**EECS3311B – First Software Project 3-3-1 pages**

**Name: Andrew Ngov - 217227331**

**TA: Alireza Naeiji**

**PART I: Introduction**

**What this software is about and what is its goals?**

This software project is about creating an interface using Java swing that contains two buttons, one that loads six random shapes on the screen that can be either a rectangle, square or circle and the other button sorts the shapes based on their surface area and displays them on the screen in order of area. The goal of this software is to help me to implement programs that have object-oriented design principles and implement design patterns better.

**What are the challenges associated with the software project?**

The software project requires Java Swing which I have not used before, so I had to learn it as I wrote the code for the software. Implementing object-oriented design principles and design patterns are also new to me so it would be a challenge to implement.

**What are the concepts carried out in this software project?**

The software project specifies software objects(such as shapes that have attributes such as x, y, width, height etc.) and the way that they collaborate to satisfy requirement(behaviour such as the surfaceArea, getWidth, getHeight methods). It also contains other OOD principles such as encapsulation(shapes have private attributes with getter and setter methods) and inheritance(rectangle, square and circle all implement shape class). The software also uses a factory design pattern that uses a ShapeFactory to instantiate shape objects.

**How is this report going to be structured?**

For each section of the report(Part 1-4) it will be highlighted to separate each section of the report. Each section will contain multiple bolded questions to answer each of the questions for the report. For part II(Design of the solution), an image will be shown at the beginning follow by the explanation after. In part III and IV, it will be structured similarly to the introduction to answer each of the questions.

**PART II: Design of the solution**

**Create a first UML class diagram of your system (use at least two design patterns), add the corresponding figure in the report and comment its elements**

**A picture containing timeline

Description automatically generated**

This UML diagram and software contains two design patterns. A design pattern that is used in this UML class diagram is a creational pattern. The creational pattern that is used in this UML diagram is the Factory pattern. This pattern is used because we need to find the best way to instantiate objects of a sub-class whose parent class has several sub-classes. The factory design pattern allows us to instantiate objects without revealing the instantiation logic to the client. The responsibility of instantiating the class is assigned to the factory class rather than the client program. This allows us to create objects dynamically. In this UML diagram there is a shape interface with 3 classes that implement it(circle, rectangle and square). The factory class ShapeFactory instantiates the shape objects which are circles, rectangles and squares. In the software for the UML diagram the ShapeFactory method creates 6 random shapes that can be any of the 3 concrete classes(circle, rectangle and square) and also creates random widths and heights for the shapes. FactoryPatternDemo is the demo class that relies on(asks) ShapeFactory to instantiate the Shape objects. FactoryPatternDemo specifies information to ShapeFactory to get the instance of the required object(circles, rectangles and squares).

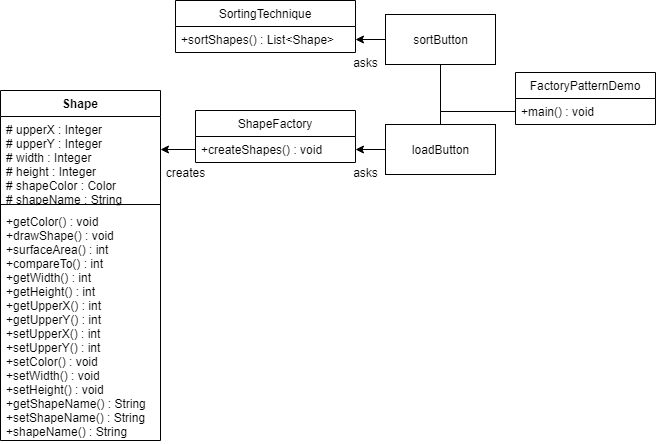
The other design pattern that is used in this UML diagram is a structural pattern. The structural pattern used is the Facade design pattern which connects a simplified interface to the overall functionality of a complex subsystem. Instead of using the subsystem objects directly, the façade is used which calls the subsystem objects.

**Use OO design principles in your class diagram (explain how you have used them: name the corresponding classes, interfaces, and if possible most relevant methods)**

Some OO design principles that I have used in this class diagram include encapsulation and abstraction. For abstraction, the program has a class Shape, which is an interface. An interface is abstract which hides the unnecessary details(internal implementation) of a class to other classes and only exposes methods that are relevant to interact with other classes. The interface(abstraction) only contains methods that needed to be used in other classes. For example, the main method needs to drawShape(), getColor() and getter and setters to create the shapes and to display it on the software. It also needs the compareTo method(which uses surfaceArea() to compare the area between squares, rectangles and circles) in the sorting algorithm to determine the areas to sort and setUpperX and setUpperY to update the shapes (squares, rectangles and circles) display to be properly sorted. The unnecessary methods that are not shown in the interface are not used by the other classes.

As explained above, an OO main design principle that was used was encapsulation. Encapsulation implies that the logic(state and methods) of an object is kept in a class, which means that all the concrete implementations of Shape(Circle, Rectangle and Square) all have protected modifiers on the variables. The class then has getters and setters (getWidth(), setWidth(), getUpperX(), setUpperX(), etc.) which can be used if the objects of the class need to interact with other classes.

**Propose a design alternative by creating a second UML class diagram. Does this second class diagram yield a better design than your first class diagram? Explain why.**

****

In this design alternative, rather than using a shape interface that implements concrete classes, the shape class is concrete. To differentiate the shapes, a new shapeName variable is added to store the name of the shape(either Circle, Rectangle or Square). The surfaceArea() would be changed to have 3 if statements with 3 different return statements to calculate the shapes depending on their shapeName. This second class diagram does not yield a better design than in my first class diagram because it does not implement OO design principles. It does not contain abstraction design principles and also does not contain any creational design patterns that the first class diagram did(factory design pattern).

