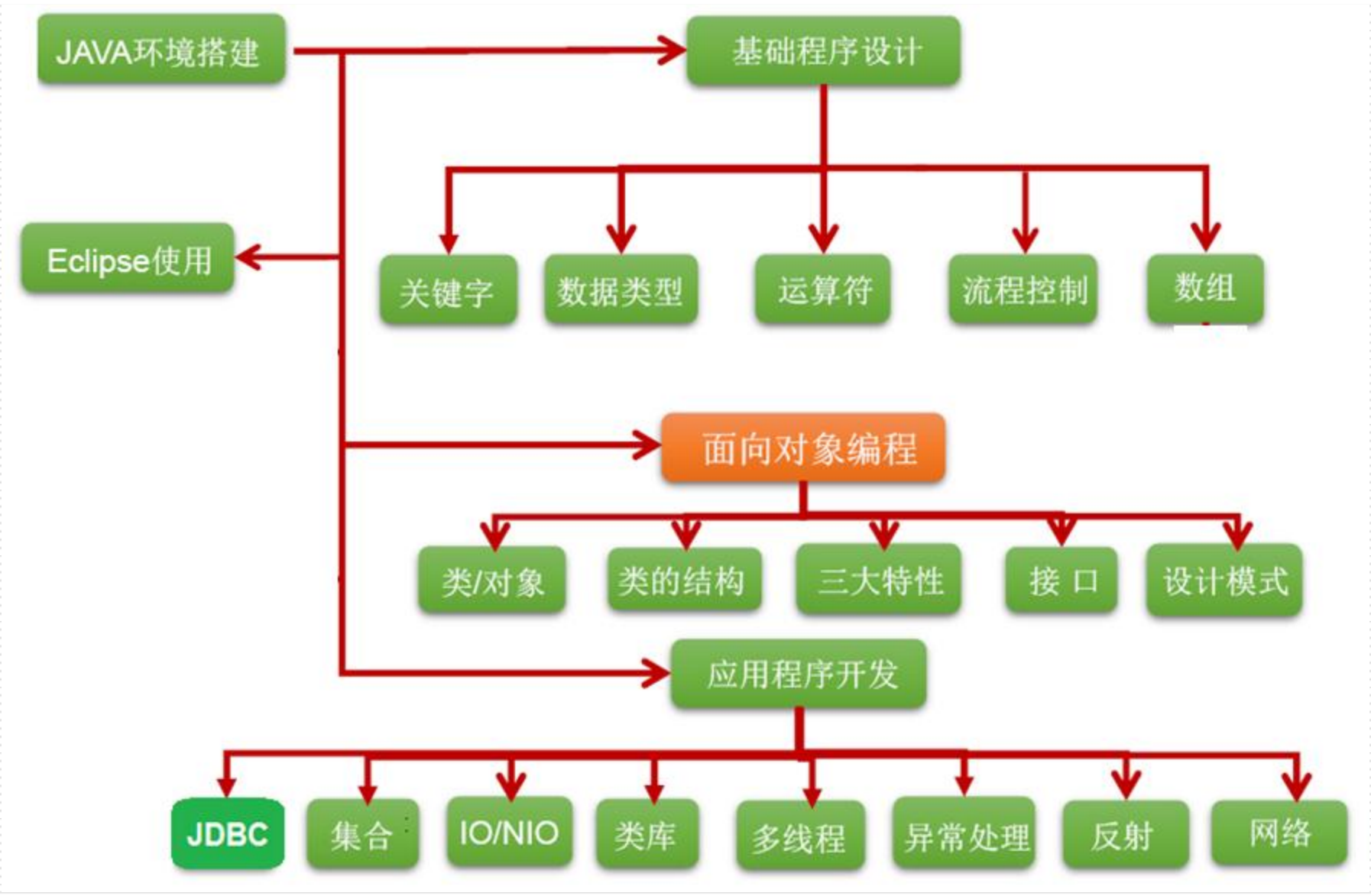
Java语言与系统设计

第5讲 类和对象

- 面向对象编程基本思想
- 类和对象
- 属性（数据成员）
- 方法成员
- 方法重载
- 方法参数
- 构造方法


JAVA环境搭建
基础程序设计
Eclipse使用
关键字
数据类型
运算符
流程控制
数组
面向对象编程
类/对象
类的结构
三大特性
接 口
设计模式
应用程序开发
JDBC
集合
IO/NIO
类库
多线程
异常处理
反射
网络

1. 面向对象编程思想

- 传统面向过程的程序设计方法对于规模比较小的程序，编程者可以直接编写出一个面向过程的程序，详细地描述每一瞬时的数据结构及对其的操作过程。
- 但是当程序规模较大时，就显得力不从心了。面向对象编程就是为了解决编写大程序过程中的困难而产生的。

1. 面向对象编程思想

- 面向对象的程序设计的思路和人们日常生活中处理问题的思路是相似的。在自然世界和社会生活中，一个复杂的事物总是由许多部分组成的。
- 当人们生产汽车时，分别设计和制造发动机、底盘、车身和轮子，最后把它们组装在一起。在组装时，各部分之间有一定的联系，以便协调工作。

1. 面向对象编程思想

- 面向过程(POP) 与面向对象(OOP)：
  - 面向过程，强调的是功能行为，以函数为最小单位，考虑怎么做。
  - 面向对象，将功能封装进对象，强调具备了功能的对象，以类/对象为最小单位，考虑谁来做。

1. 面向对象思想

1.打开冰箱

2.把大象装进冰箱

3.把冰箱门关住

```
人{
    打开（冰箱）{
                冰箱.开门();
    }
    操作(大象){
            大象.进入(冰箱);
    }
    关闭(冰箱){
          冰箱.关门();
    }
}

冰箱{
     开门(){ }
     关门(){ }
}

大象{
     进入(冰箱){ }
}
```

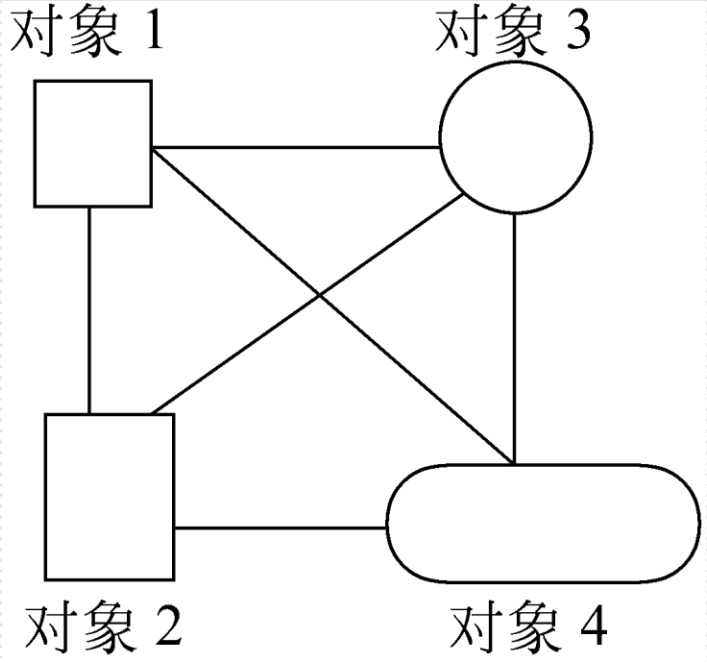
1. 面向对象编程思想

- 面向对象更加强调运用人类在日常的思维逻辑中采用的思想方法与原则，如抽象、分类、继承、聚合、多态等。
- 面向对象的三大特征
  - 封装(Encapsulation)
  - 继承(Inheritance)
  - 多态(Polymorphism)

1. 对象

- 客观世界中任何一个事物都可以看成一个对象(object)。对象可大可小。对象是构成系统的基本单位。
- 任何一个对象都应当具有这两个要素，即属性(attribute)和行为(behavior)，它能根据外界给的信息进行相应的操作。一个对象往往是由一组属性和一组行为构成的。一般来说，凡是具备属性和行为这两种要素的，都可以作为对象。

- 在一个系统中的多个对象之间通过一定的渠道相互联系。要使某一个对象实现某一种行为(即操作)，应当向它传送相应的消息。对象之间就是这样通过发送和接收消息互相联系的。



