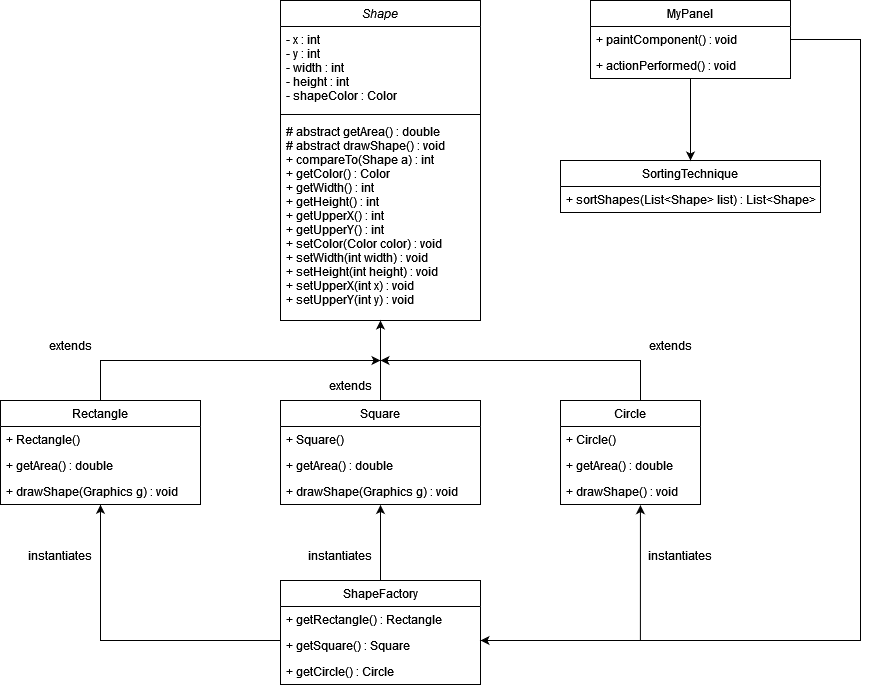
**Introduction**

* This software project is about creating either a circle, rectangle or square with random attributes (i.e., height, width and color) when the “Load Shapes” button is clicked. The other button, “Sort Shapes” button, sorts the created shapes by their areas, using any of the sorting algorithms.
* The foremost challenge was to get the GUI working. But after studying the code provided by the professor, I managed to achieve that satisfaction. This was the first project that required to use JButton and related javax.swing components, but it was not too complicated to study and implement them.
* The Shape class will be the parent class for the three inheritors namely the classes Rectangle, Square and Circle. This part of the project uses the concept of Inheritence. I have applied Encapsulation throughout the project files. In the process of developing my code, I figured that I would need to use the Abstraction principle to implement getArea and drawShape methods.

**Design of the solution**

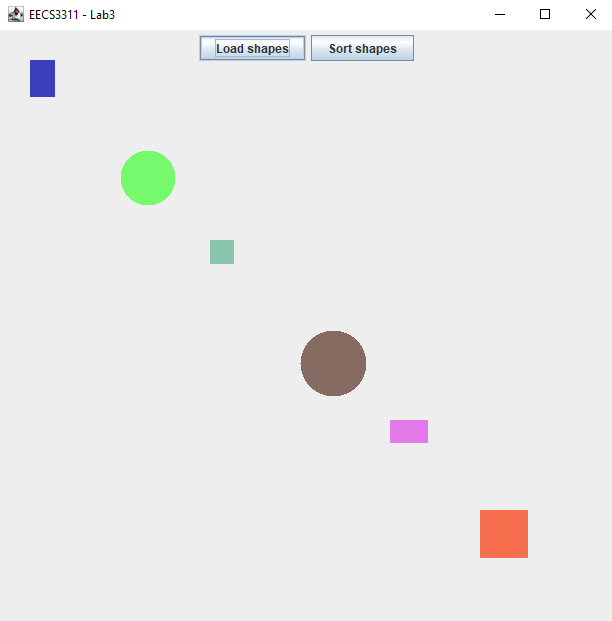
* Pattern 1:



* Abstraction is used for the class Shape, in order to implement getArea and drawShape methods. The same could have been achieved by an interface too. The classes Rectangle, Square and Circle make-use of the Shape class and there exists an IS-A relationship since they extend class Shape. Therefore, the application of Inheritence. MyPanel uses the methods in ShapeFactory class to get the randomized shapes. It also calls the SortingTechnique class to sort the shapes using Bubble Sort algorithm. The attributes of the class Shape are made private and there is no other application of encapsulation in this project, at least for my case.
* Pattern 2 : Just as mentioned above, another potential workable design would to have Shape as in interface and implement the methods accordingly. An even better modification would be to implement ShapeFactory in such a way that it returns a list of shapes instead of returning a single shape using an individual method for each shape. Other than that, the diagram on the previous page seems to fulfill most or all project requirements.

**Implementation of the solution**

* I used Bubble Sort technique to sort the shapes. The algorithm takes as input a List<Shape>. Two for-loops are used and they both run for as many times the number of shapes in the list. The adjacent elements are compared and swapped if needed under the inner for-loop. The best-case performance of this technique is O(n). And the worst-case performance of this technique is O(n2).
* Honestly, the two UML diagrams are pretty similar. I used the former diagram to code. And it seems to have made things simpler even though the other diagram would have yielded the same results but at the cost of more coding!
* I used Eclipse, JRE [JavaSE-15].



The screenshot on the left is upon clicking the “Load Shapes” button is clicked. While the other screenshot depicts that the shapes are sorted when “Sort Shapes” button is clicked.

**Conclusion**

* Using swing components was new to me but it was not very challenging. Implementing the ShapeFactory and SortingTechnique was a charming thing.
* After having written the code for SortingTechnique, I was bugged by the fact my shapes were not sorted on the display. I saw on my laptop for many minutes and only after enough struggle did I realize that I need to sync the coordinates!
* Using javax.swing components. How to implement a button and get things done.
* Understand the project requirements thoroughly. Do not delay starting the project. Refer to the materials provided by the professor and the suggested textbook to seek information pertaining the project.