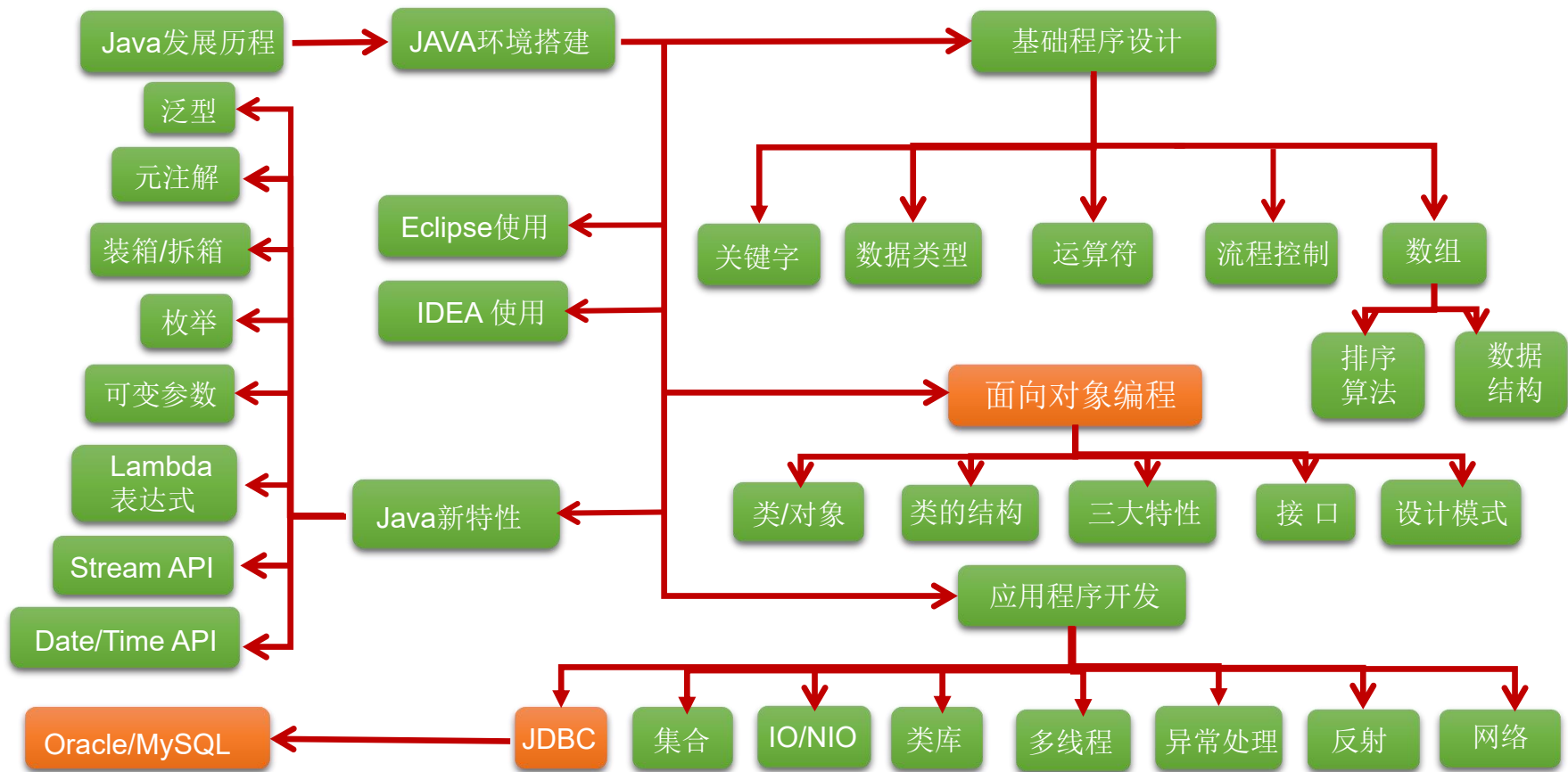




第4章

面向对象编程(上)

讲师：宋红康
新浪微博：尚硅谷-宋红康





学习面向对象内容的三条主线

1. **Java**类及类的成员
2. 面向对象的三大特征
3. 其它关键字

目录



1

面向过程与面向对象

2

Java基本元素：类和对象

3

对象的创建和使用

4

类的成员之一：属性

5

类的成员之二：方法

目录



6

再谈方法

7

OOP特征一：封装与隐藏

8

类的成员之三：构造器

9

关键字：this

10

关键字：package、import



4-1 面向过程与面向对象



何谓“面向对象”的编程思想？

首先解释一下“思想”。

先问你个问题：你想做个怎样的人？

可能你会回答：我想做个好人，孝敬父母，尊重长辈，关爱亲朋.....

