第 5 章　继承与接口

扫一扫

视频讲解

主要内容：

- 子类与父类；
- 子类的继承性；
- 子类对象的构造过程；
- 成员变量的隐藏和方法重写；
- super 关键字；
- final 关键字；
- 对象的上转型对象；
- 继承与多态；
- abstract 类与 abstract 方法；
- 接口。

难点：

- 成员变量的隐藏和方法重写；
- 继承与多态。

第 4 章主要学习了类和对象的有关知识，讨论了类的构成以及用类创建对象等内容，主要体现了面向对象编程的一个重要特点——数据的封装。面向对象编程的另外两个特点是继承和多态，本章将讲述关于这两方面的重要内容——类的继承与多态、接口的实现与多态。

扫一扫

视频讲解

5.1　子类与父类

继承是一种由已有的类创建新类的机制。利用继承，可以先编写一个共有属性的一般类，再根据该一般类编写具有特殊属性的新类，新类继承一般类的状态和行为，并根据需要增加新的状态和行为。由继承得到的类称为子类，被继承的类称为父类（超类）。Java 不支持多重继承（子类只能有一个父类）。

在类的声明中，通过使用关键字 extends 来声明一个类的子类，格式如下：

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

例如：

```
class Student extends People {
    ⋮
}
```

把 Student 类声明为 People 类的子类，People 类是 Student 类的父类。

如果一个类的声明中没有使用 extends 关键字，这个类被系统默认为是 Object 的子类。

Object 是 java.lang 包中的类。

扫一扫

视频讲解

5.2 子类的继承性

我们已经知道类可以有两种重要的成员：成员变量和方法。子类中的成员变量或方法有一部分是子类自己声明的，另一部分是从父类继承的。那么，什么叫继承呢？所谓子类继承父类的成员变量作为自己的一个成员变量，就好像它们是在子类中直接声明了一样，可以被子类中自己定义的任何实例方法操作；所谓子类继承父类的方法作为子类中的一个方法，就像它们是在子类中直接定义了一样，可以被子类中自己定义的任何实例方法调用。也就是说，如果子类中定义的实例方法不能操作父类的某个成员变量或方法，那么该成员变量或方法就没有被子类继承。

▶ 5.2.1 子类和父类在同一包中的继承性

如果子类和父类在同一个包中，那么，子类自然地继承了其父类中不是 private 的成员变量作为自己的成员变量，并且也自然地继承了父类中不是 private 的方法作为自己的方法，继承的成员变量或方法的访问权限保持不变。

▶ 5.2.2 子类和父类不在同一包中的继承性

如果子类和父类不在同一个包中，那么，子类继承了父类的 protected、public 成员变量作为子类的成员变量，并且继承了父类的 protected、public 方法作为子类的方法，继承的成员或方法的访问权限保持不变。如果子类和父类不在同一个包中，子类不能继承父类的友好变量和友好方法。

例 5.1 中有 4 个源文件：Father.java、Son.java、Grandson.java 和 Example5_1.java。其中，Father 类的包名是 england.people；Son 类的包名是 american.people；Grandson 类的包名是 japan.people；主类 Example5_1 的包名是 england.people。需要分别打开文本编辑器编写、保存这些源文件。Father.java 和 Example5_1.java 保存到 C:\ch5\england\people；Son.java 保存到 C:\ch5\american\people；Grandson.java 保存到 C:\ch5\japan\people(有关包的知识点见 4.8 节)。程序运行效果如图 5.1 所示。

```
C:\ch5>javac england/people/Example5_1.java

C:\ch5>java england.people.Example5_1
儿子：一双大手,180
孙子：一双小手,一双小脚,155
```

图 5.1 子类的继承性

例 5.1

Father.java

```
package england.people;
public class Father {
      private int money;
      protected int height;
      int weight;
}
```

Son.java

```
package american.people;
import england.people.Father;
public class Son extends Father {
```

```
        public String hand;
        public String getHand() {
            return hand;
        }
}
```

Grandson.java

```
package japan.people;
import american.people.Son;
public class Grandson extends Son {
        public String foot ;
}
```

Example5_1.java

```
package england.people;
import american.people.Son;
import japan.people.Grandson;
public class Example5_1 {
      public static void main(String args[]) {
          Son son = new Son();
          Grandson grandson = new Grandson();
          son.height = 180;
          son.hand = "一双大手";
          grandson.height = 155;
          grandson.hand = "一双小手";
          grandson.foot = "一双小脚";
          String str = son.getHand();
          System.out.printf("儿子:%s,%d\n",str,son.height);
          str = grandson.getHand();
          System.out.printf("孙子:%s,%s,%d\n",str,grandson.foot,grandson.height);
      }
}
```

▶ 5.2.3 protected的进一步说明

一个类A中的protected成员变量和方法可以被它的直接子类和间接子类继承,例如B是A的子类,C是B的子类,D又是C的子类,那么B、C和D类都继承了A类的protected成员变量和方法。在没有讲述子类之前,我们曾对访问修饰符protected进行了讲解,现在需要对protected总结得更全面些。如果用D类在D本身中创建了一个对象,那么该对象总是可以通过“.”运算符访问继承的或自己定义的protected变量和protected方法的,但是,如果在另外一个类中,例如在Other类中用D类创建了一个对象object,该对象通过“.”运算符访问protected变量和protected方法的权限如下列(1)、(2)所述。

(1) 对于子类D中声明的protected成员变量和方法,如果object要访问这些protected成员变量和方法,只要Other类和D类在同一个包中就可以了。

(2) 如果子类D的对象的protected成员变量或protected方法是从父类继承的,那么就要一直追溯到该protected成员变量或方法的“祖先”类,即A类;如果Other类和A类在同一个包中,object对象能访问继承的protected变量和protected方法。

例如,将例5.1中的Example5_1.java中的包语句:

```
package england.people;
```

删除,使得Example5_1成为无名包(保存在C:\ch5中),那么,分配给对象son和grandson的

weight 是从 Father 类继承的，而 son 和 grandson 对象出现在 Example5_1 类，Example5_1 类的包名（无名包）和 Father 的不同，导致这两个对象访问自己的 weight 非法：

```
son.height = 180;
grandson.height = 155;
```

扫一扫

视频讲解

5.3 子类对象的构造过程

当用子类的构造方法创建一个子类的对象时，子类的构造方法总是先调用父类的某个构造方法。也就是说，如果子类的构造方法没有明显地指明使用父类的哪个构造方法，子类就调用父类不带参数的构造方法。因此，当用子类创建对象时，不仅子类中声明的成员变量被分配了内存空间，而且父类的成员变量也都被分配了内存空间，但只将其中一部分（子类继承的那部分）作为分配给子类对象的变量。也就是说，父类中的 private 成员变量尽管分配了内存空间，也不作为子类对象的变量，即子类不继承父类的私有成员变量。同样，如果子类和父类不在同一个包中，尽管父类的友好成员变量分配了内存空间，也不作为子类的成员变量，即如果子类和父类不在同一个包中，子类不继承父类的友好成员变量。

子类对象的内存示意如图 5.2 所示，其中的“叉号”表示子类中声明定义的方法不可以操作这些内存单元，“对号”表示子类中声明定义的方法可以操作这些内存单元。

通过上面的讨论，我们有这样的感觉：子类创建对象时似乎浪费了一些内存，因为当用子类创建对象时，父类的成员变量也都分配了内存空间，但只将其中一部分作为分配给子类对象的变量。例如，父类中的 private 成员变量尽管分配了内存空间，但不作为子类对象的变量，当然它们也不是父类某个对象的变量，因为根本没有使用父类创建任何对象。这部分内存似乎成了垃圾一样，但实际情况并非如此，需注意到，子类中还有一部分方法是从父类继承的，这部分方法却可以操作这部分未继承的变量。

