Software Project #1

Jordan Eldridge

216962870

EECS3311, Section B

Professor Alvine Boaye Belle

TA Kazi Mridul

10 October 2021

Introduction

For this project, we are tasked with developing a software program that is capable of generating six random shapes (circles, squares, and rectangles) onto an interface as well as sorting them based on their surface areas.

This project has been developed using JavaSE 14. The interface uses JComponent and its respective subclasses. This was the first challenge that had to be overcome, as I was not familiar at all with creating graphical user interfaces in Java, having only used the console before. After practicing and doing some research, I was able to overcome this obstacle.

Object-oriented design (OOD) practices and principles are employed throughout this project. Encapsulation is used throughout all the classes to manage the states of the shapes. The user cannot directly modify the states of the shapes. If modification or retrieval is needed, it is handled through public “get” and “set” functions. Each class of shape (i.e., Circle, Square, and Rectangle) extends the parent abstract class Shape. Each shape inherits the methods of the parent Shape class. The allows for easy reusability and expansion to include other shapes. All the information for each shape object is stored in the Shape class, which allows other shapes to be created simply by passing relevant information to the Shape class. Additionally, each shape has its own implementation of the abstract fill(Graphics2D g2d) and calculateArea() methods. This allows each shape to specify how to show itself on the screen to the user and what formula to use to calculate its area. This demonstrates the object-oriented principles of abstraction, inheritance, and polymorphism by allowing each shape to define itself in various ways.

Each section of this report will contain information pertaining to the design and implementation of this project, starting with a description of the design process with two different designs, followed by the implementation of one of the designs, and finally a conclusion with some final thoughts on this project.

Design of the Solution

In designing a solution for this problem, I have used the four-step object-oriented design and analysis (OOD&A) workflow to analyze and design a software system. The first step in this workflow is to define use cases. This is a very simple program with not much real-world use in the state that it is in, but it can be easily modified to perhaps have some useful applications. For instance, the program could be changed to allow the user to define shape parameters and the program will calculate and display the surface area. Multiple shape inputs could be sorted for the user as well. In its current state however, the use cases are very limited.

The next step is to define a domain model for the project. The diagram is not included in this report, but it has a similar appearance to the UML class diagram that is shown in Figure 1, which is a part of step 4 of the OOD&A workflow. I made a rough domain model diagram before starting. This helped me lay out my ideas and get a basic plan for this project's design.

Step 3 of the OOD&A workflow corresponds to assigning responsibilities to objects and designing interaction diagrams. This part of the workflow was largely excluded from my project as I was unfamiliar with the concept of interaction diagrams when I first started this project. I had a general idea in my head about the responsibilities of the different objects in the project which helped when it came to designing the full-fledged class diagram in the next step.