你看，这就是思想。这是你做人的思想，或者说，是你做人的原则。

做人有做人的原则，编程也有编程的原则。这些编程的原则呢，就是编程思想。



顿悟？ OR 渐悟？



● 面向过程(POP) 与 面向对象(OOP)

- 二者都是一种思想，面向对象是相对于面向过程而言的。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。**
- 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)

面向对象: Object Oriented Programming

面向过程: Procedure Oriented Programming



4.1 面向过程与面向对象

例子：人把大象装进冰箱



1.打开冰箱

2.把大象装进冰箱

3.把冰箱门关上



面向过程

```
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){ }  
}
```

面向对象



面向对象的思想概述

- 程序员从面向过程的**执行者**转化成了面向对象的**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，选择问题所针对的**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。



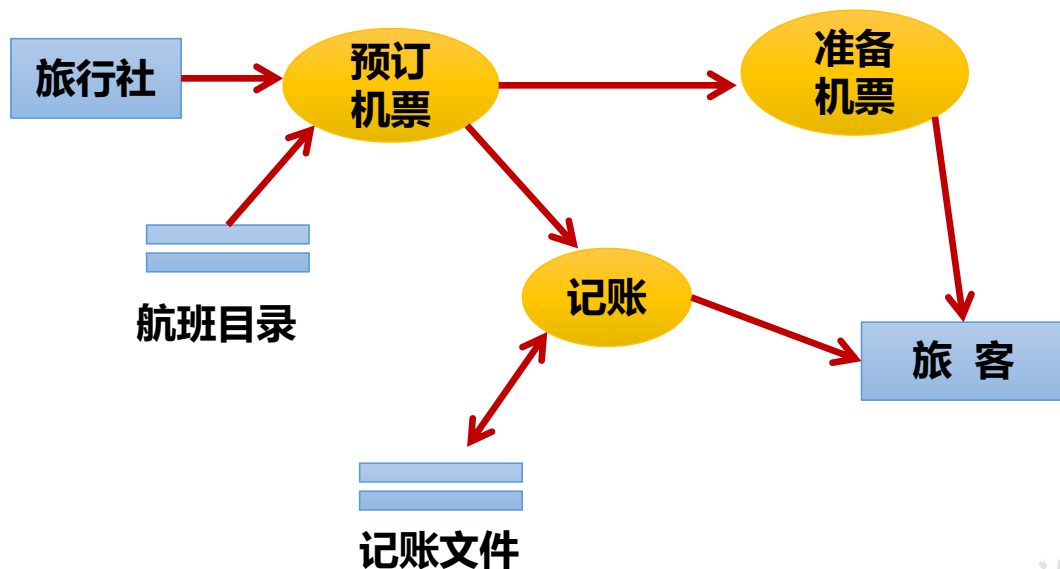
练习1

1. 我要开车去丽江，这句话包含的类有什么？
2. 体会以下几个经典案例涉及到的类。
 - 人在黑板上画圆
 - 列车司机紧急刹车
 - 售货员统计收获小票的金额
 - 你把门关上了



