第6章　收费管理

单元概述

本单元以项目为导向，介绍了 Java 语言中抽象类和接口的作用和用法；通过本单元的学习，读者能够理解抽象类和接口的作用，掌握抽象类和接口的定义，以及抽象类和接口的使用。

教学重点及难点

教学重点：抽象类与抽象方法、接口。

教学难点：抽象类与抽象方法。

6.1　项目任务

编写程序为公共汽车，出租车，电影院类提供一定的收费功能，我们可以把收费功能提取出来以接口的形式定义然后被公共汽车类，出租车类和电影院类继承。

6.2　项目分析

1. 项目完成思路

将三个风马牛不相及类的共有方法提取来作为一个规范而存在，在每个具体类中根据各自的收费特点实现收费功能。

2. 需解决的问题

(1)将共有的方法提取出来后如何存放？

(2)三个具体的类如何根据各自特点实现互不相同的收费功能？在实现时和提取出来的共有方法间有什么关系？

解决以上问题涉及的技术将在下一节技术准备中详细阐述。

6.3　技术准备

6.3.1　抽象类

在继承中学习了如何在已有类的基础上扩展出新的类，随着新类的出现，类越来越具体。但是反过来却不是这样，从子类向父类追溯，类就变得更通用。在面向对象的概念中，所有的对象都是通过类来描绘的，但是并不是所有的类都是用来描绘对象的，如果一个类中没有包含足够的信息来描绘一个具体的对象，它只能被继承，派生出子类，这样的类就是抽象类。

抽象类往往用来表征在对问题领域进行分析、设计中得出的抽象概念，是对一系列看上去不同，但是本质上相同的具体概念的抽象。比如：如果进行一个图形编辑软件的开发，就会发现问题领域存在着圆、三角形这样一些具体概念，它们是不同的，但是它们又都属于形状这样一个概念，形状这个概念在问题领域是不存在的，它就是一个抽象概念。正是因为抽象的概念在问题领域没有对应的具体概念，所以用以表征抽象概念的抽象类是不能够实例化的。

1. 创建抽象类

在 Java 中定义抽象类的结构如下：

```
[public] abstract class 类名[extends 父类][implements 接口列表]
{
    属性声明及初始化；
    抽象方法的声明；
    非抽象方法声明及方法体；
}
```

提示：

(1)修饰抽象类的修饰符有 public 和默认修饰符两种；

(2)抽象类中可以有抽象方法，也可以有非抽象的方法；

(3)抽象方法是无方法体的方法。

定义抽象类时，要在关键字 class 的前面添加 abstract。例如：

```
abstract class Myclass {
    int myint;

    public abstract void noAction();←——这是抽象方法，没有方法体
    public int getMyint() {←——这是非抽象方法，包含方法体
        return myint;
    }
}
```

下面是定义一个抽象类的例子，定义了一个表示柜子的抽象类 Chest，类有两个属性，分

别代表宽和高，还有一个方法 open，另外还有一个抽象方法 storage，表示可以存放东西。

【例 6-1】定义一个抽象类。

```
abstract class Chest {
    double width;

    double high;

    public void open() {
        System.out.println("柜子能打开");
    }

    public abstract void storage();
}
```

2. 继承抽象类

抽象类不能创建对象，只有被继承了才能创建对象，继承的方式和一般的类相同。继承上面的抽象类，如下：

```
class ChildClass extends Myclass
{
    public  void noAction(){←——实现了抽象方法，即定义了方法体
     System.out.println("This is noAction");
    }
}
```

下面来看一个继承抽象类 Chest 的例子：

【例 6-2】定义抽象类的子类。

```
class Wardrobe extends Chest {
    public void storage() {←——实现抽象方法
        System.out.println("衣柜能存放衣服");
    }
}
class Cupboard extends Chest {
    public void storage() {←——实现抽象方法
        System.out.println("橱柜能存放盘子和碗");
    }
}

public class Example6_2{
    public static void main(String[] args) {
        Wardrobe w=new Wardrobe();
        w.open();
        w.storage();
```

```
        Cupboard c=new Cupboard();
        c.open();
        c.storage();
    }
}
```

