第 5 章　面向对象高级特性

主要内容：封装、继承和多态是面向对象程序设计的三大核心，封装在第 4 章已经讲解，本章主要学习面向对象编程语言 Java 的高级特性——继承、多态、接口等。本章重点讲解 Java 语言的面向对象的高级技术，包含结合本章内容完善第 4 章中的学生类案例，继承与派生、this 和 super、final 修饰符、多态、接口、接口回调、匿名类、内部类和异常类。

教学目标：理解面向对象高级技术特性，掌握 Java 语言中继承、多态、接口的思想和实现技术。

5.1　案例：完善学生类

5.1.1　完善学生类的步骤

在 Java 项目“javasource”中新建一个包“cn.ahut.cs.mainapp.chapter5”，在包中新建一个类 EXA5_1。

(1) 在类 EXA5_1 中引用第 4 章创建的类 CDate，新建一个接口 CPeople，定义常量和方法，方法只有方法头，没有方法实现。

【例 5-1】 完善学生类。

```
1.  package cn.ahut.cs.mainapp.chapter5;
2.  import cn.ahut.cs.userdefinelib.CDate;
3.  interface CPeople{                          //人员接口
4.     final String nationality="中国";         //国籍是一个常量
5.     String getSno();
6.     void setSno(String sno);
7.     String getName();
8.     void setName(String name);
9.     CDate getDate();
10.    void setDate(CDate date);
11.    double getGrade();
12. }
```

(2) 定义 CStudent 学生类，实现 CPeople 接口中的所有方法。

```
1.  class CStudent implements CPeople{          //学生
2.     String sno;                              //学号
3.     String sname;                            //姓名
4.     CDate sdate;                             //出生日期
5.     double smark;                            //考试分数
6.     public CStudent(String sno,String sname,CDate sdate,double smark){
7.         this.sno=sno;
```

```
8.          this.sname=sname;
9.          this.sdate=sdate;
10.         this.smark=smark;
11.     }
12.     public String getSno(){return sno;}
13.     public void setSno(String sno){this.sno=sno;}
14.     public String getName(){return sname;}
15.     public void setName(String name){this.sname=name;}
16.     public CDate getDate(){return this.sdate;}
17.     public void setDate(CDate date){this.sdate=date;}
18.     public double getGrade(){return smark;}     //此时学生只有考试分数
19.     public String toString(){
20.         return "学号: "+sno+"\n 姓名: "+sname+"\n 出生日期: "+sdate+"\n 国籍: "
            +this.nationality+"\n";
21.     }
22. }
```

(3) 定义大学生类 CUniversityStudent 和研究生类 CGraduateStudent。大学生添加宿舍号字段,总评成绩根据平时成绩和考试分数评定;而研究生添加了宿舍号和导师字段,总评成绩根据导师成绩和考试分数评定。

```
1.  class CUniversityStudent extends CStudent{     //大学生类
2.      int roomNo;                                //宿舍号
3.      double usualGrade;                         //平时成绩
4.      public CUniversityStudent(int roomNo, String sno, String sname, CDate sdate,
        double smark,double usualGrade){
5.          super(sno,sname,sdate,smark);
6.          this.roomNo=roomNo;
7.          this.usualGrade=usualGrade;
8.      }
9.      public double getGrade(){return smark+this.usualGrade;}
                                                   //大学生成绩有平时成绩和考试成绩
10.     public String toString(){
11.         return "宿舍号: "+roomNo+"\n"+super.toString();
12.     }
13. }
14. class CGraduateStudent extends CStudent{       //研究生类
15.     int roomNo;                                //宿舍号
16.     String advisor;                            //导师
17.     double advisorGrade;                       //导师成绩
18.     public CGraduateStudent(int roomNo, String advisor, String sno, String
        sname,CDate sdate,double smark,double advisorGrade){
19.         super(sno,sname,sdate,smark);
20.         this.roomNo=roomNo;
21.         this.advisor=advisor;
22.         this.advisorGrade=advisorGrade;
23.     }
24.     public double getGrade(){return smark+this.advisorGrade;}
                                                   //研究生成绩有考试成绩和导师成绩
25.     public String toString(){
26.         return "宿舍号: "+roomNo+"\n 导师: "+advisor+"\n"+super.toString();
27.     }
28.}
```

