



中国地质大学

China University of Geosciences

Java面向对象程序设计

第4章 类与对象



计算机学院





核心要点

1

面向对象

2

类的结构

3

类与对象的关系

4

对象的创建

5

方法的调用

6

给方法传递对象参数





核心要点

7

变量的作用域

8

this关键字

9

static关键字

10

类与对象的应用





1.1 编程语言的几个发展阶段

➤ 面向机器语言

计算机处理信息的早期语言是所谓的机器语言，使用机器语言进行程序设计需要面向机器来编写代码，即需要针对不同的机器编写诸如0101 1100这样的指令序列。

➤ 面向过程语言

随着计算机硬件功能的提高,在20世纪60年代出现了面向过程设计语言，如C语言等。用这些语言编程也称为面向过程编程。语言把代码组成叫做过程或函数的块。每个块的目标是完成某个任务。使用这些语言编写代码指令时，不必再去考虑机器指令的细节，只要按着具体语言的语法要求去编写“源文件”。

➤ 面向对象语言

基于对象的编程更加符合人的思维模式，使得编程人员更容易编写出易维护、易扩展和易复用的程序代码，更重要的是，面向对象编程鼓励创造性的程序设计。

– 面向对象编程主要体现下列三个特性：封装性；继承；多态



1.1 理念的改变

C语言的函数，代码块是程序执行时产生的一种行为，但是面向过程语言却没有为这种行为指定“主体”，即在程序运行期间，无法说明到底是“谁”具有这个行为、并负责执行了这个行为。也就是说，面向过程语言缺少了一个最本质的概念，那就是“对象”（就好像生活中说话没主语）

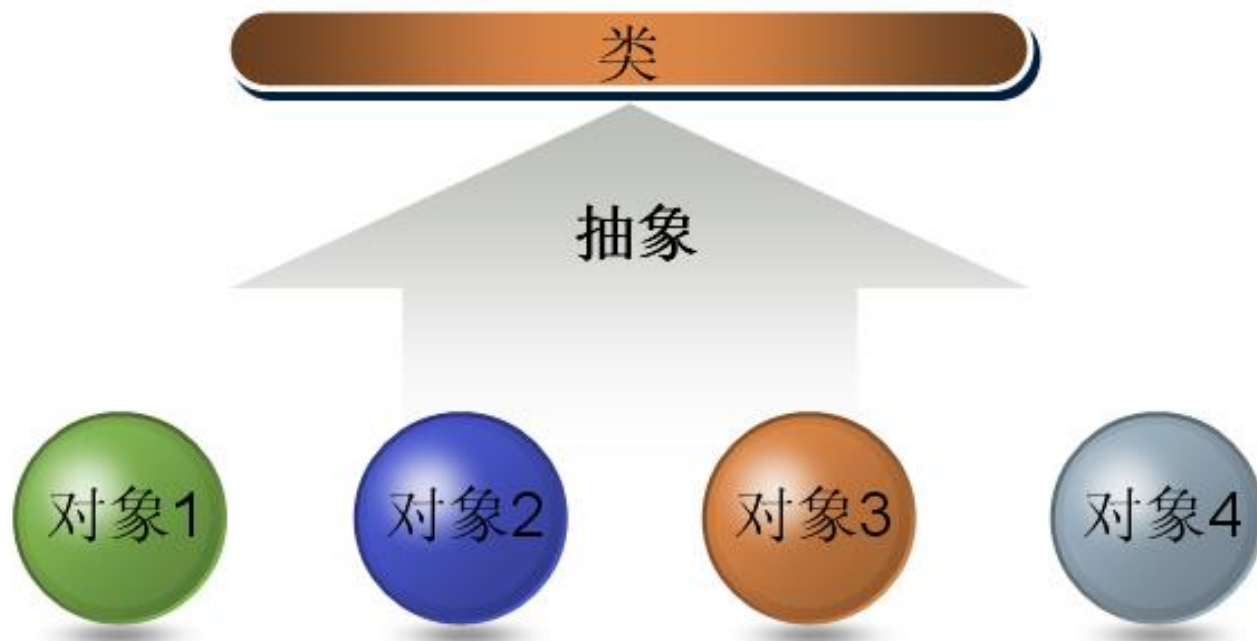
在面向对象语言中，最核心的内容就是“对象”，一切围绕着对象，比如，编写一个“刹车”方法（面向过程称之为函数），那么一定会指定该方法的“主体”，比如，某个汽车拥有这样的“刹车”方法，则该汽车负责执行“刹车”方法产生相应的行为（说话有主语：奔驰车刹车了）。

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。**



1.1 面向对象

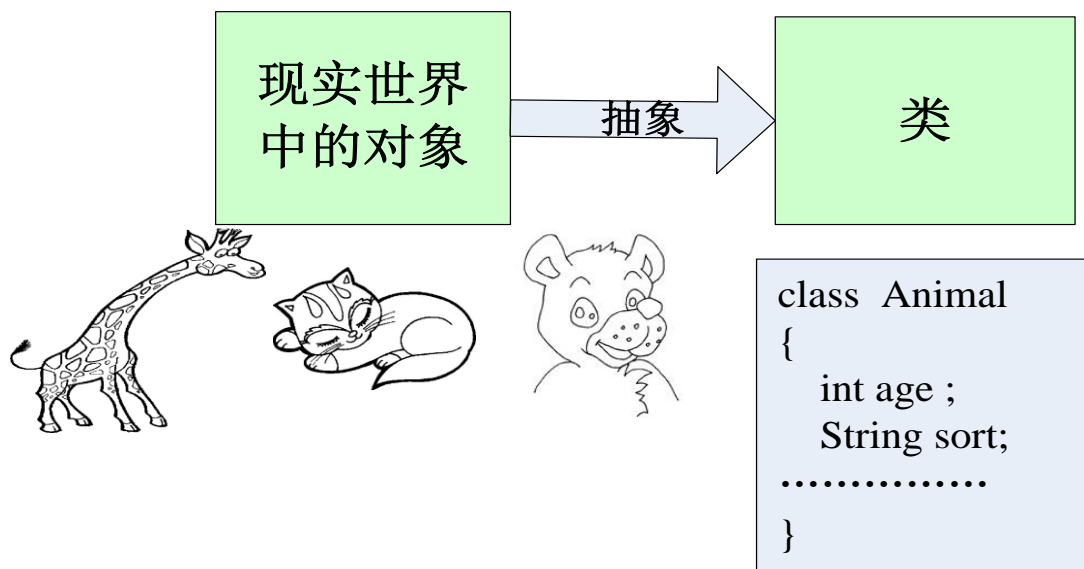
❖ 类的抽象





面向对象——对象、类和消息

- ❖ 万事万物皆**对象**
- ❖ 对相同类型的对象进行抽象就是**类**
- ❖ **消息**表示对象之间进行交互，以实现复杂的行为



面向对象的思想：按照现实世界的特点，管理复杂的事物，将事物抽象为对象。对象具有自己的状态和行为，通过对消息的反应，来完成一定的任务。



1.1 面向对象——特征

封装

数据及基于其上的**操作**被封装在对象的内部，对外通过一被授权的接口与程序其他部分交互。

继承

在**已有类**的基础上，扩充或改写其某些属性及方法，生成**新的类**，称为原有类的子类。

多态

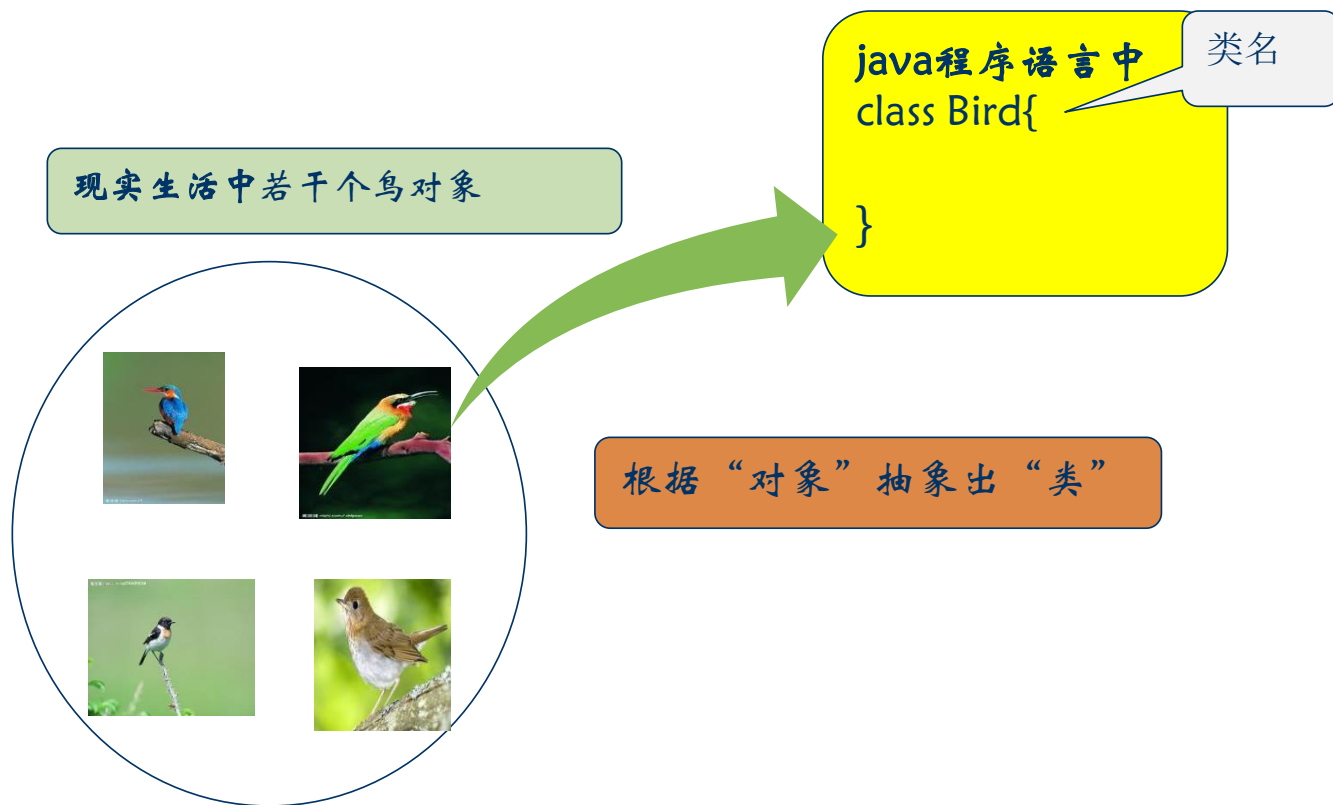
用相同的名字定义、调用不同的方法：1、子类对父类方法的**覆盖**
2、对本类中同名方法的**重载**





用面向对象的思想描述世界

第一步：发现类





用面向对象的思想描述世界

第二步：发现类的特征

名词

现实生活中

鸟类共有的特征：

1. 品种breed
2. 颜色color
3. 体重weight
4. 羽毛feather

.....



