第5章

深 入 类

Java 的继承性可以实现软件重用，Java 的多态性可以使程序增加新功能变得容易，Java 的接口可以实现多重继承的功能，通过接口还可以将设计与实现分离，使编写的代码更通用，更加容易维护。包是类和接口的集合。利用包可以把用户常用的类、功能相似的类与具有公共变量与空方法的接口放在一个包中。

本章的内容主要解决以下问题：

- 什么是父类与子类？
- 如何创建子类？
- 在同一个应用程序中如何区别子类与父类的成员？
- 如何隐藏父类的成员变量？
- 如何覆盖父类的成员方法？
- 什么是接口？它有什么特点？
- 如何创建自定义接口？
- 如何引用包及包中的类？
- 如何创建自定义包？如何在包中存放类？

5.1 类的继承性

新类可从现有的类中产生，将保留现有类的状态属性和方法并可根据需要加以修改。新类还可添加新的状态属性和方法，这些新增功能允许以统一的风格处理不同类型的数据。这种现象就称为类的继承。例如从小狗类中继承的新类除了有小狗原来的特性以外，还可以增加颜色状态和发声的方法等。

本节主要介绍声明继承父类的子类的方式、从父类继承成员变量与成员方法的方式、隐藏父类成员变量的方法、覆盖父类成员方法的方法、this 和 super 的作用。

5.1.1 类的层次关系

1. 父类与子类的关系

当建立一个新类时，不必写出全部成员变量和成员方法。只要简单地声明这个类是从一个已定义的类继承下来的，就可以引用被继承类的全部成员。

Java 提供了一个庞大的类库让开发人员继承和使用。设计这些类是出于公用的目的，因此，很少有某个类恰恰满足用户的需要。用户必须自己设计能处理实际问题的类，如果新设计的类仅仅实现了继承，则和父类毫无两样。所以，通常要对子类进行扩展，即添加新的属性和方法，使得子类比父类大，更具特殊性，代表着一组更具体的对象，有句话说“青出于蓝而胜于蓝”，继承的意义就在于此。

2. 声明子类的方式

声明子类的语法格式：

```
[修饰词] class 子类名 extends 父类名
```

可见，在类的声明语句中加入 extends 关键字，然后指定父类名即可通过继承父类声明一个子类，例如：

```
public class Animal extends Object
public class Dog extends Animal
public class Person
```

上述第 1 条语句声明子类 Animal 的父类是 Object，但编程中通常会省略 extends Object 子句；第 2 条语句声明子类 Dog 的父类是 Animal；第 3 条语句在字面上没有 extends 关键字，但实际上等价于 public class Person extends Object。

3. Java 类的层次结构

Java 类的继承是从什么地方开始的？又是如何延续下来的呢？看图 5.1 所示的模拟图。

图 5.1 反映了 Java 类的层次结构。最顶端的类是 Object，它在 java.lang 中定义，是所有类的始祖。一个类可以有多个子类，也可以没有子类，但它必定有一个父类（Object 除外）。

子类不能继承父类中的 private 成员，除此之外，其他所有的成员都可以通过继承变为子类的成员。对继承的理解可以扩展到整个父类的分支，也就是说，子类继承的成员实际上是整个父系的所有成员。例如，toString 这个方法是在 Object 中声明的，被层层继承了下来，所有的子孙类都可以继承，使用这个方法输出当前对象的基本信息。

