第5章 货 物 管 理

【本章概述】本章以项目为导向，深入介绍了关于声明类的细节问题，包括作用域问题、属性隐藏问题、访问控制问题等。通过本章的学习，读者能够理解属性与变量的区别，掌握软件包的用法，理解 this 关键字的作用，掌握访问控制修饰符和 static 修饰符的用法，能够熟练使用 String 类和 ArrayList 类的常用方法。

【教学重点】变量的作用域、访问控制、this。

【教学难点】this、static 修饰符。

【学习指导建议】学习者应首先通过学习【技术准备】，了解 Java 语言中关于声明类的细节问题。通过学习本章的【项目学做】完成本章的项目，理解和掌握变量作用域、访问控制修饰符、this 关键字的作用和用法。通过【强化训练】巩固对本章知识的理解。最后通过【课后习题】进行学习效果测评，检验学习效果。

5.1 项 目 任 务

该项目要编程实现一个货物管理的简单功能：能描述货物的编号、名称，显示某特定货物的相关信息；能描述货物的类别，显示该类别的所有货物信息以及为该类别添加一个货物；能根据货物名称查找货物，将某个货物或某些货物添加到它所隶属的类别里，并显示该系统中所有的货物类别及其该类别中的所有货物信息。

5.2 项 目 分 析

1．项目完成思路

根据项目任务描述的项目功能需求，本项目需要定义 4 个类，具体可以按照以下过程实现。

(1) 货物类，描述货物的名称、编号(要求能够自动生成，无须外界赋值实现)。这个类有带参的构造方法和无参的构造方法，同时还要提供一个显示该货物所有信息的方法。

(2) 货物类别类，描述该类货物的类别名称和该类货物的成员编号。除了构造方法、访问器方法、显示该类所有信息的方法外，还要定义一个为该类添加成员的方法。

(3) 货物管理类。该类可以按类别存放管理的所有货物信息，为方便使用应提供如下功能：

① 可以根据货物名称进行查找，如果找到，返回该类别对象的一个引用，否则返回空；

② 可以增加一个货物，首先进行查找，如果此货物类别已经存在，则只是在已有的货物成员中添加该货物的编号，否则需要创建该货物所隶属的货物类别类对象，并将该对象添加到货物列表中；

③ 可以增加一组货物；

④ 显示此系统中的所有货物类别的详细信息。

(4) 货物管理测试类，测试货物管理类的各项功能。

2．需解决的问题

(1) 这 4 个类为了方便相互引用应如何组织?

(2) 如何实现货物编号的自动生成?

(3) 如何为一个类定义多个构造方法?

(4) 如何保存特定类别中的所有货物信息?

(5) 如何遍历并显示输出特定货物类别中的所有货物?

(6) 如何在特定的货物类别中根据货物名称查找特定货物?

(7) 如何将一组货物加入到特定类别?

解决以上问题涉及的技术将在 5.3 节中详细阐述。

5.3 技 术 准 备

5.3.1 构造方法的重载

前面介绍了重载的概念,就是在一个类定义中出现多个方法名相同参数列表不同的多个同名方法共存的现象，作为方法的一个特例——构造方法也可以被重载，请看下面示例代码。

1．示例代码

【例 5-1】带有构造方法重载的类定义。

```
// StuTest.java
class Student
{
    //定义属性：姓名 性别
    String name;
    char  gender;
    //构造方法 1
    public Student(String n,char g)
    {
        name=n;
        gender=g;
    }
    //构造方法 2
    public Student(String n)
    {
        name=n;
        gender='m';
    }
    //构造方法 3
    public Student()
    {
        gender='m';
    }
    //将属性以字符串的形式返回
    public String output()
    {
        String s=null;
```

```
            s="姓名："+name;
            if(gender=='f')
                s=s+"  性别：女";
            else
                s=s+"  性别：男";
            return s;
        }
    }
    public class StuTest
    {
        public static void main(String argv[])
        {
            //调用带一个参数的构造函数创建对象
            Student s1=new Student("张三");
            System.out.println(s1.output());
            //调用带两个参数的构造函数创建对象
            Student s2=new Student("李四",'f');
            System.out.println(s2.output());
        }
    }
```