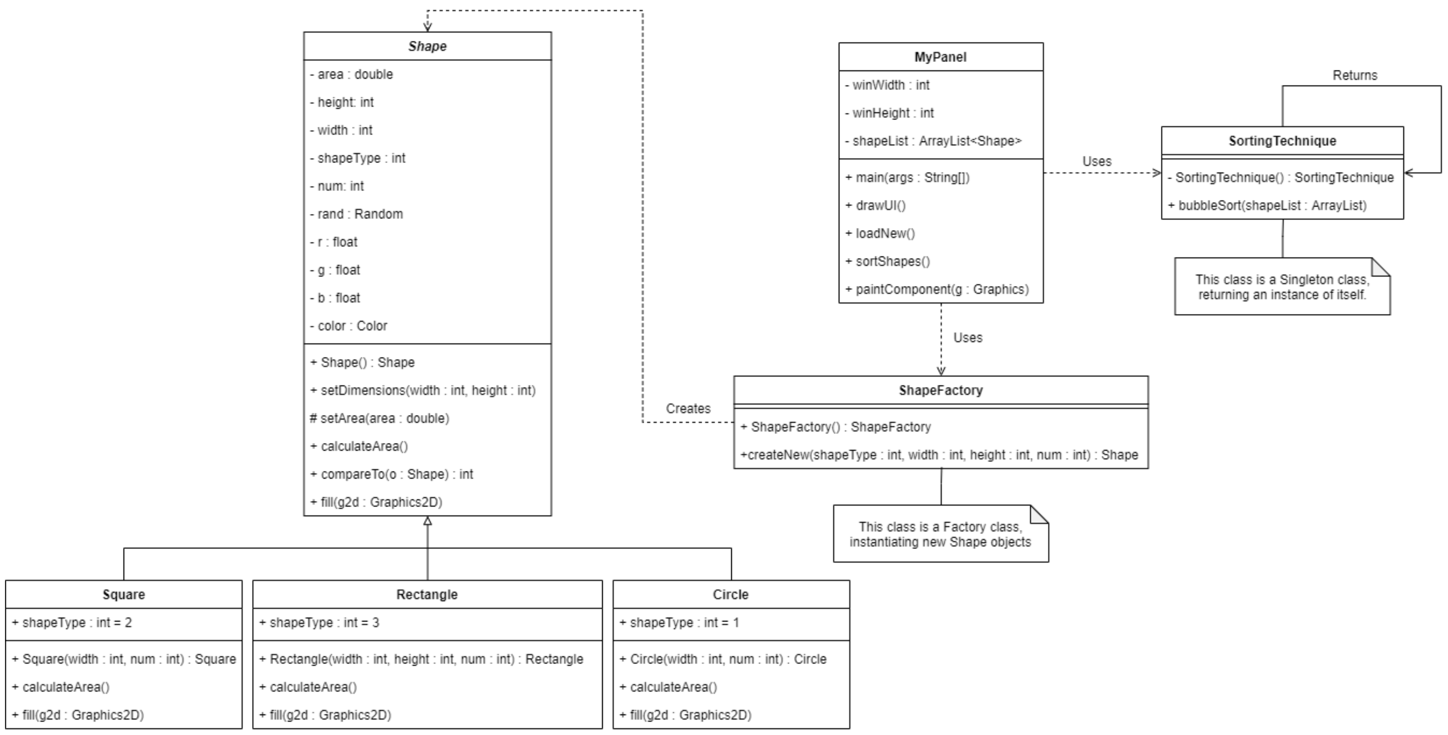
Shape
- area : double
- height: int
- width : int
- shapeType : int
- num: int
- rand : Random
- r : float
- g : float
- b : float
- color : Color
+ Shape() : Shape
+ setDimensions(width : int, height : int)
# setArea(area : double)
+ calculateArea()
+ compareTo(o : Shape) : int
+ fill(g2d : Graphics2D)
MyPanel
- winWidth : int
- winHeight : int
- shapeList : ArrayList<Shape>
+ main(args : String[])
+ drawUI()
+ loadNew()
+ sortShapes()
+ paintComponent(g : Graphics)
Uses
Returns
SortingTechnique
- SortingTechnique() : SortingTechnique
+ bubbleSort(shapeList : ArrayList)
This class is a Singleton class, returning an instance of itself.
Uses
Creates
ShapeFactory
+ ShapeFactory() : ShapeFactory
+createNew(shapeType : int, width : int, height : int, num : int) : Shape
This class is a Factory class, instantiating new Shape objects
Square
+ shapeType : int = 2
+ Square(width : int, num : int) : Square
+ calculateArea()
+ fill(g2d : Graphics2D)
Rectangle
+ shapeType : int = 3
+ Rectangle(width : int, height : int, num : int) : Rectangle
+ calculateArea()
+ fill(g2d : Graphics2D)
Circle
+ shapeType : int = 1
+ Circle(width : int, num : int) : Circle
+ calculateArea()
+ fill(g2d : Graphics2D)

Figure 1: UML Class Diagram for Design #1 (implemented)

Figure 1 shows a UML class diagram for the first design, which is one that was implemented. Designing the UML class diagram corresponds with step 4 of the object-oriented analysis and design workflow. The MyPanel class contains the programs main() method which drives the program. It contains the instructions for drawing the UI and creating a list of Shapes to add to. The ShapeFactory class is called from this class in the loadNew() method. The ShapeFactory class creates a new Shape object that is either a Circle, Square, or Rectangle. This is an example of the Factory design pattern. Shape is an abstract class and requires Circle, Square, and Rectangle to extend it and implement certain methods. They each contain their own implementations of the calculateArea() and fill() methods. The SortingTechnique class is also called by the MyPanel class when the method sortShapes() is called. SortingTechnique uses the Singleton design pattern and has a single method that is used to sort the list of shapes based on their surface areas using the bubble sort algorithm. The class diagram demonstrates object-oriented design principles such as encapsulation, abstraction, and inheritance.

