第 3 章

类的重用

软件重用是提高软件开发效率及软件质量的有效途径，面向对象的分析和编程技术为软件重用提供了新的手段。面向对象开发方法得以流行的原因之一就是基于类和对象的重用比传统程序的重用更容易。本章将介绍与类的重用有关的内容，包括类的继承、Object 类、final 类与 final 方法、抽象类、泛型、类的组合以及 Java 包的应用。

3.1 类的继承

3.1.1 继承的概念

继承是一种由已有的类创建新类的机制，是面向对象程序设计的基石之一。一个新类可以从现有的类中派生，这个过程称为类继承。派生出的新类称为已有类的子类，已有类称为超类。继承性很好地解决了软件的可重用性问题，因为子类通过继承拥有了已有类的所有功能，子类只需指明与超类的不同，即增加自己的属性和方法即可。

Java 要求声明的每个类都有超类，当没有显式指定超类时，超类隐含为 java.lang.Object。一个超类可以同时拥有多个子类，这时这个超类实际上是所有子类的公共属性及方法的集合，而每一个子类则是超类的特殊化，是在公共属性的基础上的功能的扩展。

Java 不支持类的多重继承，只支持类的单继承，即每个子类只能有一个直接超类，类的层次结构为树状结构，Object 类为树的根结点。

在类层次结构中，子类与超类的关系是：子类对象与超类对象存在 IS A（或 IS KIND OF）的关系。图 3-1 是以“动物”为超类的类层次结构。

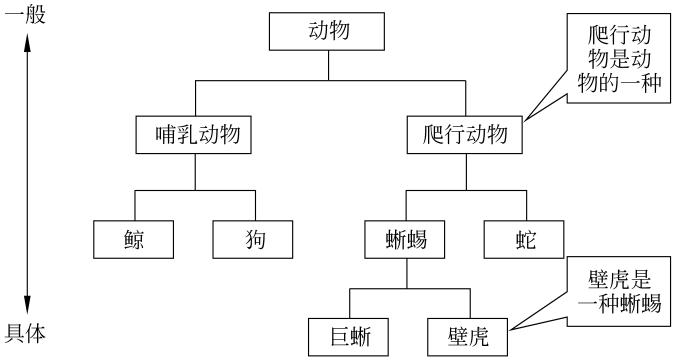

图 3-1 类层次结构举例

通常需要根据应用来定义自己的类层次，一定要小心不要在继承的类层次结构中使用 HAS A 的关系。图 3-2 为错误的继承关系，显然汽车与发动机及车身是整体与部分的关系，而不是一般与特殊的关系。

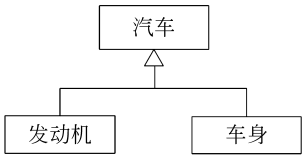

图 3-2　错误的继承关系

与多重继承(类结构为网状结构)相比，单继承比较简单，控制起来相对容易。Java 虽然只支持类的单继承，但可通过实现接口来实现多重继承功能。

采用继承的机制来设计系统中类的层次结构，可以提高程序的抽象程度，使之更接近于人类的思维方式；采用继承机制编写的程序结构更加清晰，不仅节省了编程时间，也降低了维护的工作量。

3.1.2　继承的语法

在 Java 语言中，子类对超类的继承只需要在类的声明中用关键字 extends 说明即可，语法形式为：

```
[ClassModifier] class ClassName [extends SuperClassName]
{
//类体
}
```

其中，关键字 extends 说明要声明的类需要继承超类的属性和行为，SuperClassName 是被继承的超类名称。

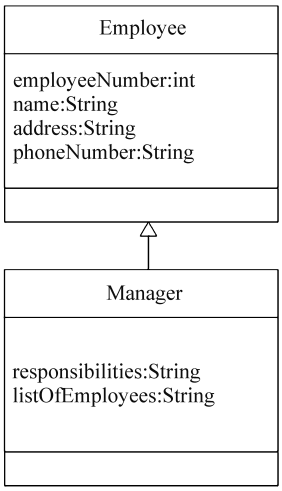

图 3-3　Employee 与 Manager 的类图

例 3-1　设计人员类结构。

在一个公司中，有普通员工(Employee)及管理者(Magager)两类人员。对这两类人员的属性进行分析，设计类及其关系。

设普通员工对象(Employee)可能有的属性信息包括：员工号(employeeNumber)、姓名(name)、地址(address)和电话号码(phoneNumber)。

管理者对象(Manager)除了具有普通员工的属性外，还可能具有下面的属性：职责(responsibilities)、所管理的职员(listOfEmployees)。