- 面向对象的程序设计采用了以上人们所熟悉的这种思路。使用面向对象的程序设计方法设计一个复杂的软件系统时，首要的问题是确定该系统是由哪些对象组成的，并且设计这些对象。

- 每个对象都是由数据和函数(即操作代码)这两部分组成的。
- 数据体现了前面提到的“属性”，如一个三角形对象，它的3个边长就是它的属性。
- 函数是用来对数据进行操作的，以便实现某些功能，例如可以通过边长计算出三角形的面积，并且输出三角形的边长和面积。
- 计算三角形面积和输出有关数据就是前面提到的行为，在程序设计方法中也称为方法(method)。调用对象中的函数就是向该对象传送一个消息(message)，要求该对象实现某一行为(功能)。

2. 封装与信息隐蔽

- 可以对一个对象进行封装处理，把它的一部分属性和功能对外界屏蔽，也就是说从外界是看不到的，甚至是不可知的。
- 这样做的好处是大大降低了操作对象的复杂程度。

- “封装性”（encapsulation）指两方面的含义：
  - 一是将有关的数据和操作代码封装在一个对象中，形成一个基本单位，各个对象之间相对独立，互不干扰。
  - 二是将对象中某些部分对外隐蔽，即隐蔽其内部细节，只留下少量接口，以便与外界联系，接收外界的消息。

- 这种对外界隐蔽的做法称为信息隐蔽(information hiding)。信息隐蔽还有利于数据安全，防止无关的人了解和修改数据。
- 对象中的函数名就是对象的对外接口，外界可以通过函数名来调用这些函数来实现某些行为(功能)。

3. 抽象

- 在程序设计方法中，常用到抽象(abstraction）这一名词。抽象的过程是将有关事物的共性归纳、集中的过程。
- 抽象的作用是表示同一类事物的本质。数据类型就是对一批具体的数的抽象。

- 对象是具体存在的，如一个三角形可以作为一个对象，10个不同尺寸的三角形是10个对象。
- 如果这10个三角形对象有相同的属性和行为，可以将它们抽象为一种类型，称为三角形类型。这种类型就称为“类(class)”。这10个三角形就是属于同一“类”的对象。
- 类是对象的抽象，而对象则是类的特例，或者说是类的具体表现形式。

4. 继承与重用

- 如果在软件开发中已经建立了一个名为A的“类”，又想另外建立一个名为B的“类”，而后者与前者内容基本相同，只是在前者的基础上增加一些属性和行为，只需在类A的基础上增加一些新内容即可。
- 这就是面向对象程序设计中的继承机制。利用继承可以简化程序设计的步骤。

- “白马”继承了“马”的基本特征，又增加了新的特征(颜色)，“马”是父类，或称为基类，“白马”是从“马”派生出来的，称为子类或派生类。
- 继承机制可以很方便地利用一个已有的类建立一个新的类。这就是常说的“软件重用”(software reusability) 的思想。

5. 多态性

- 如果有几个相似而不完全相同的对象，有时人们要求在向它们发出同一个消息时，它们的反应各不相同，分别执行不同的操作。这种情况就是多态现象。
- 如，在Windows环境下，用鼠标双击一个文件对象(这就是向对象传送一个消息)，如果对象是一个可执行文件，则会执行此程序，如果对象是一个文本文件，则启动文本编辑器并打开该文件。

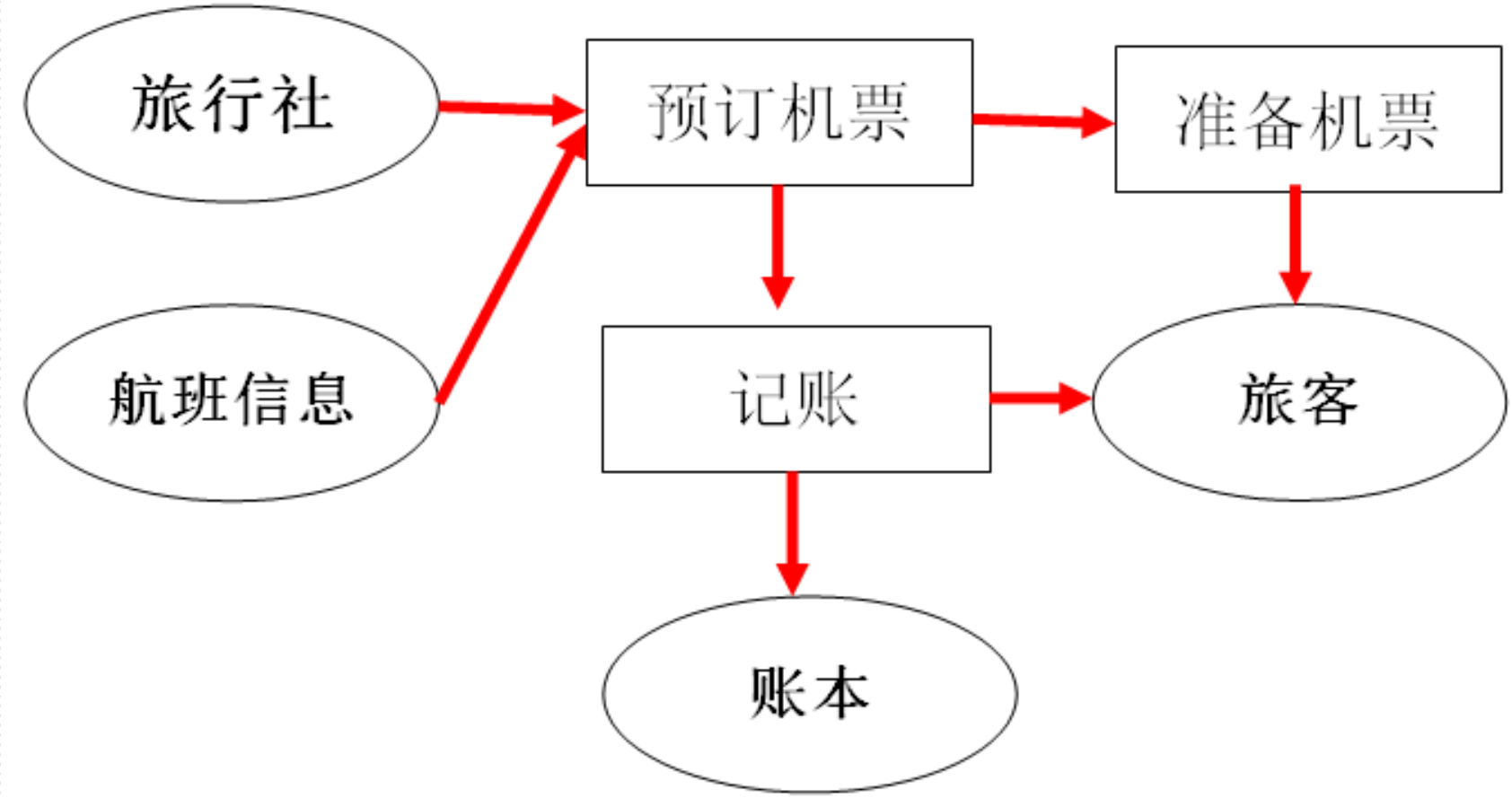
- 所谓多态性(polymorphism)是指： 由继承而产生的相关的不同的类，其对象对同一消息会作出不同的响应。多态性是面向对象程序设计的一个重要特征，能增加程序的灵活性。

1. 面向对象编程思想

- 总结：程序员从面向过程的执行者变成了面向对象的指挥者。面向对象分析方法分析问题的思路和步骤：
  - 根据问题需要，选择问题所针对现实世界中的实体。
  - 从实体中寻找解决问题相关的属性和功能，这些属性和功能就形成了概念世界中的类。
  - 把抽象的实体用计算机语言进行描述，形成计算机世界中类的定义。即借助某种程序语言，把类构造成计算机能够识别和处理的数据结构。
  - 将类实例化成计算机世界中的对象。对象是计算机世界中解决问题的最终工具。