编译运行得到如下运行结果：

```
姓名：张三  性别：男
姓名：李四  性别：女
```

2．代码分析

在例 5-1 中展示了构造方法的重载，在 Student 类中共定义了 3 个构造方法，分别为不带参数、带一个参数和带两个参数。其中，不带参数的构造方法只是完成将性别赋值为男性，而姓名采用属性的默认值，因为姓名的数据类型是 String，所以默认值是 null。带一个参数的构造方法完成为姓名属性赋值和为性别属性赋值，只是这里的性别不是通过参数传递进来的而是默认设定为男性(male)。而在带两个参数的构造方法中，姓名和性别的值都是通过参数传递进来的，这是它和带一个参数构造方法的区别。

3．知识点

构造方法的重载是指在同一个类中存在着若干个具有不同参数列表的构造方法。

有时在一个类定义内可能出现多个实现类似功能的方法，但这些方法处理的信息有所不同，因为功能类似但给不同的命名容易造成理解上的混乱，所以就要给这些方法相同的名字，也即在一个类中出现多个同名方法的现象，这叫方法的重载。

4．举一反三

使用方法重载编写程序，分别求：两个整型数据之和；两个浮点型数据之和。方法名均为 add。

5.3.2 变量的作用域

在 4.3.1 节提到：属性的声明可以放在类体内的任何位置，且在使用属性前没有给属性赋值时系统会根据其类型为其提供一个默认值，那么这个规则是否也适用于局部变量呢？请看例 5-2。

1．示例代码

【例 5-2】局部变量和全局属性的不同作用域。

```
//Scope.java
class Scope
{
    void fun1()
    {
        //定义局部变量 x
        int x=1;
        for(int i=0;i<5;i++)
        {//定义局部变量 sum
            int sum=0;
            sum+=i;
        }
        System.out.println("x="+x);
        System.out.println("y="+y);
        System.out.println("sum ="+ sum);
        System.out.println("s="+s);
        //定义局部变量 x
        int y;
    }
    //定义属性
    int x;
    String s="global variable";
}
```

编译运行得到如下错误提示：

```
Cannot find symbol variable y
Cannot find symbol variable sum
```

2．代码分析

编译过程报的两个错误定位在 12 行和 13 行，都是一个错误“找不到变量符号”。因为这里的 y 是局部变量，而它的声明出现在使用之后，所以系统报告找不到变量。但另一个局部变量 sum 的声明已经出现在使用之前，为什么也报找不到变量呢？原因在于：sum 声明是在 for 循环块内的，而使用是在块外即超出了它的作用域范围(for 循环的左右花括号)。下面修改一下代码，如例 5-3 所示，看编译是否能通过。

【例 5-3】修订例 5-2。

```
class Scope1
{   void fun1()
    {
        //定义局部变量 x
        int x=1;
        int y;//添加一段代码！！
        int sum=0;//修改的代码！！
        for(int i=0;i<5;i++)
        {   sum+=i; }
            System.out.println("x="+x);
            System.out.println("y="+y);//添加一段代码！！
```

```
            System.out.println("sum ="+ sum);
            System.out.println("s="+s);
        }
        int x; //定义属性
        String s="global variable";
        public static void main(String []argv)
        {   Scope1  obj=new Scope1();
            obj.fun1();
        }
    }
}
```

编译例 5-3，又有错误提示:

```
The local variable may not have been initialized
```

说明局部变量在使用前必须初始化赋值。将 y 值设为 0，运行此程序得到如下的输出结果：

```
x=1
y=0
sum=10
s=global variable
```

这里 x=1 而不是 x=0，原因是 x 在 fun1 方法中又声明了一次，初始值为 1，也即在 fun1 方法体内起作用的是重新声明的 x=1 变量而非属性 x。

