

第四章 类与对象

本章内容

1. 类与对象的概念
2. 封装机制
3. 数据成员
4. 成员方法

1 类与对象的概念

● 客观世界

由有形的、无形的事物构成的，程序设计的问题域。

● 对象

描述客观世界的基本单位。

● 类

抽象、归纳出对象的共同性。

1.1 抽象原则

抽象概念：

从被研究对象中舍弃个别的、非本质的、或与研究主旨无关的次要特征，

抽取与研究工作有关的实质性内容加以考察，
形成对所研究问题正确的、简明扼要的认识。



1.2 对象

- 程序设计所面对的问题域—客观世界。客观世界的每个事物都具有自己的静态特征、动态特征。
- 把客观世界的事实映射到面向对象的程序设计中，则把事物抽象成为对象。



● 对象具有以下特征：

1. **对象标识**：即对象的名字，是用户和系统识别它的唯一标志。

外部标识供对象的定义者或使用者的使用，

内部标识供系统内部唯一地识别每一个对象。

2.属性：即一组数据，用来描述对象的静态特征。

3.方法：也称为服务或操作，它是对象动态特征(行为)的描述。每一个方法确定对象的一种行为或功能。

在Java程序中，类是创建对象的模板，对象是类的实例，任何一个对象都是隶属于某个类的。

1.3 类

- 抽象与当前目标有关的本质特征，找出事物共性，把具有共性事物归为一类，得出抽象概念一类。
- 在面向对象的编程语言中，类是一个独立的程序单位，是具有相同属性和方法的一组对象的集合。类的概念使我们能对属于该类的全部对象进行统一的描述。

● 在定义对象之前应先定义类。描述一个类需要指明下述三个方面内容：

(1) 类标识：类的一个有别于其他类的名字。

(2) 属性说明：用来描述相同对象的静态特征。

(3) 方法说明：用来描述相同对象的动态特征。

【例4-1】对dog类进行的描述

`class dog` // `class`指出这是一个类, `dog`是类标识

{

String name;

int AverageWeight;

int AverageHeight;

public void move()

{ }

public void ShowDog()

{ }

}

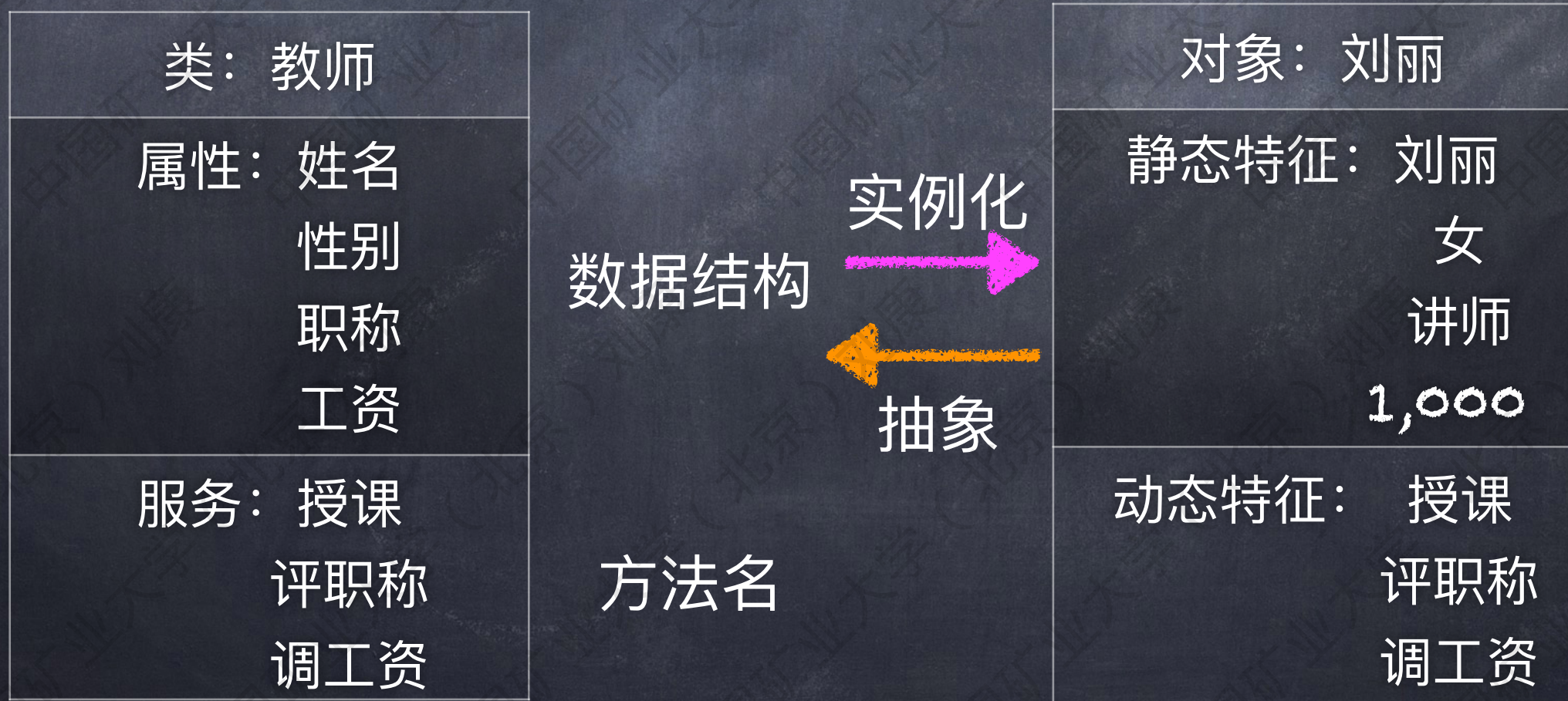
类的属性(静态特征)

