Domenico Calautti 216 374 530
EECS3311 Section B Sorting Shapes TA: Naeiji Alireza

Introduction:

The goal of this software project is to create a shape generating and sorting program. The program will have two available buttons with different functions. The “load” button will be used to randomly generate 6 random shapes with random dimensions within a hardcoded min and max value. The shapes that are possible for the program to generate are circles, squares, and rectangles. The “Sort” button will be used to sort the list of random shapes once generated from least to greatest surface area. The sorting will be done by using the comparable interface inside the shape class, and an insertion sort algorithm. Once sorted, the shapes will then be displayed on the screen in the sorted order.

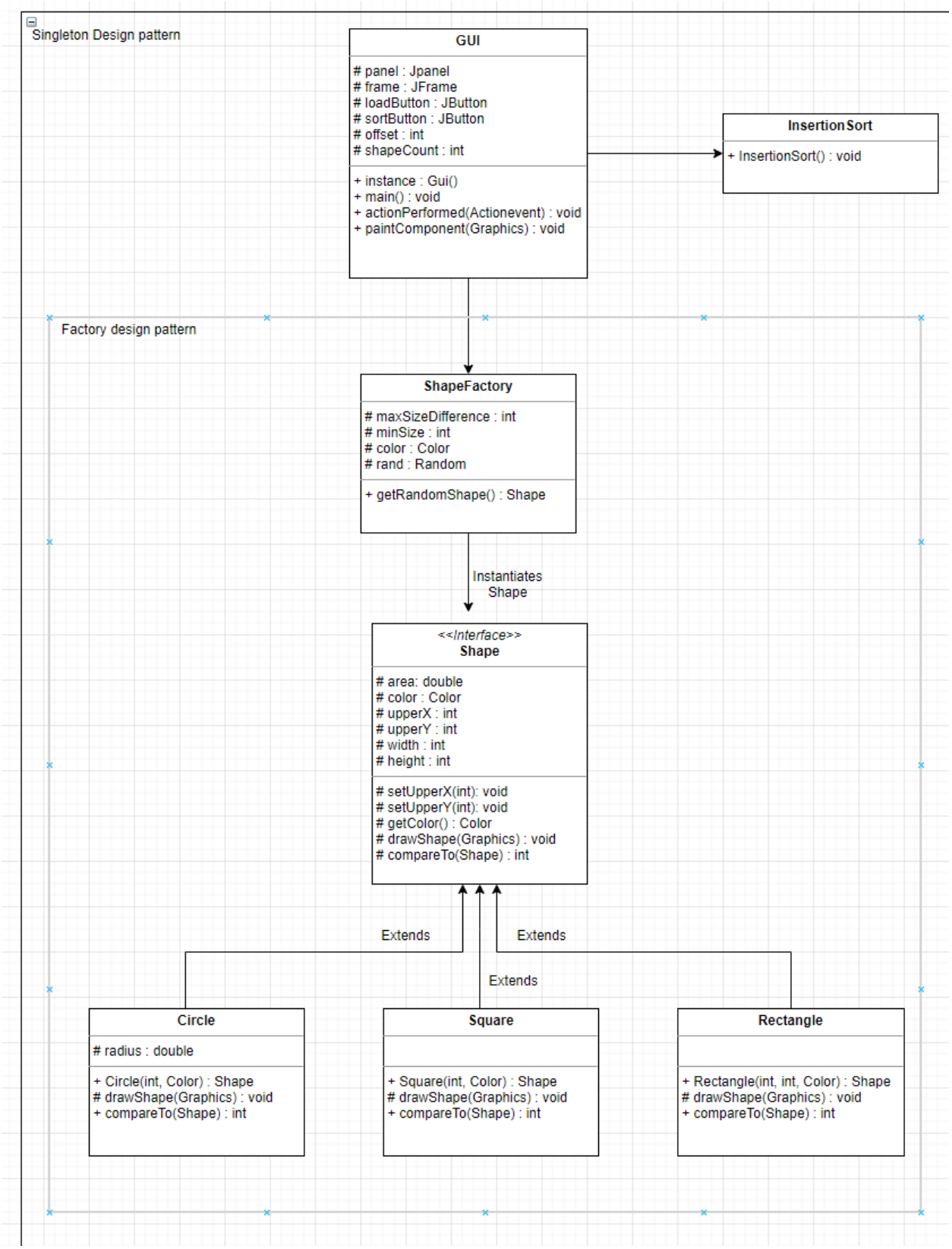
The possible challenges associated with this software project will be learning the Java swing framework, as well as implementing the sorting algorithm. I have never previously designed a GUI, and there are many abstracted concepts that I will have to understand before being able to comfortably use the swing framework. Such as the ideas of frames, panels, painting graphics, action event listeners, etc. The sorting algorithm should be easy enough to implement because I have studied algorithms in my past, but I have only used sorting algorithms to sort integers, rather than other objects.

