CS 5700 Homework 1: Shapes Report

The first homework assignment was to create a shapes library using OO software development. We were supposed to extend the software to incorporate multiple shapes while using principles of abstraction and modularity. Before I dive into the code, I will first have to explain that I did not take this class because of a requirement, but more for the learning experience I hope to gain through the class. I have not used much Java or C sharp before this class, other than the brief amount that was required of me several years ago. I hope to continue the fast-paced learning that I have begun to undertake as I progress through this class.

As I worked through this assignment, I became aware of how useful UML diagrams can be. UML diagrams not only allow you to break down the design of your code, but they allow you to see how different pieces of your code will interact. This was new to me because while I have briefly used UML before, I have never really had the chance to step back and understand why people use it. I also begun to understand why it is necessary to continuously go back and update these diagrams. When I first started this assignment, I drew everything out on a whiteboard and tried to stick to that exact plan through the entirety of my code. I soon found that this was not very efficient and then had to go back and update this information as I put it into a UML diagram. Also, I created this diagram for the requirements of the class. Looking back, I realize how it would have been much easier to create a Shapes class that incorporated Points and Lines into one class. Then, I would have constructed an interface like I did, but through these two classes, I could have constructed all 2D shapes. I would have done this by using points and lines to create a center point and radius for the circles and ellipses, and then done the same for the vertices and sides of a polygon class. The polygon class would have had n number of lines constructed that connected with one another at shared vertices without crossing paths. Looking at things in retrospect however, always seems simpler than the path taken.

The path that I chose was to use the point class, and use it along with the General Shapes interface, in order to construct all shapes. I quickly realized the similarities between Ellipses and circles, creating two radii that then would be forced to similarity when being used by the Circle Class. I did the same thing with the Rectangle and Squares Class, except interchanging length for height and width. This allowed me to use the Point class to construct a rectangle class and an ellipse class, with very little code reuse in order to add a square and a circle class. Now by coding this way, I realized that I had restricted myself to rectangles and squares, instead of creating a polygon class that would have allowed me to easily create a triangle class. As shown in my diagram, I used the line class to create the triangles, with the point vertices interconnected.

Another new thing to me was using test cases. I had never used a system such as Gradle or Maven before. While writing these test cases, I began to understand how test cases really work and how simple and yet effective they can be. In the future, I plan to use them much more frequently in my coding and I hope to learn things such as PIT in order to do mutation test checking as I have been told that integrated programs like these add almost fool proof testing when used correctly.

Overall, I learned a lot from this first assignment although I struggled through the majority of it. I hope to have a better grasp on the next assignment and to get better with the design aspect of my coding as well as the actual software itself.

