

# Java 程序设计

## 第 4 章类与对象





# 导读

## □ 主要内容

- ◆ 类、构造方法与对象的创建、类与程序的基本结构
- ◆ 参数传值
- ◆ 对象的组合、实例成员与类成员、方法重载
- ◆ this 关键字、包、import 语句
- ◆ 对象数组

## □ 重点和难点

- ◆ Java 类的语法规则、类变量和实例变量 、 import 语句
- ◆ 对象的创建，对象引用与实体的关系、访问权限的理解



## 4.1 编程语言的几个发展阶段



## 4.1 编程语言的几个发展阶段

### 面向过程

教师上课



学生听课



教师提问



学生回答



# 4.1 编程语言的几个发展阶段

## 面向对象



## 4.1 编程语言的几个发展阶段

### 面向服务

提问

回答问题

留作业

听课

教师讲课

考试

学生讲课

批卷子





# 理念的改变

学习面向对象语言的过程中，一个简单的理念就是：  
需要完成某种任务时，首先要想到，谁去完成任务，  
即**哪个对象去完成任务**；提到数据，首先想到这  
个**数据是哪个对象的**





## 4.2 类

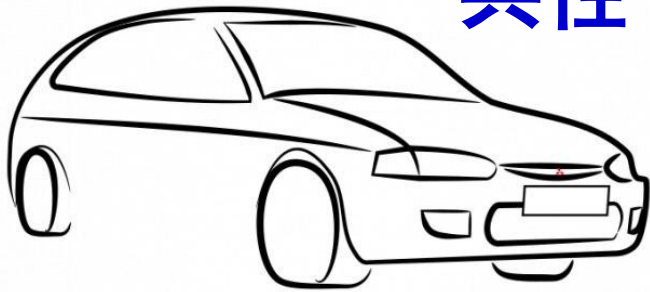
- Java 语言是面向对象语言，它的源程序是由若干个类组成，源文件是扩展名为 .java 的文本文件
- 类是 Java 语言中最重要的“数据类型”，类声明的变量被称作对象（见后面的 4.3 节），即类是用来创建对象的模板





## 4.2 类

共性

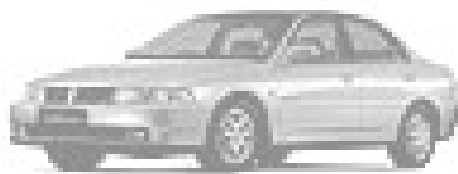


个性



## 4.2 类

汽车设计图



第 1 辆



第 2 辆



第 3 辆



## 4.2 类

- 面向对象的基本特征

➤ **封装性：** 利用抽象数据类型将数据和基于数据的操作封装在一起，保护数据并隐蔽具体的细节，只保留有限的接口与外界联系。

➤ **继承性：** 使特殊类（子类）的对象拥有一般类（父类）的全部属性与方法，同时可以增添自身的属性和方法。

➤ **多态性：** 一个程序中同名的多个不同方法共存的情况。常用重载和覆盖。



## 4.2 类

□ 类的实现包括两部分：**类声明**和**类体**

```
访问修饰符 class 类名 {  
    类体的内容  
}
```

public、**private**、**abstract**、**final**

**其中：** class 是关键字，用来定义类。“class 类名”是类的声明部分，类名必须是合法的 Java 标识符。两个大括号以及之间的内容是类体

类的目的是为了描述一类事物共有的**属性和功能**



## 4.2.1 类声明

```
class People {
```

```
...
```

```
}
```

```
class 植物 {
```

```
...
```

```
}
```

`class People` 和 `class 植物` 称作类声明,“People”和“动物”分别是类名

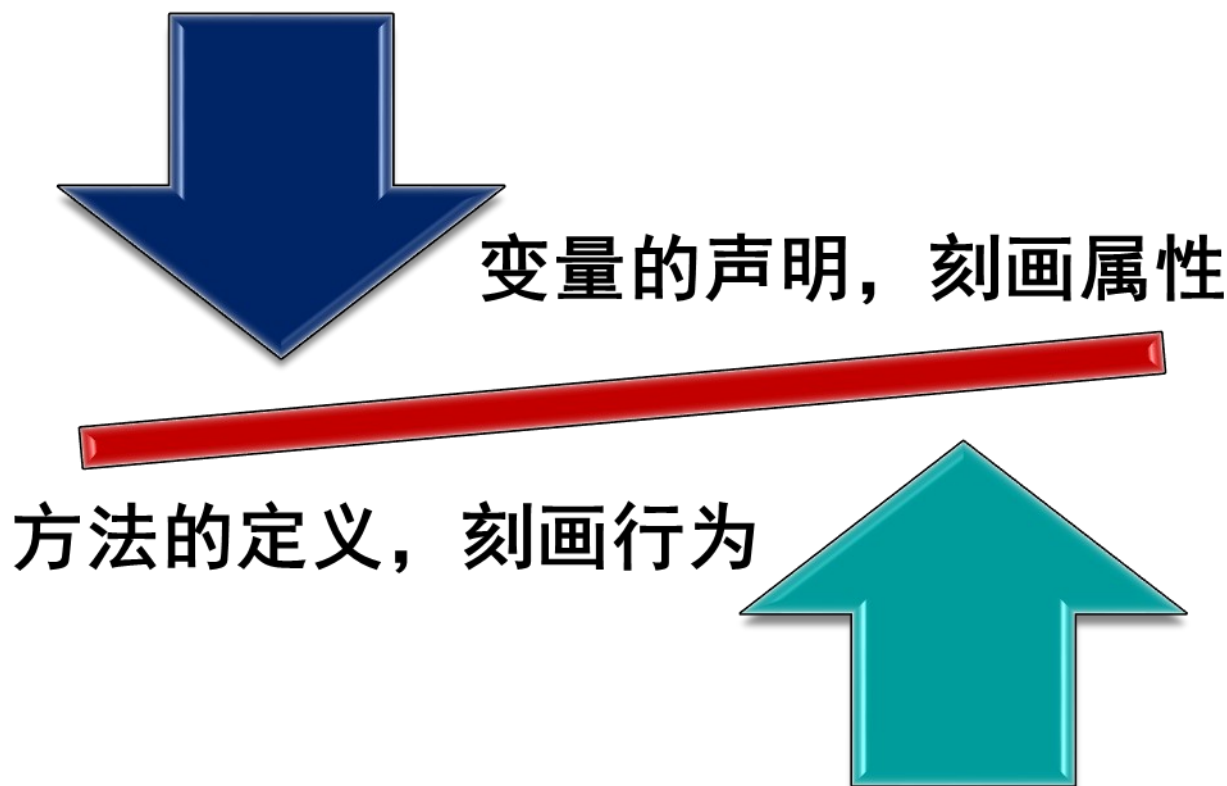
□类命名时遵守下列编程风格:

- ◆如果类名使用拉丁字母,那么名字的首字母使用大写字母。
- ◆类名最好容易识别、见名知意。当类名由几个“单词”复合而成时,每个单词的首字母使用大写



## 4.2.2 类体

□类声明之后的一对大括号“{”，“}”以及它们之间的内容称作类体，大括号之间的内容称作类体的内容



## 4.2.2 类体

```
class Lader {  
    float above; // 梯形的上底 (变量声明)  
    float bottom; // 梯形的下底 (变量声明)  
    float height; // 梯形的高 (变量声明)  
    float area; // 梯形的面积 (变量声明)  
    float computerArea() { // 计算面积 (方法)  
        area = (above+bottom)*height/2.0f;  
        return area;  
    }  
    void setHeight(float h) { // 修改高 (方法)  
        height = h;  
    }  
}
```