思考题

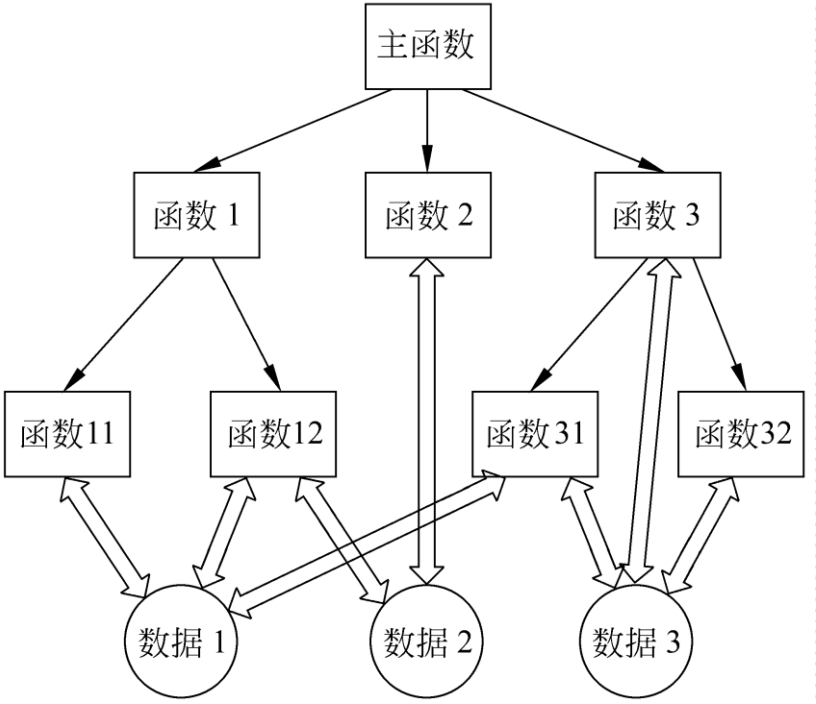
- 在下图中，应该定义哪些类？



2. 类和对象

背景:

传统的面向过程程序设计是围绕功能进行的，用一个函数实现一个功能。所有的数据都是公用的，一个函数可以使用任何一组数据，而一组数据又能被多个函数所使用.



2. 类和对象

- 在面向过程的结构化程序设计中，人们常使用这样的公式来表述程序：

  程序=算法＋数据结构

  - 算法和数据结构两者是互相独立、分开设计的，面向过程的程序设计是以算法为主体的。
  - 在实践中人们逐渐认识到算法和数据结构是互相紧密联系不可分的，应当以一个算法对应一组数据结构，而不宜提倡一个算法对应多组数据结构，以及一组数据结构对应多个算法。

2. 类和对象

- 面向对象程序设计采取的是另外一种思路。它面对的是一个个对象。实际上，每一组数据都是有特定的用途的，是某种操作的对象。也就是说，一组操作调用一组数据。
- 程序设计者的任务包括两个方面： 一是设计所需的各种类和对象，即决定把哪些数据和操作封装在一起；二是考虑怎样向有关对象发送消息，以完成所需的任务.

2. 类和对象

- 这时他如同一个总调度，不断地向各个对象发出命令，让这些对象活动起来(或者说激活这些对象)，完成自己职责范围内的工作。各个对象的操作完成了，整体任务也就完成了。
- 显然，对一个大型任务来说，面向对象程序设计方法是十分有效的，它能大大降低程序设计人员的工作难度，减少出错机会。

2. 类和对象

- 类是JAVA中十分重要的概念，它是实现面向对象程序设计的基础。类是所有面向对象的语言的共同特征，所有面向对象的语言都提供了这种类型。一个有一定规模的JAVA程序是由许多类所构成的。
- 基于对象程序设计所面对的是一个个对象。所有的数据分别属于不同的对象。

2. 类和对象

- 基于对象和面向对象程序设计就是把一个算法和一组数据结构封装在一个对象中。因此，就形成了新的观念：
  - 对象 = 算法 + 数据结构
  - 程序 = (对象+对象+对象+…) + 消息
  - 消息的作用就是对对象的控制。程序设计的关键是设计好每一个对象，及确定向这些对象发出的命令，使各对象完成相应操作。

2. 类和对象

- 对象是实际存在的该类事物的每个个体，因而也称为实例(instance)。
- 类是对一类事物的描述，是抽象的、概念上的定义。
- 定义类：

```
修饰符  class  类名{
  属性声明;
  方法声明;
}
```

2. 类和对象

- 定义类需要考虑：
  - 定义类（考虑修饰符、类名）
  - 编写类的属性（考虑修饰符、属性类型、属性名、初始化值）
  - 编写类的方法（考虑修饰符、返回值类型、方法名、形参等）

2. 类和对象

```
class Rectangle {
        double length;
        double width;
    }
```

在这个类中，定义的类名标识符为Rectangle1，由类名可知，定义的是一个矩形类，在类体中定义有两个数据成员。一个表示矩形的长度，用双精度实型变量length表示，另一个表示矩形的宽度，用双精度实型变量width表示。这个类中只定义有数据成员，而没有定义函数成员，这是允许的，也是最简单的类定义。

2. 类和对象

```
class Rectangle {
 double length;
 double width;
 void setLength(double len){length=len;}
 void setWidth(double wid){width=wid;}
 double getLength(){return length;}
 double getWidth(){return width;}
 void printData()
{System.out.println("length="+length);
 System.out.println("width="+width);}

 }
```

Rectangle2类既定义有数据成员，又定义有函数成员。在这个类中，包含有矩形的长和宽这两个数据成员，分别用标识符length和width标识；同时包含有五个成员函数，setLength()和setWidth()函数用来分别设置矩形的长度和宽度，即各自把参数值赋给相应的成员变量，getLength()和getWidth()函数用来分别返回矩形的长度和宽度，printData()函数用来向屏幕输出矩形长度和宽度。

2. 类和对象

定义对象：

- 在程序中定义一个类之后，就可以利用这个类来定义和创建对象。定义（声明）对象的语句就是变量定义语句，具体格式为：

<类名标识符> <对象标识符> [=<初值表达式>],…;

- 例如，假定已经定义了矩形类，类名标识符为Rectangle，要定义该类型的两个对象，假定对象名分别为r1和r2，则定义语句为：Rectangle r1,r2;