练习1

3. 抽象出下面系统中的“类”及其关系。





4-2 Java语言的基本元素： 类和对象

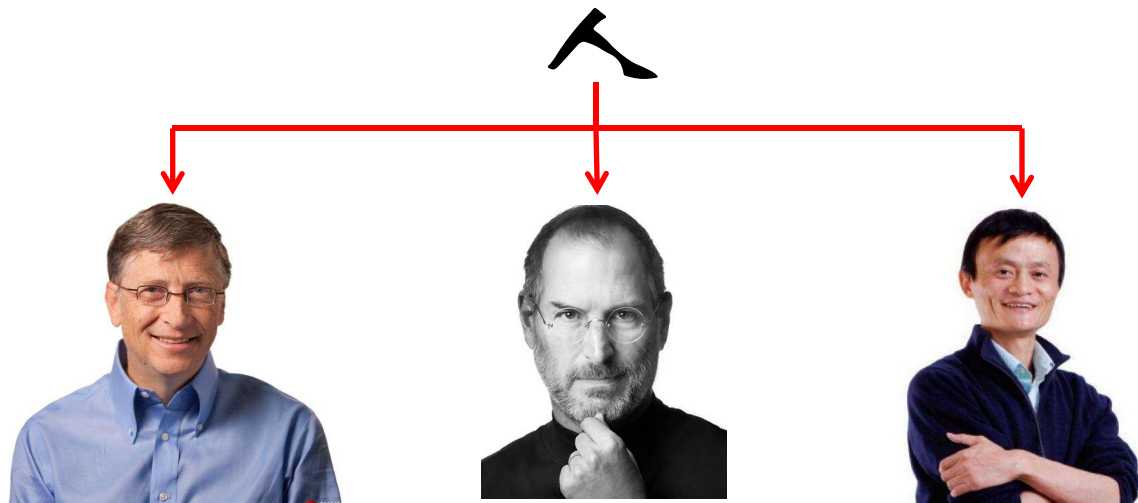


面向对象的思想概述

- 类(Class)和对象(Object)是面向对象的核心概念。
 - 类是对一类事物的描述，是抽象的、概念上的定义
 - 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)。
- “万事万物皆对象”



面向对象的思想概述

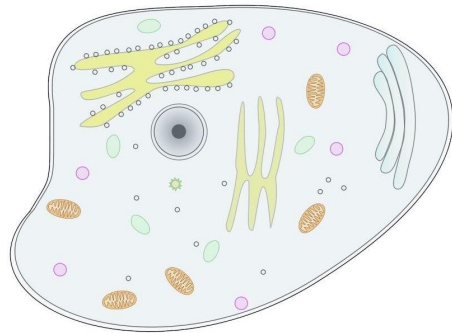


- 可以理解为：类 = 抽象概念的人；对象 = 实实在在的某个人
- 面向对象程序设计的重点是类的设计
- 类的设计，其实就是类的成员的设计



Java类及类的成员

- 现实世界的生物体，大到鲸鱼，小到蚂蚁，都是由最基本的**细胞**构成的。同理，Java代码世界是由诸多个不同功能的**类**构成的。
- 现实生物世界中的细胞又是由什么构成的呢？细胞核、细胞质、... 那么，Java中用类class来描述事物也是如此。常见的类的成员有：
 - **属性**：对应类中的成员变量
 - **行为**：对应类中的成员方法



Field = 属性 = 成员变量, Method = (成员)方法 = 函数



李梦秋

求职意向：网络推广专员

个人资料 Personal Info

出生年月：1995 年 11 月

专 业：电子商务

学 历：大学本科

现 居：广东-珠海

政治面貌：共青团员

联系方式 Contact Info

手 机：188-0000-0000

邮 箱：1880000@163.com

联系地址：北师大珠海分校

属性

// 教育背景 //

2014.09-2018.06 北京师范大学珠海分校 电子商务专业（本科）
专业课程：市场营销学、网络营销、网店运营管理、国际贸易理论、电子商务金融、电子商务法、电子商务概论、电子商务发展趋势、国际贸易理论与实务、人力资源管理、数据库原理等课程

// 实习经历 //

2015.11-2016.05 佳能珠海有限公司 品牌推广专员
✦ 实习期间，主要推广佳能新产品，耐心地为消费者讲解产品的性能指标和操作方法，并结合消费者的心理，有目的地进行专门介绍，诱导其做出购买决策；

2015.04-2015.10 珠海一统科技有限公司 网络营销员
✦ 在职期间，主要通过 qq 群和论坛发布产品信息，与有意愿购买人员做详细的产品介绍，并提供优势的客户服务，促成交易；
✦ 期间通过一系列的宣传推广，平均每月卖出各类电脑 10 余台，业绩受到上级领导的肯定，同时掌握相关的销售技巧和线上推广方式等；

// 技能&证书 //

【大学生英语四级】 【电子商务大赛一等奖】 【校三等奖学金】

【国家计算机二级】 【市场营销大赛二等奖】 【机动车驾驶证】

✦ 熟练使用 Microsoft office 办公软件，精通 word、Excel、PPT 的各项操作，通过自己设计出来的作品，上传到数字作品平台网站售卖，并获得年度优秀设计师称号；
✦ 熟悉电商平台的运营操作流程，擅长新媒体运营及营销策略的制定，具有运营过官方微博和微信公众号的工作经历，另外熟悉 SPSS、绘声绘影的操作；

行为



Java类及类的成员

```
class Person {  
    String name;  
    int age;  
    boolean isMarried;  
  
    public void walk(){  
        System.out.println("人走路...");  
    }  
    public String display(){  
        return "名字是: "+name+", 年龄是: "+age+", Married: "+isMarried;  
    }  
}
```

