



# Lab 3 Report

10/10/2021—

Bilawal Nisar

215157811

Section A

Kazi Mridul

[kmridul@yorku.ca](mailto:kmridul@yorku.ca)

## Introduction

## Overview

The Purpose of this lab is to instantiate six shapes that can be squares, rectangles, or circles and the application should be able to use one sorting technique to sort the six shapes based on their surfaces.

## Goals

1. The goals are personal for every student. In my case my first goal is to learn the concepts that I will use to carry out the software project.
2. The second goal is to learn how to create UML class diagrams while making sure that I am doing it so by using two design patterns

## Specifications

-- The concepts that I will use to carry out in this project will be pretty simple.

**Factory Method**

**Singleton Pattern**

**Inheritance**

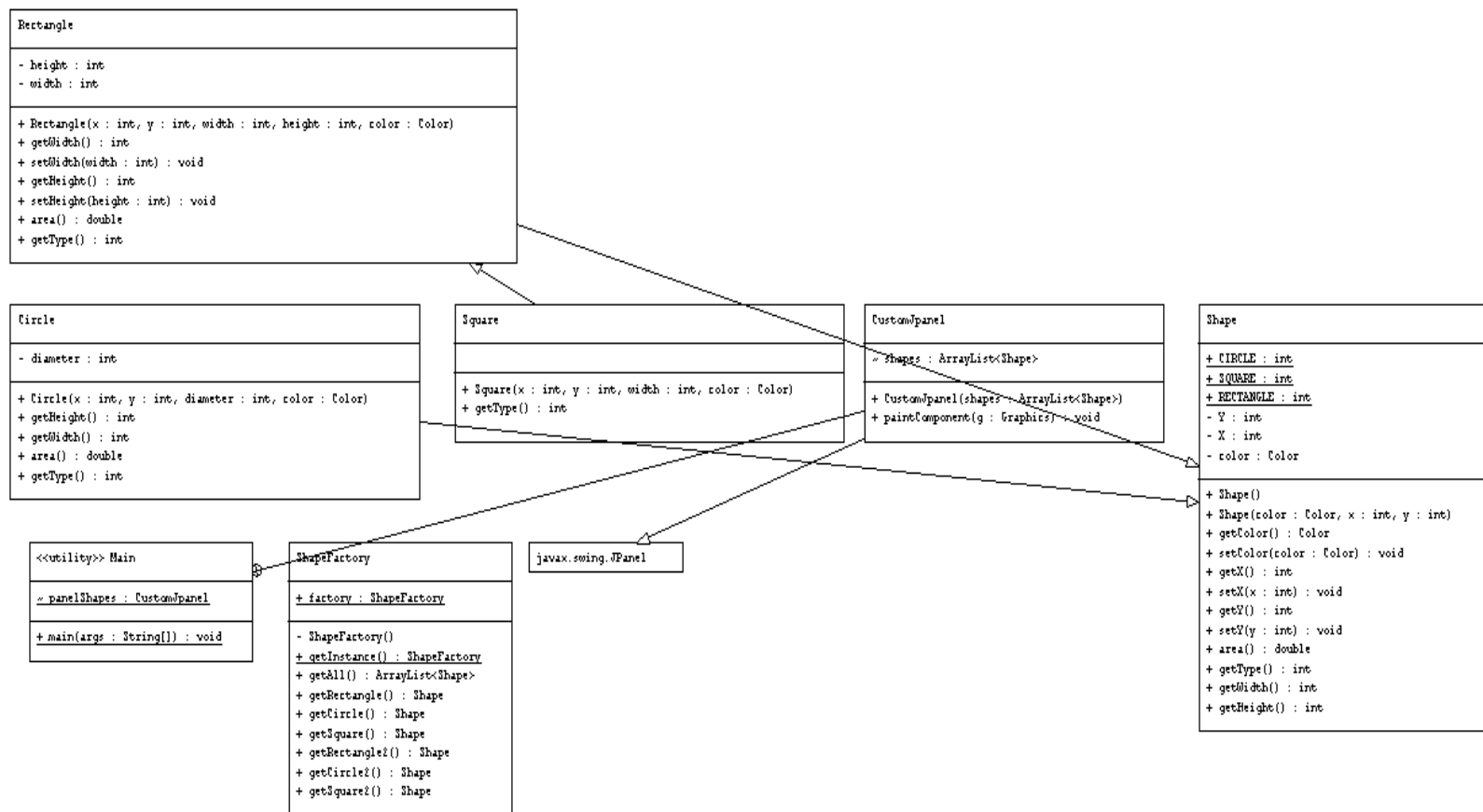
## Structure of the Report

The report will be structured in a way that Part 1 would be Introduction, Part 2 will be The Design of the Solution and part 3 will consist of implementation of the solution and lastly Conclusion.

## Part 2 Design of the Solution

- I. In the design of the solution I have not answered the questions separately. Instead I have used the diagrams and my knowledge to include all of them together to give a sightfull understanding of what really is the design of the solution.

## UML Diagram (Design 1)



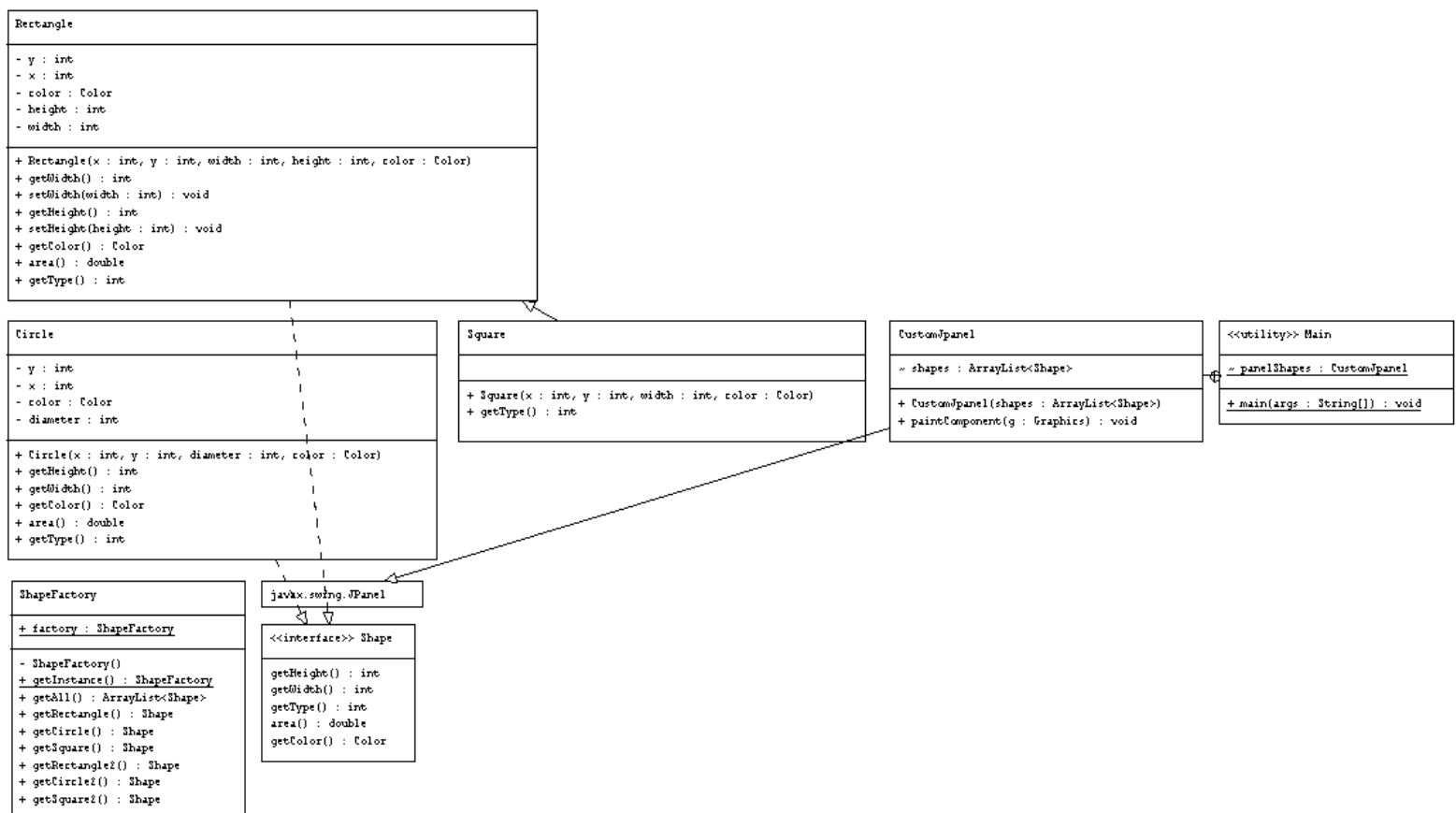
Using OO design Principles and explaining them.

