Kashyap Patel   216785339   EECS 3311A   First Software Project: ShapeShifter   Kazi Mridul

INTRODUCTION

This software project is about plotting shapes on an interface. It has the following goals:

- To instantiate six shapes that can only be squares, rectangles or circles.
- The application should display the shapes on an interface.
- The interface should have two buttons, one to draw the shapes and the other to sort the shapes
- The application should use a sorting technique i.e., quick sort, bubble sort etc.
- The shapes cannot overlap each other and should be visible on the interface.

The challenges associated with the projects are, choosing the right OO design principle and implementing it according to the given goals. However, this can be solved if we tackle this problem step by step.

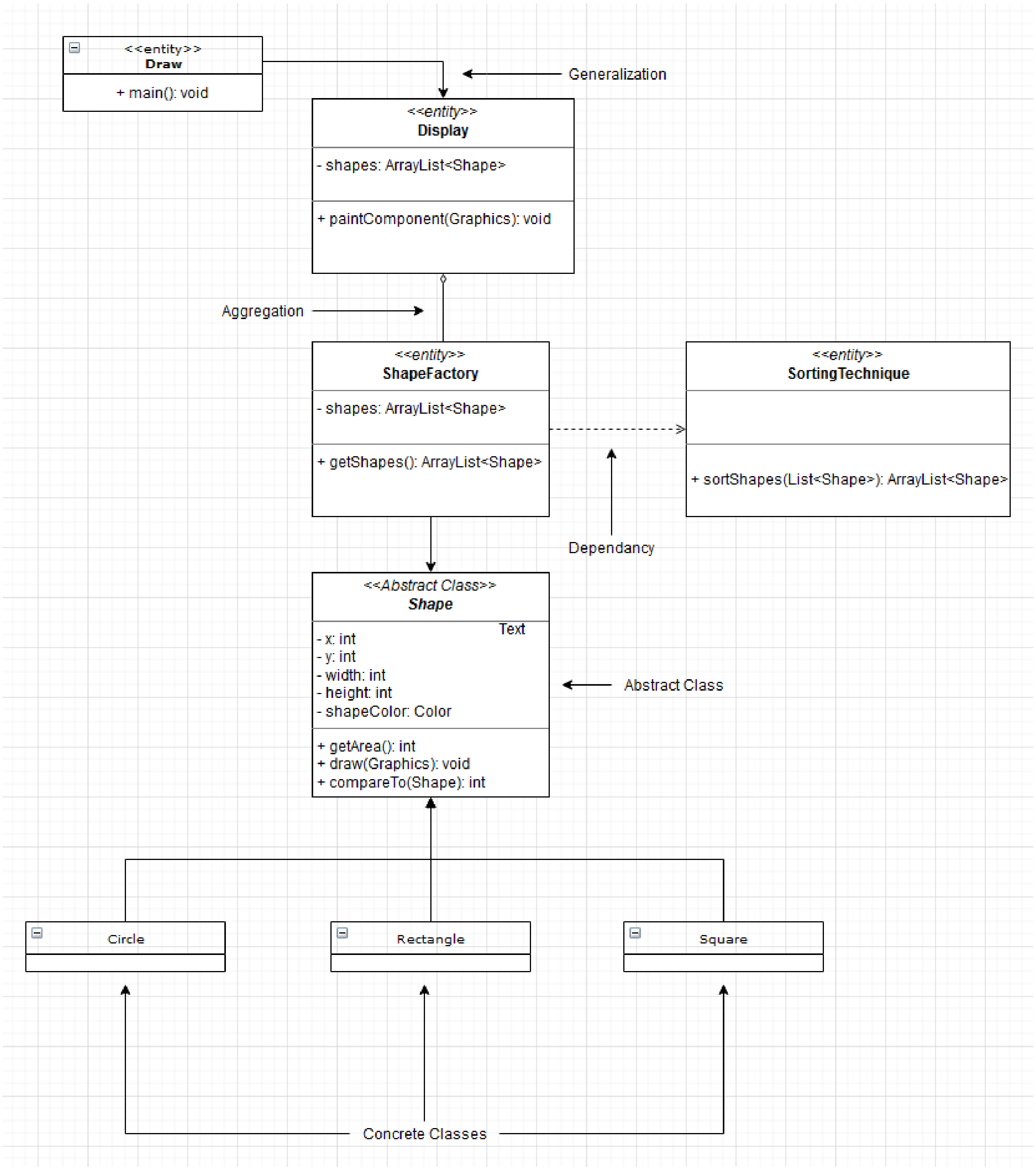
The main OOD principles that will be used are, Abstraction, Encapsulation, Inheritance and Polymorphism. This will be due to the Shape being an abstract class, and its children will have private methods. There will be ArrayLists that take in all kinds of shapes, this will take care of polymorphism. The design patterns considered for the project are, Singleton and Factory pattern. We will explore the benefits of using one over the other below.