至此，可以得出如下结论：

（1）子类只能有一个父类。

（2）如果省略了 extends，子类的父类是 Object。

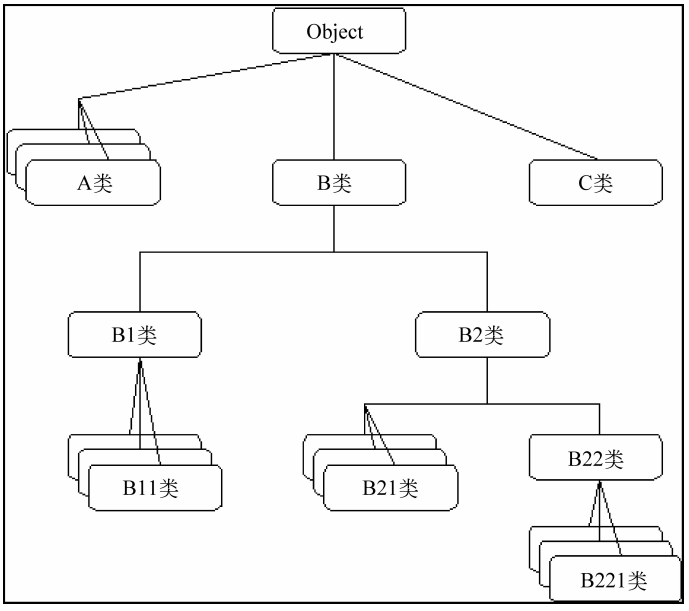

图 5.1　Java 类的层次结构

(3) 子类继承了父类和祖先的成员,可以使用这些成员。

(4) 子类可以添加新的成员变量和方法,也可以隐藏父类的成员变量或覆盖父类的成员方法。

5.1.2　成员变量的继承和隐藏

1. 成员变量的继承

子类中的成员变量可以是子类自己创建的,其他是通过其父类继承的。子类继承根据二者所在包的异同分为以下两种情况:

(1) 子类与父类在同一包,子类继承父类中非 private 修饰的成员变量和方法。

(2) 子类与父类不在同一包,父类中使用 private 与 package 修饰的私有成员变量和友好的成员变量不会被继承,子类只能继承父类中使用 public 和 protected 的成员变量与方法。

例 5.1　下面的 3 个程序说明从点 Point 类扩展到线 Line 类和圆 Circle 类的方法,这是 3 个公共类,不能放在同一个文件中。它们属于方法类,仅用来定义一个类,可以进行编译处理。它们都没有 main 方法及输出语句,如果运行程序看不到什么结果。

```
public class Point {
    protected int x, y;
    Point(int a, int b) {setPoint(a, b);}       //构造方法
    Point() {}
    public void setPoint(int a, int b) { x=a; y=b; }
    public int getX() {return x;}
```

```
    public int getY() {return y;}
}

public class Line extends Point {
    protected int x, y, endX, endY;
    Line(int x1, int y1, int x2, int y2) {}     //构造方法
    setLine(x1, y1, x2, y2);
    public void setLine(int x1, int y1, int x2, int y2) { x=x1; y=y1; endX=x2;
    endY=y2; }
    public int getX() {return x;}
    public int getY() {return y;}
    public int getEndX() {return endX;}
    public int getEndY() {return endY;}
    public double length() {
        return Math.sqrt((endX-x) * (endX-x)+(endY-y) * (endY-y));
    }
}

public class Circle extends Point {
    protected int radius;
    Circle(int a, int b, int r) {
        super(a, b);
        setRadius(r);
    }       //构造方法
    public void setRadius(int r) {radius=r;}
    public int getRadius() {return radius;}
    public double area() {return 3.14159 * radius * radius;}
}
```

说明：Point 类具备一个点的特征。Line 和 Circle 类具备线和圆的特征，它们是从 Point 继承下来的子类。

下面分析这 3 个类各自都有哪些成员。

Point：

```
x, y                        //受保护的成员变量,代表点的坐标
Point                       //点的构造方法
setPoint                    //设定点的坐标值的方法
getX, getY                  //返回坐标 x 和 y 的值的方法
```

Line：

```
x, y, endX, endY            //子类受保护的成员变量,代表线的两个端点坐标
Line                        //线的构造方法
setLine                     //设定线的两个端点坐标值的方法
getX, getY                  //返回起点坐标 x 和 y 的值的方法
```

```
getEndX, getEndY          //返回终点坐标 endX 和 endY 的值的方法
length                    //返回线的长度的方法
x, y                      //继承父类的受保护成员变量,但被子类隐藏
getX, getY                //继承父类的方法,但被子类覆盖
```

