第5章　继　　承

继承是面向对象程序设计(OOP)的三大特征之一,描述了类不同抽象级别之间的关系:“is a”的关系,即“特殊与一般”的关系。换句话说,一般(父类)是特殊(子类)更高级别的抽象。子类可以继承父类所有的非 private 类型的属性和方法,也可以具有自己独有的属性和方法。通过类的继承关系,使公共的特性能够共享,提高了软件的重用性。但在 Java 中只允许单继承。

5.1　继承的语法

在 Java 中描述两个类之间的继承关系时,使用关键字 extends,格式如下:

```
class SubClass extends SuperClass{
…
}
```

其中 SubClass 为子类,SuperClass 为父类(或超类)。

在第 4 章中定义了 Person 类:

```
class Person{
    private String name;
    private char sex='M';
    Person(String name){
        this.name=name;
        }
    Person(String name,char sex){
        this.name=name;
        this.sex=sex;
        }
    public void show(){
        String str="下面展示姓名和性别";
        System.out.println(str);
        System.out.println("姓名: "+name+" 性别: "+sex);
        }
}
```

现在要定义一个学生类(Stu),由于“学生是人”,所以学生类和 Person 类之间是“is a”的关系,即“继承”关系,那么就可以这样来定义 Stu 类:

例 5.1　Stu 类的定义(ch05\Stu.java)。

```
class Stu extends Person {
    long id;
```

```
    private String name;          //仅为演示用,实际编程中无须声明该变量
    private char sex='M';
    public Stu (String name, long id, char sex){
        super(name,sex);
        this.id=id;
    }
}
```

前面讲过子类可以继承父类的非 private 类型的属性和方法,在这个例子中可以看到:虽然在 Person 类中定义了 name、sex 属性,但它们是 private 类型的数据,如果 Stu 也想拥有这些属性的话,就必须重新定义,不能继承于父类,也可以定义这些属性为 protected;但 show 方法,在 Person 类中是以 public 的身份定义的,所以 Stu 类虽然没有显式地定义该方法,但却拥有该方法,因为它继承了父类的 show()方法,编写测试类如下:

```
class UseStu{
    public static void main(String[] args){
        Stu s=new Stu("王强",20094140213L,'M');
        s.show();
        }
}
```

运行结果:

```
下面展示姓名和性别
姓名:王强 性别:M
```

另外,还可以在子类中定义子类独有的属性和方法,例如本例中的 id。

在这里需要说明以下几点:

(1) 父类的构造函数不能被子类继承。

(2) 子类不能继承或访问父类中的 private 属性和方法。

(3) 父类中的 friendly(包访问权限)的属性和方法只有在父类和子类在同一包中时,才能被子类继承和访问。

(4) 父类中由 protected 或 public 修饰的属性和方法,都可以被子类继承访问(无论子类是否与父类在同一包中)。

5.2 成员变量的隐藏和方法的覆盖

当子类和父类中定义的成员变量的名字相同时,子类可以隐藏父类的成员变量。同样,子类也可以通过方法重写(或称为覆盖,Overriding)来隐藏从父类继承的方法。方法覆盖是指子类中定义的方法的头部(方法的名字、返回类型、参数个数和类型)与父类的方法完全相同,而方法体可以不同。但在进行方法覆盖时,请注意:在覆盖时访问权限只能放大或相同,不能缩小;覆盖方法不能抛出新的异常(关于异常,将在第 8 章介绍)。

例 5.2 成员变量的隐藏和方法的覆盖(ch05\OverriddingTest.java)。

```
class Father{
```

```
    String s="Father";
    int i=1;
    public void f(){
        System.out.println("Father s="+s);
        System.out.println("Father i="+i);
        }
    }
class Child extends Father{
    String s="Child";                    //隐藏了父类的成员变量 s
    public void f(){                     //覆盖了父类的 f()方法,但访问权限只能是 public
        System.out.println("Child s="+s);
        System.out.println("Child i="+i);
        }
    }
class OverriddingTest{
    public static void main(String[] args){
        Father f=new Father();
        Child  c=new Child();
        f.f();
        c.f();
        }
    }
```