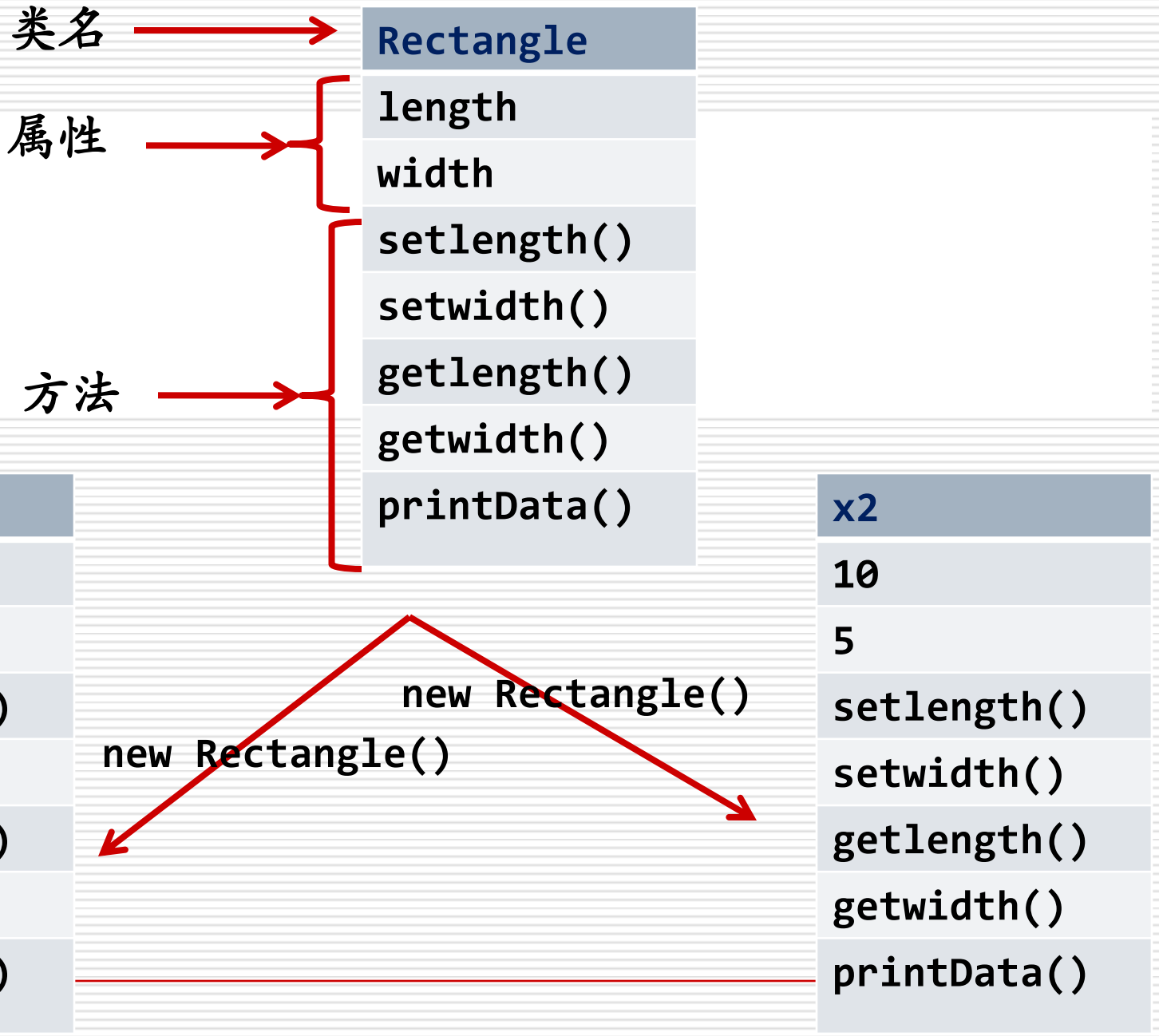
创建对象：

- 同变量定义语句一样，在对象定义语句中可以同时为对象赋初值，由于定义的对象实际上为引用对象，它的值应为实际存储对象的首地址，而实际存储对象的首地址是通过new运算得到的。
- new运算符为一元运算符，操作数是调用该类的构造函数，在内存中生成一个存储对象并进行初始化，以返回该存储对象的首地址作为运算结果。该首地址可以赋给同类型的引用对象，作为引用对象的初值或新修改的值，以后只能够通过引用对象访问所引用（指向）的存储对象。

- Rectangle r1=new Rectangle(), r2=new Rectangle();
- 在该对象定义语句中，定义了r1和r2两个对象，并对r1、2赋初值为new运算的结果，使得r1、r2各指向了一个矩形的存储对象，它们的长度length和宽度width均被在调用该类的无参构造函数时进行了初始化。若该类没有定义任何构造函数，则采用系统默认的无参构造函数进行初始化，使得长度和宽度值均为0。

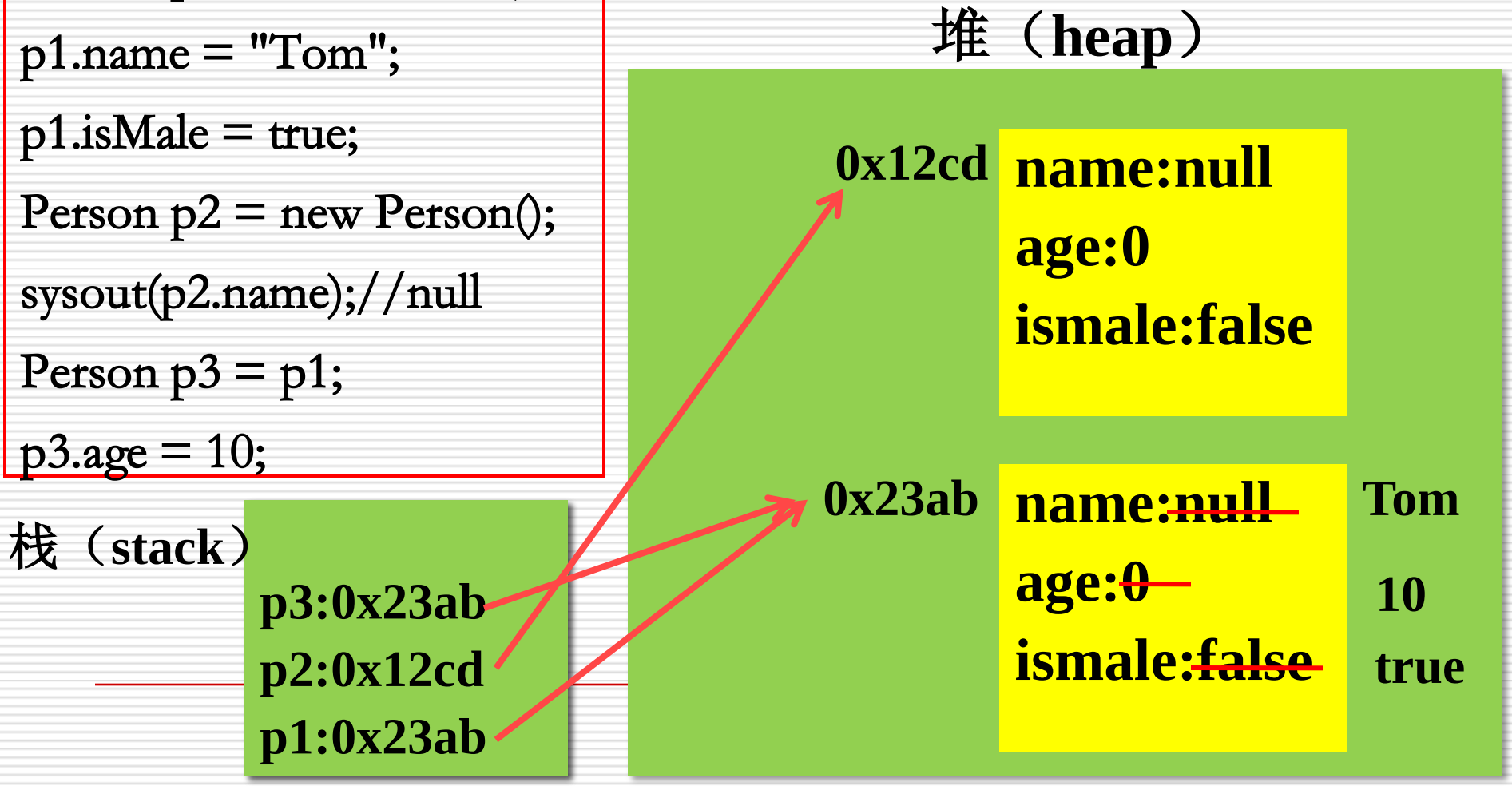
RectangleTest.java中定义主函数:

```
public class RectangleTest{
public static void main(String[] args) {
    Rectangle x1=new Rectangle();
    Rectangle x2=new Rectangle();
    x2.setLength(10); x2.setWidth(5);
    System.out.println("length="+x2.getLength());
    System.out.println("width="+x2.getWidth());
    x2.printData();
    }
}
```


类名
Rectangle
属性
length
width
方法
setlength()
setwidth()
getlength()
getwidth()
printData()
new Rectangle()
new Rectangle()
x1
0
0
setlength()
setwidth()
getlength()
getwidth()
printData()
x2
10
5
setlength()
setwidth()
getlength()
getwidth()
printData()

对象的内存存储实现

```
Person p1= new Person();
p1.name = "Tom";
p1.isMale = true;
Person p2 = new Person();
sysout(p2.name);//null
Person p3 = p1;
p3.age = 10;
```



思考题

谈谈你对面向对象中类和对象的理解，并指出两者的关系。

- 类是一类事物的抽象概念描述，由一组属性和一组行为构成。
- 对象是实际存在的事物的个体。
- 对象是类实例化，类是对象的抽象。

类和对象的创建和执行操作有哪三步?