} 属性，或成员变量

} 方法，或函数

类的成员构成 version 1.0



```
class Person {  
    //属性，或成员变量  
    String name;  
    boolean isMarried;  
    //构造器  
    public Person(){}  
    public Person(String n,boolean im){  
        name = n;isMarried = im;  
    }  
    //方法，或函数  
    public void walk(){  
        System.out.println("人走路...");  
    }  
    public String display(){  
        return "名字是: "+name+",Married:"+isMarried;  
    }  
    //代码块  
    {  
        name = "HanMeiMei";  
        age = 17;  
        isMarried = true;  
    }  
    //内部类  
    class pet{  
        String name;  
        float weight;  
    }  
}
```

类的成员构成

version 2.0



类的语法格式

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

说明：修饰符 **public**：类可以被任意访问
类的正文要用{ }括起来

举例：

```
public class Person{  
    private int age ;           //声明私有变量 age  
    public void showAge(int i) { //声明方法showAge()  
        age = i;  
    }  
}
```



创建Java自定义类

步骤：

1. 定义类（考虑修饰符、类名）
2. 编写类的属性（考虑修饰符、属性类型、属性名、初始化值）
3. 编写类的方法（考虑修饰符、返回值类型、方法名、形参等）

练习：

定义Person、Animal、ClassRoom、Zoo等类，加以体会。



4-3 对象的创建和使用



java类及类的成员



如何使用java类？

java类的实例化，即创建类的对象



4.3 对象的创建和使用

- 创建对象语法： **类名 对象名 = new 类名();**
- 使用 “**对象名.对象成员**” 的方式访问对象成员（包括属性和方法）

举例：

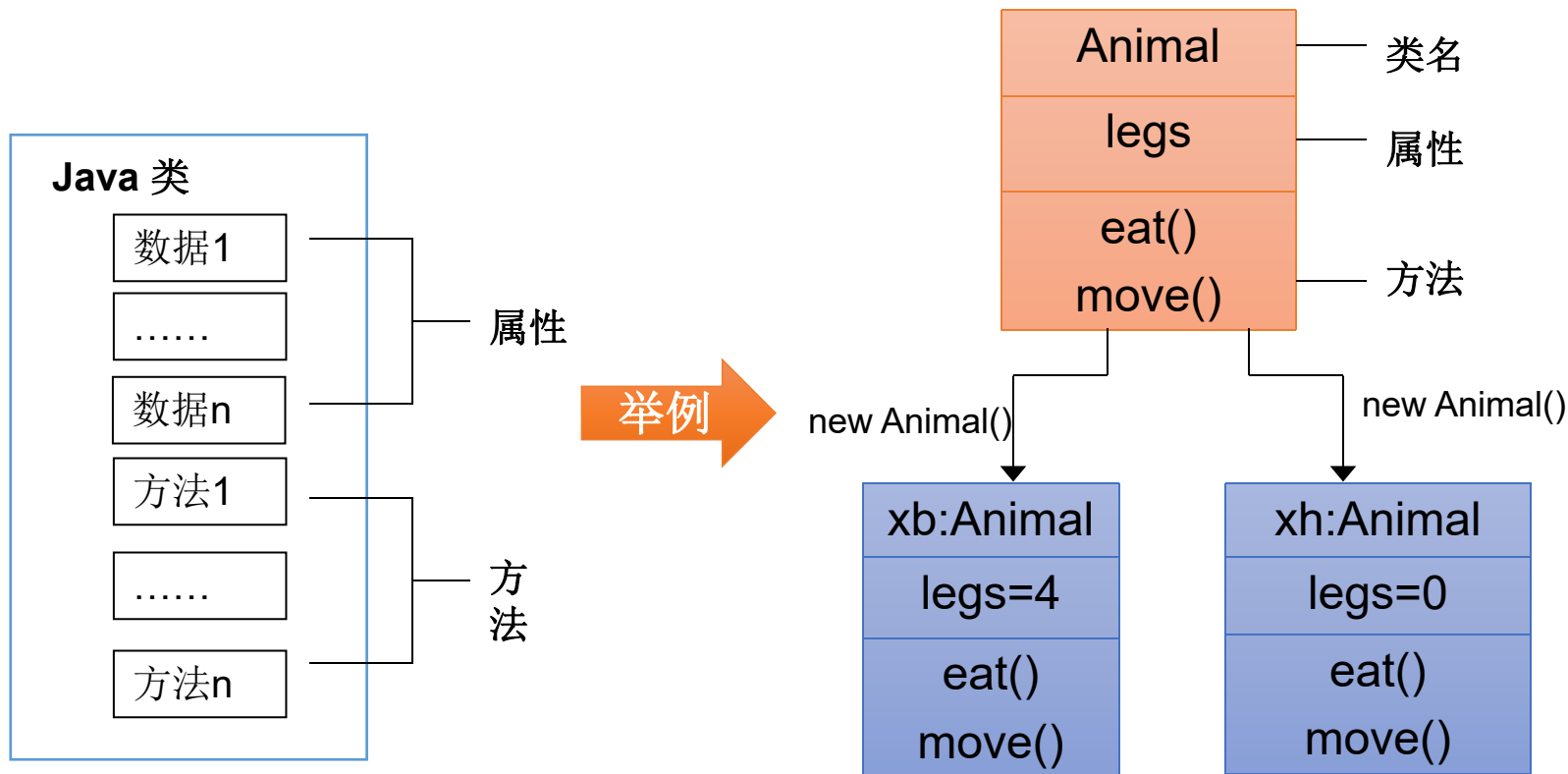
```
public class Animal {  
    public int legs;  
    public void eat(){  
        System.out.println("Eating.");  
    }  
    public void move(){  
        System.out.println("Move.");  
    }  
}
```

```
public class Zoo{  
    public static void main(String args[]){  
        //创建对象  
        Animal xb=new Animal();  
        xb.legs=4;//访问属性  
        System.out.println(xb.legs);  
        xb.eat();//访问方法  
        xb.move();//访问方法  
    }  
}
```




