Exercise 11.2 In your *network* project, add a **`display`** method in class **`Post`** again. For now, write the method body with a single statement that prints out only the username. Then modify the **`display`** methods in **`MessagePost`** and **`PhotoPost`** so that the **`MessagePost`** version prints out only the message and the **`PhotoPost`** version prints only the caption. This removes the other errors encountered above (we shall come back to those below).

You should now have a situation corresponding to Figure 11.4, with **`display`** methods in three classes. Compile your project. (If there are errors, remove them. This design should work.)

Before executing, predict which of the **`display`** methods will get called if you execute the news feed's **`show`** method.

Try it out. Enter a message post and a photo post into the news feed and call the news feed's **`show`** method. Which **`display`** methods were executed? Was your prediction correct? Try to explain your observations.

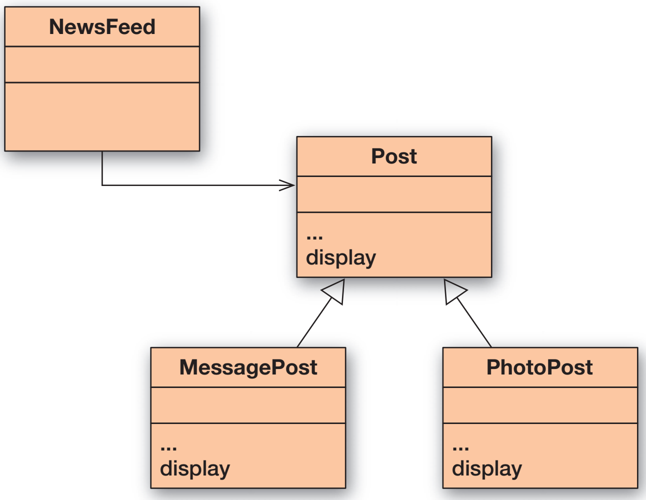

Figure 11.4
Display, version 3: **`display`** method in subclasses and superclass

11.3 Overriding

The next design we shall discuss is one where both the superclass and the subclasses have a **`display`** method (Figure 11.4). The header of all the **`display`** methods is exactly the same.

Code 11.1 shows the relevant details of the source code of all three classes. Class **`Post`** has a **`display`** method that prints out all the fields that are declared in **`Post`** (those common to message posts and photo posts), and the subclasses **`MessagePost`** and **`PhotoPost`** print out the fields specific to **`MessagePost`** and **`PhotoPost`** objects, respectively.

Code 11.1
Source code of the **`display`** methods in all three classes

```
public class Post
{
    ...
    public void display()
    {
        System.out.println(username);
        System.out.print(timeString(timestamp));

        if(likes > 0) {
            System.out.println("  -  " + likes + " people like this.");
        }
        else {
            System.out.println();
        }

        if(comments.isEmpty()) {
            System.out.println("   No comments.");
        }
        else {
            System.out.println("   " + comments.size() +
                               " comment(s). Click here to view.");
        }
    }
}
```

```
public class MessagePost extends Post
{
    ...
    public void display()
    {
        System.out.println(message);
    }
}
```

```
public class PhotoPost extends Post
{
    ...
    public void display()
    {
        System.out.println("  [" + filename + "]");
        System.out.println("  " + caption);
    }
}
```

Concept

Overriding A subclass can override a method implementation. To do this, the subclass declares a method with the same signature as the superclass, but with a different method body. The overriding method takes precedence for method calls on subclass objects.

This design works a bit better. It compiles, and it can be executed, even though it is not perfect yet. An implementation of this design is provided in the project *network-v3*. (If you have done Exercise 11.2, you already have a similar implementation of this design in your own version.)