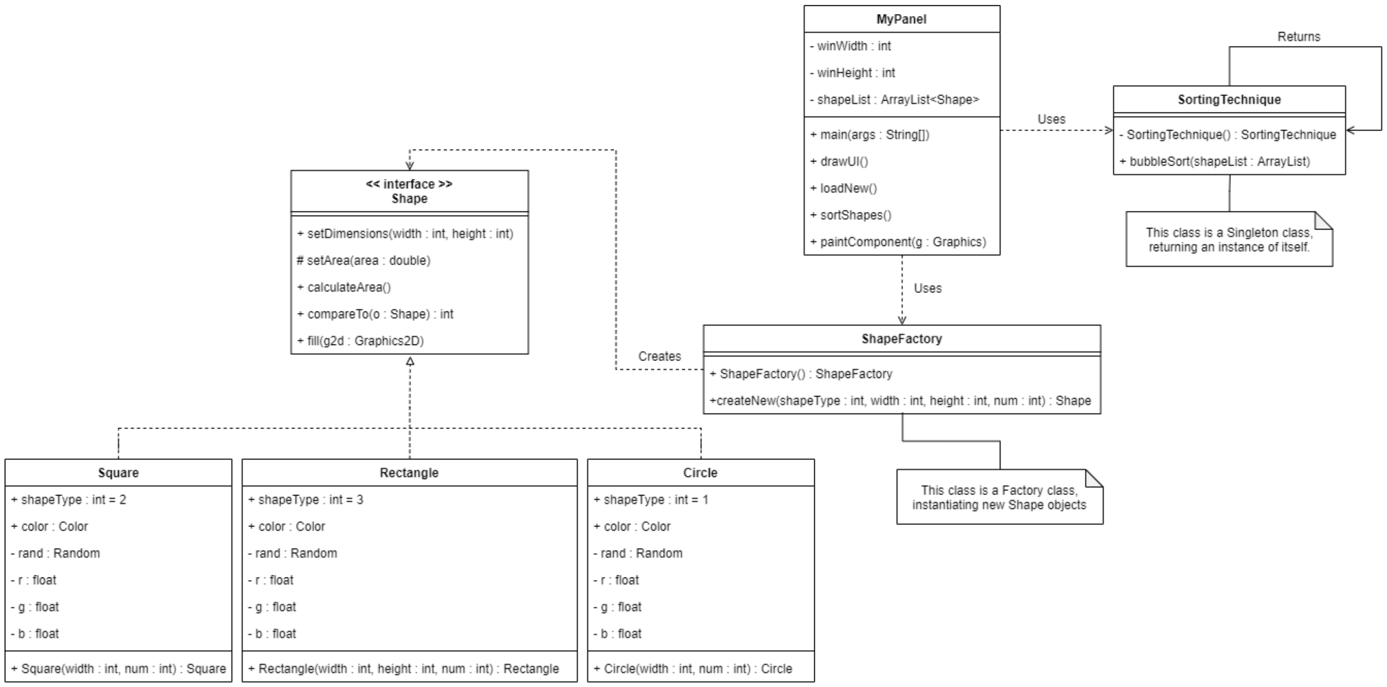
MyPanel
- winWidth : int
- winHeight : int
- shapeList : ArrayList<Shape>
+ main(args : String[])
+ drawUI()
+ loadNew()
+ sortShapes()
+ paintComponent(g : Graphics)
Uses
Returns
SortingTechnique
- SortingTechnique() : SortingTechnique
+ bubbleSort(shapeList : ArrayList)
This class is a Singleton class, returning an instance of itself.
<< interface >> Shape
+ setDimensions(width : int, height : int)
# setArea(area : double)
+ calculateArea()
+ compareTo(o : Shape) : int
+ fill(g2d : Graphics2D)
Uses
Creates
ShapeFactory
+ ShapeFactory() : ShapeFactory
+createNew(shapeType : int, width : int, height : int, num : int) : Shape
This class is a Factory class, instantiating new Shape objects
Square
+ shapeType : int = 2
+ color : Color
- rand : Random
- r : float
- g : float
- b : float
+ Square(width : int, num : int) : Square
Rectangle
+ shapeType : int = 3
+ color : Color
- rand : Random
- r : float
- g : float
- b : float
+ Rectangle(width : int, height : int, num : int) : Rectangle
Circle
+ shapeType : int = 1
+ color : Color
- rand : Random
- r : float
- g : float
- b : float
+ Circle(width : int, num : int) : Circle

