Denis Kosogov 214407381 Section A
Shape Builder – Lab 1
TA: Kazi Mridul

Shape Builder – Lab 1

Part 1: Introduction

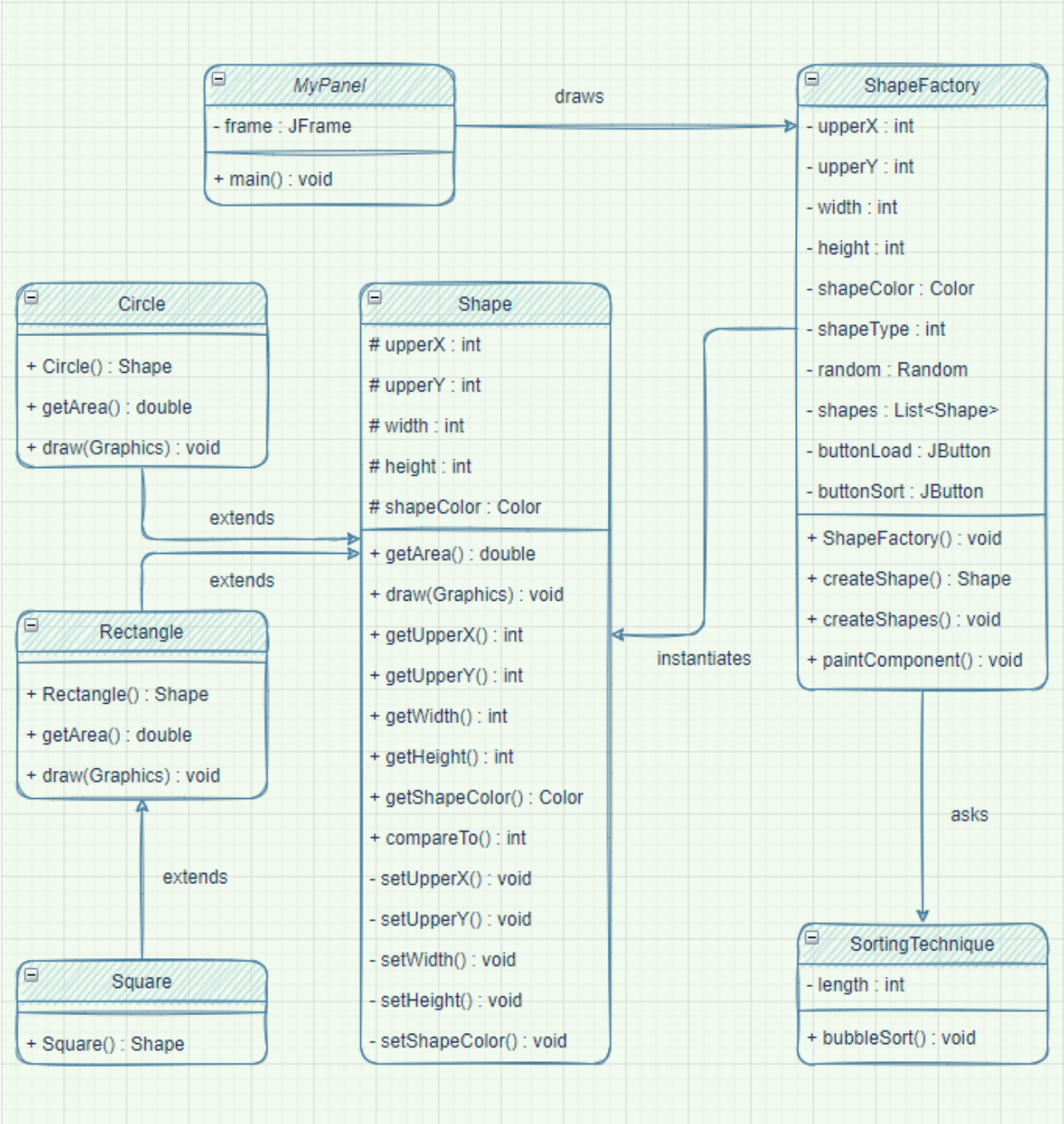
The software project is about creating a shape builder with a user interface. The two main functions of the software are to display 6 random shapes on the screen, and to be able to sort the shapes by its area. The shapes include rectangles, squares, and circles. Each shape is instantiated with a random value for width, height, and color. The project askes to create a UI with two buttons “Load Shapes” and “Sort Shapes”. When pressing “Load Shapes” 6 random shapes are instantiated in a list and are displayed on the screen. The “Sort Shapes” button compares all of the shapes in a list based on area and re-arranges them including the upper X coordinates and upper Y coordinates. The shapes are then re-displayed on the screen in the new order.

