第 5 章　继承和接口

本章主要讲解 Java 语言继承和多态的概念及其实现；介绍包和接口的概念及其使用。

- 继承和多态的概念及其实现
- super 的使用
- 包的使用
- 接口的使用

5.1　类的继承

继承性是面向对象程序设计语言的一个重要特征，通过继承可以实现代码的复用。Java 语言中，所有的类都是直接或间接地继承 java.lang.Object 类。子类继承父类的属性和方法，同时也可以增加属性和方法，在 Java 语言中不支持多继承，但可通过实现接口来实现多继承功能。

5.1.1　类继承的实现

1. 创建子类

Java 中的继承是通过 extends 关键字来实现的，在定义新类时使用 extends 关键字指明新类的父类，就在两个类之间建立了继承关系。

创建子类的一般格式为：

```
[修饰符] class 子类名 extends 父类名 {
    //类体
}
```

子类名为 Java 标识符，子类是父类的直接子类。如果父类又继承某个类，则子类存在间接父类。如果默认 extends 子句，则该类的父类为 java.lang.Object。子类可以继承父类中访问控制为 public、protected、default 的成员变量和方法，但不能继承访问控制为 private 的成员变量和方法。

例 5-1　类的继承实例。

```
class Point{                                    //定义父类
    int x,y;                                    //表示位置
    void setXY(int a,int b){ x=a;y=b;}
    void showLocation(){                        //显示位置
```

```
            System.out.println("Location:("+x+","+y+")");
        }
    }
    class Rectangle extends Point{              //定义子类
        int width,height;                       //定义矩形的宽和高
        int area(){return width*height; }       //计算矩形的面积
        void setWH(int w,int h){width=w; height=h;}
        void showLocArea(){                     //显示位置和面积
            showLocation();                     //继承父类成员
            System.out.println("Area:"+area());
        }
    }
```

程序运行结果为：

```
Location:(100,200)
Area:700
```

2. 成员变量的隐藏和方法的重写

在类的继承中，若子类定义了与父类相同名字的成员变量，则子类继承父类的成员变量被隐藏。这里所谓的隐藏是指子类拥有两个相同名字的变量，一个继承自父类，另一个是自己声明的。当子类执行继承自父类的方法时，处理的是继承自父类的成员变量；当子类执行它自己声明的方法时，操作的是它自己声明的变量，而把继承父类的相同名字的变量“隐藏”起来。

子类定义了与父类相同的成员方法，包括相同的名字，参数列表和相同返回值类型。这种情况称为方法重写。

一般出现下面情况需要使用方法重写：

（1）子类中实现与父类相同的功能，但采用的算法或计算公式不同。

（2）在子类的相同方法中，实现的功能比父类更多。

（3）在子类中需要取消从父类继承的方法。

子类通过成员变量的隐藏和方法的重写可以把父类的属性和行为改变为自身的属性和行为。

例 5-2　方法重写实例。

```
import java.io.*;
class Point{
    int x,y;
    void setXY(int a,int b){ x=a;y=b; }
    void show(){ System.out.println("Location:("+x+","+y+")"); }
}
class Rectangle extends Point{
    int y;
    int width,height;
    int area(){ return width*height; }
    void setWHY(int w,int h,int yy){width=w; height=h; y=yy; }
    void show(){
        System.out.println("Location:("+x+","+y+")");
```

```
            System.out.println("Area:"+area());
        }
    }
```

程序运行结果为：

```
Location:(100,300)
Area:700
```

5.1.2 super 关键字

super 表示当前对象的直接父类对象，是当前对象的直接父类对象的引用。所谓直接父类是相对于当前对象的其他“父类”而言。super 代表的是直接父类。若子类的数据成员或成员方法名与父类的数据成员或成员方法名相同，当要调用父类的同名方法或使用父类的同名的数据成员时，则可以使用关键字 super 来指明父类的数据成员和方法。super 的使用方法有以下 3 种：

- 用来访问直接父类中被隐藏的数据成员，其使用形式如下：
  super.数据成员
- 用来调用直接父类中被重写的的成员方法，其使用形式如下：
  super.成员方法
- 用来调用直接父类的构造方法，其使用形式如下：
  super(参数)

例 5-3 super 的使用。

```
class A{                                          //定义父类
    int x,y;
    public A(int x,int y){ this.x=x;   this.y=y; }     //父类构造方法
     public void display(){ System.out.println("In class A: x="+x+" , y="+y);}
}
class B extends A{                                //子类
    int a,b;
    public B(int x,int y,int a,int b){            //子类构造方法
        super(x,y);this.a=a; this.b=b;
    }
    public void display(){                        //方法重写
        super.display();
        System.out.println("In class B: a="+a+" , b="+b);
    }
}
```

程序运行结果为：

```
In class A: x=1 , y=2
In class B: a=3 , b=4
```

5.1.3 抽象类和抽象方法

1. 抽象类与抽象方法的声明

在 Java 中用 abstract 关键字修饰的类称为抽象类。用 abstract 关键字修饰的方法，称为抽

象方法。当一个类的定义完全表示抽象概念时，它不应该被实例化为一个对象，因此不能为抽象类实例化对象，也就是说 abstract 类必须被继承，abstract 方法必须被重写。抽象类体现数据抽象的思想，是实现程序多态性的一种手段。定义抽象类的目的是子类共享。子类可以根据自身需要扩展抽象类。

抽象类和抽象方法有以下特征：

- 抽象类不能实例化，即不能用 new 生成一个实例对象。
- 抽象方法只有方法名、参数列表及返回值类型，抽象方法没有方法体。
- 抽象方法必须在子类中给出具体实现。
- 一个抽象类里可以没有定义抽象方法。但只要类中有一个方法被声明为抽象方法，则该类必须为抽象类。
- 若一个子类继承一个抽象类，则子类需用覆盖的方式来实例化该抽象父类中的抽象方法。若没有完全实例化所有的抽象方法，则子类仍是抽象的。
- 抽象方法可与 public、protected 复合使用，但不能与 final、private 和 static 复合使用。

在以下情况中，某个类将被定义为抽象类：

- 当类的一个或多个方法为抽象方法时。
- 当类为一个抽象类的子类，并且没有为所有抽象方法提供实现细节或方法主体时。
- 当类实现一个接口，并且没有为所有抽象方法提供实现细节或方法主体时。

声明一个抽象类的格式为：

```
abstract  class 类名{
    …
}
```

抽象类中可以包含抽象方法，对抽象方法只需声明，而不需要具体的内容，格式如下：

```
abstract  数据类型  方法名([paramlist]);
```