面向对象程序语言中

```
class Bird{  
    String breed;  
    String color;  
    double weight;  
}
```

数据成员



用面向对象的思想描述世界

第三步：发现类的行为

动词

现实生活中

鸟类共有的行为

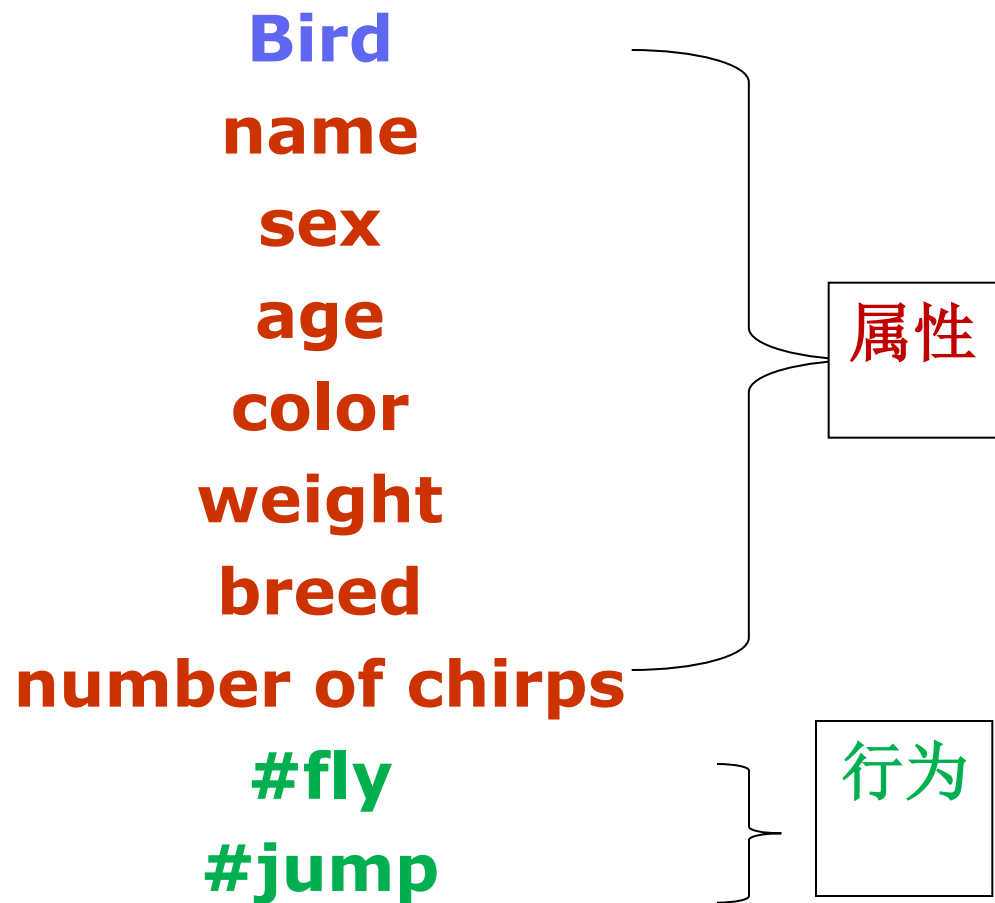
1. 鸣 chirp
 2. 飞 fly
 3. 攻击 strike
 4. 跳跃 jump
-



成员函数

面向对象程序语言中

```
class Bird{  
    String breed;  
    String color;  
    double weight;  
  
    void chirp() {  
        //输出“鸣”的代码  
    }  
    void fly () {  
        //输出“飞”的代码  
    }  
    void strike() {  
        //输出“攻击”的代码  
    }  
    .....  
}
```





面向对象程序中

- ❖ 把对象的特征由各种数据类型组合在一起，行为用方法表述出来，将这两种表述组合在一起。

Bird

String name
boolean isMale
String color
double weight
String breed
int numbchirps;
#void fly()
#void jump()

数据

方法



Bird类的对象们



.....

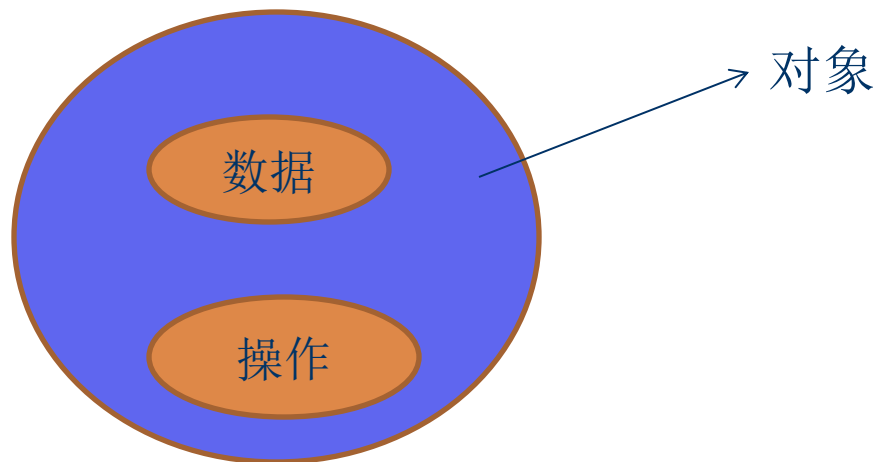


.....