I have experience working with Object Oriented Design, thus setting up the objects in a simple but effective structure was not difficult. The main challenge I faced while completing the project was the swing framework to create the UI. After instantiating the shape objects in the list, I was able to display them within the JFrame, however, I could only display either the buttons panel or the shapes themselves. After hours of trial and error, reviewing the documentation for swing framework and watching a few YouTube videos, I was able to create an Action Listener for the buttons to properly display the shapes on the screen. I believe swing is an outdated framework.

Firstly, to create the shapes I will apply the OOD concept by creating individual objects to represent each shape (Rectangle, Square, Circle). These objects will inherit from a shape class object which utilizes inheritance and polymorphism to generate these shapes. The classes also utilize abstraction by hiding the unnecessary implementation of the objects. I will also try to maintain encapsulation to bind together data and functions which manipulate this data to avoid outside interference. I will utilize the factory pattern to instantiate the shapes using a single class. The shape class can be viewed as a decorator pattern, where new functionality can be added to an existing object without altering its implementation.

I will structure my report according to the requirements given for the software project. I will show and explain the process I took to design the structure of the program using UML class diagrams and give reasonings behind my design decisions. I will also create a demo video to show how to execute the software and what the correct output will look like.

Part 2: Design of the solution



This is a class diagram for the software project. I have split up the shapes into different classes which inherit from a parent class of Shape. I have utilized principles such as abstraction, polymorphism, encapsulation, and inheritance to create the structure of the program. Here we see that the square class inherits from the parent class Rectangle which also inherits from its parent Shape class. All of the object generation, creation of the list of shape objects, displaying the graphics using the paintComponent and using the sortingTechnique is all handled by the ShapeFactory class. This class is also responsible for using the Action Listener to listen for button clicks and perform the appropriate task. The MyPanel class is responsible for running the software using the main class and generating the user interface.

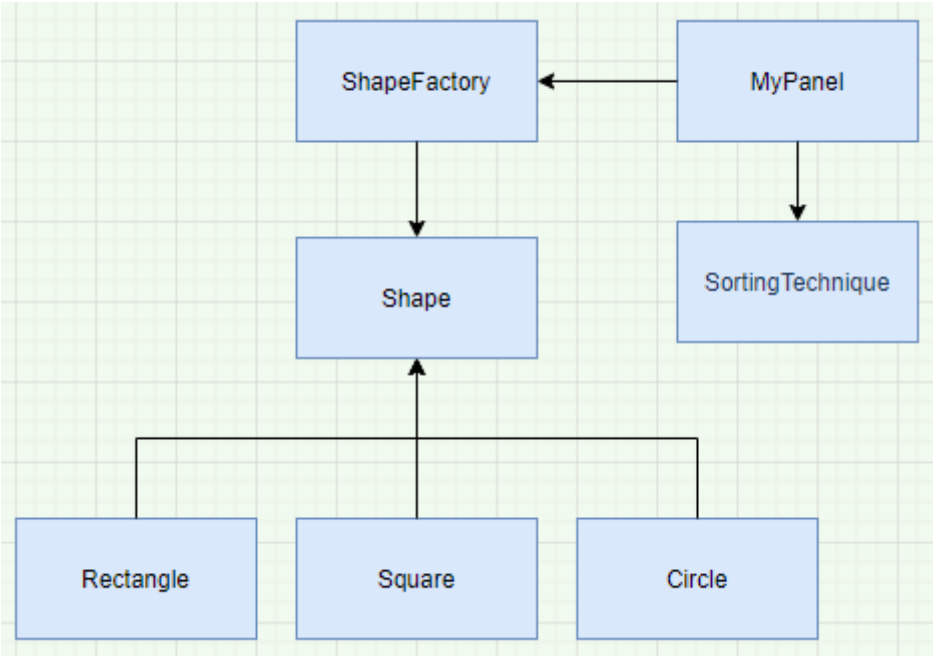
Inheritance is used throughout the project. The square class inherits from rectangle class which inherits from the shape class. Similarly, circle also inherits from the shape class. I use inheritance in this case to remove unwanted code, and instead reuse classes and methods from parent classes. The shape class declares all the variables and the common getters and setters which can be access for each of the shapes. We also see the use of inheritance when working with the swing framework. In ShapeFactory class we inherit from the JPanel class which lets us use the components within the framework. Also, when working with the buttons, I set up action listeners for the button clicks using Action Listener component.

