EECS 3311B

First software project for week3

Xinger Yao

215106578

Lassonde School of Engineering

York University

Toronto, ON, Canada

xingeryao8@foxmail.com

Part1: Introduction:

This is software project is about to creating a shape-displaying and sorting application, The goal is to create a frame and set two buttons (LoadShapes and SortShapes) at the top, when click LoadShapes button, six shapes will be displayed, the shapes are including squares, circles and rectangles. When clicking the SortShapes, the six shapes will be re-ordered by their surface area. The structure of this report will be followed steps of the lab requirements.

It is my first time to write java codes with OOD principles and design patterns, instead of typing code randomly, it took a long time to think and organize the relationships of each class to make sure the whole project could satisfy the requirements.

Some Concepts in my project:

OOD:

The project is based on the Swing development mode, that the program is bound to be an Object-oriented, because all the components are object-based. Such as, the *init*() method from *MyFrame*, it is to initialize the corresponding control first, which is based on object-oriented, if a panel is needed, just "new" a panel, if a button is needed then just "new" a button, and set the button into the matched panel. Furthermore, in *paintComponent*() from *MyPanel*, the application can draw shapes by "new" shapes. The overall development idea is grab and use, so the programmer does not need consider too much about creating these classes.

Design patterns:

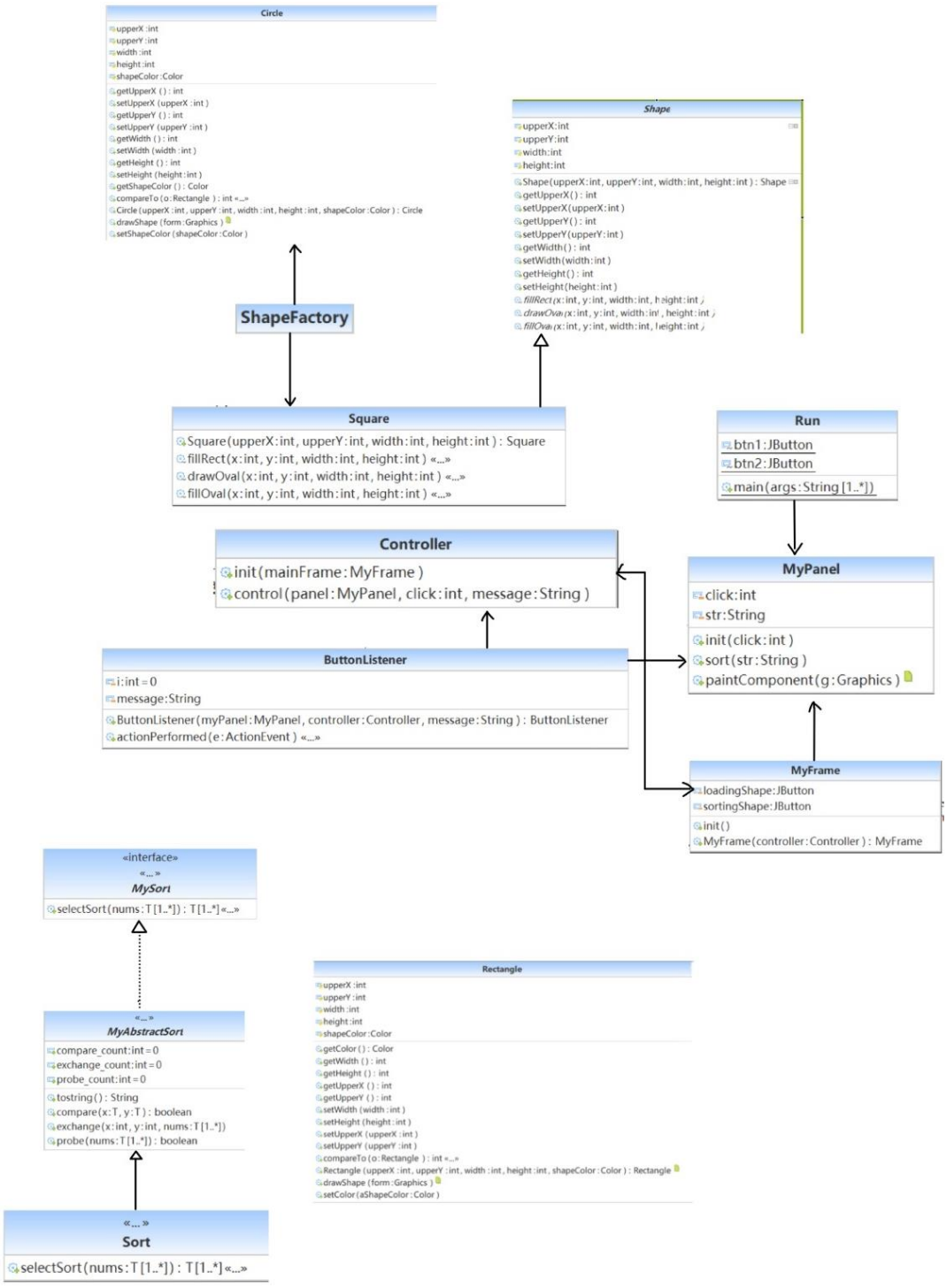
Factory pattern: corresponding to *ShapeFactory*, this Factory class creates and controls the objects of circles, squares and rectangles. In Factory pattern, an abstract *ShapeFactory* class can poses serval specific product classes and they have the same parent class.