The object-oriented design principles that I will be using are using a singleton for the GUI, and a shape factory to create the shapes. The shape factory pattern design will be used to randomly generate shapes by creating instantiations of the classes: circle, square, rectangle; using randomly generated shape, color, width/height values. These shapes will be the children of the parent class “Shape” which will hold all mutual data types, methods, as well as abstract methods meant to be inherited and overwritten by its children. I will also implement encapsulation techniques in order to unnecessary data leakage.

The structure of my report will be in a way that is appropriate to the particular instructions given in the lab manual. In the next section of the report, I will be explaining further the design of the program, including two UML class diagrams. Next, I will be explaining my sorting technique, as well as the tools I used. The last section of my report will be the conclusion where I summarize what I have learned, the outcome, and the difficulties that I have had producing the program.

Domenico Calautti 216 374 530
EECS3311 Section B Sorting Shapes TA: Naeiji Alireza

Section 2:



Domenico Calautti 216 374 530
EECS3311 Section B Sorting Shapes TA: Naeiji Alireza

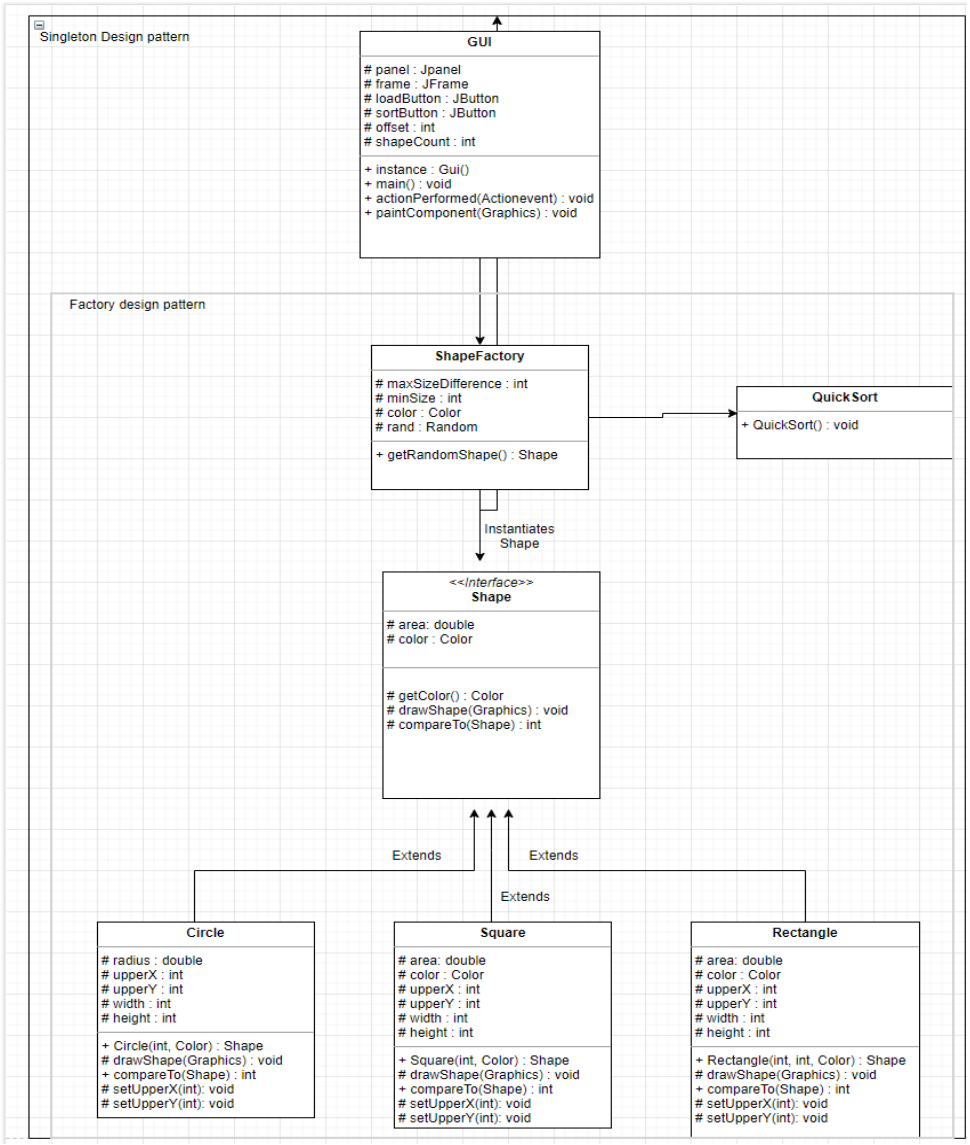
Above is the UML class diagram that I designed to model my software project. The shape factory class will handle generating instantiations of random shapes, with random dimensions, and colors. The shape generating segment of the software follows the factory design pattern. The Gui I organized in a singleton design pattern where the GUI calls an instance of itself and supporting classes such as the sorting algorithm and shape factory are called directly from the GUI class. The GUI class simply displays the shapes given by the shape factory class, both before and after they are sorted by the insertionSort class, as well as handling events followed by button clicks on the GUI. My software design uses all principles of OOD: Polymorphism, inheritance, abstraction, and encapsulation, which I will explain more in-depth next in the report.