The report will evolve as we continue though working on the project. The specifics will be changing since, software development is a process. We will use the AGILE method where we will work on small parts and test instead of big chunks.

Kashyap Patel 216785339 EECS 3311A First Software Project: ShapeShifter Kazi Mridul

DESIGN OF THE SOLUTION

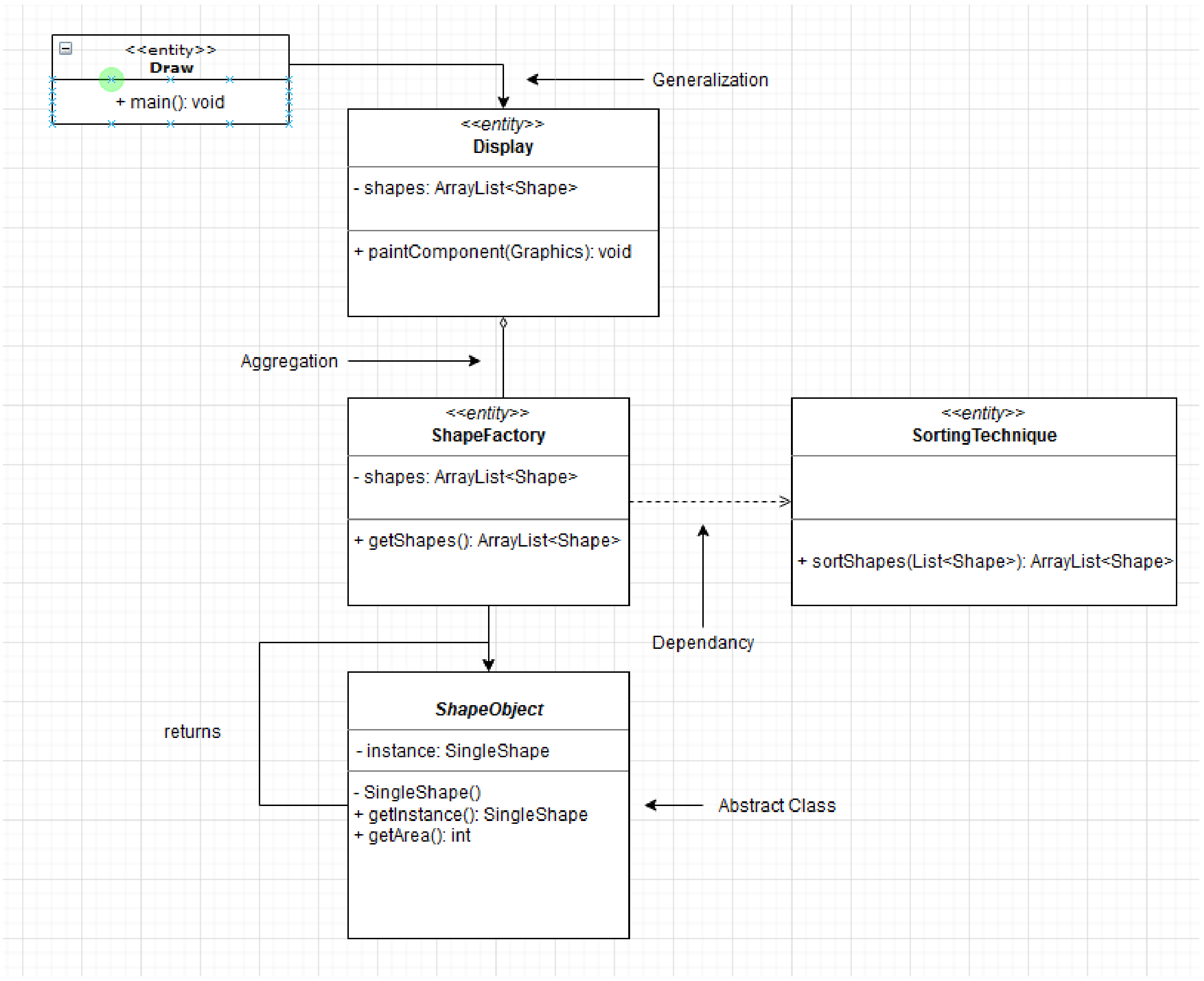
1) FACTORY PATTERN



As we can see above:

- Shape is an abstract class, with 3 concrete classes. This OO principle is Inheritance/generalization.
- ShapeFactory only exists to build shapes, Therefore, it is part of the Shape. This is an aggregation relationship.
- The Display class is dependent on Draw since that is where the JFrame is.
- All instance variables of the classes are private, this is encapsulation.
- getShapes return an ArrayList of type Shape. This means there can be variety of shapes in the ArrayList. This is polymorphism at play.
- SortingTechnique is the class where we have used InsertionSort, to sort the ArrayList of shapes for us to view on the JFrame. It doesn't have anything to sort without the ShapeFactory, therefore, it is dependent on it.