Decorator Pattern: corresponding to *MyFrame* that inherited from *Jframe*, which is encapsulated and extended at the bottom to dynamically attach responsibilities to objects. To

extend functionality, Decorator provides more flexible alternatives than inheritance.

Part2: Design

UML:



During the development of this project, the object-oriented method is adopted for development. Swing based component development is a typical object-oriented development method. Everything is a component. A button, frame or panel can be created by "new" directly. When designing the whole framework, the later extension should be considered, Therefore, I separate the window rendering startup from the corresponding window drawing and interface construction functions, then execute the corresponding main method in Run, which can isolate itself with others. The class will only focus on the operation of the project.

The Design principles that used in the project:

1. Inheritance:

In the initial stage of the project, many reusable classes will be involved. All base classes need to have many basic functions at the beginning to facilitate the inheritance of the later subclasses and copy methods from the basic classes. The *MyFrame* class I designed inherits the *JFrame* class, which actually inherits the components of the system in the Swing framework, Similarly, *MyPanel* inherits panel components, which are inherited to initialize windows forms. After initialization, the corresponding button and corresponding shapes can be placed. In addition, the corresponding *Circle* and *Rectangle* classes also implement the corresponding *comparable* interface, the *Buttonlistener* implements its corresponding *Actionlistener* interface which is mainly used to monitor button events and copy the method of button events, so that clicking button can load the corresponding shapes.

2. Polymorphism:

There are also many polymorphisms designed in this project. In fact, the key is that the reference parent class points to the subclass' objects. For example, when *paintComponent*() draws the corresponding shapes, it needs to call the corresponding methods. To get the corresponding *Graphics* class, it is necessary to receive the *Graphics* subclass passed by the corresponding *Graphics*.

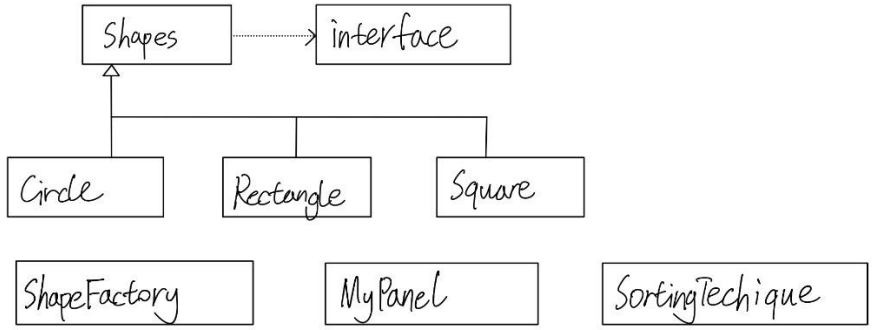
3. Encapsulation:

The encapsulation is used everywhere in this project, I designed the *init* () method in *MyFrame*, which is to initialize the corresponding control information first. There is no need to pay attention to how these controls are generated, but the focus on those objects that created by "new" is necessary and then insert them into the corresponding panels, which also facilitates my later logic of the code.

4. Abstract:

The abstract is mostly used in the abstraction of the components such as *Circle*, *Rectangle*, *Square*, *Button*, *Panel* and *Frame*. The functions and properties of these components and their corresponding methods are encapsulated into class. When I need to use the corresponding class, I can directly "new" them, and the objects from "new" have the methods and properties of the corresponding class. The examples are the shapes-creation from *MyPanel* class.

