EECS-3311 A Week 3 Lab Report

Ayush Sharma 217581075

Part 1: Introduction

The software project is about providing a graphical user interface which can draw 6 shapes on the screen and is able to sort the drawn shapes by surface area. The goals for this software project are to utilize various software design patterns and getting practical experience with the implementation of these design patterns.

The main challenges of the software project are to create an API which will allow the various shape classes to communicate with the display class and using a design pattern to implement the API. Thus, we need to define appropriate relationships between the various classes such as association, composition, aggregation, inheritance etc.

The main concepts which are used in this software project are:

OO Analysis: Analyzing the software project description and choosing various classes based on how to fulfill the project requirements.

Encapsulation: Hiding the internal variables of the shape classes and providing getters and setters to change the internal state of the class.

Inheritance: The Rectangle, Circle and Square classes all inherit the abstract shape class.

Abstraction: Providing an interface which only allows access to the class's attributes via setters and getters.

Factory Pattern: The presence of the ShapeFactory class which creates the instances of the all the shape classes.

UML Class Diagrams*:* Using UML class diagrams to describe the various classes and their relationships to each other in a graphical format.

My report will be structured by first designing the UML class diagrams for the implementation. The UML diagram will describe the various classes and the relationships among them. As per the software project requirements, there will be 1 initial UML class diagram and then after reviewing the first UML diagram we shall design another UML diagram and compare each other. Then the report will show a working implementation of the software project whilst highlighting the important elements and describing the software tools used in the implementation. At the end, the report will conclude with a review of how the software project went and will provide learning outcomes of the project and provide recommendations.

Part 2: Design

UML class diagram 1

The UML diagram above shows the various classes which comprise the software projects.

Now we first start with the abstract *Shape* class. This class embodies the abstract concept of a shape. There are a few essential fields which are associated to each shape such as the location, which is represented with *top* and *left*, specifying the top-left point of a shape. The *Shape* class also has some methods which each shape should have such as *drawShape, copy, and getArea.* As per specifications, the *Shape* class also implements the *Comparable<T>* interace. Also, applying the OO principle of Encapsulation, each of the fields are private and provide getters and setters to retrieve and manipulate the fields respectively.

There are three concrete classes which are subclasses of the the *Shape* class, namely *MyRectangle, MyCircle* and *MySquare*. Each of the classes have extra fields specific to them, for instance *MyCircle* class has *radius*, whereas *MyRectangle* has *width and height*. All these concrete classes again apply Encapsulation by providing appropriate getters and setters.

The *ShapeFactory* class embodies the Factory design pattern and thus provides an interface for other classes, such as the *MyPanel* class, to instantiate sub-classes of the *Shape* class. It also provides a great utility method *createRandomShape* which instantiates a random *Shape* class with random parameters. This method is used extensively by the *MyPanel* class.

The *InsertionSort* class embodies the Singleton design pattern, since multiple instances of this class are unresourceful. The main method for this class is the *sort* method which takes in a *ArrayList* of *Shape* and sorts the arraylist by the surface area of each shape.

MyPanel class extends the JPanel class and manages the drawing and creation of the 6 shapes required for the software project.

The two design patterns which are use in the above UML diagram are the Factory design pattern and the Singelton design pattern shown by the *ShapeFactory* and the *InsertionSort* classes respectively.

