Aaron Morgenegg
A02072659

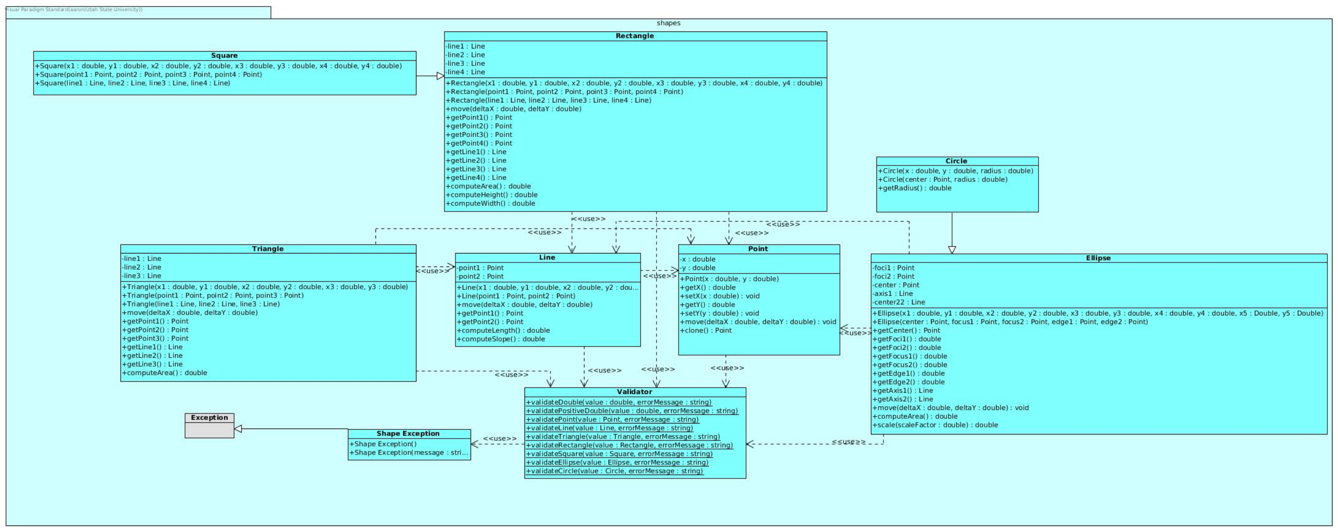
Assignment 1 : Abstraction and Modularity

The first change I made to the okay design of the shapes library was adding additional shapes to the UML diagram. The triangle, rectangle, square, circle, and ellipse shapes all used the point and line shapes as part of their implementation. I decided that the square and circle methods would inherit from rectangle and ellipse respectively. This resulted in the square and circle classes being very small, requiring another validation step and an extra method or two. I didn't do any other inheritance, as I didn't see a benefit to creating a hierarchy of shape classes. A high-level shape class in this case would just add more complexity without actually providing more utility or simplicity.

I tried to keep the design simple where I could. I consolidated the moveX and moveY functions into a single move function. For many shapes, for example the triangle, I stored the internal structure as a line, so that the user can access triangle.line1, or triangle.point1(which is the same thing as triangle.line1.point1). I overrode the constructor method to allow each shape to be constructed with coordinates, points, or lines. As a user, that seems very useful to have more construction methods available. Internally, it turned out to be a bit of a mess. I debated forcing the constructors to only use points, but decided against it. I'm not sure if it was for the best or not – maybe there is another solution I didn't think of.

Speaking of validation, I elected to make the validator class know about each shape, and contain the appropriate validation methods such as validateRectangle, validatePoint, etc. Normally I would include these methods as a part of the individual shapes, but I added them to the existing validator class. Again, not something I would normally do, but it was educational at least. I think I would prefer to have the shapes validate themselves. I don't think there is much of a benefit of making the validation functions separate, and as discussed in class, the validator becomes cluttered with functions as more shapes are added. There were some internal/utility validator functions that were used across multiple shapes, which is a benefit, but perhaps some of those could have been in a separate utility.

