Special Topic 10.5

Inheritance and the `equals` Method

You just saw how to write an `equals` method: Cast the `otherObject` parameter to the type of your class, and then compare the instance variables of the implicit parameter and the other parameter.

But what if someone called `coin1.equals(x)` where `x` wasn't a `Coin` object? Then the bad cast would generate an exception, and the program would die. Therefore, you first want to test whether `otherObject` really is an instance of the `Coin` class. The easiest test would be with the `instanceof` operator. However, that test is not specific enough. It would be possible for `otherObject` to belong to some subclass of `Coin`. To rule out that possibility, you should test whether the two objects belong to the *same class*. If not, return `false`.

```
if (getClass() != otherObject.getClass()) return false;
```

Moreover, the Java language specification demands that the `equals` method return `false` when `otherObject` is `null`.

Here is an improved version of the `equals` method that takes these two points into account:

```
public boolean equals(Object otherObject)
{
   if (otherObject == null) return false;
   if (getClass() != otherObject.getClass())
      return false;

   Coin other = (Coin) otherObject;
   return name.equals(other.name) && value == other.value;
}
```

When you implement equals in a subclass, you should first call equals in the superclass, like this:

```
public CollectibleCoin extends Coin
{
   private int year;
   . . .
   public boolean equals(Object otherObject)
   {
      if (!super.equals(otherObject)) return false;

      CollectibleCoin other = (CollectibleCoin) otherObject;
      return year == other.year;
   }
}
```