## 4.2.3 成员变量

### 1. 成员变量的类型

- ◆ 类体中变量定义部分所定义的变量被称为类的**成员变量**
- ◆ 任何类型

```
class Factory {  
    Float [] arrs;  
    Worker zhang;  
}
```







## 4.2.3 成员变量

### 2. 成员变量的有效范围

- ◆在整个类中有效

### 3. 编程风格

- ◆一行只声明一个变量
- ◆通常变量名首字母小写
- ◆名字有意义



## 4.2.3 成员变量与局部变量

### 1. 变量分为成员变量和局部变量

- ◆ 类体中变量定义部分所定义的变量被称为类的**成员变量**
- ◆ 成员变量在**整个类内**都有效，其有效性与它在类体中书写的先后位置无关
- ◆ 成员变量在定义时**有默认值**
- ◆ 在方法体中定义的变量和方法的参数被称为**局部变量**
- ◆ 局部变量只在定义它的**方法内有效**
- ◆ 局部变量在定义时**没有默认值**



## 4.2.3 成员变量与局部变量

2. 成员变量又分为实例成员变量（简称**实例变量**）和**类变量**（也称静态变量）

- ◆ 如果成员变量的类型前面加上关键字 **static**，这样的成员变量称做是类变量或静态变量
- ◆ 其他的变量统称为实例变量





# 例题

□ 定义梯形类如下：

```
class Lader
{   float above, area;
    float computerArea()
    {   area=(above+bottom)*height/2;
        return area;
    }
    float bottom;
    void setHeight(float h)
    {   height=h;
    }
    float height;
}
```





## 4.2.4 方法

- 方法的定义包括两部分：方法声明和方法体。一般格式为：

```
方法声明部分    {  
    方法体的内容  
}  
  
public static void main(String[] args) {  
    System.out.println("Hello");  
}
```



## 4.2.4 方法

### 1. 方法声明部分（方法头）

- ◆ 方法头由方法的类型、名称和名称之后的一对小括号以及其中的参数列表所构成。例如：

```
int speak() {           // 无参数的方法头
```

```
    return 23;
```

```
}
```

```
int add(int x,int y,int z) { // 有参数的方法头
```

```
    return x+y+z;
```

```
}
```

- ◆ **注意：**方法返回的数据类型可以是 Java 中的任何数据类型之一，当一个方法不需要返回数据时，返回类型必须是 void



## 4.2.4 方法

### 2. 方法体

- ◆ 方法声明之后的一对大括号 “{” , “}” 以及之间的内容称为方法的方法体。
- ◆ 方法体的内容包括局部变量的声明和 Java 语句。如：

```
int getSum(int n) {    // 参数变量 n 是局部变量
    int sum = 0;        // 声明局部变量 sum
    for (int i = 1 ; i<=n; i++) { // for 循环语句
        sum = sum + i;
    }
    return sum;        // return 语句
}
```



# 例题

```
public class A {  
    void f() {  
        int m = 10, sum = 0; // 成员变量，在整个类中有效  
        if(m>9) {  
            int z = 10; //z 仅仅在该复合语句中有效  
            z = 2*m+z;  
        }  
        for(int i=0;i<m;i++) {  
            sum = sum+i; //i 仅仅在该循环语句中有效  
        }  
        m = sum;    // 合法，因为 m 和 sum 有效  
        z = i+sum;  // 非法，因为 i 和 z 已无效  
    }  
}
```





## 4.2.4 方法

### 3. 区分成员变量和局部变量

◆如果局部变量的名字与成员变量的名字相同，则成员变量被隐藏，即该成员变量在这个方法内暂时失效。例如：

```
class Tom {  
    int x = 10, y;
```

```
    void f() {
```

```
        int x = 5;
```

```
        y = x + x;
```

```
        //y 得到的值是 10，不是 20。
```

```
        // 如果方法 f 中没有 “int x=5;”，y 的值将  
        是 20
```

```
    }
```

```
}
```

$y = x + \text{this.x};$

如果想在该方法中使用被隐藏的成员变量，必须使用关键字 **this**



## 4.2.4 方法

### 4. 局部变量没有默认值

```
class Test {  
    int x;  
    void f() {  
        int y;  
        int z1 = x + 5;  
        int z2 = y + 10;  
        System.out.println(z1);  
        System.out.println(z2);  
    }  
}
```



## 4.2.5 需要注意的问题

- 类体的内容由两部分构成：一部分是变量的声明，另一部分是方法的定义。

- 定义类需要

```
class A {  
    int a=12; //ok  
    int b;  
    b=12;    //error  
}
```

- ◆ 对成员

量和该

法可以对成员变

作。在声明成员

变量时可以同时赋予初值，但是不可以在类体中有单独

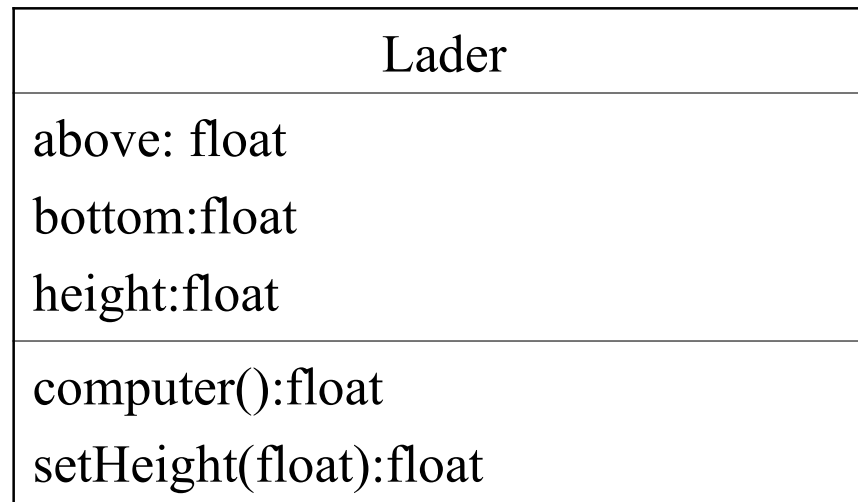
的赋值语句，但局部变量必须赋初值



## 4.2.6 类的 UML 图

### □ UML(Unified Modeling Language Diagram)

- ◆ 是被用于描述一个系统的静态结构。一个 UML 中通常包含有类 (Class) 的 UML 图，接口 (Interface) 等
- ◆ 在类的 UML 图中，使用一个长方形描述一个类的主要构成，将长方形垂直地分为三层





## 4.3 构造方法与对象的创建

- 类是面向对象语言中最重要的一种数据类型
- 在面向对象语言中，用类声明的变量被称作对象
- 用类声明对象后，还必须要创建对象
- 类是创建对象的“模板”





## 4.3.1 构造方法

### □ 构造方法：

- ◆ 一种特殊方法，名字必须与所在**类名相同**，且**没有类型**
- ◆ **构造函数可重载**
- ◆ 创建对象时使用，用来**初始化各个成员变量**

- 需要注意的是如果类中没有编写构造方法，系统会默认该类只有一个构造方法，该默认的构造方法是无参数的，且方法体中没有语句



# 构造函数例题

```
class 梯形
```

```
{    float 上底, 下底, 高;
```

```
    梯形 ()
```

```
    { 上底 =60;
```

```
      下底 =100;
```

```
      高 =20;
```

```
    }
```

```
    梯形 (float x,int y,float h)
```

```
    { 上底 =x;
```

```
      下底 =y;
```

```
      高 =h;
```

```
    }
```

```
}
```