对于抽象方法声明格式，abstract 保留字不能缺少，还需要注意如下内容：

（1）声明格式中没有{}。

（2）最后的“;”不能省略。

抽象类首先是一个类，因此具有类的一般属性。抽象类必须被其他类继承，抽象类中不一定要包含抽象方法。但是如果一个类中包含了抽象方法，也就是 abstract 关键字修饰的方法，那么该类就必须声明为抽象类。在类中的方法中除构造方法、静态方法、私有方法不能声明为抽象方法外，其他任何方法都可以被声明为抽象方法。

例如：

```
abstract class Employee {            //抽象类
    int basic = 2000;
    abstract void salary();          //抽象方法
 }
class Manager extends Employee{
    void salary() {
      System.out.println("工资等于 "+basic*5+8672);
 }
}
class Worker extends Employee   {
```

```
    void salary() {
 System.out.println("工资等于 "+basic*2+4567);
 }
}
```

2. 抽象类与抽象方法的使用

由于抽象类只预先确定了总体结构，缺少实际内容或实现过程，又不能被实例化，所以要发挥它的作用，只能被继承，把它作为父类，在子类中创建对象，同时将抽象方法重写。

例 5-4 定义车类。

```
import java.awt.Color;
abstract class Car {
        //公用数据成员声明
        public Color color;                          //车辆颜色
        public int gearNum;                          //挡位数
        public String  tiretype;                     //轮胎型号
        public float engine;                         //引擎排气量
         //公共抽象方法声明
        public abstract void shiftgear();            //换挡
        public abstract void brake();                //刹车
        public abstract void aircon();               //开冷气
        public abstract void headlight();            //开大灯
}
 class Audi extends Car{
      static int gearNum=5;                          //声明 gearNum 为类变量
       public Audi () {
            tiretype="GoodTire2085BT";               //轮胎型号
            engine=2400.0f;                          //排气量
      }
      public void   shiftgear(){                     //换挡
            System.out.println("轿车换挡方式："+ gearNum+"挡");
      }
     public void brake(){System.out.println("助力刹车系统");}      //刹车
          public void aircon(){};                                 //开冷气
          public void headlight(){};                              //开大灯
      }
      public class MyCar extends Audi{
            private Color color;
            public MyCar() {     color=Color.blue; }              //设置车辆颜色
            public void equipment(){
                System.out.println("我的车挡位数："+this.gearNum);
                System.out.println("我的车颜色："+this.color);
                System.out.println("我的车轮胎型号："+this.tiretype);
                System.out.println("我的车引擎排气量："+this.engine);
            }
            public void shiftgear(){                              //换挡-重写的新方法
                super.shiftgear();
                System.out.println("我的车换挡方式：  "+this.gearNum+"挡" );
```

```
        }
    }
```

程序的运行结果为：

```
我的车挡位数：5
我的车颜色：java.awt.Color[r=0,g=0,b=255]
我的车轮胎型号：GoodTire2085BT
我的车引擎排气量：2400.0
轿车换挡方式：5 挡
我的车换挡方式：5 挡
助力刹车系统
```

5.1.4　类对象之间的类型转换

类对象之间的类型转换，是指父类对象与子类对象之间在一定条件下的相互转换。父类对象与子类对象之间的相互转换规则如下：

（1）子类对象可以隐式，也可以显示转换为父类对象。

（2）处于相同类层次的类对象之间不能进行转换。

（3）父类对象在一定的条件下可以转换成子类对象，但必须使用强制类型转换。

例 5-5　类对象之间的类型转换的使用。

```
class SuperClass{
    int a=5,b=8,c=85;
    void show(){System.out.println("a*b="+(a*b));}
}
class SubClass extends SuperClass{
    int b=26,d=32;
    void show() { System.out.println("b+d="+(b+d));}
}
public class ClassExchangeDemo{
    public static void main(String args[]){
        SuperClass super1,super2;          //声明父类对象
        SubClass sub1,sub2;                //声明子类对象
        super1=new SuperClass();
        sub1=new SubClass();
        System.out.println("super1.a="+super1.a+"\tsuper1.b="+super1.b+"\tsuper1.c="+super1.c);
        super1.show();
        System.out.println("sub1.b="+sub1.b+"\tsub1.c="+sub1.c+"\tsub1.d="+sub1.d);
        sub1.show();
        super2=(SuperClass)sub1;           //子类对象转换为父类对象
        System.out.println("super2.a="+super2.a+"\tsuper2.b="+super2.b+"\tsuper2.c="+super2.c);
        System.out.println("super2.show():\t");
        super2.show();
        sub2=(SubClass)super2;             //父类对象转换为子类对象
        System.out.println("sub2.a="+sub2.a+"\tsub2.b="+sub2.b+"\tsub2.d="+sub2.d);
        System.out.println("sub2.show():\t");
        sub2.show();
    }
```

```
}
```

程序运行结果为：

```
super1.a=5          super1.b=8          super1.c=85
a*b=40
sub1.b=26           sub1.c=85           sub1.d=32
b+d=58
super2.a=5          super2.b=8          super2.c=85
super2.show():
b+d=58
sub2.a=5            sub2.b=26           sub2.d=32
sub2.show():
b+d=58
```

子类对象可以被看作是其父类的对象，因此在程序中子类对象可以引用父类的数据成员。子类对象转换为父类对象时，可以使用强制类型转换，也可以使用默认转换方式，直接把子类对象引用赋值给父类对象引用。若父类对象引用指向的实际是一个子类对象的引用，则这个父类对象使用强制类型转换可转换为子类对象。

5.2 类的多态

多态是面向对象的又一个重要特性，与继承密切相关。在面向对象程序设计中，类的功能通过类的方法实现。多态是为类创建多个同名的方法，通过方法的重载或子类重写父类方法实现多态。

5.2.1 方法重载

在同一个类中定义了多个同名而内容不同的成员方法，称这些方法是重载的方法。重载的方法主要通过参数列表中参数的个数、参数的数据类型和参数的顺序来进行区分。在编译时，Java 编译器检查每个方法所用的参数数目和类型，然后调用正确的方法。

例 5-6 求加法重载的例子。

```
public class AddOverridden{
    int add(int a,int b){ return (a+b);}
    int add(int a,int b,int c){ return (a+b+c);}
    double add(double a,double b){ return (a+b);}
    double add(double a,double b,double c){ return (a+b+c);}
}
```

程序运行结果为：

```
Sum(37,73)=110
Sum(10,33,67)=110
Sum(97.88,36.99)=134.87
Sum(9.8,3.73,6.97)=20.5
```

该类中定义了 4 个名为 add 的方法：第一个方法是计算两个整数的和；第二个方法是计算三个整数的和。第三个方法是计算两个双精度浮点数的和；第四个方法是计算三个双精度浮点数的和。Java 编译器根据方法引用时提供的实际参数选择执行对应的重载方法。