**Part III: Implementation of the solution**

**Describe the algorithm of the sorting technique you have used to sort the shapes**

The algorithm of the sorting technique that I have used to sort the shapes is one of the algorithms that was recommended by the TA. The algorithm that I chose to implement was the selection sort method. Although this method has very slow runtime compared to other algorithms (O(n^2)) it was assumed that runtime does not matter, and this algorithm was chosen because it is very simple to implement. How this algorithm works is using two for loops. Inside the first for loop, it starts at i = 0, and increments all the way until the second last element of the array. Inside the second for loop, it starts where the first for loop is currently at with 1 added to it(j = i + 1). Inside the second for loop, it goes through all the elements of the array after i and compares the surface area of it to the array element of i. If the surface area is smaller, then the two elements of the array swap locations. What this does is find the smallest surface area from i to the end of the array and swaps it so that the i location in the array has the smallest surface area. The second loop then exits. For assigning the x and y of the shapes(to be shown on the interface), a x and y variable is set at 50, and set to array[i] which contains the smallest surface area in the current iteration. After that, x and y is incremented by 75 and the loop is repeated, with i being incremented by 1 and the loop assigning the second smallest surface area. The two loops repeat until the first loop is at the second last element of the array and stops after the last two elements are checked. After this happens, the array is sorted from smallest to largest surface area, and the x and y are set accordingly so it will display the shapes from smallest to largest. Essentially, the selection sort algorithm splits the array into two subarrays, one that is sorted and one that is unsorted, after each iteration of the outer loop, the smallest element from the unsorted array is added into the sorted array and it continues until all elements are sorted.

**Describe how you have implemented and compiled all the classes of your class diagram in java (specify if you have implemented the first or the second class diagram)**

The first class diagram was used to implement this software. First, I made a Shape interface because all the shapes in the software(Square, Rectangle and Circle) were going to use similar methods and so that I could put the Shapes into an array list to be sorted. After, I created concrete classes for the Square, Rectangle and Circle, which was like the sample code provided which included x, y, width, height, color, and setters and getter methods for the shapes. All the shapes implemented the interface Shape and Comparable<Shape> as the shapes needed to be compared to be sorted. All the shapes were pretty similar so they were all implemented at the same time(the only difference between these 3 shapes are the calculation of the areas). I also added a surfaceArea method for the three shapes as they all had different calculations for the area. The compareTo method of the implemented Comparable compares the shapes using surfaceArea() and returns the difference of the areas. The compareTo() and surfaceArea() will be further explained in SortingTechnique.

To instantiate the shapes on load shapes click, the ShapeFactory class is called. ShapeFactory has a for loop that iterates 6 times to create 6 random shapes of random sizes. For relative location on GUI, it is not randomized. The x and y both start at 50 and increment by 75 for each shape. The shape is randomized using random to generate a random number between 1 and 3, which then creates a rectangle if number is 1, square if 2 and circle if 3. The width and height is also a random number between 1 and 75 and all the colours(r, g, b) are random numbers from 1 to 255. The shape is then created and added to shapeList. The process repeats until all 6 shapes are created and the list is returned. The program then sets paintShapes to true and panel.repaint() is called which draws the shapes on the software.

To sort shapes, the sort shapes button is clicked which calls the sortShapes() method in SortingTechnique class. The shapes are compared by size using compareTo() and the shapes are sorted which is then repainted to show in sorted order.(Full algorithm explained in previous question).

**Specify the tools you have used during the implementation**

The tools that I have was the latest version of the Eclipse IDE. This was used in the implementation of the software and used to write and run the code. The version of JDK that was used while developing the program was 15.0.2, although newer versions should still be able to run this code.

**Take a snapshot of the execution of the code(i.e., of the interface and comment it in the report)**

Chart, scatter chart

Description automatically generated

This title of the program is named Display shapes and comes with two buttons. Load shapes loads the shapes that are randomized between circles, rectangles and squares and randomizes width and height. Sort shapes sorts the shapes and displays them from smallest to greatest surface area(as explained above).

**Part IV: Conclusion**

**What went well in the software project?**

What went well in the project was the sorting algorithm of the software. I have had other classes where I have had to do sorting algorithms before and creating the method to sort the shapes to display was a simple task to do and took the shortest time out of the whole software project. The TA suggested the selection sort method(that was used in my sorting technique) and after figuring out how to get the buttons to work to display the shapes and properly designing the classes and the JFrame, the sorting algorithm was easily completed and after getting a hang of how to use Java Swing, getting the sort button to work wasn’t too difficult of a task.

**What went wrong in the software project?**

What went wrong in the software project was getting the load button to correctly display the six random shapes that were created in the ShapeFactory method. For most of the project, I was not able to properly get the load shapes to correctly display the shapes. Either the shapes were already loaded in before even pressing the load shapes button, or the load shapes button did not work at all, causing no shapes to be shown on the board. Because of this, I had to redo my code multiple times and after a few attempts, I found the solution and it was way easier than I thought it was and it resulted in a lot of wasted time. After that, sorting shapes was easily done but getting it to work took a while because I forgot to change upperX and upperY when sorting(another simple mistake I made that took a long time to fix resulting in wasted time).

**What have you learned from this project?**

Even if there were a lot of new things, learning new things is a lot easier than it seems because there are a lot of sources online to help you if you’re unable to figure out the approach to a solution to a problem or need help figuring out how to code something.

**What are your top three recommendations to ease the completion of the software?**

Three recommendations I have is to view online for sources for any help you may need because there are many resources available, ask your TA’s for help if you’re stuck on any problem because they can guide you and the problem you have may have a simple fix, and lastly to also review your code thoroughly for any errors and to find fixes that could be very simple.