Sofia University
Department of Mathematics and Informatics

Course : OO Programming Java
Date: November 6, 2018
Student Name:

Lab No. 6

Submit the IntelliJ projects developed to solve the problems listed below. Use comments and Modified-Hungarian notation.

Problem No. 1(inheritance)

1. Write a **UML class diagram** in IntelliJ for the following set of classes
   - Write a class **Point**. It **has** an array of two integer data members- the **x** and **y** coordinates. Define a **full set of constructors** (default, general purpose and a copy constructor), **set** and **get** methods for the class data members, a **set** method with a **Point argument** and a **get** method **returning a Point object**, as well as a **toString**() method.
   - Next, write a class **Line**. A **Line is a Point sPoint and has** a data member **Point ePoint denoting respectively the** starting and the ending **Point** of the line. Define a **full set of constructors** (default, general purpose and a copy constructor), **set** and **get** methods for the class data members, as well as a **toString**() method (reuse the **toString**() method defined for class Point). Write additionally a ***measure()*** method returning the length of the **Line**.
   - Finally, write a class **Rectangle**. **Rectangle is a Point, which defines the *upper left corner* and has a Point** that **defines** the ***lower right corner*** of the rectangle. Define a **full set of constructors** (default, general purpose and a copy constructor), **set** and **get** methods for the class data members, as well as a **toString**() method (reuse the **toString**() method defined for class **Point**). Write additionally a ***measure()*** method returning the perimeter of the **Rectangle**.
2. Generate the code for the above classes in a **Netbeans Java Console application project. Test** these three classes by instantiating a couple of **Point, Rectangle** and **Line** objects in and calling the respective ***measure ()*** for objects of classes **Rectangle** and **Line**.

Problem No. 2a

Write an **enum type** ***`TrafficLight`*** , whose ***constants* (RED, GREEN, YELLOW)** take one parameter – the duration of the light in milliseconds. ***Use***

long tm = System.currentTimeMillis();

to get the current time in milliseconds ***Write a program*** to test the **enum type** ***`TrafficLight`*** , so that it display in a loop the traffic lights text for the time duration set for each traffic light Exit the program test after 90 seconds pass for its start..

Problem No. 2b

Write a **JavaFX application** that displays three circles denoting a traffic light in the application window. Make use of the solution in Problem 2a to sequentially display the traffic lights where each one of the lights remains lit on for the number of milliseconds, specified in its enum definition.

Problem No. 3 (polymorphism)

Consider the following inheritance hierarchy **Shape-Point- Circle- Cylinder** all of which **implement** ***interface Comparable.***

A Point has two coordinates – x and y integer values

A Circle is a Point and has a radius (integer value)

A Cylinder is a Circle and has a height (integer value)

1. Write the respective **UML class diagram** in **NetBeans** for the above inheritance hierarchy where ***interface Comparable is*** defined as follows:

```
interface Comparable
{
        bool greater(Comparable obj);
        // this function
        // compares the this reference in the implementation class
        // with the obj reference, according to the class definition
        // of the meaning of the relation greater
        // Use operator "instanceof " to check for the obj reference
        type and
        // make an explicit type conversion
}
```

2. Write a class ***BubbleSort*** with a static method

public void sortArray(Comparable[] arr)

allowing you to sort in ascending order (*using the bubble sort algorithm)* an array of objects of any kind that implement ***interface Comparable.*** For instance, class ***BubbleSort*** should be able to sort arrays of ***Vehicles*** or ***Shapes*** that implement ***interface Comparable*** .

