10.4 Inheritance in Java

Before discussing more details of inheritance, we will have a look at how inheritance is expressed in the Java language. Here is a segment of the source code of the **Post** class:

```
public class Post
{
    private String username; // username of the post's author
    private long timestamp;
    private int likes;
    private ArrayList<String> comments;

    // Constructors and methods omitted.
}
```

There is nothing special about this class so far. It starts with a normal class definition and defines **Post**'s fields in the usual way. Next, we examine the source code of the **MessagePost** class:

```
public class MessagePost extends Post
{
    private String message;

    // Constructors and methods omitted.
}
```

There are two things worth noting here. First, the keyword **extends** defines the inheritance relationship. The phrase "**extends Post**" specifies that this class is a subclass of the **Post** class. Second, the **MessagePost** class defines only those fields that are unique to **MessagePost** objects (only **message** in this case). The fields from **Post** are inherited and do not need to be listed here. Objects of class **MessagePost** will nonetheless have fields for **username**, **timestamp**, and so on.

Next, let us have a look at the source code of class **PhotoPost**:

```
public class PhotoPost extends Post
{
    private String filename;
    private String caption;

    // Constructors and methods omitted.
}
```

This class follows the same pattern as the **MessagePost** class. It uses the **extends** keyword to define itself as a subclass of **Post** and defines its own additional fields.

10.4.1 Inheritance and access rights

To objects of other classes, **MessagePost** or **PhotoPost** objects appear just like all other types of objects. As a consequence, members defined as **public** in either the superclass or subclass portions will be accessible to objects of other classes, but members defined as **private** will be inaccessible.

In fact, the rule on privacy also applies between a subclass and its superclass: a subclass cannot access private members of its superclass. It follows that if a subclass method needed

Copyright © 2016. Pearson Education, Limited. All rights reserved.

Barnes, David, and Michael Kolling. Objects First with Java: A Practical Introduction Using BlueJ, Global Edition, Pearson Education, Limited, 2016. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/strath/detail.action?docID=5186172.
Created from strath on 2024-12-09 12:42:35.

to access or change private fields in its superclass, then the superclass would need to provide appropriate accessor and/or mutator methods. However, an object of a subclass may call any public methods defined in its superclass as if they were defined locally in the subclass—no variable is needed, because the methods are all part of the same object.

This issue of access rights between super- and subclasses is one we will discuss further in Chapter 11, when we introduce the **`protected`** modifier.

Exercise 10.4 Open the project *network-v2*. This project contains a version of the *network* application, rewritten to use inheritance, as described above. Note that the class diagram displays the inheritance relationship. Open the source code of the **`MessagePost`** class and remove the "**`extends Post`**" phrase. Close the editor. What changes do you observe in the class diagram? Add the "**`extends Post`**" phrase again.

Exercise 10.5 Create a **`MessagePost`** object. Call some of its methods. Can you call the inherited methods (for example, **`addComment`**)? What do you observe about the inherited methods?

Exercise 10.6 In order to illustrate that a subclass can access non-private elements of its superclass without any special syntax, try the following slightly artificial modification to the **`MessagePost`** and **`Post`** classes. Create a method called **`printShortSummary`** in the **`MessagePost`** class. Its task is to print just the phrase "Message post from *NAME*", where *NAME* should show the name of the author. However, because the **`username`** field is private in the **`Post`** class, it will be necessary to add a public **`getUserName`** method to **`Post`**. Call this method from **`printShortSummary`** to access the name for printing. Remember that no special syntax is required when a subclass calls a superclass method. Try out your solution by creating a **`MessagePost`** object. Implement a similar method in the **`PhotoPost`** class.

10.4.2 Inheritance and initialization

When we create an object, the constructor of that object takes care of initializing all object fields to some reasonable state. We have to look more closely at how this is done in classes that inherit from other classes.

When we create a **`MessagePost`** object, we pass two parameters to the message post's constructor: the name of the author and the message text. One of these contains a value for a field defined in class **`Post`**, and the other a value for a field defined in class **`MessagePost`**. All of these fields must be correctly initialized, and Code 10.4 shows the code segments that are used to achieve this in Java.

Several observations can be made here. First, the class **`Post`** has a constructor, even though we do not intend to create an instance of class **`Post`** directly.[2] This constructor receives the

[2] Currently, there is nothing actually preventing us from creating a **`Post`** object, although that was not our intention when we designed these classes. In Chapter 12, we shall see some techniques that allow us to make sure that **`Post`** objects cannot be created directly, but only **`MessagePost`** or **`PhotoPost`** objects.

Copyright © 2016. Pearson Education, Limited. All rights reserved.

Barnes, David, and Michael Kolling. Objects First with Java: A Practical Introduction Using BlueJ, EBook, Global Edition, Pearson Education, Limited, 2016. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/strath/detail.action?docID=5186172.
Created from strath on 2024-12-09 12:42:35.

Code 10.4
Initialization of subclass and superclass fields

```
public class Post
{
    private String username;  // username of the post's author
    private long timestamp;
    private int likes;
    private ArrayList<String> comments;

    /**
     * Constructor for objects of class Post.
     *
     * @param author    The username of the author of this post.
     */
    public Post(String author)
    {
        username = author;
        timestamp = System.currentTimeMillis();
        likes = 0;
        comments = new ArrayList<>();
    }

    Methods omitted.
}
```

```
public class MessagePost extends Post
{
    private String message;  // an arbitrarily long, multi-line message

    /**
     * Constructor for objects of class MessagePost.
     *
     * @param author    The username of the author of this post.
     * @param text      The text of this post.
     */
    public MessagePost(String author, String text)
    {
        super(author);
        message = text;
    }

    Methods omitted.
}
```

Copyright © 2016. Pearson Education, Limited. All rights reserved.

parameters needed to initialize the **`Post`** fields, and it contains the code to do this initialization. Second, the **`MessagePost`** constructor receives parameters needed to initialize both **`Post`** and **`MessagePost`** fields. It then contains the following line of code:

```
super(author);
```

Barnes, David, and Michael Kolling. Objects First with Java: a Practical Introduction Using BlueJ, EBook, Global Edition, Pearson Education, Limited, 2016. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/strath/detail.action?docID=5186172.
Created from strath on 2024-12-09 12:42:35.