Figure 2: UML Class Diagram for Design #2 (Not Implemented)

Figure 2 demonstrates a UML class diagram that models the same piece of software with a different design. The key difference between this design and the previous is that in this design, Shape is an interface instead of an abstract class. Square, Rectangle, and Circle implement the Shape interface instead of extending the Shape class. I do not think that this design is better than the first way, which is why the first design was the one implemented. This design is more complicated than the previous design and does not have the same versatility as the previous. In the first design, each child class of Shape only had to implement two methods. The rest of the implementation for each shape was in the Shape class, which also holds the parameters for the shape. Implementations of the Shape interface would have to implement each method of Shape individually, which makes it harder to add on more functionality later and makes it less modular.

Implementation

The sorting algorithm used to sort the shapes in order of their surface area is bubble sort. Bubble sort works by repeatedly swapping the elements until they are in the right order. If the algorithm detects an element that is out of order, it will swap it with the one beside. It continues to swap out-of-order elements until all the elements are sorted, when a flag “sorted” will be true, ending the loop. Bubble sort was implemented in this program because it is an easy-to-understand and easy-to-implement sorting solution. While bubble sort is not the most memory-efficient sorting algorithm, for the type of data and amount of data that needs to be sorted, bubble sort runs quickly and produces a very good output. The bubble sorting algorithm is called on an ArrayList of Shapes when the user clicks the “Sort shapes” button. If the list if empty when the user clicks the button, an error message will be generated to the console informing the user that the list of shapes is empty and cannot be sorted.

From the class diagrams in the previous section, Design #1 was the one implemented. Each class was implemented separately and in separate files. The first class developed was the MyPanel class, which runs the main program and contains the methods needed to draw the UI. The Shape class was created next and was developed to contain all the common elements of a shape, such as its height, width, and colour. The first type of Shape developed was the Circle. Developing the code to draw a circle allowed further testing of the functions of the program to be done. I used the circle to test the drawing, loading, and sorting methods before implementing the other two types of shape. If the circle was not working, then there was no point in moving on to incorporate more shapes that probably would not work either. After testing and confirming that the program was functioning as intended, I wrote the code for the other two types of shapes. Each type of shape is assigned an integer identifier between 1 and 3, with 1 being a Circle, 2 being a Square, and 3 being a Rectangle. This helps in allowing the program to choose a random shape to draw, as it simply chooses an integer between 1 and 3 and passes this to the ShapeFactory class which returns a new instantiation of the corresponding shape.

The minimum width and height for each shape is 10 pixels and the maximum width and height is 100 pixels. The program chooses a random integer between these two endpoints. These minimum and maximum dimensions were specified so that the shapes could be easily seen by the user and properly spaced out on the window. A new list of Shapes can be generated as many times as the user wishes. A new list of Shapes will simply override the previous list when the user clicks the "Load shapes" button.

This project was created using version 4.21.0 of Eclipse IDE and JavaSE 14. Java files were uploaded and shared on GitHub. A video and a README file are available in the GitHub repository with instructions on how to run the program. Figures 3-5 demonstrate the design of the user interface.

