4. Polymorphism :: Polymorphism to the Rescue

Fortunately, Java **does** support polymorphism, and the better way to implement our *Shapes* hierarchy and *WindowManager.redrawWindow*() method is to use polymorphism. The correct way is:

1. Modify the abstract *Shape* class and add an abstract method named *draw*().

```
public abstract class Shape {
  public void draw();  // A nonimplemented method in an abstract class
  ...                  // is an abstract method
}
```

2. Since *draw*() is abstract in *Shape*, each subclass of *Shape* will have to override and implement *draw*()—if not, the subclass becomes an abstract class as well, even if it is not declared as abstract.

```
public class Rectangle extends Shape {
  @Override
  public void draw() { code to draw a rectangle is here }
  ...
}

public class Oval extends Shape {
  @Override
  public void draw() { code to draw an Oval is here }
  ...
}
```

And so on for each *Shape* subclass.

3. Modify *WindowManager.redrawWindow*()

```
public class WindowManager() {
  private ArrayList<Shape> shapes;
  public void redrawWindow() {
    for (Shape shape : shapes) {
      shape.draw();  // This is a polymorphic method call
    }
  }
  ...
}
```

The method call *shape.draw*() in the enhanced for loop is a polymorphic method call. Why? The loop object variable *shape* is declared to be of the class *Shape* but the contents of *shapes are Rectangle*s, *Square*s, *Oval*s, and so on. Since each *Shape* subclass provides an implementation of the overridden *Shape* class abstract *draw*() method, the **correct** *draw*() method will be polymorphically called on *shape* depending on the class of the object that *shape* actually refers to.

This example illustrates the primary advantages of polymorphism: our program is more easily extended. To add a new shape we simply design and implement the new class, including the overridden *draw*() method. The *WindowManager* class does not have to be modified every time a new subclass of *Shape* is added to the *Shape* hierarchy. And finally, from an aesthetic standpoint, the proper locations for the *draw*() methods are in each of the *Shape* subclasses and not in the *WindowManager* class: no one knows how better to draw a rectangle on the window than a *Rectangle* object.