Uzair Arif, 216852907, EECS 3311 – Software Design, Section A, Kazi Mridul, UzairSP1

Introduction

This project is about creating a GUI that contains two buttons, one that loads 6 shapes and displays them, while the other sorts those six shapes.

The greatest challenge will be in sorting the shapes and the repainting the panel that displays the shapes; this is because the shapes that will be sorted based on the result of their calArea() method.

OOD principles I will use (further explained later):

- Abstraction
- Inheritance.
- Polymorphism

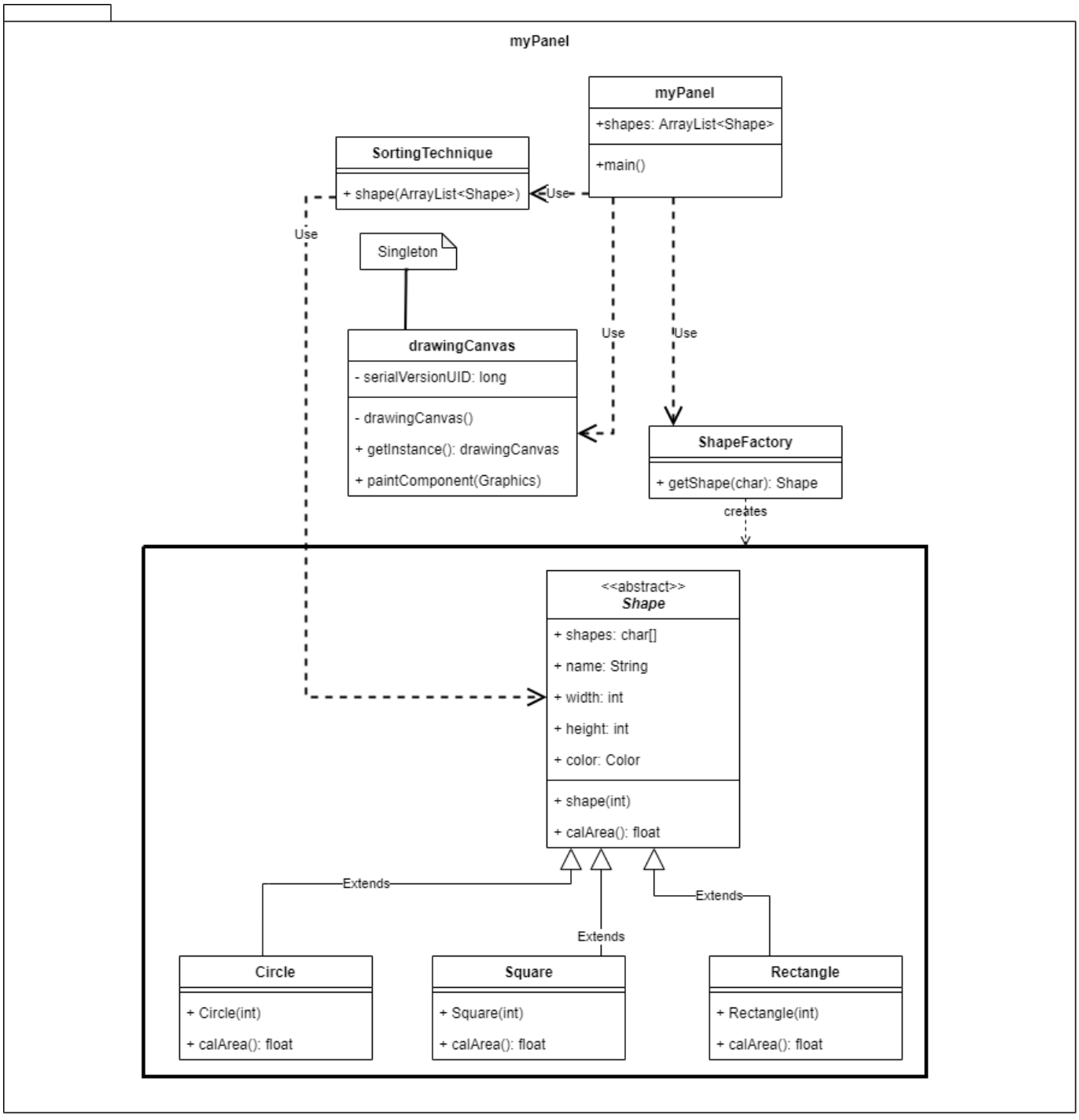
Design patterns I will use:

- Factory pattern: The ShapeFactory class is a Factory class that generates the concrete objects of Circle, Square and Rectangle.
- Singleton pattern: The drawingCanvas class is a singleton because you cannot create an instance of it; you would have to use the getInstance method to get an instance of drawingCanvas. Also, the same instance is used throughtout the whole runtime.

