Lab 8 Requirements

Create a new Eclipse workspace named "**`Lab8_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 "**`Lab8_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                    |
| + move(double dx, double dy): void  |
| + area(): double                    |
| + resize(double newSize): void      |
| + testShape(): void                 |
+-------------------------------------+
```

where the **`x`** and **`y`** instance variables store the position of the central point of the shape. The **`move`** method moves the central point of the shape by the amounts **`dx`** and **`dy`**. The **`resize`** method is used to change the size of a shape. 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                             |
| + resize(double newRadius): void             |
| + testCircle(): void                           |
+------------------------------------------------+
```

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

Resizing a circle changes its radius to be **newRadius**.

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                  |
| + resize(double newSize): void    |
| + testDot(): void                 |
+-----------------------------------+
```

It is not possible to resize a dot (a dot has no size). Therefore the **resize** method of the **Dot** class must throw an exception. The type of the exception object must be **Exception** and the message must be "**Cannot resize a dot!**"

Make sure you properly test the **resize** method using **try-catch**.

What is the problem with the **resize** method of the **Shape** class? How do you solve this problem?

Do you need to change the **resize** method of the **Circle** class then?

Question 4

Add a **CannotResizeException** class that derives from the **Exception** class. Modify the **resize** and **testDot** methods of the **Dot** class to use this new class instead of using the class **Exception**.

Modify the **Shape** class accordingly.

Question 5

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                                             |
| + resize(double newSize): void                               |
| + testRectangle(): void                                        |
+-----------------------------------------------------------------+
```

Resizing a rectangle changes its **`width`** and **`length`** to both be **`newSize`**.

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`** and **`resize`** methods or not?

Question 6

The **`resize`** method of the **`Rectangle`** class is annoying: it only allows you to resize a rectangle into a rectangle that has equal width and length. We want to be able to resize a rectangle into a rectangle that has different width and length. To do this, add to the **`Rectangle`** class a second new **`resize`** method that takes two arguments: a new width and a new length.

What is then the problem for the **`Square`** class?

Question 7

To solve the previous problem, add to the **`Square`** class a new **`resize`** method that overrides the **`resize`** method that you just added to the **`Rectangle`** class.

In the new **`resize`** method of the **`Square`** class, if the new **`width`** and the new **`length`** are equal then the **`resize`** method should resize the square; if the new **`width`** and the new **`length`** are different then the **`resize`** method should throw a **`CannotResizeException`** exception with the message "**`Cannot resize a square into a rectangle!`**"

Make sure you properly test the **`resize`** method.

What happens with the **`resize`** method of the **`Rectangle`** class?

Question 8

Add a **`BadRadiusException`** class that derives from the **`Exception`** class. Modify both the constructor and the **`resize`** method of the **`Circle`** class to use this new class when the given **`radius`** is negative. The message of the exception must be "**`Radius must be positive!`**"

Make sure you properly test both the modified constructor and the **`resize`** method.

What is then the problem for the **`Shape`** class?