运行结果：

```
Father s=Father
Father i=1
Child s=Child
Child i=1
```

方法覆盖与方法重载的区别如下：方法覆盖发生在父类和子类之间，即子类重写了父类的某个方法，子类中定义的方法的头部(方法的名字、返回类型、参数个数和类型)与父类的方法完全相同，而方法体可以不同；重载是在同一类中出现的现象，是指一个类中可以有多个方法具有相同的名字，但这些方法的参数不同。

5.3 super

如果想在子类中使用父类的非 private 类型的变量和方法(特别是被隐藏变量和方法)，可以使用 super 关键字。例如，要在例 5.2 中访问父类的变量 s，就要使用 super.s，请试验在子类 Child 的 f()方法中加入如下的代码，查看输出结果。

```
System.out.println("Father s="+super.s);
super.f();
```

如果要在子类的构造方法中访问父类的构造方法，也要使用 super 关键字，例如例 5.1 中的 super(name,sex)；但要注意，该调用语句必须出现在子类构造方法非注释语句的第

一行。

注意：如果在子类的构造方法中，没有使用 super 调用父类的构造方法，编译器将自动添加：

```
super();
```

即调用父类不带参数的构造方法，此时就应保证父类中有不带参数的构造方法（当父类未定义任何构造方法时，系统会自动合成；一旦父类定义了一个或多个构造方法，系统将不再提供默认的构造方法，必须手工定义），否则就会产生错误。如例 5.1 中，由于父类 Person 未定义无参的构造方法，所以必须用 super(name,sex)显式地调用父类中某个已定义的构造方法。

5.4 final

final 关键字可以用来修饰类、方法、变量（包括成员变量和局部变量及方法中的参数）。

（1）当 final 修饰类时，意味着该类不能被继承，即该类不能有 String 类等子类。

例 5.3 final 修饰类（ch05\FinClass.java）。

```
final class FinClass{                                    //最终类
    int i;
    FinClass(){
        System.out.println("This is a final class.");
        }
    }
class SubFinClass extends FinClass{                      //错误,不能从最终类继承
    }
```

编译时会出现下面的出错信息：

```
FinClass.java:7: 无法从最终 FinClass 进行继承
class SubFinClass extends FinClass{
```

（2）当 final 修饰方法时，代表该方法不能被重写。

（3）当 final 修饰成员变量时，该变量可以理解为常量，必须赋以初值（可在声明时赋值，或在类的构造方法中赋值），并且该变量的值不能再改变；当 final 修饰局部变量时，该局部变量只能被赋一次值；当 final 修饰方法中的参数时，该参数的值不能被改变。

例 5.4 final 修饰方法和变量（ch05\UseFinal.java）。

```
class UseFinal{
    final int i=1;
    final int j;          //最终变量若不在声明时赋值,就要在其所属的类的构造方法中赋值
    int k;
    UseFinal(){
        j=2;
        }
    final void f(){       //最终方法,在子类中不能被覆盖
```

```
        System.out.println("This is a final method.");
        }
    void g(){
        //i++;错误,不能重新指定最终变量的值
        //j++;错误,不能重新指定最终变量的值
        k++;
        final String s="Hello ";
        //s="Hi";错误,当 final 修饰局部变量时,该变量只能被赋一次值
        final String str;
        str="Java";
        System.out.println(s+str+"  i="+i+"  j="+j+"  k="+k);
        }
    void h(final int a){
        //a++;错误,不能指定最终参数
        System.out.println("a="+a);
        }
    public static void main(String[] args){
        UseFinal uf=new UseFinal();
        uf.f();
        uf.g();
        uf.h(100);
        }
    }
```

5.5 多　　态

多态是 OOP 的三大特征之一,此处结合上节讲述的覆盖来理解多态的含义。当一个类(如 Instrument 类)有多个子类(Wind、Percussion、Stringed),并且这些类都重写了父类中的某个方法(void play()方法),如图 5.1 所示。那么根据前面所讲的内容,下面的代码很容易理解。

