第5章

封装、继承与多态

在面向对象程序设计中，封装性通过对数据成员和成员方法进行访问控制，实现对象的属性和行为结合为一个整体，尽可能隐藏对象的内部细节信息。对象之间交互通过一个对象向另一个对象发送消息来请求或提供服务。继承性体现在两个类之间的层次关系，是面向对象程序设计的一种重要手段。通过继承机制明确类间关系，实现软件复用。多态性是面向对象程序设计中的同名方法、不同操作共存的机制，引入多态机制可提高类的抽象度和封闭性，统一类的对外接口。


封装机制

5.1 封装机制

封装反映事物的独立性。在面向对象程序设计中，封装机制能够保证除对象本身以外其他任何部分都不能随意存取对象的数据成员，从而有效地避免出现外部错误信息“污染”内部数据的问题。此外，对象提供对外服务的接口，减少了对象内部的修改对外部使用的影响。

实现封装机制需要遵循以下规则。

（1）在类的定义中，设置数据成员和成员方法的访问控制权限，限定本类对象以及其他类对象的使用范围。

（2）对象提供对外服务的接口，描述其他对象使用的方式。

（3）任何其他对象都无法修改当前对象的数据成员和成员方法。

在面向对象程序设计中，类的概念本身具有封装的意义，因为对象的特性是由其所属类定义来描述的。因此，将对象的属性和行为封装，其载体是类，类对外隐藏实现细节。

采用封装机制，保证类内部数据结构的完整性。Java 程序的封装机制通过对数据成员和成员方法进行访问控制来实现。一个类始终能够访问自身的数据成员和成员方法，但是，其他类是否可以访问该类的成员，由该类的访问控制权限修饰符和成员的访问控制权限修饰符共同决定。


访问控制

5.2　访 问 控 制

访问控制权限修饰符是一组限定类、数据成员和成员方法是否可被其他类访问的修饰符。Java 提供 public 修饰符、默认修饰符、protected 修饰符和 private 修饰符 4 种成员访问控制权限修饰符，实现对象的数据成员和成员方法的封装，如表 5-1 所示。

表 5-1　Java 类的成员访问控制权限修饰符

修　饰　符	说　　明	修　饰　符	说　　明
public	公有级别	private	私有级别
默认	默认级别	protected	保护级别

结合 4.3.1 节类修饰符，可分别定义类和成员的组合访问控制权限，如表 5-2 所示。

表 5-2　组合访问控制权限及其作用

数据成员和成员方法	类	
	public	默认
public	所有类	包中类
默认	包中类	包中类
protected	包中类以及所有子类	包中类
private	类自身	类自身

5.2.1　public 公有级别

Java 类通过包的形式进行组织管理。Java 规定：定义在同一个程序文件中的所有类都属于同一个包。对于不同包中的类而言，它们相互之间是不可见的，也不可相互引用。因此，为了方便不同包的类间使用，Java 提供公有级别的访问控制权限修饰符 public。

当一个类声明为 public 时，只需在其他包的程序文件中使用 import 语句引入该 public 类，就可访问和引用这个公共类，并可创建该类的对象。Java 类库中很多类都是公共类，可在程序中引入使用。

此外，公共类扩展了类的使用范围，将其视为整体提供给其他类使用。但是，如果需要访问公共类的数据成员和成员方法还需由成员的修饰符决定。只有当数据成员和成员方法被声明为 public 时，其他类才能方便使用。

在程序设计时，如果希望提供公共类给其他类使用，则应该将类自身和类的数据成员和成员方法都声明为 public。但是，当成员都声明为 public 时，影响程序封装，会出现安全性下降的问题。因此，通常情况下，类的成员很少使用 public 修饰符。

程序示例 5-1　chapter5 /chapter5_1.java

StudentInfo.java

```
package com.cumtb.student;                                          ①
public class StudentInfo {
    public int studentNumber = 1010;                                ②
    int temp;                                                       ③
    public StudentInfo() {
        temp = 0;
    }
    public void showStudent() {
        System.out.println("temp= " + temp);
    }
}

chapter5_1.java
package com.cumtb;
import com.cumtb.student.StudentInfo;                               ④
public class chapter5_1 {
    public static void main(String[] args) {
        StudentInfo stu = new StudentInfo();
        System.out.println(stu.studentNumber);                      ⑤
        System.out.println(stu.temp);       //报错                  ⑥
        studentInfo.showStudent();
    }
}
```

简要说明程序示例 5-1。

(1) 代码①在 com.cumtb.student 包中定义一个公共类 StudentInfo,表明此类可提供给其他类使用。

(2) 代码②、③在 SutdentInfo 类中定义一个公共的数据成员 studentNumber,可提供给其他类使用,对外公开;一个无修饰符的数据成员 temp。

