第 5 章 Java 继承与高级特性

本章学习目标

继承是一种基于已有类创建新类的机制，利用继承可以先创建一个具有广泛意义的类，然后通过派生创建新类，并添加一些特殊的属性和行为。类的继承是实现代码复用最有效的方法。本章将以类的继承为基础并逐渐展开，介绍若干类的高级特征：

(1) 继承的实现。
(2) 方法的重写。
(3) 抽象类与抽象方法。
(4) 接口。
(5) 内部类。
(6) Lambda 表达式。
(7) 泛型。
(8) Java 反射机制。
(9) 注解。

5.1 继承使用初探

当新建一个类时，也许会发现该类与之前的某个类非常相似，如绝大多数的属性和行为都相同。这时，可以选择复制原类中的语句，对其部分修改后加入新类中，但这意味着必须同时维护两个相似的 Java 程序。

另外一种方法是通过继承，让新类自动获得被继承类中已有的属性和方法，同时添加原类中没有的属性和方法即可。例如，根据人的特征定义类 Person 如下：

```
public class Person {
    private String name;
    private int age;
    public void say(){
        System.out.println(name+"can say");
    }
    public void setName(String name) {
        this.name = name;
    }
```

```
    public String getName(){
        return name;
    }
}
```

该类对所有人均适用，但如果根据学生的特点需要定义一学生类，则可以肯定的是学生类中，除了姓名、年龄属性外，还可能有所在学校名称（这是一般人没有的），此外学生的行为中还包括在校学习（study）这一方法（这也是一般人没有的）。因此，可以通过继承的方式建立类 Student 如下：

```
public class Student extends Person{
    String schoolname;                          //增加新的属性 schoolname
    public void study(){                        //增加新的方法 study
        System.out.println("I am studying in school");
    }
    public static void main(String[] args) {
        Student student1 = new Student();
        student1.name="MingM";student1.age=10;//引用继承自父类的变量
        student1.schoolname="CQ";
        student1.say();                         //调用继承自父类的方法
        student1.study();                       //调用子类新增加的方法
        System.out.println("My name is "+student1.name);
        System.out.println("My schoolname is "+student1.schoolname);
    }
}
```

程序运行结果：

```
I can say
I am studying in school
My name is MingM
My schoolname is CQ
```

分析：通过关键字 extends 定义了类 Person 的子类 Student，然后添加了只有学生才有的属性 schoolname 和方法 study()。

在 main()方法中，可以看到尽管 Student 中没有定义变量 name、age 以及方法 say()，但是子类却可以通过继承的方式自动取得，并像访问自己的成员变量和方法一样引用即可。

5.2 类的继承

5.2.1 继承的实现

继承的实现其实非常简单，其格式如下：

```
class 子类名 extends 父类名{
    类体
}
```

extends 是关键字，后跟父类的类名，如果没有父类，则缺省父类是 java.lang.Object。Java 只支持单继承，即只能有一个父类，但类之间的继承可以具有传递性。

子类可以通过继承自动获得父类中访问权限为 public、protected、default 的成员变量和方

法,但不能继承权限为 private 的成员变量和方法。

【例 5.1】 类的继承示例。

```
package code0502;
class A {
    int i;
    void showi() {
        System.out.println("i: " + i);
    }
}
class B extends A {
    int k;

    void show() {
        System.out.println("k: " + k);
        showi();
    }
    void sum() {
        System.out.println("i+k: " + (i + k));
    }
}
public class Simple {
    public static void main(String args[]) {
        A superOb = new A();
        B subOb = new B();
        superOb.i = 10;                          //对父类对象的成员赋值.
        System.out.println("Contents in 父类: ");
        superOb.showi();
        subOb.i = 7;                             //对子类中继承得到的变量 i 赋值
        subOb.k = 9;
        System.out.println("Contents in 子类: ");
        subOb.show();
        System.out.println("Sum of i and k in 子类:");
        subOb.sum();
    }
}
```

程序运行结果:

```
Contents in 父类:
i: 10
Contents in 子类:
k: 9
i: 7
Sum of i and k in 子类:
i+k: 16
```

分析:类 B 中虽然没有定义变量 i 和方法 showi(),但却可以通过继承关系获得,因此在类 B 中直接引用 i 和方法 showi()是正确的。但如果在类 A 的语句 int i 前加上修饰符 private,则上述程序会出现编译错误,其原因在于类 B 无法继承到 private 类型的变量 i,因此 i 对类 B 是不可见的。