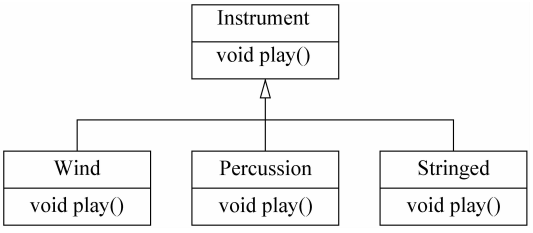

图 5.1　Instrument 及其子类

```
Wind w=new Wind();                          //产生 Wind 类的对象
w.play();                                   //调用 Wind 类中的 play 方法
Percussion p=new Percussion();              //产生 Percussion 类的对象
p.play();                                   //调用 Percussion 类中的 play 方法
Stringed s=new Stringed();                  //产生 Stringed 类的对象
s.play();                                   //调用 Stringed 类中的 play 方法
```

那么下面的代码又如何理解呢？

```
Instrument insw=new Wind();
insw.play();
```

把 Wind 类的对象赋值给 Instrument 类型的变量(insw)对吗？insw 调用的是 Instrument 类中的 play 方法，还是 Wind 类中的 play 方法呢？

子类和父类之间的关系是“is a”的关系，即“特殊与一般”的关系，可以说管乐器(Wind)是乐器(Instrumen)，打击乐器(Percussion)是乐器(Instrument)，弦乐器(Stringed)是乐器(Instrument)，所以下面的代码是正确的。

```
Instrument insw=new Wind();
Instrument insp=new Percussion();
Instrument inss=new Stringed();
```

这就是常说的向上转型(upcasting)。向上转型后的对象(简称上转型对象)，如 insw、insp、inss。例如 play()，在调用方法时，其实调用的仍是子类中所重写的 play 方法，而不是父类的 play 方法。这是因为 Java 对 Override 方法调用采用的是运行时绑定，也就是按照对象的实际类型来决定调用的方法，不是按照对象的声明类型来决定调用的方法。但 Overload 方法则相反，在编译时已经进行了方法绑定，按照对象的声明类型决定调用的方法。

例 5.5 多态示例(ch05\Music.java)。

```
class Instrument {
    public void play() {
        System.out.println("Instrument.play()");
    }
}
class Wind extends Instrument {
    public void play() {
        System.out.println("Wind.play()");
    }
}
class Percussion extends Instrument {
    public void play() {
        System.out.println("Percussion.play()");
    }
}
class Stringed extends Instrument {
    public void play() {
        System.out.println("Stringed.play()");
    }
}
public class Music {
    static void tune(Instrument i) {
        i.play();
```

```
    }
    public static void main(String[] args) {
        Instrument[] ins=new Instrument[3];
        int i=0;
        ins[i++]=new Wind();
        ins[i++]=new Percussion();
        ins[i++]=new Stringed();
        for( i=0; i<ins.length; i++)
            tune(ins[i]);
    }
}
```

运行结果：

```
Wind.play()
Percussion.play()
Stringed.play()
```

那么能否将父类的对象赋值给子类类型的变量呢？答案是否定的，除非进行强制类型转换。例如：

```
Wind w=new Instrument();                                    //错误
Wind w= (Wind) new Instrument();                            //正确
```

5.6 继承与组合

通过使用继承，提高了类的可重用性，减少了代码的重复书写，提高了效率。除了继承这种方式外，还可以通过组合的方式来重复使用类。所谓组合，就是在一个新类中创建已有类的对象，即新类由已有类的对象组成。例如，下面的学生成绩管理程序，Score 类中使用了已有类(Student 类和 Course 类)中的对象，所以这种重复使用类的方式就是组合。

