Return to Module Overview Page
(https://onlinelearning.berkeley.edu/courses/665040/pages/module-five)

5.5: Polymorphism: More than Just a Twenty-Dollar Word

Now imagine that we've defined Shape as above, as well as Circle, Square, Triangle, Rhombus, etc., all of which extend Shape. Through some means (a graphical interface, perhaps) a picture has been created. The picture is represented by an array of Shapes. Each element of the array is really a Circle, or a Square, or a Triangle; but we can draw the picture like this:

```
void drawPicture(Shape [] s)
{
  for (int i=0; i < s.length; i++)
    s[i].draw();
}
```

This unremarkable looking code illustrates an incredibly powerful concept. As this simple loop traverses the array of Shapes, calling draw() for each one, the version of draw() that is called is chosen at runtime based on the actual type of the contents of the particular element of the Shape array. In other words, if Shape contains a Circle, a Square, and a Triangle, in that order, then the functions Circle.draw( ), Square.draw( ), and Triangle.draw( ) are called! Contrast this with pseudocode to accomplish something similar in C or Visual Basic:

```
for each element in s
     if (s[i] instanceof Circle) then
        drawCircle();
     else if (s[i] instanceof Square) then
        drawSquare();
     else if (s[i] instanceof Triangle) then
        drawTriangle();
```

(The instanceof operator, a real Java operator, returns true if the left-hand operand could legally be assigned to a variable whose type is the right-hand argument. This means that the left hand side must be of the type the right hand side, or one of its subclasses.)

This non-Java code is vastly inferior, for several reasons. It's bigger, for one thing. Second, it includes each type name twice, making mismatching a possible source of bugs. And third, it contains all the type names, which is bad in and of itself. If tomorrow I write a Pentagon class, I must also change this loop to add Pentagon code. If loops like this appear in many places in a program, I might forget to update them all, leading to more bugs. In the Java version, on the other hand, the loop will always work, regardless of what new Shape classes I add to the program. I will not need to touch drawPicture( ), ever, no matter how many new kinds of Shapes are added.

The idea of implementing class-specific behaviors and then invoking them through a generic interface, allowing each object to "do the right thing," is called *polymorphism*. It is really the crux of object-oriented programming. Redefining a method defined in a superclass is called "overriding" the method. Polymorphism and overridden methods always work together.