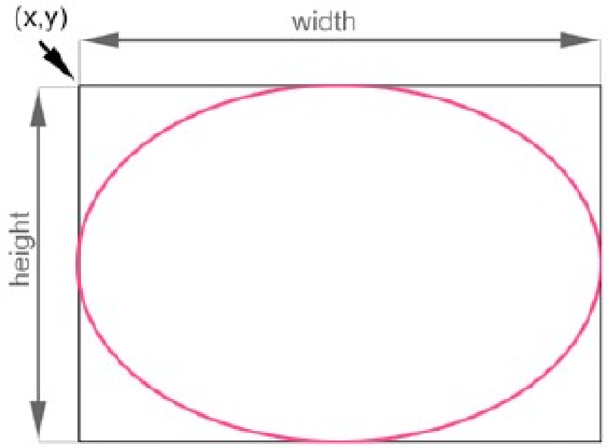
17. Inheritance :: An *Oval* Class

Suppose we are going to create more classes to represent additional shapes that we can draw on the graphical window. In particular, let's consider a class named *Oval* to represent ovals (i.e., ellipses). Without getting in to the analytic geometry definition of what an ellipse is, an alternative representation is to specify an oval with four parameters: the x coordinate of the upper left corner of a **bounding box**; the y coordinate of the upper left corner of the bounding box; the *width* of the bounding box; and the *height* of the bounding box.



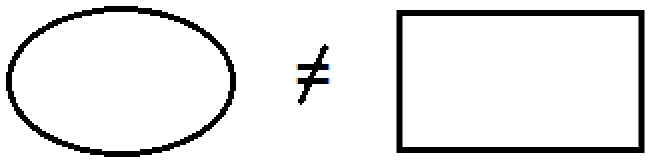
The Java Class Library has a method to draw ovals on a graphical window using this representation so this is the one we will use.

17. Inheritance :: Ovals ain't Rectangles

Before moving to the design of the *Oval* class, you may have noticed that an *Oval* object will have the same instance variables as a *Rectangle* object: *mX*, *mY*, *mWidth*, and *mHeight* and the same methods, e.g., *getX*(), *getWidth*(), *setHeight*(), etc.

Does this mean that we should make the *Oval* class inherit from *Rectangle*? Or why don't we just represent *Oval* objects using the *Rectangle* class and do away with the *Oval* class altogether?

Although we *could* actually write the code using either of those techniques, there would be something fundamentally flawed about it:



Or in English: **ovals ain't rectangles**. Philosophically, there is no **is-a** relationship between *Oval*s and *Rectangle*s. They do not look anything alike. From an analytic geometry perspectives, the equations to represent ovals (ellipses) and rectangles do not have anything in common. So, because *Oval*s **are not** *Rectangle*s it would be improper to represent an *Oval* object using the *Rectangle* class. And because an *Oval* **is not a specific type** of *Rectangle* it would be improper to declare *Oval* as a subclass of *Rectangle*.

17. Inheritance :: *Oval*/*Rectangle* Similarities

However, there *are* some similarities between *Oval*s and *Rectangle*s, specifically, the graphical window locations of each are represented as an (x, y) coordinate. Does it make sense, then, for the *Oval* class to declare mX and mY instance variables and provide **public** $getX()$, $setX()$, $getY()$, $setY()$, and $move()$ methods? The answer is easy once you recognize that the **code dealing with the coordinates would be identical in both classes**.

And what did we conclude earlier about copying-and-pasting code from the *Rectangle* class to the *Oval* class? *Anytime* you find yourself copying-and-pasting code because you need to the same thing somewhere else, there is generally a better way to do what you are doing.

So what *is* the best way to handle this? Before you answer, think about this: ovals and rectangles are types of shapes, right? In fact, a square is a shape, a circle is a shape, a triangle is a shape, a line is a shape, a quadrilateral is a shape, and so on. I repeat:

> A square **is a** shape, a circle **is a** shape, a triangle **is a** shape, a line **is a** shape, a quadrilateral **is a** shape.

When *SomeClassInYourDesign* **is a** *SomeOtherClassInYourDesign* the proper way to design *SomeClassInYourDesign* is to make it a subclass of *SomeOtherClassInYourDesign*.

The answer: let's create a class named *Shape* and make *Oval*, *Rectangle*, *Square*, *Circle*, *Triangle*, *Line*, and *Quadrilateral* all subclasses of *S*
hape.