**Khalid Gobin**
**216853483**
**EECS 3311**
**Section B**
**Software Project 1**
**Alireza Naeiji**

## Introduction

Software Project 1 is about creating a Java application that allows the user to load a set of six random shapes from a ___ of a circle, square or rectangle using one button, then sort by area using another button.

Some challenges experienced when designing Software Project 1 includes:

- Learning how to use Swing
- Learning how to create JPanels and JFrames
- Learning how to add functionality to buttons inclusive of adding event listeners and handlers
- Aligning panels on screen

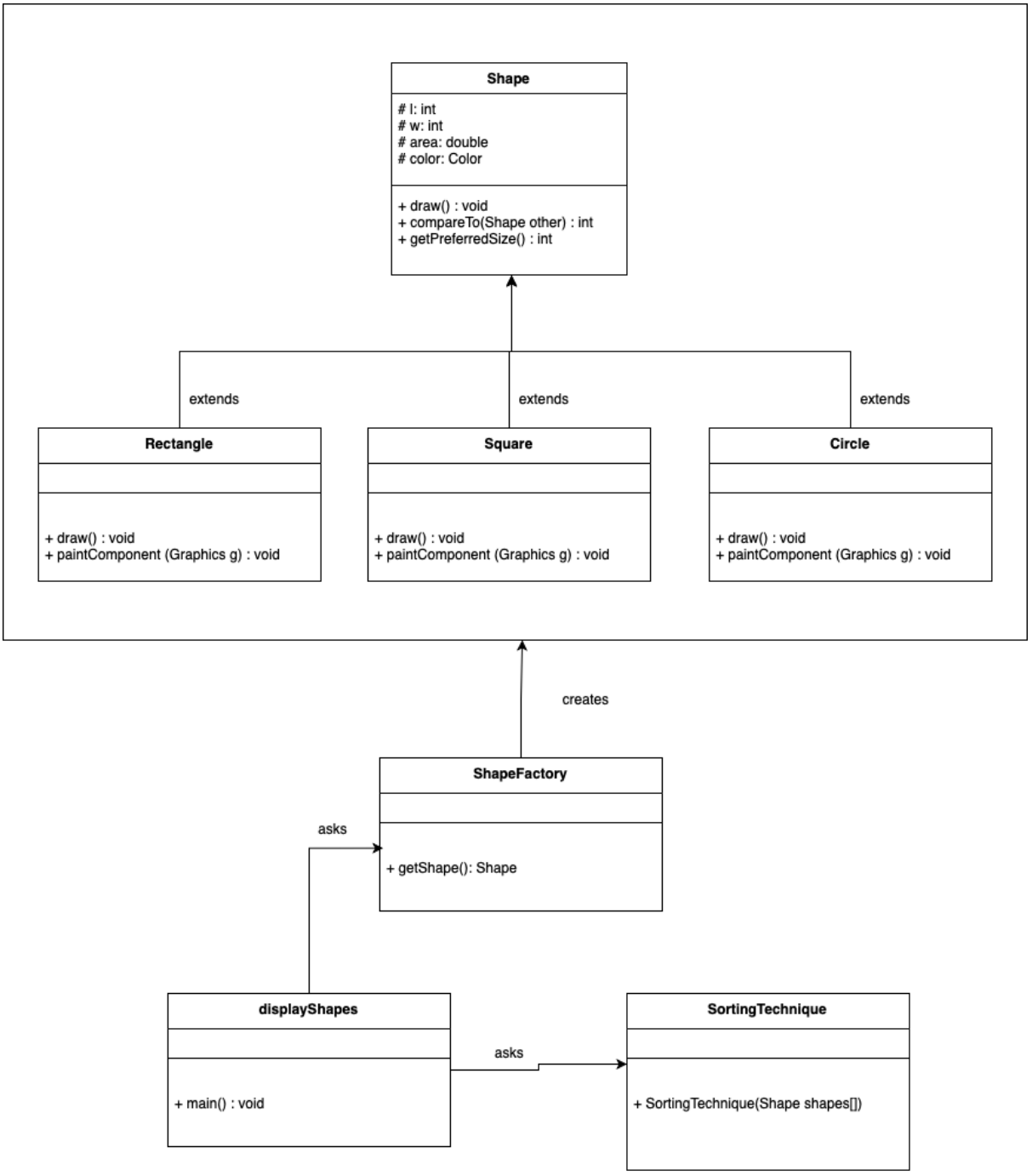
Throughout the creation of Software Project 1, object-oriented design and its four principles: Abstraction, Encapsulation, Polymorphism and Inheritance were used. Abstraction refers to hiding the unnecessary details (internal implementation) of a class to other classes. Encapsulation refers to keeping the logic (state and methods) of an object inside a class. Polymorphism refers to the ability of a method to take different shapes while Inheritance refers to making the child class reuse the parent state and methods without changing them.