(4) 在主函数中测试类的多态和接口回调。

```
1.    public class EXA5_1 {
2.    public static void main(String[] args) {
3.        //上转型对象和多态
4.          CStudent csu = new CUniversityStudent (211," 159074111"," 李 四", new
            CDate(1997,1,10),98.5,85.0);
5.        System.out.println("期末总评: "+csu.getGrade());
6.        System.out.println(csu.toString());
7.        csu=new CGraduateStudent(301,"王三强","156111","王五",new CDate(1993,
          11,23),89.0,90.0);
8.        System.out.println("期末总评: "+csu.getGrade());
9.        System.out.println(csu.toString());
10.
11.       //接口变量和接口回调
12.       CPeople cpu=new CUniversityStudent(511,"139074123","李大有",new
          CDate(1996,11,11),94.5,95.0);
13.       System.out.println("期末总评: "+cpu.getGrade());
14.       System.out.println(cpu.toString());
15.       cpu=new CGraduateStudent(301,"孙大四","156111","丁巳",new CDate(1994,
          11,21),85.0,95.0);
16.       System.out.println("期末总评: "+cpu.getGrade());
17.       System.out.println(cpu.toString());
18.   }
19. }
```

5.1.2 程序解析

本案例中设计一个接口——人员接口，以及学生类、大学生类、研究生类 3 个类，涉及以下知识点：

- Java 接口中只能定义常量和方法，方法只有方法头，没有方法体。
- 方法的实现类——学生类必须要实现接口中所有的方法，否则学生类不能创建对象。
- 由学生类派生大学生类和研究生类，其中 getGrade()和 toString()方法都实现了父类中同名方法的重写。
- 在主方法中，通过上转型对象调用不同子类中的同名方法，实现多态；通过接口变量调用实现类中的同名方法，实现接口回调，请参考 5.6 节和 5.7 节的相关内容。

5.2 继承

继承是面向对象程序设计的基本特征，继承是利用已有的基类（父类）派生出新的派生类（子类），对基类进行功能的扩充。派生类可以不用定义而直接使用继承基类中的属性和方法，当然，在此基础上派生类通常要添加一些新的属性和方法，以满足新的设计需求。

继承的目的是程序代码的重用，面向对象程序设计方法的核心思想就是利用已存在的类的属性和方法，通过继承创建出功能更为强大的新类，这样会节省程序开发的时间、加快开发进度、降低代码的出错率。

大家都知道，类是利用分类的思想，类也有包含关系。例如，汽车可以包含轿车、卡车、特种车辆等，而轿车又可以包含小轿车、越野车，小轿车又可以包含两厢和三厢……这些都

属于包含关系(is-a 关系)。按照这种分类方式,如果已经定义汽车类别,那么在定义轿车时,作为汽车共同的特征在轿车类中就不需要重复定义了。否则,汽车类、轿车类、三厢轿车类中都单独列出了所有的特征,那么相当多的内容就会重复定义,不利于代码的重用和编程效率的提高。