Abstraction can be seen in action within the swing framework and Action Listeners. We simply import the appropriate methods needed for our use and do not care about how these methods are implemented. Similarly, in our software, withing the ShapeFactory class we create new Rectangle, Square, and Circle objects using abstraction. To this class the implementation of these objects is not important, and we simply instantiate new objects for our needs.

Encapsulation is visible through the constructor methods within our classes. I have bundled a set of attributes which store current state of the object with a set of methods using these attributes. This is one of the most commonly used principles in Java. I have created getter and setter methods to access the attributes and change their states within the Shape class, and the Rectangle, Square, and Circle classes simply inherit from its parent class.

Polymorphism also has many instances within this software project. This occurs when we have various classes which are related to each other by inheritance. I have created two methods in the Shape class and kept them abstract. These methods include the getArea() method and the draw() method. While any shape can utilize these methods each individual object has its own implementation of these methods. For example, the implementation of getting the area of a circle differs from a rectangle. Similarly, the implementation of drawing a circle as a graphic differs from rectangle and square.

Alternative UML:



In this diagram we use MyPanel as the main driver of the program instead of the ShapeFactory. It is responsible for creating the instances of the shapes and also for calling the sorting algorithm to output the sorted shapes on the screen. Also, we do not utilize the inheritance to its best form by not inheriting the square class from the rectangle class. This is not wise, since we have repeat code, while the implementation of both shapes is very similar.