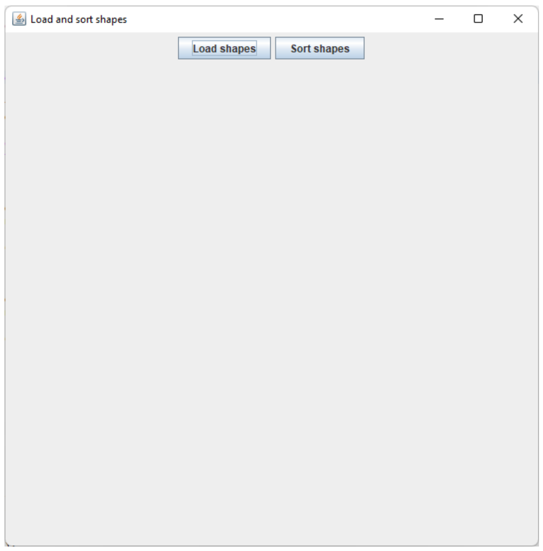

Figure 3: Initial screen

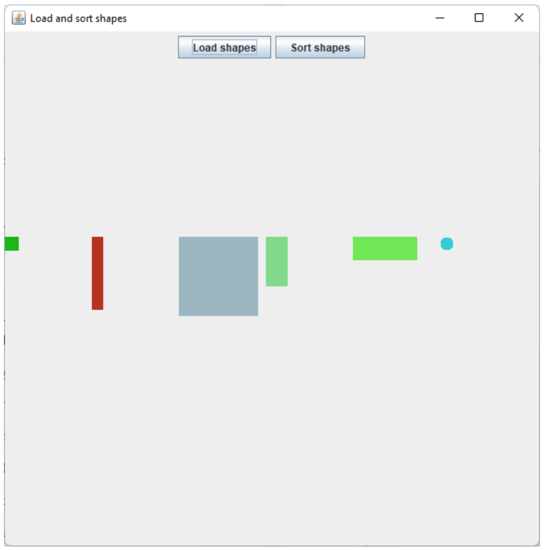

Figure 4: After clicking "Load shapes"

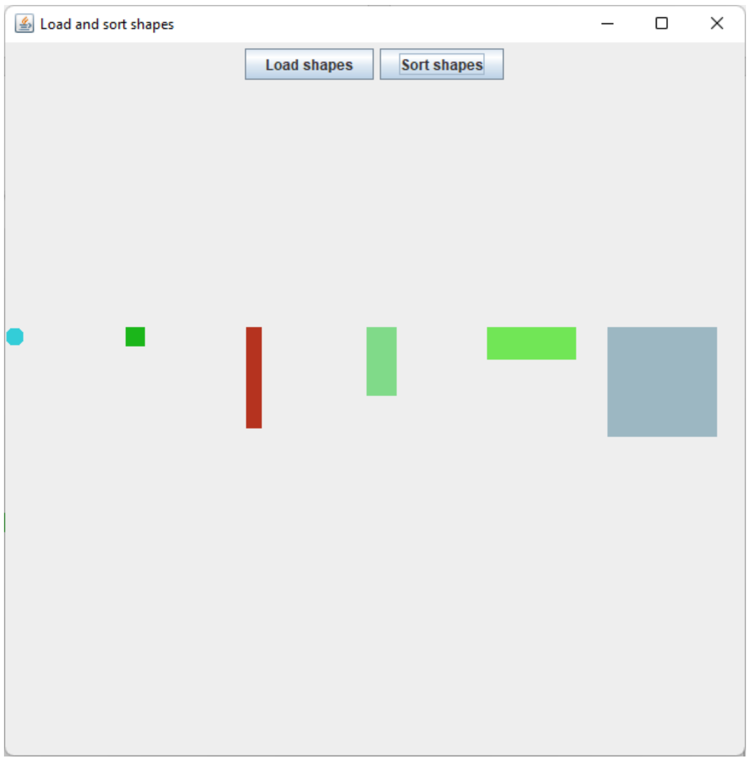

Figure 5: After clicking "Sort shapes"

Conclusion

I found this software project to be a good introduction to software design. I am familiar with Java and have been working with it for the past three years now, so the coding was not challenging. Implementing the design and writing the code was not a challenge for me and I believe that the end product works very well and functions exactly as intended. The more challenging part for me the design aspect, including the UML diagram. Although I have been exposed to UML and class diagrams in other classes before, this project required a more fleshed out diagram and design than some of my previous classes have. I would not say that anything necessarily went wrong with the project, but I did spend a fair amount of time researching UML and refreshing myself on Java, as it has been a few months since I last programmed. There was an instance where I had to redo a class because I failed to notice a part of the instructions. However, the issue was not hard to correct.

All in all, this project was a great opportunity to become more familiar with software design and object-oriented analysis and design. There is still lots to learn and I am excited to continue!

My top three recommendations to ease the completion of this project are:

1. **Get an early start.** I started this project the day it was posted and have been working on it in bits over the past few weeks. This helps to massively reduce my stress and allows me time to revise and think over my solutions, instead of rushing to complete two days before the due date.
2. **Do the software design first.** A lot of students will jump right into the coding first, as that is what they are most familiar with. I think it's best to follow the workflow and design a solution before implementing it.
3. **Ask questions when unsure.** Whenever I was unsure about a project requirement of specification, I found it best to send a quick email to ask my question, rather than guessing and hoping I did it right. Asking questions is an easy way to be sure that you are on the right track.