# The Rectangle/Square Controversy

A software design problem that has often been discussed in the literature is whether a square should inherit from a rectangle or whether a rectangle should inherit from a square? Two opposing arguments are made by Meyer [1] and Martin, et.al. [2], both of which seem to have some merit. How is a software developer to decide which is the correct design? In this post I will discuss why I believe both approaches are wrong and I will provide the solution I prefer.

Meyer [1] proposes the design where square inherits from rectangle with square imposing the invariant that the width must equal the height. The conceptual design is shown in Figure 1. This design makes sense since a square “is-a” rectangle. Remember, that “is-a” is the litmus test for inheritance [3]. To state this more precisely: inheritance defines a subset relation between a child and a parent class. Thus, the Square class represents the set of squares which is a subset of the set of rectangles which is represented by the Rectangle class. This is shown in Figure 2. For a detailed discussion on the mathematical semantics of inheritance, see Meyer [1]. The implementation is shown in Figure 3.

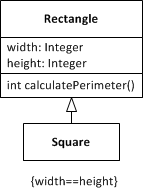


Figure : Square extends Rectangle

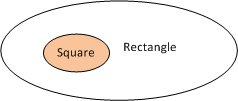


Figure : The set of squares is a subset of the set of rectangles

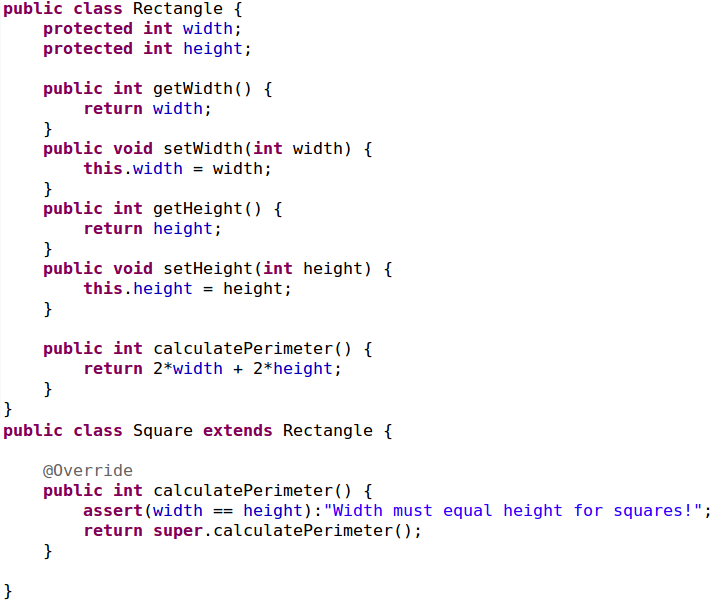


Figure : The implementation of a Square extending a Rectangle

Martin’s criticism against the design of Figure 1 is that it violates the Liskov substitution principle, which states that subtypes must be substitutable for their base types [2]. Clients of instances of the Square class cannot set the width and height of a square as if they are using a rectangle. In particular they need to ascertain that the width and height of instances of square are equal. The main problem is that a square does not require both a width and a height and it therefore needs to ignore either of the attributes.

An alternative design that Martin discusses is where the setWidth (respectively setHeight) method is changed to set the height equal to the width (respectively to set the width equal to the height). However, the problem is when setWidth (respectively setHeight) is called it will overwrite the value of the height (respectively width) (see Figure 4). Hence, a client that first sets the width to 5 and then the height to 3 will expect the perimeter to be 16, but instead it will be 12. This again violates the Liskov substitution principle.

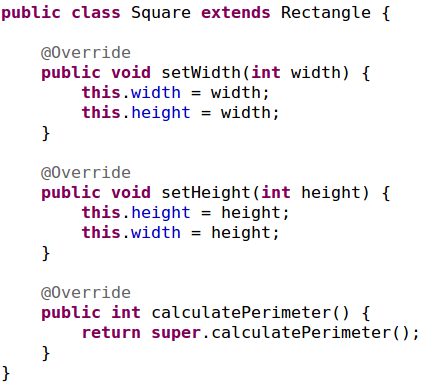


Figure : Forcing the width and height to be equal

The solution that Martin [2] proposes is that the Rectangle class must inherit from the Square class (see Figure 5). The objection that Meyer has against this design is that it violates the “is-a” relation in that a rectangle is not a square. Now you may reason that a rectangle is indeed a square when its width and height are equal, but the crucial point is that a rectangle is not **always** a square. Remember, inheritance enforces a subset relation between a child and a parent. That is, all instances of the child are necessarily instances of the parent, but not all instances of the parent are necessarily instances of the child.

