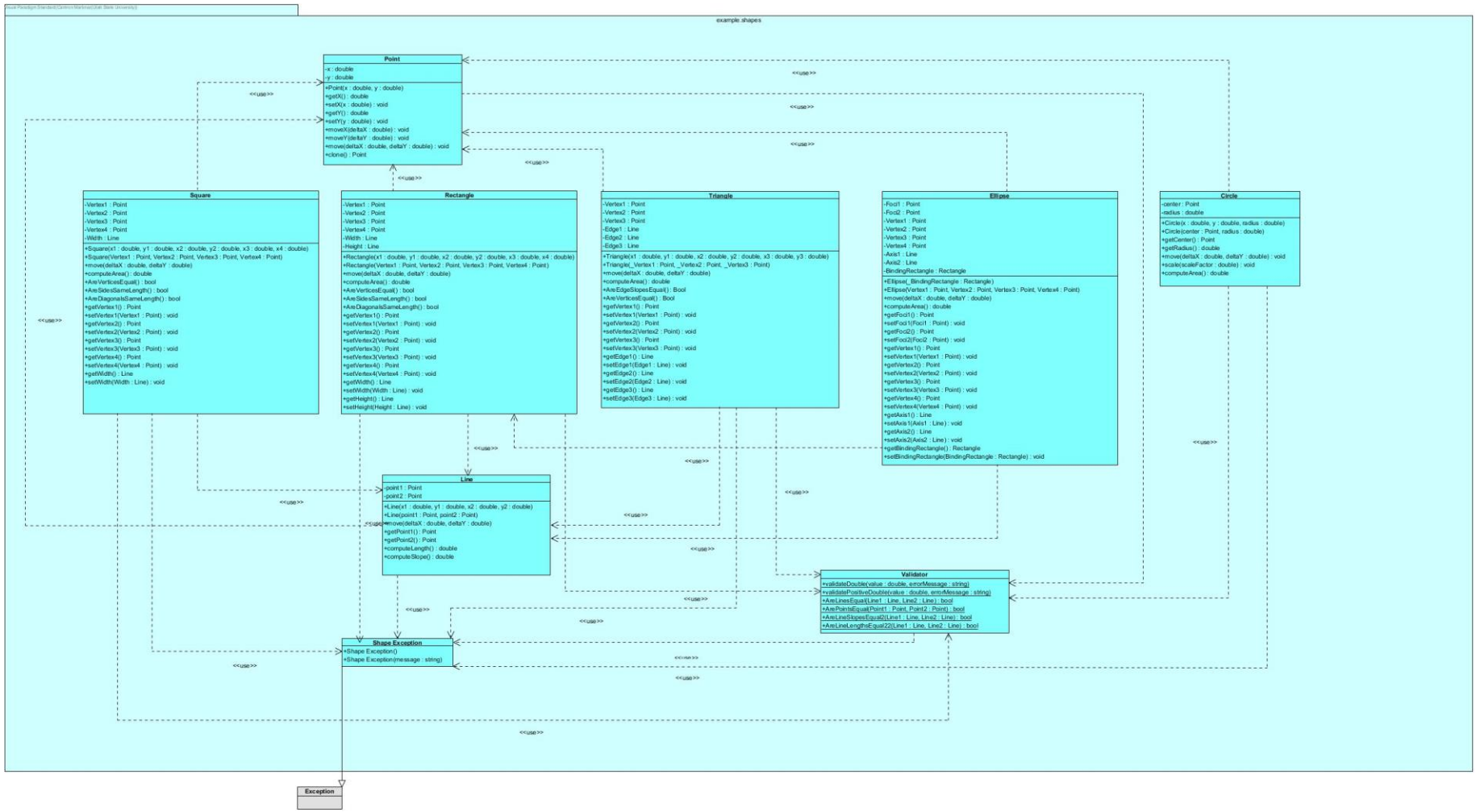
Hw1-Report Camron Martinez

While working on this report I discovered many things, and I really enjoyed working on it! I decided to use the validator class as an area for holding useful validation functions that would be used across many classes. I tried to avoid putting all of my validation logic in there though. I wanted to put only those things that wouldn't be class specific, such as comparing the lengths of lines or determining if two vertices are in the same location. This came to be very useful later when designing and creating the square, rectangle and triangle classes.

While trying to decide on how to validate shapes, I realized that if I validated the low level shapes perfectly, for example the line and point shapes, then I could reuse them in other classes to avoid rewriting validation. In the triangle class, in order to create a triangle of appropriate sides and vertices, I just made sure that you couldn't create a line of 0 length and then made the triangle out of those lines. I also realized that instead of dealing with angles, I could deal with slopes. If any 2 triangle edges have the same slope, then it is an invalid triangle! I thought that was pretty neat and that logic helped me to create tests and validation for rectangles and squares. If the opposite sides of a rectangle or square had the same slope, then it was more likely that they were rectangles or squares. I then just had to test for parallelograms. Pretty simple stuff. I learned that if you can create simple, yet robust lower classes, you can use that to your advantage when creating more complicated classes.