由上面的分析，可设计 Employee 及 Manager 两个类，并将 Manager 类作为 Employee 类的子类。Employee 与 Manager 的类图如图 3-3 所示。

类代码如下：

```
//声明超类 Employee
class Employee {
    //数据成员
    int  employeeNumber;
```

```
    String  name, address, phoneNumber;
    //方法略
}
class  Manager extends Employee {
    //子类增加的数据成员
    String responsibilities, listOfEmployees;
    //方法略
}
```

在上面的程序中,子类 Manager 不但继承了超类 Employee 的属性,还增加了自己的属性。一个对象继承的内容取决于此对象所属的类在类层次中的位置,一个对象从其所有的超类中继承属性及方法。

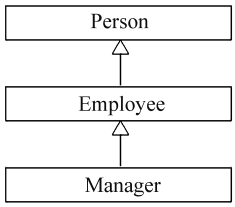

图 3-4 类层次

例 3-2 公有(及保护)属性及方法的继承。

设有三个类:Person、Employee、Manager,其类层次关系如图 3-4 所示。

类声明代码如下:

```
public class Person {
    public String name;
    public String getName() {
        return name;
    }
}
public class Employee extends Person {
    public int employeeNumber;
    public int getEmployeeNumber() {
        return employeeNumber;
    }
}
public class Manager extends Employee {
    public String responsibilities;
    public String getResponsibilities() {
        return responsibilities;
    }
}
```

Employee 及 Manager 类的对象可以使用其超类中声明的公有(及保护)属性方法,就如同在其自己的类中声明一样。下面对其进行测试:

```
public class Tester {
    public static void main(String[] args) {
        Employee li = new Employee();
        li.name = "Li Ming";
        li.employeeNumber = 123456;
```

```
        System.out.println(li.getName());
        System.out.println(li.getEmployeeNumber());
        Manager he = new Manager();
        he.name = "He Xia";
        he.employeeNumber = 543469;
        he.responsibilities = "Internet project";
        System.out.println(he.getName());
        System.out.println(he.getEmployeeNumber());
        System.out.println(he.getResponsibilities());
    }
}
```

运行结果如下：

```
Li Ming
123456
He Xia
543469
Internet project
```

子类不能直接访问从超类中继承的私有属性及方法，但可以间接地使用超类提供的公有或保护方法访问超类的私有属性。

例 3-3　私有属性及方法的继承。

```
//B.java
public class B {
    public int a = 10;
    private int b = 20;
    protected int c = 30;
    public int getB() {
        return b;
    }
}
//A.java
public class A extends B {
    public int d;
    public void tryVariables() {
        System.out.println(a);        //允许
        System.out.println(b);        //不允许
        System.out.println(getB());   //允许
        System.out.println(c);        //允许
    }
}
```

对类 A 进行编译，系统会提示编译错误："The field B.b is not visible"。

在上面的例子中,b 是从类 B 继承来的,由于 b 是私有属性,不能在 A 类中直接存取,但可以使用继承来的公共方法 getB()取得。

3.1.3 隐藏和覆盖

隐藏和覆盖是指子类对从超类继承来的属性变量及方法可以重新加以定义。

1. 属性的隐藏

子类对从超类继承来的属性变量重新加以定义,则从超类继承的属性将被隐藏。

例 3-4 属性的隐藏举例 1。

设有三个类:PhoneCard 类、Number_PhoneCard 类和 D200_Card 类,它们的类层次结构如图 3-5 所示。

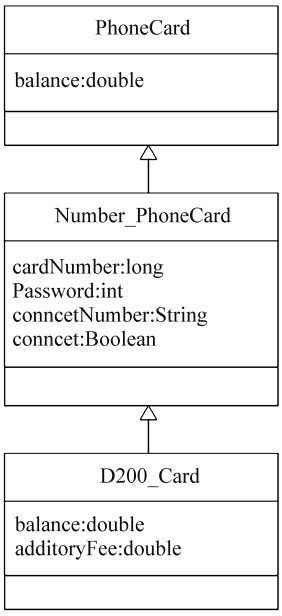

图 3-5 类层次结构

PhoneCard 类有一个用来保存余额的成员变量 balance,其子类 Number_PhoneCard 类继承了 PhoneCard 类的 balance 成员变量,Number_PhoneCard 类的各子类(例如 200 卡、IP 卡)的最初余额可能各不相同。为了表示这一情况,可以为 D200_Card 类重新定义余额,并定义获取这个量的方法。代码如下:

```
class  D200_Card extends Number_PhoneCard {
    double  balance;
    double  additoryFee;
    double  getBalance() {
        return banlance;
    }
    boolean performDial(){   }
}
```

