EECS 3311 Section B

2021 Fall

TA: Naeiji Alireza

Project 1: Display shapes

Boho Kim

217303033

Part 1: Introduction

This project is making a simple interface applying OOP concepts and sorting algorithms using Java and Git-hub. Also, it gives experience about how to design and document it using the OOD concept for software design.

This software's goal is to make a simple interface that displays 6 shapes picked randomly among rectangle, square, and circle. If a user wants to sort it based on their surface, software sorts them based on their surface in ascending order and displays it in the interface.

At the first, it was a challenge to construct classes. Personally, constructing classes that are inside such as Shape, ShapeFactory was not difficult but classes to instantiate interface such as panel, buttons for the user was hard to construct for me because I had to connect it to the inner class to make new shape objects and use it. Also, since I have to document the whole project not just making a program, I had to consider OOD concepts and design patterns while I make it. It was a challenge for me.

This project will use OOD principles such as abstraction, encapsulation, and inheritance. Abstraction will help to create shape class in a simple way. The shape will have properties such as upper x and y coordinate, color, and area. Encapsulation will be used to hide information. It will be applied by public getter, setter, constructor but attributes will be hidden from the user. Inheritance will construct shape hierarchy. The shape will have Rectangle, Circle classes as children class and Square class will be inherited from the Rectangle class.

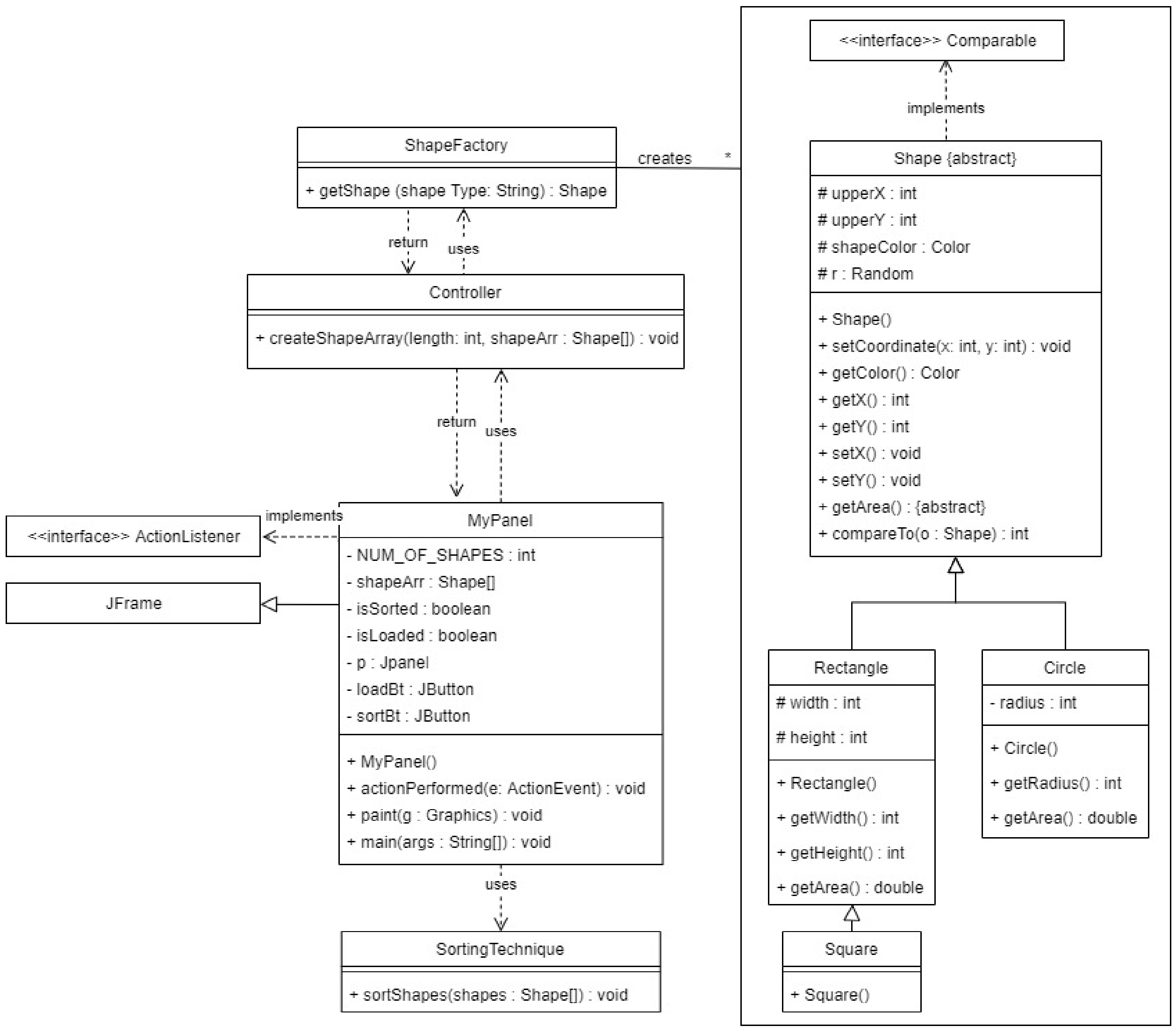
For architectural design, it will use an MVC pattern. The controller class will interact between the user interface and inner classes such as Rectangle, Square, and Circle. The view is MyPanel class to make windows and buttons that users will use and the Model is other classes that are associated to make shape objects and sort them properly.