5.2.2　方法重写

通过面向对象系统中的继承机制，子类可以继承父类的方法。但是，子类的某些特征可能与从父类中继承来的特征有所不同，为了体现子类的这类特性，Java 允许子类对父类的同名方法重新进行定义，即在子类中定义与父类中已定义的名称相同而内容不同的方法。这种多态称为方法重写，也称为方法覆盖。

由于重写的同名方法存在于子类对父类的关系中，所以只需要在方法引用时指明引用的是父类的方法还是子类的方法，就可以很容易把它们区分开来。对于重写的方法，Java 运行时系统根据调用该方法的实例的类型来决定选择哪个方法调用。对于子类的一个实例，如果子类重写了父类的方法，则运行时系统调用子类的方法。如果子类继承父类的方法，则运行时系统调用父类的方法。

例 5-7　重写方法的调用。

```
class A{
    void display(){ System.out.println("A's method display() called!");}
    void print(){System.out.println("A's method print() called!");}
}
class B extends A{
    void display(){ System.out.println("B's method display() called!");}
}
public class OverRiddenDemo1{
    public static void main(String args[]){
        A a1=new A();
        a1.display();
        a1.print();
        A a2=new B();
        a2.display();
        a2.print();
    }
}
```

程序运行结果为：

```
A's method display() called!
A's method print() called!
B's method display() called!
A's method print() called!
```

在上例中，定义了类 A 和类 A 的子类 B。然后声明类 A 的变量 a1、a2，用 new 建立类 A 的一个实例和类 B 的一个实例，并使 a1、a2 分别存储 A 类实例引用和 B 类实例引用。Java 运行时系统分析该引用是类 A 的一个实例还是类 B 的一个实例，从而决定调用类 A 的方法 display()还是调用类 B 的方法 display()。

方法重写时要遵循两个原则：

（1）改写后的方法不能比被重写的方法有更严格的访问权限。

（2）改写后的方法不能比被重写的方法产生更多的例外。

进行方法重写时必须遵循这两个原则，否则编译器会指出程序出错。编译器加上这两个

限定，是为了与 Java 语言的多态性特点一致而做出的。

例 5-8 方法重写注意的原则。

```
class SuperClass{
    public void fun(){}
}
class SubClass extends SuperClass{
    protected void fun(){}
}
public class OverRiddenDemo2{
    public static void main(String args[]){
        SuperClass a1=new SuperClass();
        SuperClass a2=new SubClass();
        a1.fun();
        a2.fun();
    }
}
```

上面程序编译时产生下列错误：

```
fun() in SubClass cannot override fun() in SuperClass;
attempting to assign weaker access privileges; was public
```

产生错误的原因在于：子类中重写的方法 fun()的访问权限比父类中被重写的方法有更严格的访问权限。

5.3 接口（interface）

在 Java 中通过 extends 实现单继承，从类的继承上来讲，Java 只支持单继承，这样可避免多继承中各父类含有同名成员时在子类中发生引用无法确定的问题。但是，为了某些时候的操作方便、增加 Java 的灵活性，达到多继承的效果，可利用 Java 提供的接口来实现。

一个接口允许从几个接口继承而来。Java 程序一次只能继承一个类但可以实现几个接口。接口不能有任何具体的方法。接口也可用来定义由类使用的一组常量。

5.3.1 接口的定义

Java 中的接口是特殊的抽象类，是一些抽象方法和常量的集合，其主要作用是使得处于不同层次上以至于互不相干的类能够执行相同的操作、引用相同的值，而且可以同时实现来自不同类的多个方法。

接口与抽象类的不同之处在于：接口的数据成员必须被初始化；接口中的方法必须全部都声明为抽象方法。

接口的一般定义格式为：

```
[public] interface 接口名{
//接口体
}
```

其中 interface 是接口的关键字，接口名是 Java 标识符。如果缺少 public 修饰符，则该接口只能被与它在同一个包中的类实现。接口体中可以含有下列形式的常量定义和方法声明：

```
[public] [static] [final] 类型　常量名=常量值;　　//数据成员必须被初始化
[public] [abstract] 方法类型　方法名([参数列表]);　　//方法必须声明为抽象方法
```

其中，常量名是 Java 标识符，通常用大写字母表示；常量值必须与声明的类型相一致；方法名是 Java 标识符，方法类型是指该方法的返回值类型。final 和 abstract 在 Java 中可以省略。

例如，下列程序段声明了一个接口：

```
//定义程序使用的常量和方法的接口
public interface myinterface {
    double price = 8750.00;
    final int counter = 555;
    public void add(int x, int y);
    public void volume(int x,int y, int z);
}
```

5.3.2　接口的实现

接口中只包含抽象方法，因此不能像一般类一样，使用 new 运算符直接产生对象。用户必须利用接口的特性来打造一个类，再用它来创建对象。利用接口打造新的类的过程，称为接口的实现。接口实现的一般语法格式为：

```
class 类名 implements 接口名称{　　//接口的实现
        //类体
}
```

例 5-9　求给定圆的面积。

```
interface shape{
    double PI=3.14159;
    abstract double area();
}
class Circle implements shape{
    double radius;
    public Circle(double r){radius=r;}
    public double area(){return PI*radius*radius;}
}
```

程序运行结果为：

```
Circle area=314.159
```

在类实现一个接口时，如果存在接口中的抽象方法，在类中没有具体实现，则该类是一个抽象类，不能生成该类的对象。

5.3.3　接口的继承

接口也可以通过关键字 extends 继承其他接口。子接口将继承父接口中所有的常量和抽象方法。此时，子接口的非抽象实现类不仅要实现子接口的抽象方法，而且需要实现父接口的所有抽象方法。

例 5-10　接口的继承。

```
interface shape2D{
    double PI=3.14159;
```

```
    abstract double area();
}
interface shape3D extends shape2D{
    double volume();
}
class Circle implements shape3D{
    double radius;
    public Circle(double r)  {radius=r;}
    public double area(){return PI*radius*radius; }                    //实现间接父接口的方法
    public double volume(){return 4*PI*radius*radius*radius/3; }       //实现直接父接口的方法
}
```

程序运行结果为：

```
Circle area=706.85775
Circle vloume=14137.155
```

在 Java 中不允许有多个父类的继承，因为 Java 的设计是以简单、实用为导向，而利用类的多继承将使得问题复杂化，与 Java 设计意愿相背。虽然如此，但通过接口机制，可以实现多重继承的处理。通过将一个类实现多个接口，就可以达到多重继承的目的。

