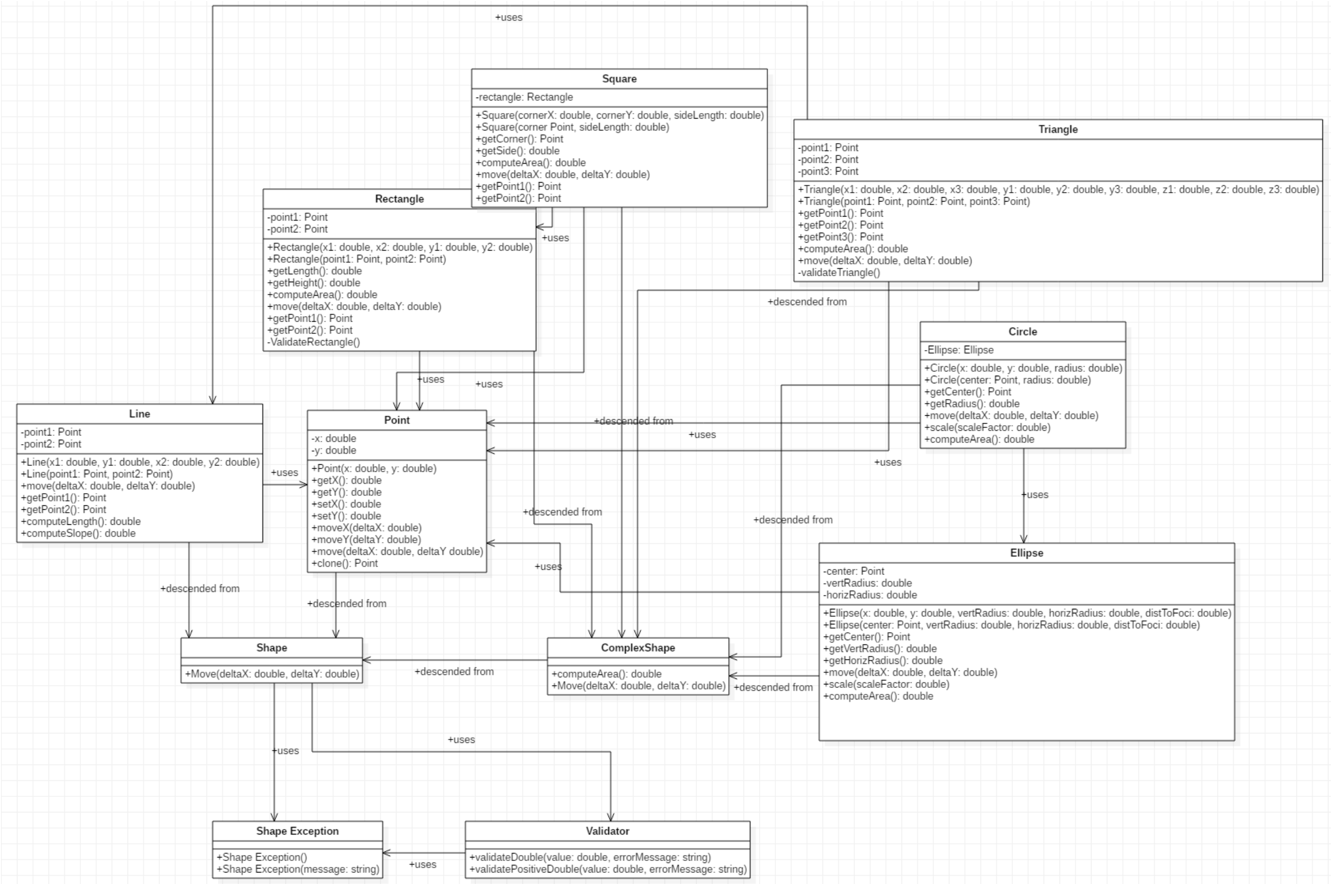
Jacob Ownsbey Assignment #1 CS5700-001


+uses
Square
-rectangle: Rectangle
+Square(cornerX: double, cornerY: double, sideLength: double)
+Square(corner Point, sideLength: double)
+getCorner(): Point
+getSide(): double
+computeArea(): double
+move(deltaX: double, deltaY: double)
+getPoint1(): Point
+getPoint2(): Point
Triangle
-point1: Point
-point2: Point
-point3: Point
+Triangle(x1: double, x2: double, x3: double, y1: double, y2: double, y3: double, z1: double, z2: double, z3: double)
+Triangle(point1: Point, point2: Point, point3: Point)
+getPoint1(): Point
+getPoint2(): Point
+getPoint3(): Point
+computeArea(): double
+move(deltaX: double, deltaY: double)
-validateTriangle()
Rectangle
-point1: Point
-point2: Point
+Rectangle(x1: double, x2: double, y1: double, y2: double)
+Rectangle(point1: Point, point2: Point)
+getLength(): double
+getHeight(): double
+computeArea(): double
+move(deltaX: double, deltaY: double)
+getPoint1(): Point
+getPoint2(): Point
-ValidateRectangle()
+uses
+descended from
Circle
-Ellipse: Ellipse
+Circle(x: double, y: double, radius: double)
+Circle(center: Point, radius: double)
+getCenter(): Point
+getRadius(): double
+move(deltaX: double, deltaY: double)
+scale(scaleFactor: double)
+computeArea(): double
+uses
+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
+uses
Point
-x: double
-y: double
+Point(x: double, y: double)
+getX(): double
+getY(): double
+setX(): double
+setY(): double
+moveX(deltaX: double)
+moveY(deltaY: double)
+move(deltaX: double, deltaY double)
+clone(): Point
+descended from
+uses
+uses
+uses
+descended from
+descended from
+uses
Ellipse
-center: Point
-vertRadius: double
-horizRadius: double
+Ellipse(x: double, y: double, vertRadius: double, horizRadius: double, distToFoci: double)
+Ellipse(center: Point, vertRadius: double, horizRadius: double, distToFoci: double)
+getCenter(): Point
+getVertRadius(): double
+getHorizRadius(): double
+move(deltaX: double, deltaY: double)
+scale(scaleFactor: double)
+computeArea(): double
+uses
+descended from
+descended from
Shape
+Move(deltaX: double, deltaY: double)
+descended from
ComplexShape
+computeArea(): double
+Move(deltaX: double, deltaY: double)
+descended from
+uses
+uses
Shape Exception
+Shape Exception()
+Shape Exception(message: string)
+uses
Validator
+validateDouble(value: double, errorMessage: string)
+validatePositiveDouble(value: double, errorMessage: string)

I chose to create two parent abstract classes: Shape, and ComplexShape. Shape required that all objects descended from it implement the Move() function. ComplexShape required that all object descended from it implement the ComputeArea() function (as well as the Move() function inherited from Shape). I chose to make Square a wrapper for Rectangle; and Circle a wrapper for Ellipse. In both cases, the wrappers simply carry their wrapped object as a class variable, and use their constructors to essentially constrain the user into creating the wrapped shape as if it were the wrapper.

During the implementation stage, it became clear that I could benefit from using Line to compute the area of a Triangle object. Also during this stage, I decided to make the wrapper classes keep their wrapped object as a private member variable, accessible only with getter functions (this is inconsistent with the other classes, but prevents the user from needing to treat the wrapper class as a wrapped class when accessing its variable).

There was little changed during the testing stage, aside from creating a utility function in Point to allow Points to be compared to see if their coordinates were equal.