类的方法(动态特征)

1.4 类与对象的关系

- 类与对象之间的关系看成是抽象与具体的关系。

在面向对象的程序设计中，对象被称作类的一个实例(instance)，而类是对象的模板(template)。类是多个实例的综合抽象，而实例又是类的个体实物。



1.5 定义类的格式

1. 系统定义的类，即Java类库。

Java类库是Java语言的重要组成部分。Java语言由语法规则和类库两部分组成，语法规则确定Java程序的书写规范；类库则提供了Java程序与运行它的系统软件(Java虚拟机)之间的接口。

2. 用户自己定义的类

用户程序仍然需要针对特定问题的特定逻辑来定义自己的类。用户按照Java的语法规则，把所研究的问题描述成Java程序中的类，以解决特定问题。

Java程序中，用户自己定义类的一般格式：

```
class 类名
```

```
{
```

```
    数据成员;
```

```
    成员方法;
```

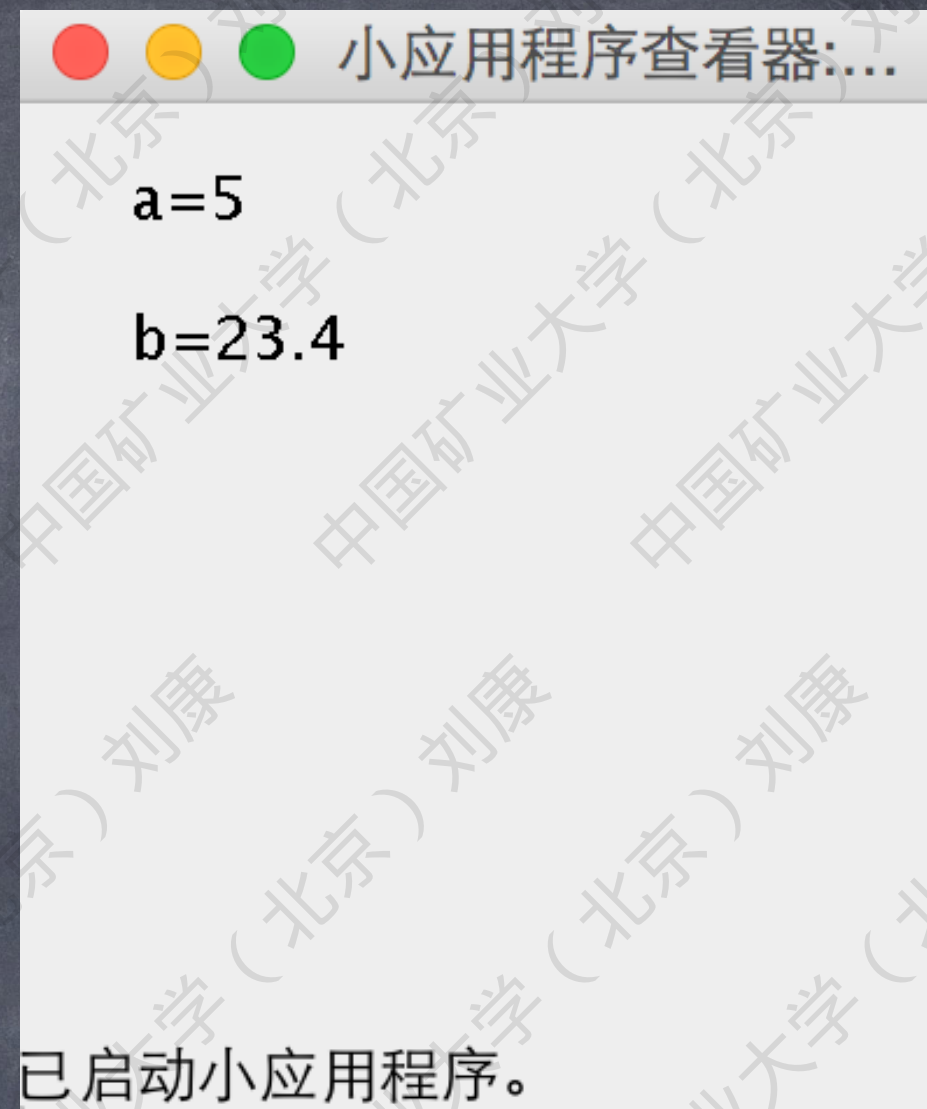
```
}
```


【程序c4_1.java】

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

public class c4_1 extends Applet
{
    int a=5;           //数据成员a
    double b=23.4;     //数据成员b

    public void paint(Graphics g) //成员方法paint
    { //以下使用g对象的drawString方法
        g.drawString("a="+a, 25, 25);
        g.drawString("b="+b, 25, 55);
    }
}
```



1.6 Java类库

- Java 的类库是系统提供的已实现的标准类的集合，是Java编程的API，它可以帮助开发者方便、快捷地开发Java程序。
- Java类库的主要部分是由它的发明者SUN公司提供的，这些类库称为基础类库(JFC)，也有少量则是由其他软件开发商以商品形式提供的。

●使用类库中系统定义好的类有三种方式：

1. **直接使用系统类**。例如在字符界面向系统标准输出设备输出字符串时使用的方法 `System.out.println()`，就是系统类 `System` 的动态属性 `out` 的方法；