Overall I'm pleased with my design. It could be improved, but each class is simple, consistent, and well tested. If I had more time I might restructure things a bit, but it works well enough as is. I'd be interested to see more examples on this assignment in class, either from other students or from Professor Clyde. I'm curious about what design directions other people took on this assignment


Visual Paradigm Standard(aaron(Utah State University))
shapes
Square
+Square(x1 : double, y1 : double, x2 : double, y2 : double, x3 : double, y3 : double, x4 : double, y4 : double)
+Square(point1 : Point, point2 : Point, point3 : Point, point4 : Point)
+Square(line1 : Line, line2 : Line, line3 : Line, line4 : Line)
Rectangle
-line1 : Line
-line2 : Line
-line3 : Line
-line4 : Line
+Rectangle(x1 : double, y1 : double, x2 : double, y2 : double, x3 : double, y3 : double, x4 : double, y4 : double)
+Rectangle(point1 : Point, point2 : Point, point3 : Point, point4 : Point)
+Rectangle(line1 : Line, line2 : Line, line3 : Line, line4 : Line)
+move(deltaX : double, deltaY : double)
+getPoint1() : Point
+getPoint2() : Point
+getPoint3() : Point
+getPoint4() : Point
+getLine1() : Line
+getLine2() : Line
+getLine3() : Line
+getLine4() : Line
+computeArea() : double
+computeHeight() : double
+computeWidth() : double
Circle
+Circle(x : double, y : double, radius : double)
+Circle(center : Point, radius : double)
+getRadius() : double
Triangle
-line1 : Line
-line2 : Line
-line3 : Line
+Triangle(x1 : double, y1 : double, x2 : double, y2 : double, x3 : double, y3 : double)
+Triangle(point1 : Point, point2 : Point, point3 : Point)
+Triangle(line1 : Line, line2 : Line, line3 : Line)
+move(deltaX : double, deltaY : double)
+getPoint1() : Point
+getPoint2() : Point
+getPoint3() : Point
+getLine1() : Line
+getLine2() : Line
+getLine3() : Line
+computeArea() : double
Line
-point1 : Point
-point2 : Point
+Line(x1 : double, y1 : double, x2 : double, y2 : dou...
+Line(point1 : Point, point2 : Point)
+move(deltaX : double, deltaY : double)
+getPoint1() : Point
+getPoint2() : Point
+computeLength() : double
+computeSlope() : double
Point
-x : double
-y : double
+Point(x : double, y : double)
+getX() : double
+setX(x : double) : void
+getY() : double
+setY(y : double) : void
+move(deltaX : double, deltaY : double) : void
+clone() : Point
Ellipse
-foci1 : Point
-foci2 : Point
-center : Point
-axis1 : Line
-center22 : Line
+Ellipse(x1 : double, y1 : double, x2 : double, y2 : double, x3 : double, y3 : double, x4 : double, y4 : double, x5 : Double, y5 : Double)
+Ellipse(center : Point, focus1 : Point, focus2 : Point, edge1 : Point, edge2 : Point)
+getCenter() : Point
+getFoci1() : double
+getFoci2() : double
+getFocus1() : double
+getFocus2() : double
+getEdge1() : double
+getEdge2() : double
+getAxis1() : Line
+getAxis2() : Line
+move(deltaX : double, deltaY : double) : void
+computeArea() : double
+scale(scaleFactor : double) : double
Validator
+validateDouble(value : double, errorMessage : string)
+validatePositiveDouble(value : double, errorMessage : string)
+validatePoint(value : Point, errorMessage : string)
+validateLine(value : Line, errorMessage : string)
+validateTriangle(value : Triangle, errorMessage : string)
+validateRectangle(value : Rectangle, errorMessage : string)
+validateSquare(value : Square, errorMessage : string)
+validateEllipse(value : Ellipse, errorMessage : string)
+validateCircle(value : Circle, errorMessage : string)
Exception
Shape Exception
+Shape Exception()
+Shape Exception(message : stri...
<<use>>