This explains the behavior that we observe in our *network* project. Only the **display** methods in the subclasses (**MessagePost** and **PhotoPost**) are executed when posts are printed out, leading to incomplete listings. In the next section, we discuss how to fix this.

11.5 super call in methods

Now that we know in detail how overridden methods are executed, we can understand the solution to the problem. It is easy to see that what we would want to achieve is for every call to a **display** method of, say, a **PhotoPost** object, to result in both the **display** method of the **Post** class and that of the **PhotoPost** class being executed for the same object. Then all the details would be printed out. (A different solution will be discussed later in this chapter.)

This is, in fact, quite easy to achieve. We can simply use the **super** construct, which we have already encountered in the context of constructors in Chapter 10. Code 11.2 illustrates this idea with the **display** method of the **PhotoPost** class.

Code 11.2
Redefining method with a super call

```
public void display()
{
    super.display();
    System.out.println("   [" + filename + "]");
    System.out.println("   " + caption);
}
```

When **display** is now called on a **PhotoPost** object, initially the **display** method in the **PhotoPost** class will be invoked. As its first statement, this method will in turn invoke the **display** method of the superclass, which prints out the general post information. When control returns from the superclass method, the remaining statements of the subclass method print the distinctive fields of the **PhotoPost** class.

There are three details worth noting:

- Contrary to the case of **super** calls in constructors, the method name of the superclass method is explicitly stated. A **super** call in a method always has the form

  super.*method-name*(*parameters*)

- The parameter list can, of course, be empty.
- Again, contrary to the rule for **super** calls in constructors, the **super** call in methods may occur anywhere within that method. It does not have to be the first statement.
- And contrary to the case of **super** calls in constructors, no automatic **super** call is generated and no **super** call is required; it is entirely optional. So the default behavior gives the effect of a subclass method completely hiding (i.e., overriding) the superclass version of the same method.

Exercise 11.3 Modify your latest version of the *network* project to include the super call in the **display** method. Test it. Does it behave as expected? Do you see any problems with this solution?