2. **继承系统类**。在用户程序里创建系统类的子类，例如每个 `Java Applet` 的主类都是 `java.applet` 包中的 `applet` 类的子类；

3. **创建系统类的对象**。例如图形界面的程序中要接受用户的输入时，就可以创建一个系统类 `TextField` 类的对象来完成这个任务。

无论采用哪种方式，使用系统类的前提条件是这
个系统类应该是用户程序可见的类。用户程序需要用
import语句引入它所用到的系统类或系统类所在的
包。

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

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

引入java.awt包和java.awt.event包。

1.7 创建对象

- 创建对象通常包括声明对象、建立对象和初始化对象。

1. 声明对象

确定对象名称，并指明该对象所属的类。

类名 对象名表

```
class_name object_one, object_two;
```


2. 建立对象

用Java提供的new关键字为对象分配存储空间。

对象名 = new 构造方法()

```
object_one = new class_name();
```

```
object_two = new class_name();
```

也可以在声明对象的同时建立对象，称为创建一个对象。

类名 对象名 = new 构造方法();

```
class_name object_one = new class_name();
```


3.初始化对象

由一个类生成一个对象时，为这个对象确定初始状态，即为它的**数据成员赋初始值**的过程。

由于初始化操作是最常用的操作之一，为简化过程，`Java`还提供了专用的方法来完成它，这个方法被称为构造方法。关于构造方法的详细内容将在1.9节讲授。

1.8 使用对象

- 在面向对象的系统中，一个对象的属性和方法被紧密地结合成一个整体，二者是不可分割的，并且限定一个对象的属性值只能由这个对象或它的方法来读取和修改。这便是封装(信息隐藏)。

对象的数据成员引用方式：

对象名.数据成员名

对象的成员方法引用方式：

对象名.成员方法名(参数表)

定义一个dogs类，包括Name、Weight和Height三个数据成员和一个名为ShowDog的成员方法。为dogs类创建Dane和Setter两个对象，确定两个对象的属性后引用ShowDog方法显示这两个对象。

