# 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:

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:

+-----+

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:

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:

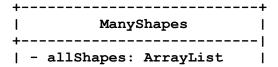
The constructor for **Square** takes three arguments: the  $\mathbf{x}$  and  $\mathbf{y}$  positions of the center of the square, and the  $\mathtt{size}$  of the square.

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:



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

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).

In the ManyShapes constructor you need to create a new ArrayList object and store it in the instance variable allShapes (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).

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:

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 "Shape has area XXX", where XXX is the result of the area method from the same class Shape. 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).