

CHAPTER

8

Inheritance

Inheritance is one of the cornerstones of object-oriented programming because it allows the creation of hierarchical classifications. Using inheritance, you can create a general class that defines traits common to a set of related items. This class can then be inherited by other, more specific classes, each adding those things that are unique to it. In the terminology of Java, a class that is inherited is called a *superclass*. The class that does the inheriting is called a *subclass*. Therefore, a subclass is a specialized version of a superclass. It inherits all of the members defined by the superclass and adds its own, unique elements.

Inheritance Basics

To inherit a class, you simply incorporate the definition of one class into another by using the **extends** keyword. To see how, let's begin with a short example. The following program creates a superclass called **A** and a subclass called **B**. Notice how the keyword **extends** is used to create a subclass of **A**.

```
// A simple example of inheritance.

// Create a superclass.
class A {
    int i, j;

    void showij() {
        System.out.println("i and j: " + i + " " + j);
    }
}

// Create a subclass by extending class A.
class B extends A {
    int k;

    void showk() {
        System.out.println("k: " + k);
    }
}
```

```

    void sum() {
        System.out.println("i+j+k: " + (i+j+k));
    }
}

class SimpleInheritance {
    public static void main(String args []) {
        A superOb = new A();
        B subOb = new B();

        // The superclass may be used by itself.
        superOb.i = 10;
        superOb.j = 20;
        System.out.println("Contents of superOb: ");
        superOb.showij();
        System.out.println();

        /* The subclass has access to all public members of
           its superclass. */
        subOb.i = 7;
        subOb.j = 8;
        subOb.k = 9;
        System.out.println("Contents of subOb: ");
        subOb.showij();
        subOb.showk();
        System.out.println();

        System.out.println("Sum of i, j and k in subOb:");
        subOb.sum();
    }
}

```

The output from this program is shown here:

```

Contents of superOb:
i and j: 10 20

```

```

Contents of subOb:
i and j: 7 8
k: 9

```

```

Sum of i, j and k in subOb:
i+j+k: 24

```

As you can see, the subclass **B** includes all of the members of its superclass, **A**. This is why **subOb** can access **i** and **j** and call **showij()**. Also, inside **sum()**, **i** and **j** can be referred to directly, as if they were part of **B**.

Even though **A** is a superclass for **B**, it is also a completely independent, stand-alone class. Being a superclass for a subclass does not mean that the superclass cannot be used by itself. Further, a subclass can be a superclass for another subclass.

The general form of a **class** declaration that inherits a superclass is shown here:

```
class subclass-name extends superclass-name {
    // body of class
}
```

You can only specify one superclass for any subclass that you create. Java does not support the inheritance of multiple superclasses into a single subclass. You can, as stated, create a hierarchy of inheritance in which a subclass becomes a superclass of another subclass. However, no class can be a superclass of itself.

Member Access and Inheritance

Although a subclass includes all of the members of its superclass, it cannot access those members of the superclass that have been declared as **private**. For example, consider the following simple class hierarchy:

```
/* In a class hierarchy, private members remain
   private to their class.

   This program contains an error and will not
   compile.
*/

// Create a superclass.
class A {
    int i; // public by default
    private int j; // private to A

    void setij(int x, int y) {
        i = x;
        j = y;
    }
}

// A's j is not accessible here.
class B extends A {
    int total;

    void sum() {
        total = i + j; // ERROR, j is not accessible here
    }
}

class Access {
    public static void main(String args[]) {
        B subOb = new B();

        subOb.setij(10, 12);

        subOb.sum();
        System.out.println("Total is " + subOb.total);
    }
}
```

This program will not compile because the use of `j` inside the `sum()` method of `B` causes an access violation. Since `j` is declared as `private`, it is only accessible by other members of its own class. Subclasses have no access to it.

REMEMBER A class member that has been declared as `private` will remain private to its class. It is not accessible by any code outside its class, including subclasses.