```
class dogs
{
    //以下三行定义dogs类的数据成员
    public String Name;
    public int Weight;
    public int Height;

    //成员方法ShowDog( )定义完成
    public void ShowDog(Graphics g,int x,int y)
    {
        g.drawString("Name:"+Name,x,y);
        g.drawString("Weight:"+Weight,x,y+20);
        g.drawString("Height:"+Height,x,y+40);
    }
}
```



```
import java.awt.*;
import java.applet.*;
public class c4_2 extends Applet
{
    public void paint(Graphics g)
    {
```

//以下为创建对象

dogs Dane; //声明了一个名为Dane的对象，它属于dogs类

Dane= new dogs(); //建立Dane对象，为它分配存储空间

dogs Setter= new dogs();//声明Setter对象的同时建立该对象

//以下六行引用对象的数据成员，将一组值赋给对象的数据成员

Dane.Name= "Gread Dane";

Dane.Weight= 100;

Dane.Height= 23;

Setter.Name= "Irish Setter";

Setter.Weight= 20;

Setter.Height= 30;

//以下两行引用对象的成员方法

Dane.ShowDog(g,25,25);

Setter.ShowDog(g,25,120);

}

}



1.9 对象初始化与构造方法

● 对象的初始化

创建对象时，通常首先要为对象的数据成员赋初始值。Java系统提供**构造方法**来完成这一操作。

● 构造方法

方法名与类名相同的类方法。每当使用**new**关键字创建一个对象，分配内存空间后，Java系统将**自动调用**构造方法初始化新建对象。

构造方法的特点：

- (1) 构造方法的方法名与类名相同。
- (2) 构造方法是类的方法，它能够简化对象数据成员的初始化操作。
- (3) 不能对构造方法指定类型，有隐含的返回值，该值由系统内部使用。

- (4) 构造方法一般不能显式地直接调用，在创建一个类的对象的同时，系统会自动调用该类的构造方法将新对象初始化。
- (5) 构造方法可以重载，即可定义多个具有不同参数的构造方法。
- (6) 构造方法可以继承，即子类可以继承父类的构造方法。
- (7) 如果用户在一个自定义类中未定义该类的构造方法，系统定义缺省的空构造方法。


```
class dogs //定义 dogs类
```

```
{
```

```
    public String Name;
```

```
    public int Weight;
```

```
    public int Height;
```

```
    public dogs(String CName, int CWeight, int CHeight) //构造方法
```

```
    { Name = CName;
```

```
      Weight = CWeight;
```

```
      Height = CHeight;
```

```
    }
```

```
}
```


● 特殊情况下，构造方法中的参数名可能与数据成员名相同，可用下面两种形式区分数据成员名与参数名。

(1) 默认法。赋值号左边的标识符默认为对象的数据成员，赋值号右边的标识符为参数。

```
public dogs(String Name, int Weight, int Height)
{
    Name = Name;
    Weight = Weight;
    Height = Height;
}
```


(2)使用代表本类对象的关键字`this`指出数据成员名之所在。

```
public dogs(String Name, int Weight, int Height)
{
    this.Name= Name;
    this.Weight= Weight;
    this.Height= Height;
}
```


2 封装机制

● 封装的概念

也称信息隐藏，利用抽象数据类型将**数据**和**基于数据的操作**封装在一起，使其构成一个不可分割的独立实体，数据被保护在抽象数据类型的内部，保留对外接口。

面向对象系统的封装单位是**对象**，类概念本身也具有封装的意义，因为对象的特性是由它所属的类说明来描述的。

👁封装的特点

1. 在类的定义中设置访问对象的属性(数据成员)及方法(成员方法)的权限，限制本类对象及其他类的对象使用的范围。
2. 提供一个接口来描述其他对象的使用方法。
3. 其他对象不能直接修改本对象所拥有的属性和方法。

2.2 类的严谨定义

- 在类的严谨定义格式中，类的说明部分增加了[类修饰符]、[extends 父类名]和[implements 接口列表]三个可选项，充分地展示封装、继承和信息隐藏等面向对象的特性。

class 类名

{

数据成员;

成员方法;

}



[类修饰符] class 类名 [extends 父类名] [implements 接口列表]

{

数据成员;

成员方法;

}

2.3 类修饰符

- 类的修饰符用于说明对它的访问限制，一个类可以没有修饰符、`public`、`final`、`abstract`等几种不同的修饰符。

1. 无修饰符的情况

如果一个类前无修饰符，则这个类只能被**同一个包里的类**使用。

【例4-2】