## 4.3.2 创建对象

对

类

```
class XiyoujiRenwu {  
    float height,weight;  
    String head, ear;  
    void speak(String s) {  
        System.out.println(s);  
    }  
}  
  
public class Example4_1 {  
    public static void main(String args[]) {  
        XiyoujiRenwu zhubajie;    // 声明对象  
        zhubajie = new XiyoujiRenwu();  
    }  
}
```

当用类创建一个对象时，类中的成员变量被分配内存空间，这些内存空间称作该对象的实体或对象的变量，而对象中存放着引用值





# 3 . 对象的内存模型

- 通过对象的声明和分配内存后，每个对象都对应两个值：引用值和实体值。

## (1) 声明对象时的内存模型

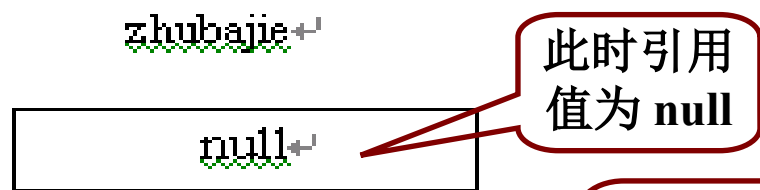


图 4.3 · 未分配变量的对象

## (2) 对象分配变量后的内存模型

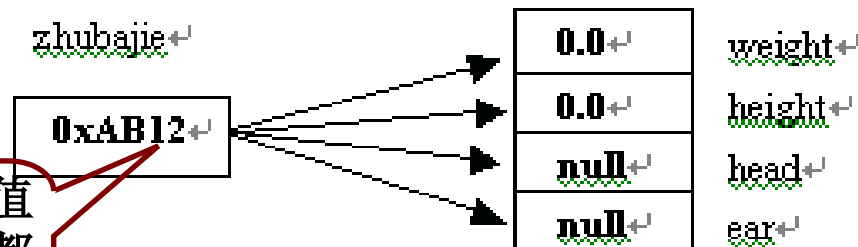


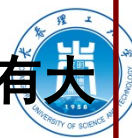
图 4.4 · 分配变量（实体）后的对象

### 分析：

当系统见到：`zhubajie=new XiyoujiRenwu();`

系统会使用 `new` 运算符为变量 `height`，`weight`，`head`，`ear` 等分配内存，将返回一个引用值给对象变量 `zhubajie`。同时调用构造函数为各个变量初始化。

结论：每个对象都有属于自己的空间；都有属于自己的成员；但有时也有大



# 3 . 对象的内存模型

## (3) 创建多个不同的对象

- ◆ 一个类通过使用 new 运算符可以创建多个不同的对象。例如创建两个对象： zhubajie 、 sunwukong

如： `zhubajie = new XiyoujiRenwu();`  
`sunwukong = new XiyoujiRenwu`

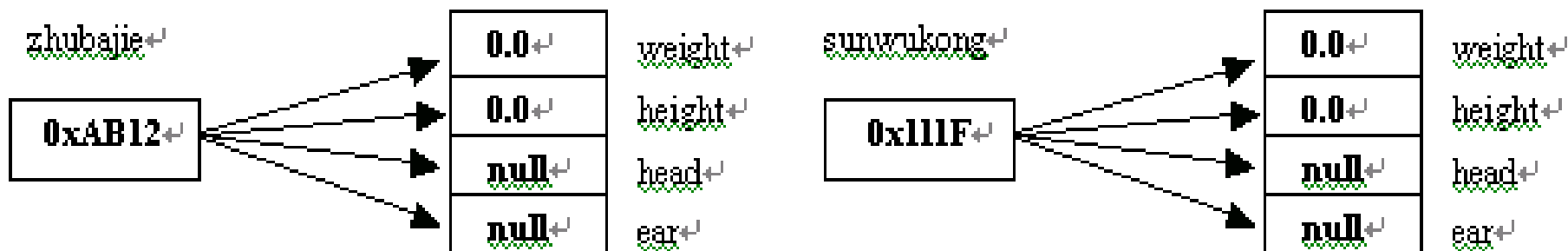


图 4.5 · 创建多个对象

是互不相同的位置。内存模型如下图 4.5 所示：



## 4.3.3 使用对象

- 对象创建成功后，可以操作类中的变量和方法：
  - ① 对象操作自己的变量（体现对象的属性）
    - ◆ 通过使用运算符 “.” 对象操作自己的变量（对象的属性）
  - ② 对象调用类中的方法（体现对象的行为）
    - ◆ 对象创建之后，可以使用点运算符 “.” 调用创建它的类中的方法，从而产生一定的行为（功能）

### Example4\_3

```
zhubajie的身高：1.8  
zhubajie的头：大头  
sunwukong的重量：1000.0  
sunwukong的头：绣花飘飘  
俺老猪我想娶媳妇  
zhubajie现在的头：歪着头  
老孙我重1000斤，我想骗八戒背我  
sunwukong现在的头：歪着头
```

..... 图 4.6 .. 使用对象



## 4.3.4 对象的引用和实体

- 一个类创建的两个对象，如果具有相同的引用，那么就具有完全相同的实体，  
例如 **point** 类

```
class Point {  
    double x,y;  
    Point(double x,double y) {  
        this.x=x;  
        this.y=y;  
    }  
}
```

使用 Point 类创建了两个对象  
p1 , p2 :