For detailed design, it will use Factory pattern which is a creational design pattern. It makes it easy to make 6 shape objects, randomly.

This is the introduction of this report. The next part is the Design part. In this section, design concepts for this project will be represented by class diagrams using OO design principles. To compare it to other design concepts, a design alternative will be proposed. The third part will explain implementation in detail. Algorithm to sort shapes, tools that I have used for the system, the result of execution (snapshot, video) will be described. For the last, as a conclusion, I will explain what was well and wrong, what I learned, and my recommendation for the software project.

Part 2: Design

Class diagram

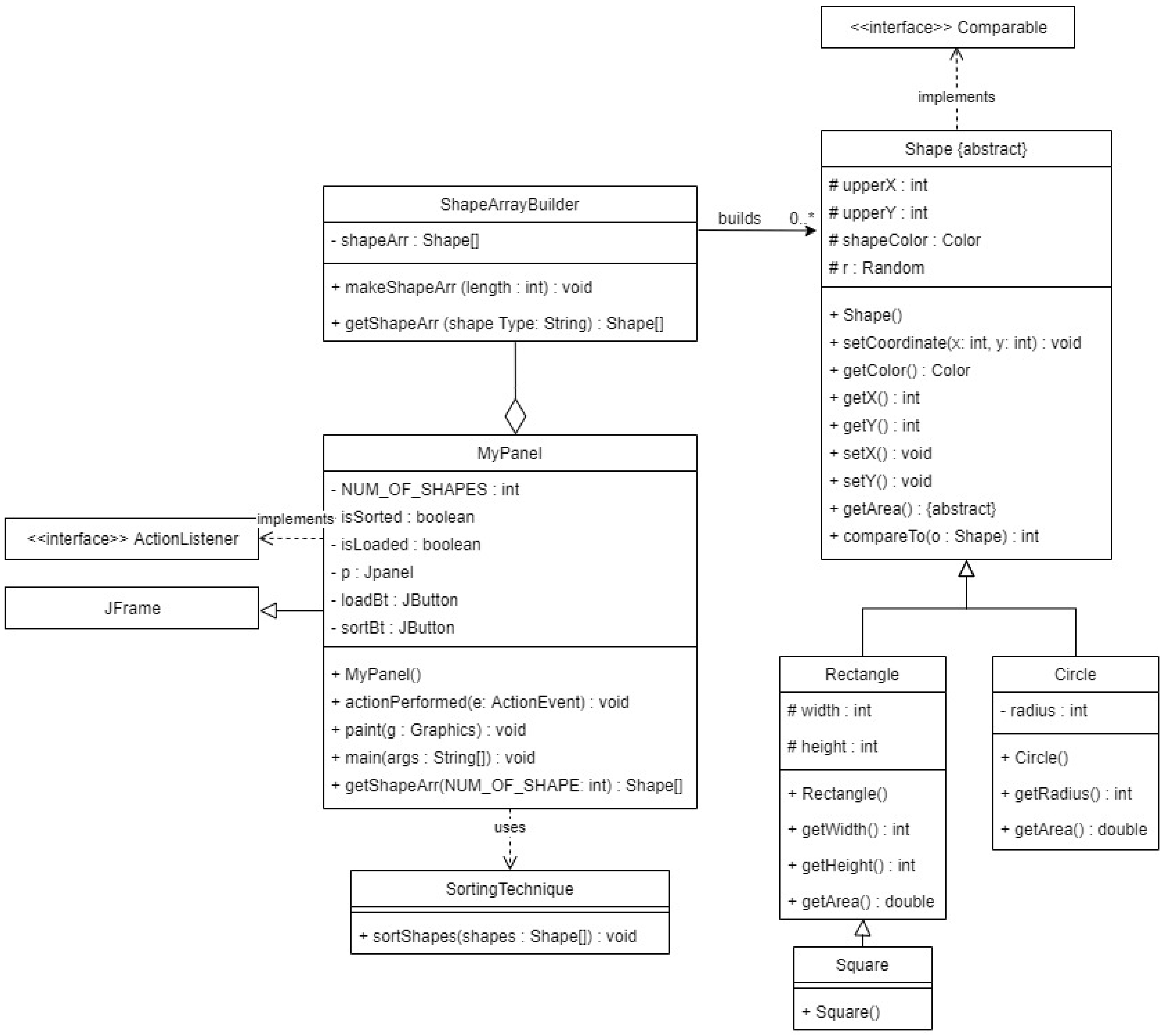


The system used a Factory pattern to create random shape objects, easily. Also, it used the MVC pattern to connect interface and inner classes that create shape objects.

The system starts at the main method in MyPanel that is responsible for to interface. MyPanel object is created in the main method and it makes shape array using Controller class. The Controller class creates several shape objects using ShapeFactory. ShapeFactory instantiates shape objects and returns them to Controller. The returned objects by ShapeFactory are also returned to MyPanel class by Controller class. These are displayed in the interface by MyPanel. If the user wants to sort it (if the user clicks a button to sort), the class sorts it using the SortingTechnique utility class and displays it in the interface. After sorting shapes, MyPanel displays sorted shapes in the interface and the system ends.

First, for shape hierarchy, I used inheritance to make classes simple. Rectangle and Circle classes don't need to set upper x and y coordinates in their classes and just define gerArea() method because shapes have a different calculation to get area. Since a square has the same way to get an area with a rectangle, it just set width and height in the class. To express shapes simply for the system, I used abstraction. The shape is expressed by the upper x and y coordinate, color, and their width, height, or radius. Also, using public getter, setter, and private or protected attributes, I used the encapsulation concept.