I spent quite a bit of time on the circle and ellipse. In geometry, a circle is a special case of an ellipse where the foci are in the same location. I thought it might be a good idea to have one inherit from the other, but this proved to be challenging because they didn't implement all of the same things. I also thought about creating a square and rectangle hierarchy since they are similar. A square is a special case of a rectangle. While these assumptions are good in geometry, they are less so in OO representation. I found some pretty neat articles on this very dilemma and realized that deriving circles from ellipses breaks the Liskov Substitution Principle in SOLID. Just goes to show that how something is represented in real life doesn't transfer exactly to OO programming.


example.shapes
Point
-x : double
-y : double
+Point(x : double, y : double)
+getX() : double
+setX(x : double) : void
+getY() : double
+setY(y : double) : void
+moveX(deltaX : double) : void
+moveY(deltaY : double) : void
+move(deltaX : double, deltaY : double) : void
+clone() : Point
Square
-Vertex1 : Point
-Vertex2 : Point
-Vertex3 : Point
-Vertex4 : Point
-Width : Line
+Square(x1 : double, y1 : double, x2 : double, y2 : double, x3 : double, x4 : double)
+Square(Vertex1 : Point, Vertex2 : Point, Vertex3 : Point, Vertex4 : Point)
+move(deltaX : double, deltaY : double)
+computeArea() : double
+AreVerticesEqual() : bool
+AreSidesSameLength() : bool
+AreDiagonalsSameLength() : bool
+getVertex1() : Point
+setVertex1(Vertex1 : Point) : void
+getVertex2() : Point
+setVertex2(Vertex2 : Point) : void
+getVertex3() : Point
+setVertex3(Vertex3 : Point) : void
+getVertex4() : Point
+setVertex4(Vertex4 : Point) : void
+getWidth() : Line
+setWidth(Width : Line) : void
Rectangle
-Vertex1 : Point
-Vertex2 : Point
-Vertex3 : Point
-Vertex4 : Point
-Width : Line
-Height : Line
+Rectangle(x1 : double, y1 : double, x2 : double, y2 : double, x3 : double, x4 : double)
+Rectangle(Vertex1 : Point, Vertex2 : Point, Vertex3 : Point, Vertex4 : Point)
+move(deltaX : double, deltaY : double)
+computeArea() : double
+AreVerticesEqual() : bool
+AreSidesSameLength() : bool
+AreDiagonalsSameLength() : bool
+getVertex1() : Point
+setVertex1(Vertex1 : Point) : void
+getVertex2() : Point
+setVertex2(Vertex2 : Point) : void
+getVertex3() : Point
+setVertex3(Vertex3 : Point) : void
+getVertex4() : Point
+setVertex4(Vertex4 : Point) : void
+getWidth() : Line
+setWidth(Width : Line) : void
+getHeight() : Line
+setHeight(Height : Line) : void
Triangle
-Vertex1 : Point
-Vertex2 : Point
-Vertex3 : Point
-Edge1 : Line
-Edge2 : Line
-Edge3 : Line
+Triangle(x1 : double, y1 : double, x2 : double, y2 : double, x3 : double, y3 : double)
+Triangle(_Vertex1 : Point, _Vertex2 : Point, _Vertex3 : Point)
+move(deltaX : double, deltaY : double)
+computeArea() : double
+AreEdgeSlopesEqual() : Bool
+AreVerticesEqual() : Bool
+getVertex1() : Point
+setVertex1(Vertex1 : Point) : void
+getVertex2() : Point
+setVertex2(Vertex2 : Point) : void
+getVertex3() : Point
+setVertex3(Vertex3 : Point) : void
+getEdge1() : Line
+setEdge1(Edge1 : Line) : void
+getEdge2() : Line
+setEdge2(Edge2 : Line) : void
+getEdge3() : Line
+setEdge3(Edge3 : Line) : void
Ellipse
-Foci1 : Point
-Foci2 : Point
-Vertex1 : Point
-Vertex2 : Point
-Vertex3 : Point
-Vertex4 : Point
-Axis1 : Line
-Axis2 : Line
-BindingRectangle : Rectangle
+Ellipse(_BindingRectangle : Rectangle)
+Ellipse(Vertex1 : Point, Vertex2 : Point, Vertex3 : Point, Vertex4 : Point)
+move(deltaX : double, deltaY : double)
+computeArea() : double
+getFoci1() : Point
+setFoci1(Foci1 : Point) : void
+getFoci2() : Point
+setFoci2(Foci2 : Point) : void
+getVertex1() : Point
+setVertex1(Vertex1 : Point) : void
+getVertex2() : Point
+setVertex2(Vertex2 : Point) : void
+getVertex3() : Point
+setVertex3(Vertex3 : Point) : void
+getVertex4() : Point
+setVertex4(Vertex4 : Point) : void
+getAxis1() : Line
+setAxis1(Axis1 : Line) : void
+getAxis2() : Line
+setAxis2(Axis2 : Line) : void
+getBindingRectangle() : Rectangle
+setBindingRectangle(BindingRectangle : Rectangle) : void
Circle
-center : Point
-radius : double
+Circle(x : double, y : double, radius : double)
+Circle(center : Point, radius : double)
+getCenter() : Point
+getRadius() : double
+move(deltaX : double, deltaY : double) : void
+scale(scaleFactor : double) : void
+computeArea() : double
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
Validator
+validateDouble(value : double, errorMessage : string)
+validatePositiveDouble(value : double, errorMessage : string)
+AreLinesEqual(Line1 : Line, Line2 : Line) : bool
+ArePointsEqual(Point1 : Point, Point2 : Point) : bool
+AreLineSlopesEqual2(Line1 : Line, Line2 : Line) : bool
+AreLineLengthsEqual22(Line1 : Line, Line2 : Line) : bool
Shape Exception
+Shape Exception()
+Shape Exception(message : string)
Exception
<<use>>