The design patterns used are the Factory pattern and the Model View Controller pattern. The Factory pattern allows objects to be instantiated without revealing the instantiation logic to the client. The MVC Pattern stands for Model-View-Controller Pattern. This pattern is used to separate the application's concerns.

- Model - Model represents an object carrying data. It can also have logic to update controllers if its data changes.
- View - View represents the visualization of the data that model contains.
- Controller - Controller acts on both model and view. It controls the data flow into the model object and updates the view whenever data changes. It keeps view and model separate.

This introduction will be followed by in-depth details of the design of the application followed by the implementation.

## Design


Shape
# l: int
# w: int
# area: double
# color: Color
+ draw() : void
+ compareTo(Shape other) : int
+ getPreferredSize() : int
extends
extends
extends
Rectangle
+ draw() : void
+ paintComponent (Graphics g) : void
Square
+ draw() : void
+ paintComponent (Graphics g) : void
Circle
+ draw() : void
+ paintComponent (Graphics g) : void
creates
ShapeFactory
+ getShape(): Shape
asks
displayShapes
+ main() : void
asks
SortingTechnique
+ SortingTechnique(Shape shapes[])

**Object-oriented Principles Used in Class Diagram**

Abstraction: Abstraction was not used in the implementation of the application.

Inheritance: The subclasses Square, Rectangle and circle all inherited common features such as their length, width, area, and color from the Shape class. They also inherited methods that were common to them such as compareTo() which is used in the sorting algorithm. They also inherited a method called getPreferredSize() which makes the shapes appear bigger in the frame.

Polymorphism: This was used since the classes rectangle, circle and square inherited the draw method from their parent class (shape) and changed what the draw() method did every time.

Encapsulation: In order to sort the shapes, the SortingTechnique class simply called the compareTo() method of the shapes class. This is an example of encapsulation since the logic behind compareTo() is kept inside the shapes class.