3．知识点

- 变量的作用域是指程序的一部分，在这部分中，变量可以被引用。
- 属性不管在何处声明，其作用域范围都是整个类。
- 局部变量必须先声明再使用，在使用前必须给定初始值，局部变量的作用域范围是从定义的那一点到距离它最近的反花括号之间。
- 当局部变量和属性重名时，在局部变量的作用域内局部变量屏蔽掉属性。

4．举一反三

请给下述代码查找错误。

```
public class Test1
{
    public static void main(String argv[])
    {   int y=99;
        a();
        b();
        System.out.println("in function main+"+y);
    }
    public void a()
    {
        int x=1;
        System.out.println("in function a+"+x);
    }
    public void b()
    {
         System.out.println("in function b+"+y);
    }
}
```

5.3.3 this 关键字

在 5.3.2 节的知识点中我们提到，当局部变量和属性重名时，在局部变量的作用域内局部变量屏蔽掉属性。而在类中定义方法时我们又知道：方法的形式参数可以随意命名，那么如果在一个类的构造方法中定义的形式参数恰好和属性重名会有什么事情发生呢？

1．示例代码

【例 5-4】形式参数和类的属性重名时会发生的问题。

```
//RectangleTest3.java
public class RectangleTest3
{
           public static void main(String[] args)
           {
              Rectangle  obj=new Rectangle (2,3);
              System.out.println("矩形的长和宽是：  "+obj.length+","+obj.width);
              int result=obj.area();
              System.out.println("矩形面积是:"+result);
           }

}
class Rectangle
{
    int length=1;
    int width=1;
    public int area()
    {
        int temp=length*width;
        return temp;
    }
    /*修改的代码！！将形参修改成和属性同名*/
    public Rectangle(int length,int width)
    {
        length=length;
        width=width;
    }
    public Rectangle()
    {
        length=5;
        width=5;
    }
}
```

2．代码分析

首先编译运行此程序，得到如下运行结果：

```
矩形的长和宽是：1,1
矩形面积是：1
```

为什么实例化对象时的实际参数 2 和 3 没有被对象 obj 得到呢？问题就出在形式参数和属性重名。根据 5.3.2 节知识点提到的：当局部变量和全局属性重名时局部变量将屏蔽全局属性，

例 5-4 中的形式参数是局部于构造方法的局部变量，它的作用域范围就是该带参的构造方法，在它的作用域范围内全局性变量属性 length 和 width 被屏蔽掉，所以这个带参的构造方法中赋值符号左右出现的 length 和 width 都是指形参局部变量，和全局属性没有任何关系，自然 obj 对象的属性得不到 2 和 3。那么如果形参一定要和属性重名又想让属性得到指定的值该怎么办呢？我们把带参的构造方法修改如下即可编译通过：

```
public Rectangle(int length,int width)
{
    this.length=length;
    this.width=width;
}
```

这里出现了一个新的关键字 this，this 是 Java 系统默认的为每一个类都提供的一个关键字，该关键字代表了当前类对象的一个引用(引用可以理解为是别名)。因为该构造方法的形参变量的名字和待赋值属性的名字相同，为了能够区分二者中哪一个是属性，在属性前加上 this.，即利用 this.可以调用当前对象的属性和方法。其实在类内调用本类的属性和方法时，系统都默认在属性和方法前加上了 this.。我们知道一个类中的方法可以互相调用，那么如果一个类中出现了多个构造方法定义即构造方法的重载，我们如何在多个构造方法间相互调用？这就涉及了 this 的第二个用法。如例 5-4 中，我们可以将无参的构造方法改成如下形式：

```
public Rectangle()
{
    this(5,5);
}
```

这里 this(5,5)代表调用本类带两个参数的构造方法，可见我们可以使用 this(实参)调用本类的其他构造方法，但该语句出现的位置要求是所隶属的构造方法的第一条可执行语句。this 还可以作为方法的返回值，见例 5-5。