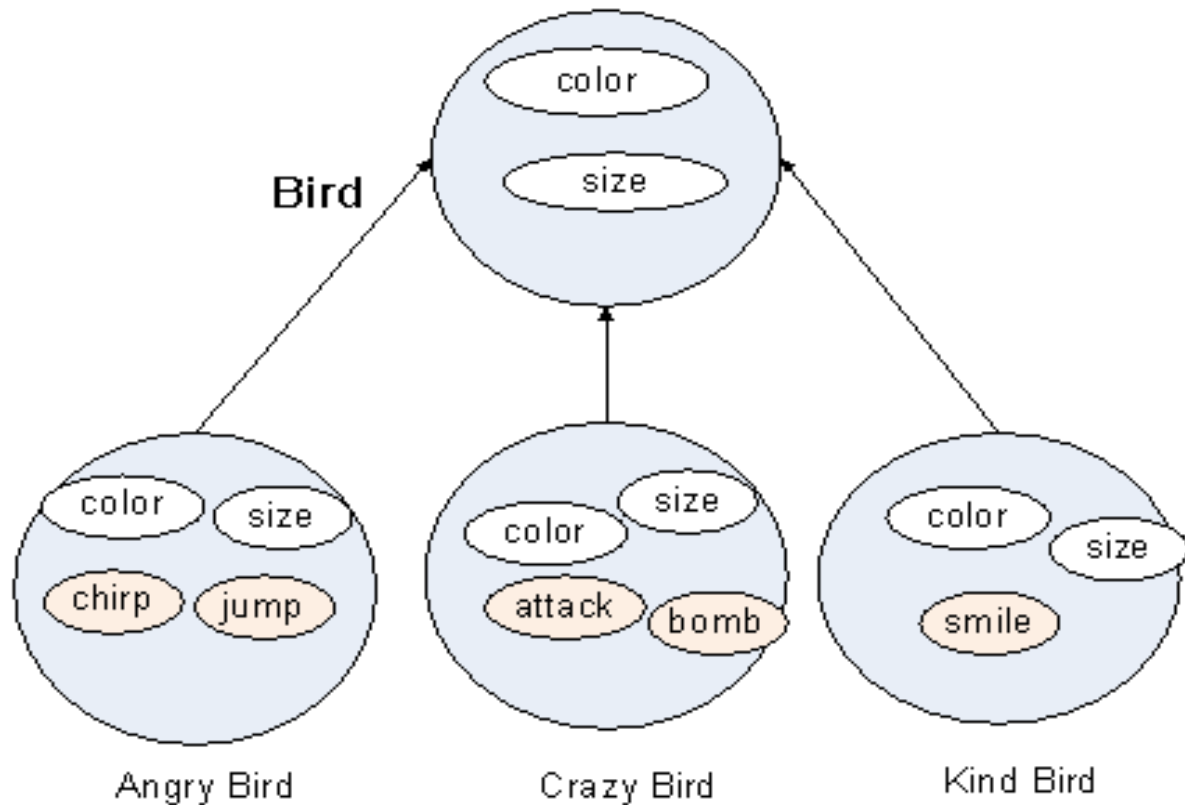
面向对象程序设计的三大特征

❖ **封装**，将对象的**数据**和**基于数据**
的操作封装成一个独立性很强的模
块



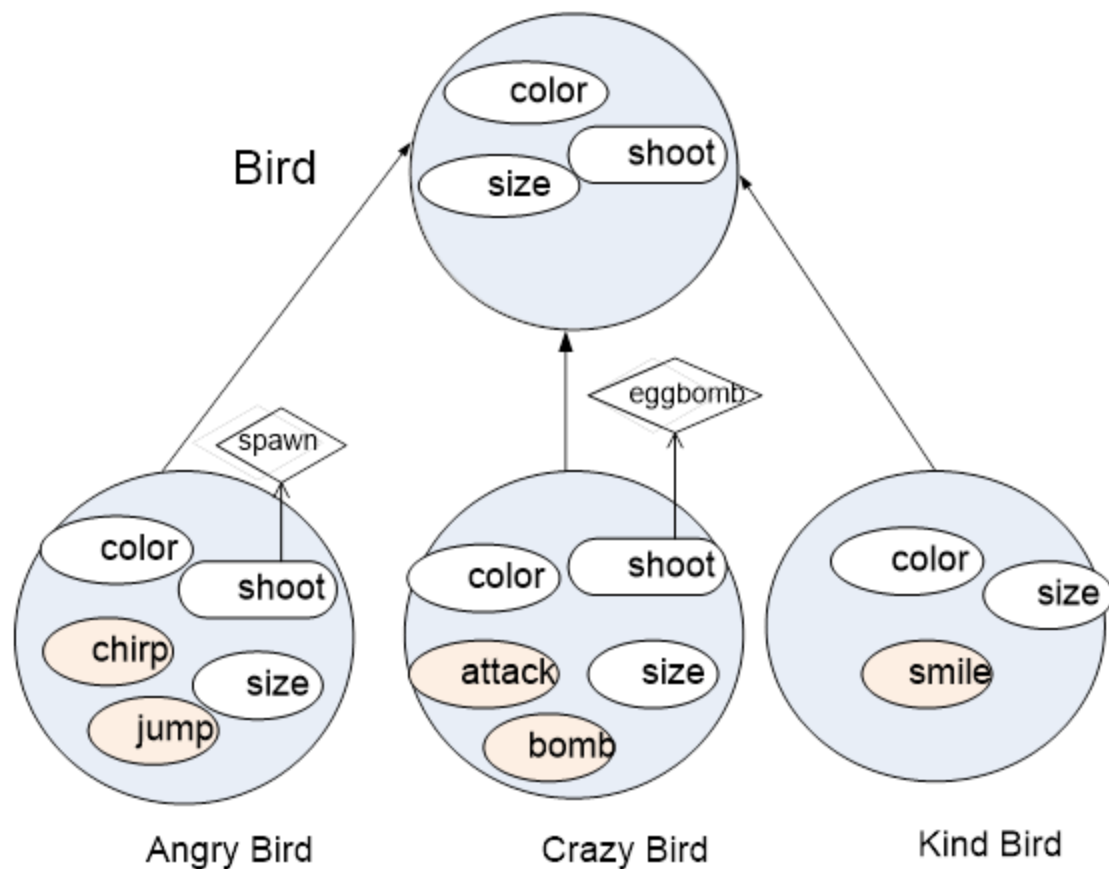


❖ 继承，在当前类的基础上创建新类，在其中添加新的属性和功能。当前类与新类之间是一种一般性与特殊性的关系。





❖ 多态，一个程序中同名的不同方法共存，子类的对象可以响应同名的方法，具体的实现方法却不同，完成的功能也不同。

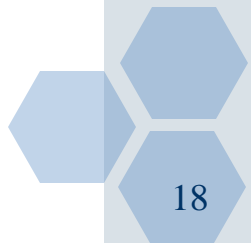




面向对象的程序设计

1.3.1 面向对象软件开发的基本流程：

- ❖ **第一阶段：**面向对象需求分析 (Object Oriented Analysis, OOA);
- ❖ **第二阶段：**面向对象设计 (Object Oriented Design, OOD);
- ❖ **第三阶段：**面向对象编程 (Object Oriented Programming, OOP);
- ❖ **第四阶段：**面向对象测试 (Object Oriented Test, OOT)





2. 类的结构

- 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。
- 类的实现包括两部分：类声明和类体。基本格式为：

```
class 类名 {  
    类体的内容  
}
```

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。





2. 类的结构

- 写类的目的是为了描述一类事物共有的属性和功能。
- 类声明：**class** 类名
- 以下是两个类声明的例子。

```
class People {
```

```
...
```

```
}
```

```
class 植物 {
```

```
...
```

```
}
```

如：class People”和“class 植物”称作类声明；

“People”和“动物”分别是类名。

- 给类命名时，遵守下列编程风格(这不是语法要求,但应当遵守):
 - 1.如果类名使用拉丁字母，那么名字的首字母使用大写字母。
 - 2.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。



2. 类的结构

❖ 定义的语法:

```
[修饰符] class 类名  
{  
    属性定义（声明）  
    方法定义（声明）  
}
```

类的成员有两种

在类里，属性称为类的数据成员，方法称为类的成员方法。





2. 类的结构

例题 编写一个员工类。

```
public class Employee{
```

```
    //属性声明
```

```
    String name;
```

```
    int age;
```

```
    double salary;
```

```
    //方法声明
```

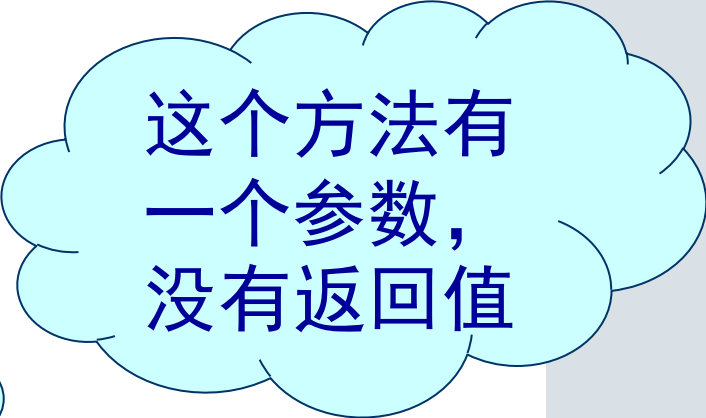
```
    void raise(double p){
```

```
        salary = salary + p;
```

```
        System.out.println(name + "涨工资之后的工资为: "  
+ salary);
```

```
    }
```

```
}
```



这个方法有一个参数，
没有返回值





属性和方法的声明

- ❖ 属性声明的语法:
- ❖ **[修饰符] 类型 属性名[= 初值];**
 - 类型可以是任何类型，包括类。
 - 属性有默认值。

如: **String name;**
int length;





属性和方法的声明

❖ 方法声明的语法:

[修饰符] 返回值类型 方法名(参数列表)

{

语句（组）；

return语句；

}

如: **public void sing(){**

// I can sing;

}





属性和方法的声明

- ❖ 修饰符是可选的；
- ❖ 方法可以返回一个值。
 - 1、返回值类型是方法要返回的值的数据类型，使用关键字`return`返回一个值。
 - 2、若方法不返回值，则返回值类型为`void`。
- ❖ 方法可以有一个参数列表，按方法的规范称为形式参数。当方法被调用时，形式参数用变量或数据替换，这些变量或数据称为实际参数。





变量的作用域

属性的
作用域

```
public class Employee {  
    String name;  
    int    age;  
    double salary;  
    void raise(double p){  
        salary=salary+p;  
        System.out.println(name+  
            "涨工资之后的工资为: " + salary);  
    }  
}
```

形参
的作用域

局部
变量i
的作用域

```
public class Narcissus {  
    public static void main(String args[]) {  
        for (int i = 100; i < 1000; i++) {  
            int a = i % 10;  
            int b = (i / 10) % 10;  
            int c = i / 100;  
            if (a*a*a + b*b*b + c*c*c == i)  
                System.out.println(i);  
        }  
    }  
}
```

局部
变量b
的作用域





变量的作用域

例题:

```
public class Example4_6 {  
    int x = 0;  
  
    int y = 0;  
  
    void method() {  
        int x = 1;  
        System.out.println("x=" + x);  
        System.out.println("y=" + y);  
    }  
  
    public static void main(String[] args) {  
        Example4_6 e = new Example4_6();  
        e.method();  
    }  
}
```

@ Javadoc Declaration Console

<terminated> Example4_6 [Java Application] C:\Program Files\Java\jre1.8.0_65\bin\javaw.e

x=1

y=0





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**。

➤ 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。





成员变量与局部变量

➤ 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;
}
```





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  
    }  
}
```

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**（在4.9节**this关键字**）

```
class Tom {  
    int x = 10,y;  
    void f() {  
        int x = 5;  
        y = x+this.x; //y得到的值是15  
    }  
}
```



局部变量没有默认值

- 成员变量有默认值，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。例如：

```
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，因为在使用m之前未指定m的值  
    }  
}
```





成员变量与局部变量

- 和类的成员变量不同的是，局部变量只在方法内有效，而且与其声明的位置有关。

```
public class A {  
    int m = 10, sum = 0; //成员变量，在整个类中有效  
    void f() {  
        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  
    }  
}
```





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)的UML图以及泛化关系(Generalization)的UML图、关联关系(Association)的UML图、依赖关系(Dependency)的UML图和实现关系(Realization)的UML图。

➤ 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。

- 第1层是名字层；
- 第2层是变量层，也称属性层；
- 第3层是方法层，也称操作层。

Lader
above: float bottom:float height:float
computer():float setHeight(float):float



练习1

1. 定义一个圆类**Circle**，描述圆对象。圆有一个属性**radius**表示半径，有一个**findArea()**方法用于计算圆的面积。





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。方法的返回类型可以相同也可以不同。
- 方法重载是一种多态的体现。

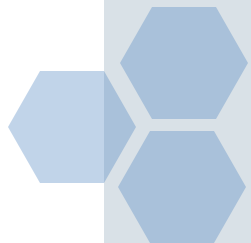
区分下面类中的方法：

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



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**，那么就可以用它来声明变量。在面向对象语言中，用**类声明的变量被称作对象**。
- 和基本数据类型不同，在**用类声明对象**后，还**必须要创建对象**，即为声明的对象分配变量(确定对象所具有的属性)，当使用一个类创建一个对象时，也称给出了这个类的一个实例。通俗的讲，类是创建对象的“模板”，没有类就没有对象。构造方法和对象的创建密切相关。





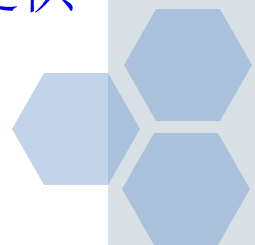
构造方法

➤ 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。
- 允许一个类中编写若干个构造方法，但必须保证他们的参数不同，即参数的个数不同，或者是参数的类型不同。
- 他的作用是在创建对象时使用，主要是用来初始化各个成员变量，以便给类所创建的对象一个合理的初始状态。

➤ 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。

- 如果类里定义了一个或多个构造方法，那么Java不提供默认的构造方法。
- 需要特别注意的是，构造方法没有类型。





构造方法

- ✧ 用于对象的初始化
- ✧ 构造方法的结构：

【修饰符】 类名（参数列表）

{

//方法体

}





构造方法

```
public Employee()
```

```
{  
    name="小明";  
    age=32;  
    salary=2000;  
}
```

不带参的构造方法

带参的构造方法

```
public Employee(String n,int a,double s)  
{  
    name=n;  
    age=a;  
    salary=s;  
}
```





构造方法

❖ 家中养了3只小狗，为了构造他们方便，我的做法如下：

```
public class Dog{  
    String name;  
    int size;  
    Dog(){  
        name="fofo";  
        size=20;  
    }  
    Dog(String n,int s){  
        name=n;  
        size=s;  
    }  
    void bark(){  
        System.out.println( name + "ruff,ruff!!");  
    }  
}
```





构造方法

- ❖ 作用：构造方法用于对象的初始化；
- ❖ 与一般方法的区别：
 1. 构造方法名与类名一致
 2. 构造方法没有返回值类型
 3. 如果没有定义构造方法，系统会生成一个默认的无参的构造方法；但是如果有构造方法,系统就不会再提供默认构造方法
 4. 构造方法只能用**new**在创建对象时调用，不能通过对象名调用





类的构造方法

■ 类的构造方法介绍

什么是构造方法呢?在回答这个问题之前,我们来看一个需求:前面我们在创建人类的对象时,是先把一个对象创建好后,再给他的年龄和姓名属性赋值,如果现在我要求,在创建人类的对象时,就直接指定这个对象的年龄和姓名,该怎么做?