```
class pp          //无修饰符的类pp
{
    int a=45;    //pp类的数据成员a
}

public class e4_2 //公共类e4_2
{
    public static void main(String[] args)
    {
        pp p1=new pp( );    //类e4_2中创建了一个无修饰符类pp的对象p1
        System.out.println(p1.a);
    }
}
```


2. public修饰符

如果一个类的修饰符是`public`，则这个类是公共类。公共类不但可供它所在包中的其他类使用，也可供其他包中的类使用。在程序中可以用`import`语句引用其他包中的`public`类。

Java规定，在一个程序文件中，只能定义一个`public`类，其余的类可以是无修饰符的类，也可以是用`final`修饰符定义的最终类，否则编译时会报错。

3. final修饰符

用final修饰符修饰的类被称为最终类，不能被任何其他类继承。

定义最终类(final)的目的有三个好处：

- a. 用来完成某种标准功能。
- b. 提高程序的可读性。
- c. 提高安全性。

4. `abstract`修饰符

`abstract`修饰符修饰的类称为抽象类。抽象类刻画了研究对象的共有行为特征，并通过继承机制将这些特征传送给它的派生类。

作用：将许多有关的类组织在一起，提供一个公共的基类，为派生具体类提供基础。

注意：当一个类中出现一个或多个用`abstract`修饰符定义的方法时，则必须在这个类的前面加上`abstract`修饰符，将其定义为抽象类。

5. 注意事项

可以同时使用两个修饰符来修饰一个类，当使用两个修饰符修饰一个类时，这些修饰符之间用空格分开，写在关键字`class`之前，修饰符的顺序对类的性质没有任何影响。

`public abstract`



`public final`



`abstract final`



3 数据成员

● 数据成员声明

数据成员是用来描述事物的静态特征的。一般情况下，声明一个数据成员必须给出这个数据成员的标识符并指明它所属的数据类型。

[修饰符] 数据成员类型 数据成员名表;

访问权限修饰符: public、private、protected

非访问权限修饰符: static、final

3.2 static 修饰静态数据

● 用static修饰符修饰的数据成员是不属于任何一个类的具体对象，而是属于类的静态数据成员。其特点如下：

- 1) 它被保存在类的内存区的公共存储单元中,而不是保存在某个对象的内存区中。
- 2) 可通过类名加`点操作符`访问它。
- 3) static类数据成员仍属于类的作用域，还可以使用`public static`、`private static`等进行修饰。修饰符不同，可访问的层次也不同。

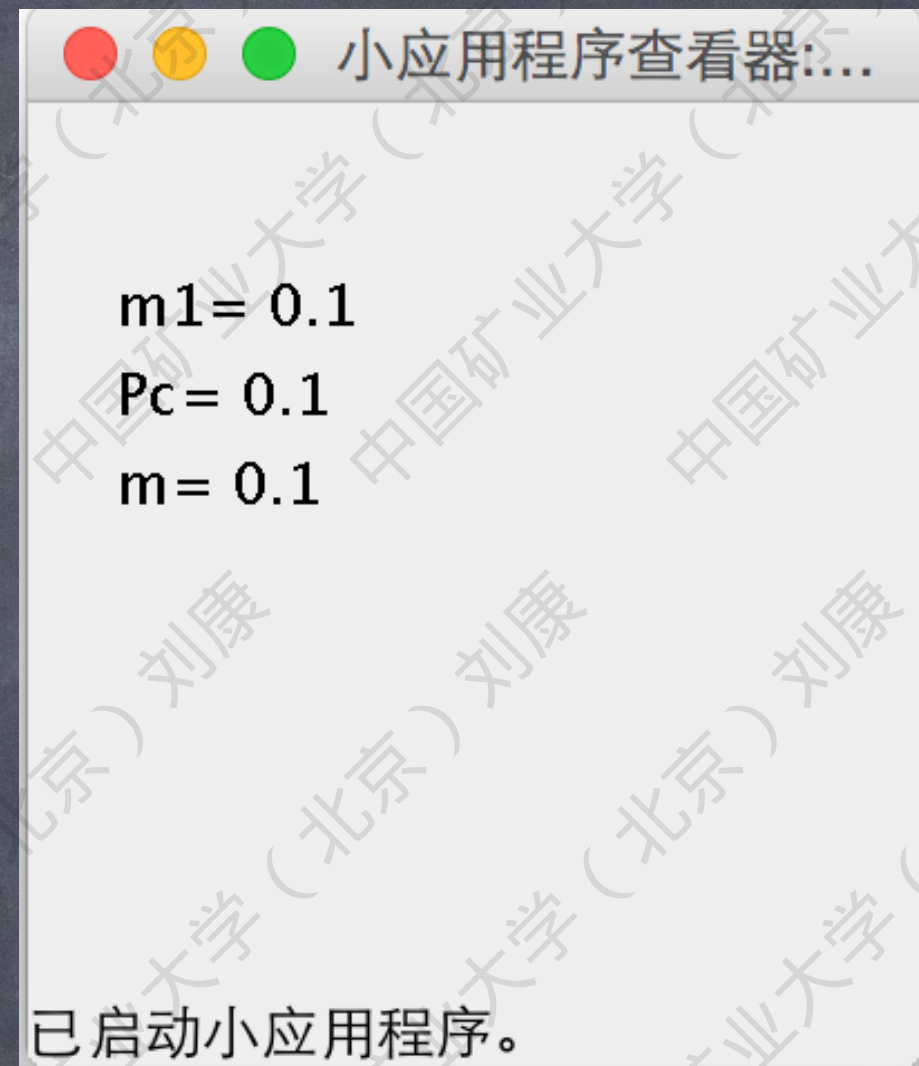