在程序设计中,如果设计的类可以已存在的类为基础进行设计,那么该特性即为面向对象编程思想中的继承性,Java 的继承有以下几个特征。

- Java 的单继承性:Java 不支持多重继承,只支持单继承。也就是说,Java 的类只能有一个父类。Java 支持多层继承,如人员类 CPeople 可以派生学生类 CStudent,学生类 CStudent 还可以派生出自己的子类大学生类 CUniversityStudent,当然大学生类 CUniversityStudent 还可以派生出研究生类 CGraduateStudent。
- 继承关系的传递性:CGraduateStudent 类继承 CUniversityStudent 类,而 CUniversityStudent 类继承 CStudent 类,则 CGraduateStudent 类中不仅有从 CUniversityStudent 类继承下来的属性和方法,还有从 CStudent 类继承下来的属性和方法,当然,CGraduateStudent 类还有自己定义的属性和方法,这些都是 CGraduateStudent 类拥有的属性和方法。
- 类的层次结构:多层继承不仅简化了类的设计,还能清晰地反映相关类间的层次结构关系。
- 软件复用:软件复用(Software Reuse)是将已有软件的各种有关知识用于建立新的软件,以减少软件开发和维护的花费。软件复用是提高软件生产力和质量的一种重要技术。早期的软件复用主要是代码级复用,被复用的知识专指程序,后来扩大到包括领域知识、开发经验、设计决定、体系结构、需求、设计、代码和文档等的有关方面。
- 增强程序的易维护性:继承通过软件设计的一致性减少了软件模块间的接口和界面的设计工作量,增强了程序的易维护性。

5.2.1　创建子类

在 Java 语言中,创建子类通过继承来声明子类,使用 extends 关键字来创建一个类的子类,语法格式如下。

【格式 5-1】 创建子类的语法格式。

```
访问控制符 [修饰符] class 子类名标识符 extends 父类名标识符{
    访问控制符 [修饰符] 数据类型  成员变量 1;
    …
    访问控制符 [修饰符] 数据类型  成员变量 n;
    [访问控制符] 子类名 (参数 1,参数 2,…){              //构造方法
        代码块
    }
    访问控制符 [修饰符] 返回值的数据类型 方法名称(参数 1,参数 2,…){
        代码块
        [return 返回值;]
    }
}
```

(1) extends 关键字表明父类和子类之间的继承关系。extends 关键字前面是要创建的

子类，后面是要继承的父类（父类必须已经存在），Java 采用单继承，所以 extends 后面只能写一个父类名称。

（2）子类继承父类中的属性和方法，当然也可以定义自己的属性和方法。如果子类和父类中的属性或方法同名，那么父类的同名属性或方法会被隐藏或重写。

注意：如果不写 extends 父类名标识符，则当前创建的类会自动继承 Object 类，所有 Java 类都直接或间接地继承 java.lang.Object 类。当创建一个类 Car 时，如果没有写父类，相当于 class Car extends Object。

在例 5-1 中，创建子类 CUniversityStudent 的类头为“class CUniversityStudent extends CStudent”，父类为 CStudent；类 CUniversityStudent 继承了父类的属性 sno、sname、sdate、smark，以及方法 getSno、setSno 等，新增了属性 roomNo、usualGrade，重写了方法 getGrade、toString。

5.2.2 子类的继承性

子类的继承性随着访问控制符、同包或不同包而不同。子类的继承性分为两种情况，即子类和父类在同一包中；子类和父类不在同一包中。

1. 子类和父类在同一包中

如果子类和父类在同一包中，则子类不能直接访问父类中 private 的属性和方法，但是可以调用非 private 的属性和方法。private 成员在子类中不可见，不能直接访问，非 private 成员在子类中的访问权限保持不变。子类和父类定义在同一个文件中，肯定在同一个包中，当然可以定义在不同的文件中，也可以在同一个包中。

【例 5-2】 点类 Point 是线类 Line 的父类。

```
1.  package cn.ahut.cs.mainapp.chapter5;
2.  import java.util.*;
3.  class Point{
4.     private int x;
5.     int y;
6.     void setX(int x){this.x=x;}
7.     int getX(){return x;}
8.     void print(){
9.         System.out.println("("+x+","+y+")");
10.     }
11. }
12. class Line extends Point{
13.     //继承 Point 类中 x 不可见，y 是可见的；方法都是可见的
14.     double getLen(Point p){
15.         //错误：成员变量 p.x 和 this.x 不可见
16.         //double Len=Math.sqrt(Math.pow(this.x-p.x,2)+Math.pow(this.y,p.y));
17.         //修改成下面的语句
18.         double Len=Math.sqrt(Math.pow(this.getX()-p.getX(),2)+Math
            .pow(this.y,p.y));
19.         return Len;
20.     }
21. }
22. public class EXA5_2 {
23.     public static void main(String[] args) {
```

```
24.         //Point 类的对象创建和成员访问
25.         Point p=new Point();
26.         p.setX(10);p.y=20;
27.         p.print();
28.         //Line 类的对象创建和成员访问
29.         Point p1=new Point();
30.         p1.setX(20);p.y=40;
31.         Line line=new Line();
32.         System.out.printf("距离: %10.2f\n",line.getLen(p1));
33.     }
34. }
```

