArea();
}
```

To use an interface, you write a class that implements the interface. When an instantiable class implements an interface, it provides a method body for each of the methods declared in the interface. For example class ***Circle*** implements the interface ***IFigure*** (Listing 2).

Listing 2.

```
public class Circle implements IFigure {
        protected double radius=1;
        protected String name="Circle";
        //add constructor for Circle
        //add method set/get
        //...
        //implement all methods from interface IFigure:
        void show() {
                System.out.println("Figure: "+name+", r="+radius);
        }
        double calculateArea(){
                return Math.PI*radius*radius;
        }
}
```

Ex. 1. Interface IFigure, classes Circle and Rectangle

a) Create the new **Lab4** project (***Java application***) and add new file to the project (***File/New file -> Interface***) entitled **IFigure**.
b) Complete the code by adding methods to interface **IFigure** (**Listing 1**)
c) Add other new file to the project (***File/New file->Java class***) entitled ***Circle.*** The class **Circle** should implement the interface **IFigure** - complete the class code (**Listing 2**). Remember that Netbeans help you add ***Getter and Setter*** methods (check what methods were created, their names and what are the parameters)
d) Go to the file with the application (***Lab4.java***) and in the ***main*** method, create two objects of ***Circle*** class and show the information about them and their areas.

e) Add the new java class **Rectangle**. The **Rectangle** class should also implement the ***IFigure*** interface and should contain the similar method as **Circle** class. Test the **Rectangle** class in main **Lab4.java** file.

Ex. 2. Interface IBase, Vector and Matrix classes

1. In the same **Lab4** project define other interface (in a separate file) ***IBase.java***
2. Interface **IBase** should contain constant integer field ***N=3*** and two methods: ***show*** and ***norm*** (Listing 3).
3. Define the **Vector** class (Listing 4), which implements the **IBase** interface and have:
   a. private ***name*** (equals "v" by default);
   b. private **one dimensional array** field ***v[N]*** - the coordinates of vector in N dimensional space (eg. for ***N=3***, vector ***x*** has three coordinates: ***x=[x0, x1, x2]***),
   c. two public ***constructors*** (one with no parameters, second with string and array parameter),
   d. the **setter/getter** methods,
   e. ***show*** method (the ***IBase*** method implementation) – show the name and coordinates of a vector (eg. ***a[1,2,3]),***
   f. ***norm*** method (the ***IBase*** method implementation) – calculate and return the norm of this vector,
   g. the other methods: ***sum, product,***
   (ex. ***a[a0,a1,a2], b[b0,b1,b2],***
   sum of two vectors: ***c=a+b***, ***c[a0+b0,a1+b1,a2+b2]*** (the result is vector),
   dot product of two vectors: ***d=a∘b=a0∘b0+a1∘b1+a2∘b2*** (the result is scalar),
4. Test the **Vector** class object in main **Lab4.java** application. Define two Vector object ***v1[1,2,3], v2[2,4,6]*** and show the sum and product of them. Calculate the norm of all vectors (Listing 5).
5. Define the **Matrix** class, which implements the **IBase** interface and have:
   a. one private **two dimensional array** field ***x[N][N]*** representing the matrix NxN,
   b. the public *constructors*,
   c. the ***setter/getter*** methods
   d. ***show*** method (the ***IBase*** method implementation) – show the name and coordinates of a vector(eg. ***a[1,2,3])***
   e. ***norm*** method (the ***IBase*** method implementation) – calculate and return the norm of vector
   f. the other methods: ***sum, product (of two matrix)***
6. Test the Matrix class objects in main ***Lab4.java*** application.

Listing 3.

```
public interface IBase {
    final static int N=3; //constant field in interface
    public void show();
    public double norm();
}
```

Listing 4.

```
public class Vector implements IBase{
    private String name="v";
    private double[] v;
    public Vector() { //constructor 1
         this.v=new double[IBase.N]; //allocates memory for array, the N is from IBase
    }

    public Vector(String name, double[] t) { //constructor 2
        this.name=name;
        this.v=t.clone(); //copy the array (from t to v)
    }

    @Override
    public void show() { //method of IBase interface - ovveriden
```

```
        System.out.print(name+"[");
        // ...
    }

    @Override
    public double norm() { //method of IBase interface - ovveriden
       double norm=0;
       //calculate the norm of vector
       return Math.sqrt(norm);
    }

    //define method sum
    //define method product
}
```

Listing 5.

```
public static void main(String[] args) {
        Vector a=new Vector();
        double t[]={0,3,4};   //the array definition
        Vector b=new Vector("b",t); //creates the vector b using the t array
        a.show();
        b.show();
        //test other methods
    }
```