<Proposed design alternative>

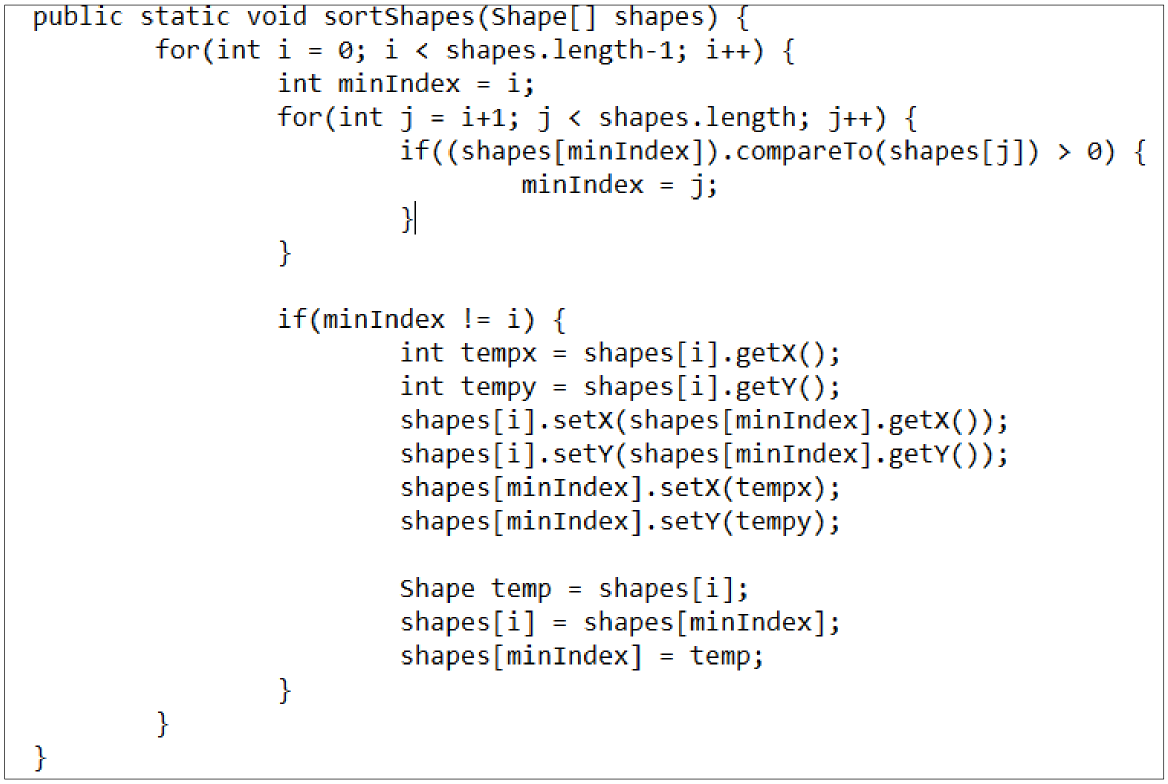


The proposed second class diagram used the Builder pattern instead of the Factory pattern. It can build more complex products using the Builder pattern but the shape object is not much complex in the system. So, I chose to use the first diagram.

But if the shape was complex such that it can not be made by a method requiring several methods, the Builder pattern would be more appropriate.

Part 3: Implementation

To sort shapes by their area in ascending order, I used the selection sort algorithm.



```
public static void sortShapes(Shape[] shapes) {
        for(int i = 0; i < shapes.length-1; i++) {
                int minIndex = i;
                for(int j = i+1; j < shapes.length; j++) {
                        if((shapes[minIndex]).compareTo(shapes[j]) > 0) {
                                minIndex = j;
                        }
                }

                if(minIndex != i) {
                        int tempx = shapes[i].getX();
                        int tempy = shapes[i].getY();
                        shapes[i].setX(shapes[minIndex].getX());
                        shapes[i].setY(shapes[minIndex].getY());
                        shapes[minIndex].setX(tempx);
                        shapes[minIndex].setY(tempy);

                        Shape temp = shapes[i];
                        shapes[i] = shapes[minIndex];
                        shapes[minIndex] = temp;
                }
        }
}
```

Start from the first index and set it minimum index for current.

Compare it to the next element. If the next element is smaller than the element in the current minimum index, set the minimum index to a new index whose element is smaller.

Repeat until the last element is compared. Then the element of minimum index is smallest among checked elements by a loop.

If the minimum index is not the first current minimum index, it means that the minimum element is not in the first index that we started to check. So, we should switch them.

Using getter and setter, switch shapes and upper x,y coordinate.

Also, repeat it until the last 2 elements are compared(index of array length-2 and index of array length-1)

I used the first diagram in part 2. First, I made Shape hierarchy using inheritance. Also, to compare it, it implemented a Comparable interface. Since the object that I should create was not complex, I used a simple Factory pattern to create objects. To connect the Factory class and MyPanel class that is responsible for the user interface, I used the middle class that is Controller. The Controller just makes ShapeArray and delivers it to MyPanel. Its whole class doesn't need to be created. So, it is a utility class such as SortingTechinique. There is its shape array object in MyPanel class. Therefore, it saves the object to its attribute. Buttons in MyPanel are regulated by boolean index to control repainting the background of the interface. When the system wants to sort shapes, MyPanel uses SortingTechnique utility class. In this process, the array is not replaced. It just switches objects in the array and their upper x and y coordinate.

