CHAPTER

11 More about Inheritance

Main concepts discussed in this chapter:

- method polymorphism
- overriding
- static and dynamic type
- dynamic method lookup

Java constructs discussed in this chapter:

`super` (in method), **`toString`**, **`protected`**, **`instanceof`**

The last chapter introduced the main concepts of inheritance by discussing the *network* example. While we have seen the foundations of inheritance, there are still numerous important details that we have not yet investigated. Inheritance is central to understanding and using object-oriented languages, and understanding it in detail is necessary to progress from here.

In this chapter, we shall continue to use the *network* example to explore the most important of the remaining issues surrounding inheritance and polymorphism.

11.1 The problem: *network's* display method

When you experimented with the *network* examples in Chapter 10, you probably noticed that the second version—the one using inheritance—has a problem: the **`display`** method does not show all of a post's data.

Let us look at an example. Assume that we create a **`MessagePost`** and a **`PhotoPost`** object with the following data:

The message post:

> Leonardo da Vinci
>
> Had a great idea this morning.
> But now I forgot what it was. Something to do with flying . . .
>
> 40 seconds ago. 2 people like this.
>
> No comments.

The photo post:

```
Alexander Graham Bell
[experiment.jpg]    I think I might call this thing 'telephone'.
12 minutes ago. 4 people like this.
No comments.
```

If we enter these objects into the news feed[1] and then invoke the first version of the news feed's **show** method (the one without inheritance), it prints

```
Leonardo da Vinci
Had a great idea this morning.
But now I forgot what it was. Something to do with flying . . .
40 seconds ago - 2 people like this.
  No comments.

Alexander Graham Bell
[experiment.jpg]
I think I might call this thing 'telephone'.
12 minutes ago - 4 people like this.
  No comments.
```

While the formatting isn't pretty (because, in the text terminal, we don't have formatting options available), all the information is there, and we can imagine how the **show** method might be adapted later to show the data in a nicer formatting in a different user interface.

Compare this with the second *network* version (with inheritance), which prints only

```
Leonardo da Vinci
40 seconds ago - 2 people like this.
  No comments.

Alexander Graham Bell
12 minutes ago - 4 people like this.
  No comments.
```

We note that the message post's text, as well as the photo post's image filename and caption, are missing. The reason for this is simple. The **display** method in this version is implemented in the **Post** class, not in **MessagePost** and **PhotoPost** (Figure 11.1). In the methods of **Post**, only the fields declared in **Post** are available. If we tried to access the **MessagePost**'s **message** field from **Post**'s **display** method, an error would be reported. This illustrates the important principle that inheritance is a one-way street: **MessagePost** inherits the fields of **Post**, but **Post** still does not know anything about fields in its subclasses.

[1] The text for the message post is a two-line string. You can enter a multiline text into a string by using "\n" in the string for the line break.