【例 5-5】this 作为方法的返回值。

```
class Rectangle
{
    int length=1;
    int width=1;
    public int area()
    {
        int temp=length*width;
        return temp;
    }
    public Rectangle(int length,int width)
    {
        this.length=length;
        this.width=width;
    }
    //新添加代码！！找到两个 Rectangle 对象中面积大的矩形对象并将该对象返回
    public Rectangle larger(Rectangle r)
    {
        int area1=this.area();
        int area2=r.area();
        if(area1>area2)
```

```
            return this;
        else
            return r;
    }
}
```

这段新添加的方法的功能是比较两个 Rectangle 对象，并返回面积较大的那个对象。那么首先明确要比较的两个对象是谁？形式参数是一个，另外一个这里用 this 来表示。因为未来在调用该 larger 方法时一定是以对象名.larger(实参)的形式调用的，但当我们在定义类时还不知道对象是谁，只好用该对象的引用 this 来代替，所以方法的返回值有两种可能，一是参数 r，另一个就是未来要涉及的对象即现在的 this，这是 this 的第三个用法。

3．知识点

this 是 Java 系统默认的为每一个类都提供的一个关键字，该关键字代表了当前类对象的一个引用。this 的用法有如下 3 条：

① 利用 this.可以调用当前对象的方法或属性。

② 一个类的若干个构造方法之间可以互相调用，当一个构造方法需要调用另一个构造方法时应使用 this(实参列表)，同时这条调用语句应该是整个构造方法中的第一条可执行语句。在利用 this 调用构造方法时，根据实参的个数匹配调用的是哪个其他的构造方法。

③ 当方法需返回当前正在讨论的对象时，可以采用 return this 的形式。

4．举一反三

定义一个圆类 Circle，该类有整型属性 radius，有带参(要求形参和属性同名)和无参的构造方法(要求在无参的构造方法中使用 this(实参)调用带参的构造方法)，有计算圆面积的方法 area，有比较圆大小的 compare 方法(半径大的圆即是要返回的圆)，再编写一个测试类测试。

5.3.4 包

利用面向对象技术进行实际项目开发时，通常需要设计许多类共同工作，为了更好地管理这些类，Java 中引入了包的概念。那么如何定义包？包又是如何组织类的？存放到特定包中的类如何被其他类引用？见例 5-6。

1．示例代码

【例 5-6】定义一个包。

```
//A.java
//定义包，包名为 first
package first;
//定义类 A
public class A
{
    //定义属性,字符串类型
    String  a;
    //定义方法
    public void setA(String x)
    {
        a=x;
    }
    public String getA()
```

```
    {
        return a;
    }
    public void output()
    {
        System.out.println("a 的值为："+a);
    }
}
```

编译运行后，到原文件保存的路径中查看，会发现在当前路径下生成了一个名为 first 的文件夹，则此文件夹对应的就是源码中定义的包 first，打开此文件夹发现包含 A.class 文件。

例 5-6 定义了一个包，则该包中的被 public 修饰的类 A 原则上就能被其他类引用，那么如何引用？请看例 5-7。

【例 5-7】引用其他包中的公有类。

```
//  PackageTest.java
//定义包，包名为 second
package second;
//导入包 first 中的类 A
import first.*;
//定义测试类
public  class PackageTest
{
    public static void main(String argv[])
    {
        //声明、创建 A 的对象 object
        A object=new A();
        //调用 set 方法给对象 object 的属性 a 赋值
        object.setA("hello world");
        //调用 output 方法，输出对象 object 的属性 a 值
        object.output();
    }
}
```

2．代码分析

例 5-6 和例 5-7 代码中都利用 package 关键字定义了包，这样的包称为有名包。包名的定义要遵循 Java 语言定义标识符的规定，同时定义包的语句应是.java 源文件中的第一条可执行语句。如果要引用有名包中定义的类，则使用 import 包名.类名，如例 5-6 中的 import first.A，也可以把该包中的所有类都导入，如 import first.*，效果相同。在例 5-6 中又出现了一种新的数据类型——字符串类型(String)，该类型是系统定义的一个类，但此类可以直接使用，无须引入任何包。