Tools that I have used: Eclipse IDE 2020-12 (4.18.0) with JDK 15

Short video to run the system: https://drive.google.com/file/d/1EmR7dAsPGQ72YYZCKhG-4rmGohS9qCKK/view?usp=sharing

Snapshots

1. Go to MyPanel class and execute the main method.
2. Then, you can see this interface. (Figuare 1)

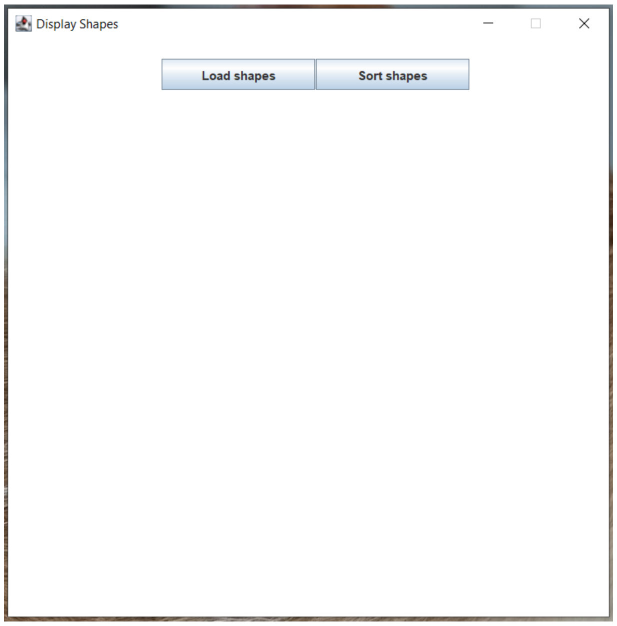


Figure 1

3. If you click the Load shapes button, it will load 6 shapes randomly.(Figure 2-1) If you click Sort shapes without loading shapes, warning message pops up. You need to load first. (Figure 2-2)

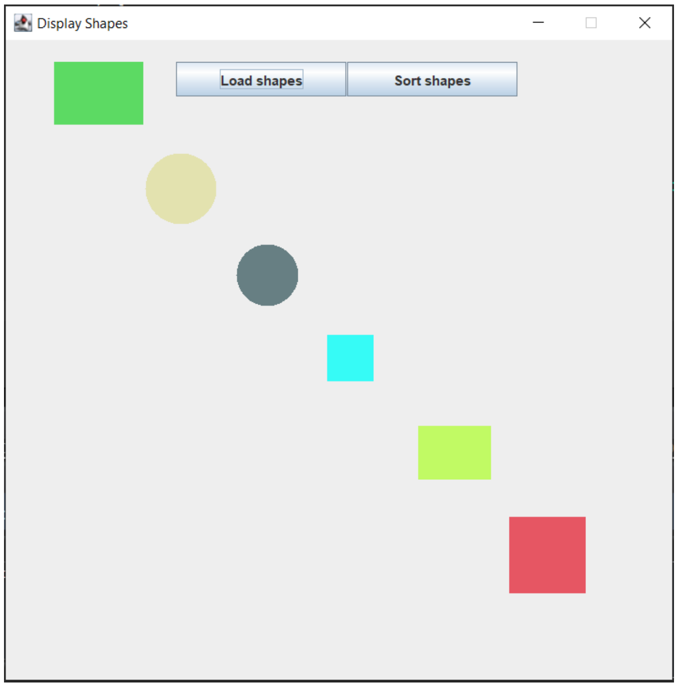
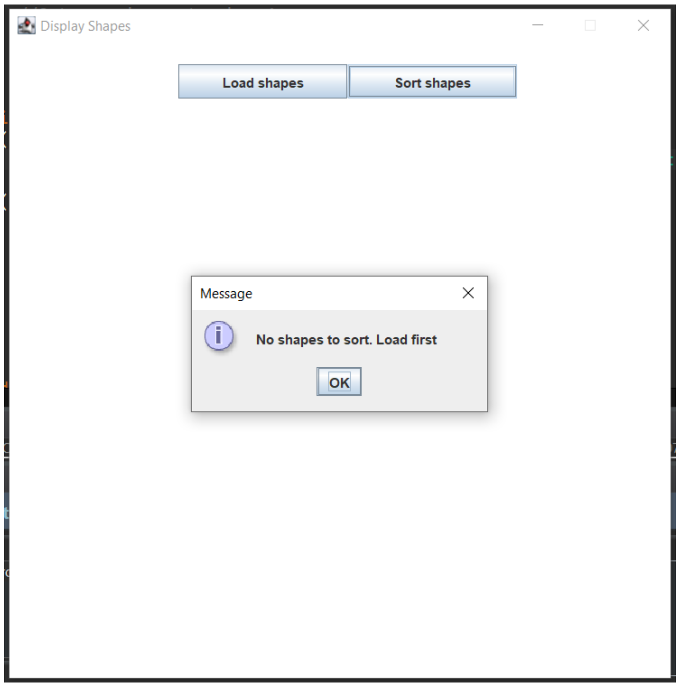