We have applied both the factory design pattern and Singleton pattern in our project. As we can see the Shape class is the superclass of all other shapes. The shape class defines the basic attributes of a shape. As every shape should have an X and Y coordinate, width and color, it creates necessary variables to store those info for every other shape. It also provides functions like **area()**, **getHeight()**, **getWidth()** so that the other real shapes can override and implement those methods. Also it defines the **getType()** method so that other shapes may implement that method and provide us a way to determine their type in runtime. Each class performs a separate task while calculating their area. The classes perfectly realize the idea of inheritance to make the software building process smoother. By inheriting the Shape class we didn't have to declare X, Y, color attribute again in our Rectangle, Circle and Square class. Also we made the square class to inherit the Rectangle class. That really made writing code for the Square class easy.

We extended the JPanel class to create a custom JPanel that prints the Shape when its **paint()** method is called.

We created the ShapeFactory class as a Singleton. Its primary objective is to supply a list of Shape objects so we don't actually need to create a lot of new objects of this class. Instead, we can just use only one instance of this class throughout our program.

## UML Diagram (Design 2 Alternative)




## Proposing an alternative

Instead of making a shape as a super class we can declare it as an interface and make all other shapes implement this interface. This works the same as before. But in this case we had to write X and Y and color attributes in all the three classes. That's just repetition of code. This was avoided in our previous design. And that design also provided the same facilities. So I think the previous model was a better option as we had to write and manage less code and still had all the same facilities.

## Part III: Implementation of the solution

The shape objects are stored in an ArrayList to facilitate shorting. While sorting we called the area() method on each of the shape objects. All the three shape class have different version of area() method

