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.

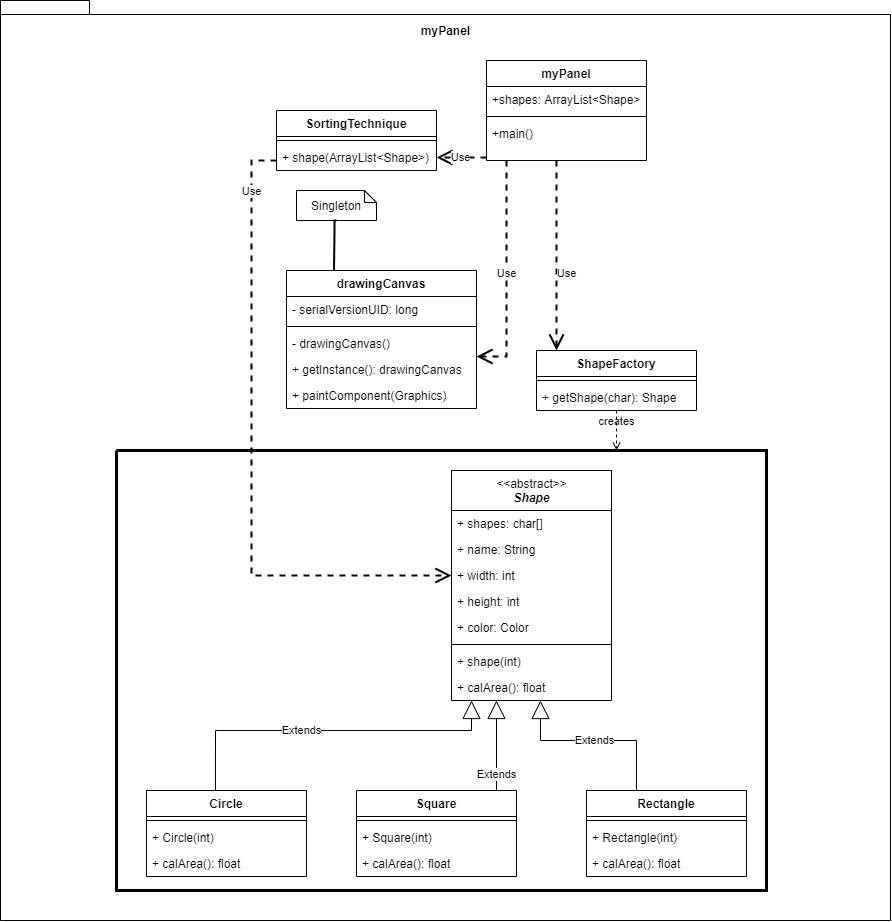
OOD principles I will use (further explained later):

* 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.

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.

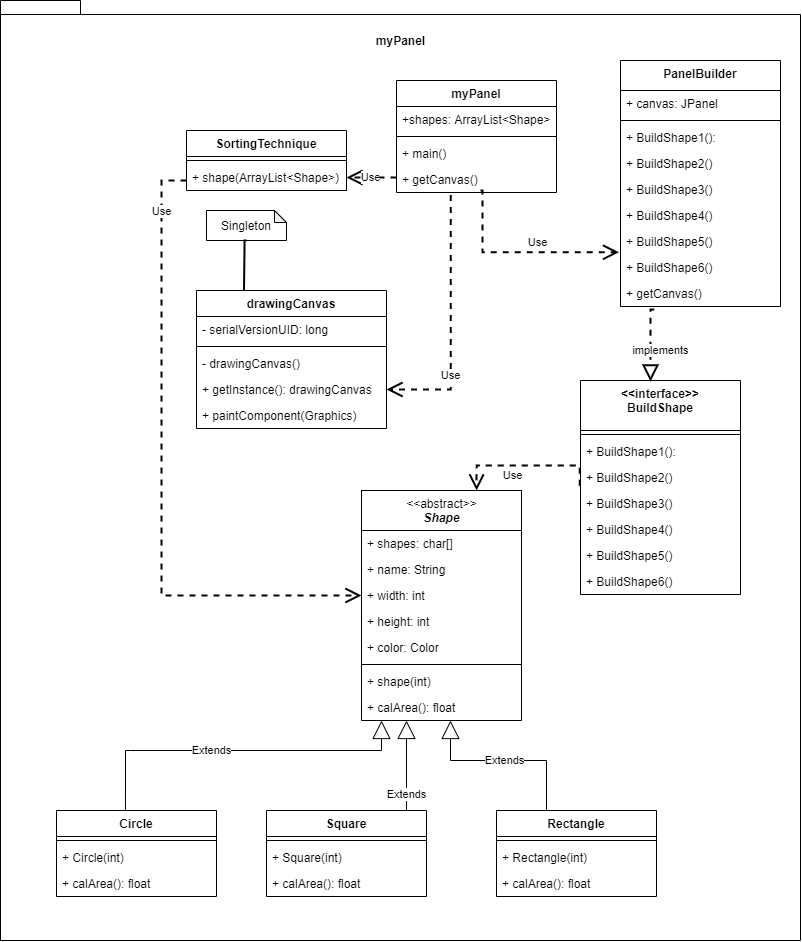
SortingTechnique class contains the methods that sorts the shapes.