(3) 代码④如需在 com.cumtb 包中 chapter5_1 类中使用 StudentInfo 类,则必须先通过 import 语句引入该公共类,并创建该类的对象。

(4) 代码⑤、⑥使用公共类对象 stu 可以访问公共类中公共的数据成员 studentNumber。但是,无法访问无修饰符的数据成员 temp,其不能提供包间访问,但可通过公共的 showStudent()方法访问。

5.2.2 默认级别

如果一个类没有给定修饰符,其具有默认的访问控制特点,也称包访问或友好访问。Java 规定:只有在同一个包中的对象才能访问和引用包中的类。同理,如果类内的数据成员和成员方法没有给定修饰符,它们也具有包访问特性,可被同一个包中其他类访问。

程序示例 5-2　chapter5 /chapter5_2.java

```
StudentInfo2.java
```

```
package com.cumtb;                                                    ①
class StudentInfo2 {
    public int studentNumber = 1010;
    int temp;
    public StudentInfo2() {
        temp = 0;
    }

    public void showStudent() {
        System.out.println("temp= " + temp);
    }
}

chapter5_2.java
package com.cumtb;                                                    ②
public class chapter5_2 {
    public static void main(String[] args) {
        StudentInfo2 studentInfo = new StudentInfo2();
        System.out.println(studentInfo.studentNumber);                ③
        System.out.println("temp= " + studentInfo.temp);              ④
        studentInfo.showStudent();                                    ⑤
    }
}
```

简要说明程序示例 5-2。

(1) 代码①、②同时在 com.cumtb 包中定义两个类：无修饰符类 StudentInfo2 和公共类 chapter5_2。

(2) 代码③、④、⑤可在 chapter5_2 类中直接创建 StudentInfo2 类的对象，并访问其数据成员和成员方法。

5.2.3　protected 保护级别

在 Java 中，protected 修饰符只能修饰类的数据成员和成员方法。使用 protected 修饰的成员有 3 种形式：类自身使用、与其同属一个包中的其他类使用及其他包中当前类的子类使用。

程序示例 5-3　chapter5 /chapter5_3.java

```
StudentInfo3.java
package com.cumtb.student;
public class StudentInfo3 {
    protected int studentNumber = 1010;
    int temp;
    public StudentInfo3() {
        temp = 0;
```

```
    }

    protected void showStudent() {
        System.out.println("temp= " + temp);
    }
}

chapter5_3.java
package com.cumtb;
import com.cumtb.student.StudentInfo3;
public class chapter5_3 extends StudentInfo3{                    ①
    public static void main(String[] args) {
        chapter5_3 chap = new chapter5_3();
        chap.showStudent();                                      ②
        System.out.println(chap.studentNumber);                  ③
    }
}
```

简要说明程序示例 5-3。

(1) 代码①中类 chapter5_3 继承 com.cumtb.student 包中 StudentInfo3 类,有关继承内容见 5.4 节。

(2) 代码②、③创建子类 chapter5_3 的对象,并访问父类中的 protected 修饰的数据成员 studentNumber 和成员方法 showStudent()。

5.2.4 private 私有级别

在 Java 中,private 修饰符只能修饰类的数据成员和成员方法。使用 private 修饰符的数据成员和成员方法只能在类自身内部访问和修改,不能被其他任何类直接访问和引用,提供最高级别的保护。如果其他类需要访问或修改当前类的私有成员时,需要在当前类中定义属性访问方法,实现访问类私有成员的功能。

属性访问方法只返回数据成员的值,通常有两种形式。

(1) 设置数据成员值的属性访问成员方法。

```
public void setXXX(形参){
    方法体
}
```

(2) 获取数据成员值的属性访问成员方法。

```
public 返回值类型 getXXX(){
    方法体
}
```

其中,按照编码规范的约定,XXX 表示方法的辅助名称,前面的 set 和 get 为约定名称,不建议更改。通常情况下,属性访问方法使用 public 修饰符修饰,可提供给其他类使用。

属性访问方法提供对外的数据访问接口，即使改变类内部的实现，不会影响其他类的使用。

程序示例 5-4　chapter5 /chapter5_4.java

```
package com.cumtb;
class StudentInfo4 {
    private int studentNumber = 1010;
    int temp;
    public StudentInfo4() {
        temp = 0;
    }

    public void setStudentNumber(int number) {
        studentNumber = number;
        temp = ++studentNumber;
    }
    public int getStudentNumber() {
        return studentNumber;
    }
    void showStudent() {
        System.out.println("temp= " + temp);
    }
}

public class chapter5_4 {
    public static void main(String[] args) {
        StudentInfo4 studentInfo4= new StudentInfo4();
        studentInfo4.showStudent();
        studentInfo4.setStudentNumber(1012);
        studentInfo4.showStudent();
        System.out.println(studentInfo4.getStudentNumber());
    }
}
```