在例 5.2 中，子类对象调用继承的方法操作这些未被子类继承却分配了内存空间的变量。程序运行效果如图 5.3 所示。

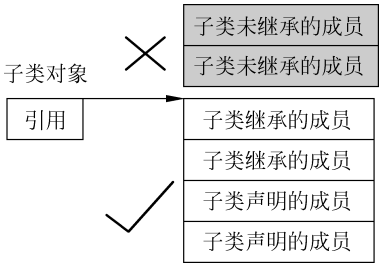

图 5.2 子类对象的内存示意图

```
C:\chapter5>java Example5_2
子类对象未继承的x的值是:888
子类对象的实例变量y的值是:12.678
```

图 5.3 子类对象调用方法

例 5.2

A.java

```
public class A {
    private int x;
    public void setX(int x) {
       this.x = x;
    }
    public int getX() {
```

```
        return x;
    }
}
```

B.java

```
public class B extends A {
    double y = 12;
    public void setY(int y)
    {   //this.y = y + x;            //非法,子类没有继承 x
    }
    public double getY() {
        return y;
    }
}
```

Example5_2.java

```
public class Example5_2 {
  public static void main(String args[]) {
      B b = new B();
      b.setX(888);
      System.out.println("子类对象未继承的 x 的值是:" + b.getX());
      b.y = 12.678;
      System.out.println("子类对象的实例变量 y 的值是:" + b.getY());
  }
}
```

扫一扫

视频讲解

5.4 成员变量的隐藏和方法重写

▶ 5.4.1 成员变量的隐藏

子类也可以隐藏继承的成员变量,对于子类,可以从父类继承成员变量,只要子类中声明的成员变量和父类中的成员变量同名,子类就隐藏了继承的成员变量,即子类创建的对象或子类中定义的实例方法所访问操作的成员变量一定是子类重新声明的这个成员变量(和父类中的成员变量同名)。需要注意的是,子类对象可以调用从父类继承的方法操作隐藏的成员变量。

在下面的例 5.3 中,父类 People 有一个名字为 x 的 double 型成员变量,本来子类 Student 可以继承这个成员变量,但是子类 Student 又重新声明了一个 int 型的名字为 x 的成员变量,这样就隐藏了继承的 double 型的名字为 x 的成员变量。但是,子类对象可以调用从父类继承的方法操作隐藏的 double 型成员变量。程序运行效果如图 5.4 所示。

```
C:\chapter5>java Example5_3
对象stu的x的值是:98
对象stu隐藏的x的值是:98.98
```

图 5.4 子类隐藏继承的成员变量

例 5.3

Example5_3.java

```
class People {
    public double x;
    public void setX(double x) {
        this.x = x;
    }
    public double getDoubleX() {
        return x;
    }
}
```

```
class Student extends People {
    int x;
    public int getX() {
       //x = 20.56;                //非法,因为子类的 x 已经是 int 型,不是 double 型
       return x;
    }
}
public class Example5_3 {
  public static void main(String args[]) {
     Student stu = new Student();
     stu.x = 98;                   //合法,子类对象的 x 是 int 型
     System.out.println("对象 stu 的 x 的值是:" + stu.getX());
     //stu.x = 98.98;              //非法,因为子类对象的 x 已经是 int 型
     stu.setX(98.98);              //子类对象调用继承的方法操作隐藏的 double 型变量 x
     double m = stu.getDoubleX();//子类对象调用继承的方法操作隐藏的 double 型变量 x
     System.out.println("对象 stu 隐藏的 x 的值是:" + m);
  }
}
```

▶ 5.4.2 方法重写

子类通过重写(Override)可以隐藏已继承的实例方法(方法重写也称为方法覆盖)。

❶ 重写的语法规则

如果子类可以继承父类的某个实例方法,那么子类就有权重写这个方法。方法重写是指在子类中定义一个方法,这个方法的类型和父类的方法的类型一致,或者是父类的方法的类型的子类型(所谓子类型,是指如果父类的方法的类型是“类”,那么允许子类的重写方法的类型是“子类”),并且这个方法的名字、参数个数、参数的类型和父类的方法完全相同。子类如此定义的方法称为子类重写的方法(不属于新增的方法)。

❷ 重写的目的

子类通过方法的重写可以隐藏继承的方法,可以把父类的状态和行为改变为自身的状态和行为。如果父类的方法 f(非 final 方法,见 5.6 节)可以被子类继承,子类就有权重写 f,一旦子类重写了父类的方法 f,就隐藏了继承的方法 f,那么子类对象调用方法 f 一定是调用的重写方法 f。重写方法既可以操作继承的成员变量,也可以操作子类新声明的成员变量。如果子类想使用被隐藏的方法,必须使用关键字 super,本书将在 5.5 节讲述 super 的用法。

在例 5.4 中,子类重写了父类的方法 f,运行效果如图 5.5 所示。

```
C:\chapter5>java Example5_4
调用重写方法得到的结果:30.0
调用继承方法得到的结果:8
```

图 5.5 方法重写

例 5.4

Example5_4.java

```
class A {
     double f(float x,float y) {
         return x + y;
     }
     public int g(int x,int y) {
         return x + y;
     }
}
class B extends A {
     double f(float x,float y) {
         return x * y;
     }
}
```

```
public class Example5_4 {
     public static void main(String args[]) {
       B b = new B();
       double result = b.f(5,6);          //b 调用重写的方法
       System.out.println("调用重写方法得到的结果:" + result);
       int m = b.g(3,5);                  //b 调用继承的方法
       System.out.println("调用继承方法得到的结果:" + m);
     }
}
```

在例 5.4 中,如果子类如下重写方法 f 将产生编译错误:

```
float f(float x,float y) {
        return x * y;
}
```

其原因是,父类的方法 f 的类型是 double,子类的重写方法 f 没有和父类的方法 f 保持类型一致,这样子类就无法隐藏继承的方法,导致子类出现两个方法的名字相同,并且参数也相同,这是不允许的(见 4.2.5 节)。

请读者思考,如果子类如下定义方法 f,是否属于重写方法呢? 编译可以通过吗? 运行结果怎样?

```
double f(float x,float y,double z) {
     return x * y;
}
```

❸ JDK 1.5 对重写的改进

子类在重写可以继承的方法时,可以完全按照自己的意图编写新的方法体,以便体现重写方法的独特行为(学习后面的 5.7 节后,读者会更深刻地理解重写方法在设计上的意义)。在 JDK 1.5 版本之后,允许重写方法的类型是父类方法的类型的子类型,即不必完全一致(JDK 1.5 版本之前要求必须一致)。也就是说,如果父类的方法的类型是“类”(类是面向对象语言中最重要的一种数据类型,类声明的变量称为对象,见 4.3 节),重写方法的类型可以是“子类”。

在例 5.5 中,有 3 个 Java 源文件,即 People.java、Chinese.java 和 Example5_5.java。其中,Chinese 类是 People 类的子类,在 Example5_5.java 中,People 类的 createPeople()方法的类型是 People 类,People 类的子类 Chinese 重写了父类的 createPeople()方法,重写方法的类型是 Chinese 类。程序运行效果如图 5.6 所示。

```
C:\chapter5>java  Example5_5
我是中国人
```

图 5.6 重写方法的类型是“类”

例 5.5

People.java

```
public class People {
    public void speak(){
       System.out.println("我是 People");
    }
}
```

Chinese.java

```
public class Chinese extends People {
   public void speak(){
       System.out.println("我是中国人");
   }
}
```