例 5-11　实现多个接口。

```
interface shape{
    double PI=3.14159;
    double area();
    double volume();
}
interface color{void setcolor(String str); }
class Circle implements shape,color{
    double radius;
    String color;
    public Circle(double r){ radius=r;}
    public double area(){return PI*radius*radius;}
    public double volume(){ return 4*PI*radius*radius*radius/3;}
    public void setcolor(String str){ color=str;}
    String getcolor(){ return color;}
}
```

程序运行结果为：

```
Circle area=706.85775
Circle vloume=14137.155
Circle color=Red
```

注意：在实现接口的类中，定义抽象方法的方法体时，一定要声明方法为 public，否则编译会出现如下的出错信息：

attempting to assign weaker access privileges;

中文含义为：试图缩小方法的访问权限。

5.3.4　接口的多态

接口的使用使得方法的描述和方法的功能实现分开处理，有助于降低程序的复杂性，使程序设计灵活，便于扩充和修改。

例 5-12　求给定图形的面积。

```
// InterfaceDemo4.java
//接口 shape
interface shape{
      double PI=3.14159;
      abstract double area();
}
//定义 Rectangle 类实现接口 shape
class Rectangle implements shape{
         double width,height;
         public Rectangle(double w,double h){width=w; height=h;}
          public double area(){ return width*height; }
 }
 //定义 Circle 类实现接口 shape
class Circle implements shape{
          double radius;
          public Circle(double r){radius=r;}
          public double area(){return PI*radius*radius;}
}
```

程序运行结果为：

```
Rectanle area=300.0
Circle area=314.159
```

在上面的例子中，定义了接口 shape，并在类 Rectangle 和 Circle 中实现了接口，但两个类对接口 shape 的抽象方法 area()的实现是不同的，实现了接口的多态。

5.4　包（package）

在 Java 中，把复用的代码组织到一起，称为“包”。包是一种将相关类、接口或其他包组织起来的集合体。目的是为了将包含类代码的文件组织起来，易于查找和使用。包不仅能包含类和接口，还能包含其他包，形成有层次的包空间。包有助于避免命名冲突。当使用很多类时，确保类和方法名称的唯一性是非常困难的。包形成层次命名空间，缩小了名称冲突的范围，易于管理名称。

5.4.1　包的创建

建立一个包的方法很简单，只要将 package 这个关键字放在一个程序中所有类或实现接口的类或接口声明前，并选定一个包名称，这样所有用到此包名称的类及接口就成了此包的成员。创建包的一般语法格式为：

```
package   PackageName;
```

关键字 package 后面的 PackageName 是包名。利用这个语句可以创建一个具有指定名字的包，当前.java 文件中的所有类都被放在这个包中。

在 Java 程序中，package 语句必须是程序的第一个非注释、非空白行、行首无空格的一行语句来说明类和接口所属的包。

例如：

```
package MyPackage;
package MyPackage1. MyPackage2;
```

创建包就是在当前文件夹下创建一个子文件夹，存放这个包中包含的所有类的.class 文件。在 package MyPackage1. MyPackage2;语句中的符号“.”代表目录分隔符，说明这个语句创建两个文件夹：第一个是当前文件夹下的子文件夹 MyPackage1；第二个是 MyPackage1 文件夹下的 MyPackage2 文件夹，当前包中的所有类文件就存在这个文件夹下。

若源文件中未使用 package，则该源文件中的接口和类位于 Java 的默认包中。在默认包中，类之间可以相互使用 public、protected 或默认访问权限的数据成员和成员方法。默认包中的类不能被其他包中的类引用。

5.4.2 包的引用

将类组织成包的目的是为了更好地利用包中的类。一般情况下一个类只能引用与它在同一个包中的类。如果需要使用其他包中的 public 类，则可以通过 import 这个关键词来引入，例如：

```
import java.awt.Color;
```

就是把 java.awt 包里的 Color 类引用进来，如果需要引用整个包内所有的类及接口时，就使用*号：

```
import java.awt.*;
```

在一个类中引用一个包中的类时，可采用两种方式：

（1）类长名（Long Name），即加上包名称的类名，如：

```
javax.swing.JButton button1=new javax.swing.JButton ();
```

（2）类短名（Short Name），需要在类程序最前面引入包，然后使用该类名，如：

```
import java.awt.*;
…
Color color1=new Color();
```

例 5-13 包的应用实例。

```
//shapes.java
package shape;
public interface shapes{
    abstract double area();
    abstract double circulms();
}
//locate.java
package shape;
class locate{
    public int x,y;
```

```
        public locate(int x,int y){ this.x=x;this.y=y;}
}
//rectangle.java
package shape;
public class rectangle extends locate implements shapes{
        public int width,height;
        public double area(){return width*height;}
        public double circulms(){return 2*(width+height);}
        public rectangle(int x,int y,int w,int h){ super(x,y); width=w; height=h;}
}
//circle.java
package shape;
public class circle extends locate implements shapes{
        public double radius;
        public double area(){return Math.PI*radius*radius;}
        public double circulms(){return 2*Math.PI*radius;}
        public circle(int x,int y,double r){ super(x,y); radius=r;}
}
//PackageDemo.java
package mypackage;
import shape.*;
public class PackageDemo{
        public static void main(String []args){
                …
        }
}
```

分别编译程序 shapes.java、locate.java、rectangle.java、circle.java，则建立包 shape，包中包含接口 shapes 和类 locate、rectangle、circle。编译并运行 PackageDemo.java。

5.4.3　设置 CLASSPATH 环境变量

包是一种组织代码的有效手段，包名实际上就是指出了程序中需要使用的.class 文件的所在之处。另一个能指明.class 文件夹所在结构的是环境变量 CLASSPATH。当一个程序找不到它所需使用的其他类的.class 文件时，系统就会自动到 CLASSPATH 环境变量所指明的路径下去查找。

环境变量的设置在前面已经叙述过，不再赘述。

5.5　Java 类库及主要类的使用

Java 提供了强大的应用程序接口（Java API），即 Java 类库或 Java API 包。它包括大量已经设计好的工具类，帮助编程人员进行字符串处理、绘图、数学计算、网络应用等方面的工作。在程序中合理和充分利用 Java 类库提供的类和接口，可以大大提高编程的效率，写出短小精悍的程序，取得良好效果。

5.5.1 Java API 常用包

绝大多数 Java API 包都是以“java.”开头，以区别用户创建的包。Java API 包含多种包，下面主要介绍常用的几种包：

1. java.lang 包

在所有的 Java API 类库中，java.lang 包是核心包，它提供 Java 语言中 Object、String 和 Thread 等核心类与接口。这些核心类与接口自动导入到每个 Java 程序中，没有必要显示地导入它们。java.lang 包中还包含基本类型包装类、访问系统资源的类、数值类和安全类等。