你可以在定义类的时候,定义一个构造方法即可。

构造方法是类的一种特殊的方法,它的主要作用是完成对新对象的初始化。它有几个特点:

- ①方法名和类名相同
- ②没有返回值
- ③在创建一个类的新对象时,系统会自动的调用该类的构造方法完成对新对象的初始化。





类的构造方法

■ 类的默认构造方法

有些同志可能会问？亲爱的老师，我们在没有学习构造函数前不是也可以创建对象吗？

是这样的，如果程序员没有定义构造方法，系统会自动生成一个默认构造方法。比如Person类

```
Person () {  
};
```

当创建一个Person对象时 `Person per1=new Person();`
默认的构造函数就会被自动的调用。

■ 类的构造方法小结

我来总结



- ①构造方法名和类名相同
- ②构造方法没有返回值
- ③主要作用是完成对新对象的初始化
- ④在创建新对象时，系统自动的调用该类的构造方法
- ⑤一个类可以有多个构造方法
- ⑥每个类都有一个默认的构造方法



3.类与对象的关系

```
public class Dog{  
    String name;  
    public void run(){  
        // the dog can run.  
    }  
}
```

抽象





3.类与对象的关系

❖ 类和对象的关系

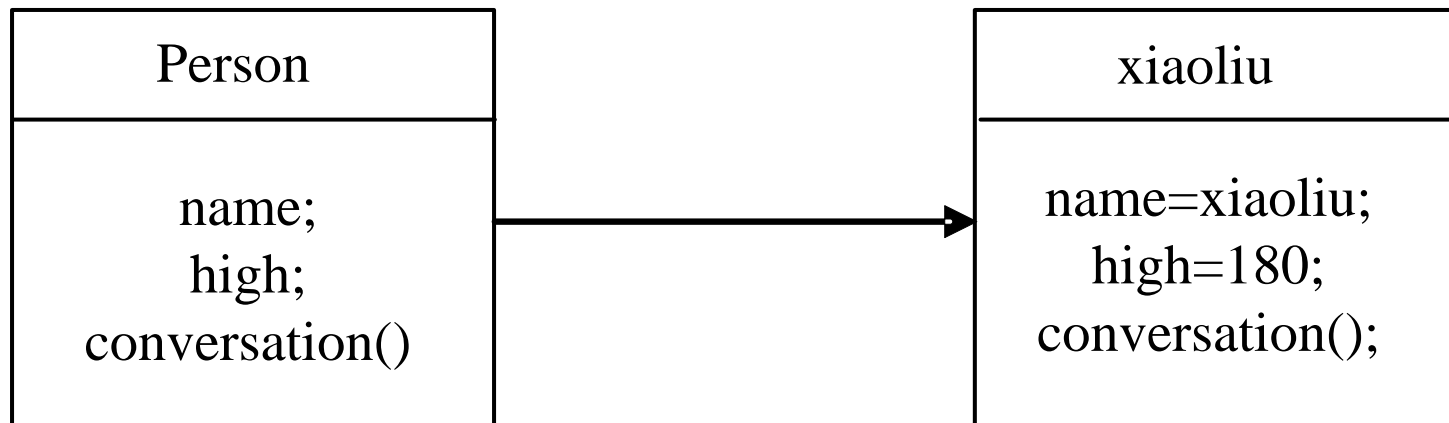


- ❖ 注意：从狗类到对象，目前有几种说法：**1.** 创建一个对象。**2.** 实例化一个对象。**3.** 把类实例化...大家听到这些说法，不要迷糊。
- ❖ 当然，上面的狗也可以是鱼，猫，人...**java**最大的特点就是面向对象。





3.类与对象的关系



- ①类是抽象的，概念的，代表一类事物，
比如人类，猫类……
- ②对象是具体的，实际的，代表一个事物。
- ③类是对象的模板，对象是类的一个个体。





4.对象的创建

- 创建一个对象包括对象的声明和为对象分配变量两个步骤。

1.对象的声明

一般格式为： 类的名字 对象名字;

如： **XiyoujiRenwu zhubajie;**

2.为声明的对象分配变量

- 使用**new运算符**和一个**引用值**给对象。系统会调用默认的方法体中没有语句。
 - 例如：**zhubajie =**
- 当用类创建一个对象时，这些内存空间称作放着引用值。

```
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();  
    }  
}
```

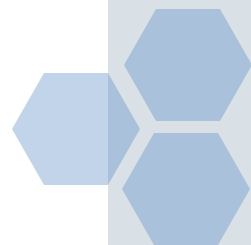


4. 创建对象

```
public class DogTestDrive{  
    public static void main(String[] args){  
        Dog d1=new Dog();  
  
        Dog d2=new Dog("pp",30);  
    }  
}
```

调用默认的
构造方法

调用带参数
的构造方法





4. 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

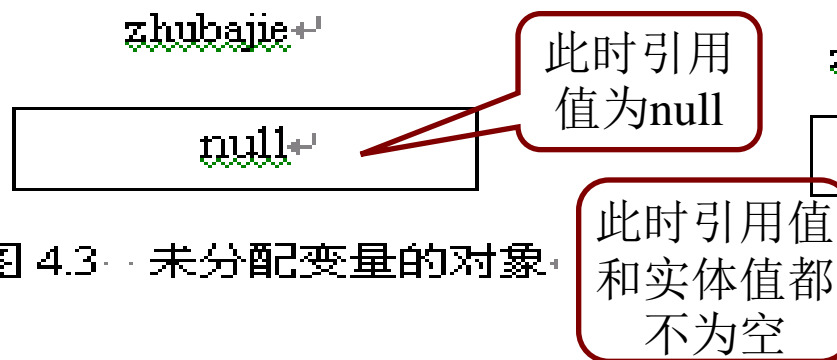


图 4.3 · 未分配变量的对象

(2) 对象分配变量后的内存模型

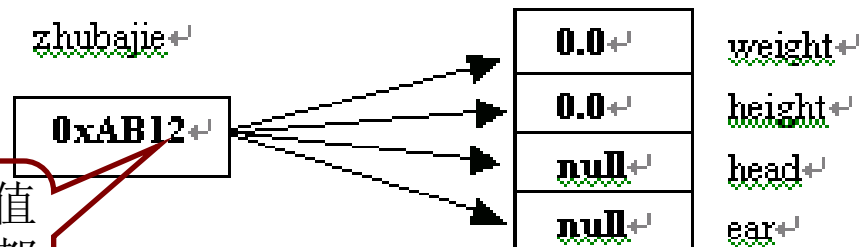


图 4.4 · 分配变量（实体）后的对象

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；但有时也有大家共有的空间和成员。



4. 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

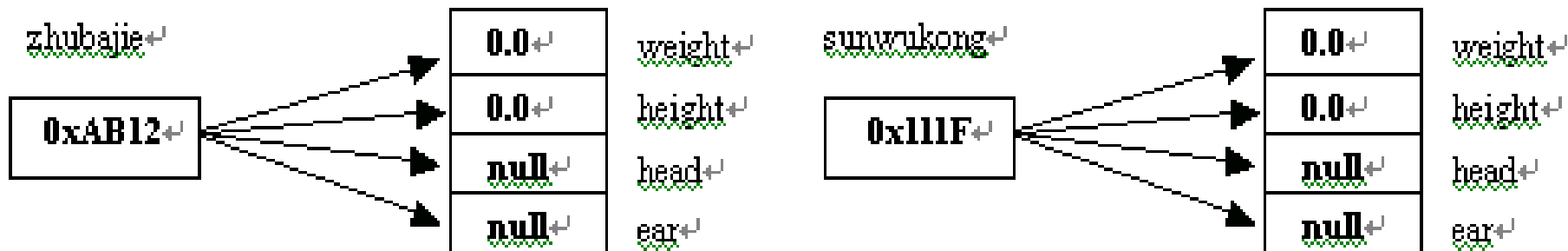


图 4.5 · 创建多个对象



4.对象的创建

■ 一个小思考题



为了让大家加深印象，我们定义一个人类(**Person**)(包括 名字,年龄)。用一步到位法去创建一个对象 (**demo.java**)



我们看看下面一段代码:
`Person a=new Person();`
`a.age=10;`
`a.name="小明";`
`Person b;`
`b=a;`

`System.out.println(b.age);`
请问: `b.age`究竟是多少?

```
//定义个人类
class Person
{
    int age;
    String name;
}
```

```
Person a=new Person();
a.age=10;
a.name="小明";
Person b;
b=a;
System.out.println(b.age);
```

```
D:\myJavaDemo>javac Demo4.java
D:\myJavaDemo>java Demo4
10
```