- 1、创建类
- 2、对象的实例化（new操作）
- 3、调用对象的成员：“对象.属性”或“对象.方法”

3. 属性（数据成员）

- 一个类包括类头和类体两个部分，类体中包含有对所有数据成员和函数成员的定义，数据成员又称为属性、变量成员或成员变量、成员域(field)等，函数成员又称为方法成员或成员方法(method)。
- 数据成员用来反映类的属性，方法成员用来反映类的行为。

3. 属性（数据成员）

数据成员的定义格式：

[<访问权限>][static][final] <数据类型> <变量名列表>;

- 类中数据成员的定义就是一条变量定义语句，其中<数据类型>和<变量名列表>是必选项，其他都是可选项。数据类型可以是基本数据类型，也可以是类（对象）类型。

3. 属性（数据成员）

例如：下面的每条语句都可以作为类中数据成员的定义。

```
(1) int age; //定义int型数据成员age
(2) int count=0; //定义int型数据成员count并赋初值0
(3) int[] a={2,3,4,5};
//定义一维int型数组成员a并对每个元素赋初值
(4) double x,y; //定义double型数据成员x和y
(5) String s1,s2="xxk";
//定义两个字符串对象成员s1和s2，对s2赋初值
```

3. 属性（数据成员）

信息隐藏和封装：

- 程序设计追求“高内聚，低耦合”。
  - 高内聚：类的内部数据操作细节自己完成，不允许外部干涉；
  - 低耦合：仅对外暴露少量的方法用于使用。
- 隐藏对象内部的复杂性，只对外公开简单的接口。便于外界调用，从而提高系统的可扩展性、可维护性。通俗的说，把该隐藏的隐藏起来，该暴露的暴露出来。这就是封装性的设计思想。

实例1

如果类Point的成员变量被访问控制权限被设为public，这样就破坏了类的封装性，因为调用代码可以自由地使用类里面的数据，而Point类却甚至连这些数据什么时候被改动过都不知道。

```
public class Point {
    public double x;
    }
 public class PointTest{
  public static void main(String[] args) {
        Point origin = new Point();
        origin.x=11;
        System.out.println("x="+origin.x);
    }
}
```

```
public class Point {
private double x;
public double getX()
   {  return x;    }
public void setX(double x)
   {   this.x = x;    }}
public class PointTest{
public static void main(String[] args) {
        Point origin = new Point();
        origin.setX(10);
   System.out.println("x="+origin.getX()); }}
```

通常时候，类的成员变量设置为private，可以通过公共的存取方法来访问类成员变量值。

实例2

```
class Customer {
String name;
String sex;
int age; }
public class EncTest1 {
public static void main(String[] args) {
Customer zhangsan = new Customer();
zhangsan.age = 25;
System.out.println("zhangsan.age=" + zhangsan.age);
} }
```

```
class Customer {
String name;
String sex;
        return; }
    this.age = age; }
 public int getAge(){
 return this.age;
} }
```

隐藏一个类中不需要对外提供的实现细节；

使用者只能通过事先定制好的方法来访问数据，可以方便地加入控制逻辑限制对属性的不合理操作；

便于修改，增强代码的可维护性；

3. 属性（数据成员）

数据成员的访问权限：

- 有四种可选择的访问权限（访问属性），其中三种分别用关键字public、protectd和private表示，它们分别表示公有、保护和私有访问权限，剩余一种访问权限不需要使用任何关键字，为默认（缺省、隐含）的访问权限。类中成员的访问权限涉及到4个访问范围，每一种访问权限都对应着一定的访问范围。

修饰符	类内部	同一个包	不同包的子类	同一个工程
private	Yes			
(缺省)	Yes	Yes		
protected	Yes	Yes	Yes	
public	Yes	Yes	Yes	Yes

对于class的权限修饰只可以用public和default(缺省)。

- public类可以在任意地方被访问。
- default类只可以被同一个包内部的类访问。

✲包（packet）类似文件夹，定义方法：packet 包名。

3. 属性（数据成员）

- 访问权限（访问范围）由小到大的次序是:

  私有、默认、保护、公有。

- 让可访问性尽可能低是促成封装的原则之一。当在犹豫某个成员方法的可访问性应该设为public、private或protected时，经验之举是应该采用最严格且可行的访问级别。

(1) int age;

//age具有默认的访问属性，能被同一包访问

(2) private int count=0;

//count具有私有的访问属性，只能被本类访问

(3) public int[] a={2,3,4,5};

//a具有公有的访问属性，被所有包访问

(4) protectd double x,y;

//x和y具有保护的访问属性，能被子类访问

3. 属性（数据成员）

带有final修饰的数据成员：

- 若一个数据成员带有关键字final修饰，则必须给该数据成员初始化（包括在构造函数中赋值），并且以后不能够再被修改。也就是说，使用final修饰的成员变量是一个标识符常量。
- 例如：final int max=100; 就定义了max是一个标识符常量，其值为100，以后不允许再对max赋值，否则将是语法错误。

3. 属性（数据成员）

带有static修饰的数据成员：

- 若一个数据成员带有关键字static修饰，则表示它是类的静态成员，若不带有此关键字修饰，则为一般的对象成员，或称为实例(instance)成员。
- 类中的静态成员（变量）对类中的所有对象都是公共的，共享同一个存储空间，而类中的实例成员变量对类中的每个对象都是各自独立的，分别占有不同的存储空间。也就是说，静态成员属于整个类的，而实例成员则属于该类的每个对象的，不同的对象对应的实例成员的值可以不同，但不同的对象对应的静态成员的值必然相同，因为它们共同使用同一个存储空间。

实例：

```
class Customer{
 String name;
  static String bankName;
}
 public class test1 {
  public static void main(String[] args) {
  Customer zhangsan = new Customer();
  zhangsan.name = “zhangsan";
  zhangsan.bankName = “China Bank";
  Customer lisi = new Customer();
  lisi.name = “lisi";
 System.out.println("lisi.bankName="+lisi.bankName);
 } }
```

3. 属性（数据成员）

对象属性的默认初始化赋值

- 当一个对象被创建时，会对其中各种类型的成员变量自动进行初始化赋值。除了基本数据类型之外的变量类型都是引用类型

成员变量类型	初始值
byte	0
short	0
int	0
long	0L
float	0.0F
double	0.0
char	0 或写为:'\u0000'(表现为空)
boolean	false
引用类型	null

4. 方法成员

- 在类中一般既包含有数据成员，又包含有函数成员，函数成员实现对数据成员的处理操作。
- 类中函数成员的定义格式:

[<访问权限>][abstract][static][final] <返回类型> <函数名>(<参数表>){<函数体>}

[<访问权限>][abstract][static][final] <返回类型> <函数名>(<参数表>){<函数体>}

- <函数名>是用户给该函数所起的名字，通常用一个符合函数功能的英文单词或缩写作为函数名标识符；
- <返回类型>是该函数返回一个值的类型，它可以是基本数据类型，也可以是类（对象）类型，若是基本数据类型中的void类型，则表示该函数不需要返回任何值；
- <参数表>是函数与调用它的对象之间传递信息的接口，参数表可以为空，表明该函数不需要任何参数，也可以带有一个或多个参数，参数之间用逗号分开，每个参数包括参数类型和参数名两个部分；
- <函数体>是一个语句序列，包含有一条或若干条语句。

4. 方法成员

- abstract修饰符：若在成员函数的定义中使用abstract修饰符，则表示它为抽象成员函数。
- 该成员函数只有函数头，没有函数体，在函数头末尾使用分号表示结束定义。
- 抽象成员函数只能出现在抽象类的定义中，不能出现在一般类的定义中。

4. 方法成员

- static修饰符：使得该函数成为静态成员函数，若没有此修饰则为实例成员函数。
- 静态成员函数可以通过类名来访问，不象一般的实例成员函数那样，必须经过对象来访问。
- 另外，在静态成员函数中，只能访问静态成员变量和调用静态成员函数，不允许访问实例成员变量和函数。

4. 方法成员

- final修饰符：使得该函数成为终结性成员函数，该函数不允许被继承类中的成员函数所覆盖，也就是说，不允许在继承类中被重新定义，或者说，不允许具有多态性。若不使用此修饰符，则为一般成员函数，允许被继承类重新定义。

4. 方法成员

函数成员定义举例：

```
(1) void setLength(double len) {length=len;}
//访问权限为缺省，函数名setLength、参数len、返回void、
    函数体为赋值语句
(2) public double getLength() {return length;}
//访问权限公有，函数名getLength、无参数、返回double、
    函数体为返回语句
(3) public static double pi() {return Math.PI;} //
    访问权限公有
 //函数名pi、无参数、返回double、函数体为返回语句、静
    态函数
```

5. 方法重载

重载(overload)

- 在同一个类中，允许存在一个以上的同名方法，只要它们的参数个数或者参数类型不同即可。
- 与返回值类型无关，只看参数列表，且参数列表必须不同。(参数个数或参数类型)。调用时，根据方法参数列表的不同来区别。