**Point p1 = new Point (5,15);**

**Point p2 = new Point(8,18);**

内存模型如图 4.9 所示

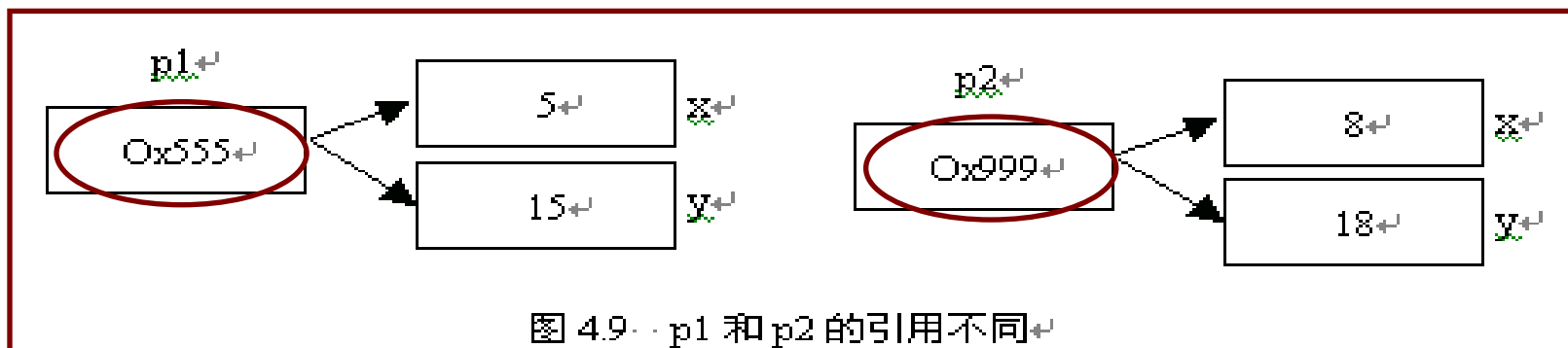


图 4.9 · p1 和 p2 的引用不同

## 4.3.4 对象的引用和实体

□ 假如在程序中使用了如下的赋值语句：

例子 4

`p1 = p2;`

□ 一个类创建的两个对象，如果具有相同的引用，那么就具有完全相同的实体（变量）。

□ 内存模型由图 4.9 变成图 4.10 所示。

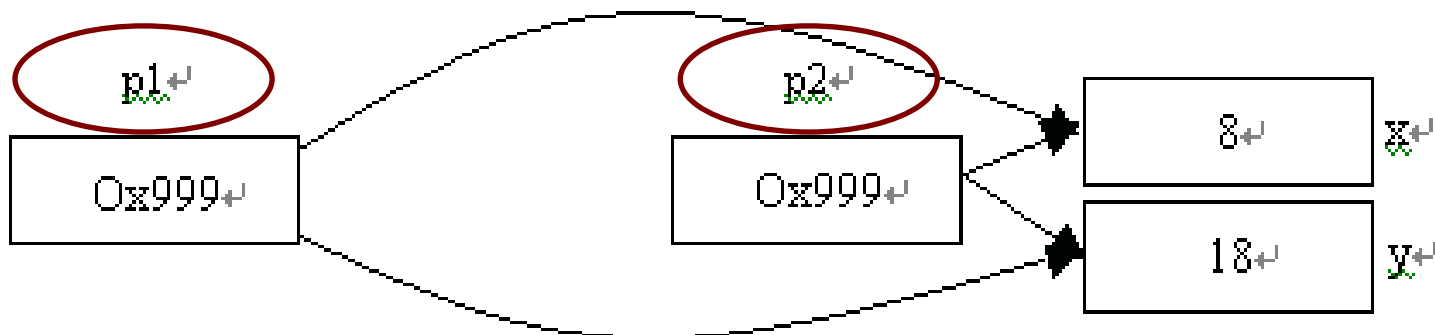


图 4.10 · `p1` 和 `p2` 的引用相同



## 4.4 类与程序的基本结构

- 一个 Java 应用程序（也称为一个工程）是由若干个类所构成，这些类可以在一个源文件中，也可以分布在若干个源文件中，如图 4.12 所示。

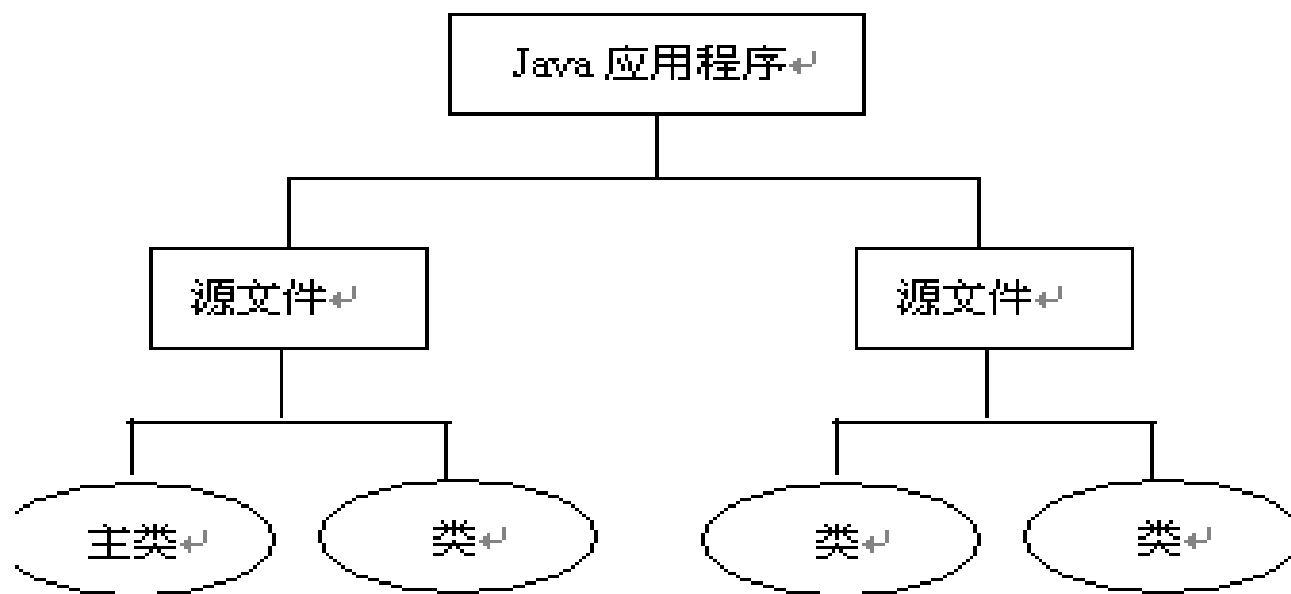


图 4.12 程序的结构



# 例题

- 例 5 中一共有三个 Java 源文件 (**Example4\_5.java** (含有主类)、**Rect.java**、**Lader.java**)
- Java 应用程序从主类的 **main** 方法开始执行，每个 Java 源文件编译后产生一个类的字节码文件

```
public class Example4_5 {  
    public static void main(String args[]) {  
        Rect.....Lader....  
    }  
}
```

```
public class Rect {  
    double width; //  
    double height; //  
    double getArea() {  
        double area=width*height;  
        return area;  
    }  
}
```

```
double height;  
double getArea() {
```

运行主类，程序的输出结果是：

矩形的面积 :2766.5266

梯形的面积 :1517.07888 。

的上底

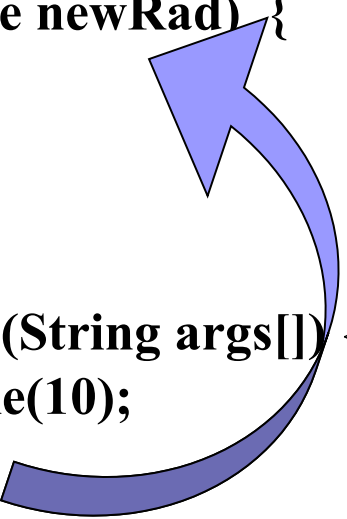
bottom)\*height/2;



## 4.5 参数传值

- 方法中最重要的部分之一是方法参数，参数属于**局部变量**，当对象调用方法时，参数被分配内存空间，并要求调用者向参数传递值，即方法被调用时，参数变量必须有具体的值。