这时，子类中声明了与超类同名的属性变量，即出现了子类变量对同名超类变量的隐藏。这里所谓隐藏是指子类拥有了两个相同名字的变量：一个继承自超类，另一个由自己声明。当子类执行继承自超类的操作时，处理的是继承自超类的变量，而当子类执行它自己声明的方法时，所操作的就是它自己声明的变量，而把继承自超类的变量“隐藏”起来了。

如何访问被隐藏的超类属性呢？只要调用从超类继承的方法，操作的就是从超类继承的属性；另一个方法是使用“super.属性名”来访问。如：

```
class D200_Card extends Number_PhoneCard {
    final double additoryFee = 0.1;
    double balance;
    boolean performDial() {
        if (balance > (0.5 + additoryFee)) {
            balance -= (0.5 + additoryFee);
            return true;
        } else {
            return false;
        }
    }
    double getBalance() {
        return balance;
    }                                    //覆盖
    double getSuperBalance() {
        return super.balance;            //访问继承域
    }
}
```

下面看另一个比较简单的例子。

例 3-5　属性的隐藏举例 2。

```
class A {
    int x = 2;
    public void setx(int i) {
        x = i;
    }
    void printa() {
        System.out.println(x);
    }
}
class B extends A {
    int x = 100;
    void printb() {
        super.x = super.x + 10;
        System.out.println("super.x=" + super.x + "  x=" + x);
    }
```

```
}
public class Tester {
    public static void main(String[] args) {
        A a1 = new A();
        a1.setx(4);
        a1.printa();
        B b1 = new B();
        b1.printb();
        b1.printa();
        b1.setx(6);                    //将继承来的 x 值设置为 6
        b1.printb();
        b1.printa();
        a1.printa();
    }
}
```

运行结果如下：

```
4
super.x=12  x=100
12
super.x=16  x=100
16
4
```

值得注意的是：子类并不能继承超类中的静态属性，但可以对超类中的静态属性进行操作。如在上面的例子中，将“int x = 2;”改为“static int x = 2;”，再编译及运行程序，会得到下面的结果：

```
4
super.x=14  x=100
14
super.x=16  x=100
16
16
```

在上面的结果中，第一行及最后一行都是语句“a1.printa();”输出的结果，显然类 B 中的 printb 方法修改的是类 A 中的静态属性 x。

2. 方法覆盖(Method Overriding)

如果子类不需要使用从超类继承来的方法，则可以声明自己的方法。在声明的时候，使用相同的方法名及参数表，但执行不同的功能，这种情况称为方法覆盖。

一般在下面几种情况下需要使用方法覆盖。

(1) 子类中实现与超类相同的功能，但采用不同的算法或公式。

在前面的例 3-1 中，假设普通员工按小时来支付工资，管理者按年薪制来支付工资，则

在类 Employee 中声明下面的方法来计算月工资：

```
public float computePay() {
    return (regularHoursWorked * payRate) +
        (overTimeHoursWorked * 1.5f * payRate);
}
```

在类 Manager 中对此方法覆盖如下：

```
@Override
public float computePay() {
    return (yearlySalary / 12f);
}
```

在上面的代码中，使用了@Override 注解，表明该方法重载超类的方法。

(2) 在名字相同的方法中，要做比超类更多的事情。

在银行账号中，设有三类账号：普通账号(BankAccount)、存款账号(SavingAccount)及支票账号(CheckingAccount)，它们的类层次关系如图 3-6 所示。

假定从支票账号(CheckingAccount)中取款需要支付 1.25 元的手续费，BankAccount 类中的方法 withdraw(float amount)保持不变，SavingAccount 类从 BankAccount 类中继承之。在 CheckingAccount 类中需要对此方法进行覆盖，代码如下：

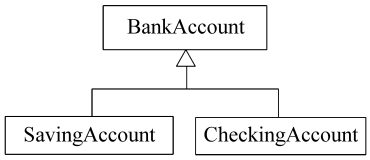

图 3-6　类层次关系 1

```
@Override
public void withdraw(float amount) {
    super.withdraw(amount + 1.25f);
}
```

