Assignment 2, problem 3

The main issue with having the Square class as the base class is that it is naturally limited in assuming that any functionality relies on all dimensions being the same value. While this is just fine for a square, it does not work so well when dealing with the different side lengths of a rectangle.

By having the rectangle as the base class as in the first problem, we were able to instantiate the square class, derive from the rectangle base class and then process any request for area or perimeter through the base class without issue. Handling things the other way around made for a messy subclass that had to perform much more work that was necessary to complete its tasks.

In both instances I created my base classes as abstract classes. This by definition allowed them to follow the OCP. The fact that these base classes are not sealed means they could be changed. But, by being a base class, the subclass could implement its own methods (such as the subclass of rectangle did) and then use the base class properties and methods as well as its own required methods.

One other way of performing these tasks would be to have a base class as an Interface. This would actually be a better approach in a few regards. By using an interface I could have further locked down the subclass and how the base class itself would be used. This could also allow for the use of IOC though a container such as Ninject and provide even greater control over the flow and use of base classes within the program.

The clear best base class in this program example is the rectangle as the base class. With this extra functionality the base methods of rectangle can handle any rectangle object as well as any square object just as well. Through knowing how classes can build off of and extend one another it is key to keep these thoughts in mind when creating new classes in production code.