The encapsulation and inheritance factors that I used were putting all shared variables within the shape parent class, while only letting the subclasses: rectangle, circle, square, have variables that are unique to that class. In this project only circle had a variable that was unique (radius), therefore the other child classes of Shape did not have any unique variables. Color, x, y, width, and height are all stated in the Shape class and inherited by all its children. The shape class also had abstract methods meant to be overwritten by its child classes. These classes inherited include getter and setter methods from the Shape class.

The abstraction factors in this software project can be observed within the java swing framework, as many of the concepts such as panel, frame, paint, etc, are all abstract in order to be readable and intuitive. Although this is easier than understanding the logic behind the classes, it requires an understanding of how to implement such as class and its uses. The abstraction that I have provided, are the shapefactory, Shape, Rectangle, Circle, and Square classes. Where the purpose of the classes is to be intuitively named and used and to instantiate randomly generated shapes.

The Polymorphism factors that I included in the software design are by designing the classes based on an IS-A, or Has-A, basis. Meaning that a rectangle, circle, square IS-A Shape, therefore they will inherit all of the Shape class. The subclasses then can overwrite/overload the inherited methods, and provide a unique way of usage for the specific subclass. The job of the Shape class is to be an interface at which is the building block of its subclasses, rather than being an object that can be instantiated. The only objects that should be instantiated are the subclasses of Shape, not Shape itself.

Domenico Calautti 216 374 530
EECS3311 Section B Sorting Shapes TA: Naeiji Alireza



Above I have provided my alternative UML class diagram. The difference between this diagram and the previous diagram is that I chose to let my shape subclasses (rectangle, square,circle) have its own variables rather than inheriting the shared variables from shape. This is not ideal, as all shapes share these variables and there is no value in having them in each distinct subclass. Another noticeable difference is that I moved the sorting method inside the factory design pattern such that ShapeFactory calls to the sorting algorithm rather than the GUI calling to the sorting algorithm. This is a reasonable change but I prefer to keep the sorting method separate from shapefactory as I want to maintain the standard factory design pattern. Lastly, I replaced the insertion sort class with a quicksort class. This is of minimal benefit as the ArrayList to be sorted will always be 6 objects, so a minimal increase in efficiency is not important. Especially since the difficulty to implement quicksort is a significant amount higher than implementing insertion sort.

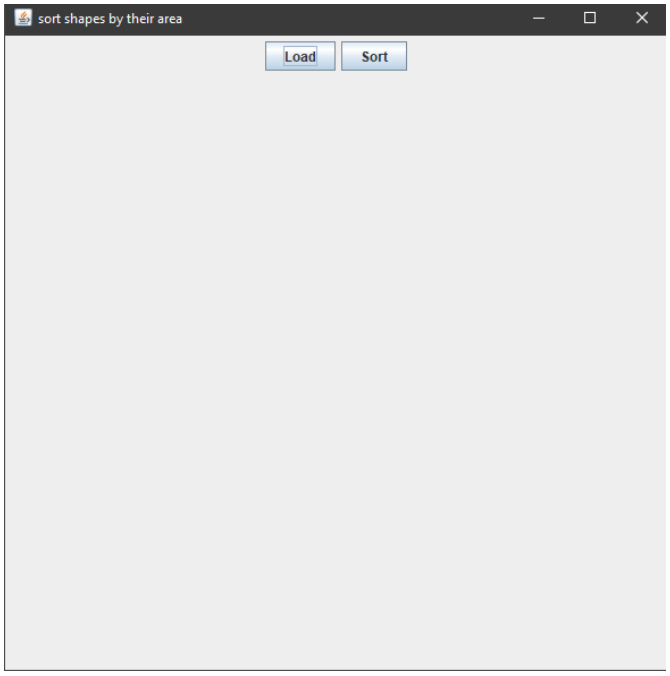
Domenico Calautti 216 374 530
EECS3311 Section B Sorting Shapes TA: Naeiji Alireza