```
import java.awt.*;  
import java.applet.*;
```

```
class pc  
{  
    static double ad= 8;  
}
```

```
public class c4_3 extends Applet  
{  
    public void paint(Graphics g)  
    {  
        pc m=new pc();  
        pc m1=new pc();
```

```
        m.ad=0.1;
```

```
        g.drawString("m1=" + m1.ad, 20, 50);  
        g.drawString("Pc=" + pc.ad, 20, 70);  
        g.drawString("m=" + m.ad, 20, 90);  
    }  
}
```



3.3 静态数据成员初始化

- 静态数据成员的初始化可以由用户在定义时进行，也可以由静态初始化器来完成。
- 静态初始化器是由关键字`static`引导的一对花括号括起的语句块，其作用是在加载类时，初始化类的静态数据成员。

```
static
```

```
{
```

待初始化的静态数据成员；

```
}
```


● 静态初始化器与构造方法不同，它有下列特点：

1. 静态初始化器用于对类的静态数据成员进行初始化。而构造方法用来对新创建的对象进行初始化。
2. 静态初始化器不是方法，没有方法名、返回值和参数表。
3. 静态初始化器是在它所属的类加载到内存时由系统调用执行的，而构造方法是在系统用new运算符产生新对象时自动执行的。

3.4 final 修饰数据成员

- 如果一个类的数据成员用 `final` 修饰符修饰，则这个数据成员就被限定为最终数据成员。
- 最终数据成员可以在 **声明时初始化**，也可通过 **构造方法赋值**，但不能在程序的其他部分赋值，它的值在程序的整个执行过程中不能改变。

`final` 修饰的数据成员是 **标识符常量**

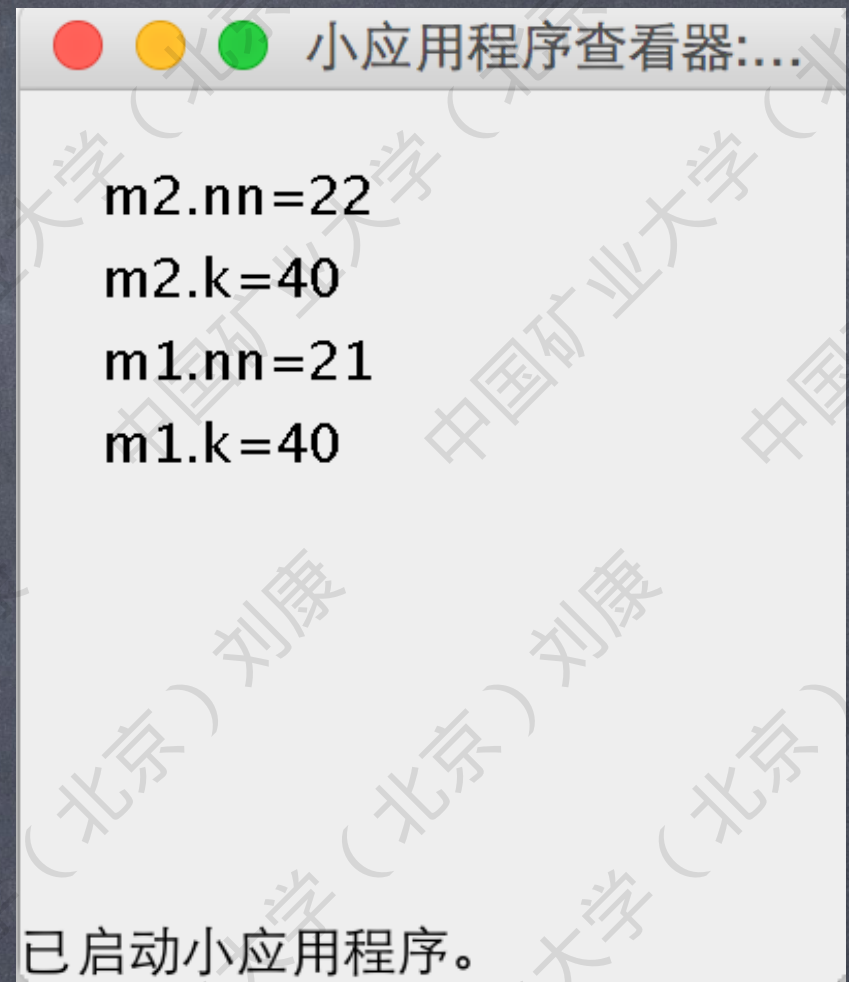