运行结果如图 6-1 所示：

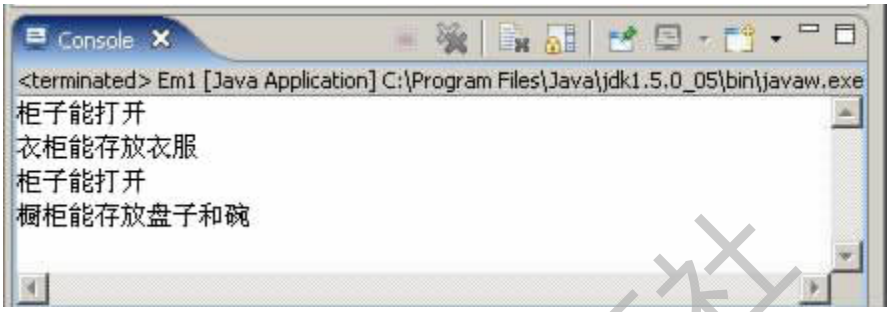

图 6-1　Example6_2 的运行结果

6.3.2　接口

接口是一种用于描述类对外提供功能规范的、能够多重继承的、特殊的抽象类。接口中只能定义静态常量和抽象方法。

那么为什么要定义接口呢？主要是某些现实问题需要用多重继承描述，但又不适合放在父类中。例如下面这种情况，如图 6-2 所示。

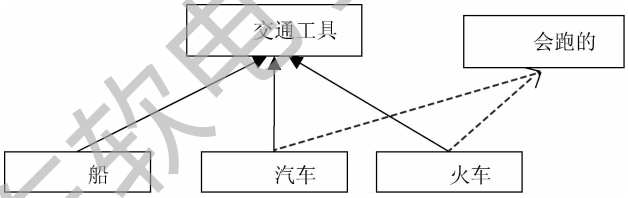

图 6-2　继承和接口的区别

由于类的多重继承能够导致方法调用的冲突，所以 Java 中的类只能单继承。但是很多时候还是需要多重继承的，Java 中的接口就可以实现多重继承，接口中不存在具体方法，不会引起方法调用的冲突。

1. 创建接口

接口规定了类的共同行为。在 Java 中，接口的声明采用 interface 关键字，接口定义的语法如下：

```
[public] interface 接口名 [extends 父接口列表]{
    //属性声明
    [public] [static] [final]属性类型 属性名=常量值;
    //方法声明
    [public] [abstract]返回值类型 方法名 ( 参数列表 );
}
```

提示：

(1)修饰接口的修饰符只有 public 和默认修饰符两种；

(2)接口可以是多重继承，接口只能继承接口，不能继承类；

(3)属性必须是常量(有初值)，方法必须是抽象的(无方法体)。

例如：

```
public  interface  IA {          ← 接口 IA
    public  abstract  int  Action1();     ← 抽象方法
    public  int  Action2();      ← 接口中，不使用 abstract 声明的方法也是抽象方法
}
```

Java 接口中不能包含具体实现的方法，如：

```
public  interface  IB{
    public void function(){
        System.out.println("Hello!");
    }
}
```

上面的用法是错误的，正确的写法是：

```
public  interface  IB{
    public void function();
}
```

在接口中除了包含抽象方法外，还可以包含常量的声明，如：

```
public  interface  IA{
    public  static final  int  CODE=1001;     ← 常量
    public  int  Action1();
    public  int  Action2();
}
```

接口与类之间的关系：类实现了接口，一个类可以同时实现多个接口，一个接口可以被多个类实现。

【例 6-3】定义一个圆柱体接口。

```
//定义一个圆柱体接口，代表所有圆柱体对象的共同行为
public interface ICylinder{
    static final double PI=3.14;  //说明圆周率常量
    public  double  area();          //计算圆柱体表面积的方法
    public  double  bulk();          //计算圆柱体体积的方法
}
```

2. 实现接口

用 implements 子句表示一个类用于实现某个接口。一个类可以同时实现多个接口，接口之间用逗号“,”分隔。在类体中可以使用接口中定义的常量，由于接口中的方法为抽象方法，所以必须在类体中加入要实现接口方法的代码，如果一个接口是从别的一个或多个父接口中继承而来，则在类体中必须加入实现该接口及其父接口中所有方法的代码。在实现一