下面分别介绍 java.lang 包中的常用类库。

（1）java.lang.Object

Object 类处于 Java 层次结构的最上层，是所有类的父类。Java 语言中所有类都是直接或间接继承 Object 类得到的。

（2）java.lang.Object

└java.lang.Boolean

Boolean 与 Java 中的 boolean 布尔数据类型有所不同，该类将一个基本类型 boolean 的值封装在一个对象里。

（3）java.lang.Object

└java.lang.Character

该类封装了 Java 的基本字符类型数据及其对象的属性特征。

（4）java.lang.Object

└java.lang.Character.Subset

该类是 Unicode 字符集的一个特殊子集。

（5）java.lang.Object

└java.lang.Class

Class 类封装了类和对象的属性特征，包含解释一个 Java 类的信息。没有公共的构造方法，Class 对象由 Java 虚拟机在通过调用类装载器中的 defineClass 方法装载类时自动构造。

（6）java.lang.Object

└java.lang.ClassLoader

抽象字节码的文件装载类，定义将 Java 执行文件装载到 Java 执行环境内的规则。

（7）java.lang.Object

└java.lang.Compiler

用于支持 Java 本地码编译器以及相关的服务。

（8）java.lang.Object

└java.lang.Number

└java.lang.Byte

字节类，该类为 byte 类型数值提供一个对象包装器，该类对象包含一个 byte 类型数据。

主要数据成员：

- Static byte MAX_VALUE：代表 byte 类型整数最大值。
- Static byte MIN_VALUE：代表 byte 类型整数最小值。

主要成员方法：

- Static byte parse Byte (String s)：将字符串 s 转换为 byte 类型数据。
- Static byte parse Byte(String s, int radix)：将字符串 s 转换为 radix 进制 byte 类型数据。
- String toString()：将当前 byte 对象转换为字符串。
- static String toString(byte d)：将 double 类型数据 d 转换为字符串。
- static Byte valueOf(String s)：返回表示字符串 s 的 Byte 类型对象。
- static Byte valueOf(String s, int radix)：返回表示字符串 s 的 radix 进制 Byte 类型对象。

（9）java.lang.Object

└java.lang.Number

└java.lang.Double

浮点数类，该类为 double 数值提供一个对象包装器，同时还提供将双精度数转换或传递给 Java 对象作为参数接收的方法。

主要数据成员：

- static double MAX_VALUE：代表 double 类型整数最大值。
- static double MIN_VALUE：代表 double 类型整数最小值。

主要成员方法：

- static double parseDouble(String s)：将字符串 s 转换为 double 类型数据。
- String toString()：将当前 Double 对象转换为字符串。
- static String toString(double d)：将 double 类型数据 d 转换为字符串。
- static Double valueOf(String s)：返回表示字符串 s 的 Double 类型对象。

（10）java.lang.Object

└java.lang.Number

└java.lang.Float

单精度浮点数类，该类将 float 类型数据及其操作封装在对象中。

主要数据成员：

- static float MAX_VALUE：代表 float 类型整数最大值。
- static float MIN_VALUE：代表 float 类型整数最小值。

主要成员方法：

- static float parseFloat(String s)：将字符串转换为 float 类型数据。
- String toString()：将当前 Float 类对象转换为字符串。
- static String toString(float f)：将 float 类型数据转换为字符串。
- static Float valueOf(String s)：返回表示字符串 s 的 Float 类对象。

（11）java.lang.Object

└java.lang.Number

└java.lang.Interger

将 Java 的基本类型 interger 封装在一个对象中。

主要数据成员：

- public static final int MIN_VALUE：代表 int 类型整数最小值。

- public static final int MAX_VALUE：代表 int 类型整数最大值。

主要成员方法：

- public static String toString(int i)：将 i 转换成字符串。
- public static String toString(int i,int radix)：将 i 转换成 radix 进制整数字符串。
- public static String toHexString(int i)：将 i 转换成 16 进制整数字符串。
- public static String toOctalString(int i)：将 i 转换成 8 进制整数字符串。
- public static String toBinaryString(int i)：将 i 转换成 2 进制整数字符串。
- public static int parseInt(String s,int radix) throws NumberFormatException：将字符串 s 转换成 radix 进制整数。
- public static int parseInt(String s) throws NumberFormatException：将字符串 s 转换成整数。
- static Integer valueOf(String s)：将字符串 s 转换成 Integer 对象。
- static Integer valueOf(String s, int radix)：将字符串 s 转换成 radix 进制的 Integer 对象。

（12）java.lang.Object

└java.lang.Number

└java.lang. Long

将 Java 的基本类型 long 封装在一个对象中。

主要数据成员：

- public static final int MIN_VALUE：代表 long 类型整数最小值。
- public static final int MAX_VALUE：代表 long 类型整数最大值。

主要成员方法：

- public static String toString(long i)：将 i 转换成字符串。
- public static String toString(long i,int radix)：将 i 转换成 radix 进制整数字符串。
- public static String toHexString(long i)：将 i 转换成 16 进制整数字符串。
- public static String toOctalString(long i)：将 i 转换成 8 进制整数字符串。
- public static String toBinaryString(longt i)：将 i 转换成 2 进制整数字符串。
- public static long parseInt(String s,int radix) throws NumberFormatException：将字符串 s 转换成 radix 进制 long 型整数。
- public static long parseInt(String s) throws NumberFormatException：将字符串 s 转换成 long 型整数。
- static Long valueOf(String s)：返回表示字符串 s 的 Long 对象。
- static Long valueOf(String s, int radix)：返回表示字符串 s 的 radix 进制的 Long 对象。

（13）java.lang.Object

└java.lang.Number

└java.lang.Short

将 Java 的基本类型 short 封装在一个对象中。

主要数据成员：

- public static final short MIN_VALUE：代表最小 short 类型整数。
- public static final short MAX_VALUE：代表最大 short 类型整数。

主要成员方法：

- public static String toString(short i)：将 i 转换成字符串。
- public static String toString(short i,int radix)：将 i 转换成 radix 进制整数字符串。
- public static short parseInt(String s,int radix)throws NumberFormatException：将字符串 s 转换成 radix 进制 short 型整数。
- public static int parseInt(String s)throws NumberFormatException：将字符串 s 转换成 short 型整数。
- static Short valueOf(String s)：返回表示字符串 s 的 Short 对象。
- static Short valueOf(String s, int radix)：返回表示字符串 s 转换成 radix 进制的 Short 对象。