消息

5.3　消　　息

在真实校园里，张同学和李同学两个是彼此独立封装的对象，李同学可以请求张同学讲授课本知识，李同学还可以给张同学排队买饭。也就是说，李同学不仅可以请求张同学的服务，还可以服务张同学。那么，请求便是人与人之间交流的手段。在面向对象程序设计中，将请求抽象为消息。

对象是独立封装的实例，对象之间相互作用通过一个对象向另一个对象发送消息的方式完成。当系统中的某个对象请求另一个对象提供服务时，对象响应请求信息，完成指

定的服务。通常情况下,发送消息的对象称为发送者,接收消息的对象称为接收者,对象之间只能通过消息进行传递信息。

消息,向对象发送服务请求,是对数据成员和成员方法的引用。一个有效的消息包含4方面内容。

(1) 对象名。提供服务的对象标识。使用对象名来指定某个具体存在的对象来提供服务,用于区分提供服务的对象。

(2) 方法名。提供服务的具体方法。对象中的成员方法非常多,需要具体指明是哪个方法来提供服务。

(3) 实参。消息传递的输入信息。实参将传递给方法的形参,告诉其提供服务的方法要做什么,怎么做。

(4) 返回值。回答信息。方法的返回值,告诉消息的发送者服务完成的效果。

从上可以看出,消息本质上是对象成员方法的使用。因此,消息具有3个重要的性质。

(1) 同一个对象可以接收多个消息,产生不同的响应。根据对象的成员方法的不同,可以提供不同的服务。

(2) 相同的消息可以发送给不同的对象,产生不同的响应。在继承机制中,子类可以继承并覆盖父类同名、同参数的成员方法,那么子类对象和父类对象接收同一个成员方法调用时,会产生截然不同的响应。

(3) 消息的发送可以不考虑具体的接收者,对象可响应消息,也可不响应消息,即对消息的响应不是必需的。某些对象的成员方法体内没有执行逻辑的方法体,即使接收到消息,也不会产生有效的响应。

在面向对象程序设计中,消息分为两大类:公有消息和私有消息。

(1) 公有消息。当一批消息属于同一个对象时,由外界对象直接发送给这个对象的消息称为公有消息。外界对象只能发送公有消息。

(2) 私有消息。对象发给自身的消息称为私有消息。私有消息不对外开放,外界不必了解内部细节。

程序示例 5-5　chapter5 /chapter5_5.java

```
package com.cumtb;
class StudentInfo5 {
    public int studentNumber;
    public String studentName;
    int temp;
    public StudentInfo5(int number, String name) {
        studentNumber = number;
        studentName = name;
    }
    public void showName() {
        System.out.println(studentName);
    }
```

```
    public void showNumber() {
        System.out.println(studentNumber);
    }
    public void sendTemp(int t) {
        //对象发给自身的消息，给自己数据成员赋值，私有消息
        temp = t;
    }
    public void showTemp() {
        System.out.println(temp);
    }
}

public class chapter5_5 {
    public static void main(String[] args) {
        StudentInfo5 stu5_1 = new StudentInfo5(1010,"刘同学");
        StudentInfo5 stu5_2 = new StudentInfo5(1011,"王同学");
    //公有消息,向对象 stu5_1 发送显示学号、姓名消息。同一个对象接收不同形式消息
        stu5_1.showNumber();
        stu5_1.showName();
    //公有消息,向对象 stu5_1、stu5_2 发送显示姓名消息。不同对象接收同一形式消息
        stu5_1.showNumber();
        stu5_2.showNumber();
    //私有消息,向对象 stu5_1 发送一个临时数值
        stu5_1.sendTemp(1000);
        stu5_1.showTemp();
    }
}
```

5.4　继 承 机 制


继承机制

继承是面向对象程序设计的一个重要手段，可使整个程序架构具有一定弹性。在面向对象程序设计中，使用继承机制可有效地组织程序结构，设计系统中的类，明确类之间的关系，充分利用已有的类来完成更复杂的软件开发，提高程序开发效率，减少程序维护工作量。

5.4.1　继承的概念

同类事物具有共性，但在同类事物中，每个事物又有其特殊性。使用 4.2 节的抽象原则抽取事物的共性，删除特殊性，可得到适用于一批对象的类，即一般类。但是，那些具有特殊性的类称为特殊类。也就是说，特殊类具有一般类的全部属性和方法，但特殊类中又有自己的特殊属性和方法。例如，本科生类具有学生类的全部属性和方法，但其自身又具有某些特殊的属性和方法，那么，学生类称为一般类，本科生类称为学生类的特殊类。