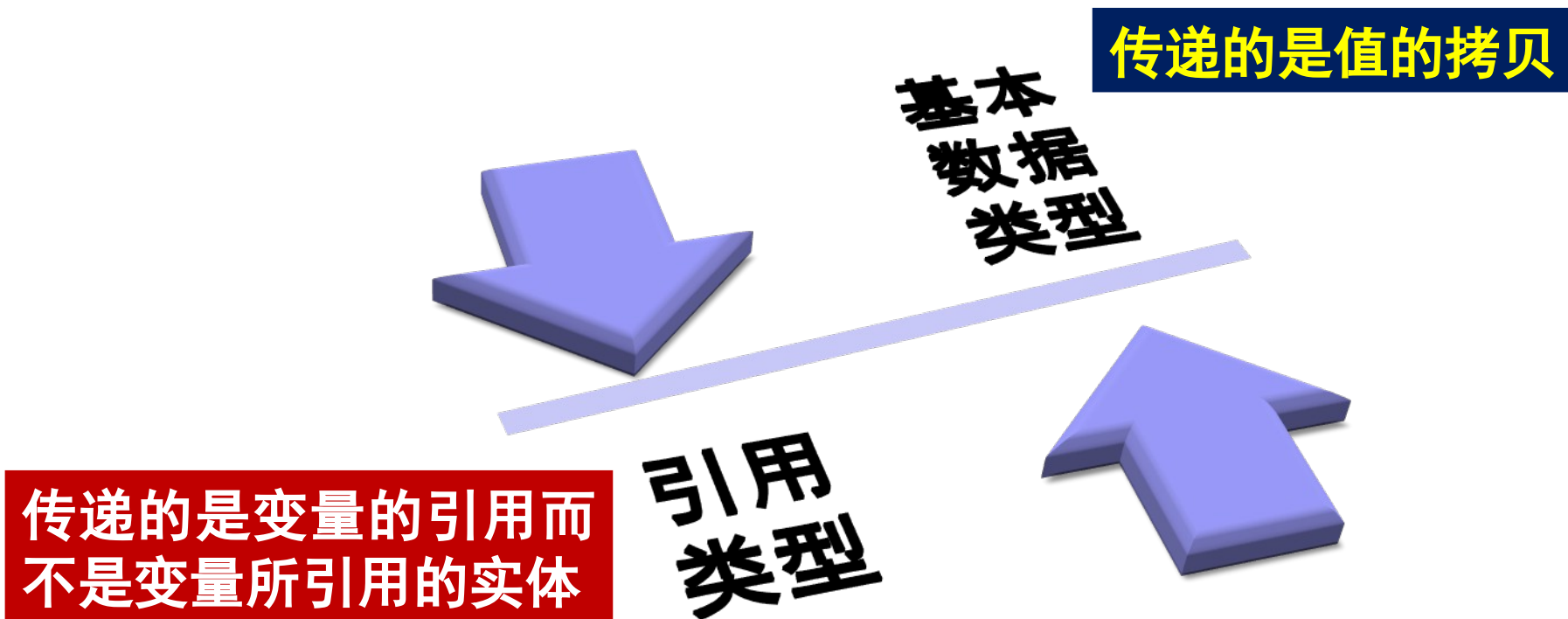
```
class Circle {  
    double rad;  
    void changeRad(double newRad) {  
        rad=newRad;  
    }  
}  
class Test {  
    public static void main(String args[]) {  
        Circle cir=new Circle(10);  
        cir.changeRad(100);  
    }  
}
```





## 4.5.1 传值机制

- 在 Java 中，方法的所有参数都是“**传值**”的，也就是说，方法中参数变量的值是调用者指定的值的拷贝。

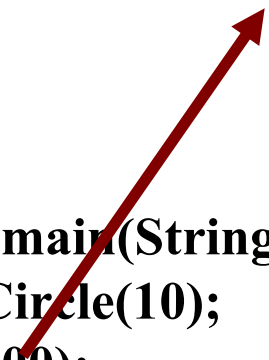


## 4.5.2

# 基本数据类型参数的传值

- 对于基本数据类型的参数，传递的是值的**拷贝**。同时向该参数传递的值的级别不可以高于该参数的级别

```
class Circle {  
    double rad;  
    Circle(double r) {  
        rad=r;  
    }  
    void changeRad(double newRad) {  
        rad=newRad;  
    } }  
class Test {  
    public static void main(String args[]) {  
        Circle cir=new Circle(10);  
        cir.changeRad(100);  
    } }
```

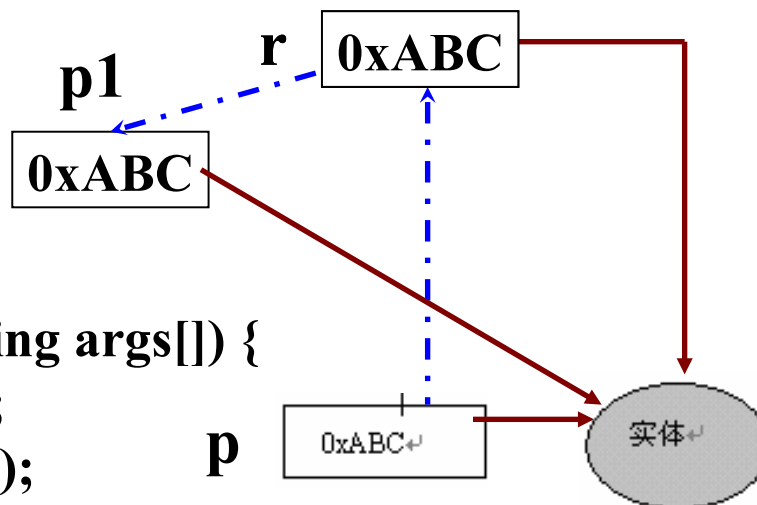


## 4.5.3

# 引用类型参数的传值

- 当参数是引用类型时，“传值”传递的是变量中存放的“引用”，而不是变量所引用的实体

```
class Circle {  
    Point p1;  
    Circle(Point r) {  
        p1=r;  
    }  
}  
  
class Test {  
    public static void main(String args[]) {  
        Point p = new Point(1,2);  
        Circle cir = new Circle(p);  
    }  
}
```



# 例题

- 例 7 模拟收音机使用电池。例 7 中使用的主要类如下
  - ◆ Radio 类负责创建一个“收音机”对象 (Radio 类在 Radio.java 中)。
  - ◆ Battery 类负责创建“电池”对象 (Battery 类在 Battery.java 中)。
  - ◆ Radio 类创建的“收音机”对象调用 openRadio(Battery battery) 方法时，需要将一个 Battery 类创建“电池”对象传递给该方法的参数 battery，即模拟收音机使用电池。
  - ◆ 在主类 (Example4\_7.java) 中将 Battery 类创建“**电池**”对象：**nanfu**，传递给 openRadio(**Battery battery**) 方法的参数 battery，该方法消耗了 **battery 的储电量**（打开收音机会消耗电池的储电量），那么 **nanfu 的储电量就发生了同样的变化**。

## 收音机和 4.14

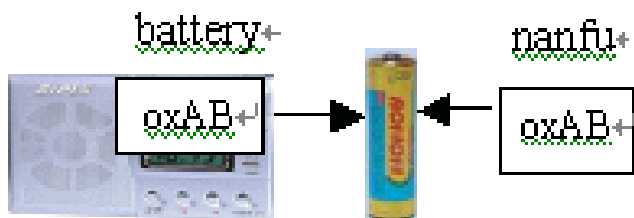


图 4.14(a) · 收音机使用电池

南孚电池的储电量是:100  
收音机开始使用南孚电池  
目前南孚电池的储电量是:90

图 4.14(b) · 收音机消耗电池的电量



## 4.5.4 可变参数

- 可变参数是指在声明方法时不给出**具体参数个数**
  - 使用 “...” 表示若干个参数，参数**类型**必须**相同**
  - 必须是方法的参数列表中的**最后一个**参数
- ◆ `public void fun(int... x)`    **x 等价于数组**

```
static void show(int... x) {  
    int sum = 0;  
    for (int i : x) {  
        sum += i;  
    }  
    System.out.println(x.length);  
    System.out.println(sum);  
}
```





## §4.6 对象的组合

- 一个类可以把对象作为自己的成员变量，如果用这样的类创建对象，那么该对象中就会有其它对象，也就是说该对象将其他对象作为自己的组成部分，或者说该对象是由几个对象组合而成
- ① 通过组合对象来复用方法也称“**黑盒**”复用，因为当前对象只能委托所包含的对象调用其方法
- ② 当前对象随时可以更换所包含的对象，即对象与所包含的对象属于**弱耦合**关系