Section 3:

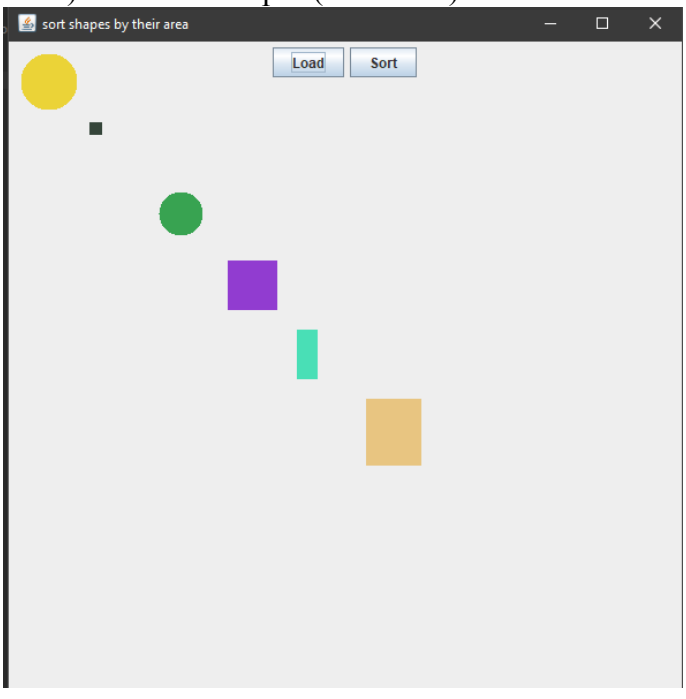
The sorting algorithm that I have decided to use was insertion sort. I created an Arraylist of shapes which I passed to the insertion sort function, which sorts the list. Insertion sort is an in-place sorting algorithm therefore its space complexity is low. It was also very easy to implement due to the simplicity of the algorithm.

The tools that I have used for the implementation were Eclipse version 4.21.0 as my IDE, using Java development kit version 1.8.0_251 to write and run my code. Along with GitHub, to upload share my program with my designated class TA.

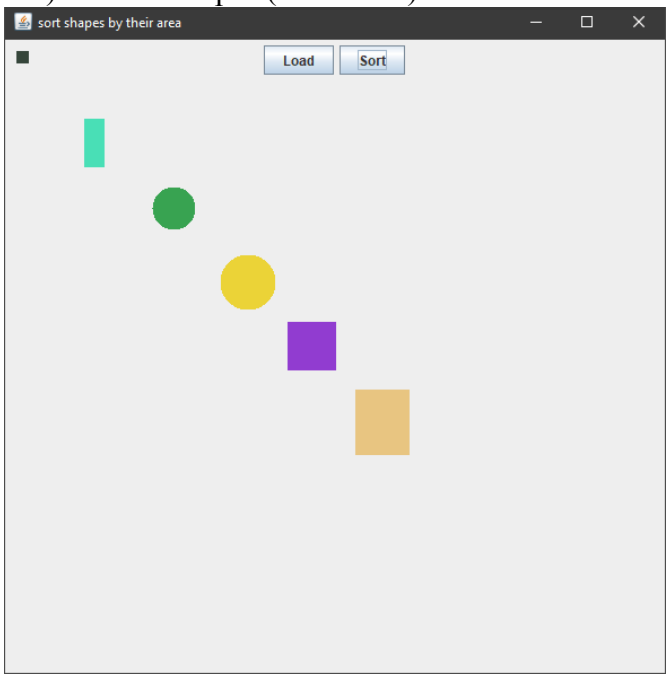
1) Launch the app



2) Load the shapes (click load)



3) Sort the shapes (Click Sort)



Domenico Calautti 216 374 530

Conclusion:

After completing this software project I learned a lot about both my capabilities and weaknesses. Some things that went well are the sorting algorithm design, as I feel comfortable implementing most algorithms. Other than that, creating the shape class and its subclasses was a basic concept that was also very easy to implement. Things that went wrong during the development were mostly java swing framework issues. It took a while to understand how to properly use JFrame, JPanel, painting, and repainting onto the screen. Sometimes it would display nothing, others it would paint the loaded shapes but not the sorted ones. From this software project, I have learned that it is a good practice to design and plan a software project instead of having an aimless approach. I also got more comfortable with the idea of using a GUI, since most of my past software projects have been through console logging of text. My top three recommendations to ease the completion of this software project are to get comfortable with the GUI first before anything, to take breaks when you are stuck as I find it helps me think differently, and to work on the project over a large period of time rather than all in a few days.