A More Practical Example

Let's look at a more practical example that will help illustrate the power of inheritance. Here, the final version of the `Box` class developed in the preceding chapter will be extended to include a fourth component called `weight`. Thus, the new class will contain a box's width, height, depth, and weight.

```
// This program uses inheritance to extend Box.
class Box {
    double width;
    double height;
    double depth;

    // construct clone of an object
    Box(Box ob) { // pass object to constructor
        width = ob.width;
        height = ob.height;
        depth = ob.depth;
    }

    // constructor used when all dimensions specified
    Box(double w, double h, double d) {
        width = w;
        height = h;
        depth = d;
    }

    // constructor used when no dimensions specified
    Box() {
        width = -1; // use -1 to indicate
        height = -1; // an uninitialized
        depth = -1; // box
    }

    // constructor used when cube is created
    Box(double len) {
        width = height = depth = len;
    }

    // compute and return volume
    double volume() {
        return width * height * depth;
    }
}

// Here, Box is extended to include weight.
class BoxWeight extends Box {
```

```

double weight; // weight of box

// constructor for BoxWeight
BoxWeight(double w, double h, double d, double m) {
    width = w;
    height = h;
    depth = d;
    weight = m;
}

class DemoBoxWeight {
    public static void main(String args[]) {
        BoxWeight mybox1 = new BoxWeight(10, 20, 15, 34.3);
        BoxWeight mybox2 = new BoxWeight(2, 3, 4, 0.076);
        double vol;

        vol = mybox1.volume();
        System.out.println("Volume of mybox1 is " + vol);
        System.out.println("Weight of mybox1 is " + mybox1.weight);
        System.out.println();

        vol = mybox2.volume();
        System.out.println("Volume of mybox2 is " + vol);
        System.out.println("Weight of mybox2 is " + mybox2.weight);
    }
}

```

The output from this program is shown here:

```

Volume of mybox1 is 3000.0
Weight of mybox1 is 34.3

```

```

Volume of mybox2 is 24.0
Weight of mybox2 is 0.076

```

BoxWeight inherits all of the characteristics of **Box** and adds to them the **weight** component. It is not necessary for **BoxWeight** to re-create all of the features found in **Box**. It can simply extend **Box** to meet its own purposes.

A major advantage of inheritance is that once you have created a superclass that defines the attributes common to a set of objects, it can be used to create any number of more specific subclasses. Each subclass can precisely tailor its own classification. For example, the following class inherits **Box** and adds a color attribute:

```

// Here, Box is extended to include color.
class ColorBox extends Box {
    int color; // color of box

    ColorBox(double w, double h, double d, int c) {
        width = w;
        height = h;
        depth = d;
        color = c;
    }
}

```

Remember, once you have created a superclass that defines the general aspects of an object, that superclass can be inherited to form specialized classes. Each subclass simply adds its own unique attributes. This is the essence of inheritance.

A Superclass Variable Can Reference a Subclass Object

A reference variable of a superclass can be assigned a reference to any subclass derived from that superclass. You will find this aspect of inheritance quite useful in a variety of situations. For example, consider the following:

```
class RefDemo {
    public static void main(String args[]) {
        BoxWeight weightbox = new BoxWeight(3, 5, 7, 8.37);
        Box plainbox = new Box();
        double vol;

        vol = weightbox.volume();
        System.out.println("Volume of weightbox is " + vol);
        System.out.println("Weight of weightbox is " +
            weightbox.weight);
        System.out.println();

        // assign BoxWeight reference to Box reference
        plainbox = weightbox;

        vol = plainbox.volume(); // OK, volume() defined in Box
        System.out.println("Volume of plainbox is " + vol);

        /* The following statement is invalid because plainbox
           does not define a weight member. */
        // System.out.println("Weight of plainbox is " + plainbox.weight);
    }
}
```

Here, **weightbox** is a reference to **BoxWeight** objects, and **plainbox** is a reference to **Box** objects. Since **BoxWeight** is a subclass of **Box**, it is permissible to assign **plainbox** a reference to the **weightbox** object.

It is important to understand that it is the type of the reference variable—not the type of the object that it refers to—that determines what members can be accessed. That is, when a reference to a subclass object is assigned to a superclass reference variable, you will have access only to those parts of the object defined by the superclass. This is why **plainbox** can't access **weight** even when it refers to a **BoxWeight** object. If you think about it, this makes sense, because the superclass has no knowledge of what a subclass adds to it. This is why the last line of code in the preceding fragment is commented out. It is not possible for a **Box** reference to access the **weight** field, because **Box** does not define one.

Although the preceding may seem a bit esoteric, it has some important practical applications—two of which are discussed later in this chapter.