个接口时，类中对方法的定义要和接口中的相应的方法的定义相匹配，其方法名、方法的返回值类型、方法的访问权限和参数的数目与类型信息要一致，语法格式如下：

```
class 类名 [extends 父类] [implements 接口列表]
{
    覆盖所有接口中定义的方法；
}
```

提示：

(1)一个类可以同时实现多个接口，但只能继承一个类。

(2)类中必须覆盖接口中的所有方法，而且都是公开的。

【例 6-4】实现多个接口的例子。

```
public class  A  implements IA, IB{
    public  int  Action1(){←——IA 中抽象方法的实现
        System.out.println("this is Action1! ");
    }
    public  int  Action2(){
        System.out.println("this is Action2! ");
    }
    public void function(){←——IA 中抽象方法的实现
        System.out.println("this is  function from  IB ! ");
    }
}
```

Java 中，不允许一个类继承多个类，但允许一个类同时实现多个接口。

接口与抽象类之间的关系：抽象类是类，因此，接口与类之间的关系也适用于抽象类；此外应该注意的是，一个最常用的设计模式就是，抽象类实现接口，多个具体类继承抽象类，则多个具体类也间接地实现了接口。

下面来看一个实现接口的例子：

【例 6-5】定义一个类实现圆柱体接口。

```
public class Cylinder implements ICylinder {
    double r;
    double h;
    public Cylinder(double r, double h) {
        this.r=r;
        this.h=h;
    }
    public double area() {←——实现接口规定的面积方法
        return 2 * PI * r * (h+r);
    }
    public double bulk() {←——实现接口体积方法
        return PI * r * r * h;
```

```
    }
    public static void main(String args[]) {
        ICylinder c1=new Cylinder(10, 6);
        double arearesult;
        arearesult=c1.area();
        double bulkresult;
        bulkresult=c1.bulk();
        System.out.println("面积为"+arearesult);
        System.out.println("体积为"+bulkresult);
    }
}
```

运行结果如图 6-3 所示：

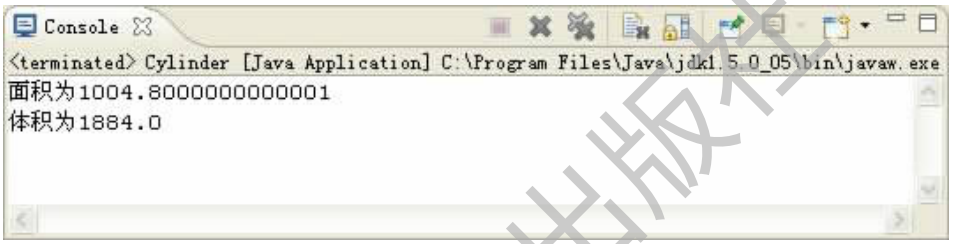

图 6-3 Cylinder 的运行结果

6.3.3 内部类

内部类就是定义在另一个类内部的类。内部类对于同一包中的其他类来说，内部类能够隐藏起来。

内部类的使用需注意以下几点：

(1)内部类可以访问其外部类中所有的属性和方法。

(2)无需创建外部类的对象，即可从内部类访问外部类的变量和方法。

(3)必须创建内部类的对象，否则无法从外部类访问内部类的变量和方法。

(4)如果内部类中有和外部类同名的变量或方法，则内部类的变量和方法将获得比外部类的变量和方法更高的优先级。

(5)不能定义 static 变量。

(6)一个类只会被这个类所调用，其他类不会使用它，你可以把它定义成一内部类，这样可以隐藏实现细节，避免错误调用。

内部类的定义如图 6-4 所示。

内部类和普通类在访问权限修饰符的使用上有所不同。普通类的访问权限修饰符可以使用 default 和 public。内部类的访问权限修饰符可以使用四个，分别是 default、public、protected 和 private。