Point 类中的 x 成员变量是私有的，在派生子类中不可以直接访问，必须通过 getX()方法进行访问。

2. 子类和父类不在同一包中

如果子类和父类不在同一包中，则子类不能直接访问父类中 private、友好的属性和方法，但是可以调用 protected、public 的属性和方法。private、友好的成员在子类中不可见，不能直接访问，protected、public 的成员在子类中的访问权限保持不变。子类和父类不能定义在同一个文件中。

【例 5-3】 点类 Point 是线类 Line 的父类，两个文件不在同一包中。

```
1.  package cn.ahut.cs.mainapp.chapter5.sub;
2.  public class Point2{
3.     private int x;
4.     public int y;                          //添加 public
5.     //void setX(int x){this.x=x;}          //友好成员不能被不同包中的类继承
6.     public void setX(int x){this.x=x;}     //添加 public
7.     public int getX(){return x;}           //添加 public
8.     public void print(){                   //添加 public
9.         System.out.println("("+x+","+y+")");
10.    }
11. }
12. package cn.ahut.cs.mainapp.chapter5;
13. import java.util.*;
14. import cn.ahut.cs.mainapp.chapter5.sub.Point2;
15. class Line2 extends Point2{
16.     //继承 Point 类中 x 不可见，y 是可见的;方法都是可见的
17.     double getLen(Point2 p){
18.         //错误: 成员变量 p.x 和 this.x 不可见
19.         //double Len=Math.sqrt(Math.pow(this.x-p.x,2)+Math.pow(this.y,p.y));
20.         //修改成下面的语句
21.         double Len=Math.sqrt(Math.pow(this.getX()-p.getX(),2)+Math.pow
            (this.y,p.y));
22.         return Len;
23.     }
24. }
25. public class EXA5_3 {
26.     public static void main(String[] args) {
27.         //Point 类的对象创建和成员访问
28.         Point2 p=new Point2();
29.         p.setX(10);p.y=20;
30.         p.print();
```

```
31.         //Line 类的对象创建和成员访问
32.         Point2 p1=new Point2();
33.         p1.setX(20);p.y=40;
34.         Line2 line=new Line2();
35.         System.out.printf("距离: %10.2f\n",line.getLen(p1));
36.     }
37. }
```

在包 cn.ahut.cs.mainapp.chapter5.sub 中定义类 Point2，其中的友好成员（在 EXA5_2 中）修改为 public 成员，那么在另外一个包 cn.ahut.cs.mainapp.chapter5 中的派生子类 Line2 中可以访问。

为什么没有修改成 protected 保护型呢？

因为，如果修改成保护型，在子类 Line2 中访问没有问题，但是在主类 EXA5_3 中的 p.y＝20、p.setX()、getLen()成员就不能访问了。

5.2.3 子类对象的内存构造

子类继承父类的成员变量和方法，继承的私有成员在子类中是不可见的，那么私有成员在子类中分配空间吗？

答案是肯定的。父类中的成员变量在创建子类对象时都分配了内存空间，但只是一部分在子类中可以访问。例如，父类的私有成员在继承时，子类对象尽管分配了空间，但在子类中也是不可见的，即不可访问，如图 5.1 所示。

