25. Inheritance :: The *Object.toString*() Method

Now, *Object* **is not** an abstract class, but it contains two methods that all subclasses are expected to **override**:

```
String toString();
boolean equals(Object pAnotherObject);
```

The *toString*() method is supposed to return a **string representation** of the object where the definition of "string representation" is up to the programmer. Since *toString*() is commonly used during debugging to print information about an object to the output window, the string representation of an object usually displays the values of some or all of the object's instance variables.

For example, we can modify our *Point* class to include a *toString*() method:

```
public class Point {
  ...
  @Override  // This annotation tells the compiler we intend to override toString()
  String toString() {
    return "[Point: mX = " + getX() + ", mY = " + getY() + "]\n";
  }
}
```

25. Inheritance :: The *Object.toString*() Method (continued)

Given a *Point* object, we can print information about it:

```
Point pete = new Point(10, 20);
...
System.out.println(pete.toString());
```

which will print:

```
[Point: mX = 10, mY = 20]
```

Note that we do not actually have to call *toString*() on *pete*. Whenever an object is used in a place where a *String* is expected (the parameter to *println*() is a *String*) the compiler will generate code that automatically calls the *toString*() method on the object. Thus, we could write:

```
System.out.println(pete);
```

to the same effect.