12. Inheritance :: Accidental Overloading

Accidental overloading occurs when an attempt is made to **override** a method in a subclass but the signatures of the methods in the superclass and subclass do not match (which of course, results in **overloading**). Example:

```
public class Super {
  public void aMethod(int pX, int pY) { ... }
}

public class Sub extends Super {
  // Assume we wish to override aMethod().
  public void aMethod(int pX, double pY) { ... }  // Overloads aMethod()
}
```

Here, an attempt is made to override *aMethod*() in *Sub* but note that the signatures of the methods in *Super* and *Sub* are different:

```
<Super.aMethod: int, int>
<Sub.aMethod: int, double>
```

Consequently, *Sub.aMethod*() is **accidentally overloaded**.

12. Inheritance :: Accidental Overloading (continued)

What happens when a method is accidentally overloaded? Consider this code:

```
Sub sub = new Sub();
sub.aMethod(10, 20);  // Calls Super.aMethod(int, int)
```

The intent in the above statement is to call the (incorrectly) overridden *Sub.aMethod*(**int**, **double**) on *sub* but since *Sub.aMethod*() was accidentally overloaded, the result is that *Super.aMethod*(**int**, **int**) is actually called on *sub*. This will generally lead to a bug.

12. Inheritance :: @Override Attribute

To avoid making the accidental overloading mistake, Java 7 supports an **attribute** that can be specified on an **overridden** method to tell the compiler that an attempt is being made to override a superclass method:

```
public class Super {
  public void aMethod(int pX, int pY) { ... }
}

public class Sub extends Super {
  @Override  // We wish to override aMethod().
  public void aMethod(int pX, double pY) { ... } // Syntax error!
}
```

A Java attribute is (for our purposes) written as **@*attribute-name***. The convention is to write the attribute on a separate line of code above the method header. There are various Java attributes but this is the only one we will discuss for now.