Example5_5.java

```
class CreatePeople {
    public People createPeople() {      //方法的类型是 People 类
        People p = new People();
        return p;
    }
}
class CreateChinese extends CreatePeople {
    public Chinese createPeople() {        //重写方法的类型是 People 类的子类 Chinese,即子类型
        Chinese chinese = new Chinese();
        return chinese;
    }
}
public class Example5_5 {
    public static void main(String args[]) {
      CreateChinese create = new CreateChinese();
      Chinese zhang = create.createPeople();  //create 调用重写的方法
      zhang.speak();
    }
}
```

❹ 重写的注意事项

重写父类的方法时,不可以降低方法的访问权限。在下面的代码中,子类重写父类的方法 f,该方法在父类中的访问权限是 protected 级别,子类重写时不允许级别低于 protected。例如:

```
class A {
     protected  float f(float x,float y) {
         return x - y;
     }
}
class B extends A {
     float f(float x,float y) {          //非法,因为降低了访问权限
          return x + y ;
     }
}
class C extends A {
     public float f(float x,float y) {  //合法,提高了访问权限
          return x * y ;
     }
}
```

扫一扫

视频讲解

5.5 super 关键字

子类可以隐藏从父类继承的成员变量和方法,如果在子类中想使用被子类隐藏的成员变量或方法,可以使用关键字 super。

▶ 5.5.1 使用 super 调用父类的构造方法

子类不继承父类的构造方法,因此,子类如果想使用父类的构造方法,必须在子类的构造方法中使用关键字 super 来表示,而且 super 必须是子类构造方法中的头一条语句。

在例 5.6 中,UniverStudent 类的构造方法使用 super 调用父类 Student 的构造方法,程序运行效果如图 5.7 所示。

```
C:\chapter5>java Example5_6
张三的学号是:20111
张三未婚
```

图 5.7 super 调用父类的构造方法

例 5.6

Student.java

```
public class Student {
     int number;
     String name;
     Student() {
     }
     Student(int number,String name) {
        this.number = number;
        this.name = name;
     }
     public int getNumber() {
        return number;
     }
     public String getName() {
        return name;
     }
}
```

UniverStudent.java

```
public class UniverStudent extends Student {
   boolean isMarriage;        //子类新增的结婚属性
   UniverStudent(int number,String name,boolean b) {
       super(number,name);    //调用父类的构造方法,即执行 Student(number,name)
   }
   public boolean getIsMarriage(){
       return isMarriage;
   }
}
```

Example5_6.java

```
public class Example5_6 {
     public static void main(String args[]) {
          UniverStudent zhang = new UniverStudent(20111,"张三",false);
          int number = zhang.getNumber();
          String name = zhang.getName();
          boolean marriage = zhang.getIsMarriage();
          System.out.println(name + "的学号是:" + number);
          if(marriage == true) {
            System.out.println(name + "已婚");
          }
          else{
            System.out.println(name + "未婚");
          }
     }
}
```

需要注意的是,如果在子类的构造方法中,没有明显地写出 super 关键字来调用父类的某个构造方法,那么默认有:

```
super();
```

语句,即调用父类的不带参数的构造方法。

如果类中定义了一个或多个构造方法,那么 Java 不提供默认的构造方法(不带参数的构造方法)。因此,当在父类中定义多个构造方法时,应当包括一个不带参数的构造方法(如例 5.6 中的 Student 类),以防子类省略 super 时出现错误。请读者思考,如果在例 5.6 的 UniverStudent 类的构造方法中省略 super,程序的运行效果是怎样的?

5.5.2 使用 super 操作被隐藏的成员变量和方法

如果在子类中想使用被子类隐藏的成员变量或方法，可以使用关键字 super，super. x、super. play()就是访问和调用被子类隐藏的成员变量 x 和方法 play()。

需要注意的是，当子类创建一个对象时，除了子类声明的成员变量和继承的成员变量要分配内存外(这些内存单元是属于子类对象的)，被隐藏的成员变量也要分配内存，但该内存单元不属于任何对象，这些内存单元必须用 super 调用。同样，当子类创建一个对象时，除了子类声明的方法和继承的方法要分配入口地址外(这些方法可供子类对象调用)，被隐藏的方法也要分配入口地址，但该入口地址只对 super 可见，所以必须由 super 来调用。当 super 调用隐藏的方法时，该方法中出现的成员变量是指被隐藏的成员变量，如图 5.8 所示。

在例 5.7 中，子类 Average 使用 super 调用隐藏的方法，程序运行效果如图 5.9 所示。

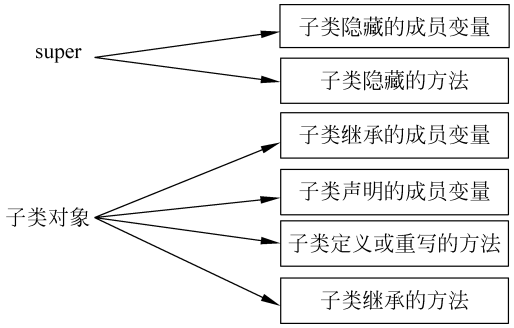

图 5.8 super 与隐藏

```
C:\chapter5>java Example5_7
result1=5150.5878
result2=4949.4322
```

图 5.9 super 调用隐藏的方法

例 5.7

Sum. java

```
public class Sum {
   int n;
   public double f() {
       double sum = 0;
       for(int i = 1;i <= n;i++ ){
              sum = sum + i;
       }
       return sum;
   }
}
```

Average. java

```
public class Average extends Sum {
   double n;                  //子类继承的 int 型变量 n 被隐藏
   public double f() {
       double c;
       super.n = (int)n;      //double 型变量 n 做 int 转换，将结果赋给隐藏的 int 型变量 n
       c = super.f();
       return c + n;
   }
   public double g() {
       double c;
       c = super.f();
       return c - n;
   }
}
```

Example5_7.java

```
public class Example5_7 {
    public static void main(String args[]) {
       Average aver = new Average();
       aver.n = 100.5678;
       double result1 = aver.f();
       double result2 = aver.g();
       System.out.println("result1 = " + result1);
       System.out.println("result2 = " + result2);
    }
}
```

注意,如果将 Example5_7 类中的代码:

```
double result1 = aver.f();
double result2 = aver.g();
```

改写成(颠倒次序):

```
double result2 = aver.g();
double result1 = aver.f();
```

那么运行结果是:

```
result1 = 5150.5678
result2 = - 100.5678
```

这是因为执行"aver.g();"的过程中需要执行"super.f();",super.f()中出现的 n 是隐藏的 n,且 n 还没有赋值(默认值是 0)。

扫一扫

视频讲解

5.6 final 关键字

final 关键字可以修饰类、成员变量和方法中的局部变量。

▶ 5.6.1 final 类

可以使用 final 将类声明为 final 类。final 类不能被继承,即不能有子类。例如:

```
final class A {
⋮
}
```

A 就是一个 final 类,将不允许任何类声明成 A 的子类。有时出于安全性的考虑,将一些类修饰为 final 类。例如,Java 提供的 String 类对于编译器和解释器的正常运行有很重要的作用,不能轻易地改变,它被修饰为 final 类。

▶ 5.6.2 final 方法

如果用 final 修饰父类中的一个方法,那么这个方法不允许子类重写。也就是说,不允许子类隐藏可以继承的 final 方法(不许做任何篡改)。

▶ 5.6.3 常量

如果成员变量或局部变量被修饰为 final,就是常量。常量在声明时没有默认值,所以在声明常量时必须指定该常量的值,而且不能发生变化。

下面的例 5.8 使用了 final 关键字，程序运行效果如图 5.10 所示。

```
C:\chapter5>java Example5_8
面积：31415.926000000003
您好，How's everything here ?
```