本节之前的内容尽管并没有使用 package 关键字定义包，系统也为每一个.java 文件创建了一个无名包，该文件中定义的所有类都属于这个无名包，无名包中的所有类都可以互相引用。但无名包中的类是不能被本节所述的有名包中的类引用的。

如果两个包中包含相同的类名，且这两个包又被一个程序同时引用，那么如何区分这个同名类呢？例如，在包 aa.bb 中包含一个 Base 类，而在 rt.ql 中也包含一个 Base 类，在某一个程序中又都同时引入这两个包，即出现了如下语句：

```
import aa.bb.*;
import rt.q1.*;
```

那么此时如果想创建 rt.ql 包中 Base 类的对象，如何确定？假定该包中的类有默认的构造方法，此时为了创建该类对象则必须写上所指类所隶属的完整路径，即如下所示：

```
rt.q1.Base  对象名=new  rt.q1.Base();
```

3．知识点

① 定义包的语法结构：package　包名;

例如：package　ch;

【注意】

- 定义包的语句应是.java 源文件中的第一条可执行语句。
- 无名包中的类不能被有名包中的类引用，而有名包中的类可以被无名包中的类引用。

② 引入包中类的语法结构：import　包名.类名;　或　import　包名.* ;

例如：import　ch.A;　　import　ch.*;

③ 如果在一个程序中涉及两个包中的同名类时，则创建此类对象时需明确指出该类所属的包，如下所示：

包名.类名 对象名=new　包名.类名.构造方法

4．举一反三

下面仿照例 5-6 和例 5-7 来完成一道练习，题目描述如下。

第一个 java 源文件在包 ch.jyfs 下定义一个类 B，类中有一个 String 型的属性 b，该类有 show 方法，用来输出属性 b 的值。

第二个 java 源文件定义一个类 B 的测试类 B_Test，该类只有 main 方法，在 main 方法中完成声明，创建类 B 的对象，为其属性 b 赋值，然后调用 show 方法输出创建对象的属性 b 的值。完成此程序，注意观察第一个 java 文件编译后生成了哪些文件夹、这些文件夹之间的关系如何、各自代表什么、B.class 存放在哪里。

5.3.5　访问控制修饰符

在介绍前述内容时涉及一些访问控制修饰符，如修饰类的 public 访问控制修饰符、修饰类的默认访问控制修饰符、修饰属性的默认访问控制修饰符、修饰方法的 public 访问控制修饰符等，那么这些访问控制修饰符对被修饰的类、属性或方法有怎样的限制作用？修饰类、属性、方法分别有哪些访问控制修饰符？将是本节介绍的内容。

1．类的访问控制修饰符

(1) 示例代码

【例 5-8】公有类可以被其他包中的类引用。

```
//代码第一部分 AA.java
//在包 ch.jp.exam1 中定义一个 public 类 AA，类的内容为空
package ch.jp.exam1;
public class  AA
{}
//代码第二部分 AATest.Java
//在无名包中定义一个测试类 AATest
//导入包 ch.jp.exam1 中的类 AA
import ch.jp.exam1.AA;
```

```
public class AATest
{
    public static void main(String args[])
    {
        AA a=new AA();
    }
}
```

(2) 代码分析

本例中分别在两个包中定义了两个类，那么在进行调试时就要分别放到两个源文件中。在包 ch.jp.exam1 中定义了一个 public 修饰的类 AA，在无名包中定义了一个测试类 AATest，经测试发现 public 修饰的类 AA 能在其他包(无名包)中被其他类(类 AATest)引用。

下面对例 5-8 做第一次修改，将 AA 类前的 public 修饰符去掉，其他地方不变，如下所示：

```
package ch.jp.exam1;
class  AA  //修改的地方
{}
```