Define the function ***greater()*** for Point- Circle- Cylinder objects as follows:

a) a **Point** object P1 is greater than another **Point** object P2, if P1.mX > P2.mX and P1.mY > P2.mY, when P1.mX = P2.mX. (for instance point (1,2) is greater than point (1,1) **) Two points are equal when their coordinates are equal**

b) a **Circle** object C1 is greater than another **Circle** object C2, if the center point of C1 (which is a point object) is greater than the center point of C2 (which is also a point object) and C1.mRadius > C2.mRadius, when the center point of C1 is equal to the center point of C2. **Two circles are equal when their center points and radiuses are equal**

c) a **Cylinder** object C1 is greater than another **Cylinder** object C2, if the circle of C1 (which is a Circle object) is greater than the circle of C2 (which is also a Circle object) and C1.mHeight > C2.mHeight, when the circle of C1 is equal to the circle of C2 **Two cyllinders are equal when their circles and heights are equal**

3. Write a class ***BubbleSortTest*** to test class ***BubbleSort*** where class ***BubbleSortTest is a*** **Console** application. The application class must have a class member- a reference to an array ***arrComparable*** of type ***Comparable*** with **3** elements. **Employ** ***arrComparable*** to sort in sequence with ***sortArray(Comparable[] arr)***:
    a) three **Point**s (the coordinates should be random generated in the interval [10,50]) by assigning the **Point**s to ***arrComparable***.
    b) three **Circle**s (the centers of the circles should be the three points, the radiuses of the circles should be random generated in the interval [10,30]) by assigning the **Circle**s to ***arrComparable***.
    c) three **Cylinder**s (the circles of the circles should be the above defined Circle objects, the heights of the cylinders should be random generated in the interval [10,60]) by assigning the **Cylinder**s to ***arrComparable***.

    Display in a Dialog box (JOptionPane) the objects for each case (a- c) in the respective array sorted by the ***sortArray(Comparable[] arr)*** method of class ***BubbleSort*** in ascending order. You **must** use **overriding of function** *toString()* and late binding (**polymorphism**) to display each one object in the message box to get the whole number of marks allocated for this part of the problem.

Problem No. 4

Create a ***Singleton*** class in ***Java***

A ***Singleton*** is a ***class*** that returns the same and the same object everytime it is used. It should have ***private*** data members, which are initialized by a ***private*** constructor and a ***private static*** reference to a ***Singleton*** object, instantiated by the ***private Singleton*** constructor. A reference to the ***private static Singleton*** is provided through a ***public static Singleton getInstance***() method.

Create the ***Singleton*** class and compare two references of that class to make sure they have the same memory references in a **Console** application.

Problem No. 5

Create a ***Months enumeration*** in Java

It should be possible to create **only 12** instances of the objects ***JAN, FEB,…, DEC***. There should be a ***toString***() method in that class allowing each month to display the full name "***January", "February***, …, "***December***" by referring to the respective object name or by an index in an **array** ***Month***

Months.JAN→ displays "***January***"

Months.Month[[0]→ displays "***January***"

Create the ***Months*** class and test the above class properties in a ***Console*** application.

Problem No. 6

Create three interfaces, each with two methods. **Inherit a new fourth interface** from the three, adding a new method void m(). **Create a *class* A by implementing the fourth interface** and also inheriting from a concrete class B with implementation of method m() declared in the fourth interface(each of the implementations should write a string on the ***Console*** indicating the method name). Now write four methods, each of which takes one of the four interfaces as an argument(each of the methods should make sure the argument is derive from the respective interface and run the respective interface methods with the so passed argument reference). In ***main***(), create an object of your class and pass it to each of the methods. Create a UML class diagram and write the definitions of the interfaces and the classes involved in this UML class diagram

Problem No. 7

Consider the following class definitions and describe what would be output by the code segment.

```
public class A {
    public A() { System.out.println("A"); }
}
public class B extends A {
    public B() { System.out.println("B"); }
}
public class C extends B {
    public C() { System.out.println("C"); }
}
          // Determine the output.
A a = new A();
B b = new B();
C c = new C();
```

Optional problems:

a) Prove that the fields in an interface are implicitly **static** and **final**.

b) **Create an interface** containing three methods, in its own package. Implement the interface in a different package.

c) Prove that all the methods in an interface are automatically **public**.

d) **Create three interfaces**, each with two methods. **Inherit a new interface** from the three, adding a new method. **Create a class by implementing the new interface** and also inheriting from a concrete class. Now write four methods, each of which takes one of the four interfaces as an argument. In main(), create an object of your class and pass it to each of the methods