图 5.10 使用 final 关键字

例 5.8

Example5_8.java

```
class A {
   //final double PI;                    //非法，因为没有给常量指定值
   final double PI = 3.1415926;          //PI 是常量
   public double getArea(final double r) {
       //r = 89;                         //非法，因为不允许再改变 r 的值
       return PI * r * r;
   }
   public final void speak() {
       System.out.println("您好，How's everything here ?");
   }
}
class B extends A {
   /* 非法，不能重写 speak 方法
   public void speak() {
       System.out.println("您好");
   }
    */
}
public class Example5_8 {
   public static void main(String args[]) {
       B b = new B();
       System.out.println("面积：" + b.getArea(100));
       b.speak();                          //调用继承的方法
    }
}
```

扫一扫

视频讲解

5.7 对象的上转型对象

我们经常说“老虎是哺乳动物”“狗是哺乳动物”等，若哺乳类是老虎类的父类，这样说当然正确，但是，如果说老虎是哺乳动物，老虎将失掉老虎独有的属性和功能。下面介绍对象的上转型对象。

假设 A 类是 B 类的父类，当用子类创建一个对象，并把这个对象的引用放到父类的对象中。例如：

```
A a;
a = new B();
```

或

```
A a;
B b = new B();
a = b;
```

这时，称对象 a 是对象 b 的上转型对象（例如说“老虎是哺乳动物”）。

对象的上转型对象的实体是子类负责创建的，但上转型对象会失去原对象的一些属性和功能（上转型对象相当于子类对象的一个“简化”对象）。上转型对象具有以下特点（如图 5.11 所示）。

（1）上转型对象不能操作子类新增的成员变量（失掉了这部分属性），不能调用子类新增的方法（失掉了一些功能）。

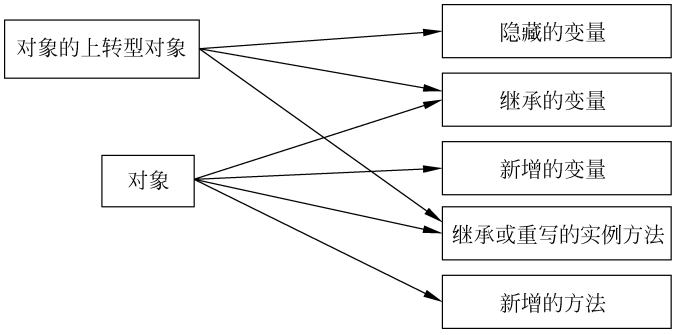

图 5.11　上转型对象示意图

(2) 上转型对象可以访问子类继承或隐藏的成员变量,也可以调用子类继承的方法或子类重写的实例方法。上转型对象操作子类继承的方法或子类重写的实例方法,其作用等价于用子类对象去调用这些方法。因此,如果子类重写了父类的某个实例方法,当对象的上转型对象调用这个实例方法时一定是调用了子类重写的实例方法。

注:① 不要将父类创建的对象和子类对象的上转型对象混淆。

② 可以将对象的上转型对象强制转换到一个子类对象,这时,该子类对象又具备了子类的所有属性和功能。

③ 不可以将父类创建的对象的引用赋给子类声明的对象(不能说"哺乳动物是老虎")。

④ 如果子类重写了父类的静态方法(static 方法),那么子类对象的上转型对象不能调用子类重写的静态方法,只能调用父类的静态方法。

在例 5.9 中,Anthropoid(类人猿)类声明的对象 monkey 是 People 类创建的对象 people 的上转型对象,程序运行效果如图 5.12 所示。

```
C:\chapter5>java Example5_9
A**I love this game**A
12.58
A
55加33等于88
T
```

图 5.12　使用上转型对象

例 5.9

Anthropoid.java

```
public class Anthropoid {
   double m = 12.58;
   void crySpeak(String s) {
       System.out.println(s);
   }
}
```

People.java

```
public class People extends Anthropoid {
   char m = 'A';
   int n = 60;
   void computer(int a,int b) {
       int c = a + b;
       System.out.println(a + "加" + b + "等于" + c);
   }
   void crySpeak(String s) {
       System.out.println(m + " ** " + s + " ** " + m);
   }
}
```

Example5_9.java

```
public class Example5_9 {
    public static void main(String args[]) {
       People  people = new People();
       Anthropoid monkey = people;                  //monkey 是 people 对象的上转型对象
       monkey.crySpeak("I love this game");         //等同于 people 调用重写的 crySpeak 方法
       //monkey.n = 100;                            //非法,因为 n 是子类新增的成员变量
       //monkey.computer(12,19);                    //非法,因为 computer()是子类新增的方法
       System.out.println(monkey.m) ;               //操作隐藏的 m,不等同于 people.m
       System.out.println(people.m) ;               //操作子类的 m
       People zhang = (People)monkey;               //把上转型对象强制转化为子类的对象
       zhang.computer(55,33);                       //zhang 是子类的对象
       zhang.m = 'T';                               //操作子类声明的成员的变量 m
       System.out.println(zhang.m) ;
    }
}
```

扫一扫

视频讲解

5.8 继承与多态

我们经常说“哺乳动物有很多种叫声”,如“吼”“嚎”“汪汪”“喵喵”等,这就是叫声的多态性。

当一个类有很多子类,并且这些子类都重写了父类中的某个方法时,把子类创建的对象的引用放到一个父类的对象中,就得到了该对象的一个上转型对象。这个上转型对象在调用这个方法时可能具有多种形态,因为不同的子类在重写父类的方法时可能产生不同的行为。例如,狗类的上转型对象调用“叫声”方法时产生的行为是“汪汪”,而猫类的上转型对象调用“叫声”方法时产生的行为是“喵喵”等。

多态性是指父类的某个方法被其子类重写时,可以各自产生自己的功能行为。

```
C:\chapter5>java Example5_10
这是狗的叫声：汪汪……汪汪
这是猫的叫声：喵喵……喵喵...
```

图 5.13 多态

例 5.10 展示了多态性,程序运行效果如图 5.13 所示。

例 5.10

Example5_10.java

```
class 动物 {
    void cry() {
    }
}
class 狗 extends 动物 {
    void cry() {
       System.out.println("这是狗的叫声：汪汪……汪汪");
    }
}
class 猫 extends 动物 {
    void cry() {
       System.out.println("这是猫的叫声：喵喵……喵喵……");
    }
}
public class Example5_10 {
    public static void main(String args[]) {
       动物 animal = new 狗();                  //animal 是狗的上转型对象
       animal.cry();
       animal = new 猫();                       //animal 是猫的上转型对象
```

```
            animal.cry();
        }
    }
```

扫一扫

视频讲解

5.9 abstract 类和方法

用关键字 abstract 修饰的类称为 abstract 类(抽象类)。例如:

```
abstract class A {
   ⋮
}
```

用关键字 abstract 修饰的方法称为 abstract 方法(抽象方法),对于 abstract 方法,只允许声明,不允许实现,而且不允许使用 final 和 abstract 同时修饰一个方法。例如:

```
abstract int min(int x,int y);
```

5.9.1 abstract 类的特点与理解

❶ abstract 类的特点

abstract 类具有以下特点:

(1) abstract 类中可以有 abstract 方法。

和普通类(非 abstract 类)相比,abstract 类中可以有 abstract 方法(非 abstract 类中不可以有 abstract 方法),也可以有非 abstract 方法。

下面的 A 类中的 min()方法是 abstract 方法,max()方法是普通方法(非 abstract 方法)。

```
abstract class A {
       abstract int min(int x,int y);
       int max(int x,int y) {
           return x>y?x:y;
       }
}
```