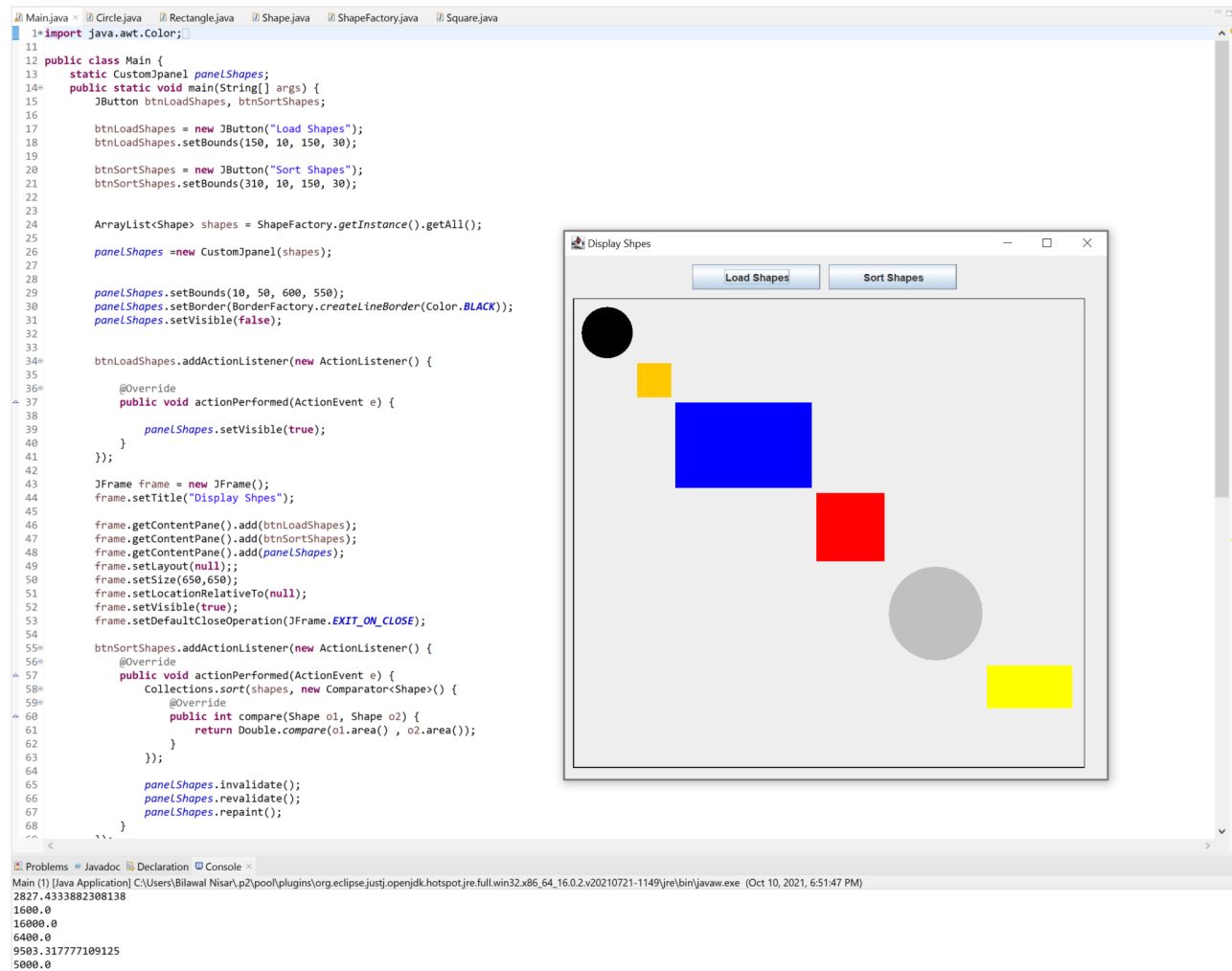


inside them. Which version of the area method to execute totally depended on the type of original object. This is the polymorphism concept of OO design. We could store different types of Object all to their superclass variable, but it is decided on runtime which version of a method is to call. Similar to the area() method we have used getType() to determine the type of the shape and print them accordingly. The getType() was also called on the shape variable but returned values according to the objects. Same as before As i have mentioned above I have implemented and compiled all of the classes of my diagram in IntelliJ using the superclass variable.

The tools That i have used during the implementation: Mostly Eclipse and IntelliJ.