Circle：

```
radius                    //子类受保护的成员变量,代表圆的半径
Circle                    //圆的构造方法
setRadius                 //设定半径值的方法
getRadius                 //返回半径值的方法
area                      //返回圆面积的方法
x, y                      //继承父类的受保护成员变量
super                     //调用父类的构造方法
getX, getY                //继承父类的方法
```

2. 成员变量的隐藏

成员变量的隐藏是指子类重新定义了父类中的同名变量，如子类 Line 重新定义了 x 为 x1，y 为 y1，隐藏了父类 Point 中的两个成员变量 x 和 y。子类执行自己的方法时，操作的是子类的变量；子类执行父类的方法时，操作的是父类的变量。在子类中要特别注意成员变量的命名，防止无意中隐藏了父类的关键成员变量，这有可能给子类使用变量带来麻烦。

Line 还覆盖了 Point 的两个方法 getX 和 getY。关于方法覆盖的内容，参见下一小节。

5.1.3 成员方法的继承与覆盖

如果子类的方法与父类的方法同名，则不会继承父类的方法而用子类的方法，此时称子类的方法覆盖了父类的方法，简称为方法覆盖(override)。方法覆盖为子类提供了修改父类成员方法的能力。例如，子类可以修改层层继承下来的 toString 方法，让它输出一些更有用的信息。

例 5.2 子类对 Object 类 toString 方法的覆盖，结果如图 5.2 所示。

```
class Circle2 {
    private int radius;
    Circle2(int r){setRadius(r);}
    public void setRadius(int r){radius=r;}
    public int getRadius(){return radius;}
    public double area(){
        return 3.14159 * radius * radius;
    }
    public String toString(){
        return "圆半径: "+getRadius()+"圆面积:"+area();
    }
```

```
}

public class exp5_2 {
    public static void main(String args[]){
        Circle2 c=new Circle2(10);
        System.out.println("\n"+c.toString());
    }
}
```

```
圆半径：10圆面积:314.159
```

图 5.2　成员方法覆盖

说明：程序中改写了上一节介绍的 Circle 类定义了 Circle2 类，在其中添加了 toString 方法并修改了它的返回值。由于 toString 和父类 Object 的 toString 方法名相同、返回值类型相同，因此就覆盖了父类中的 toString 方法。

1. 方法覆盖注意问题

(1) 用来覆盖的子类方法应和被覆盖的父类方法同名、同返回值类型、相同参数个数和参数类型。如果被覆盖的方法没有声明抛出异常，子类的覆盖方法可以有不同的抛出异常子句。

(2) 可以部分覆盖一个方法。部分覆盖是在原方法的基础上添加新的功能，即在子类的覆盖方法中添加一条语句：super.原父类方法名，然后加入其他语句。

(3) 不能覆盖父类中的 final 方法，因为设计这类方法的目的就是为了防止覆盖。

(4) 不能覆盖父类中的 static 方法，但可以隐藏这类方法。可在子类中声明同名的静态方法来隐藏父类中的静态方法。

(5) 子类必须覆盖父类中的抽象方法。

2. 子类继承父类的原则

子类可以继承父类中所有可被子类访问的成员变量和成员方法，但必须遵循以下原则：

(1) 子类能够继承父类中被声明为 public 和 protected 的成员变量和成员方法，但不能继承被声明为 private 的成员变量和成员方法；

(2) 子类能够继承在同一个包中的由默认修饰符修饰的成员变量和成员方法；

(3) 如果子类声明了一个与父类的成员变量同名的成员变量，则子类不能继承父类的成员变量，此时称子类的成员变量隐藏了父类的成员变量；

(4) 如果子类声明了一个与父类的成员方法同名、同返回值类型、相同参数个数和参数类型的成员方法，则子类不能继承父类的成员方法，此时称子类的成员方法覆盖了父类的成员方法。