My report contains this pdf, a video and a myPanel folder. The myPanel contains all the data needed for Eclipse. The video shows launch the code and run it.

Design of the solution

First UML class diagram:



myPanel is the class that contains the main methods, it's the one that displays the GUI.

Shape is an abstract class that contains the fields and methods its children will have.

Circle class is used to create a circle object, Square class creates the square object and Rectangle class creates the rectangle object.

drawingCanvas is a singleton which returns the drawingCanvas instance, its paintComponent is what draws the shape on the interface.

SortingTechnique class contains the method that sorts the shapes.

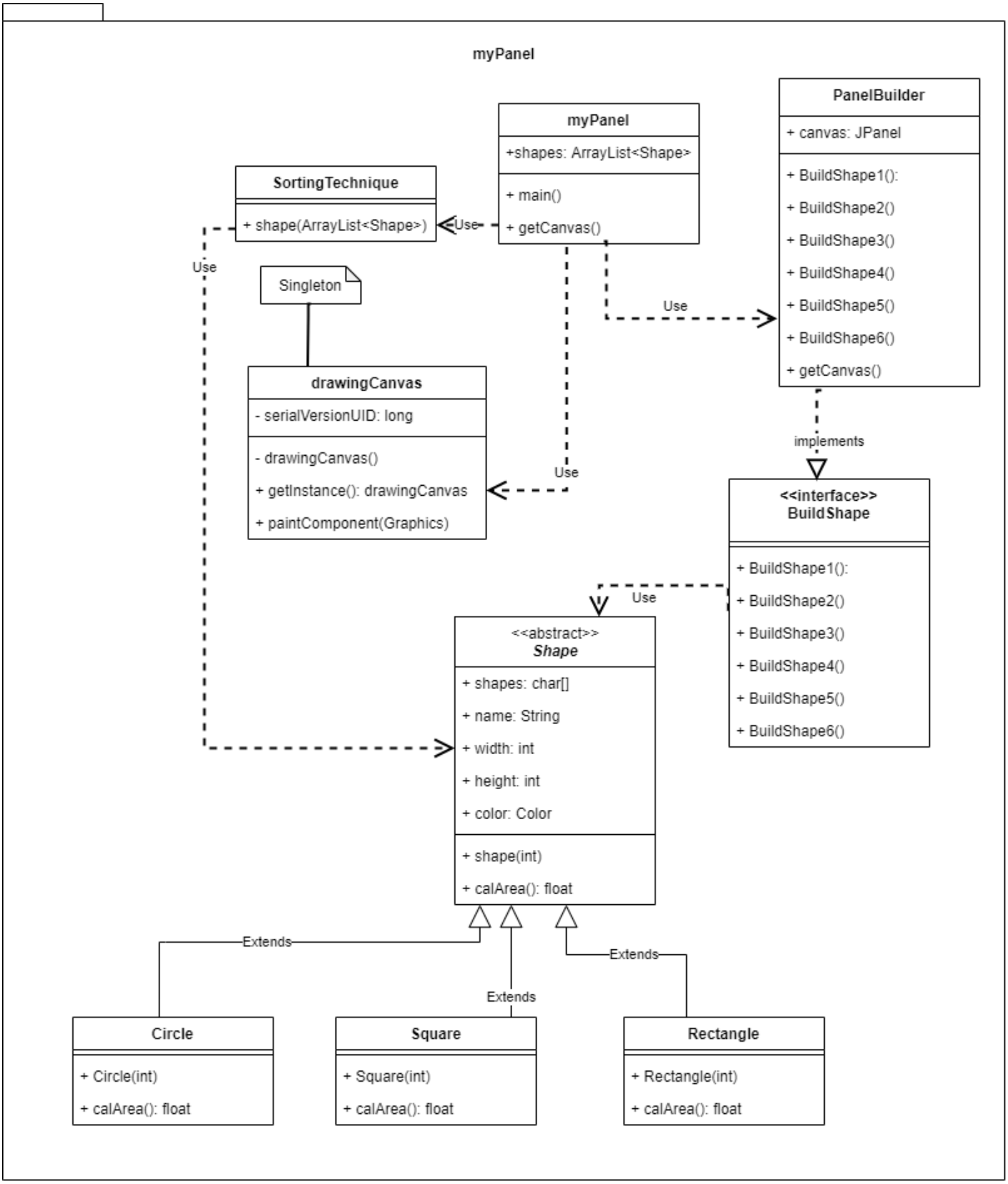
ShapeFactory is a factory class that produces the shapes and sends it to myPanel.