提示:①尽管一个子类可以从父类继承所有允许的方法和变量,但它不能继承构造函数,掌握这点很重要。一个类要得到构造函数,只有两个办法:重写构造函数;根本不写构造

函数，这时系统为每个类生成一个缺省构造函数。②为了防止继承被滥用，Java15 引入了一种特殊的密封类(使用 sealed 修饰 class)，并通过 permits 明确写出能够从该 class 继承的子类名称，如 public sealed class Shape permits Circle{}，这表明只允许类 Circle 继承 Shape。

5.2.2 继承与重写

在类的继承过程中，如果子类中新增的变量和方法与父类中原有的数据和方法同名，则会重写(也称覆盖)从父类继承来的同名变量和方法。重写又分变量重写和方法重写，变量重写是指父类和子类中的变量名相同，数据类型也相同。方法重写与之前介绍的方法重载相似，但更严格，不仅要求父类与子类中的方法名称相同，而且参数列表也要相同，只是实现的功能不同。

【例 5.2】 重写父类中的同名方法和变量。

```
package code0502;
class SuperCla {
    int a = 3, b = 4;
    void show() {
        System.out.println("super result=" + (a + b));
    }
}
class SubCla extends SuperCla {
    int a = 10;                          //重写父类中同名的变量 a
    void show() {                        //重写父类中同名的方法 show
        int c = a * b;
        System.out.println("sub result=" + c);
    }
}
public class OverrideTest {
    public static void main(String args[]) {
        SuperCla sp = new SuperCla();
        SubCla sb = new SubCla();
        sp.show();                       //此处调用的是父类中的方法 show
        System.out.println("In super Class:a=" + sp.a);  //此处引用的是父类中的变量 a
        sb.show();                       //此时子类对象的 show 方法覆盖了父类的同名方法
        System.out.println("In sub Class:a=" + sb.a);
    }
}
```

程序运行结果：

```
super result=7
In super Class:a=3
sub result=40
In sub Class:a=10
```

分析：子类 SubCla 中定义有与父类 SuperCla 同名的变量 a 和方法 show()，因此使用子类对象 sb 时访问变量 a 和方法 show()时，引用的是子类中的成员，父类的同名变量被覆盖，同名的方法被重写。

如果想在子类中访问父类中被覆盖的成员怎么办呢？这时可以使用关键字 super 来解决这一问题，基本格式如下。

访问父类成员：

```
super.成员变量
```

或


```
super.成员方法([参数列表])
```

访问父类构造方法：

```
super([参数列表])
```

【例 5.3】 super 关键字的使用。

```
package code0502;
//定义员工类
class Employee {
    private String name;
    private int salary;
    public String getDetails() {
        return "Name:" + name + "\nSalary:" + salary;
    }
    Employee() {
        name = "Tom";
        salary = 1234;
    }
}
//定义经理类
class Manager extends Employee {
    public String department;
    /*重写 getDetails 方法*/
    public String getDetails() {
        System.out.println("I am in Manager");
        return super.getDetails();          //调用父类的 getDetails 方法
    }
    Manager() {
        super();                            //访问父类的无参构造方法,即 Employee()
        department = "sale";
    }
}
public class Inheritance {
    public static void main(String arg[]) {
        Manager m = new Manager();
        System.out.println(m.getDetails());
        System.out.println("department:" + m.department);
    }
}
```

程序运行结果：

```
I am in Manager
Name:Tom
Salary:1234
department:sale
```

分析：程序首先对 Manager 实例化，并自动调用无参构造方法 Manager()，主要完成两项主要任务：一是通过 super()调用父类的无参构造函数，即 Employee()；二是对子类中新增变量 department 初始化。接着，语句 m.getDetails()调用子类的同名方法 getDetails()，并

在方法体中通过 super.getDetails()实现对父类中 getDetails()方法的调用。

注意：生成一个子类的实例时，首先要执行子类的构造方法，但如果该子类继承自某个父类，则执行子类的构造方法前，系统自动调用父类的无参构造函数。因此，例 5.3 的方法 Manager()中去掉语句 super()，效果是完全相同的。

与 super 不同，关键字 this 的主要作用是表示当前对象的引用，当局部变量和类的成员变量同名时，该局部变量作用区域内成员变量就被隐藏了，必须使用 this 来指明。

