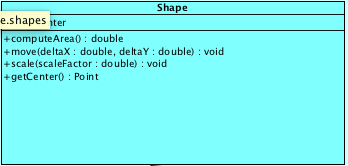
Abstract and Modularity

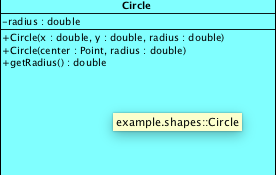
To start the project off, I made all the other shape classes in the same exact format that the Circle class was made with in the okay design given. Once done it was time to abstract. I did this by determining what the common factor was between all these classes. All the classes were Shapes, so I made an abstract class called shape. Then it was time to figure out what items could be defined in the Shape class that all these other classes would simply inherit. For example, every shape has a center point, and has a few of the same functions (getCenter, move, scale, and computeArea).



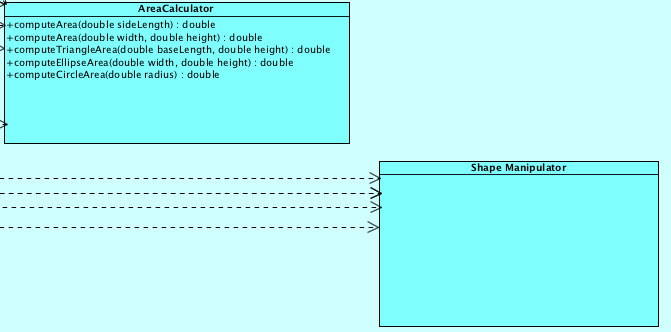
The benefit from this is that when creating a new shape and extending the Shape class it will tell you all of the functions that need to be defined, and a few of the functions won’t need to be written at all). It also organizes everything mush much better. My code would be all over the place if I freely created each shape class without any connection.

The original Circle class contained a Point a move function, scale function etc, each needed to be created and defined in the class.

I was able to simply it to the following through abstraction:



The anything else in the class is inherited from the shape class. Not sure if it’s the best design but I created an AreaHelper class that handles the calculating area as opposed to doing it in the classes. I also planned on creating a ManipulateShapes that that would have handled things such as the move and scale functions. I wasn’t sure if this was the best idea so I didn’t follow through, but I left it there to discuss.



The Junit testing was pretty straight forward when looking at the template. The only question I had was how good is good enough. For example (and maybe this is bad design) my Triangle class takes 6 doubles minimum as parameters. So for testing invalid constructors there is an insanely large combinations to test those parameters for inputs. If I think about it, there is the value for positive infinity, negative infinity, Nan, and for 6 parameters that is more than 6 factorial combinations that would need to be created if we were to cover all bases. This is even the case for the Line class (which was gen to us) that takes 4 doubles as parameters. There are so many combinations, here’s a few:

