EECS3311 A F21 LAB REPORT 1

First Name: Hasnain Saifee

Student ID: 216879694

Course: Software Design

TA: Kazi Mridul

Part 1: Introduction:

1) The software project is about an application that consists of two buttons: The one (load button) loads six different shapes onto the interface and the other (sort button) displays the shapes based on their sorting order. The goal is to output shapes based on their sorting order.

2) Some of the challenges include:

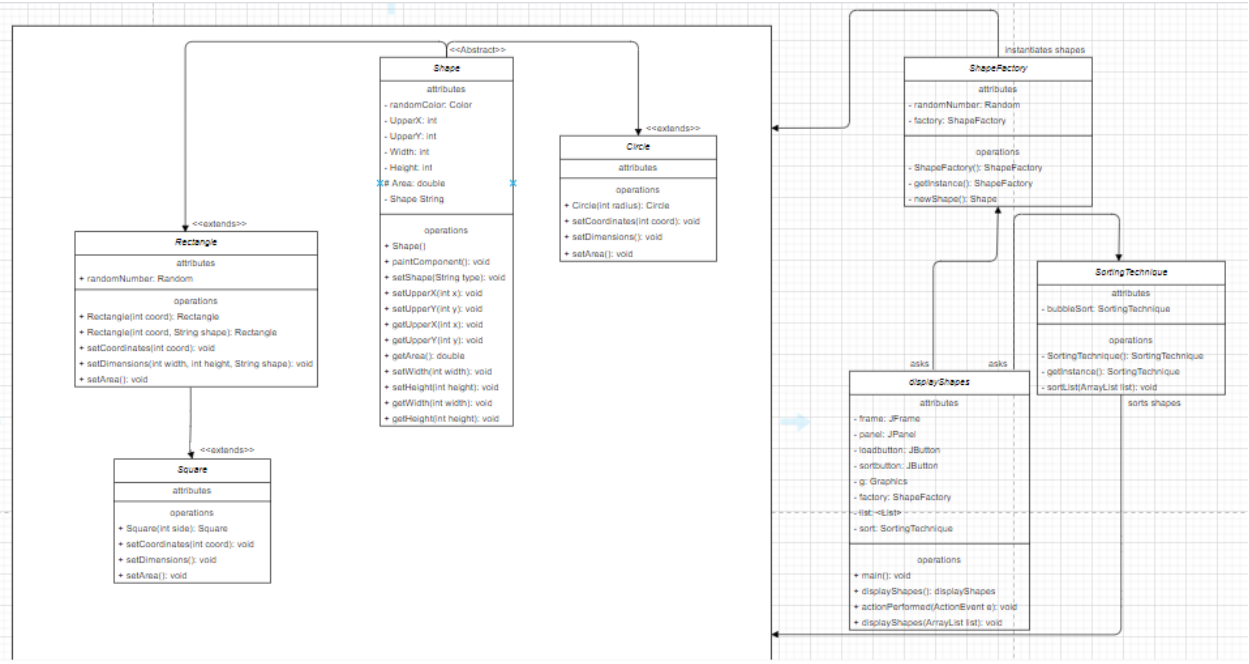
    - Not able to fit all the objects into one window
    - Overlapping the objects
    - Not sorting the objects properly according to size
    - Buttons not showing up on the frame
    - Action listener on the button was not implemented properly

3) - In this project, all the OOD principles should be used. Abstraction will be used to hide unnecessary details from the user. Methods such as *setArea(), setCoordinates()* are easily identifiable and readable by users. Each class will be functioning differently, and the methods and states will be encapsulated in those classes. As the lab states that, a Square is a Rectangle and so we understand that the *Square class* will be inherit all the methods and states from the *Rectangle class*. Knowing that a Square is a Rectangle some of the methods will be the same or have different parameter sizes to adapt to different areas, so polymorphism comes into place.

    - Moreover, the design patterns will help us construct the foundation of the program that is to be created. It will give us the blueprint, and help us identify loopholes that needs to be filled in.

4) The report will be structured as follows: Introduction, OOD principles explained, design patterns, UML Class diagrams, Implementation of the code & conclusion.

Part 2: Design:



1) Inheritance

- The Inheritance principle is used a lot throughout this project. For instance, the *Square Class* extends the *Rectangle Class* on the basis that a Square is a Rectangle. In addition, a Rectangle, Square, and a Circle is a Shape and so these shapes extends the *Shape Class*.

2) Abstraction

- Abstraction implies the underlining hidden implementation of a class. For instance, we create new shapes such as **new** *Rectangle()*, however it is not necessary to know how it was made. In addition, methods such as *setDimensions()*, *setArea()*, and *setCoordinates()* are all abstracted from the programmer as it is not necessary to understand those methods in details.