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.

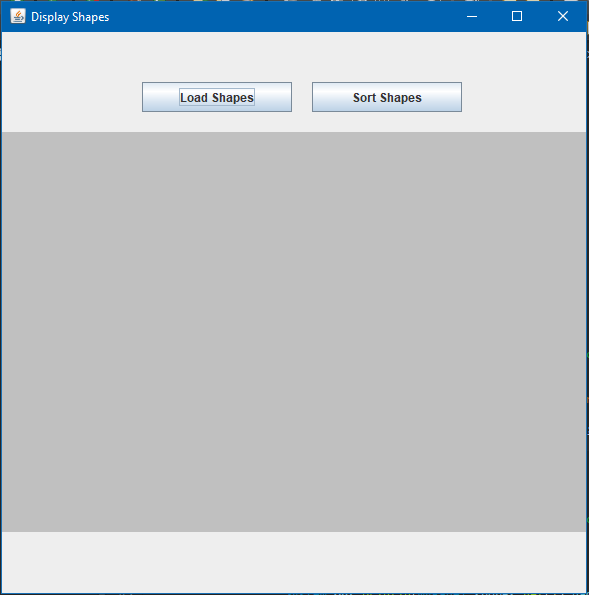
Both diagrams would produce the same result; however, the first diagram is a better design because:

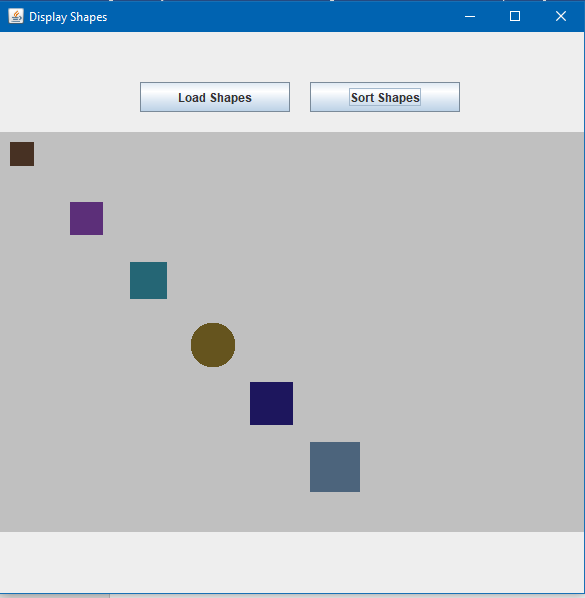
1. The implementation is easier
2. Uses less memory space as there are less classes and methods
3. More flexible compared to the second, because it easy to make changes if needed.

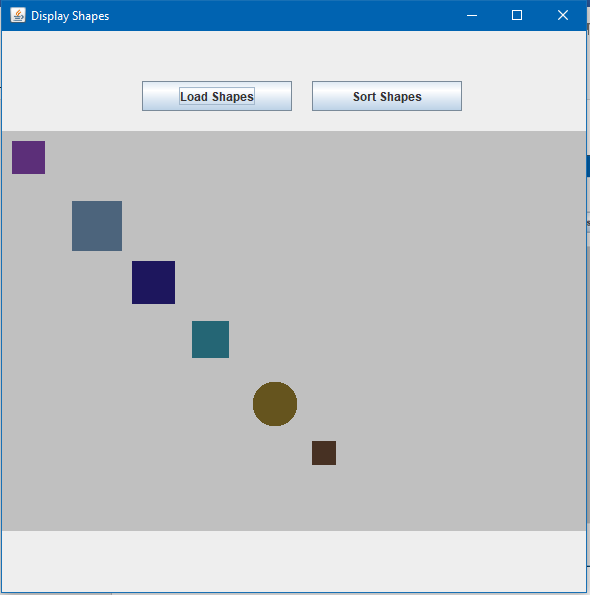
Implementation of the solution

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. 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 create 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>