5. 方法重载

重载示例:

```
//返回两个整数的和
int add(int x, int y){return x+y;}
//返回三个整数的和
int add(int x, int y, int z){return x+y+z;}
//返回两个小数的和
double add(double x, double y){return x+y;}
```

5. 方法重载

重载示例:

```
public class PrintStream {
    public static void print(int i) {……}
    public static void print(float f) {……}
    public static void print(String s) {……}

    public static void main(String[] args) {
        print(3);
        print(1.2f);
        print("hello!");
    }
}
```

5. 方法重载

与void show(int a,char b,double c){}构成重载的有：

a) void show(int x,char y,double z){} // no

b) int show(int a,double c,char b){} // yes

c) void show(int a,double c,char b){} // yes

d) boolean show(int c,char b){} // yes

e) void show(double c){} // yes

f) double show(int x,char y,double z){} // no

g) void shows(){double c} // no

6. 方法参数

- 调用成员函数时，要根据需要带有相应的参数表。
- 调用时的参数表称为实际参数表，简称实参表，其中的每个参数称为实参。
- 类中成员函数定义时的参数表称为形式参数表，简称形参表，其中的每个参数称为形参。

6. 方法参数

- 实参表中的参数个数和对应的形参表中的参数个数必须相同，并且每个参数的类型必须相同或者兼容，即能够被系统调用时进行自动转换。
- 形参表中的每个参数项是对形参变量的定义，实参表中的每个参数项，是准备传送给对应形参的具体值。所以，每个实参可以是常量、变量、函数、或者表达式。

6. 方法参数

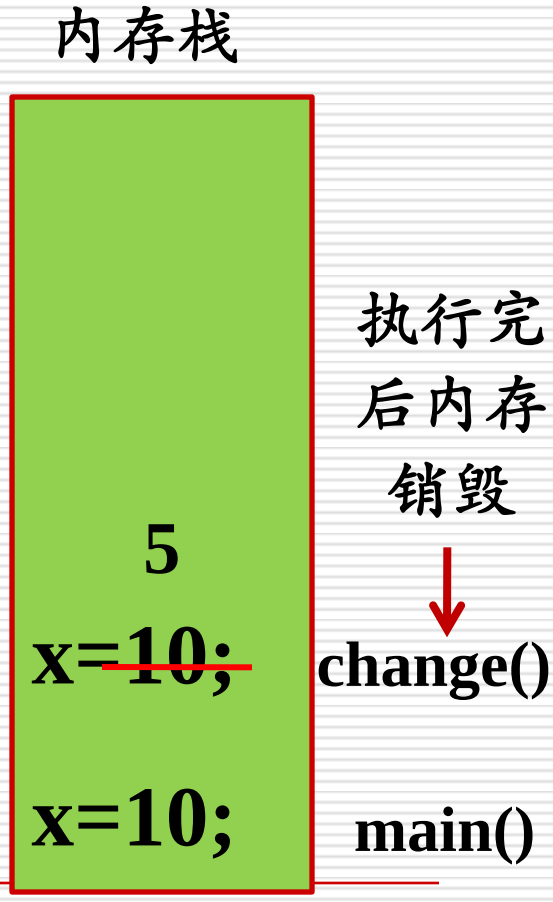
- 方法调用中的按值传递和引用传递
- 在方法调用过程中，首先要进行参数传递，就是把实参的值传送给形参变量，形参变量得到值后，接着执行方法体，执行结束后返回。
- 形参是基本数据类型：将实参基本数据类型变量的“数据值”传递给形参。
- 形参是引用数据类型：将实参引用数据类型变量的“地址值”传递给形参。

6. 方法参数

- 实参向形参传递分为按值传递和引用传递两种方式，当形参为基本数据类型时，则采用按值传递，方法体对形参的修改不影响实参变量的值；
- 当形参为数组类型或类（对象）类型时，则采用引用传递，实参对象和形参对象则为同一个存储空间所保存的对象，方法体对形参对象内容的修改实际上就是对实参对象内容的修改。也就是说，对形参对象的任何操作都是在实参对象上进行的。

值传递

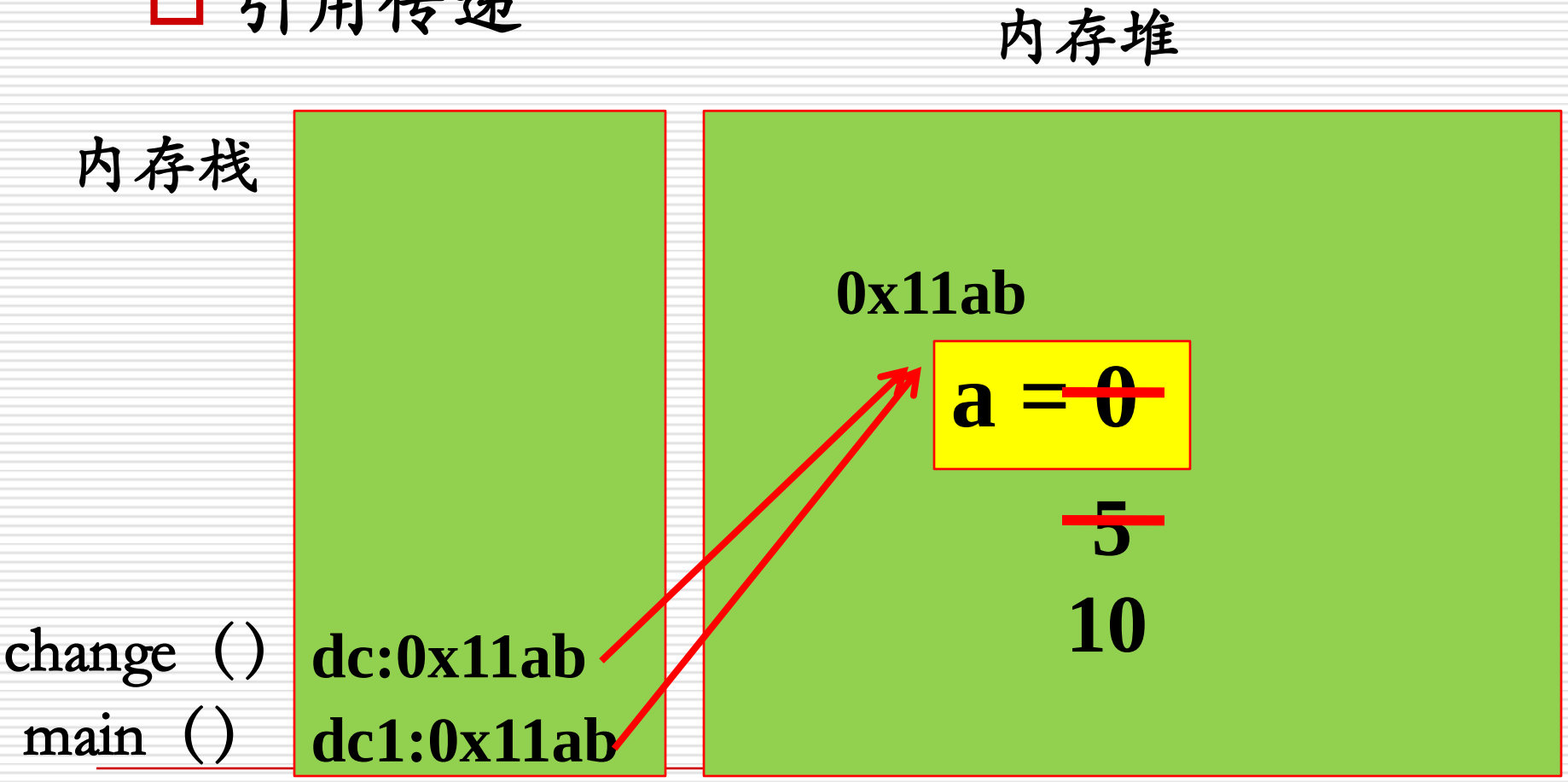
```
public static void main(String[] args) {
x=10;
System.out.println("修改之前x = "+ x);// 10
change(x);
System.out.println("修改之后x = "+ x);// 10
}
public static void change (int x){
System.out.println("change:修改之前x = "+ x);
x=5;
System.out.println("change:修改之后x = "+ x);
}
```