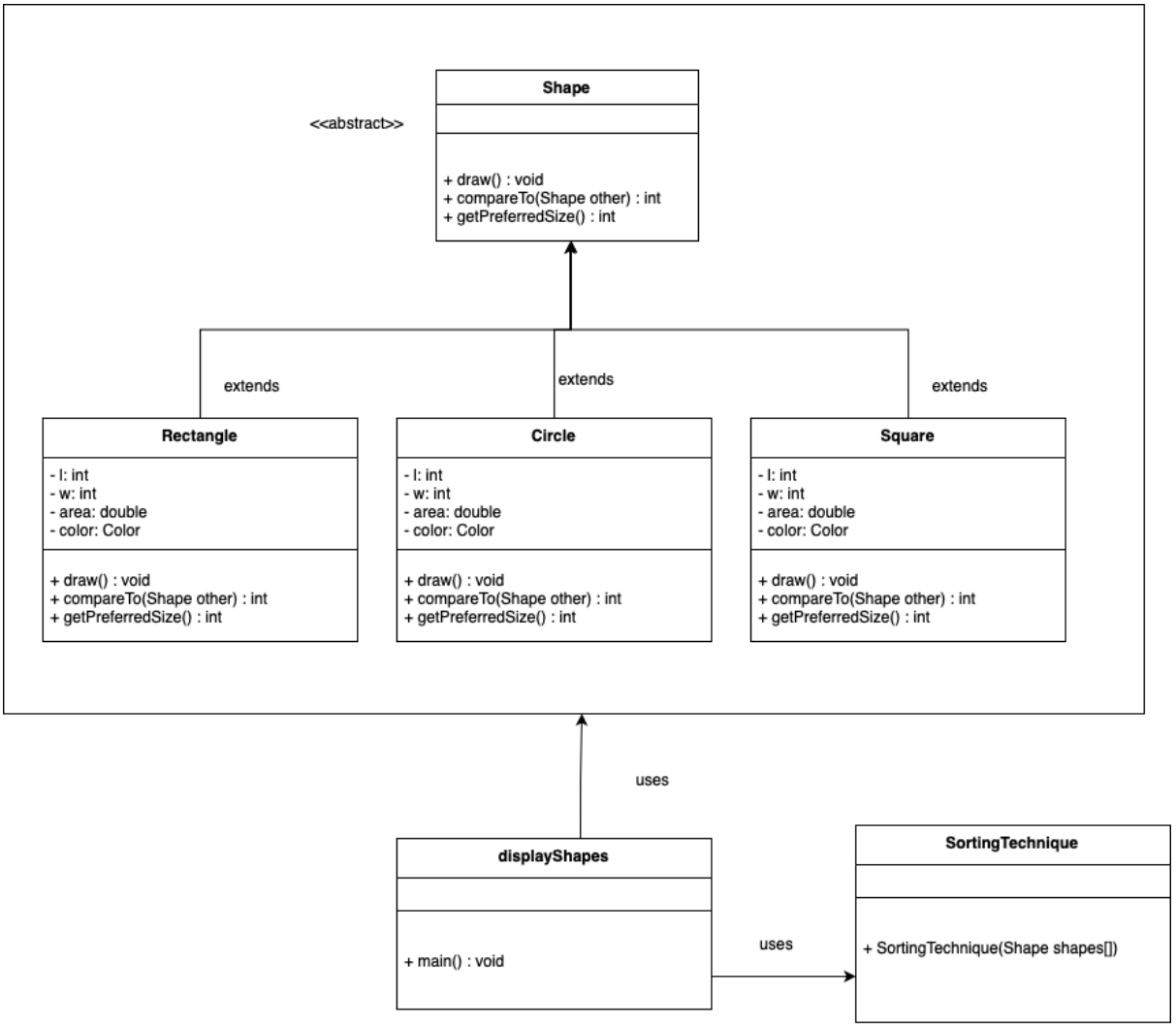
The Factory Design Pattern

The class, shapeFactory, was used to generate random shapes without letting the client know anything. The user just instantiates a new instance of the class shapeFactory then it runs a getShape() method which returns a random shape.

The Model View Controller Design Pattern

The model used was the displayShapes() class. This class allowed the viewing of the Shapes which were displayed on JPanels, and buttons for load and sort. The load and sort buttons were all linked to controllers which called the ShapeFactory class to generate random shapes and the SortingMechanism class to sort the shapes.

## Alternative Implementation




This alternative implementation uses the Template Pattern. An abstract class (Shape) created the template for how the subclasses (Rectangle, Circle and Square) must be implemented. This implementation involves writing more unnecessary code than the main implementation because methods that are the same in all three subclasses, must be implemented individually. The subclasses also have to declare their common variables rather than inheriting it from their superclass (Shape). The displayShapes method will also have direct access to the shapes which includes more client side code involved in its operations.

## Implementation

### Sorting Technique

Selection sort was used to sort the shapes.

**Selection sort** works by having two subarrays, the first array that is sorted then the second that is unsorted. The algorithm looks for the minimum element in the unsorted array and swaps it with the front element of the same array. The index of the unsorted array is then increased by one and the process of finding the minimum element and swapping it is repeated until the list is fully sorted.

For example:

10 25 40 2 -5 6
-5 25 40 2 10 6
-5 2 40 25 10 6
-6 2 6 25 10 40
-6 3 6 10 25 40 //fully sorted

### Description of Implementation

In this project, the first class diagram was used to implement the solution. Initially, Window Builder was used to design the layout of the interface and its components . This was connected to the DisplayShapes class which is used to control the interface. The displayShapes controlled the JFrame, JPanels and two JButtons namely the Load button and the Sort button. When the load button was clicked it created the ShapeFactory class. A method getRandomShape() was then called on the instance of the ShapeFactory class until six shapes (stored in an array) populated the JPanels. The getRandomShape() returns a shape that is either a rectangle, circle or a square. It does this by generating a random number between 0 and 1 and creating the specific shape if the number falls inside the specified range. It also assigns it a random color. After the shape is created, the draw() method is called which instructs the system on how to draw the shape. Each shape has its own unique draw method which is overridden in each subclass. The common attributes of the shapes are stored in their parent class: Shape. This shape class also contains methods that are not overridden such as the compareTo() method and the getPreferredSize() method.

When the sort button is clicked, it creates a new instance of the SortingMechanism class whose constructor accepts the array of Shapes (pass by reference). The array is then sorted by selection sort and the panels that the shapes are assigned to are cleared, revalidated and

repainted. This updates the panels with the sorted shapes and the sort function of the program is now complete.