注:abstract 类中也可以没有 abstract 方法。

(2) abstract 类不能用 new 运算符创建对象。

对于 abstract 类,不能使用 new 运算符创建该类的对象。如果一个非抽象类是某个抽象类的子类,那么它必须重写父类的抽象方法,给出方法体,这就是不允许使用 final 和 abstract 同时修饰一个方法或类的原因。

(3) abstract 类的子类。

如果一个非 abstract 类是 abstract 类的子类,它必须重写父类的 abstract 方法,即去掉 abstract 方法的 abstract 修饰,并给出方法体。如果一个 abstract 类是 abstract 类的子类,它可以重写父类的 abstract 方法,也可以继承父类的 abstract 方法。

(4) abstract 类的对象作上转型对象。

可以使用 abstract 类声明对象,尽管不能使用 new 运算符创建该对象,但该对象可以成为其子类对象的上转型对象,那么该对象就可以调用子类重写的方法。

❷ 理解 abstract 类

抽象类的语法很容易被理解和掌握,但更重要的是理解抽象类的意义,这一点是更为重要

的。理解的关键点是：

(1) 抽象类可以抽象出重要的行为标准，该行为标准用抽象方法来表示，即抽象类封装了子类必须有的行为标准。

(2) 抽象类声明的对象可以成为其子类的对象的上转型对象，调用子类重写的方法，即体现子类根据抽象类里的行为标准给出的具体行为。

人们已经习惯给别人介绍数量标准。例如，在介绍人的时候，可以说，人的身高可以是float型的，头发的个数可以是int型的。介绍人的头发，强调数量类型是int型，但不介绍有多少根头发。学习了类以后，也要习惯介绍行为标准。行为标准只需要方法的名字、方法的类型。例如，介绍人的行为时，强调人具有run行为，或speak行为(仅仅说出行为标准)，但不介绍speak是用英语或中文说话。只强调行为标准，即方法的名字、方法的类型，这就是抽象方法(没有方法体的方法)。开发者可以把主要精力放在一个应用中需要的那些行为标准(不用关心行为的细节)，不仅节省时间，而且非常有利于设计出易维护、易扩展的程序(见后面的5.9.2节以及7.2节)。抽象类的重要特点是，特别关心方法名字、类型以及参数，但不关心这些操作具体实现的细节，即不关心方法体。在设计程序时，可以给出若干个抽象类表明程序的重要特征，也就是说，可以通过在一个抽象类中声明若干个抽象方法，表明这些方法的重要性。抽象类可以让程序设计者忽略具体的细节，以便更好地设计程序，例如，在设计地图时，首先考虑地图最重要的轮廓，不必去考虑诸如城市中的街道名、门牌号等细节。细节应当由抽象类的非抽象子类去实现，这些子类可以给出具体的实例，来完成程序功能的具体实现。

一个男孩要找女朋友，他可以提出一些行为标准。例如，女朋友具有speak和cooking行为，但不给出speak和cooking行为的细节。例5.11使用了abstract类封装了男孩对女朋友的行为要求，即封装了他要找的任何具体女朋友都应该具有的行为。

例 5.11

Example5_11.java

```
abstract class GirlFriend { //抽象类,封装了两个行为标准
    abstract void speak();
    abstract void cooking();
}
class ChinaGirlFriend extends GirlFriend {
    void speak(){
       System.out.println("你好");
    }
    void cooking(){
       System.out.println("水煮鱼");
    }
}
class AmericanGirlFriend extends GirlFriend {
    void speak(){
       System.out.println("hello");
    }
    void cooking(){
       System.out.println("roast beef");
    }
}
class Boy {
    GirlFriend friend;
    void setGirlfriend(GirlFriend f){
        friend = f;
    }
```

```
    void showGirlFriend() {
        friend.speak();
        friend.cooking();
    }
}
public class Example5_11 {
    public static void main(String args[]) {
        GirlFriend girl = new ChinaGirlFriend();      //girl 是上转型对象
        Boy boy = new Boy();
        boy.setGirlfriend(girl);
        boy.showGirlFriend();
        girl = new AmericanGirlFriend();              //girl 是上转型对象
        boy.setGirlfriend(girl);
        boy.showGirlFriend();
    }
}
```

例 5.12 中有一个 abstract 类：机动车，该类中有 3 个 abstract 方法：启动、加速和刹车，即机动车类将启动、加速和刹车功能视为一些重要的功能。机动车类的非抽象子类必须给出启动、加速和刹车的细节。程序运行效果如图 5.14 所示。

```
C:\chapter5>java Example5_12
手动档轿车的操作：
踏下离合器，换到一挡
然后慢慢抬起离合器
踩油门
踏下离合器，踏下刹车板
然后将挡位换到一挡
自动档轿车轿车的操作：
使用前进挡
然后轻踩油门
踩油门
踏下刹车板
```

图 5.14　子类实现细节

例 5.12

Example5_12.java

```
abstract class 机动车 {
    abstract void 启动();
    abstract void 加速();
    abstract void 刹车();
}
class 手动挡轿车 extends 机动车 {
    void 启动() {
        System.out.println("踏下离合器,换到一挡");
        System.out.println("然后慢慢抬起离合器");
    }
    void 加速() {
        System.out.println("踩油门");
    }
    void 刹车() {
        System.out.println("踏下离合器,踏下刹车板");
        System.out.println("然后将挡位换到一挡");
    }
}
class 自动挡轿车 extends 机动车 {
    void 启动() {
        System.out.println("使用前进挡");
        System.out.println("然后轻踩油门");
    }
    void 加速() {
        System.out.println("踩油门");
    }
    void 刹车() {
        System.out.println("踏下刹车板");
    }
}
public class Example5_12 {
    public static void main(String args[]) {
```

```
            机动车 car = new 手动挡轿车();
            System.out.println("手动挡轿车的操作:");
            car.启动();
            car.加速();
            car.刹车();
            car = new 自动挡轿车();
            System.out.println("自动挡轿车的操作:");
            car.启动();
            car.加速();
            car.刹车();
        }
    }
```

▶ 5.9.2 abstract 类与多态

在设计程序时,经常会使用 abstract 类,其原因是,abstract 类只关心操作,不关心这些操作具体实现的细节,可以使程序的设计者把主要精力放在程序的设计上,而不必拘泥于细节的实现(将这些细节留给子类的设计者),即避免设计者把大量的时间和精力花费在具体的算法上。在设计一个程序时,可以通过在 abstract 类中声明若干 abstract 方法,表明这些方法在整个系统设计中的重要性,方法体的内容细节由它的非 abstract 子类去完成。

使用多态进行程序设计的核心技术之一是使用上转型对象,即将 Abstract 类声明对象作为其子类的上转型对象,那么这个上转型对象就可以调用子类重写的方法。

利用多态设计程序的好处是,可以体现程序设计的所谓“开-闭”原则(Open-Closed Principle),关于这一点,在第 7 章和第 8 章还会详细介绍。简单地说,“开-闭”原则强调一个程序应当对扩展开放,对修改关闭,增强代码的可维护性。例如,程序的主要设计者可以设计出图 5.15 所示的一种结构关系,从该图可以看出,当程序再增加一个子类时(由其他人员去编写子类),上转型对象所在的类不需要做任何修改就可以调用该子类重写的方法。