（14）java.lang.Object

　　└java.lang.Math

数学类，封装了一个标准的数学库，包含一些完成基本算术运算的方法。前面讲述过，不再赘述。

（15）java.lang.Object

　　└java.lang.Runtime

运行类，封装了 Java 的执行环境，可以通过 getRunTime()方法获得该类的实例。

（16）java.lang.Object

　　└java.lang.System

提供与平台有关的系统功能，包括输入、输出、加载动态库等。

（17）java.lang.Object

　　└java.lang.String

字符串类，用于常量字符串。Java 程序中的所有字符串都是通过该类的实例来实现的。

（18）java.lang.Object

　　└java.lang.StringBuffer

字符串缓冲区类，用于可变长度的字符串。

（19）java.lang.Object

　　└java.lang. Throwable

是 Java 程序中的所有错误类与异常类的父类。

（20）java.lang.Object

　　└java.lang.Thread

线程类，该类用于线程的创建以及线程控制。

（21）java.lang.Object

　　└java.lang.ThreadGroup

线程组类，用于将多个线程放在一个线程组中进行操作控制。

（22）java.lang.Object

　　└java.lang.Error

Java 应用程序发生的严重错误，不会被捕捉。

（23）java.lang.Object

└java.lang.Exception

Java 应用程序发生的异常，能够被捕捉。

2. java.io 包

该包提供一系列用来读/写文件或其他输入/输出源的输入/输出流。其中有基本输入/输出类、缓冲流类、比特数组与字符串流类、数据流类、文件流类、管道类、流连接类和异常类等。

3. java.util 包

包含一些低级的实用工具类。这些工具类实用方便，而且很重要。主要有：日期类、堆栈类、随机数类、向量类。

4. java.net 包

包含一些与网络相关的类和接口，以方便应用程序在网络上传输信息，分为：主机名解析类、Socket 类、统一资源定位器类、异常类和接口。

5. java.awt 包

提供了 Java 语言中的图形类、组成类、容器类、排列类、几何类、事件类和工具类。

6. java.applet 包

java.applet 是所有小应用程序的父类。它只包含一个 Applet 类，所有小应用程序都是从该类继承的。

7. java.security 包

包含 java.security.acl 和 java.security.interfaces 子类库，利用这些类可对 Java 程序进行加密，设定相应的安全权限等。

8. javax.swing 包

该类库提供一个“轻量级”控件集。所有 swing 控件都是由 Java 程序写成，并且尽可能地实现平台的无关性。该类库中具有完全的用户界面组件集合，是在 AWT 基础上的扩展，因此，对于图形方面的 java.awt 组件，大多数都可以在 javax.swing 类库中找到对应的组件。

想要了解更详细的 Java 类库中的内容，参考 JDK 帮助文档，其中包含 Java API 的所有包、类和接口。

5.5.2 Math 类

Math 类是 java.lang 包的一部分。在 Math 类中提供一些方法以完成常见的数学运算，Math 类常用的方法如表 5-1 所示。

表 5-1 Math 类一些常用方法

方法	功能
public static int abs(int x)	取绝对值
public static long abs(long x)	取绝对值
public static float abs(float x)	取绝对值
public static double abs(double x)	取绝对值

续表

方法	功能
public static double sin(double x)	求正弦
public static double cos(double x)	求余弦
public static double tan(double x)	求正切
public static double asin(double x)	求反正弦
public static double acos(double x)	求反余弦
public static double atan(double x)	求反正切
public static double exp(double x)	求指数
public static double log(double x)	取自然对数
public static double sqrt(double x)	求平方根
public static double toRadius(double x)	将角度值转为弧度值
public static double toDegrees(double x)	将弧度值转为角度值
public static double pow(double x, double y)	x 的 y 次方
public static int max(int x, int y)	x、y 中较大值
public static long max(long x, long y)	x、y 中较大值
public static float max(float x, float y)	x、y 中较大值
public static double max(double x, double y)	x、y 中较大值
public static int min(int x, int y)	x、y 中较小值
public static long min(long x, long y)	x、y 中较小值
public static float min(float x, float y)	x、y 中较小值
public static double min(double x, double y)	x、y 中较小值

5.5.3　Date 类

Date 类在包 java.util 中，表示时间。从 JDK 1.1 开始大多数功能，由 Calendar 类的方法替代，这里仅介绍没有过时的一些方法。

1. *构造方法与创建 Date 类对象*

public Date()：分配 Date 对象并初始化此对象，以表示分配它的时间（精确到毫秒）。

public Date(long date)：分配 Date 对象并初始化此对象，以表示自从标准基准时间（称为“新纪元（epoch）”，即 1970 年 1 月 1 日 00:00:00 GMT）以来的指定毫秒数。

2. *Date 类常用方法*

（1）比较两个日期。

public boolean before(Date when)：测试此日期是否在指定日期之前。当且仅当此 Date 对象表示的瞬间比 when 表示的瞬间早，才返回 true；否则返回 false。

public boolean after(Date when)：测试此日期是否在指定日期之后。 当且仅当此 Date 对象表示的瞬间比 when 表示的瞬间晚，才返回 true；否则返回 false。

public boolean equals(Object obj)：比较两个日期的相等性。当且仅当参数不为 null，并且是一个表示与此对象相同的时间点（到毫秒）的 Date 对象时，结果才为 true。当且仅当 getTime 方法对于两个 Date 对象返回相同的 long 值时，这两个对象才是相等的。

public int compareTo(Date anotherDate)：比较两个日期的顺序。如果参数 Date 等于此 Date，则返回值 0；如果此 Date 在 Date 参数之前，则返回小于 0 的值；如果此 Date 在 Date 参数之后，则返回大于 0 的值。

（2）复制日期。

public Object clone()：返回此对象的副本。

（3）获取时间。

public long getTime()：返回自 1970 年 1 月 1 日 00:00:00 GMT 以来此 Date 对象表示的毫秒数。

（4）设置时间。

public void setTime(long time)：设置此 Date 对象，以表示 1970 年 1 月 1 日 00:00:00 GMT 以后 time 毫秒的时间点。

（5）转换为字符串。

public String toString()：把此 Date 对象转换为以下形式的 String：

dow mon dd hh:mm:ss zzz yyyy

其中：

- dow 是一周中的某一天（Sun，Mon，Tue，Wed，Thu，Fri，Sat）。
- mon 是月份（Jan，Feb，Mar，Apr，May，Jun，Jul，Aug，Sep，Oct，Nov，Dec）。
- dd 是一月中的某一天（01～31），显示为两位十进制数。
- hh 是一天中的小时（00～23），显示为两位十进制数。
- mm 是小时中的分钟（00～59），显示为两位十进制数。
- ss 是分钟中的秒数（00～61），显示为两位十进制数。
- 如果不提供时区信息，则 zzz 为空，即根本不包括任何字符。
- yyyy 是年份，显示为 4 位十进制数。