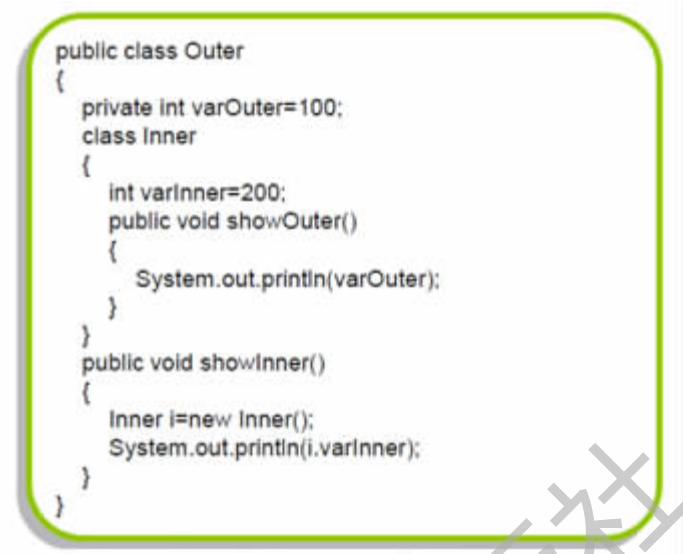

图 6-4　内部类

在 Outer 内访问 Inner，只需如下：

```
inner in = new Inner();
```

在 Outer 外访问 Inner，必须如下：

```
Outer o = new Outer();//实例化外部类
Outer.Inner oi = o.new Inner();//实例化内部类
```

【例 6-6】内部类例子。

```
/*
  *内部类与外部类的属性同名
  */
public class Outer
{
private int varOuter=100;
    private double x=3.14;  //同名变量 x

class Inner
    {
int varInner=200;
        double x=3.1415926;   //同名变量 x


public void showOuter()
        {
            System.out.println(varOuter);
            System.out.println(x);  //输出内部类 x
            System.out.println(Outer.this.x);  //输出外部类 x
        }
```

```
    }

public void showInner()
    {
        Inner i=new Inner();
        i.showOuter();  //调用内部类方法
    }
public static void main(String[] args)
    {
        Outer o=new Outer();  //调用外部类方法
        o.showInner();
    }
}
public class OuterTest
{
public static void main(String[] args)
    {
        Outer o=new Outer();
        Outer.Inner i=o.new Inner();
        System.out.println(i.varInner);
    }
}
```

运行结果如图 6-5 所示：

```
100
3.1415926
3.14
```

图 6-5　OuterTest 运行结果

还有一种内部类是静态内部类，这样的类具有以下特征：

(1)用 static 标识的内部类为静态内部类。

(2)静态内部类作为外部类的静态成员，不能访问外部类非静态成员。

(3)非静态内部类只能定义非静态成员，而静态内部类可以定义静态成员和非静态成员。

(4)使用 Outer.Inner inn=new Outer.Inner()方式实例化静态内部类。

(5)非静态内部类不可以使用上面的方式实例化。

【例 6-7】静态内部类例子。

```
public class OuterStatic
{
    private int varOuter=100;
    private static class Inner    //静态内部类
```

```
    {
      int varInner=200;
      public void showOuter()
        {
            //System.out.println(varOuter);//出错
        }
    }

public void showInner()
    {
        Inner i=new Inner();
        System.out.println(i.varInner);
    }
    public static void main(String args[]){
        OuterStatic.Inner in=new OuterStatic.Inner();
        System.out.println(in.varInner);
    }
}
```