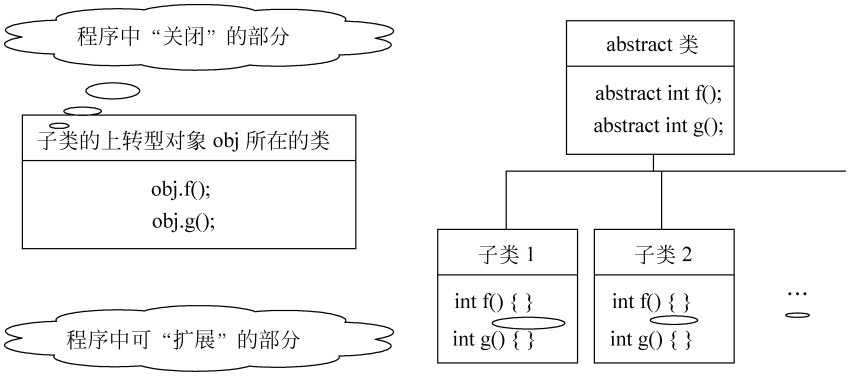

图 5.15 abstract 类与多态的使用

当然,在程序设计好后,首先应当对 abstract 类的修改进行“关闭”,否则,一旦修改 abstract 类(例如,为它再增加一个 abstract 方法),该 abstract 类的所有子类都需要做出修改。在程序设计好后,应当对增加 abstract 的子类“开放”,即在程序中增加 abstract 的子类时,不需要修改其他重要的类。

在例 5.13 中,准备设计一个动物声音“模拟器”,希望所设计的模拟器可以模拟许多动物的叫声。

首先设计一个抽象类 Animal,该抽象类有 cry()和 getAnimalName()两个抽象方法,那

么 Animal 的子类必须实现 cry 和 getAnimalName()方法,即要求各种具体的动物给出自己的叫声和种类名称。

然后设计 Simulator 类(模拟器),该类有一个 playSound(Animal animal)方法,该方法的参数是 Animal 类型。显然,参数 animal 可以是抽象类 Animal 的任何一个子类对象的上转型对象,即参数 animal 可以调用 Animal 的子类重写的 cry()方法播放具体动物的声音,调用 Animal 的子类重写的 getAnimalName()方法显示动物种类的名称。

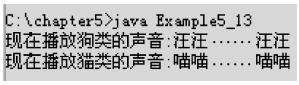

图 5.16 体现“开-闭”原则

程序运行效果如图 5.16 所示。

例 5.13

Animal.java

```
public abstract class Animal {
    public abstract void cry();
    public abstract String getAnimalName();
}
```

Simulator.java

```
public class Simulator {
   public void playSound(Animal animal) {
       System.out.print("现在播放" + animal.getAnimalName() + "类的声音:");
       animal.cry();
   }
}
```

Dog.java

```
public class Dog extends Animal {
   public void cry() {
       System.out.println("汪汪……汪汪");
   }
   public String getAnimalName() {
       return "狗";
   }
}
```

Cat.java

```
public class Cat extends Animal {
   public void cry() {
       System.out.println("喵喵……喵喵");
   }
   public String getAnimalName() {
       return "猫";
   }
}
```

Example5_13.java

```
public class Example5_13 {
    public static void main(String args[]) {
       Simulator simulator = new Simulator();
       simulator.playSound(new Dog());
       simulator.playSound(new Cat());
    }
}
```

注：(1) 在例 5.13 中，如果再增加一个 Java 源文件(对扩展开放)，该源文件有一个 Animal 的子类 Tiger(负责模拟老虎的声音)，那么模拟器 Simulator 类不需要做任何修改(对 Simulator 类的修改关闭)，应用程序的主类就可以使用代码：

```
simulator.playSound(new Tiger());
```

模拟老虎的声音。

(2) 如果将例 5.13 中的 Animal 类、Simulator 类及 Dog 类和 Cat 类看作一个小的开发框架，将 Example5_13 看作是用户程序，那么框架满足“开-闭”原则，该框架相对用户的需求就比较容易维护，因为当用户程序需要模拟老虎的声音时，只需简单地扩展框架，即在框架中增加一个 Animal 的 Tiger 子类，而无须修改框架中的其他类。

5.10 接口

Java 不支持多继承性，即一个类不能有多个父类(至多一个父类)。单继承性使得 Java 简单，易于管理程序。为了克服单继承的缺点，Java 使用了接口，一个类可以实现多个接口。

使用关键字 interface 来定义一个接口。接口的定义和类的定义很相似，分为接口的声明和接口体。

扫一扫

视频讲解

▶ 5.10.1 接口的定义与使用

接口定义中含有一个接口声明(给出接口的名字)和接口体。

❶ 接口声明

接口通过使用关键字 interface 来声明，格式如下：

```
interface 接口的名字 //接口声明
{
      //接口体
}
```

❷ 接口体

接口体中包含 static 常量和方法定义两部分。

1) 接口体中的抽象方法和常量

JDK 8 版本之前，接口体中只有抽象方法，所有的抽象方法的访问权限一定都是 public(允许省略 public、abstract 修饰符)。接口体中所有的 static 常量的访问权限一定都是 public(允许省略 public、final 和 static 修饰符，接口中不会有变量)。例如：

```
interface Printable {
     public static final int MAX = 100;              //等价写法:int MAX = 100;
     public abstract void add();                     //等价写法:void add();
     public abstract float sum(float x ,float y);    //等价写法:float sum(float x ,float y);
}
```

2) 接口体中的 default 实例方法

从 JDK 8 版本开始，允许使用 default 关键字，在接口体中定义称作 default 的实例方法(不可以定义 default 的 static 方法)，default 的实例方法和通常的普通的方法比就是用关键字 default 修饰的带方法体的实例方法。default 实例方法的访问权限一定是 public(允许省略 public 修饰符)。例如，下列接口中的 max 方法就是 default 实例方法：

```
interface Printable {
      public final int MAX = 100;              //等价写法:int MAX = 100;
      public abstract void add();              //等价写法:void add();
      public abstract float sum(float x ,float y);
      public default int max(int a,int b) {    //default 方法
          return a > b?a:b;
      }
}
```

注意不可以省略 default 关键字,因为接口中不允许定义通常的带方法体的实例方法。

3) 接口体中的 static 方法

从 JDK 8 版本开始,允许在接口体中定义 static 方法。例如,下列接口中的 f 方法就是 static 方法:

```
public interface Printable {
      public static final int MAX = 100;          //等价写法:int MAX = 100;
      public abstract void on();                  //等价写法:void on();
      public abstract float sum(float x ,float y);
      public default int max(int a,int b) {       //default 方法
          return a > b?a:b;
      }
      public static void f() {
         System.out.println("注意是从 Java SE 8 开始的");
      }
}
```

注:不可以用 static 和 abstract 同时修饰一个方法。

❸ 接口的使用

1) 使用接口中的常量和 static 方法

可以用接口名访问接口的常量、调用接口中的 static 方法。例如:

```
Printable.MAX;
Printable.f();
```

2) 类实现接口

一个类通过使用关键字 implements 声明自己实现一个或多个接口。如果实现多个接口,用逗号隔开接口名,例如,A 类实现 Printable 和 Addable 接口:

```
class A implements Printable,Addable {
}
```

再如,Animal 的子类 Dog 类实现 Eatable 和 Sleepable 接口:

```
class Dog extends Animal implements Eatable,Sleepable {
}
```

如果一个类实现了某个接口,那么这个类就自然拥有了接口中的常量和 default 方法(去掉了 default 关键字),该类也可以重写接口中的 default 方法(注意,重写时需要去掉 default 关键字)。如果一个非 abstract 类实现了某个接口,那么这个类必须重写该接口的所有 abstract 方法,即去掉 abstract 修饰给出方法体(有关重写的要求见 5.4.2 节)。如果一个 abstract 类实现了某个接口,该类可以选择重写接口的 abstract 方法或直接拥有接口的 abstract 方法。