注意，关键词 super 说明是调用超类的方法而不是本类中的方法。

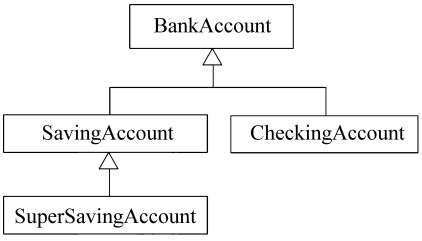

图 3-7　类层次关系 2

(3) 在子类中需要取消从超类继承的方法。

如果在存款账号(SavingAccount)中还有一种超级存款账号(SuperSavingAccount)，如图 3-7 所示，其特点是平时不允许用户从这种账号中取款。仍然保持 BankAccount 类中的方法 withdraw(float amount)不变，使其子类 SavingAccount 不用覆盖而从 BankAccount 类中继承此方法。在 SuperSavingAccount 类中覆盖此方法。代码如下：

```
@Override
public void withdraw(float amount) {
    throw new Exception("Opertation not allowed!")
}
```

由于超级存款账号不允许取款,当程序调用取款方法时,我们认为是系统发生了异常情况(比如非法操作),所以抛出异常(Exception)进行警告。如果不这样做,非法操作等情况将被隐藏,使得系统的安全性和稳定性受到影响。关于异常的使用,我们将在第 5 章做详细讲解。

在方法的覆盖中,由于同名方法隶属于不同的类,所以要解决调用时如何区分它们的问题,只需在方法名前面使用不同的类名或不同类的对象名即可。

方法的覆盖中另一个需要注意的问题是,子类在覆盖超类已有的方法时,应保持与超类完全相同的方法签名,即相同的方法名、返回值和参数列表。

3.1.4 有继承时的构造方法

构造方法是类的一种特殊方法,它可以重载,但不能从超类那里继承。它的名字必须与它所在的类的名字完全相同,并且不返回任何数据类型,也不能是 void 类型。在 Java 中,使用构造方法是生成实例对象的唯一方法。如果在类的声明中没有声明构造方法,则 Java 提供一个默认的构造方法生成对象,对象的属性值为默认值。

有继承时的构造方法遵循以下的原则:

- 子类不能从超类继承构造方法。
- 好的程序设计方法是在子类的构造方法中调用某一个超类构造方法。
- super 关键字也可以用于构造方法中,其功能为调用超类的构造方法。
- 如果在子类的构造方法的声明中没有明确调用超类的构造方法,则系统在执行子类的构造方法时会自动调用超类的默认构造方法(即无参的构造方法)。
- 如果在子类的构造方法的声明中调用超类的构造方法,则调用语句必须出现在子类构造方法的第一行。

例 3-6 有继承时的构造方法举例。

```
//Person.java
public class Person {
    protected String name, phoneNumber, address;
    public Person() {
        this("", "", "");
    }
    public Person(String aName, String aPhoneNumber, String anAddress) {
        name = aName;
        phoneNumber = aPhoneNumber;
        address = anAddress;
    }
}
//Employee.java
public class Employee extends Person {
    protected int employeeNumber;
    protected String workPhoneNumber;
    public Employee() {
        //此处隐含调用构造方法 Person()
```

```
        this(0, "");
    }
    public Employee(int aNumber, String aPhoneNumber) {
        //此处隐含调用构造方法 Person()
        employeeNumber = aNumber;
        workPhoneNumber = aPhoneNumber;
    }
}
//Professor.java
public class Professor extends Employee {
    protected String research;
    public Professor() {
        super();
        research = "";
    }
    public Professor(int aNumber, String aPhoneNumber, String aResearch) {
        super(aNumber, aPhoneNumber);
        research = aResearch;
    }
}
```

3.1.5　应用举例

类继承的层次结构设计取决于具体的应用。

例 3-7　类的继承设计举例。

在一个公司管理信息系统中，除了普通员工(Employee)及管理者(Magager)之外，还有顾客(Customer)。普通员工(Employee)可能有的属性信息包括：员工号(employeeNumber)、姓名(name)、地址(address)和电话号码(phoneNumber)。

管理者(Manager)是一种特殊的 Employee 类，除了具有普通员工所具有的属性及方法外，还可能具有下面的属性和行为。

增加的属性：职责(responsibilities)，所管理的职员(listOfEmployees)。

增加的行为：如工资的计算方法与一般员工不同；福利与一般员工不同。

下面考虑如何表示一个顾客(Customer)，一个顾客可能有的属性信息包括：姓名(name)、地址(address)及电话号码(phoneNumber)。

图 3-8 所示的两个类层次都是不合理的，原因是职员并不总是顾客，顾客也不总是职员。

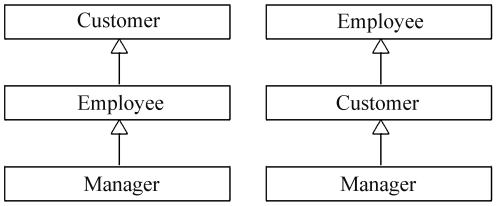

图 3-8　不合理的类层次