5.1.4　This 和 super 关键字

1. this 和 super 的应用方式

在什么情况下使用 this 和 super？看下面的例子。

例 5.3 运用 super 关键字实现子类调用父类的构造方法，用 this 关键字实现当前类的对象调用成员变量或成员方法。运行结果如图 5.3 所示。

```
class Point2 {                          //父类
    protected int x,y;
    Point2(int a,int b){
        setPoint(a,b);
    }
    public void setPoint(int a,int b){
        x=a;y=b;
    }
}
class Line2 extends Point2 {            //子类
    protected int x,y;
    Line2(int a,int b){super(a,b);setLine(a,b);}
    public void setLine(int x,int y){
        this.x=x+x;this.y=y+y;
    }
    public double length(){
        int x1=super.x, y1=super.y,x2=this.x,y2=this.y;
        return Math.sqrt((x2-x1) * (x2-x1)+(y2-y1) * (y2-y1));
    }
    public String toString(){
        return "直线端点：["+super.x+","+super.y+"]["+x+","+y+"]直线长度:"+
this.length();
    }
}
public class exp5_3{                    //主类
    public static void main(String args[]) {
        Line2 line=new Line2(50, 50);
        System.out.println("\n"+line.toString());
    }
}
```

```
直线端点：[50,50][100,100]直线长度:70.71067811865476
```

图 5.3 直线端点和长度输出结果

说明：

(1) 在例 5.3 创建的 Point2 类中定义了成员变量 x 和 y，其子类 Line2 中也定义了成员变量 x 和 y，其构造方法 Line2()中也用 x 和 y 作为参数，这种变量同名多次引发了变量的隐藏。首先是 Line2 中的 x 和 y 隐藏了 Point2 中的 x 和 y，然后 setLine 方法中的参数 x 和 y 又隐藏了 Line2 的 x 和 y。使用这些变量时，如果分不清它们是属于谁的，就会带来混乱。

(2) Point2 中已经有了两个变量表示点的位置，子类 Line2 只需再定义两个变量表示另一个点的位置。计算线的长度时，可用这 4 个变量来表示一条直线的两个端点。因此，在构造方法中将代表初值的两个参数分别使用：通过 super(a,b)调用父类的构造方法为父类的 x 和 y 赋值；通过 setLine(a,b)为子类的 x 和 y 赋值。

(3) 在 setLine 方法中，因为参数名和成员变量名相同，如果为成员变量赋值，就必须加上 this 引用，告诉编译器是为当前类的成员变量赋值。同理，在 length 和 toString 方法中使用父类成员变量时，就必须加上 super 引用，告诉编译器使用的是父类的成员变量。

(4) 设计这个程序的目的是为了说明 This、super 和 super()的使用方法，如果不用变量隐藏，或者在子类中定义足够多的成员变量，则不需要 this 和 super，但这样就发挥不了继承的优势。

例 5.4 在子类继承父类的情况下用 super 关键字调用父类的同名方法和变量。运行结果如图 5.4 所示。

```
class ClassA {                          //父类
    boolean var;
    void method(){var=true;}
}
public class exp5_4 extends ClassA {   //子类
    boolean var;
    void method(){
        var=false;
        super.method();
        System.out.println("子类变量 var 为 "+var);
        System.out.println("父类变量 var 为 "+super.var);
    }
    public static void main(String args[]){
        exp5_4 a=new exp5_4();
        a.method();
    }
}
```

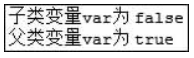

图 5.4 子类和父类变量的输出结果

说明：exp5_4 子类中隐藏了 ClassA 的成员变量 var，将 var 赋值为 false，但调用 super.method()后，又将 var 赋值为 true，因此输出结果分别是 false 和 true。这里使用了构造方法 exp5_4()，但在 exp5_4 类中没有定义，其原因将在后面解释。

2. this 和 super 的作用

总结上面的例子，可以概括出 this 和 super 的作用如下。

(1) this 可以代表当前类或对象本身。

在一个类中，this 表示对当前类的引用，在使用类的成员时隐含着 this 引用，尽管可

以不明确地写出，例如 length 和 toString 中对 x 和 y 的使用。当一个类被实例化为一个对象时，this 就是对象名的另一种表示。通过 this 可顺利地访问对象，凡在需要使用对象名的地方均可用 this 代替。