```
import java.awt.*;
import java.applet.*;
class ca
{
    static int n=20;
    final int nn;
    final int k=40;

    ca()
    { nn= ++n; }
}

public class c4_4 extends Applet
{
    public void paint(Graphics g)
    {
        ca m1=new ca( );
        ca m2=new ca( );

        m1.nn=90;

        g.drawString("m2.nn="+m2.nn, 20,30);
        g.drawString("m2.k="+m2.k, 20,50);
        g.drawString("m1.nn="+m1.nn, 20,70);
        g.drawString("m1.k="+m1.k, 20,90);
    }
}
```



4 成员方法

- 成员方法描述对象所具有的**功能或操作**，反映对象的行为，是具有某种相对独立功能的程序模块。
- 一个类或对象可以有多个成员方法，对象通过执行它的**成员方法对传来的消息作出响应**，完成特定的功能。
- 成员方法一旦定义，便可在不同的程序段中**多次调用**，故可增强程序结构的清晰度，提高编程效率。

4.1 成员方法的分类

● 从成员方法的来源看，可将成员方法分为：

1. 类库成员方法

由Java类库提供的，用户只需要按照Java提供的调用格式去使用这些成员方法即可。

2. 用户自己定义的成员方法

为了解决用户的特定问题，由用户自己编写的成员方法。程序设计的主要工作就是编写用户自定义类、自定义成员方法。

●从成员方法的形式看，可将成员方法分为：

1.无参成员方法

```
void printStar( )  
{ ..... }
```

2.带参成员方法

```
int add(int x,int y)  
{ ..... }
```


4.2 成员方法格式

- 在Java程序中，成员方法的声明只能在类中进行。

[修饰符] 返回值的类型 成员方法名(形式参数表) throw [异常表]

{

变量声明部分

执行语句部分

}

4.3 方法体的局部变量

局部变量注意事项：

1. 在方法体内可定义本方法所使用的变量，这种变量是局部变量，生存期与作用域是在本方法内，离开本方法则这些变量被**自动释放**。
2. 方法体内定义变量时，变量前**不能加修饰符**。
3. 局部变量在**使用前必须明确赋值**，否则编译时会出错。
4. 在一个方法内部，可以在复合语句中定义变量，这些变量**只在复合语句中有效**。

【例4-3】 局部变量

```
static int add(int x,int y){  
    public int zz;  
    int z,d;  
    z=x+y; z= x+d;  
    return z;  
}
```

```
public class e4_3_1  
{  
    public static void main(String[] args)  
    { int a=2,b=3;  
      int f=add(a,b);  
      System.out.println("f="+f);  
      System.out.println("z="+z);  
    }  
}
```



