26. Inheritance :: The *Object.equals*() Method

The other method that objects are expected to override is *equals*(), which compares two objects to see if they are equal, i.e., they have the same contents. Remember that object variables actually contain references:

```
Point pete = new Point(10, 20);
Point patty = new Point(10, 20);
if (pete == patty) {
  System.out.println("pete equals patty");
} else {
  System.out.println("pete does not equal patty");
}
```

which will print:

```
pete does not equal patty
```

since *pete* and *patty* contain different references. To actually compare *pete* and *patty* to see if their mX and mY instance variables are equal, we override *Object.equals*():

```
public class Point {
  @Override
  boolean equals(Object pObject) {
    Point point = (Point)(pObject);
    // This code will be completed in a bit...
  }
}
```

26. Inheritance :: The *Object.equals*() Method (continued)

First, why is the parameter to *Point.equals*() an *Object* rather than a *Point*?

Remember, we are **overriding** *Object.equals(Object)* and to override a superclass method, the method signature must be the same in the superclass (*Object*) and the subclass (*Point*). If we wrote:

```
public class Point {
  @Override
  boolean equals(Point pAnotherPoint) {  // Because of the @Override attribute
    Point p = (Point)(pAnotherPoint);    // this code will not compile because
    ...                                  // we are accidentally overloading
  }                                      // Object.equals(Object).
}
```

then we would be accidentally **overloading** *equals*().

26. Inheritance :: The *Object.equals*() Method (continued)

If you read the Java documentation, the rules for defining two objects to be "equal" is a bit complicated so in the interest of time, we will define two *Point* objects *point1* and *point2* as being equal if:

1. If *point2* is null then *point1.equals*(*point2*) returns false.
2. If *point1* and *point2* refer to the same object (i.e., *point1* == *point2* is true) then *point1.equals* (*point2*) returns true.
3. If the mX and mY instance variables of *point1* and *point2* are equal then *point1.equals*(*point2*) returns true.
4. Otherwise, *point1.equals*(*point2*) returns false.

26. Inheritance :: The *Object.equals*() Method (continued)

Here is the completed overridden *Point.equals*() method:

```
public class Point {
  @Override
  boolean equals(Object pObject) {
    // Must typecast pObject to Point.
    Point point = (Point)(pObject);

    // Rule 1: If point is null, return false.
    if (point == null) return false;

    // Rule 2: If this and point refer to the same object, return true.
    else if (this == point) return true;

    // Rule 3: If the mX and mY instance variables of this Point and point are
    // equal, return true.
    else if (getX() == point.getX() && getY() == point.getY()) return true;

    // Rule 4: Otherwise, return false.
    else return false;
  }
}
```

26. Inheritance :: The *Object.equals*() Method (continued)

To use the *equals*() method to compare two *Points* for equality:

```
Point pete = new Point(10, 20);
Point patty = new Point(10, 20);
if (pete.equals(patty)) {
  System.out.println("pete equals patty");
} else {
  System.out.println("pete does not equal patty");
}
```

which will print:

```
pete equals patty
```