【例 5.4】 this 关键字的使用。

```
package code0502;
public class ThisTest {
    public static void main(String[] args) {
        Local aa = new Local();
    }
}
class Local {
    public int i = 1;                                  //这个 i 是成员变量
    Local(int i) {                                     //这个 i 是局部变量
        System.out.println("this.i ="+ this.i);       //this.i 指的是对象本身的成员变量 i
        System.out.println("i = "+i);                  //变量 i 前没有 this,因此是局部变量
    }
    Local(){
        this(6);
    }
}
```

程序运行结果：

```
this.i =1
i = 6
```

分析：关键字 this 的主要作用有：①通过它引用成员变量，如上例中 this.i 即指的是当前对象中的成员变量 i；②通过 this 调用类的构造方法，如上例中的 this(6)将调用对应的构造方法 Local(int i)。

5.2.3 继承与类型转换

和标准类型数据的转换一样，不同类的对象之间也可以相互转换，但前提是源和目标类之间必须通过继承相联系。转换可分为显式和隐式两种，显式转换格式如下：

```
(类名)对象名
```

它将对象转换成类名所表示的其他对象。Java 支持父类和子类对象之间的类型转换，如果是子类对象转换为父类，可进行显式转换或隐式转换；如果是父类对象转换成子类，编译器首先要检查这种转换的可行性，如果可行，则必须进行显式转换。

【例 5.5】 对象类型的转换示例。

```
package code0502;
class CA{
    String s="class CA";
}
class CB extends CA{
    String s="class CB";
```

```
}
class Convert{
    public static void main(String args[]){
        CB bb,b=new CB();
        CA a,aa;
        a=(CA)b;                                    //显式转换
        aa=b;                                       //隐式转换
        System.out.println(a.s);
        System.out.println(aa.s);
        bb=(CB)a;                                   //显式转换
        System.out.println(bb.s);
    }
}
```

程序运行结果:

```
class CA
class CA
class CB
```

分析:b 是子类 CB 的实例,将其转换为父类 CA 的实例时可以进行显式或隐式转换。而父类对象 a 转换为子类 CB 的对象时,必须进行显式转换。

5.2.4 实用案例

【例 5.6】 通过继承定义员工和经理类。

普通员工和经理作为职员有很多共同之处,如都可以取得工资报酬,但经理可能还会获得额外的奖金,因此在成员属性和方法上可能有特殊之处。可以将 NewEmployee 类定义为父类,NewManager 类作为子类,子类继承了父类,并添加了一个新的 setBonus()方法,用于增加奖金,最后打印出结果。参考实现代码如下:

```
package code0502;
import java.util.*;
class NewEmployee {
    private String name;
    private double salary;
    private Date hireDay;

    public NewEmployee(String n, double s, int year, int month, int day) {
        name = n;
        salary = s;
        GregorianCalendar calendar = new GregorianCalendar(year, month - 1, day);
        hireDay = calendar.getTime();
    }
    public String getName() {
        return name;
    }
    public double getSalary() {
        return salary;
    }
    public Date getHireDay() {
        return hireDay;
    }
```

```
    public void raiseSalary(double byPercent) {
        double raise = salary * byPercent / 100;
        salary += raise;
    }
}
package code0502;
class NewManager extends NewEmployee {
    private double bonus;
    public NewManager(String n, double s, int year, int month, int day) {
        super(n, s, year, month, day);
        bonus = 0;
    }
    public double getSalary() {
        double baseSalary = super.getSalary();
        return baseSalary + bonus;
    }
    public void setBonus(double b) {
        bonus = b;
    }
}
package code0502;
public class ManagerTest {
    public static void main(String[] args) {
        NewEmployee e= new NewEmployee("Harry Hacker", 50000, 1989, 10, 1);
        e.getName();
        System.out.println(e.getName()+":"+e.getSalary());
        NewManager boss = new NewManager("Carl Cracker", 80000, 1987, 12, 15);
        boss.setBonus(5000);
        System.out.println(boss.getName()+":"+boss.getSalary());

    }
}
```

程序运行结果：

```
Harry Hacker:50000.0
Carl Cracker:85000.0
```

5.3 多　　态

5.3.1 多态性的概念

简单地说，多态性就是一个名称可以对应多种不同的实现方法。Java 语言的多态性体现在两个方面：编译多态和运行多态。

编译多态是指在程序编译过程中体现出的多态性，如方法重载。尽管方法名相同，但由于参数不同，在调用时系统根据传递参数的不同确定被调用的方法，这个过程是在编译时完成的。例如，定义一个作图的类，它有一个 draw()方法用于绘图，根据不用的使用情况，可以接收字符串、矩形、圆形等参数。对于每一种实现，方法名都为 draw()，只不过具体实现方式不同，不用另外重新起名，这样大大简化了方法的实现和调用，程序员无须记住很多的方法名，只需传递相应的参数即可。