Alternate UML:



In this UML, I designed to put the button control and main method in to *MyPanel*, all shape classes are extended from *Shapes* and can be reached by call interface. This UML will have a clearer structure than the first one, but loss the flexibility of the post code modification.

Part3: Implementation

Sorting techniques:

In this project I tired two different sorting techniques, both of them are worked but I finally choose to use bubble sorting. The bubble sorting is to compare two adjacent surfaces. If the first is larger than the second, swap them both. Do the same for each pair of adjacent surfaces, from the first pair at the beginning to the last pair at the end. At this point, the last surface should be the largest number. Repeat the above steps for all surfaces, except the last one. Keep repeating the above steps for fewer and fewer surfaces at a time until there is no pair of numbers to compare.

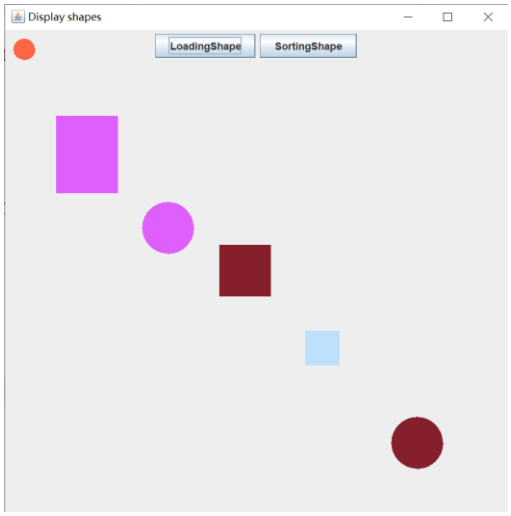
implementation and compile in the first UML:

In the design of the project is mainly the corresponding components of the inheritance and the use of components, the design of the corresponding *MyFrame* inherits *JFrame* class, *MPanel* inherits *JPanel*, this is the inheritance relationship, that is "is a" relationship, class inheritance abstract class, class inheritance parent class are both belong to this relationship. On the other hand, in the design of the corresponding *Rectangle*, *Circle*, *Square* the composition is used, these classes contain the corresponding *Color* class, so that when they call more than one class they will have the corresponding *Color* properties. Their life cycles are the same, if the whole does not exist, the part will also die out.

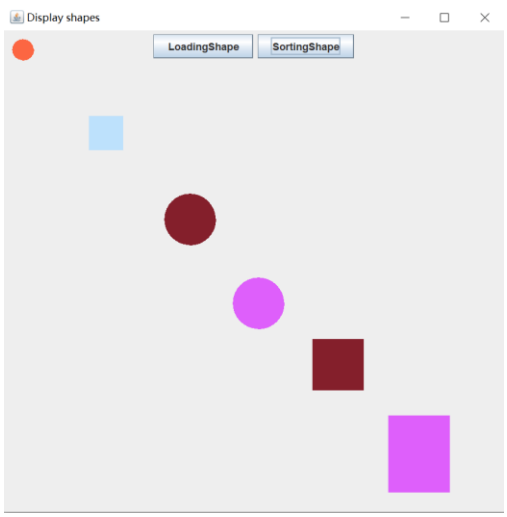
The above is the production of the basic components, then is to draw the overall framework and layouts. First of all, setting the corresponding windows name and size in *MyFrame*, then using *Button* to set buttons and name them, after that, adding the corresponding *ButtonListener* event to monitor the click. The goal is to click on the button to load shapes inside the panel, the panel will have to rewrite the method of loading them, the *Graphics* is used to draw out the shapes and re-color them. The last step is to sort shapes with bubble sorting, monitor the click of the button to sort the shapes by their surfaces.

The code is written with Eclipse 2021-09, version of JDK is JDK-13.0.1.

Code execution:



Loadingshape click



SortingShape click

Part4: Conclusion:

I tried two different sorting algorithms and I write them correctly in one time. During the Project, I found that I don't have a deep understanding of some basic concepts in Java and I spent plenty of time on reviewing it, meanwhile, the idea of Object-Oriented Design and Design patterns confused me in the process of programming but fortunately my codes executes the result at last.

This project gave me a new thought on how to write java projects which is how to think about the logic as a professional programmer, it also reminds me that the importance of the basic Java concepts and I need to practice more to understand them.

Three recommendations of mine:

1. Read books. Best way to learn new concepts outside the lecture.
2. Communications. Communicate with other people and share thoughts with each other.
3. Have a good understanding of the Java basic. Or it will really delay the process.