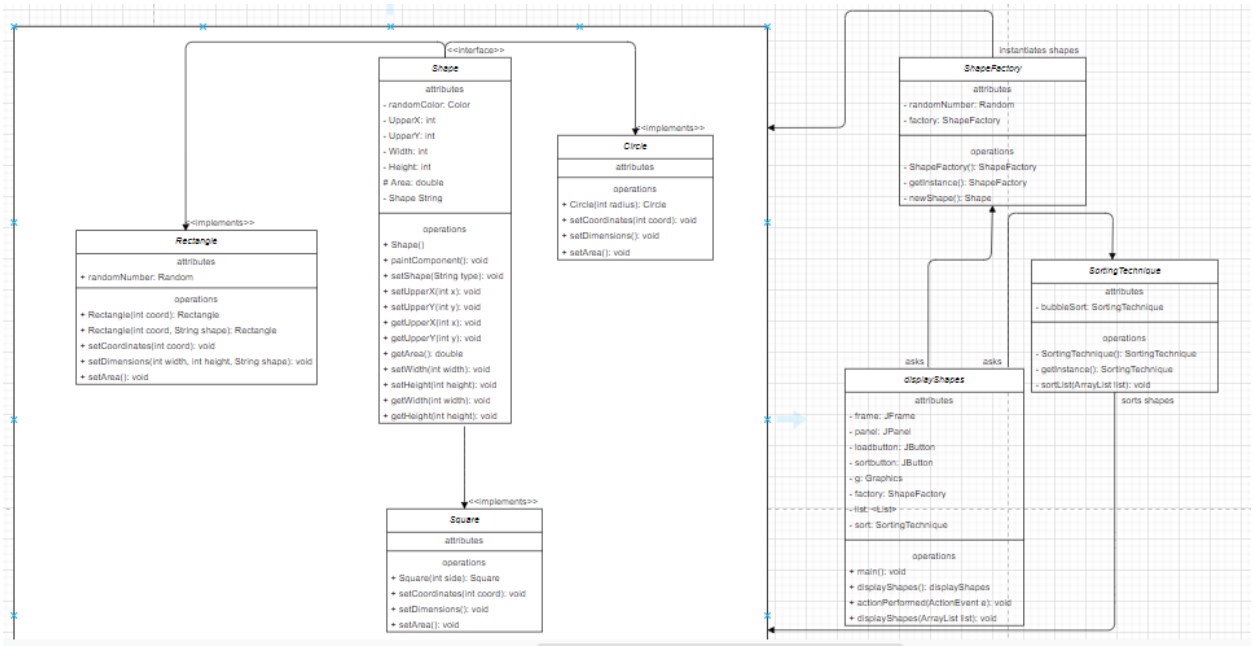
3) Encapsulation

- Encapsulation implies that all states and methods are to be kept inside a class. This is the case for all the classes. We encapsulate methods and states to not only keep the logic of the code organized but to understand which states and methods belong to which class when used elsewhere.

4) Polymorphism

- Polymorphism implies the ability of a method to take different shapes. For instance, the Rectangle class has more than one constructor, each having different number of parameters to account for different sizes. In addition, the *setArea()* method is implemented differently in each class to adapt to different shapes.

Alternate UML:



The only difference here compared to the one above is that I made the Shape class into an interface, that implements the 3 other classes. In my opinion, the alternative wouldn't yield a bad design except that the Shape class will incur more methods compared to the first one. The first diagram is flexible, such as changing area formulas in different classes, and hence I would choose the first diagram over the alternate one.

Part 3: Implementation:

1) The algorithm that I used is the **Bubble Sort** algorithm. The bubble sorting algorithm compares to surface areas. If the first is greater than the second, then swap them. Do the same for each adjacent pairs of surface areas. By the end of the loop, the first shape should have the smallest surface area and the last shape should have the largest surface

area. The above steps are repeated for all surface areas, except the last one. The steps are repeated until there are no more pairs to compare.

2) I used the first UML diagram to implement the code. The implementation goes as follows:

   The *Shape Class* is an abstract class. We know there are different shapes such as the ones implemented in the project (Rectangle, Circle, Square). So the three classes will inherit states and methods of the Shape Class (parent class). Moreover, I created two Singleton classes: *ShapeFactory()* and *SortingTechnique()*. The *ShapeFactory()* class will be asked by the main class to load the shapes. This is done when the load button is clicked (using Java Swing library). These shapes will be loaded in a horizontal manner. Then, the *SortingTechnique()* class will be asked by the main class to sort the shapes. This is done when the sort button is clicked. These shapes will be sorted in a horizontal manner. Lastly, the main class will have the interface implemented, and a few methods. If the load button is clicked, then the shapes will be loaded randomly, else the sort button is clicked and it will sort the shapes after the shapes are loaded. There is a possibility where the user might click the sort button accidently before the load button. Then in that case, the program does not crash, it just does not display anything.

3) I used Eclipse 2021-09, version is JDK-13.0.1

4) **Load Shapes button is clicked** **Sort Shapes button is clicked**

Part 4: Conclusion:

1) I enjoyed creating the interface. I wasn't aware what Java Swing library has to offer and now I have some experience using it. Moreover, I learnt how to implement a sorting algorithm using a Comparable interface.

2) I did not thoroughly understand the design patterns, and how to combine more than one design patterns into the UML class diagram.

3) As said above, I learnt how to implement a sorting algorithm using a Comparable interface. In addition, I also learnt how to use the UML diagram to create my first ever software project.

4) Three recommendations are:
   - Clear all doubts, even if they are as small as learning how to inherit a class.
   - Communicate with your peers and get to know their thought process and how they proceed creating software.
   - Strengthen your basics in Java by watching tutorials or reading articles.