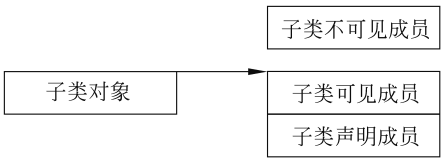

图 5.1 子类对象的内存空间构造

在图 5.1 中，在子类中可以访问继承父类中的可见成员，对于不可见成员是不可以直接访问的，当然子类中有自己声明的新的成员变量和方法。

注意：子类不可见成员分配变量内存空间吗？

答案是分配的。那么，既然分配，岂不是浪费空间吗？

当然不是。子类创建对象时，继承的父类中的所有成员属性都分配了内存空间，但只有可见的成员变量通过子类对象才可以访问。例如私有成员变量，在创建子类对象时分配了内存空间，但是子类不可以直接访问，这时候没有父类对象，也不是某个父类对象的成员，那么，这部分内存空间就像垃圾空间一样，似乎浪费了，但事实不是如此。在例 5-3 中，Point2 中的 x 变量被定义为 private，在子类中就不能通过变量名来读取或修改 x 变量的值，必须通过对外接口 getX 和 setX 成员方法来访问。如果在 Line2 中不分配 x 变量的空间，那么线段两端的点从何而来。所以子类继承父类中的所有成员，只是有些成员是不可以直接访问的，即不可见，而有些成员是可以访问的，但成员变量都是分配空间的。

5.2.4 父类与子类的同名成员

当在子类中定义和父类中同名的成员变量和方法时，会产生成员变量隐藏和成员方法的重写两种情况。

1. 子类隐藏父类中的成员变量

如果子类中定义了与父类中同名的成员变量，那么子类就隐藏了从父类中继承的成员

变量，即子类重新定义了这个同名的成员变量。

【例 5-4】 定义父类长方形，派生子类正方形，子类中定义了和父类同名的成员变量 width。

```
1.  package cn.ahut.cs.mainapp.chapter5;
2.  class CRect{                                    //长方形类
3.     int width,length;                            //长和宽
4.     double getArea(){
5.         return width * length;
6.     }
7.  }
8.  class CSquare extends CRect{                    //正方形类
9.     double width;                               //只有一个边长，变量名和父类同名，但是类型不同
10.    double getArea(){
11.        return width * width;
12.    }
13. }
14. public class EXA5_4 {
15.    public static void main(String[] args) {
16.        //长方形
17.        CRect r=new CRect();
18.        r.width=10;
19.        r.length=20;
20.        System.out.println("长方形的面积: "+r.getArea());
21.        //正方形
22.        CSquare s=new CSquare();
23.        s.width=20.4;
24.        System.out.printf("正方形的面积: %10.2f\n",s.getArea());
25.    }
26. }
```

运行结果如下：

```
长方形的面积: 200.0
正方形的面积: 416.16
```

注意：

(1) 子类中重新定义了继承父类中的同名成员变量，类型不必相同。

(2) 父类中被隐藏的成员变量在子类中仍然可以操作，使用“super.同名成员变量名”就可以访问被隐藏的父类中的同名成员变量。

2. 子类重写父类中的方法

子类可以隐藏父类中的同名成员变量，也可以隐藏从父类中继承的方法。如果子类和父类中的某个方法的定义完全一致，那么子类就重写了父类中被隐藏的同名方法，这就构成了方法重写或方法覆盖。

方法重写是指子类中定义了一个方法，并且这个方法的方法名、返回类型、参数个数、参数类型和从父类中继承的方法完全相同。

那么，方法重写有什么作用呢？通过方法重写，子类可以把父类的状态和行为改变为自身的状态和行为，如例 5-4 中同名了方法 getArea()，父类长方形中是长和宽的乘积，而子类正方形中是边长和边长的乘积，所以子类通过重写父类中的同名方法实现适合自己行为的

方法体。

注意：

（1）父类中被隐藏的成员方法在子类中仍然可以操作，使用“super.同名成员方法()”就可以访问被隐藏的父类中的同名成员方法。

（2）重写要求方法的声明必须完全一致，即方法头必须完全相同。例 5-4 中长方形类中的 getArea 方法头如果修改为 int getArea()，那么在子类中就会出现方法覆盖不兼容错误，这样就构不成重写，也构不成方法重载。