The flaw in the solution that Martin proposes is that he uses inheritance for the sole purpose of reuse without there being an “is-a” relation. Meyer calls this “convenience inheritance” which should be avoided.

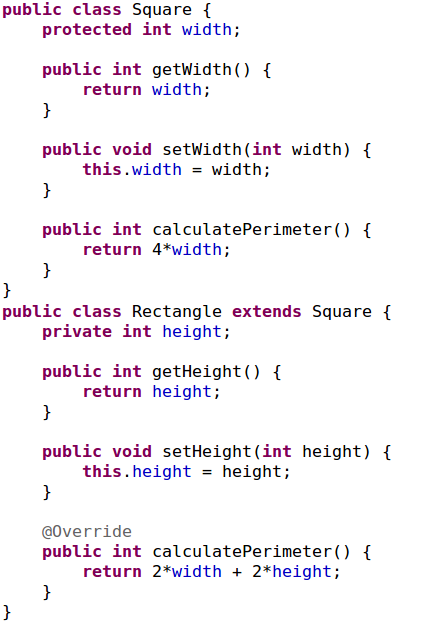


Figure 5: Rectangle extends Square

From a conceptual perspective a way around these objections is to introduce a class (i.e. Quadrilateral) from which both the Rectangle and Square classes can inherit. The related UML class diagram is shown in Figure 6. The {incomplete, overlapping} annotation indicates that other quadrilaterals may exist and that rectangles may occasionally be squares (when width=height).

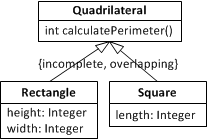


Figure 6: Conceptual solution for Square and Rectangle problem

The implementation of this conceptual solution can be done where Quadrilateral is either an interface or an abstract class. Figure 7 shows the implementation where Quadrilateral is defined as an interface.

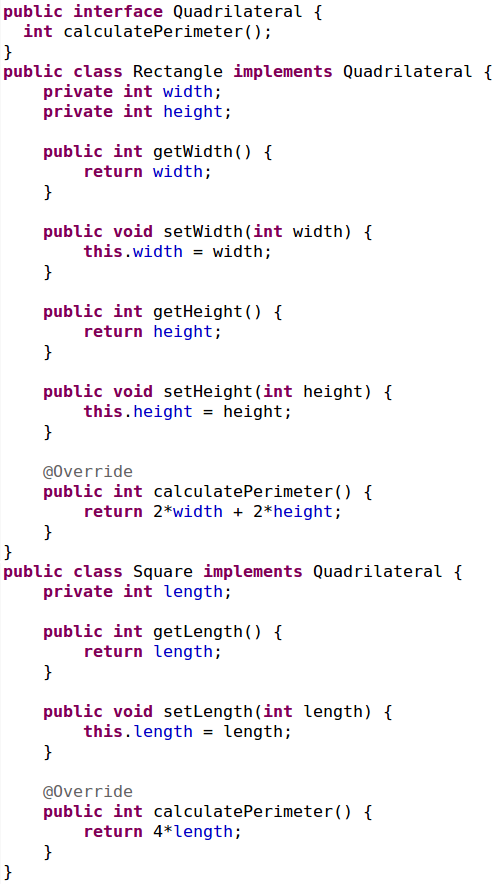


Figure 7: A correct implementation for the Square and Rectangle classes

Naturally I am not the first to point this solution out. In conclusion I to elsewhere on the Internet where this solution has been discussed/implied [4] [5].

# Works Cited

|  |  |
| --- | --- |
| [1] | B. Meyer, Object-oriented software construction, Prentice Hall, 1997. |
| [2] | R. Martin and M. Micah, Agile principles, patterns, and practices in C#, Prentice Hall, 2006. |
| [3] | G. Booch, R. A. Maksimchuk, M. W. Engel, B. J. Young, J. Conallen and K. A. Houston, Object-oriented analysis and design with applications, Addison-Wesley Professional, 2007. |
| [4] | R. Carr, "Is a Square a Rectangle?," [Online]. Available: http://www.blackwasp.co.uk/SquareRectangle.aspx. |
| [5] | H. Makabee, "When a square is not a rectangle," [Online]. Available: http://effectivesoftwaredesign.com/2010/09/20/when-a-square-is-not-a-rectangle/. |