保存后，先编译 AA.java，编译通过。然后再编译 AATest，会出现错误提示：

```
cannot Access AA
```

当把类 AA 前的 public 去掉后，此时说明类 AA 是被默认的访问控制修饰符修饰的，这样的类是不能被其他包中的类访问的。

再对例 5-8 做第二次修改，将类 AA 和测试类 AATest 都放到一个包中，也放到一个.java 源文件中，如下所示：

```
package ch.jp.exam1;
public class  AA
{}
public class AATest
{
    public static void main(String args[])
    {
        AA a=new AA();
    }
}
```

此时应以 AATest 作为 java 源文件的名字，编译时出现如下错误提示：

```
Class AA is public, should be declared in a file named AA.java
```

说明在一个 java 源文件中只能有一个类能被声明为 public 的，且源文件的名字必须和声明为 public 的类的类名相同。

再对例 5-8 做第三次修改，将类 AA 和测试类 AATest 都放到一个包中，也放到一个.java 源文件中，但去掉类 AA 前的 public 修饰符，如下所示：

```
package ch.jp.exam1;
class  AA
{}
public class AATest
{
    public static void main(String args[])
    {
        AA a=new AA();
```

```
    }
}
```

此时应以 AATest 作为 java 源文件的名字，编译通过。此次修改说明当在同一个包中时，被默认的访问控制修饰符修饰的类可以被其他类访问。

再对例 5-8 做第四次修改，将类 AA 和测试类 AATest 都放到一个包中，但存储到两个.java 源文件中，如下所示：

```
//AA.java 源码
package ch.jp.exam1;
public class  AA
{}
// AATest.java 源码
package ch.jp.exam1;
public class AATest
{
    public static void main(String args[])
    {
        AA a=new AA();
    }
}
```

分别以 AA 和 AATest 作为 java 源文件的名字，先编译 AA.java，通过后再编译 AATest.java 也通过。此次修改说明当在同一个包中时，被 public 访问控制修饰符修饰的类可以被其他类访问。

(3) 知识点

通过例 5-8 及对其做过的 4 次修改，总结一下类的访问控制修饰符有哪些，当特定的类被这些访问控制修饰符修饰时，其他类对该类的访问是否被允许。

修饰类的访问控制修饰符有 public 和默认(什么都不写)两种，被这样两种访问控制修饰符修饰的类被其他类访问的关系如表 5-1 所示。

表 5-1　被不同修饰符修饰的类与被访问的关系

	同一包中的其他类	不同包中的其他类
被 public 修饰的类	允许	允许
被默认修饰符修饰的类	允许	不允许

2. 属性和方法的访问控制修饰符

(1)示例代码

【例 5-9】可被其他包中的类引用的属性和方法。

```
//代码第一部分
//在包 ch.jp.exam2 中定义一个 public 类 AA
package ch.jp.exam2;
public class  AA
{
    //私有属性
    private    int x;
    //公有属性
    public     double y;
```

```
    //保护属性
    protected  char  z;
    //默认属性
    String w;
    //public 修饰的构造方法
    public  AA(int a, double b,char c,String d)
    {
        x=a;
        y=b;
        z=c;
        w=d;
    }
    // private 修饰的访问器
    private int getX()
    {
        return x;
    }
    // public 修饰的访问器方法
    public double getY()
    {
        return y;
    }
    //protected 修饰的访问器方法
     protected  char getZ()
    {
        return z;
    }
    //默认修饰符修饰的访问器方法
    String getW()
    {
        return w;
    }
}
//代码第二部分
//在无名包中定义一个测试类 ATest
import ch.jp.exam2.AA;
public class ATest
{
    public static void main(String args[])
    {
        AA obj=new AA(1,1.11,'q',"hello");
        /*//语句组 1
        obj.x=2;
        obj.getX(); */
        /*//语句组 2
        obj.y=2.22;
        obj.getY(); */
        /*//语句组 3
```

```
        obj.z='e';
        obj.getZ(); */
        /*//语句组 4
        bj.w="lala";
        obj.getW(); */
    }
}
```