2) SINGLETON PATTERN



For the Singleton Pattern, we can see that:

- There is a static factory method, which will give us a single instance of a Shape every time we call the static method.
- Since, there is no inheritance relations, to specify the type of shapes we will need to create more static factory methods for the other shapes. This is tedious and hard to keep track of.
- We cannot use this pattern in this task since we are given inheritance relations and types of shapes.
- Overall, this is a very simple design and one of the best ways to create an object, but it falls short when there are “multiple” types of objects.

In conclusion, The Factory Pattern is the superior for this task; therefore, I will be using that for the project.

IMPLEMENTATION OF THE SOLUTION

The algorithm used to sort the shapes is **InsertionSort**. InsertionSort is a simple algorithm. It is like shuffling cards, whereby values from the unsorted part are picked and placed at the correct position in the sorted part. The Average Time complexity is $O(n^2)$. This is okay but not the best, however since we only had 6 elements to sort through, it was my favorite pick.

The classes are implemented in a **Factory Pattern**. There is an inheritance relation between the Shape class and the Rectangle, Square and Circle classes. This seemed the most logical and efficient way to design and implement this task. It was much easier to keep track of all the classes this way instead of having the singleton pattern which would have been a headache to keep track of.

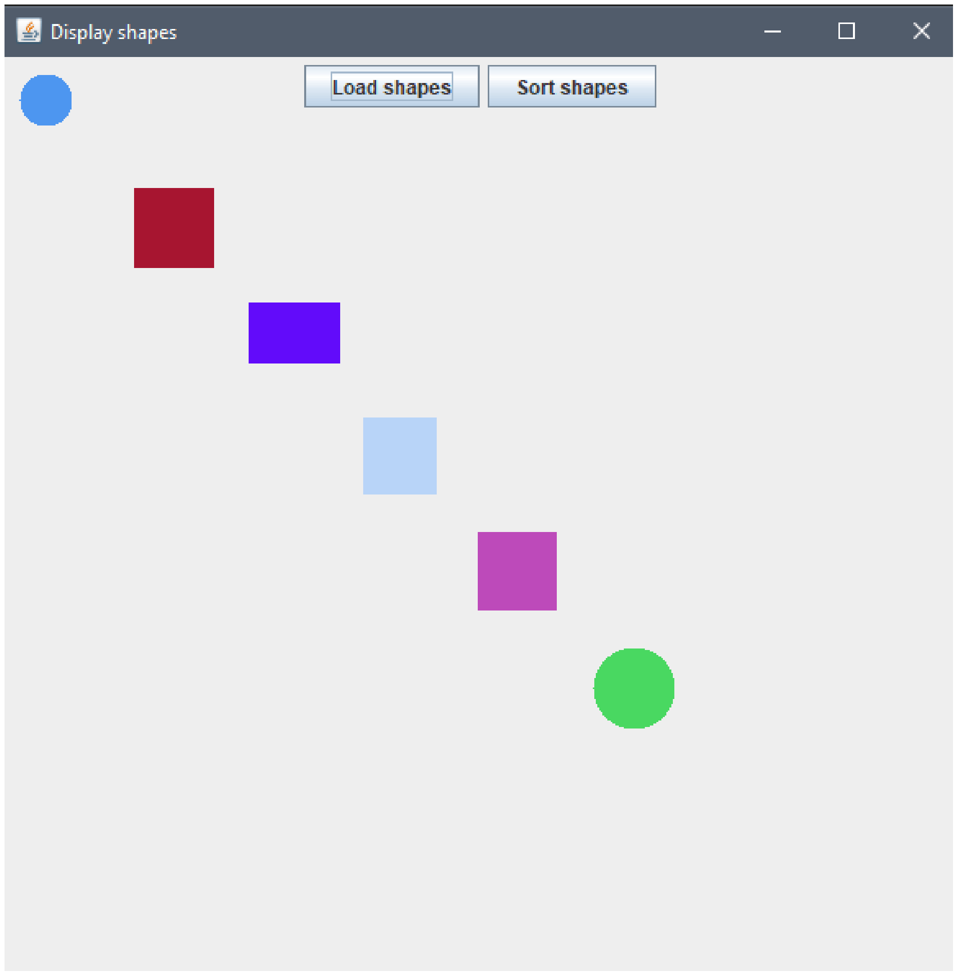
The main tools used during this task are Eclipse as the IDE, JDK 17 is used. To create the UML diagrams, draw.io was used.