# 例题

例 8 展示了圆锥和圆的组合关系，圆锥的底是一个圆，即圆锥有一个圆形的底。圆锥对象在计算体积时，首先委托圆锥的底（一个 Circle 对象）bottom 调用 `getArea()` 方法计算底的面积，然后圆锥对象再计算出自身的体积。

圆锥对象组合了 Circle 对象，可以委托所包含的对象调用其方法，这样一来，圆锥对象对所包含的 Circle 对象的 `getArea()` 方法的细节（计算圆面积的算法细节）是一无所知的。



# 例题

- 例 8 中（运行效果如图 4.15）模拟圆锥用圆作为底，涉及的类如下。

- Circle 类负责创建圆对象。
- Circular 类负责创建圆锥对象， 该圆锥对象可以调用方法

**setBottom(Circle c)**

将 Circle 类的实例：

即“圆”对象的引用传递给

自己所组合 Circle 类型的对象 bottom

- 圆锥对象的 Circle 类型的成员变量

Circle.java ,

Circular.java ,

Example4\_8.java

```
circle的引用:Circle@15db9742
圆锥的bottom的引用:null
circle的引用:Circle@15db9742
圆锥的bottom的引用:Circle@15db9742
圆锥的体积:523. 33333333333334
修改circle的半径, bottom的半径同样变化
bottom的半径:20.0
重新创建circle, circle的引用将发生变化
circle的引用:Circle@6d06d69c
但是不影响circular的bottom的引用
圆锥的bottom的引用:Circle@15db9742
```

图 4.15 向圆锥的底传递圆对象的引用







## §4.6.1 组合与复用

- 如果一个对象 a 组合了对象 b，那么对象 a 就可以委托对象 b 调用其方法，即对象 a 以组合的方式复用对象 b 的方法
- 例：手机功能



## 4.6.2

# 关联关系和依赖关系的 UML 图

## 1 关联关系

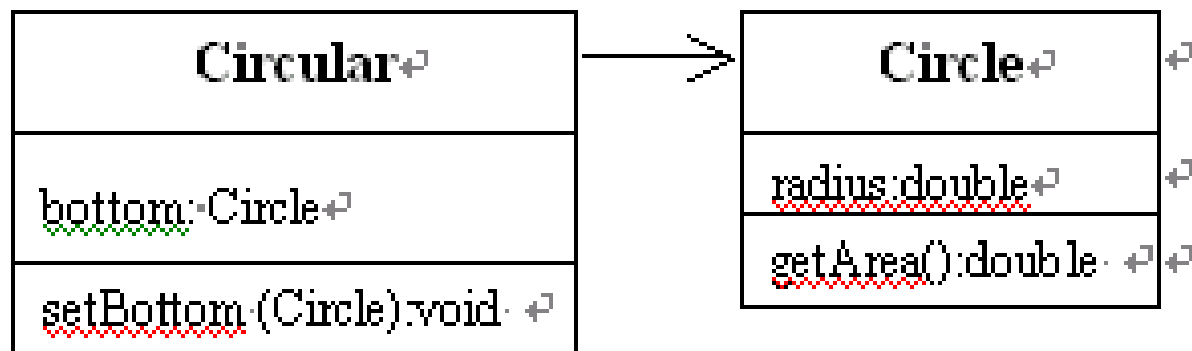


图 4.21 关联关系的 UML 图

## 2 依赖关系

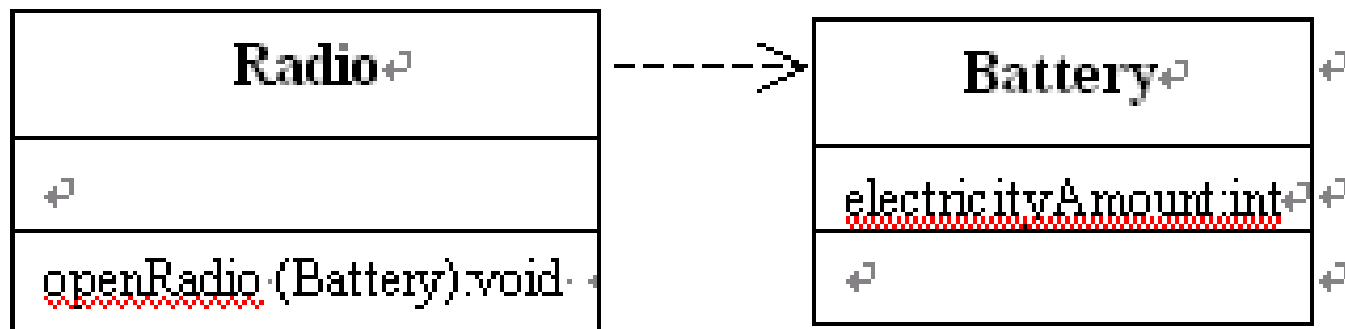


图 4.22 依赖关系的 UML 图

## 4.7 实例成员与类成员

关键字 `static`

实例变量

类变量

- 不同对象的实例变量互不相同
- 所有对象共享类变量
- 通过类名直接访问类变量





## 4.7 实例成员与类成员

```
class Person {  
    String name;  
    static String usedName;  
    public Person(String s) {  
        this.name = s;  
        this.usedName = s; }  
    public void show() {  
        System.out.println(“ 姓名: ” + name + “\n 曾用  
名: ” + usedName);  
    }  
}
```



## 4.7 实例成员与类成员

调用实例方法时，该方法中出现的实例变量就是分配给该对象的实例变量；该方法中出现的类变量也是分配给该对象的变量，只不过这个变量和所有的其他对象**共享**而已



**类方法**不仅可以被类创建的任何**对象调用执行**，也可以直接通过**类名调用**。和实例方法不同的是，类方法**不可以操作实例变量**，因为在类创建对象之前，实例成员变量还没有分配内存



## 4.7 实例成员与类成员

```
class Person {  
    String name;  
    static String usedName;  
    public Person(String s) {  
        this.name = s;  
        this.usedName = s; }  
    public void show() {  
        System.out.println(" 姓名: " + name + "\n 曾用名: " +  
                             usedName); }  
    public static void showStatic() {  
        System.out.println(" 姓名: " + name + "\n 曾用名: " +  
                             usedName); }  
}
```





## 4.8 方法重载与多态

- 方法重载的意思是：一个类中可以有多多个方法具有相同的名字，但这些方法的参数必须不同，即或者是参数的**个数不同**，或者是参数的**类型不同**

例：图形面积计算

```
void shout(int a, double b)
```

```
void shout(double a, int b)
```





## 4.9 this 关键字

- this 是 Java 的一个关键字，表示某个对象。this 可以出现在实例方法和构造方法中，但不可以出现在类方法中。
  - ◆ this 关键字出现在类的构造方法中时，代表使用该构造方法所创建的对象。
  - ◆ 当 this 关键字出现在实例方法中时，this 就代表正在调用该方法的当前对象。







## 4.9 this 关键字

- 实例方法必须只能通过对象来调用，不能用类名来调用。
- 实例方法可以操作类的成员变量，当实例成员变量在实例方法中出现时，默认的格式是：

**this. 成员变量 ;**

- static 成员变量在实例方法中出现时，默认格式是：

**类名 . 成员变量 ;**





## 4.9 this 关键字

```
Class Cylinder {  
    double radius;  
    void SetCylinder(double radius) {  
        radius = radius;  
    }  
    void GetCylinder() {  
        return radius;  
    }  
}
```