```
public class e4_3_2
```

```
{
```

```
    public static void main(String[] args)
```

```
    { int a=2,b=3;
```

```
      {
```

```
        int z=a+b;
```

```
        System.out.println("z="+z);
```

```
      }
```

```
        System.out.println("z="+z);
```

```
    }
```

```
}
```


4.4 形参与实参

- 一般来说，可通过如下的格式来引用成员方法：

成员方法名（实参列表）

4.4 形参与实参

● 引用时应注意以下问题：

1. 对于无参成员方法来说，是没有实参列表的，但方法名后的括弧不能省略。

4.4 形参与实参

● 引用时应注意以下问题：

2. 对于带参数的成员方法来说，

- 1) 实参的个数、顺序、数据类型必须与形式参数保持一致，
- 2) 各个实参间用逗号分隔。
- 3) 实参名与形参名可以相同，也可以不同。

4.4 形参与实参

● 引用时应注意以下问题：

3. 实参可以是表达式，

- 1) 要注意使表达式的数据类型与形参的数据类型相同
- 2) 表达式的类型按Java类型转换规则达到形参指明的数据类型

4.4 形参与实参

● 引用时应注意以下问题：

4. 实参变量对形参变量的数据传递是“**值传递**”，即只能由实参传递给形参，而不能由形参传递给实参。

4.5 成员方法引用

1. 方法语句

成员方法作为一个独立的语句被引用。

```
add(a,b);
```

2. 方法表达式

成员方法作为表达式中的一部分，通过表达式被引用。

```
f3 = 2 + add(f1, f2);
```


3.方法作为参数

一个成员方法作为另一个成员方法的参数被引用。

```
add(a, add(f1,f2) );
```

4.通过对象来引用

两重含义，一是通过“对象名.方法名”的形式引用对象；二是当对象作为成员方法的参数时，通过这个对象参数来引用对象的成员方法。

```
Dane.ShowDog(g, 25, 25);
```


👁 注意事项:

1. 如果被引用的方法存在于本文件中，而且是本类的方法，则可直接引用。
2. 如果被引用的方法存在于本文件中，但不是本类的方法，则要考虑类的修饰符与方法的修饰符来决定是否能引用。


3. 如果被引用的方法**不是本文件**的方法，而是Java类库的方法，则必须在文件的开头处**用import命令**引用有关库方法所需要的信息写入本文件中。

4. 如果被引用的方法是用户在其他的文件中自己定义的方法，则必须通过**加载用户包**的方式来引用。

4.6 成员方法递归引用

- 成员方法的递归引用就是指在一个方法中直接或间接引用自身的情况

```
int f1 (int n)
{
    int p;
    ...
    p = f1(n-1);
    return p;
}
```



4.7 static修饰静态方法

● static方法使用特点：

1. static方法是属于整个类的，它在内存中的代码段将随着类的定义而分配和装载。
2. 引用静态方法时，可以使用对象名做前缀，也可以使用类名做前缀。
3. static方法只能访问static数据成员，不能访问非static数据成员，但非static方法可以访问static数据成员。

4. `static`方法只能访问`static`方法，不能访问非`static`方法，但非`static`方法可以访问`static`方法。

5. `static`方法不能被覆盖，也就是说，这个类的子类，不能有相同名、相同参数的方法。

6. `main`方法是`static`方法。在Java的每个Application程序中，都必须有且只能有一个`main`方法，它是Application程序运行的入口点。

4.8 数学函数方法

- 作为static方法的典型例子，Java类库提供的标准数学函数方法，都是static方法。
- 标准数学函数方法在Java.Lang.Math类中，使用方法比较简单，格式如下：

类名.数学函数方法名(实参表列)

具体函数方法参见课本第87页

4.9 final 修饰最终方法

- 在面向对象的程序设计中，子类可以利用覆盖机制修改从父类那里继承来的某些数据成员及成员方法，这给程序设计带来方便的同时，也给系统的安全性带来了威胁。
- Java 提供 `final` 修饰符修饰的方法称为最终方法，则该类的子类就不能覆盖父类的方法，仅能使用从父类继承来的方法。

4.10 native 修饰本地方法

- 修饰符 `native` 修饰的方法称为本地方法，此方法是为了将其他语言(例如 `C/C++/FORTRAN/汇编` 等)嵌入到 `Java` 语言中。这样 `Java` 可以充分利用已经存在的其他语言的程序功能模块，避免重复编程。
- 在 `Java` 程序中使用 `native` 方法时，应该**特别注意平台问题**。