运行结果如图 6-6 所示：

图 6-6　OuterStatic 运行结果

内部类还包括局部内部类，就是在一个类的方法体中或程序块内定义的内部类。如图 6-7 所示。

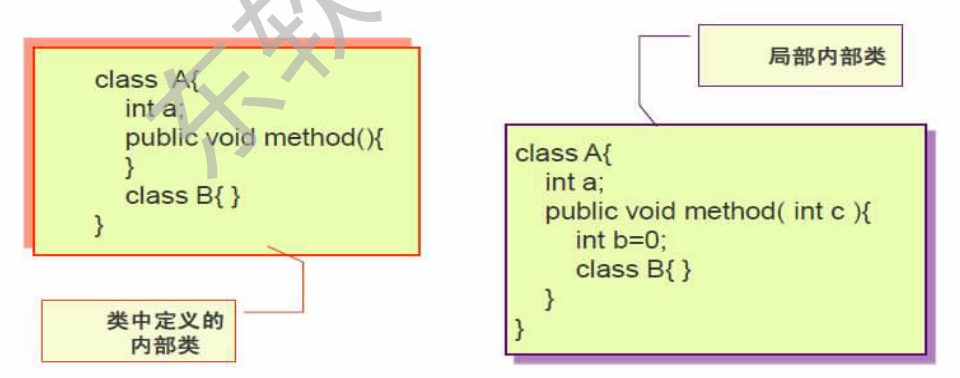

图 6-7　内部类和局部内部类

局部内部类是在方法中定义的内部类，只能访问方法中的 final 类型的局部变量，例子如下所示。

【例 6-8】方法中定义内部类例子。

```
public class Outer2 {

    int out_x=100;
```

```
    public void test() {
        class Inner {
            String x="x";

            void display() {
                System.out.println(out_x);
            }
        }
        Inner inner=new Inner();
        inner.display();
    }

    public void showStr(String str) {
        // public String str1="test Inner";//不可定义,只允许 final 修饰
        // static String str4="static Str";//不可定义,只允许 final 修饰
        String str2="test Inner";
        final String str3="final Str";
        class InnerTwo {
            public void testPrint() {
                System.out.println(out_x);// 可直接访问外部类的变量
                // System.out.println(str);//不可访问本方法内部的非 final 变量
                // System.out.println(str2);//不可访问本方法内部的非 final 变量
                System.out.println(str3);// 只可访问本方法的 final 型变量成员
            }
        }
        InnerTwo innerTwo=new InnerTwo();
        innerTwo.testPrint();
    }

    public void use() {
        // Inner innerObj=new Inner();//此时 Inner 已不可见了
        // System.out.println(Inner.x);//此时 Inner 已不可见了
    }

    public static void main(String[] args) {
        Outer2 outer=new Outer2();
        outer.test();
        outer.showStr("Str");
    }
}
```

局部内部类如图 6-8 所示。

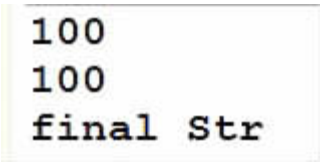

图 6-8　局部内部类

从上面的例程我们可以看出定义在方法内部的内部类的可见性更小，它只在方法内部可见，在外部类及外部类的其他方法中都不可见了。同时，它有一个特点，就是方法内的内部类连本方法的成员变量都不可访问，它只能访问本方法的 final 型成员。同时另一个需引起注意的是方法内部定义成员，只允许 final 修饰或不加修饰符，其他像 static 等均不可用。

6.4　项目学做

从项目描述可以看出除了有 Bus、Taxi、Cinema 三个类外，还应该定义一个接口 Charge，代码如下所示。

(1)定义收费接口

```
public interface Charge
{
    void receive();
}
```

(2)公共汽车类的收费功能

```
public  class Bus implements Charge
{
    public void receive()
    {System.out.println("公交收费 1 元");}
}
```

(3)出租车类的收费功能

```
public class Taxi implements Charge {
    private double distance;

    public void receive() {
        double r=10+distance / 2 * 0.4;
        System.out.println("出租车收费:"+r);
    }

    public void setDistance(double d) {
        this.distance=d;
    }
}
```

(4)电影院类的收费功能

```
class Cinema   implements Charge
{
    public void receive()
    {
        System.out.println("电影院收费 80 元");
    }
}
```

(5)测试类

```
public class TestCharge {

    public static void main(String[] args) {
        Bus bus=new Bus();
        Taxi taxi=new Taxi();
        taxi.setDistance(40);
        Cinema cinema=new Cinema();
        chargeMethod(bus);
        chargeMethod(taxi);
        chargeMethod(cinema);


    }
    public static void chargeMethod(Charge charge){
        charge.receive();
    }
}
```

项目定义了接口来实现不同类共有的一种功能,如果此处用抽象类来实现,从继承关系上是说不通的。因此接口是用来封装在不具有继承关系的类之间共有的功能,而抽象类是将子类共有的特征抽取而封装的,这是这两个类之间的区别。此外接口也是 Java 中实现多重继承的一种手段。