（3）父类和子类的方法可以构成方法重载，例如下面的程序段：

```
1.  class A{
2.     int f(int x,int y){
3.     //public int f(int x,int y){
4.        return x+y;
5.     }
6.  }
7.  class B extends A{
8.     double f(int x,double y){
9.        return x+y;
10.    }
11. }
```

（4）子类方法不能缩小父类方法的访问权限。例如，上面程序段中在“int f(int x,int y)”前面加上 public 修饰符，而在派生子类中的 f 函数前面没有 public 修饰符，程序会报错“覆盖时不能降低继承方法的可视性”，即 f 方法的权限缩小，程序正在尝试分配更低的访问权限，父类中的权限是 public，而子类是友好访问权限。

（5）父类的静态方法不能被子类重写为非静态方法，父类的非静态方法也不能被子类重写为静态方法。子类可以定义与父类的静态方法同名的静态方法，子类重写并隐藏了父类的同名静态方法。例如下面的程序段：

```
1.  class CRect{                              //长方形类
2.     int width,length;                      //长和宽
3.     static double  getArea(){
4.        return 1.1;                         //width * length;
5.     }
6.  }
7.  class CSquare extends CRect{              //正方形类
8.     double width;                       //只有一个边长，变量名和父类同名，但是类型不同
9.     static double getArea(){
10.       return 0.5;                         //width * width;
11.    }
12. }
```

（6）在这段代码中，getArea 在父类和子类中都为静态方法，如果有一个类中不是静态方法就会报错。如果去掉长方形类中该方法的 static 修饰符，那么会报错“静态方法不能隐藏父类中的实例方法”；如果去掉正方形类中该方法的 static 修饰符，那么会报错“实例方法不能覆盖父类中的静态方法”。

（7）那么，在长方形类中该方法的语句“return width * length”为什么会报错呢？这是因为静态方法中不能引用非静态成员。

(8) 父类中的私有成员方法不能被子类重写。

5.3　关键字 this 和 super

this 关键字和 super 关键字在前面程序中多次使用,它们的作用比较难以理解,而且使用方法多变。this 表示当前正在创建的对象本身,通常使用 this 来引用本类中定义的成员变量或成员方法,用于区别同名的形参变量、同名的局部变量等。super 表示当前类的父类,用于引用父类中和子类同名的成员变量和成员方法,以及构造方法等。

5.3.1　在构造方法和实例方法中使用 this

1. 用 this 引用类中的成员

【格式 5-2】 用 this 引用类中成员的格式。

```
this.成员变量;
this.成员方法;
```

【例 5-5】 用 this 引用本类中的成员。

```
1.  package cn.ahut.cs.mainapp.chapter5;
2.  class CRect55{
3.     int width,length;
4.     void display(){
5.         this.width=10;
6.         this.setLength(20);
7.         System.out.println("宽="+this.width+"\n 长="+this.length);
8.     }
9.     void setLength(int L){
10.    //static void setLength(int L){
11.        this.length=L;
12.    }
13. }
```

static 修饰的方法是静态方法,在静态方法中不能引用 this。如例 5-5 中的第 10 行语句,如果前面加上 static 修饰符,setLength()方法就从实例方法变为类方法,大家都知道类方法不需要创建对象就可以通过类名来访问,但此时 length 实例变量可能还没有被分配内存空间,更谈不上访问了,所以类方法中是不能使用 this 关键字的。

2. 用 this 区别形参局部变量和成员变量同名问题

当形参名称和类中的成员变量同名时,使用“this.成员变量名”的格式引用成员变量名,从而区分形参。

【例 5-6】 形参和类中成员变量同名。

```
1.  package cn.ahut.cs.mainapp.chapter5;
2.  class CRect56{
3.     int width,length,x=12;
4.     CRect56(int width,int length){
5.        this.width=width;
6.        this.length=length;
7.        double x=12.56;                    //方法中定义的局部变量
8.        this.x=(int)x;
```