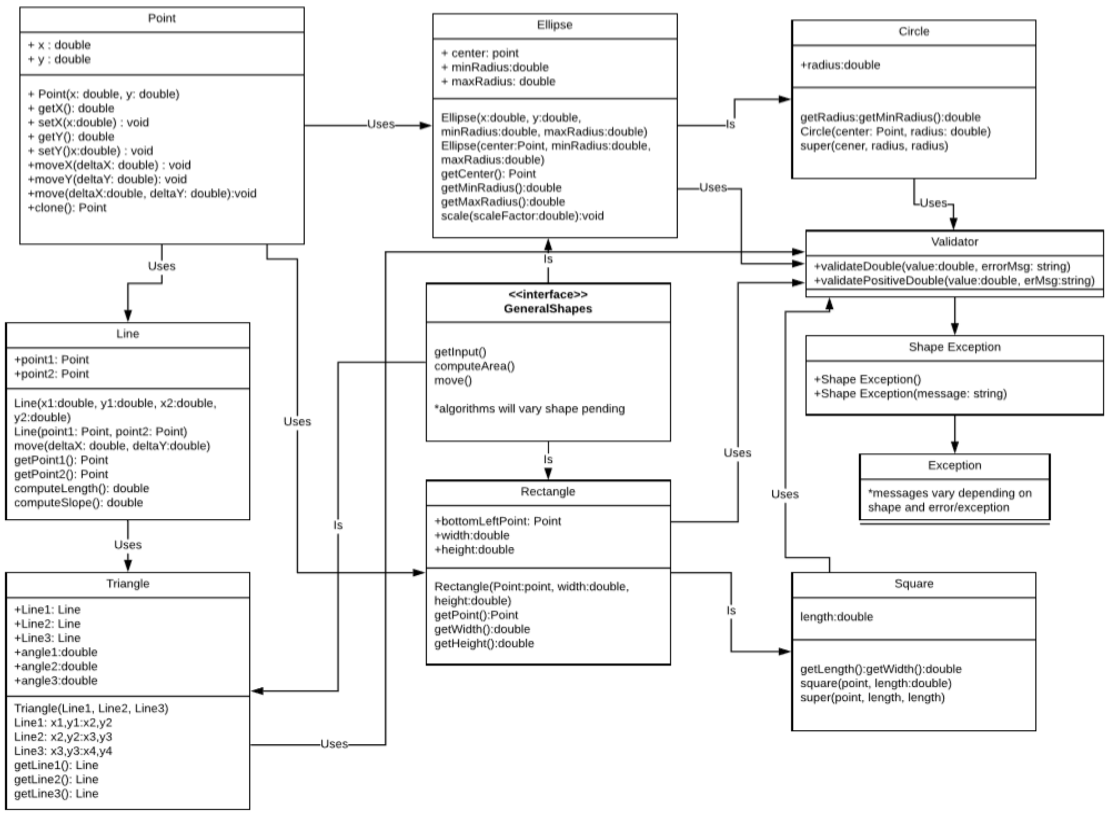
Point
+ x : double
+ y : double
+ Point(x: double, y: double)
+ getX(): double
+ setX(x:double) : void
+ getY(): double
+ setY()x:double) : void
+moveX(deltaX: double) : void
+moveY(deltaY: double): void
+move(deltaX:double, deltaY: double):void
+clone(): Point
Uses
Ellipse
+ center: point
+ minRadius:double
+ maxRadius: double
Ellipse(x:double, y:double, minRadius:double, maxRadius:double)
Ellipse(center:Point, minRadius:double, maxRadius:double)
getCenter(): Point
getMinRadius():double
getMaxRadius():double
scale(scaleFactor:double):void
Is
Circle
+radius:double
getRadius:getMinRadius():double
Circle(center: Point, radius: double)
super(cener, radius, radius)
Uses
Uses
Validator
+validateDouble(value:double, errorMsg: string)
+validatePositiveDouble(value:double, erMsg:string)
Shape Exception
+Shape Exception()
+Shape Exception(message: string)
Exception
*messages vary depending on shape and error/exception
Uses
Line
+point1: Point
+point2: Point
Line(x1:double, y1:double, x2:double, y2:double)
Line(point1: Point, point2: Point)
move(deltaX: double, deltaY:double)
getPoint1(): Point
getPoint2(): Point
computeLength(): double
computeSlope(): double
Is
<<interface>>
GeneralShapes
getInput()
computeArea()
move()
*algorithms will vary shape pending
Uses
Is
Is
Uses
Uses
Rectangle
+bottomLeftPoint: Point
+width:double
+height:double
Rectangle(Point:point, width:double, height:double)
getPoint():Point
getWidth():double
getHeight():double
Uses
Triangle
+Line1: Line
+Line2: Line
+Line3: Line
+angle1:double
+angle2:double
+angle3:double
Triangle(Line1, Line2, Line3)
Line1: x1,y1:x2,y2
Line2: x2,y2:x3,y3
Line3: x3,y3:x4,y4
getLine1(): Line
getLine2(): Line
getLine3(): Line
Uses
Is
Square
length:double
getLength():getWidth():double
square(point, length:double)
super(point, length, length)

Figure 1: UML of Shapes Library Diagram