## **Snapshot of the execution of the code ( Design 1 )**

## Snapshot of Load Shapes

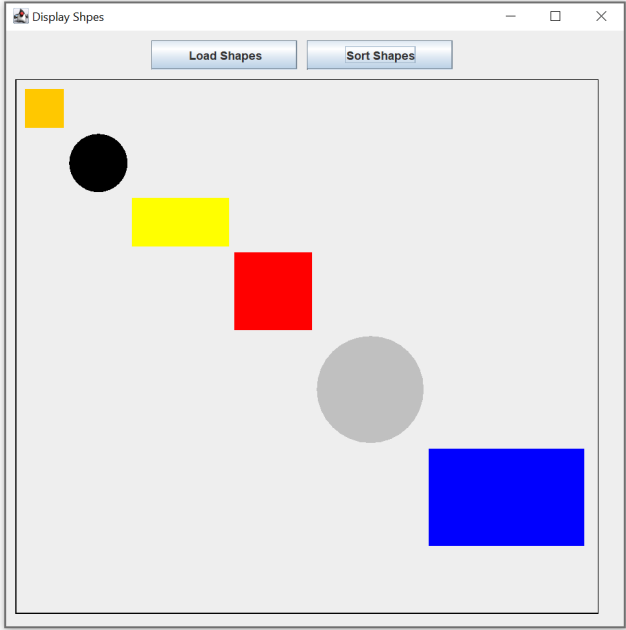


## Snapshot of Sort Shapes

```

Main.java x Circle.java Rectangle.java Shape.java ShapeFactory.java Square.java
1 *import java.awt.Color;
11
12 public class Main {
13     static CustomJpanel panelShapes;
14     public static void main(String[] args) {
15         JButton btnLoadShapes, btnSortShapes;
16
17         btnLoadShapes = new JButton("Load Shapes");
18         btnLoadShapes.setBounds(150, 10, 150, 30);
19
20         btnSortShapes = new JButton("Sort Shapes");
21         btnSortShapes.setBounds(310, 10, 150, 30);
22
23
24         ArrayList<Shape> shapes = ShapeFactory.getInstance().getAll();
25
26         panelShapes = new CustomJpanel(shapes);
27
28
29         panelShapes.setBounds(10, 50, 600, 550);
30         panelShapes.setBorder(BorderFactory.createLineBorder(Color.BLACK));
31         panelShapes.setVisible(false);
32
33
34         btnLoadShapes.addActionListener(new ActionListener() {
35
36             @Override
37             public void actionPerformed(ActionEvent e) {
38                 panelShapes.setVisible(true);
39             }
40         });
41
42
43         JFrame frame = new JFrame();
44         frame.setTitle("Display Shpes");
45
46         frame.getContentPane().add(btnLoadShapes);
47         frame.getContentPane().add(btnSortShapes);
48         frame.getContentPane().add(panelShapes);
49         frame.setLayout(null);
50         frame.setSize(650,650);
51         frame.setLocationRelativeTo(null);
52         frame.setVisible(true);
53         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
54
55         btnSortShapes.addActionListener(new ActionListener() {
56             @Override
57             public void actionPerformed(ActionEvent e) {
58                 Collections.sort(shapes, new Comparator<Shape>() {
59                     @Override
60                     public int compare(Shape o1, Shape o2) {
61                         return Double.compare(o1.area(), o2.area());
62                     }
63                 });
64
65                 panelShapes.invalidate();
66                 panelShapes.revalidate();
67                 panelShapes.repaint();
68             }
69         });
70
71     }
72 }

```



```

Problems x Javadoc Declaration Console x
Main (1) [Java Application] C:\Users\Bilawal Nisar\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.16.0.2.v20210721-1149\jre\bin\javaw.exe (Oct 10, 2021, 6:51:47 PM)
2827.4333882308138
16000.0
16000.0
64000.0
9503.317777109125
5000.0
16000.0
2827.4333882308138
5000.0
64000.0
9503.317777109125
16000.0

```

## Conclusion

- Everything went smoothly as I expected, however when I was sorting the shapes it would overlap it with the same X and Y coordinates and a lot of other students had the same problem. Just had to make sure that the coordinates didn't overlap and that solved the problem.
- I have learned from this software project that first of all It should be worth more than 5% and second of all that even though things seem easy at first it was a simple jpanel frame where you had to sort and load but because of the techniques and requirements it can often tend to be a bit of hassle.