Any changes to ShapeFactory, SortingTechnique, and drawingCanvas will directly affect myPanel.

OOD principles used:

- Abstraction: The Shape class is an abstract class.
- Inheritance: The Circle, Square and Rectangle inherit the fields and methods from the Shape class.
- Polymorphism: The calArea() method its first defined in the Shape class and then inherited to Circle, Square and Rectangle, but all these classes have their own implementation of the calArea() method.

The second UML diagram:



This class diagram is like the first except that it uses a builder class instead of a factory class.

The BuildShape is an interface class contains separate methods for creating each of the 6 shapes.

The PanelBuilder uses the 6 methods from BuildShape to build the panel that will contain the 6 shapes and send it to myPanel, which will then use the main method to display it.

Both diagrams would produce the same result; however, the first diagram is a better design because:

1. The implementation is easier
2. Uses less memory space as there are less classes and methods
3. More flexible compared to the second, because it easy to make changes if needed.

Implementation of the solution

The sorting algorithm is like selection sort, but instead of bringing the smallest element to the start, my algorithm brings the largest element to the end.

I have implemented the first class diagram using Eclipse.

First I implement the main class (myPanel) which displays the GUI. I used the swing class to make to interface. The first thing I made was the two buttons, and then the gray panel which will later contain the shapes. The variable that stores the gray panel is called ‘shapePanel’.

Next I made the Shape class along with the subclasses Circle, Square and Rectangle; any fields and methods that was common among the three classes was then implement in the Shape class and then inherited from there to the subclasses.

The Shape class has 5 fields and 1 method:

- Shapes which is static and contains the letters c, r and s
- name, which would be either ‘circle’, ‘square’ or ‘rectangle’
- width, which is the horizontal length of a square, rectangle and circle
- height, which is the same as width for a square and circle
- color, which will be randomized
- calArea(), this calculates the area, so this method would be different for each of the children.

The rectangle calculates the area by multiplying the width and the height. The square calculates the area by doing [width ^ 2]. The circle calculates the area by dividing the width by half, squaring it and then multiplying it by pi.

Next, I made the ShapeFactory class which take a character as an input (c, s or r) and would then create the appropriate shape. The myPanel class select a random character from c, s and r, and then executes the ShapeFactroy’s getShape() method to get the appropriate shape. The getShape() chooses a random integer between 20 and 55, and this random number becomes the shape’s width. If the shape happens to be a rectangle, then the method chooses a random integer between 20 and 55 again for the height.

The myPanel has a static shapes array which stores all the 6 shapes.

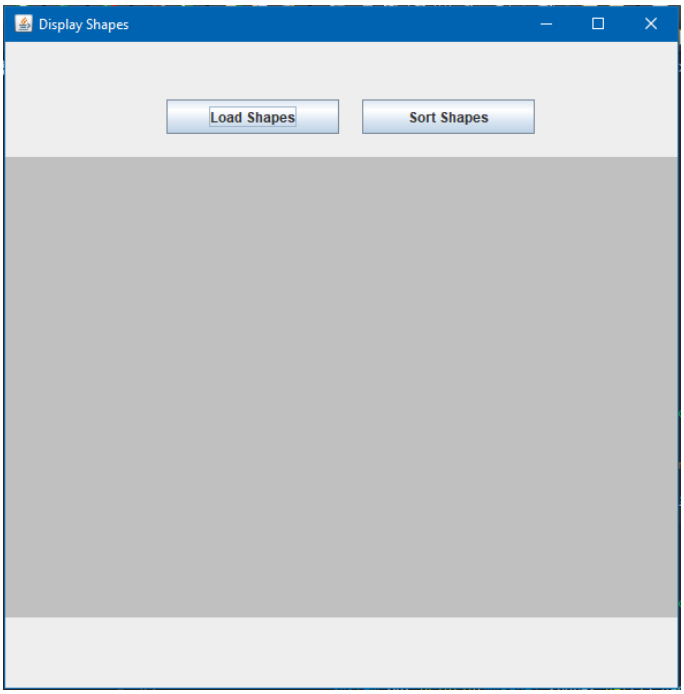
Next, I made the drawingCanvas class which uses the paintComponent method to draw on the shapePanel. Everytime the paintComponent executes, the frame repaints to display those shapes.

Lastly, I made the SortingTechnique class, which sort the shapes based on its area. The algorithm uses myPanel’s shapes array and another array called arealist, this array is a list of 6 values, and each one is an area corresponding to the shape in the shapes array in the same index.