例 5.6 组合示例，学生成绩管理程序(ch05\StuApp.java)。

```
package app;
//学生类
class Student {
    String name;
    long id;
    public Student() {
    }
    public Student(String name,long id){
        this.name=name;
        this.id=id;
    }
}
//课程类
class Course{
```

```
    long id;
    String name;
    Course(long id, String name){
        this.id=id;
        this.name=name;
    }
}
//成绩类
class Score{
    Student stu;
    Course course;
    double grade;
    Score(Student stu,Course course,double grade){
        this.stu=stu;
        this.course=course;
        this.grade=grade;
    }
}
//应用类
class StuApp{
    private static Course[] courses=new Course[5];
    private static Student[] stus=new Student[10];
    private static Score[] scores=new Score[50];
    //添加课程
    private static int coursesIndex=0;
    public static void addCourse(Course course){
        if(coursesIndex>courses.length) {
            System.out.println("too many courses");
            return;
        }
        courses[coursesIndex++]=course;
    }
    //添加学生
    private static int stusIndex=0;
    public static void addStu(Student stu){
        if(stusIndex>stus.length) {
            System.out.println("too many student");
            return;
        }
        stus[stusIndex++]=stu;
    }
    //添加学生的成绩
    private static int scoreIndex=0;
    public static void addScoreToStu(Student stu,String courseName,double grade){
            if(scoreIndex>scores.length){
```

```
                System.out.println("too many student");
                return;
            }
        for(int i=0;i<=coursesIndex;i++){
                if(courses[i].name.equals(courseName)){
                    scores[scoreIndex++]=new Score(stu,courses[i],grade);
                    break;
                }
            }
    }

    public static void main(String[] args){
        //添加课程
        Course java=new Course(1,"java");
        addCourse(java);
        Course os=new Course(2,"os");
        addCourse(os);
        Course math=new Course(3,"math");
        addCourse(math);
        //添加学生张三并为其添加成绩
        Student stua=new Student("张三",1L);
        addStu(stua);
        addScoreToStu(stua,"java",86);
        addScoreToStu(stua,"math",89.5);
        //添加学生李四,并为其添加成绩
        Student stub=new Student("李四",2L);
        addStu(stub);
        addScoreToStu(stub,"os",100);
        //打印标题栏
        System.out.println("学号\t 姓名\t\t 课程名\t\t 成绩");
        //显示学生的成绩
        for(int i=0;i<scoreIndex;i++){
            StringBuilder sb=new StringBuilder();
            sb.append(scores[i].stu.id).append("\t").append(scores[i].stu.name);
            for(int j=1;j<=16-(scores[i].stu.name.getBytes()).length;j++)
            sb.append(" ");
            sb.append(scores[i].course.name);
            for(int k=1;k<=16-(scores[i].course.name.getBytes()).length;k++)
                sb.append(" ");
            sb.append(scores[i].grade);
            System.out.println(sb);
        }
    }
}
```

对于重复使用类的两种方式——继承和组合,如何选择呢?这里给出一个小诀窍:如

果类之间的关系是“B is a A”的关系，如猫是动物，弦乐器是乐器等，那么 A、B 之间就应该是继承的关系，即 B 继承 A，B 是子类，A 是父类。如果类之间的关系是“A has a B ”，那么 A、B 之间的关系就应该是组合的关系，例如汽车有门、窗、引擎等，那么在定义汽车（A）类时，就可以采用组合的方式，即将门类、窗类、引擎类的对象作为汽车的成员。

5.7 初始化顺序*

在第 4 章曾讲过，类在加载后，其内部变量的初始化顺序是按照它们在类中的定义的顺序进行的，当遇到静态变量或语句块时，静态数据要先于实例变量（静态数据在加载类时只初始化一次，而实例变量则是在创建对象时初始化的，也就是每创建一个对象，就要初始化其实例变量），并且变量的初始化总是在方法（包括构造方法）调用前进行。那么类在何时会被加载呢？当一个类首次被使用时系统就会加载该类。首次使用类的情况可以是以下方式之一：

（1）第一次创建该类的对象；

（2）首次访问该类的静态变量或静态方法时。

为了更好地理解变量的初始化顺序，请看下面的例子。

例 5.7 初始化顺序(1)(ch05\InitOrder.java)。

```
class Star {
    Star(int marker) {
        System.out.println("Star(" +marker +")");
    }
    void f1(int marker) {
        System.out.println("f(" +marker +")");
    }
}
class Flag {
    Star s1=new Star(11);
    Flag() {
        System.out.println("Flag()");
        s3=new Star(555);
        s3.f1(66);
    }
    static Star s2=new Star(22);
    void f() {
        System.out.println("f()");
    }
    Star s3=new Star(33);
}
public class InitOrder {
        Star s1=new Star(1);
    public static void main(String[] args) {
        System.out.println("Creating new Flag() in main");
```