Tools used

Java Development Kit Version 16
Eclipse version 3.18.600.v20201202-1800
Window Builder version 1.9.7

Snapshots

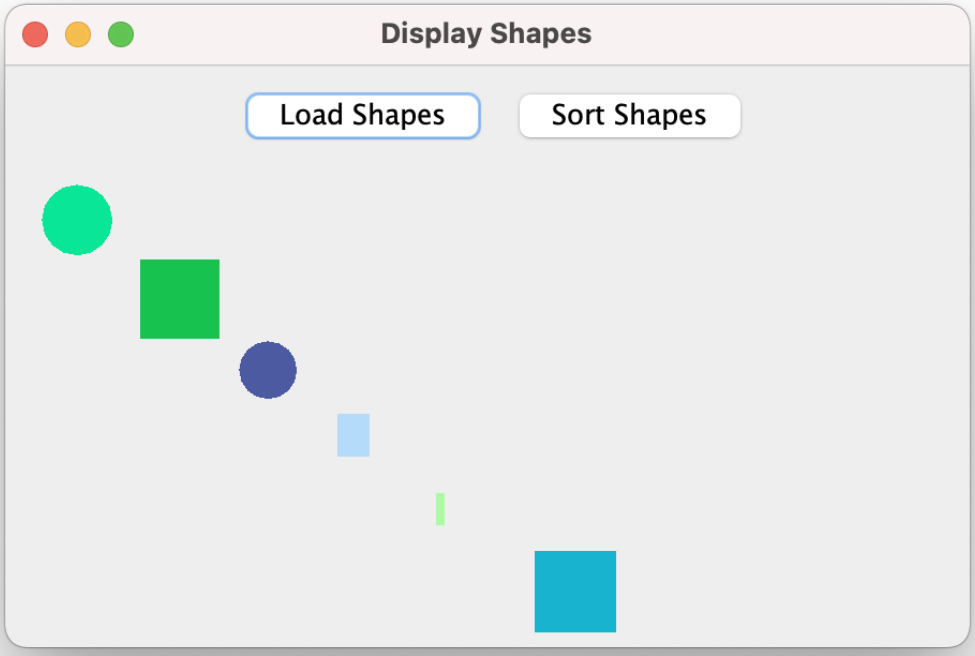



*Image1: When the "Load Button" is clicked*

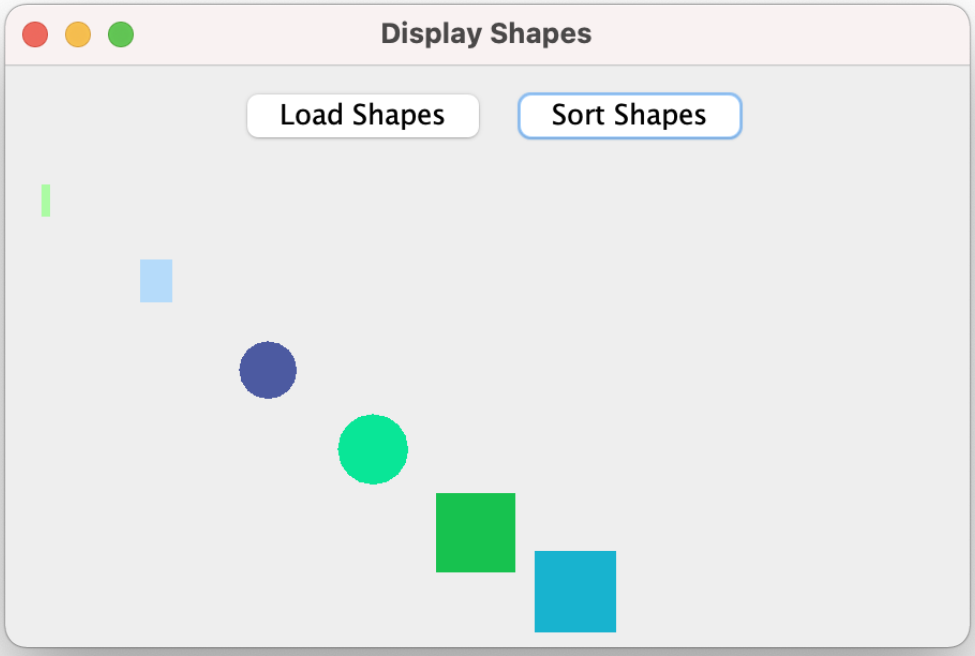



*Image 2: When the sort button is clicked*

## Conclusion

In this software project, many challenges were encountered and many hurdles were overcome. A few things that went ideally include: creating the GUI and structuring the classes correctly. However, a few things that went wrong include: incorrectly structuring the JPanels which caused an error for them to update after new data was provided to them. The interface class had to be redone from scratch after that.
I learned many things in this software project including the steps to creating a java application with a GUI as well as the procedure of adding action listeners to buttons.

My top three recommendations to ease the completion of the software project are:

- Use WindowBuilder to create the GUI
- Use different classes for the three different types of shapes
- Use removeAll(), revalidate() and repaint() after updating every JPanel.