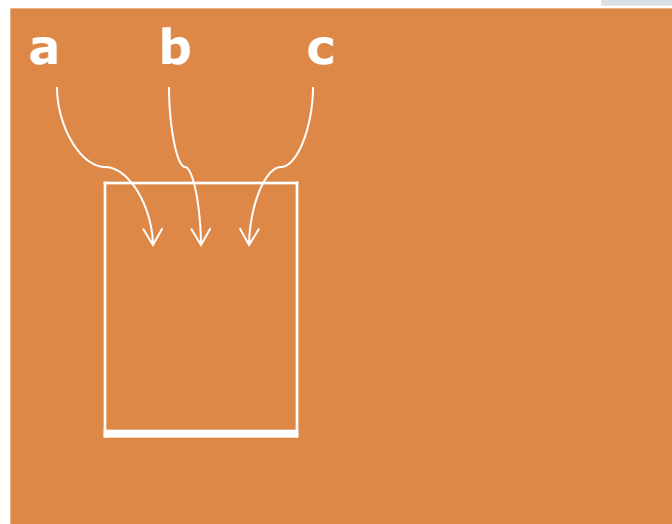
4.对象的创建

❖ 对象总是存在内存中，那么对象在内存中是怎样存在的？

```
Person a=new Person();  
a.age=10;  
a.name="小明";  
Person b;  
b=a;  
System.out.println(b.age);  
Person c;  
c=b;  
System.out.println("c.name="+c.name);  
//通过c.age去修改值。  
c.age=9;  
  
System.out.println("a.age="+a.age);  
System.out.println("b.age="+b.age);  
System.out.println("c.age="+c.age);
```

```
D:\myJavaDemo>java Demo4  
10  
c.name=小明  
a.age=9  
b.age=9  
c.age=9
```

内存





5. 方法的调用

❖ 调用对象属性和方法的语法:

- 对象名.属性名
- 对象名.方法名()



5. 方法的调用

是重点，
也是难点



■ 类-成员方法的初步介绍

在某些情况下，我们需要定义成员方法。比如人类：除了有一些属性外（成员变量表示的 年龄，姓名..），我们人类还有一些行为比如：可以说话、跑步..，通过学习，我们人类还可以做算术题。这时就要用成员方法才能完成。现在要求对Person类完善：

- ①添加speak 成员方法，输出 我是一个好人
- ②添加jisuan 成员方法，可以计算从 $1+..+1000$ 的结果
- ③修改jisuan 成员方法，该方法可以接收一个数n，计算从 $1+..+n$ 的结果
- ④添加add 成员方法，可以计算两个数的和

画图说明程序执行过程：



5. 方法的调用

```
class Person
{
    int age;
    String name;
    //1. 可以输出我是好人
    public void speak()
    {
        System.out.println("我是一个好人");
    }

    //可以计算
    public void jiSuan()
    {
        int result=0;

        for(int i=1;i<=1000;i++)
        {
            result=result+i;
        }

        //输出结果
        System.out.println("结果是"+result);
    }

    //带参数的成员方法
    public void jiSuan(int n)
    {
        int result=0;
        for(int i=1;i<=n;i++)
        {
            result+=i;
        }
        System.out.println("结果是:"+result);
    }
}
```

```
Person p1=new Person();
//调用speak方法
p1.speak();
//调用计算方法
p1.jiSuan();
//调用可以传入参数的计算方法
p1.jiSuan(100);
```

```
D:\myJavaDemo>java Demo4
我是一个好人
结果是500500
结果是:5050
```



5.属性和方法的使用

例题

```
public class Example 4_3{  
    public static void main(String[] args) {  
        Employee e1=new Employee();  
        e1.Employee(); //错了  
        e1.name="王一";  
        e1.salary=1600;  
        e1.raise(100);  
        Employee e2=new Employee("张敏  
",29,3000);  
        e2.raise(500);  
    }  
}
```





练习

- ❖ 定义一个测试**Circle**类的测试类**CircleTest**，在其**main**方法中创建一个半径为**5**的圆，并将其面积输出到屏幕上。





练习

- ❖ 定义一个笔记本类，该类有：
 - 品牌（**String**）和状态（**int**）两个属性
 - 有无参和有参的两个构造方法
- ❖ 然后编写一个测试类，测试笔记本类的各个方法。
 -





布雷克·罗斯 Blake Ross

- 1985年出生在美国。
- 七岁迷上电玩“虚拟城市”（SimCity）
- 十岁起就自行架设了网站、并设计自己所需的应用程序与网络游戏
- 火狐（Firefox）浏览器作者，自发行以来，已经约1500万用户下载，火狐成为世界最受欢迎的网络浏览器之一。

19岁开发出火狐浏览器！

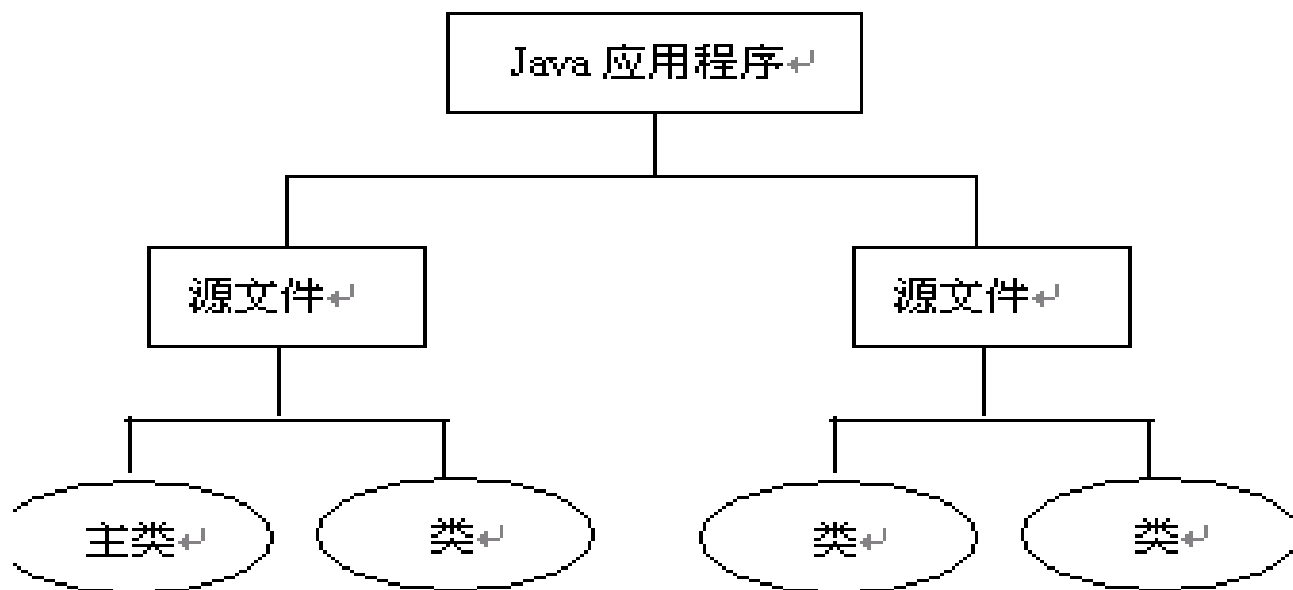


Firefox®



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，如图所示。



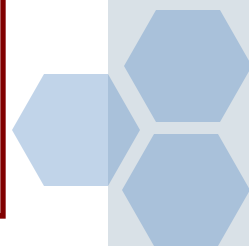
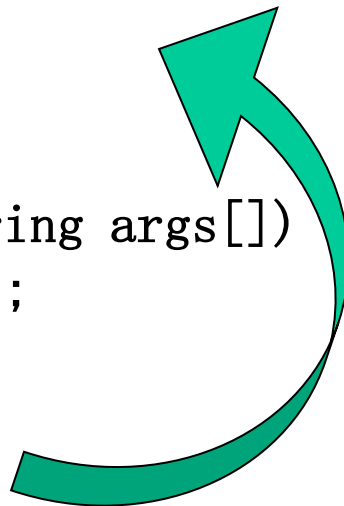


6. 给方法传递对象参数

- 方法中最重要的部分之一就是方法的参数，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);
    }
}
```





6. 给方法传递对象参数

➤ 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。

1) 对于基本数据类型的参数，向该参数“传值”，传递的是值的拷贝。

例如，如果向方法的int型参数x传递一个int值，那么参数x得到的值是传递的值的拷贝。

2) 对于参数是引用类型时，“传值”传递的是变量的引用而不是变量所引用的实体。Java的引用型数据包括对象、数组和接口。



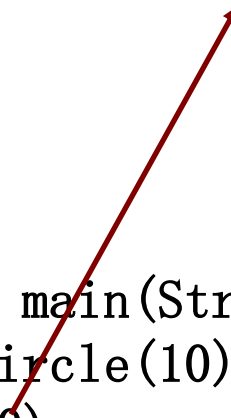


6. 给方法传递对象参数

- 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);
    }
}
```

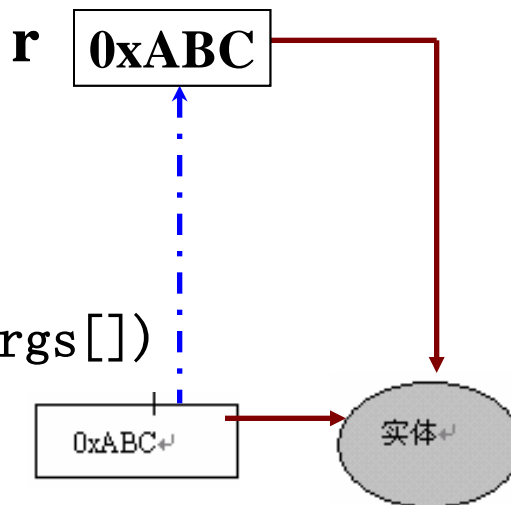


6. 给方法传递对象参数

- 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。如图4.13所示

```
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);
    }
}
```





6. 给方法传递对象参数

❖ 例题

```
class A {  
    int a;  
    public A() {  
        a = 1;  
    }  
    public void add(int m, A n) {  
        m++;  
        n.a++;  
    }  
}  
  
public class TestPassObject {  
    public static void main(String[] args) {  
        int x = 5;  
        A y = new A();  
        System.out.println("调用前简单类型变量x="+x);  
        System.out.println("调用前引用类型变量y的属性y.a="+y.a);  
        y.add(x, y);  
        System.out.println("调用后简单类型变量x="+x);  
        System.out.println("调用后引用类型变量y的属性y.a="+y.a);  
    }  
}
```

<terminated> TestPassObject [Java Applica

调用前简单类型变量x=5

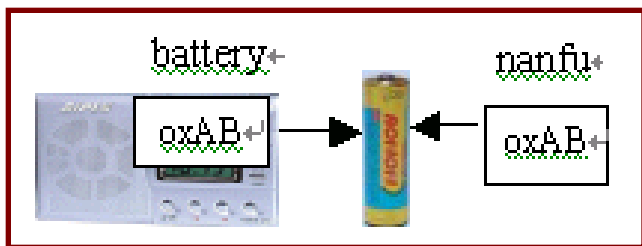
调用前引用类型变量y的属性y.a=1

调用后简单类型变量x=5

调用后引用类型变量y的属性y.a=2

6. 给方法传递对象参数

- 例子7模拟收音机使用电池。例子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



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



6. 给方法传递对象参数

Battery.java

```
1 public class Battery {
2     int electricityAmount;
3     Battery(int amount){
4         electricityAmount = amount;
5     }
6 }
```