引用传递

```
class DataChange{
public int a;}
public class ChangeTest{
public static void change(DataChange dc){
dc.a=10;}
public static void main(String[] args) {
DataChange dc1=new DataChange();
dc1.a=5;
System.out.println("x="+dc1.a);
change(dc1);
System.out.println("x="+dc1.a);}}
```

引用传递



实例1（变量赋值）：

```
class Rectangle  {      //矩形类
    double length;      //长度
    double width;       //宽度
}
```

实例1：

```
public class Example_4_1
{ public static void main(String[] args) {
   Rectangle x1=new Rectangle();
   x1.length=10;
   Rectangle x2=x1;
   System.out.println("x1: "+x1.length+"x2: "+x2.length+);
   x2.length=20;
   System.out.println("x1: "+x1.length+"x2: "+x2.length+);
}
}
//x1:10,x2:10
//x1:20,x2:20
```

实例2（参数值传递）：

```
class Rectangle  {     //矩形类
    double length;     //长度
    double width;      //宽度
    Rectangle(double len, double wid) {//构造函数
        length=len; width=wid;
    }
    static void increase(Rectangle rec) {
    rec.length++; rec.width++;  //rec的长和宽增1
    }
    double area() {return length*width;}  //计算面
  积
}
```

实例2：

```
public class Example_4_1
{
    public static void main(String[] args) {
    Rectangle x=new Rectangle(4,5);
    System.out.println("x-area: "+x.area());
   Rectangle.increase(x);
    System.out.println("x-area: "+x.area
}
}
```

实例3（参数引用传递）：

```
class Rectangle  {
    double length;
    double width;
    Rectangle(double len, double wid) {
       length=len; width=wid;
    }
   static double increase(double x, double y){
        x++; y++; return x+y;
    }
    double area() {return length*width;}
}
```

实例3：

```
public class Example_4_1
{
    public static void main(String[] args) {
     Rectangle x=new Rectangle(4,5);
     System.out.println("x-area: "+x.area());
     double z=Rectangle.increase(x.length, x.width);
     System.out.println("x-area: "+x.area());
     System.out.println("return: "+z); }
}
```

思考题1

```
public class test
public static void main(String[] args)[
int a=10;
int b=10;
method(a,b);//要求执行完后，打印a=100，b=200
System.out.printlin("a=,"+a);
System.out.printlin("b=,"+b);
}
```

思考题1

```
public static void method(int a, int b){
a=a*10;
b=b*20;
System.out.printlin("a=,"+a);
System.out.printlin("b=,"+b);
Syetem.exit(0);//终止当前程序
}
```

思考题2

定义一个int型的数组:

```
int[] arr = new int[]{1,2,3,4,5};
```

让数组的每个位置上的值去除以首位置的元素，得到的结果，作为该位置上的新值。遍历新的数组。

```
for(int i= 0; i < arr.length;i++){
arr[i] = arr[i] / arr[0];
}
```

- 答案1:

```
for(int i = arr.length -1;i >= 0;i--){
arr[i] = arr[i] / arr[0];
}
```

- 答案2:

```
int temp = arr[0];
for(int i= 0; i < arr.length;i++){
arr[i] = arr[i] / temp;
}
```

思考题3

```
public class StringTest{
public static void main(String[] args){
String s1="hello";
StringTest test=new StringTest();
test.change(s1);
System.out.println(s1); //hello
}
public void change(String s){
s="hi";
}
```

//字符串常量池不存在堆栈中。由于字符串由字符数组组成，形式参数s会在字符串常量池中生成一个字符型数组，存“hi”。

7. 构造方法

- 类中的构造函数，或者叫构造方法，是类中的特殊成员函数，它的作用是为创建的对象进行初始化。
- 构造方法（函数）名与类名相同，不带返回值类型，只允许使用访问权限的修饰符，不允许使用其他任何修饰符，没有return语句返回值。

实例：

在类中定义构造函数举例：

```
class Rectangle3 {
    double length;
    double width;
    Rectangle3 (double len, double wid) {
        length=len; width=wid;
    }
    double getLength() {return length;}
    double getWidth() {return width;}
}
```

该类中的构造函数的函数名与类名Rectangle3相同，不带有返回类型，访问权限为缺省方式，即允许本类和同一个包中的类在使用new运算符创建对象时自动调用，参数表中带有两个双精度参数len和wid，分别用它们初始化类中的成员变量length和width，函数体中实现对这两个成员变量的赋值。

- 在同一个类中可以定义多个构造函数:
- 在同一个类中定义的成员函数，允许具有相同的名字，但参数表要有所不同，这叫做函数（方法）重载，对于构造函数也是如此，允许使用多个构造函数。
- 参数表不同是指：要么参数个数不同，要么对应的参数类型不同。

实例：

```
class Rectangle3
    {
     double length; double width;
     Rectangle3 () {//无参构造函数
     length=0; width=0;
      //成员值均为0
        }
     Rectangle3 (double reg) {
     //带有一个参数的构造函数
      length=width=reg;
     //长宽值均为同一参数值，正方形
        }
```

```
Rectangle3 (double len, double wid)
      {//带两个参数的构造函数
       length=len; width=wid;
    //长宽值不同，值不同则为长方形
          }
    void printData(){
System.out.println("length="+length
      );
System.out.println("width="+width);
          }
     }
```

利用类来定义和创建对象：

```
Rectangle3 r1,r2;r3;  //定义3个矩形对象
r1=new Rectangle3();  //创建对象时自动调用无参构造函数
r2=new Rectangle3(5); //自动调用带有一个参数的构造函数
r3=new Rectangle3(2,6); //自动调用带两个参数的构造函数
```

- 默认构造函数:
- 当类中没有定义任何构造函数时，系统将隐含定义一个构造函数，该构造函数不带任何参数，函数体将初始化成员变量为默认值，具体地说，对数值和字符变量初始化为0，对逻辑变量初始化为假false，对类对象成员初始化为空值null。若类中定义有任何构造函数，则系统不会再自动定义一个构造函数。

例如：

```
class Rectangle1 {
    double length;
    double width;
    public double area() {return length*width;}
}
```

该类中没有定义任何构造函数，系统将隐含（默认、缺省）定义的构造函数为：

```
Rectangle1(){length=0; width=0;} //初始化长宽值为0
Rectangle1 x=new Rectangle1();  //定义和创建x对象，
//创建时，自动调用默认构造函数，初始化长和宽的值为0
```

又如： class Rectangle1 {

```
double length; double width;
Rectangle1 (double len, double wid) {
    length=len; width=wid;
}
void printData() {
    System.out.println("length="+length);
    System.out.println("width="+width);
}}
```