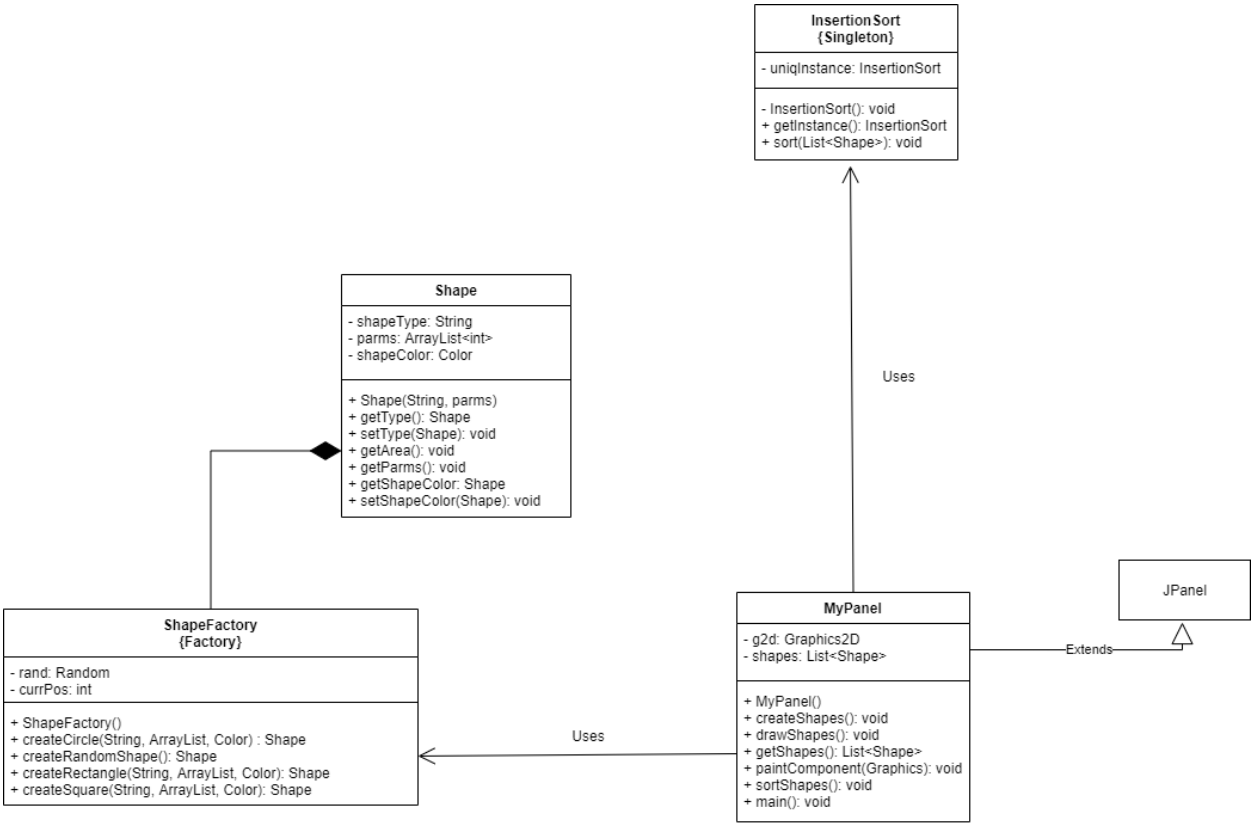
The main OO design principles used are Inheritance, Encapsulation and Abstraction.

Inheritance: Inheritance is shown by the *MyRectangle, MyCircle, MySquare* classes. These classes inherit from a single abstract *Shape* class.

Abstraction: Abstraction is shown by the presence of the *ShapeFactory* class. This class abstracts the instantiation of various shapes. This can be seen in the usage of *ShapeFactory* class in the *MyPanel* class, which uses the *ShapeFactory*'s methods to instantiate shape objects instead of directly instantiating the various *Shape* classes.

Encapsulation: Usage of Encapsulation can be seen in all the classes by the presence of private attributes which have explicit getters and setters. These getters and setters provide a consistent interface for retrieving and changing the attributes of a class effectively encapsulating the data within the class.

Now providing an alternate UML class diagram.



UML class diagram 2

The above class diagram delegates the responsibilities of the *MyReactangle, MyCircle* and *MySquare* into a single concrete *Shape* class with extra attributes such as the shapeType and an arraylist of parameters to the shape. For instance, a circle has a radius for parameters and a rectangle has a width and height for parameters. In this implementation, the importance of the *ShapeFactory* class is more significant since it provides a more cleaner interface for shape creation.

This UML diagram is more concise as compared to *UML class diagram 1* but it achieves this conciseness at the cost of clarity. And I believe that clarity in a codebase is more important than brevity. Thus, I believe that the design in *UML class diagram 1* contains a better design.

The tools that I chose for this project are as follows:

Intellij IDEA IDE, JDK Version: Java SDK 16, Git version Control, draw.io for UML modelling

Part 3: Implementation

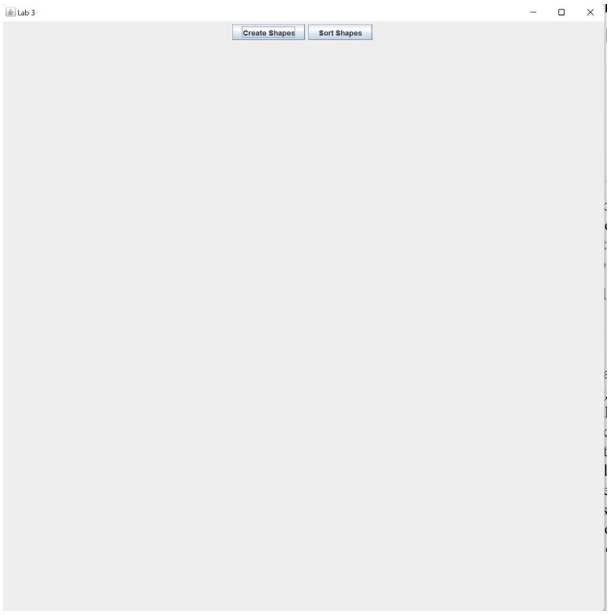
The insertion sort algorithm is used to sort the various shapes. This algorithm is described in the *InsertionSort* class. The pseudocode for the insertion sort elements goes as follows:

```
InsertionSort(A) {
     for i from 1 to A.lenght {
          for j from i to 1 {
               if (A[j] < A[j-1]) {
                    swap( A[j] , A[j-1] );
               }
          }
     }
}
```