Radio.java

```
1 public class Radio {
2     void openRadio(Battery battery){
3         battery.electricityAmount = battery.electricityAmount - 10;
4     }
5 }
```

Example4_7.java

```
1 public class Example4_7 {
2     public static void main(String args[]) {
3         Battery nanfu = new Battery(100);
4         System.out.println("南孚电池的储电量是:"+nanfu.electricityAmount);
5         Radio radio = new Radio();
6         System.out.println("收音机开始使用南孚电池");
7         radio.openRadio(nanfu);
8         System.out.println("目前南孚电池的储电量是:"+nanfu.electricityAmount);
9     }
10 }
```





可变参数

可变参数（**The variable arguments**）数是指在声明方法时不给出参数列表中从某项开始直至最后一项参数的名字和个数，但这些参数的类型必须相同。可变参数使用“...”表示若干个参数，这些参数的类型必须相同，并且最后一个参数必须是方法的参数列表中的最后一个参数。例如：

```
public void f(int ... x)
```

参数代表可以通过下标运算来表示参数列表中的具体参数，即 $x[0]$ ， $x[1] \dots x[m-1]$ 分别表示 x 代表的第1个至第 m 个参数。

```
public int getSum(int... x) { //x可变参数的参数代表
    int sum=0;
    for(int i=0;i<x.length;i++) {
        sum=sum+x[i];
    }
    return sum;
}
```

getSum (203,178,56,2098)返回
203,178,56,2098的求和结果，
getSum (1,2,3) 返回1,2,3的求和结果。



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。

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。

通过组合对象来复用方法有以下特点。

（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，这样一来，当前对象对所包含的对象的方法的细节（算法的细节）是一无所知的。

（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。

注 在学习对象的组合时，一定要记住：**一个类声明的两个对象如果具有相同的引用，二者就具有完全相同的变量。**





对象的组合

例子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。





对象的组合

➤ 例子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:523.3333333333334
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
```





对象的组合

Circle.java

```
1 public class Circle {
2     double radius,area;
3     void setRadius(double r) {
4         radius=r;
5     }
6     double getRadius() {
7         return radius;
8     }
9     double getArea(){
10        area=3.14*radius*radius;
11        return area;
12    }
13 }
```

Circular.java

```
1 public class Circular {
2     Circle bottom;
3     double height;
4     void setBottom(Circle c) {
5         bottom = c;
6     }
7     void setHeight(double h) {
8         height = h;
9     }
10    double getVolme() {
11        if(bottom == null)
12            return -1;
13        else
14            return bottom.getArea()*height/3.0;
15    }
16    double getBottomRadius() {
17        return bottom.getRadius();
18    }
19    public void setBottomRadius(double r){
20        bottom.setRadius(r);
21    }
22 }
```

Example4_8.java

```
1 public class Example4_8 {
2     public static void main(String args[]) {
3         Circle circle = new Circle();           //【代码1】
4         circle.setRadius(10);                   //【代码2】
5         Circular circular = new Circular();      //【代码3】
6         System.out.println("circle的引用:"+circle);
7         System.out.println("圆锥的bottom的引用:"+circular.bottom);
8         circular.setHeight(5);
9         circular.setBottom(circle);              //【代码4】
10        System.out.println("circle的引用:"+circle);
11        System.out.println("圆锥的bottom的引用:"+circular.bottom);
12        System.out.println("圆锥的体积:"+circular.getVolme());
13        System.out.println("修改circle的半径, bottom的半径同样变化");
14        circle.setRadius(20);                    //【代码5】
15        System.out.println("bottom的半径:"+circular.getBottomRadius());
16        System.out.println("重新创建circle,cirlce的引用将发生变化");
17        circle = new Circle(); //重新创建circle 【代码6】
18        System.out.println("circle的引用:"+circle);
19        System.out.println("但是不影响circular的bottom的引用");
20        System.out.println("圆锥的bottom的引用:"+circular.bottom);
21    }
22 }
```





对象的组合

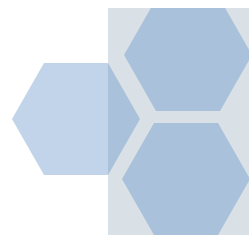
- 如果一个对象a组合了对象b，那么对象a就可以委托对象b调用其方法，即对象**a以组合的方式复用对象b的方法**。

例子9 模拟手机和SIM卡的组合关系。涉及的类如下：

- SIM类负责创建SIM卡 [SIM.java](#)。
- MobileTelephone类负责创建手机 [MobileTelephone.java](#)，手机可以组合一个SIM卡，并可以调用setSIM (SIM card) 方法更改其中的SIM卡。程序运行效果如图。

[SIM.java](#) , [MobileTelephone.java](#) , [Example4_9.java](#)

```
手机号码:13889776509  
手机号码:15967563567
```





对象的组合

SIM.java

```
1 public class SIM {
2     long number;
3     SIM(long number){
4         this.number = number;
5     }
6     long getNumber() {
7         return number;
8     }
9 }
```

MobileTelephone.java

```
1 public class MobileTelephone {
2     SIM sim;
3     void setSIM(SIM card) {
4         sim = card;
5     }
6     long lookNumber(){
7         return sim.getNumber();
8     }
9 }
```

Example4_9.java

```
1 public class Example4_9 {
2     public static void main(String args[]) {
3         SIM simOne = new SIM(13889776509L);
4         MobileTelephone mobile = new MobileTelephone();
5         mobile.setSIM(simOne);
6         System.out.println("手机号码:"+mobile.lookNumber());
7         SIM simTwo = new SIM(15967563567L);
8         mobile.setSIM(simTwo);
9         System.out.println("手机号码:"+mobile.lookNumber());
10    }
11 }
```



关联关系和依赖关系的UML图

1 关联关系

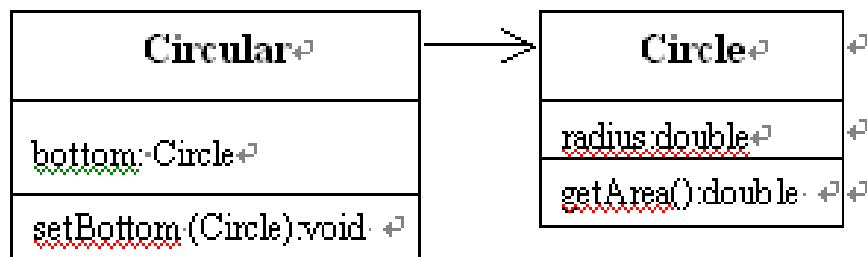


图 4.21 · 关联关系的 UML 图

2 依赖关系

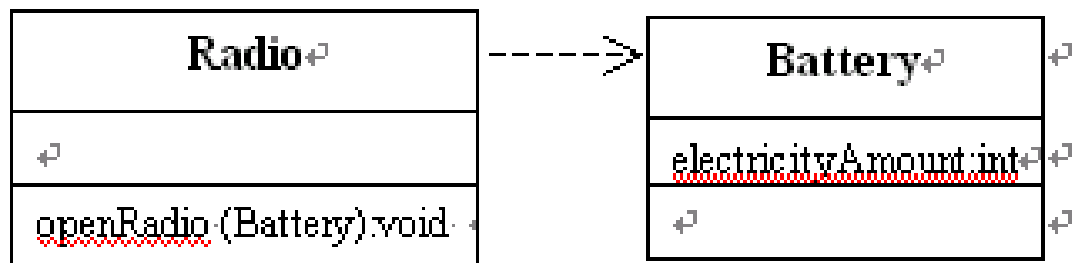


图 4.22 · 依赖关系的 UML 图





8. this关键字

❖ **this**代表类的当前对象

- 利用“this.”可以调用当前对象的成员

❖ **this**可以实现构造方法的调用

- 利用this()可以调用构造方法
- 必须写在构造方法的第一条





8. this关键字

例题

```
public class Platypus {  
    String name;  
    Platypus(String name){  
        this.name = name;  
    }  
  
    Platypus(){  
        this("John/Mary Doe");  
    }  
  
    public static void main(String args[]){  
        Platypus p1 = new Platypus("digger");  
        Platypus p2 = new Platypus();  
    }  
}
```

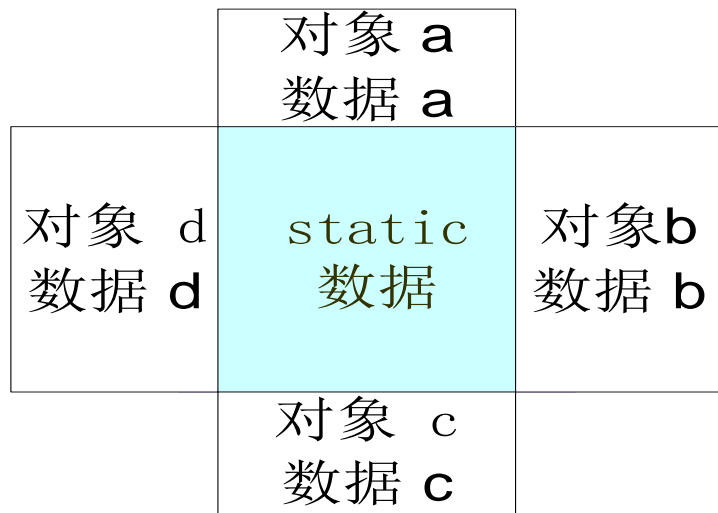




9. static关键字

❖ 类属性（静态属性）

- 用static修饰的属性，称为类属性（不是类的属性）
- 用于描述一个类下所有对象共享的属性
- 比如：员工的最低工资，学生的学校名称等等
- 可通过类名直接调用，也可通过对象调用





9. static关键字

❖ 类方法（静态方法）

- 用**static**修饰的方法，叫类方法
- 静态方法中不能访问非静态**成员**(属性和方法)
- 非静态的方法可以调用静态**成员**(属性和方法)
- 如果一个方法中没有访问非静态成员，则这个方法可以声明成静态的。





9. static关键字

❖ 使用

- 在类外，静态属性和静态方法可以通过类名直接调用，也可以通过对象名调用。
- 在类外，非静态属性和非静态方法只能通过对象名调用。





9. static

❖ 类方法的创建:

```
static void setMin(double min)  
{    min_salary=min;    }
```

❖ 类方法的使用:

```
Employee .setMin(600);  
Employee e1=new Employee ();  
e1 .setMin(600);
```





9. static

❖ 例题

```
public class Example4_9 {  
    public static void main(String[] args) {  
        Employee2.setMin(600);  
        Employee2 e1 = new Employee2("张三", 29, 3000);  
        System.out.println("e1中员工最低工资: " + e1.getMin());  
        Employee2 e2 = new Employee2("李四", 22, 300);  
        System.out.println("e2中员工最低工资: " + e2.getMin());  
        e1.raise(500);  
        e2.raise(400);  
    }  
}
```

```
class Employee2 {  
    String name;  
    int age;  
    double salary;  
    static double min_salary;  
    public Employee2(String n, int a, double s){  
        name = n;  
        age = a;  
        salary = s;  
    }  
    public static double getMin(){  
        return min_salary;  
    }  
    public static void setMin(double min){  
        min_salary = min;  
    }  
}
```

```
void raise(double p){  
    if(salary < min_salary)  
        salary = min_salary;  
    else  
        salary = salary+p;  
    System.out.println(name + "涨工资之后的工资为: " + salary);  
}
```

Example4_9 [Java Applicat

<terminated> Example4_9 [Java Applicat

e1中员工最低工资: 600.0

e2中员工最低工资: 600.0

张三涨工资之后的工资为: 3500.0

李四涨工资之后的工资为: 600.0

练习

计算学费总和:

```
public class Demo4_2 {  
    public static void main(String args[]){  
        //创建一个学生  
        Stu stu1=new Stu(29,"aa",340);  
        Stu stu2=new Stu(39,"bb",240);  
        System.out.println(stu2.getTotalFee());  
    }  
}  
//学生  
class Stu{  
    int age;  
    String name;  
    int fee;  
    static int totalFee;  
    public Stu(int age,String name,int fee){  
        this.age=age;  
        this.name=name;  
        totalFee+=fee;  
    }  
    public static int getTotalFee(){  
        return totalFee;  
    }  
}
```

@ Javadoc Declaration Console

<terminated> Demo4_2 [Java Application] C:\Program Files\Java\jre1.8.0_65\bin\javaw.exe [2

580

9. 类方法

■ 类方法小结



1. 什么时候需要用类方法

案例：定义学生类，统计学生共交多少钱？

- 类方法属于与类相关的，公共的方法
- 实例方法属于每个对象个体的方法
- 类方法可以通过类名.类方法名直接访问



9. 类变量

■ 什么是类变量?

类变量是该类的所有对象共享的变量,任何一个该类的对象去访问它时,取到的都是相同的值,同样任何一个该类的对象去修改它时,修改的也是同一个变量。这个从前面的图也可看出来。

■ 如何定义类变量?

定义语法:

访问修饰符 `static` 数据类型 变量名;

■ 如何访问类变量?

类名.类变量名 或者 对象名.类变量名



练习

- ❖ 有一群小孩在玩堆雪人，不时有新的小孩加入，请问如何知道现在共有多少人在玩？请使用面向对象的思想，编写程序解决。

```
public class Child {
    int age;
    String name;
    static int total=0;
    public Child(int age,String name){
        this.age=age;
        this.name=name;
    }
    public void joinGame(){
        total++;
        System.out.println("有一个小孩加入了");
    }
    public static void main(String[] args){
        Child ch1=new Child(3,"妞妞");
        ch1.joinGame();
        Child ch2=new Child(4,"小小");
        ch2.joinGame();
        Child ch3=new Child(4,"大大");
        ch3.joinGame();
        System.out.println("共有="+ch3.total);
    }
}
```

@ Javadoc Declaration Console

<terminated> Child [Java Application] C:\Program

有一个小孩加入了
有一个小孩加入了
有一个小孩加入了
共有=3

9. 类变量

■ 类变量小结



1. 什么时候需要用类变量

案例：定义学生类，统计学生共交多少钱？

用类变量，属于公共的属性

2. 类变量与实例变量区别：

- 加上static称为类变量或静态变量，否则称为**实例变量**
- 类变量是与类相关的，公共的属性
- 实例变量属于每个对象个体的属性
- 类变量可以通过类名.类变量名直接访问



总结：实例成员与类成员

1 实例变量和类变量的声明

- 在声明成员变量时，用**关键字static给予修饰**的称作**类变量**，否则称作实例变量（类变量也称为static变量，静态变量）。

2 实例变量和类变量的区别

- 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

➤ **例子10**中，**上底**、**高**和**下底**是类变量，**laderOne**和**laderTwo**是实例变量。每个梯形的对象共享一个下底。

Lader.java, E

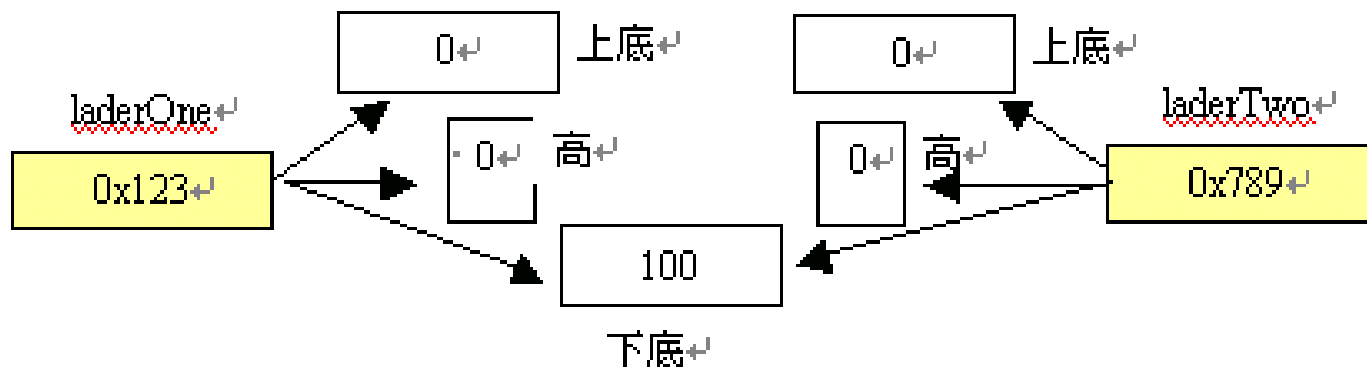


图 4.25 · 对象共享类变量: **下底**



总结：实例成员与类成员

Lader.java

```
1 public class Lader {
2     double 上底,高;           //实例变量
3     static double 下底;       //类变量
4     void 设置上底(double a) {
5         上底 = a;
6     }
7     void 设置下底(double b) {
8         下底 = b;
9     }
10    double 获取上底() {
11        return 上底;
12    }
13    double 获取下底() {
14        return 下底;
15    }
16 }
```

Example4_10.java

```
1 public class Example4_10 {
2     public static void main(String args[]) {
3         Lader.下底=100;           //Lader的字节码被加载到内存,通过类名操作类变量
4         Lader laderOne=new Lader();
5         Lader laderTwo=new Lader();
6         laderOne.设置上底(28);
7         laderTwo.设置上底(66);
8         System.out.println("laderOne的上底:"+laderOne.获取上底());
9         System.out.println("laderOne的下底:"+laderOne.获取下底());
10        System.out.println("laderTwo的上底:"+laderTwo.获取上底());
11        System.out.println("laderTwo的下底:"+laderTwo.获取下底());
12    }
13 }
```





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  
    {  
    }  
}
```



类方法和实例方法

3 实例方法和类方法的定义

- 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修饰的是实例方法、加static关键字修饰的是类方法(静态方法)。

4 实例方法和类方法的区别

1) 对象调用实例方法

- 当对象调用实例方法时，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。

2) 类名调用类方法

- 从而**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，这是因为在类创建对象之前，实例成员变量还没有分配内存。



类和对象的应用

案例：编写一个Telephone类，类中包含有电话品牌、电话号码、通话时间、费率和余额等属性，以及计算话费和显示信息等方法，程序中包含一个主类来使用Telephone类并显示相应的信息。

```
public class Example4_11 {  
    public static void main(String[] args) {  
        Telephone tel;  
        tel = new Telephone("TCL", "8309600", 100);  
        tel.rate = 0.2;  
        tel.dialledTime = 150;  
        tel.display();  
        tel.callCost();  
        tel.recharge(50);  
    }  
}  
  
class Telephone {  
    String brand;  
    String number;  
    double dialledTime;  
    double rate;  
    double balance;  
    public Telephone(String brand,String number,double balance){  
        this.brand = brand;  
        this.number = number;  
        this.balance = balance;  
    }  
}
```





类和对象的应用

```
public void recharge(double cost) {  
    balance = balance + cost;  
    System.out.println("充值后的余额: " + balance);  
}  
  
public void callCost() {  
    double callcost = dialledTime * rate;  
    balance = balance - callcost;  
    System.out.println("话费: " + callcost);  
    System.out.println("余额: " + balance);  
}  
  
public void display() {  
    System.out.println("电话品牌: " + brand + "电话号码: " + number);  
    System.out.println("通话时间: " + dialledTime + "费率: " + rate);  
}  
}
```

@ Javadoc Declaration Console

<terminated> Example4_11 [Java Application] C:\Program Files\Java\jre1.8.0_65\b

电话品牌: TCL电话号码: 8309600

通话时间: 150.0费率: 0.2

话费: 30.0

余额: 70.0

充值后的余额: 120.0





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。





10. 定义包

❖ 包的三大作用

- 1、区分相同名字的种类
- 2、当类很多时，可以很好的管理类
- 3、控制访问范围

❖ 定义包（将类放入包中）

`package` 包名

- 例如： `package java.myPackage;`
- 注意， **package** 必须写在程序的第一行





10. 使用包

❖ 使用其他包中的类

`import 包名.类名;`

- 例: `import myPackage.myClass;`
- 注意: 只能引入其他包中的`public`类






❖ 例题：包的引入例子

```
package myweb.person;

public class Package_1
{
    public void intro()
    {
        int age=12;
        String name="甘罗";
        System.out.println(name+": "+age);
    }
}
```

//在工作目录下建立文件来引用上面的包

```
import myweb.person.*;
public class Import_1
{
    public static void main(String args[])
    {
        Package_1 p=new Package_1();
        p.intro();
    }
}
```