该类中显式地定义有构造函数，这里是一个带有两个参数的构造函数，则系统不会再定义一个缺省的无参构造函数。

Rectangle1 r1,r2;r3; //定义3个矩形对象

r1=new Rectangle1(); //错误，没有无参构造函数可调用

r2=new Rectangle1(5); //错误，没有带有一个参数的构造函数

r3=Rectangle1(2,6); //正确，自动调用带两个参数的构造函数

JavaBean：

JavaBean是一种Java语言写成的可重用组件，符合如下标准：

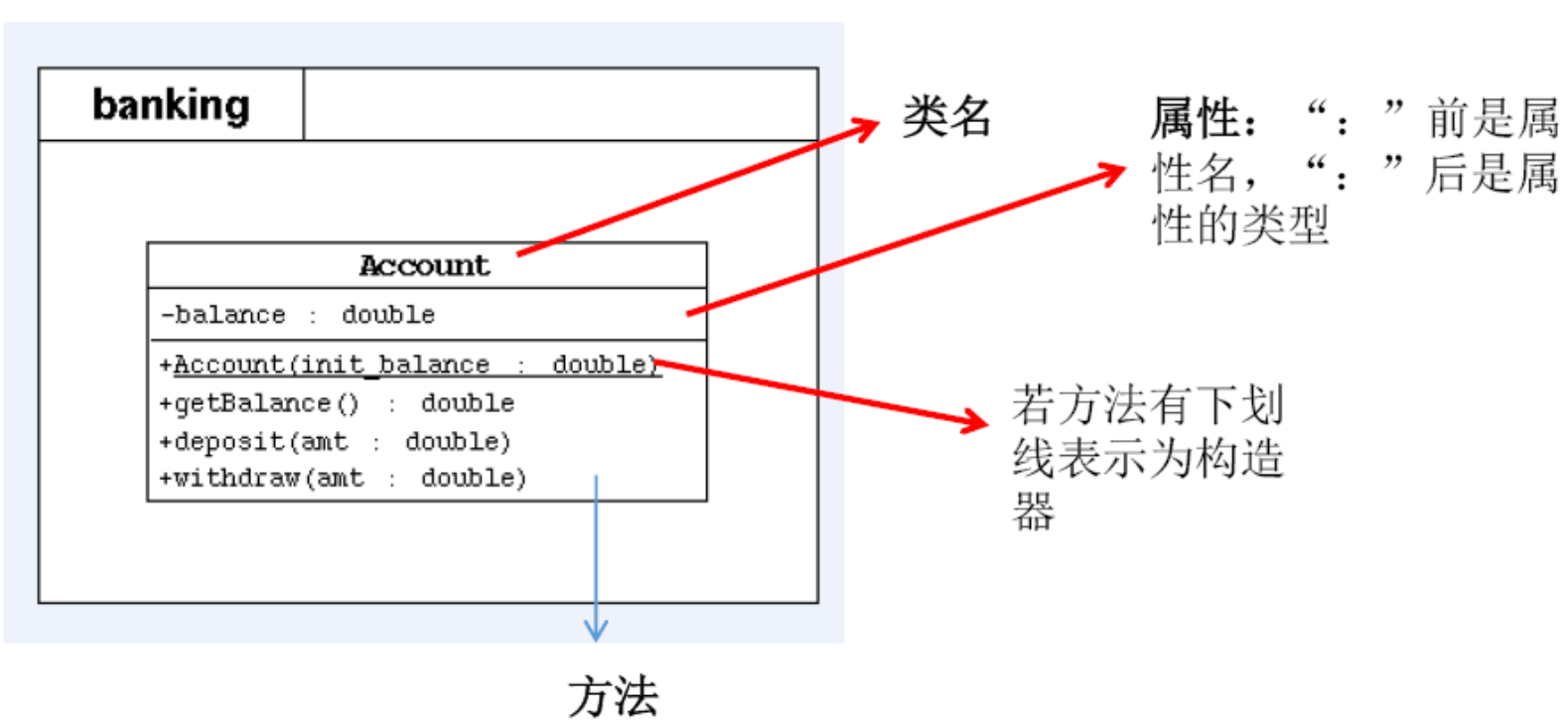
- 类是公共的
- 有一个无参的公共的构造器
- 有属性，且有对应的get、set方法

- 用户可以使用JavaBean将功能、处理、值、数据库访问和其他任何可以用Java代码创造的对象进行打包，并且其他的开发者可以通过内部的JSP页面、Servlet、其他JavaBean、applet程序或者应用来使用这些对象。

JavaBean：

```
public class JavaBean {
    private String name; // 属性一般定义为private
    private int age;
    public JavaBean() {
    }
    public int getAge() {
        return age;
    }
    public void setAge(int a) {
        age = a;
    }
    public String getName() {
        return name;
    }
    public void setName(String n) {
        name = n;
    }
}
```

UML：统一建模语言



- ＋表示public 类型，-表示private 类型，#表示protected类型
- 方法：方法的类型(+、-)方法名(参数名：参数类型)：返回值类型

this关键字：

- this 关键字如果在方法内部使用，即这个方法所属对象的引用；如果在构造方法内部使用，表示该构造器正在初始化的对象。
- 当在方法内需要用到调用该方法的对象时，就用this。this 可以调用类的属性、方法和构造方法。

实例：

```
public class XXK2 {
   private int a,b;
 public XXK2(int aa, int bb)
   {a=aa; b=bb;}
 public int f1(int x) {
  if(x>10) return a+b+3*x;
  else return a*b*x;}
```

```
 public static void
   main(String[] args) {
  XXK2 x=new XXK2(3,4);
  int y=x.f1(8);
   //3*4*8=96
System.out.println("y="+y);
        }
    }
```

public XXK2(int aa, int bb)

改为public XXK2(int a, int b)？

函数体对a赋值的语句应修改为this.a=a，

对b赋值的语句应修改为this.b=b，

其中赋值号左边表示成员变量，右边表示参数变量。

package关键字：

- package语句作为Java源文件的第一条语句，指明该文件中定义的类所在的包。(若缺省该语句，则指定为无名包)。它的格式为：

  package 顶层包名.子包名;

- 包对应于文件系统的目录，package语句中，用“.”来指明包(目录)的层次；

package关键字：

举例：pack1\pack2\PackageTest.java

```
package pack1.pack2;    //指定类PackageTest属于包pack1.pack2
public class PackageTest{
        public void display(){
          System.out.println("in  method display()");
        }
}
```

package关键字：

- 包帮助管理大型软件系统，将功能相近的类划分到同一个包中。
- 包可以包含类和子包，划分项目层次，便于管理
- 解决类命名冲突的问题

（1） java.lang包

- java.lang包是Java语言的核心类库，包含了运行Java程序必不可少的系统类，如基本数据类型、基本数学函数、字符串处理、线程、异常处理类等。每个Java程序运行时，系统都会自动地引入java.lang包，所以这个包的加载是缺省的。

(2) java.io包

java.io包是Java语言的标准输入/输出类库，包含了实现Java程序与操作系统、用户界面以及其他Java程序做数据交换所使用的类，如基本输入/输出流、文件输入/输出流、过滤输入/输出流、管道输入/输出流、随机输入/输出流等。凡是需要完成与操作系统有关的较底层的输入输出操作的Java程序，都要用到java.io包。

(3) java.util包

java.util包包括了Java语言中的一些低级的实用工具，如处理时间的Date类，处理变长数组的Vector类，实现栈和杂凑表的Stack类和HashTable类等，使用它们开发者可以更方便快捷地编程。

（4） java.awt包

- java.awt包是Java语言用来构建图形用户界面(GUI)的类库，它包括了许多界面元素和资源，主要在三个方面提供界面设计支持：低级绘图操作，如Graphics类等；图形界面组件和布局管理，如Checkbox类、Container类、LayoutManager接口等；以及界面用户交互控制和事件响应，如Event类。

（5） java.applet包

- java.applet包是用来实现运行于Internet浏览器中的Java Applet的工具类库，它仅包含少量几个接口和一个非常有用的类：java.applet.Applet。

（6） java.net包

- java.net包是Java语言用来实现网络功能的类库。由于Java语言还在不停地发展和扩充，它的功能，尤其是网络功能，也在不断地扩充。目前已经实现的Java网络功能主要有：底层的网络通信，如实现套接字通信的Socket类、ServerSocket类；编写用户自己的Telnet、FTP、邮件服务等实现网上通信的类；用于访问Internet上资源和进行CGI网关调用的类，如URL等。

（7） java.awt.event包

- java.awt.event包是对JDK 1.0版本中原有的Event类的一个扩充，它使得程序可以用不同的方式来处理不同类型的事件，并使每个图形界面的元素本身可以拥有处理它上面事件的能力。

（8） java.sql包

- java.sql包是实现JDBC(Java database connection)的类库。利用这个包可以使Java程序具有访问不同种类的数据库的功能，如Oracle，Sybase，DB2，SQLServer等。

import关键字：

- 为使用定义在不同包中的Java类，需用import语句来引入指定包层次下所需要的类或全部类(.*)。
- import语句告诉编译器到哪里去寻找类。
- 语法：import 包名.类名;

import关键字：

```
import  pack1.pack2.Test;
//import pack1.pack2.*;表示引入pack1.pack2包中的所有类
public class PackTest{
public static void main(String args[]){
Test t = new Test(); //Test类在pack1.pack2包中定义
t.display();
}
}
```

import关键字：

- 声明在包的声明和类的声明之间。
- 如果需要导入多个类或接口，并列使用多个import语句。
- 使用如java.util.*的方式，一次性导入util包下所有的类或接口。
- 如果导入的类或接口是java.lang包下的，或者是当前包下的，则可以省略此import语句。
- 如果在代码中使用不同包下的同名的类。那么就需要使用类的全类名的方式指明调用的是哪个类。
- 如果已经导入java.a包下的类。那么如果需要使用a包的子包下的类的话，仍然需要导入。