特别注意的是:

(1) 类实现某接口,但类不拥有接口的 static 方法。

(2) 接口中方法的访问权限都是 public,重写时不可省略 public(否则就降低了访问权限,这是不允许的)。

实现接口的非 abstract 类一定要重写接口的 abstract 方法,因此也称这个类实现了接口中的方法。

Java 核心 API 为我们提供的接口都在相应的包中,通过 import 语句不仅可以引入包中的类,也可以引入包中的接口。例如:

```
import java.io.*;
```

不仅引入了 java.io 包中的类,同时也引入了该包中的接口。

我们还可以自己定义接口,一个 Java 源文件就是由类和接口组成的。

例 5.14 使用了接口,程序运行效果如图 5.17 所示。

```
接口中的常量100
调用on方法(重写的):
打开电视
调用sum方法(重写的):30.0
调用接口提供的default方法78
注意是从Java SE 8开始的
```

图 5.17 接口的使用

例 5.14

Printable.java

```
public interface Printable {
    public static final int MAX = 100;                    //等价写法:int MAX = 100;
    public abstract void on();                            //等价写法:void on();
    public abstract float sum(float x,float y);
    default int max(int a,int b) {                        //default 方法
        return a>b?a:b;
    }
    public static void f() {
       System.out.println("注意是从 Java SE 8 开始的");
    }
}
```

AAA.java

```
public class AAA implements Printable {                   //实现 Printable 接口
     public void on(){                                    //必须重写接口的 abstract 方法 on
         System.out.println("打开电视");
     }
     public float sum(float x,float y){                   //必须重写接口的 abstract 方法 sum
         return x+y;
     }
}
```

Example5_14.java

```
public class Example5_14 {
    public static void main(String args[]) {
       AAA a = new AAA();
       System.out.println("接口中的常量"+AAA.MAX);
       System.out.println("调用 on 方法(重写的):");
       a.on();
       System.out.println("调用 sum 方法(重写的):"+a.sum(12,18));
       System.out.println("调用接口提供的 default 方法"+a.max(12,78));
       Printable.f();
    }
}
```

接口声明时,如果在关键字 interface 前面加上 public 关键字,就称这样的接口是一个

public 接口。public 接口可以被任何一个类声明实现。如果一个接口不加 public 修饰,就是友好接口类,友好接口可以被与该接口在同一包中的类声明实现。

如果父类实现了某个接口,那么子类也就自然实现了该接口,子类不必再显式地使用关键字 implements 声明实现这个接口。

接口也可以被继承,即可以通过关键字 extends 声明一个接口是另一个接口的子接口。由于接口中的方法和常量都是 public 的,所以子接口将继承父接口中的全部实例方法和常量。

扫一扫

视频讲解

▶ 5.10.2 接口回调

接口回调是指可以把实现某一接口的类创建的对象的引用赋给该接口声明的接口变量中,那么该接口变量就可以调用被类重写的接口方法以及接口中的 default 方法。实际上,当接口变量调用被类重写的接口方法或接口中的 default 方法时,就是通知相应的对象调用这个方法。

接口回调非常类似 5.7 节介绍的上转型对象调用子类的重写方法。

例 5.15 使用了接口的回调技术。

例 5.15

Example5_15.java

```
interface ShowMessage {
    void 显示商标(String s);
    default void f(){
        System.out.println("default 方法");
    }
}
class TV implements ShowMessage {
    public void 显示商标(String s) {
       System.out.println(s);
    }
    public void f(){
        System.out.println("重写了 default 方法");
    }
}
class PC implements ShowMessage {
    public void 显示商标(String s) {
      System.out.println(s);
    }
}
public class Example5_15 {
    public static void main(String args[]) {
       ShowMessage sm = null;                    //声明接口变量
       sm = new TV();                            //接口变量中存放对象的引用
       sm.显示商标("长城牌电视机");               //接口回调
       sm.f();                                   //接口回调
       sm = new PC();                            //接口变量中存放对象的引用
       sm.显示商标("联想奔月 5008PC");            //接口回调
       sm.f();                                   //接口回调
    }
}
```

扫一扫

视频讲解

▶ 5.10.3 理解接口

接口的语法规则很容易记住,但真正理解接口更重要。理解的关键点是:

(1) 接口可以抽象出重要的行为标准,该行为标准用抽象方法来表示。

(2) 可以把实现接口的类的对象的引用赋值给接口变量,该接口变量可以调用被该类实现的接口方法,即体现该类根据接口中的行为标准给出的具体行为。

假如轿车、卡车、拖拉机、摩托车和客车都是机动车的子类,其中机动车是一个抽象类。机动车中有诸如“刹车”“转向”等方法是合理的,即要求轿车、卡车、拖拉机、摩托车、客车都必须具体实现“刹车”“转向”等功能。但是,如果机动车类包含两个抽象方法——“收取费用”和“调节温度”,那么所有的子类都要重写这两个方法,即给出方法体,产生各自的收费或控制温度的行为。这显然不符合人们的思维逻辑,因为拖拉机可能不需要有“收取费用”或“调节温度”的功能,而其他的一些类,例如飞机、轮船等可能也需要具体实现“收取费用”“调节温度”。

接口的思想在于它可以要求某些类有相同名称的方法,但方法的具体内容(方法体的内容)可以不同,即要求这些类实现接口,以保证这些类一定有接口中所声明的方法(即所谓的方法绑定)。接口在要求一些类有相同名称的方法的同时,并不强迫这些类具有相同的父类。例如,各式各样的电器产品,它们可能归属不同的种类,但国家标准要求电器产品都必须提供一个名称为 on 的功能(为达到此目的,只需要求它们实现同一接口,该接口中有名字为 on 的方法),但名称为 on 的功能的具体行为由各个电器产品去实现。

在例 5.16 中,要求 MotorVehicles 类(机动车类)的子类 Taxi(出租车)和 Bus(公共汽车)必须有名称为 brake 的方法(有刹车功能),但额外要求 Taxi 类有名字为 controlAirTemperature 和 charge 的方法(有空调和收费功能),即要求 Taxi 实现两个接口,要求客车类有名字为 charge 的方法(有收费功能),即要求 Bus 只实现一个接口。程序运行效果如图 5.18 所示。

```
公共汽车使用鼓式刹车技术
公共汽车:一元/张,不计算千米数
出租车使用盘式刹车技术
出租车:2元/km,起价3km
出租车安装了Hair空调
电影院:门票,十元/张
电影院安装了中央空调
```

图 5.18 理解接口

例 5.16

Example5_16.java

```
abstract class MotorVehicles {
    abstract void brake();
}
interface MoneyFare {
    void charge();
}
interface ControlTemperature {
    void controlAirTemperature();
}
class Bus extends MotorVehicles implements MoneyFare {
     void brake() {
         System.out.println("公共汽车使用鼓式刹车技术");
     }
     public void charge() {
         System.out.println("公共汽车:一元/张,不计算千米数");
     }
}
class Taxi extends MotorVehicles implements MoneyFare,ControlTemperature {
     void brake() {
         System.out.println("出租车使用盘式刹车技术");
     }
     public void charge() {
         System.out.println("出租车:2 元/km,起价 3km");
     }
     public void controlAirTemperature() {
         System.out.println("出租车安装了 Hair 空调");
```

```
        }
    }
    class Cinema implements MoneyFare,ControlTemperature {
         public void charge() {
              System.out.println("电影院:门票,十元/张");
         }
         public void controlAirTemperature() {
            System.out.println("电影院安装了中央空调");
         }
    }
    public class Example5_16 {
        public static void main(String args[]) {
            Bus bus101 = new Bus();
            Taxi buleTaxi = new Taxi();
            Cinema redStarCinema = new Cinema();
            MoneyFare fare;
            ControlTemperature temperature;
            fare = bus101;
            bus101.brake();
            fare.charge();
            fare = buleTaxi;
            temperature = buleTaxi;
            buleTaxi.brake();
            fare.charge();
            temperature.controlAirTemperature();
            fare = redStarCinema;
            temperature = redStarCinema;
            fare.charge();
            temperature.controlAirTemperature();
        }
    }
```