(2) super 代表着父类。

如果子类的变量隐藏了父类的变量，使用不加引用的变量一定是子类的变量，如果使用父类的变量，就必须加上 super 引用。同样的道理，如果有方法覆盖的发生，调用父类的方法时也必须加上 super 引用。

(3) super() 可用来调用父类的构造方法。

Java 规定类的构造方法只能由 new 操作符调用，但子类可以使用 super()调用父类的构造方法。同理，this() 也可用来间接调用当前类或对象的构造方法。

类的构造方法是不能继承的，因为构造方法不是类的成员，没有返回值，也不需要修饰符。因为父类的构造方法和父类同名，在子类中继承父类的构造方法肯定和子类不同名，这样的继承是无意义的。所以，要在子类中调用父类的构造方法，需要使用 super()，如在例 5.3 中：

```
class Line2 extends Point2 {
protected int x,y;
    Line2(int a,int b) {
        super(a,b);
        setLine(a,b);
    }
}
```

其中 super(a,b)完成了在子类中调用父类构造方法的任务，即为父类的成员变量赋初值。

注意：super()只能出现在子类的构造方法中，而且必须是子类构造方法中的第一条可执行语句。

5.2 类的多态性

类的继承发生在多个类之间，而类的多态发生在同一个类上。在一个类中，可以定义多个同名的方法，但方法的参数个数和数据类型是不同的。这种现象称为类的多态性。

本节主要介绍实现类多态性的成员方法的重载与构造方法的重载。

5.2.1 成员方法的重载

方法的重载是指对同名方法的不同定义方式。看下面这个例子。

例 5.5 本程序用来定义一个类，在该类中定义了名称为 getArea()的方法(参数为可变的)和两个名称为 draw()的方法，用来说明方法的重载特性，运行结果如图 5.5 所示。

```
长为20.0 宽为30.0的矩形的面积是：600.0
半径为7.0的圆的面积是：153.93803
画7个任意形状的图形
画一个三角形
```

图 5.5 成员方法重载运行结果

```
public class exp5_5 {
    final float PI=3.1415926f;
    public void getArea(float r){
        float area=PI * r * r;
        System.out.println("半径为"+r+"的圆的面积是: "+area);
    }
    public void getArea(float a,float b){
        float area=a * b;
        System.out.println("长为"+a+" 宽为"+b+"的矩形的面积是: "+area);
    }
    public void draw(int num){ System.out.println("画"+num+"个任意形状的图
    形");}
    public void draw(String shape){ System.out.println("画一个"+shape); }
    public static void main(String[] args) {
        exp5_5 cal=new exp5_5();
        cal.getArea(20, 30);
        cal.getArea(7);
        cal.draw(7);
        cal.draw("三角形");
    }
}
```

说明:

在本例中,向同名方法 getArea()与 draw()输入不同种类的参数,都得到了正确的输出,它们的区别在于参数的个数和类型的差别,Java 解释器在运行这个程序时,可以根据参数的不同来调用不同的方法。

成员方法的多态性使类能够向外提供一个较为一致的接口,对程序员来说,不必关心同名方法内部的细节差别,只要掌握它们在使用时参数的个数和类型就可以了。

5.2.2 构造方法的重载

每一个类都有一个默认的构造方法,这就是和类同名的无参构造方法。它实际上是父类的构造方法,创建子类时由父类自动提供。因此,每个类的对象都可以使用这种方式来初始化对象。

如果在初始化对象时需要对象具有更多的特性,可重载构造方法。看下面的例子。

例 5.6 在类中定义 3 个不同的构造方法并验证其结果,运行结果如图 5.6 所示。

```
class RunDemo {
    private String userName, password;
    RunDemo(){ System.out.println("All is null!"); }     //无参构造方法
    RunDemo(String name){ userName=name; }               //一个参数的构造方法
    RunDemo(String name, String pwd){                    //两个参数的构造方法
        this(name);
        password=pwd;
```