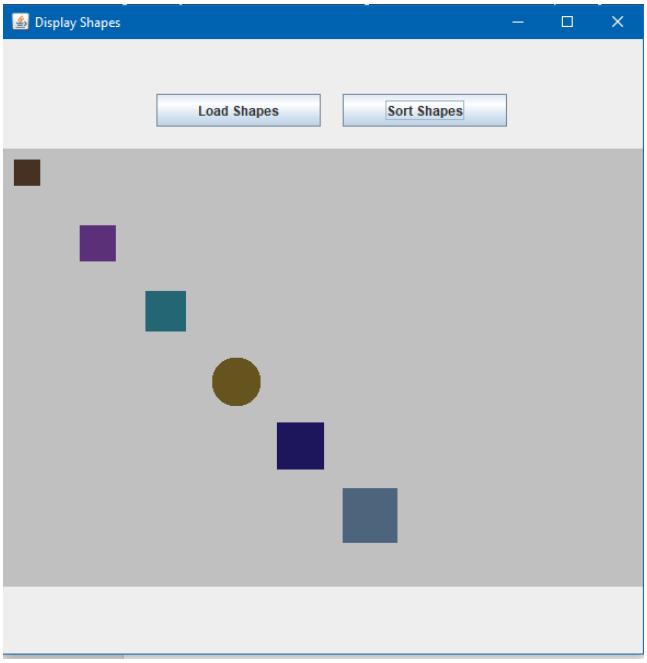
The SortingTechnique class sorts arealist, but everytime the two values in the arealist are swapped, the appropriate shapes in the shapes array is also swapped.

All the classes mentioned will be in one package (myPanel). Also the Java version is 11.0.3.

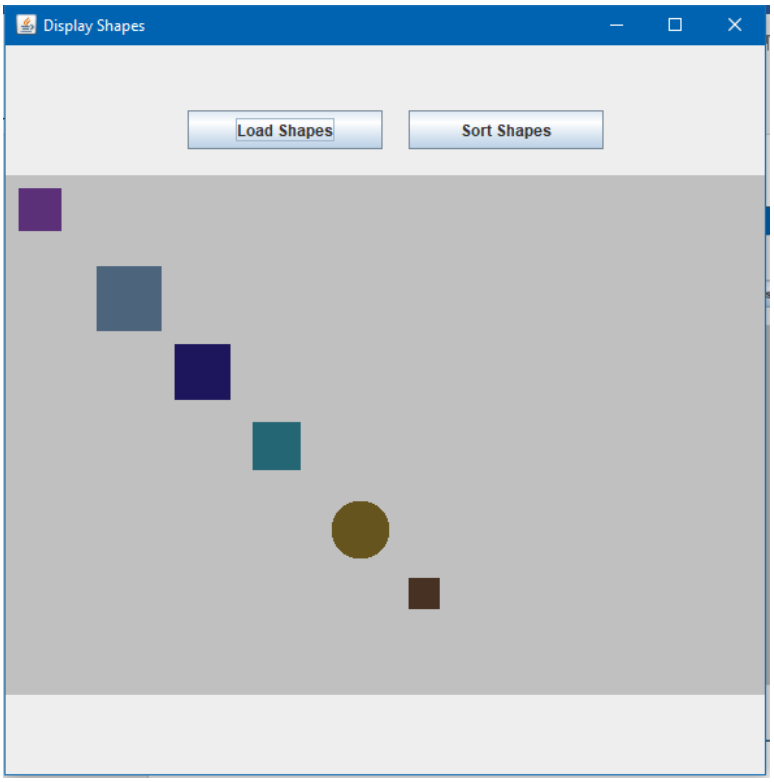
This is what my interface looks like:



The gray part is where the shapes will be displayed. Here is an example with shapes loaded:



And here is the same example with the shapes sorted:



Conclusion

The actual execution of the software went very well. It loads and sorts just as expected.

When I first implemented the sorting algorithm, the shapes were not sorted properly, I had to rework my algorithm and after a few tries, it worked.

I learnt how to make a software using Java's Swing class, and how to properly implement a design pattern. I also learnt how to efficiently make UML diagrams.

Top three recommendations:

- Javapoint's SWING tutorial: https://www.javatpoint.com/java-swing
- TutorialPoint's Design Pattern tutorial: https://www.tutorialspoint.com/design_pattern/index.htm
- Learning about different sorting techniques to find the most suitable one for this project: https://www.tutorialspoint.com/data_structures_algorithms/sorting_algorithms.htm