扫一扫

视频讲解

▶ 5.10.4 接口与多态

5.10.3 节学习了接口回调,即当把实现接口的类的实例的引用赋值给接口变量后,该接口变量就可以回调类重写的接口方法。由接口产生的多态就是指不同的类在实现同一个接口时可能具有不同的实现方式,那么接口变量在回调接口方法时就可能具有多种形态。

在设计程序时,经常会使用接口,其原因是,接口只关心操作,但不关心这些操作的具体实现细节,可以使我们把主要精力放在程序的设计上,而不必拘泥于细节的实现。也就是说,可以通过在接口中声明若干 abstract 方法,表明这些方法的重要性,方法体的内容细节由实现接口的类去完成。使用接口进行程序设计的核心思想是使用接口回调,即接口变量存放实现该接口的类的对象的引用,从而接口变量就可以回调类实现的接口方法。

利用接口也可以体现程序设计的“开-闭”原则,即对扩展开放,对修改关闭。例如,程序的主要设计者可以设计出如图 5.18 所示的一种结构关系,从该图可以看出,当程序再增加实现接口的类(有其他设计者去实现)时,接口变量所在的类不需要做任何修改,就可以回调类重写的接口方法。

当然,在程序设计好后,首先应当对接口的修改“关闭”,否则,一旦修改接口,例如,为它再增加一个 abstract 方法,那么实现该接口的类都需要做出修改。但是,程序设计好后,应当对增加实现接口的类“开放”,即在程序中再增加实现接口的类时,不需要修改其他重要的类。

下面的例 5.17 中,我们准备设计一个广告牌,希望所设计的广告牌可以展示许多公司的广告词。

首先设计一个接口 Advertisement，该接口有两个方法：showAdvertisement()和getCorpName()，那么实现 Advertisement 接口的类必须重写 showAdvertisement()和getCorpName()方法，即要求各个公司给出具体的广告词和公司的名称。

然后设计 AdvertisementBoard 类(广告牌)，该类有一个 Advertisement 接口类型的成员变量 adver(就像人们常说的，广告牌对外留有接口)。adver 可以存放任何实现 Advertisement 接口的类的对象的引用，并回调类重写的接口方法 showAdvertisement()来显示公司的广告词、回调类重写的接口方法 getCorpName()来显示公司的名称。

例 5.17 的详细代码如下，运行效果如图 5.19 所示。

```
C:\chapter5>java Example5_17
广告牌显示飞利浦公司的广告词：
@@@@@@@@@@@@@@@@@@@@@@@@
没有最好，只有更好
@@@@@@@@@@@@@@@@@@@@@@@@
广告牌显示联想集团公司的广告词：
**************
让世界变得很小
**************
```

图 5.19 接口体现“开-闭”原则

例 5.17

Advertisement.java

```
public interface Advertisement {                    //接口
       public void showAdvertisement();
       public String getCorpName();
}
```

AdvertisementBoard.java

```
public class AdvertisementBoard {
   Advertisement adver;
   public void setAdvertisement(Advertisement adver){
       this.adver = adver;
   }
   public void show() {
       if(adver != null){
          System.out.println("广告牌显示" + adver.getCorpName() + "公司的广告词:");
          adver.showAdvertisement();
       }
       else {
           System.out.println("广告牌无广告");
       }
   }
}
```

PhilipsCorp.java

```
public class PhilipsCorp implements Advertisement {      //PhilipsCorp 实现 Advertisement 接口
   public void showAdvertisement(){
      System.out.println("@@@@@@@@@@@@@@@@@@@@@@@@");
      System.out.println("没有最好,只有更好");
      System.out.println("@@@@@@@@@@@@@@@@@@@@@@@@");
   }
   public String getCorpName() {
      return "飞利浦" ;
   }
}
```

LenovoCorp.java

```
public class LenovoCorp implements Advertisement {      //LenovoCorp 实现 Advertisement 接口
   public void showAdvertisement(){
      System.out.println(" ************** ");
      System.out.println("让世界变得很小");
      System.out.println(" ************** ");
   }
```

```
    public String getCorpName() {
        return "联想集团";
    }
}
```

Example5_17.java

```
public class Example5_17 {
    public static void main(String args[]) {
        AdvertisementBoard board = new AdvertisementBoard();
        board.setAdvertisement(new PhilipsCorp());
        board.show();
        board.setAdvertisement(new LenovoCorp());
        board.show();
    }
}
```

注：在例5.17中，如果再增加一个Java源文件(对扩展开放)，该源文件有一个实现Advertisement接口的类IBMCorp，那么AdvertisementBoard类不需要做任何修改(对AdvertisementBoard类的修改关闭)，应用程序的主类就可以使用代码：

```
board.setAdvertisement(new IBMCorp());
board.show();
```

显示IBM公司的广告词。

如果将例5.17中的Advertisement接口、AdvertisementBoard类及LenovoCorp和PhilipsCorp类看作是一个小的开发框架，将Example5_17看作用户程序，那么框架满足“开-闭”原则，该框架相对用户的需求就比较容易维护，因为当用户程序需要使用广告牌显示IBMCorp公司的广告词时，我们只需简单地扩展框架，即在框架中增加一个实现Advertisement接口的IBMCorp类，而无须修改框架中的其他类。

扫一扫

视频讲解

▶ 5.10.5 abstract类与接口的比较

abstract类和接口的比较如下：

(1) abstract类和接口都可以有abstract方法。

(2) 接口中只能有常量，不能有变量；而abstract类中既可以有常量，也可以有变量。

(3) abstract类中也可以有非abstract方法(不是default方法，还带有方法体的方法)，但不可以有default实例方法；接口不可以有非abstract方法，但可以有default实例方法。

在设计程序时应当根据具体的分析来确定是使用抽象类还是接口。abstract类除了提供重要的需要子类去实现的abstract方法外，还提供了子类可以继承的变量和非abstract方法。如果某个问题需要使用继承才能更好地解决，例如，子类除了需要实现父类的abstract方法，还需要从父类继承一些变量或继承一些重要的非abstract方法，就可以考虑用abstract类。如果某个问题不需要继承，只是需要若干类给出某些重要的abstract方法的实现细节，就可以考虑使用接口。

5.11 小结

(1) 继承是一种由已有的类创建新类的机制。利用继承可以先创建一个共有属性的一般类，再根据该一般类创建具有特殊属性的新类。

（2）所谓子类继承父类的成员变量作为自己的一个成员变量，就像它们是在子类中直接声明一样，可以被子类中自己声明的任何实例方法操作。

（3）所谓子类继承父类的方法作为子类中的一个方法，就像它们是在子类中直接声明一样，可以被子类中自己声明的任何实例方法调用。

（4）多态是面向对象编程的又一重要特性。子类可以体现多态，即子类可以根据各自的需要重写父类的某个方法，子类通过方法的重写可以把父类的状态和行为改变为自身的状态和行为。接口也可以体现多态，即不同的类在实现同一接口时，可以给出不同的实现手段。

（5）在使用多态设计程序时，要熟练地使用上转型对象或接口回调，以便体现程序设计所提倡的“开-闭”原则。

习 题 5

扫一扫

习题

扫一扫

自测题