6.5 强化训练

在项目的基础上为公共汽车和出租车增加一个为交通工具的父类,设计交通工具的属性和方法。

6.6　课后习题

一、选择题

1. Java 语言的类间的继承关系是(　　)。

A) 多重的　　B) 单重的

C) 线程的　　D) 不能继承

2. 以下关于 Java 语言继承的说法正确的是(　　)。

A) Java 中的类可以有多个直接父类　B) 抽象类不能有子类

C) Java 中的接口支持多继承　　D) 最终类可以作为其他类的父类

3. 下列选项中,用于定义接口的关键字是(　　)。

A) interface　　B) implements

C) abstract　　D) class

4. 下列选项中,用于实现接口的关键字是(　　)。

A) interface　　B) implements

C) abstract　　D) class

5. 现有类 A 和接口 B,以下描述中表示类 A 实现接口 B 的语句是(　　)。

A) class　A　implements　B　　B) class　B　implements A

C) class　A　extends　B　　D) class　B　extends A

6. 下列选项中,定义接口 MyInterface 的语句正确的是 (　　)。

A) interface MyInterface{　}　　B) implements MyInterface {　}

C) class MyInterface{　}　　D) implements interface My{　}

二、填空题

1. 接口中所有属性均为_____、_____和_____的。

2. Java 语言的接口中可以包含_____常量和_____方法。

3. 一个类如果实现一个接口,那么它就必须实现接口中定义的所有方法,否则该类就必须定义成_____的。

4. 接口中所有方法均为_____和_____的。

5. Java 语言中,定义一个类 A 继承自父类 B,并实现接口 C 的类头是_____。

6. 下面是定义一个接口 A 的程序,完成程序填空。

```
public _____ A
{
   public static final double PI=Math.PI;
public _____ double area(double a, double b);
}
```

7. 下面是定义一个接口 A 的程序，完成程序填空。

```
public interface   A
{
   public static _____ double PI=3.14159;
   public abstract double area(double a, double b)_____
}
```

8. 定义一个抽象类，包括能求面积的抽象方法。

```
public_____ class Test6 {
    public abstract double area()____}
```

三、编程题

1. 定义商品类 Goods，包含单价 unitPrice 和数量 account 两个属性，方法包括构造方法和价格计算方法 totalPrice()。

定义接口 VipPrice，包含 DISCOUNT 属性和 reducedPrice()方法，使 VIP 会员享受商品价格 8.5 折待遇。

定义服装子类 Clothing，它继承商品类 Goods 并实现接口 VipPrice，并有服装样式 style 属性、构造方法和 toString 方法。

编写一个测试类，创建一种服装(200,1,男装)，利用 toString 方法输出服装信息。

2. 创建一个接口，接口的名字叫 TestInterface，接口里至少有一个常量 String myVar="Helo Interface"，两个抽象方法 write()和 read()。

3. 定义一个有抽象方法 display()的超类 SuperClass，以及提供不同实现方法的子类 SubClassA 和 SubClassB，并创建一个测试类 PolyTester，分别创建 SubClassA 和 SubClassB 的对象。调用每个对象的 display()。

要求：输出结果为：display A
display B

4. 编写 2 个接口：InterfaceA 和 InterfaceB；在接口 InterfaceA 中有个方法 void printCapitalLetter()；在接口 InterfaceB 中有个方法 void printLowercaseLetter()；然后写一个类 Print 实现接口 InterfaceA 和 InterfaceB，要求 printCapitalLetter()方法实现输出大写英文字母表的功能，printLowercaseLetter()方法实现输出小写英文字母表的功能。再写一个主类 E，在主类 E 的 main 方法中创建 Print 的对象并赋值给 InterfaceA 的变量 a，对象 a 调用 printCapitalLetter 方法；最后再在主类 E 的 main 方法中创建 Print 的对象并赋值给 InterfaceB 的变量 b，对象 b 调用 printLowercaseLetter 方法。

5. 按要求编写 Java 程序：

(1)编写一个接口：InterfaceA，只含有一个方法 int method(int n)；

(2)编写一个类：ClassA 来实现接口 InterfaceA，实现 int method(int n)接口方法时，要求计算 1 到 n 的和；

(3)编写另一个类：ClassB 来实现接口 InterfaceA，实现 int method(int n)接口方法时，要求计算 n 的阶乘(n!)；

(4)编写测试类 E，在测试类 E 的 main 方法中使用接口回调的形式来测试实现接口的类。