Date 对象表示时间的默认顺序是星期、月、日、小时、分、秒、年，若要按照希望的格式输入，可以使用 DateFormat 的子类 SimpleDateFormat 来实现日期的格式化。

5.5.4 Calendar 类

Calendar 类在 java.util 包中，它是一个抽象类，它为特定瞬间与一组诸如 YEAR、MONTH、DAY_OF_MONTH、HOUR 等日历字段之间的转换提供了一些方法，并为操作日历字段（例如获得下星期的日期）提供了一些方法。瞬间可用毫秒值来表示，它是距历元（即格林威治标准时间 1970 年 1 月 1 日的 00:00:00.000）的偏移量。

（1）初始化日历对象。Calendar 提供了一个类方法 getInstance，以获得此类型的一个通用的日历对象。

public static Calendar getInstance()：使用默认时区和语言环境获得一个日历。返回的 Calendar 基于当前时间，使用了默认时区和默认语言环境。例如：

```
Calendar rightNow = Calendar.getInstance();
```

初始化一个 Calendar 对象，其日历字段已由当前日期和时间值进行初始化。

（2）获得并设置日历字段值。可以通过调用 set 方法来设置日历字段值。在需要计算时间值（距历元所经过的毫秒）或日历字段值之前，不会解释 Calendar 中的所有字段值设置。调用 get、getTimeInMillis、getTime、add 和 roll 涉及此类计算。

该类为具体日历系统提供了一些常用的字段，有：

- public static final int YEAR：指示特定日历的年。
- public static final int MONTH：指示特定日历的月份，一年中的第一个月是 JANUARY，它为 0；最后一个月取决于一年中的月份数。
- public static final int DAY_OF_MONTH：指示一个月中的某天。它与 DATE 是同义词。一个月中第一天的值为 1。
- public static final int HOUR：指示上午或下午的小时。HOUR 用于 12 小时制时钟 (0 - 11)。中午和午夜用 0 表示，不用 12 表示。例如，在 10:04:15.250 PM 这一时刻，HOUR 为 10。
- public static final int MINUTE：指示一小时中的分钟。例如，在 10:04:15.250 PM 这一时刻，MINUTE 为 4。
- public static final int SECOND：指示一分钟中的秒。例如，在 10:04:15.250 PM 这一时刻，SECOND 为 15。
- public static final int DAY_OF_WEEK：指示一个星期中的某天。该字段可取的值为 SUNDAY、MONDAY、TUESDAY、WEDNESDAY、THURSDAY、FRIDAY 和 SATURDAY。

常用的 set 方法有：

- public void set(int field, int value)：将给定的日历字段设置为给定值。其中参数 field 表示给定的日历字段，value 为给定日历字段所要设置的值。
- public final void set(int year,int month,int date)：设置日历字段 YEAR、MONTH 和 DAY_OF_MONTH 的值。保留其他日历字段以前的值。如果不需要这样做，则先调用clear()。参数 year 用来设置 YEAR 日历字段的值；month 用来设置 MONTH 日历字段的值。Month 值是基于 0 的。例如，0 表示 January；Date 用来设置 DAY_OF_MONTH 日历字段的值。
- public final void set(int year,int month, int date, int hourOfDay, int minute)：设置日历字段 YEAR、MONTH、DAY_OF_MONTH、HOUR_OF_DAY 和 MINUTE 的值。保留其他字段以前的值。参数 year 用来设置 YEAR 日历字段的值；month 用来设置 MONTH 日历字段的值。Month 值是基于 0 的。例如，0 表示 January；date 用来设置 DAY_OF_MONTH 日历字段的值；hourOfDay 用来设置 HOUR_OF_DAY 日历字段的值；minute 用来设置 MINUTE 日历字段的值。
- public final void set(int year,int month,int date,int hourOfDay, int minute, int second)：设置字段 YEAR、MONTH、DAY_OF_MONTH、HOUR、MINUTE 和 SECOND 的值。保留其他字段以前的值。参数 year 用来设置 YEAR 日历字段的值；month 用来设置 MONTH 日历字段的值。Month 值是基于 0 的。例如，0 表示 January；date 用来设

置 DAY_OF_MONTH 日历字段的值；hourOfDay 用来设置 HOUR_OF_DAY 日历字段的值；minute 用来设置 MINUTE 日历字段的值；second 用来设置 SECOND 日历字段的值。

常用的 get 方法有：

- public long getTimeInMillis()：返回此 Calendar 的时间值，以毫秒为单位。
- public int get(int field)：返回给定日历字段的值。

（3）修改日历字段方法。

- public abstract void add(int field,int amount)：根据日历的规则，为给定的日历字段添加或减去指定的时间量。参数 field 为日历字段；amount 表示为字段添加的日期或时间量。

例如，要从当前日历时间减去 5 天，可以通过调用以下方法做到这一点：add(Calendar.DAY_OF_MONTH, -5)。

- public void roll(int field,int amount)：向指定日历字段添加指定（有符号的）时间量，不更改更大的字段。负的时间量意味着向下滚动。

例 5-14 输出当前日期和时间，并计算 1969 年 11 月 6 日到今天 2009 年 4 月 6 日之间共有多少天？

```
import java.io.*;
import java.util.*;
public class CalendarTest{
    public static void main(String args[]){
        Calendar calendar=Calendar.getInstance();
        calendar.setTime(new Date());
        int year=calendar.get(Calendar.YEAR),
        month=calendar.get(Calendar.MONTH)+1,
        day=calendar.get(Calendar.DAY_OF_MONTH),
        week=calendar.get(Calendar.DAY_OF_WEEK)-1,
        hour=calendar.get(Calendar.HOUR),
        minute=calendar.get(Calendar.MINUTE),
        second=calendar.get(Calendar.SECOND);
        System.out.println("现在日期是："+year+"年"+month+"月"+day+"日 星期"+week);
        System.out.println("      时间为："+hour+"时"+minute+"分"+second+"秒");
        calendar.set(1969,11,6);
        long time69116=calendar.getTimeInMillis();
        calendar.set(2009,4,6);
        long time0946=calendar.getTimeInMillis();
        System.out.println("1969 年 11 月 6 日 到 2009 年 4 月 6 日相隔：
        "+(time0946-time69116)/1000/60/60/24+"天");
    }
}
```

程序运行结果为：

```
现在日期是：2009 年 4 月 6 日 星期 1
    时间为：10 时 46 分 44 秒
1969 年 11 月 6 日 到 2009 年 4 月 6 日 相隔：14396 天
```