Figure 2-1



No shapes to sort. Load first

Figure 2-2

4. If you want to change shapes, you can load over and over until it’s sorted using Load shapes button. But, after sorting shapes, the system ends. (Buttons are not enabled) (Figure 3)

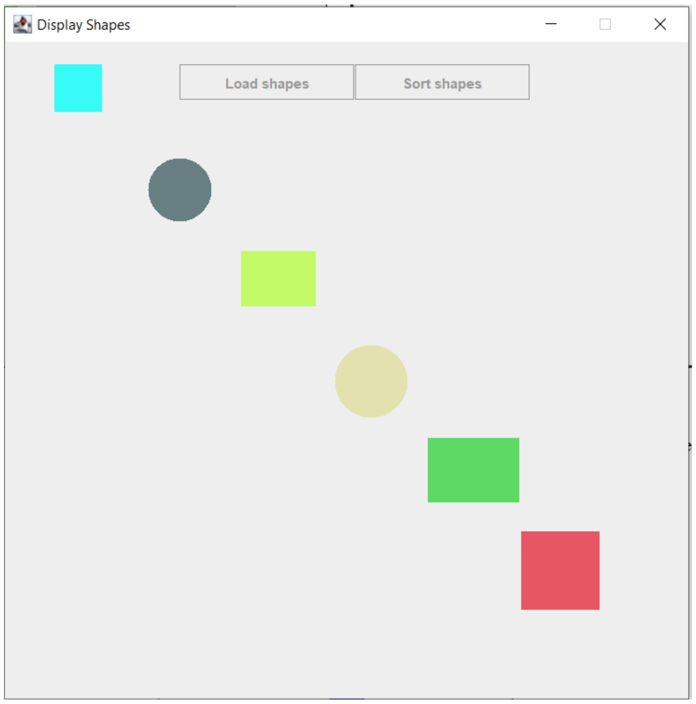


Figure 3

Part 4: Conclusion.

Constructing classes and implementation was not hard because it is a simple project. However, documenting such as drawing class diagrams, explaining implementation or sorting methods was the first time while making a program and it was quite difficult for me. During the project, I realized that if the project is huge and should be processed by a team, these software design processes would be necessary to understand each other's code and concept for the project. Also, before this project, I thought that the most important part is the implementation part but I think that construction of the first structure is hugely important too because it should be well-structured to make implementation followed it. I didn't know it because I usually just write code without documenting it.

So, I personally recommend that

1. Construct the first structure well spending a lot of time.

2. Have knowledge of useful design concepts to apply proper design for the system

3. Familiar with documenting tools such as diagram tool or MS Word.