(2) 代码分析

① 类 AA 被 public 修饰，类 ATest 与其不在同一包中。

例 5-9 中在包 ch.jp.exam2 中定义了一个 public 修饰的 AA 类，在无名包中定义了一个 public 修饰的 ATest 类。其中，AA 类中有被不同的访问控制修饰符修饰的属性和方法，下面对这个例子进行修改来总结被不同修饰符修饰的属性和方法是否能被其他包中的类访问。

修改 1：去掉语句组 1 的注释，main 方法体如下所示：

```
public static void main(String args[])
{
    AA obj=new AA(1,1.11,'q',"hello");
    //语句组 1
    obj.x=2;
    obj.getX();
    /*//语句组 2
    obj.y=2.22;
    obj.getY(); */
    /*//语句组 3
    obj.z='e';
    obj.getZ(); */
    /*//语句组 4
    obj.w="lala";
    obj.getW(); */
}
```

编译得到如下错误提示：

```
getX() has private access in ch.exam2.AA
```

说明 public 类 AA 中的被 private 修饰的属性和方法在其他包中是不能通过 AA 类的对象 obj 来调用的。

修改 2：为语句组 1 重新加上注释，然后去掉语句组 2 的注释，main 方法体如下所示：

```
public static void main(String args[])
{
    AA obj=new AA(1,1.11,'q',"hello");
    /*//语句组 1
    obj.x=2;
    obj.getX();*/
    //语句组 2
    obj.y=2.22;
    obj.getY();
    /*//语句组 3
    obj.z='e';
    obj.getZ(); */
```

```
    /*//语句组 4
    obj.w="lala";
    obj.getW(); */
}
```

经编译通过，说明 public 类 AA 中的被 public 修饰的属性和方法在其他包中能通过 AA 类的对象 obj 来调用。

修改 3：为语句组 2 重新加上注释，然后去掉语句组 3 的注释，main 方法体如下所示：

```
public static void main(String args[])
{
    AA obj=new AA(1,1.11,'q',"hello");
    /*//语句组 1
    obj.x=2;
    obj.getX();*/
    /*//语句组 2
    obj.y=2.22;
    obj.getY();*/
    //语句组 3
    obj.z='e';
    obj.getZ();
    /*//语句组 4
    obj.w="lala";
    obj.getW(); */
}
```

编译得到如下错误提示：

```
getX( ) has protected access in ch.exam2.AA
```

说明 public 类 AA 中的被 protected 修饰的属性和方法在其他包中是不能通过 AA 类的对象 obj 来调用的。

修改 4：为语句组 3 重新加上注释，然后去掉语句组 4 的注释，main 方法体如下所示：

```
public static void main(String args[])
{
    AA obj=new AA(1,1.11,'q',"hello");
    /*//语句组 1
    obj.x=2;
    obj.getX();*/
    /*//语句组 2
    obj.y=2.22;
    obj.getY();*/
    /*//语句组 3
    bj.z='e';
    obj.getZ();*/
    //语句组 4
    obj.w="lala";
    obj.getW();
}
```

编译得到如下错误提示：

```
getY( ) is not public in ch.exam2.AA: cannot be accessed from outside package.
```