Part III: Implementation of the solution

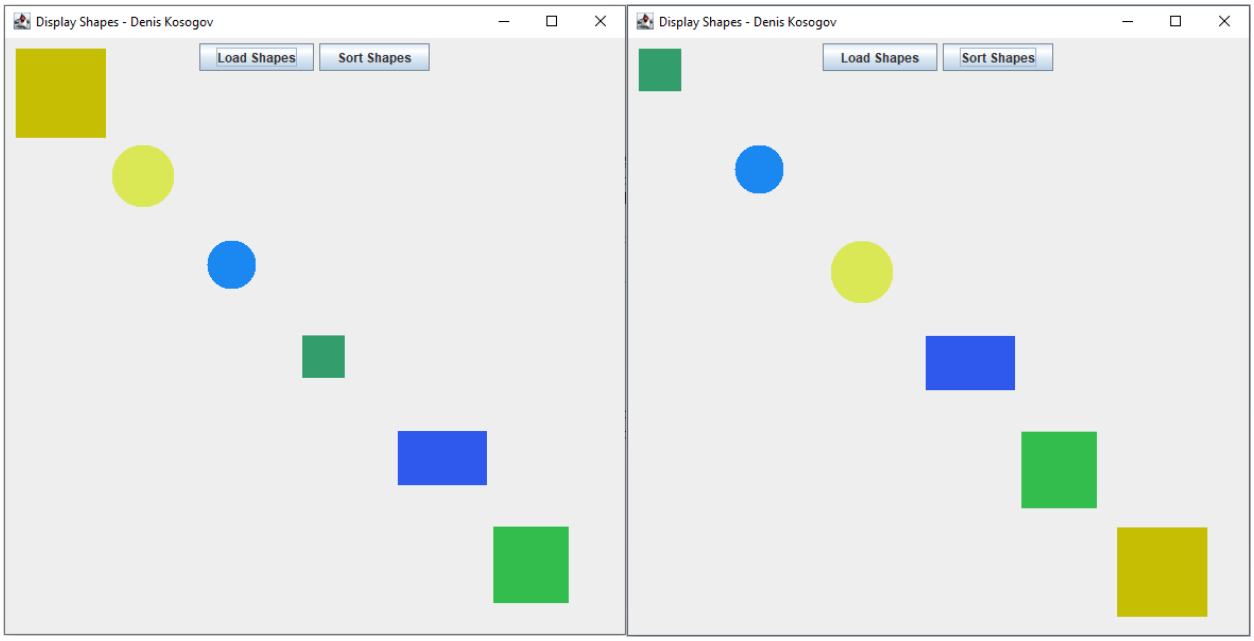
The algorithm of the sorting technique I have chosen is the bubble sort. Bubble sort is the simplest sorting algorithm which repeatedly swaps the adjacent elements if they are in the wrong order. Considering the simplicity of the software I found bubble sort to be the easiest to implement. Since there are only 6 objects that are instantiated, I really do not have to worry about the complexity or poor performance. However, the optimized implementation of such algorithm yields in O(N^2) time. Worst case occurs when array is reverse sorted and results in O(N^N) time, and best case is when the array is already sorted which results in O(n) time.

I create three different classes for Rectangle, Square, and Circle. The Square class inherits from the Rectangle class since the object is basically the same with the difference being width and heigh are the same. These classes have a simple constructor and two methods getArea() which calculates the surface area of the shape based on its width and height, and a draw() method which creates the graphics to represent the shapes. This method takes in the location of the upper x coordinate and the upper y coordinate, as well as the width, height and color. All

these classes inherit from a decorator class called the Shape class. The reason for this class is to declare the variables needed to fulfill the constructors and the getters and setters for manipulating such variables. With all this in mind, I have created a factory class name the ShapeFactory which handles the creating of the shapes with random values passed on to the variable parameters. This class is responsible for instantiating a list of shape objects and displaying them in the JFrame. I have implemented Action Listeners in the constructor of the class to pick up whether the “Load Shapes” or “Sort Shapes” button is clicked. The “Sort Shapes” calls a method from a different class to sort the shapes based on its surface area. Finally, MyPanel class is responsible for the execution of the software by utilizing the main method and creating the JFrame interface. You can see the relationships of the classes by looking at the first UML diagram.

While being most comfortable working in VS Code, I chose Eclipse to carry out this project. Eclipse makes it very easy to generate getters and setters for my objects and saves a lot of time. The version of Eclipse I used is 2021-09 build (4.21.0). I simply downloaded the latest available and began to work. I used the JavaSE-16 JRE and utilized the swing framework to create my user interface.

Execution of the code:



Load Shapes is Clicked Sort Shapes is Clicked

PART IV: Conclusion

What went well in the project was settings up the different classes to represent the shape objects. Using Eclipse to generate getters and setters and creating simple constructors was very easy and quick. The toughest part of the software project for me was figuring out the swing framework to display the shapes on the screen upon button clicks. I have gained experience working with the swing framework, and refreshed my knowledge on the Object-Oriented Design and its patterns to develop a simple yet comprehensive software. After reviewing the requirements for the project, I developed a plan in mind and started executing it.

Three recommendations:

1. Plan out the structure of the classes before beginning the implementation.
2. Keep everything modular by creating simple methods for easy maintenance.
3. Make sure to keep the structure simple but reasonable to fulfill the requirements.