在面向对象程序设计中，使用继承原则：将一般类对象和特殊类对象都具有的属性

和方法统一在一般类中定义,在特殊类中就不再重复定义一般类中已经定义的内容。那么,特殊类自动地拥有一般类或更高层次类中定义的属性和方法。特殊类的对象拥有一般类的全部或部分属性与方法,称为特殊类继承一般类。

继承表明一种对象的类之间的层次关系,某类的对象可继承另一个类对象的数据成员和成员方法。例如,若本科生类继承学生类,那么本科生类就拥有学生类的全部或部分数据成员和成员方法。在面向对象程序设计中,被继承的学生类称为父类、基类或超类;继承的本科生类称为子类或派生类。父类和子类的层次关系如图 5-1 所示。

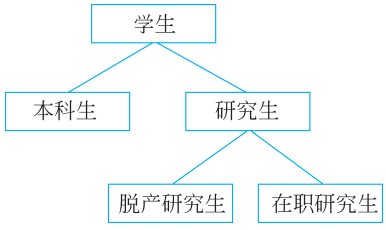

图 5-1 父类与子类的层次关系

使用继承机制,避免了一般类和特殊类之间共性特征的重复定义描述。通过继承可以清晰地描绘出共性特征的适用范围。在一般类中定义的数据成员和成员方法适用当前类以及其下属每个特殊类的对象。

5.4.2 继承的特点

继承的特点如下。

(1) 继承关系具有**传递性**。例如,脱产研究生类继承研究生类,研究生类继承学生类,则脱产研究生类既有从研究生类继承的属性和方法,也有从学生类继承的属性和方法,甚至还可以自定义属性和方法。继承而来的属性和方法虽是隐式的,但仍属于脱产研究生类的属性和方法。继承是建立和扩充新类的有效手段。

(2) 继承关系是**层次关系**,简化了对事物的描述。

(3) 继承提供软件复用功能。例如,研究生类继承学生类,那么在建立研究生类时,只需定义与研究生相关的少量属性和方法即可。软件复用增强了代码一致性,减少了模块间的接口,提高了程序的易维护性。

(4) **多重继承机制**,一个特殊类可继承多个一般类。从安全性和可靠性角度出发,Java 仅提供单继承,通过使用接口机制实现多重继承,如图 5-2 所示。

5.4.3 继承的使用

在 Java 中,使用关键字 extends 实现继承机制。在定义类时,使用 extends 指明该类的直接父类,那么新定义的类称为指定父类的子类,两个类之间建立了继承关系。在新定义的子类中,可继承使用直接父类中全部非私有的数据成员和成员方法。

如果在定义新类中没有使用 extends 指明父类,系统会默认为新类是系统类 Object 的子类。在 Java 中,所有的类都直接或间接继承了 java.lang.object 类。Object 类是一个

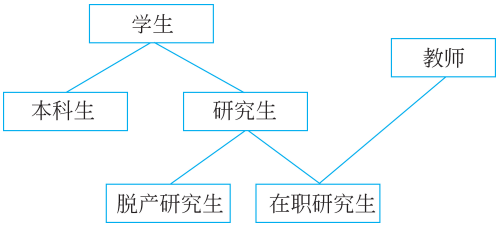

图 5-2　单继承与多重继承

比较特殊的类，它是所有类的父类，是 Java 类层中的最高层类。Object 类常用方法如表 5-3 所示。

表 5-3　Object 类常用方法

方　法	说　明
Class getClass()	返回对象执行时的 Class 实例
String getName()	返回对象执行时所对应的类名
String toString()	将对象返回为字符串形式，返回一个 String 实例
boolean equals()	比较两个对象的实际内容是否相等

如果在定义新类中显式地通过 extends 指明父类，在使用继承机制时，需要从以下 4 个方面考虑继承的使用。

1. 数据成员的继承

子类可以直接继承使用父类中全部非私有的、不同名的数据成员。

程序示例 5-6　chapter5 /chapter5_6.java

```
package com.cumtb;
class StudentInfo6 {
    private int studentNumber = 1010;     //私有数据成员
    public String studentName = "刘同学";
}

public class chapter5_6 extends StudentInfo6{
    public static void main(String[] args) {
        chapter5_6 chap = new chapter5_6();
        System.out.println(chap.studentName);
//报错，无法访问父类私有数据成员
        System.out.println(chap.studentNumber);
    }
}
```

在程序示例 5-6 中，私有的数据成员只能提供给类自身使用，其子类无法访问。如果