Console  
<terminated> Import_1 [Java Application] C:\Progr
甘罗 : 12





10. 包

❖ 注意：

- **package**必须是第一条语句
- 只能使用其他包中的**public**类
- 没有定义包名的类属于无名包，不能被有名包引用。
- 如果一个程序中使用两个包中的类同名，在使用类名前加上包名前缀





10.包

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）。





10.包

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\jiafei`目录中，运行时必须到`tom\jiafei`的上一层（即`tom`的父目录）目录中去运行主类。
- 假设`tom\jiafei`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`

例子15中的`Student.java`和`Example4_15.java`使用了包语句。

```
C:\1000>javac tom\jiafei\Example4_15.java
```

```
C:\1000>java tom.jiafei.Example4_15
```

```
Student类的包名是tom.jiafei,我的学号：10201
```

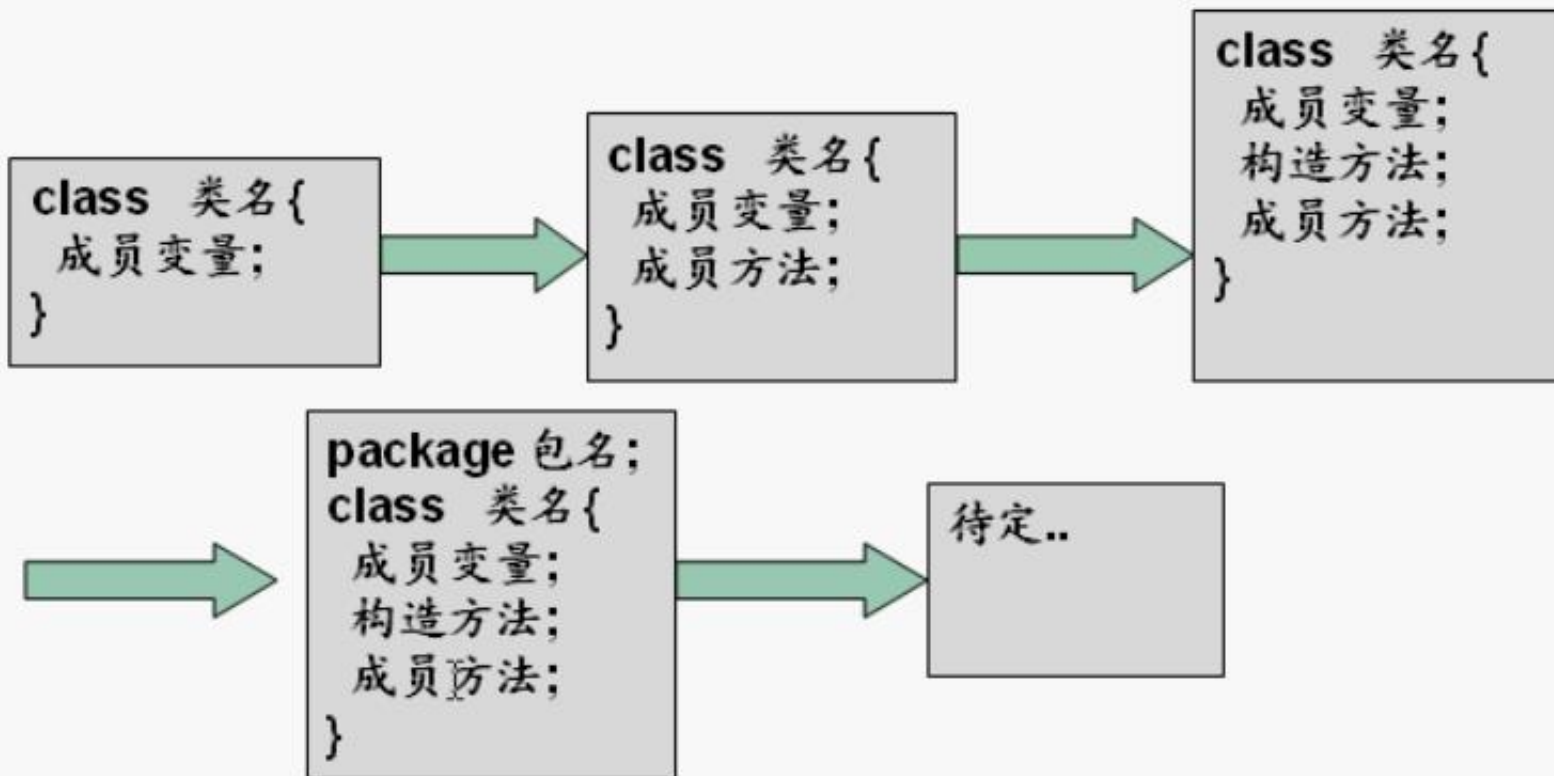
```
主类的包名也是tom.jiafei
```



10. 包

■ 定义类的改进

在提出包后, 我们类的定义就更加完善了:





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。
- 如果一个类想要使用的那个类和它不在一个包中，要使用

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✚



11. import 语句

- 用户程序也可以使用**import**语句引入非类库中有包名的类，如：

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

如果用户不希望更新**classpath**的值，一个简单、常用的办法是：把程序使用的自定义的包名所形成的目录都放在同一文件夹中(见后面的例子17和例子18)。





12. 访问权限

- 当用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。





12. 访问权限

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**不希望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。

例子19(Example4_19.java, Student.java)所示。





12. 访问权限

如: `class tom`

```
{private float weight;*
```

```
  Private float f(float a,float b) {}
```

```
}*
```

```
class jerry*
```

```
{ void g()*
```

```
  { tom tt=new tom();*
```

```
    tt.weight=23; //error*
```

```
    tt.f(3,4); //error  }  }*
```





12. 访问权限

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）。





12. 访问权限

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。





12. 访问权限

- 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





12. 访问权限

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。





12. 总结：可见性修饰符

❖ 类的可见性修饰符

Java提供了访问权限修饰词，以供开发人员向客户端程序员指名哪些是可用的，哪些是不可用的。

❖ 访问权限控制的等级，从最大到最小权限为：

`public` → `protected` → 包访问权限 → `private`





12. 总结：类的可见性修饰符

❖ 类的可见性修饰符

名称	说明	备注
public	可以被所有类访问 (使用)	public 类必须定义在和类名相同的同名文件中
默认的	可以被同一个包中的类访问 (使用)	默认访问权限，可以省略此关键字，可以定义在和 public 类的同一个文件中





12. 总结：类成员的可见性修饰符

❖ 类的成员的可见性修饰符

名称	说明	备注
public	可以被任何类访问	
protected	可以被同一包中的所有类访问 可以被所有子类访问	子类没有在同一包中也可以访问
private	只能够被当前类的方法访问	
缺省的	可以被同一包中的所有类访问	如果子类没有在同一个包中，不能访问



12.总结：类的成员的可见性修饰符

❖ 建议

- 方法和构造方法一般为public
- 属性一般为private
- 对私有属性的访问通过访问器方法完成（set方法和get方法）





12.1 访问器方法-针对属性的

❖ 设置方法

- void **set**属性名（属性类型的参数）

- 比如：对属性radius

```
public void setRadius(double r) {  
    radius = r;  
}
```

❖ 获取方法

- 属性类型 **get**属性名()

```
public double getRadius() {  
    return radius;    }
```





12.1 访问器方法-针对属性的

❖ 例题:访问器方法的应用（计算话费）

```
public class Example5_1 {  
    public static void main(String[] args) {  
        Telephone2 tel;  
        tel = new Telephone2("TCL", "8309600", 100,150,0.2);  
        tel.display();  
        tel.callCost();  
        tel.recharge(50);  
    }  
}  
  
class Telephone2{  
    private String brand; //电话品牌  
    private String number; //电话号码  
    private double dialledTime; //通话时间  
    private double rate; //费率  
    double balance; //话费余额  
  
    public Telephone2(String brand,String number, double balance,double dialledTime,double rate){  
        this.brand = brand;  
        this.number = number;  
        this.balance = balance;  
        this.dialledTime=dialledTime;  
        this.rate=rate;  
    }  
}
```



```
public String getBrand(){
    return brand;
}

public String getNumber() {
    return number;
}

public double getDialledTime() {
    return dialledTime;
}

public double getRate() {
    return rate;
}

public double getBalance() {
    return balance;
}

public void recharge(double cost) {
    balance = balance + cost;
    System.out.println("充值后的余额: " + balance);
}
```

```
public void callCost() {
    double callcost = dialledTime * rate;
    balance = balance - callcost;
    System.out.println("话费: " + callcost);
    System.out.println("余额: " + balance);
}

public void display() {
    System.out.println("电话品牌: " + brand + "电话号码: " + number);
    System.out.println("通话时间: " + dialledTime + "费率: " + rate);
}
}
```

<terminated> Example5_1 [Java Application]

电话品牌: TCL电话号码: 8309600

通话时间: 150.0费率: 0.2

话费: 30.0

余额: 70.0

充值后的余额: 120.0



练习

- ❖ 写一个类描述学生**Student**
- ❖ 有三个属性，分别是：
 - String name
 - int number
 - double score
- ❖ 分别针对三个属性写出他们的访问器方法（共**6**个,两个**getter**，两个**setter**）





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。**





13. 基本类型的类封装

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。
- 请看下面的例题





13. 基本类型的类封装

```
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);
}
}
```





13. 基本类型的类封装

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。





13. 基本类型的类封装

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.

例子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 .





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();**





反编译和文档生成器

- 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public成员变量的名字。

例如：**javap java.awt.Button**

- 使用javadoc.exe可以制做源文件类结构的html格式文档。

例如：**javadoc Example.java**





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：**Jerry.jar**的步骤:

1. 清单文件:hello.mf(保存至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
```



总结与提示

- 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。类变量被该类的所有对象共享；不同对象的实例变量互不相同。
- 除构造方法外，其它方法分为实例方法和类方法。
- 实例方法即可以操作实例变量也可以操作类变量，当对象调用实例方法时，方法中的成员变量就是指分配给该对象的成员变量，其中的实例变量和其它对象的不相同，即占有不同的内存空间；类变量和其它对象的相同，即占有相同的内存空间。
- 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的限制。



不要只因一次失败，就放弃你
原来决心想达到的目的。

Do not, for one repulse, give up
the purpose that you resolved to
effect.

——莎士比亚