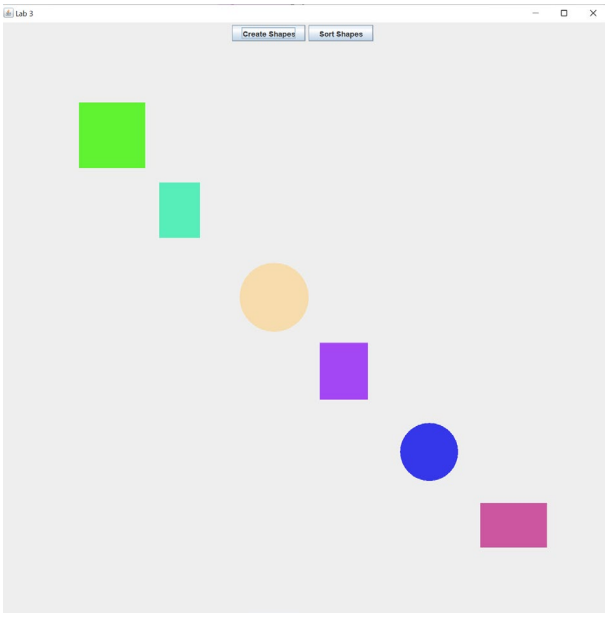
This algorithm runs in O(n^2). The idea behind the algorithm is to keep all the elements below the index i in sorted order and as the algorithm goes on i is incremented until i == A.lenght. Now, when an element at index j = i is considered, we go through the sorted array backwards starting from j and keep shifting elements to the right for which A[j-1] > A[j], when A[j] > A[j-1] that means that A[j] is in the correct place with respect to the sorted subarray A[0 … i].

Since we only have a small number of elements in our shapes array, insertion sort is a valid sorting algorithm to choose.

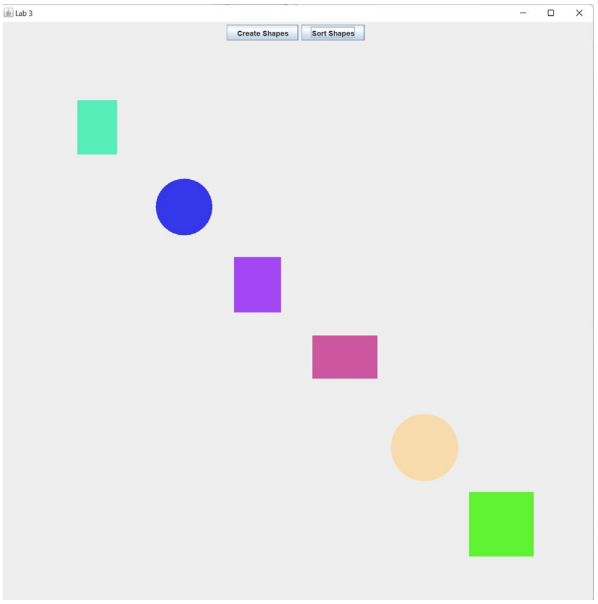
I have implemented the first UML diagram for my implementation. I have an abstract *Shape* class which is extended by *MyRectangle, MyCircle and MySquare* classes. Each of these classes have attributes specific to them such as the *MyRectangle* class . These 3 classes are then used by the *ShapeFactory* class in order to provide a clear interface for instantiating shapes. My *InsertionSort* class implements the insertion sort algorithm which is used to sort the various shapes drawn on screen. The *MyPanel* class is the main graphics window on which the shapes and the buttons are drawn. At the end the *App* class creates a JFrame which creates a new window and then uses the *MyPanel* class to draw the various buttons and shapes. The *App* class also has the main method from which program execution starts. All the classes are in the shapes package. The classes are automatically compiled by the Intellij IDEA IDE and the compiled classes are placed in out directory.



(1)



(2)



(3)

The first figure (1) is at the very start of the program. The figure (2) shows the creation of shapes after clicking the Create Shapes button. And at the end figure (3) shows the shapes being sorted after hitting the sort shapes button. The above figures shows that my code satisfies the software project requirements.

Part 4: Conclusion

Since I have already used JFrame and JPanel in a previous project of mine, I was already familiar with it. Thus, I was able to get the graphics window up and running in a relatively small amount of time. Another thing that went well with the software project is that I just took the EECS-2011 course and thus having to implement the insertion sort sorting algorithm, I was able to reinforce my understanding of algorithms and data structures.

Some of the things which went wrong with the software project is the extensive use of UML diagrams. Since I have not used UML diagrams before, it was a steep learning curve for me in order to get up to speed on UML diagrams and syntax.

From this software project I learnt the usage of various design patterns such as the singleton and factory design pattern. I also learnt about the usage of UML diagrams and how to effectively convert them into a concrete implementation. I also learnt the usage of various OO principles such as the Encapsulation, Abstraction and Inheritance.

Three things that I would recommend to someone are:

- Studying up on design patterns, such as the singleton pattern or the design pattern, on sites like geekforgeek or tutorialspoint.
- Reading on the documentation of the JFrame and the JPanel classes. Especially for someone who's not familiar with these classes.
- Thoroughly documenting the code.