运行多态则是由类的继承和方法重写引起的,由于子类继承了父类的属性和方法,因此,凡是父类对象可以使用的地方,子类对象也可以使用。如例 5.5 中的类 CA 和 CB,可以直接生成子类 CB 的对象,并将该引用赋给父类 CA 的对象,即

```
CA a=new CB();
```

它等价于下面两个子句:

```
CB b=new CB(); CA a=b;          //隐式类型转换
```

现在有一个问题:如果子类重写了父类的成员方法,调用该方法时,到底应该调用父类中的方法还是子类中的方法?这无法在编译时确定,需要系统在运行时根据实际情况来决定,所以这种由方法重写引起的多态叫运行多态。

Java 规定:对重写的方法,Java 根据调用该方法的实例的类型来决定选择哪种方法。对子类的实例,如果子类重写了父类的方法,则调用子类的方法;如果子类没有重写父类的方法,则调用父类的方法。

【例 5.7】 类的多态性示例。

```
package code0503;
class A {
    void callme() {
        System.out.println("inside A");
    }
}
class B extends A {
    void callme() {
        System.out.println("inside B");
    }
}
class Poly {
    public static void main(String args[]) {
        A a = new A();
        B b = new B();
        A c = new B();
        a.callme();
        b.callme();
        c.callme();
    }
}
```

程序运行结果:

```
inside A
inside B
inside B
```

分析:对实例 a 和 b,它们的类型为类 A 和 B,所以分别调用父类和子类中的方法 callme()即可。对实例 c,它被实例化为子类 B 的一个对象,且子类重写了方法 callme(),因此根据转型规则,(由于调用该方法的实例类型为 B)这时应该调用子类 B 的方法,所以输出为 inside B。

多态性在实际应用中有很多好处。

(1) 可替换性:多态对已存在代码具有可替换性。例如,多态对圆 Circle 类有效,对其他

任何圆形几何体(如圆环)也同样有效。

(2) 可扩充性：多态对代码具有可扩充性。增加新的子类不影响已有类的多态性、继承性。实际上，新加子类更容易获得多态功能。例如，在实现了圆锥、半圆锥以及半球体的多态基础上，很容易增添球体类的多态性。

5.3.2 实用案例

【例 5.8】 将员工和经理存入同一数组。

数组一般要求存入的数据必须是同一类型，如整形数组要求每个数组成员均为整数。但之前定义的类 NewEmployee 和 NewManager 显然不是同一类型，那么如何将其存入到同一个数组中呢？实现代码如下：

```
package code0502;
public class EmployeeArray {
    public static void main(String[] args) {
        NewEmployee[] staff = new NewEmployee[3];
        NewManager boss = new NewManager("Carl", 80000, 1987, 12, 15);
        boss.setBonus(5000);
        staff[0] = boss;
        staff[1] = new NewEmployee("Harry", 50000, 1989, 10, 1);
        staff[2] = new NewEmployee("Tommy", 40000, 1990, 3, 15);
        for (NewEmployee e : staff)
            System.out.println("name=" + e.getName() + ",salary=" + e.getSalary());
    }
}
```

程序运行结果：

```
name=Carl,salary=85000.0
name=Harry,salary=50000.0
name=Tommy,salary=40000.0
```

分析：满足上述需求的关键在于数组类型的定义，如本例中将数组申明为父类 NewEmployee 类型，这样将子类对象存入数组时，系统将利用隐式类型转换规则，将其统一到父类类型中，即可实现将不同类对象存入同一数组的目标。

如何测试一个对象是否为某种类型的实例呢？如本例中，我们想知道 staff[0]是否为 NewManager 的实例，可以通过 instanceof 运算符来实现，格式为 variable instanceof TypeName，如果 variable 为 TypeName 或 TypeName 父类型的实例，运算结果为 true，否则返回 false。因此，表达式 staff[0] instanceof NewManager 的结果为 true，staff[0] instanceof NewEmployee 的结果也为 true，staff[1] instanceof NewManager 的结果则为 false。

5.4 抽象类与抽象方法

5.4.1 定义抽象类及实现抽象方法

关键字 abstract 修饰的类称为抽象类，抽象类是一种没有完全实现的类。不能用它实例化任何对象，它的主要用途是用来描述一些概念性的内容，然后在子类中具体去实现这些概念，这样可以提高开发效率，统一用户接口，所以抽象类更多是作为其他类的父类。