例 5-15　输出 2009 年 4 月日历。

```
import java.io.*;
import java.util.*;
public class CalendarTest{
    public static void main(String args[]){
        Calendar calendar=Calendar.getInstance();
        System.out.println("------"+calendar.get(Calendar.YEAR)+"年"+
                calendar.get(Calendar.MONTH)+"月------");
        System.out.println("");
        System.out.println(" 日 一 二 三 四 五 六");
        calendar.set(2009,4,6,9,50,22);
        int weekday=calendar.get(Calendar.DAY_OF_WEEK)-1;
        String days[]=new String[weekday+30];
        for(int i=0;i<weekday;i++)
            days[i]=String.valueOf(32-weekday+i);
        for(int i=1;i<=30;i++){
            if(i<10)   days[weekday+i-1]=" "+String.valueOf(i);
            else   days[weekday+i-1]=String.valueOf(i);
         }
         for(int i=0;i<days.length;i++) {
            if(i%7==0) System.out.println("");
            System.out.print(" "+days[i]);
         }
    }
}
```

本章主要介绍了 Java 的重要特性：继承、接口以及多态机制。继承是类之间的一般与特殊关系，子类继承父类，子类获得父类的全部属性和行为，然后在此基础上，可以新增特性。接口是面向对象的一个重要机制，使用接口可以实现多继承的功能。接口中的所有方法都是抽象的，这些抽象方法由实现这一接口的不同类来具体实现。多态性是指不同类型的对象可以响应相同的消息。包是一种将相关类、接口或其他包组织起来的集合体。目的是为了将包含类代码的文件组织起来，易于查找和使用。最后介绍了 Java 的类库、常见的包和主要的类。

一、选择题

1．Java 语言的类间的继承关系是（　　）。

A．多继承　　B．单继承　　C．线程的　　D．不能继承

2．定义包使用的关键字是（　　）。

A．import　　B．implements　　C．package　　D．extends

3．以下（　　）是接口的正确定义。

A．
```
interface  B1{
    void print(){} ;
    }
```

B．
```
abstract  interface  B1{
    void print() ;
    }
```

C．
```
abstract interface B1 extends  A1,A2 {
   /A1、A2 为已定义的接口
   abstract  void  print(){  };
    }
```

D．
```
interface  B1 {
    void  print();
    }
```

4．下列叙述中，错误的是（　　）。

A．父类不能替代子类　　B．子类能够替代父类

C．子类继承父类　　D．父类包含子类

5．下列类定义中（　　）是合法的抽象类定义。

A．class A{abstract void fun1();}

B．abstract A{ abstract void fun1();}

C．class abstract A{abstract void fun1();}

D．abstract class A{ abstract void fun1();}

E．abstract class A{ abstract void fun1(){};}

F．abstract class A{void fun1(){};}

6．定义接口使用的关键字是（　　）。

A．public　　B．implements　　C．package　　D．interface

7．如果在子类中需要调用父类带参数的构造方法，可通过 super 调用所需的父类构造方法，且该语句必须作为父类构造方法中的（　　）。

A．第一条语句　　B．第二条语句

C．倒数第二条语句　　D．最后一条语句

8．在子类中重写父类的方法的过程称为（　　）。

A．方法重载　　B．方法重用

C．方法覆盖　　D．方法继承

二、填空题

1．_____是一种软件重用形式，在这种形式中，新类获得现有类的数据和方法，并可增加新的功能。

2．Java 程序中定义接口所使用的关键字是_____，接口中的属性都是_____，接口中的方法都是_____。

3．_____是 Java 程序中所有类的直接或间接父类，也是类库中所有类的父类。

4．Java 程序引入接口的概念，是为了弥补只允许类的_____的缺憾。

5．在 Java 程序中，把关键字_____加到方法名称的前面，来实现子类调用父类的方法。

6．Java 语言通过接口支持_____继承，使类继承具有更灵活的扩展性。

7．接口是一种只含有抽象方法或_____的一种特殊抽象类。

8．接口中的成员只有静态常量和_____。

三、简答题

1．什么是继承？什么是基类和子类？
2．什么是数据成员的隐藏？什么是方法的重写？
3．基类对象与子类对象相互转换的条件是什么？如何实现它们之间的相互转换？
4．什么是接口？接口的功能是什么？接口与类有何异同？
5．如何定义接口？使用什么关键字？
6．一个类如何实现接口？实现接口的类是否一定要重写该接口中的所有抽象方法？
7．什么是包？如何创建包？如何引用包中的类？

四、编程题

1．编写一个完整的 Java Application 程序，包括 ShapeArea 接口、MyTriangle 类、Test 类，具体要求如下：

（1）接口 ShapeArea：

double getArea()：求一个形状的面积。

double getPerimeter ()：求一个形状的周长。

（2）类 MyTriangle：实现 ShapeArea 接口，另有以下属性和方法：

1）属性。

x,y,z：double 型，表示三角形的三条边。

s：周长的 1/2（注：求三角形面积公式为 $\sqrt{s(s-x)(s-y)(s-z)}$，s=(x+y+z)/2，开方可用 Math.sqrt(double)方法）。

2）方法。

MyTriangle(double x, double y, double z)：构造函数，给三条边和 s 赋初值。

toString()：输出矩形的描述信息，如“three sides:6.0,8.0,10.0,perimeter=24.0,area=24.0”。

（3）Test 类作为主类要完成测试功能：

1）生成 MyTriangle 对象。

2）调用对象的 toString 方法，输出对象的描述信息。

2．定义接口 Shape 及其抽象方法 getArea()和 getPerimeter()用于计算图形的面积和周长。定义类 Rectangle（矩形）、类 Circle（圆形）和类 Triangle（三角形）继承类 Coordinates（点）并实现接口的抽象方法。

3．编写一个程序，用来根据租用汽车的大小来支付一天的租金：汽车的规格有：大、中、小及实际大小。添加一个需要汽车大小的构造方法。添加一个用于添加汽车电话选项的子类。编写一个使用这些类的程序。

4．编写一个程序，用来计算某项物品的价格。添加一个需要数量、物品名称和物品的构造方法。添加一个子类以便根据定购的数量提供折扣。编写一个使用这些类的程序。

5．编写一个命名为 UserLoan 的程序，该程序使用命名为 Loan 的抽象类和多个子类，用来显示不同类型的贷款和每月的花费（家庭、汽车等项）。在每个具有合适参数的类中使用构

造方法。添加获取和设置方法，其中至少有一个方法是抽象的。提示用户输入显示的类型，然后创建合适的对象。还要创建一个接口，该接口至少有一个用于子类的方法。

6．设计程序要求实现：

（1）抽象类的定义。

（2）定义子类并继承抽象类，实现其抽象方法。

（3）终极类的使用。

（4）静态成员的使用。