## 4.10 包

- 包是 Java 语言中有效地**管理类**的一个机制
- 包名的目的是**有效的区分**名字相同的类。不同 Java 源文件中两个类名字相同时，它们可以通过隶属不同的包来相互区分





## 4.10.1 包语句

### □ 通过关键字 package 声明包语句

- ◆ package 语句作为 Java 源文件的**第一条语句**， 为该源文件中声明的类指定包名。
- ◆ package 语句的一般格式为：

**package** 包名 ;



## §4.10.2 有包名的类的存储目录

- 程序如果使用了包语句，例如：

```
package tom.jiafei;
```

- 那么存储文件的目录结构中必须包含有如下的结构

```
...\tom\jiafei
```

如： `c:\1000\tom\jiafei`

- 并且要将源文件编译得到的类的字节码文件保存在目录 `c:\1000\tom\jiafei` 中（源文件可以任意存放）。



## §4.10.3 运行有包名的主类

- 如果主类的包名是 `tom.jiafei`，那么主类的字节码一定存放在 `... \tom\jiefei` 目录中，运行时必须到 `tom\jiefei` 的上一层（即 `tom` 的父目录）目录中去运行主类。
- 假设 `tom\jiefei` 的上一层目录是 1000，那么，必须如下格式来运行：

`C:\1000\ java tom.jiafei. 主类名`



## 4.11 import 语句

- 一个类可能需要另一个类声明的对象作为自己的成员或方法中的局部变量，如果这两个类在同一个包中，当然没有问题

Java 提供了 130 多个包：↵

Java.applet 包含所有实现 java applet 的类；↵

Java.awt 包含抽象窗口工具集中的图形，文本，窗口 GUI 类；↵

Java.awt.image 包含抽象窗口工具集中的图像处理类；↵

Java.lang 包含所有的基本语言类；↵

Java.io 包含所有的输入输出类；↵

Java.net 包含所有实现网络功能的类；↵

Java.util 包含有用的数据类型类；↵

注：系统自动引入 java.lang 包中的所有的类↵



## 4.11.2 引入自定义包中的类

- 用户程序也可以使用 import 语句引入非类库中有包名的类

```
import tom.jiafei.*;
```

- 在用户程序所在目录下建立和包相对应的子目录结构，比如用户程序所在目录是 C:\ch4，想使用 import 语句引入 tom.jiafei 包中的类，那么根据包名建立如下的目录结构：

```
C:\ch4\tom\jiafei
```







## 4.12 访问权限

- 当一个类创建了一个对象之后，该对象可以通过 “.” 运算符操作自己的变量、使用类中的方法，但对象操作自己的变量和使用类中的方法是有一定限制的
- 所谓访问权限是指对象是否可以通过 “.” 运算符操作自己的变量或通过 “.” 运算符使用类中的方法
- 访问限制修饰符有 **private** 、 **protected** 和 **public** ，都是 Java 的关键字，用来修饰成员变量或方法



## 4.12 访问权限

```
class Tom {  
    private float weight;  
    float f(float a, float b) {  
        return a * b;  
    }  
}
```

```
class Jerry {  
    void g() {  
        Tom t = new Tom();  
        t.weight = 23;  
        t.f(3, 4);  
    }  
}
```



**package cust**

**class A**

**public int a = 1;**

**private int b = 2;**

**protected int c =**

**3;**

**int d = 4;**

**final int e = 5;**

**class B**

**class C extends A**

**package jlu**

**class D extends A**

**class E**

## 4.12.2 私有变量和私有方法

- 用关键字 **private** 修饰的成员变量和方法称为私有变量和私有方法
- 对于私有成员变量或方法，只有在本类中创建该类的对象时，这个对象才能访问自己的私有成员变量和类中的私有方法
- 某个类在另外一个类中创建对象后，如果**不希该对象直接访问自己的变量**，即通过“.”运算符来操作自己的成员变量，就应当**将该成员变量访问权限设置为 private**
- 面向对象编程**提倡对象应当调用方法来改变自己的属性**，类应当提供操作数据的方法，这些方法可以经过精心的设计，使得对数据的操作更加合理



## 4.12.3 共有变量和共有方法

- 用 `public` 修饰的成员变量和方法被称为共有变量和共有方法 。
- 我们在任何一个类中用类 `Tom` 创建了一个对象后，该对象能访问自己的 `public` 变量和类中的 `public` 方法（也可以通过类名来操作成员变量、方法） 。



## 4.12.4 友好变量和友好方法

- 当在另外一个类中用类 Tom 创建了一个对象后，如果这个类与 Tom 类在同一个包中，那么该对象能访问自己的友好变量和友好方法。
- 在任何一个与 Tom 同一包中的类中，也可以通过 Tom 类的类名访问 Tom 类的类友好成员变量和类友好方法。



## 4.12.5

# 受保护的成员变量和方法

- 用 `protected` 修饰的成员变量和方法被称为受保护的成员变量和受保护的方法。

访问权限的级别排列，按访问权限从高到低的排列顺序是：`public`，`protected`，友好的，`private`。

作用域↵	当前类↵	同一 package↵	子孙类↵	其他 package↵ ↵
<u>public</u> ↵	√↵	√↵	√↵	√↵ ↵
<u>protected</u> ↵	√↵	√↵	√↵	X↵ ↵
<u>friendly</u> ↵	√↵	√↵	X↵	X↵ ↵
<u>private</u> ↵	√↵	X↵	X↵	X↵ ↵





## 4.12.6 public 类与友好类

- 类声明时，如果在关键字 `class` 前面加上 `public` 关键字，就称这样的类是一个 `public` 类。
- 可以在任何另外一个类中，使用 `public` 类创建对象。
- 如果一个类不加 `public` 修饰，这样的类被称作友好类。
- 在另外一个类中使用友好类创建对象时，要保证它们是在同一包中。







## 4.13 基本类型的类封装


□ Java 的基本数据类型包括

➤ byte 、 int 、 short 、 long 、 float 、 double 、 char 。

□ Java 提供了基本数据类型相关的类，实现了对基本数据类型的封装。

➤ Byte 、 Integer 、 Short 、 Long 、 Float 、 Double 和 Character 类。这些类在 java.lang 包中。





## 4.13.1 Double 和 Float 类

- Double 类和 Float 类实现了对 double 和 float 基本型数据的类包装。
  - ◆ Double 类的构造方法: `Double(double num)`
  - ◆ Float 类的构造方法: `Float(float num)`
- Double 对象调用 `doubleValue()` 方法可以返回该对象中含有的 double 型数据。
- Float 对象调用 `floatValue()` 方法可以返回该对象中含有的 float 型数据。





# 例题

```
class Test {  
    public static void main(String args[]) {  
        String s1="20";  
        String s2="80";  
        printf(s1+s2);  
        Double d1=new Double(s1);  
        Double d2=new Double(s2);  
        double m1=d1.doubleValue();  
        double m2=d1.doubleValue();  
        println(m1+m2);  
    }  
}
```



## 4.13.2

# Byte 、 Short 、 Integer 、 Long 类

- 上述类的构造方法分别：

Byte(byte num)

Short(short num)

Integer(int num)

Long(long num)

- Byte 、 Short 、 Integer 和 Long 对象分别调用 **byteValue()** 、 **shortValue()** 、 **intValue()** 和 **longValue()** 方法返回该对象含有的基本型数据。





## 4.13.3 Character 类

- Character 类实现了对 char 基本型数据的类包装。

Character 类的构造方法: **Character(char c)**

- Character 类中的一些常用类方法:

- ◆ **public static boolean isDigit(char ch)** ch 是数字字符返回 true

- ◆ **public static boolean isLetter(char ch)** ch 是字母返回 true

例 20 将一个字符数组中的小写字母变成大写字母，并将大写字母变成小写字母 。



## 4.14 对象数组

- 如果程序需要某个类的若干个对象，比如 Student 类的 10 个对象，显然如下声明 10 个 Student 对象是不可取的：

Student stu1, stu2, stu3, stu4, stu5, stu6, stu7, stu8, stu9, stu10;

- 正确的做法是使用对象数组，即数组的元素是对象，例如：

**Student [] stu;**

**stu = new Student[10];**

- 需要注意的是，上述代码仅仅定义了数组 stu 有 10 个元素，并且每个元素都是一个 Student 类型的对象，但这些对象目前都是空对象，因此在使用数组 stu 中的对象之前，应当创建数组所包含的对象。

例如：

**stu[0] = new Student();**



- 反编译器 `javap.exe` 可以将字节码反编译为源码，以便查看源码类中的 `public` 方法名字和成员变量的名字

例如: `javap java.awt.Button`

- 使用 `javadoc.exe` 可制做源文件类结构的 `html` 格式文档

例如: `javadoc Example.java`





## 4.16 jar 文件

- 可以使用 `jar.exe` 命令把一些类的字节码文件压缩成一个 `jar` 文件，然后将这个 `jar` 文件存放到 Java 运行环境的扩展中，即将该 `jar` 文件存放在 JDK 安装目录的 `jre\lib\ext` 文件夹中。这样，Java 应用程序就可以使用这个 `jar` 文件中的类来创建对象了
- 将 `C:\1000\moon\star` 目录中的 `TestOne.class` 和 `TestTwo.class` (二者包名是 `moon.star`) 压缩成一个 `jar` 文件：







## 4. 16 jar 文件

□Jerry.jar 的步骤:

### 1. 清单文件 :hello.mf( 保存到 C:\1000)

**Manifest-Version: 1.0**

**Class: moon.start.TestOne moon.star.TestTwo**

**Created-By: 1.6**

### 2. 使用 jar 命令来生成一个名字为 Jerry.jar 的文件

**C:\1000> jar cfm Jerry.jar hello.mf moon\star\TestOne.class  
moon\star\TestTwo.class**



## 4.17 应用举例

□ 有理数有两个重要的成员：分子和分母，另外还有重要的四则运算。我们用 Rational 类实现对有理数的封装

例 22 给出了 Rational 类的代码

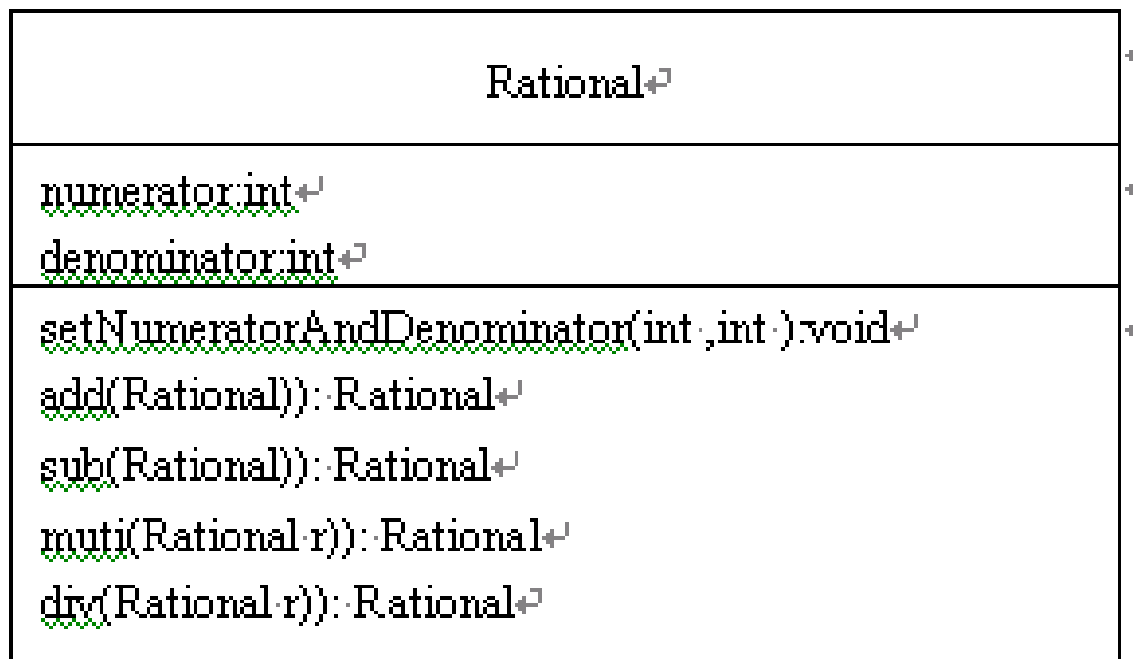


图 4.31 · Rational 类的 UML 图





## 4.17 应用举例

- 例 23 中的 Example4\_23.java 的主类使用 Rational 对象计算两个分数的四则运算，并计算了  $2/1+3/2+5/3...$  的前 10 项和

程序的运行结果如下：

$$1/5+3/2 = 17/10$$

$$1/5-3/2 = -13/10$$

$$1/5\times 3/2 = 3/10$$

$$1/5\div 3/2 = 2/15$$

计算  $2/1+3/2+5/3+8/5+13/8...$  的前 10 项和 .

用分数表示：

$$998361233/60580520$$

用小数表示：

$$16.479905306194137$$





# 总结

- 类是组成 Java 源文件的基本元素
- 类体可以有两种重要的成员：成员变量和方法。
- 成员变量分为实例变量和类变量、成员方法分为实例方法和类方法
- 类方法只能操作类变量，当对象调用类方法时，方法中的成员变量一定都是类变量，也就是说该对象和所有的对象共享类变量
- 对象访问自己的变量以及调用方法受访问权限限制






# 作业

---

- 习题 4：第 3,4 题





## 4. 局部变量没有默认值

- 成员变量有默认值（见后面的 4.3 节），但局部变量没有默认值，因此在使用局部变量之前，必须保证**局部变量有具体的值**。例如：下列 InitError 类无法通过编译。

```
class InitError {  
    int x = 10, y;    //y 的默认值是 0  
    void f() {  
        int m;        //m 没有默认值，但编译无错误  
        x = y + m; // 无法通过编译  
    }  
}
```

### 3. 类方法和实例方法

- 类中的方法也可分为**实例方法**和**类方法**。
- 方法声明时，方法类型前面不加关键字 **static** 的是实例方法、加关键字 **static** 的是类方法。

```
class Test {  
    int x = 10,y;  
    void f() {  
        System.out.println(x);  
    }  
    static int add(int a,int b) // 类方法  
    { return a+b;  
    }  
    Test( ) // 构造函数  
    {  
    }  
}
```



# 4. 方法的分类

## 1) 构造方法

- 构造方法是一种特殊方法，它的名字必须与它所在的类的名字完全相同，而且没有类型。

## 2) 重载方法

- 方法重载的意思是：一个类中但这些方法的参数必须不同，是参数的类型不同。方法的返回
- 方法重载是一种多态的体现。

区分下面类中的方法：

```
class Test {  
    int x = 10, y;  
    int add() {  
        System.out.println(x);  
    }  
    float add(int a, int b)  
    { return a+b;  
    }  
    Test()  
    {  
    }  
}
```