说明 public 类 AA 中的被默认修饰符修饰的属性和方法在其他包中是不能通过 AA 类的对象 obj 来调用的。

② 类 AA 被 public 修饰，类 ATest 与其在同一包中。

上面所做的测试都是在类 AA 和类 ATest 不在同一个包中的前提下完成的，即上述所做的修改都是在 ATest 类中的 main 方法中完成的。下面看看当 ATest 类和 AA 类放在一个包中时，不同的访问控制修饰符所修饰的属性和方法在访问时又有怎样的情况，即保持例 5-9 代码的第一部分，将例 5-9 中代码的第二部分修改成如下所示：

```
package ch.jp.exam2; //修改的语句
public class ATest
{
    public static void main(String args[])
    {
        AA obj=new AA(1,1.11,'q',"hello");
        /*//语句组 1
        obj.x=2;
        obj.getX(); */
        /*//语句组 2
        obj.y=2.22;
        obj.getY(); */
        /*//语句组 3
        obj.z='e';
        obj.getZ(); */
        /*//语句组 4
        obj.w="lala";
        obj.getW(); */
    }
}
```

编译通过后，重做上述 4 个修改，发现只有语句组 1 编译通不过，而其他 3 组均能编译通过。说明当两个类在同一包中时，被 public 所修饰的类 AA 中只有被 private 修饰的属性和方法不能被另外一个类 ATest 类中创建的 AA 类对象 obj 访问，而被 public、protected 和默认修饰符修饰的属性和方法均能被另外一个类 ATest 类中创建的 AA 类对象 obj 访问。

③ 类 AA 被默认修饰符修饰，类 ATest 与其不在同一包中。

上面介绍了当类 AA 被 public 修饰符修饰时其被不同访问控制修饰符修饰的属性和方法，在同一包中和不在同一包中被 AA 类对象 obj 访问的情况，下面来看当类 AA 被默认的访问控制修饰符修饰时的情况。

保持例 5-9 代码的第二部分不动，对例 5-9 代码的第一部分做修改，去掉类 AA 前的 public 修饰符，如下所示：

```
//代码第一部分
//在包 ch.jp.exam1 中定义一个类 AA
package ch.jp.exam2;
//修改的语句
class  AA
{
    //分别定义私有、公有、保护、默认修饰的属性
    private    int x;
```

```
    public    double y;
    protected  char  z;
    String w;
    public  AA(int a, double b,char c,String d)
    {
        x=a;
        y=b;
        z=c;
        w=d;
    }
    //分别定义私有、公有、保护、默认修饰的方法
    private int getX()
    {
        return x;
    }
    public double getY()
    {
        return y;
    }
    protected  char getZ()
    {
        return z;
    }
    String getW()
    {
        return w;
    }
}
```

编译通过后，对例 5-9 代码的第二部分重做“修改 1”到“修改 4”，发现 4 组修改在编译时全都通不过，说明当类 AA 被默认的访问控制符修饰时，不管其属性和方法的访问控制修饰符如何，均不能被其他包中的类所访问。

④ 类 AA 被默认修饰符修饰，类 ATest 与其在同一包中。

例 5-9 代码的第一部分和第二部分都要做修改，如下所示：

```
//代码第一部分
//在包 ch.jp.exam2 中定义一个类 AA
package ch.jp.exam2;
class  AA  //修改的语句
{
    //私有属性
    private   int x;
    //公有属性
    public    double y;
    //保护属性
    protected  char  z;
    //默认属性
    String w;
    //public 修饰的构造方法
```

```
    public  AA(int a, double b,char c,String d)
    {
        x=a;
        y=b;
        z=c;
        w=d;
    }
    // private 修饰的方法
    private int getX()
    {
        return x;
    }
    // public 修饰的方法
    public double getY()
    {
        return y;
    }
    //protected 修饰的方法
    protected  char getZ()
    {
        return z;
    }
    //默认修饰符修饰的方法
    String getW()
    {
        return w;
    }
}
//代码第二部分
//在无名包中定义一个测试类 ATest
//修改的语句
package ch.jp.exam2;
public class ATest
{
    Public static void main(String args[])
    {
    AA obj=new AA(1,1.11,'q',"hello");
    /*//语句组 1
    obj.x=2;
    obj.getX(); */
    /*//语句组 2
    obj.y=2.22;
    obj.getY(); */
    /*//语句组 3
    obj.z='e';
    obj.getZ(); */
    /*//语句组 4
    obj.w="lala";
```