Snapshot:

Upon clicking

“Load shapes”,

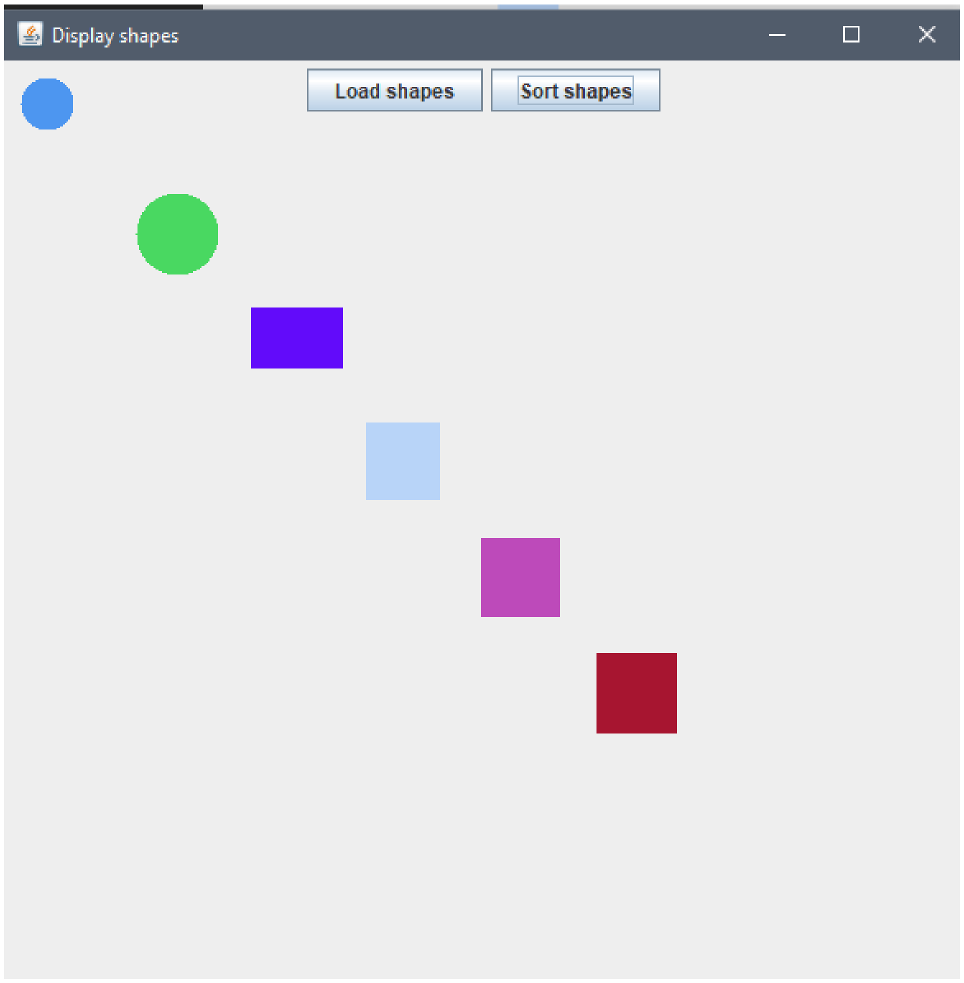
We get these shapes.



Upon clicking

“Sort shapes”,

The same shapes are sorted ascendingly.



Video is uploaded at this link, it is also included in the github files:

https://youtu.be/HSg571NhfC8

CONCLUSION

The best thing was that, due to my previous knowledge in java programming, the designing part was very easy, and I could visualize the final product. Once I planned the design the implementation was a breeze.

Towards the end, I got stuck in the “Sort button” part. It was a silly bug where the x and y coordinates weren’t updating with the sorting algorithm. Eventually, after spending a couple of hours in debug mode with breakpoints on, I figured it out and squashed the bug.

I have learnt how to produce a GUI for java applications and also learnt how to create an executable file. I also learn the right way to tackle a project and follow agile methodologies.

First recommendation would be to get a better understanding of what the user(Professor) wants by asking questions in the forum.

Secondly, to space out time to attack the problem. Take short breaks to stretch and have a snack to keep the mind fresh.

Lastly, learn how to read Javadocs of existing java libraries. I found myself reading the Javadoc often and learnt a lot from it.