

Lab 6 Requirements

Create a new Eclipse workspace named "**Lab6_1234567890**" on the desktop of your computer (replace **1234567890** with your student ID number). For each question below, create a new project in that workspace. Call each project by its question number: "**Question1**", "**Question2**", etc. If you do not remember how to create a workspace or projects, read the "*Introduction to Eclipse*" document which is on iSpace. Answer all the questions below. At the end of the lab, create a ZIP archive of the whole workspace folder. The resulting ZIP file must be called "**Lab6_1234567890.zip**" (replace **1234567890** with your student ID number). Upload the ZIP file on iSpace.

Question 1

Create a **Shape** class with the following UML specification:

```
+-----+
|                Shape                |
+-----+
| - x: double                        |
| - y: double                        |
+-----+
| + Shape(double x, double y)        |
| + getX(): double                   |
| + getY(): double                   |
| + area(): double                   |
| + testShape(): void                |
+-----+
```

where the **x** and **y** instance variables store the position of the central point of the shape. The **area** method computes as result the area of the shape: unfortunately an unknown shape has an unknown area (we only know how to compute the area of specific shapes, like triangles, for example) so the **area** method just prints a message "**An unknown shape has an unknown area!**" and returns a meaningless result such as **-1.0** (because the **area** method must return something which is of type **double**). The **testShape** method is **static**.

Add the following code to your program to test the **Shape** class:

```
public class Start {
    public static void main(String[] args) {
        Shape.testShape();
    }
}
```

Question 2

Add a **Circle** class that derives from the **Shape** class and has the following UML specification:

```
+-----+
|                Circle                |
+-----+
| - radius: double                    |
+-----+
| + Circle(double x, double y, double radius) |
| + area(): double                    |
| + testCircle(): void                |
+-----+
```

```
+-----+
```

Use `Math.PI` to compute the area of a circle.

Do not forget to change the `main` method of the `Start` class to run the unit tests of the new `Circle` class.

Question 3

Add a `Dot` class that derives from the `Shape` class and has the following UML specification:

```
+-----+
|               Dot               |
+-----+
+-----+
| + Dot(double x, double y)      |
| + area(): double               |
| + testDot(): void              |
+-----+
```

Do not forget to change the `main` method of the `Start` class to run the unit tests of the new `Dot` class.

Question 4

Add two new classes `Rectangle` and `Square`. `Rectangle` derives from `Shape` and `Square` derives from `Rectangle`. `Rectangle` has the following UML specification:

```
+-----+
|               Rectangle          |
+-----+
| - width: double                 |
| - length: double               |
+-----+
| + Rectangle(double x, double y, double width, double length) |
| + area(): double               |
| + testRectangle(): void         |
+-----+
```

The constructor for `Square` takes three arguments: the `x` and `y` positions of the center of the square, and the `size` of the square.

Does the `Square` class need its own `area` method?

Do not forget to change the `main` method of the `Start` class to run the unit tests of the new `Rectangle` and `Square` classes.

Question 5

We now want to be able to manipulate many shapes together in our software, not just one shape at a time. So add a `ManyShapes` class to your program with the following UML diagram:

```
+-----+
|           ManyShapes           |
+-----+
| - allShapes: ArrayList          |
+-----+
```

```
+-----+
| + ManyShapes ()          |
| + addShape(Shape s): void |
| + listAllShapes(): void   |
| + testManyShapes (): void |
+-----+
```

The **allShapes** instance variable is of type **ArrayList** which is a class provided to you by Java that you need to import into your program using: **import java.util.ArrayList;**

You can then define the **allShapes** instance variable like this: **private ArrayList allShapes;**

In the **ManyShapes** constructor you need to create a new **ArrayList** object and store it in the instance variable **allShapes**, like this: **this.allShapes = new ArrayList();**

(if you forget to do this then the instance variable **allShapes** will point at nothing and you will get an error when you run your program and you try to call a method of the nonexistent arraylist object).

An **ArrayList** object works both like an array and like a list, in which you can store as many other objects of type **Object** as you want:

- you can add a new object **o** to the arraylist by using the **add(o)** method of the arraylist;
- you can get the number of elements in the arraylist by using the **size()** method of the arraylist;
- you can access a specific element of the arraylist at index **i** by using the **get(i)** method of the arraylist (element indexes start at zero in an arraylist).

The **addShape** method takes a shape as argument and adds it to the arraylist.

The **listAllShapes** method prints on the screen the area of each shape in the arraylist, one by one, using a loop. For example, if the arraylist currently contains a **Square** object of size 5 and a **Dot** object then the **listAllShapes** method should print:

```
Shape has area 25.0
Shape has area 0.0
```

Here is the code of the **testManyShapes** method:

```
public static void testManyShapes() {
    ManyShapes m = new ManyShapes();
    m.addShape(new Circle(1.2, 3.4, 4.0)); // Upcast from Circle to Shape.
    m.addShape(new Dot(1.2, 3.4));         // Upcast from Dot to Shape.
    m.addShape(new Rectangle(1.2, 3.4, 4.0, 5.0)); // Upcast from Rectangle to Shape.
    m.addShape(new Shape(1.2, 3.4));
    m.addShape(new Square(1.2, 3.4, 5.0)); // Upcast from Square to Shape.
    m.listAllShapes();
}
```

Do not forget to change the **main** method of the **Start** class to run the unit tests of the new **ManyShapes** class.

Question 6

The **listAllShapes** method tells us the area of every shape in the arraylist but it does not tell us the type of the shapes in the arraylist. Modify the **listAllShapes** method to tell us both the area and the type for each shape in the arraylist. For example, if the arraylist currently contains a **Square** object of size 5 and a **Dot** object then the **listAllShapes** method should print:

```
Square has area 25.0
Dot has area 0.0
```

Use the **instanceof** operator in the **listAllShapes** method to determine the type of each shape.

Question 7

Using **instanceof** in the **listAllShapes** method works fine, but there is a nicer, more object-oriented way to do the same thing: just have every shape object tell about itself when it is asked! To do this, add a new method **toString** to the **Shape** class that overrides the **toString** method which is inherited by the **Shape** class from the **Object** class. This method should then return the string **"YYY area is XXX"**, where **XXX** is the result of the **area** method from the same class **Shape**, and **YYY** here can be **"Shape"**, **"Dot"**, **"Circle"**, **"Rectangle"** and **"Square"**, etc... It should be changed accordingly. Then override the **toString** method in every subclass of **Shape** to return the right string for the subclass in a similar way.

After you have added the **toString** method to the **Shape** class and all its subclasses, delete all the **instanceof** tests in the **listAllShapes** method of the **ManyShapes** class and use **System.out.println** to directly print every element of the arraylist (**System.out.println** will then automatically call the **toString** method of each object that you are printing; no downcast is needed then because Java's dynamic dispatch will automatically call the right **toString** method coming from the right class for each object stored in the arraylist that you are printing).

Note: the text in red color above is updated on 10:40, Mar 22, 2023.