4.3 对象的创建和使用

Java 中类与对象





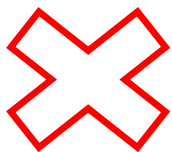
4.3 对象的创建和使用

曰：“白马非马，可乎？”

曰：“可。”

曰：“何哉？”

曰：“马者，所以命形也。白者，所以命色也。命色者，非命形也，故曰白马非马。”





4.3 对象的创建和使用

说明:

如果创建了一个类的多个对象，对于类中定义的属性，每个对象都拥有各自的一套副本，且互不干扰。

```
public class Zoo {  
    public static void main(String args[]) {  
        Animal xb = new Animal();  
        Animal xh = new Animal();  
        xb.legs = 4;  
        xh.legs = 0;  
        System.out.println(xb.legs); // 4  
        System.out.println(xh.legs); // 0  
        xb.legs = 2;  
        System.out.println(xb.legs); // 2  
        System.out.println(xh.legs); // 0  
    }  
}
```



4.3 对象的创建和使用

练习 2

编写教师类和学生类，并通过测试类创建对象进行测试

Student类

属性:

name:String

age:int

major:String

interests:String

方法: **say()**

返回学生的个人信息

Teacher类

属性:

name:String

age:int

teachAge:int

course:String

方法: **say()**

输出教师的个人信息



提示

●类的访问机制：

- **在一个类中的访问机制：**类中的方法可以直接访问类中的成员变量。
(例外：**static**方法访问非**static**，编译不通过。)
- **在不同类中的访问机制：**先创建要访问类的对象，再用对象访问类中定义的成员。

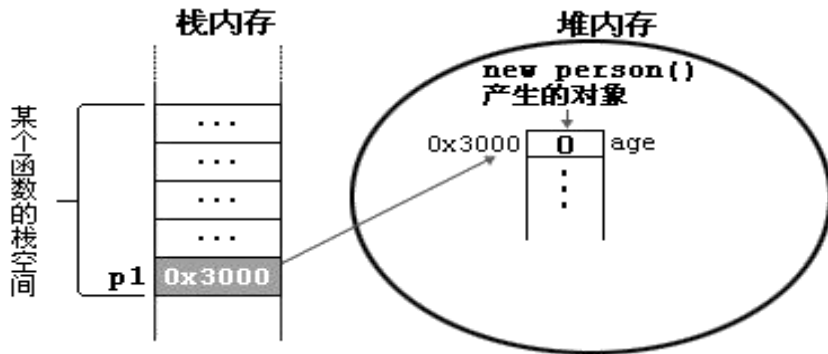


4.3 对象的创建和使用

对象的产生

Person p1 = new Person();执行完后的内存状态。其中类定义如下:

```
class Person{  
    int age;  
    void shout(){  
        System.out.println("oh,my god! I am " + age);  
    }  
}
```

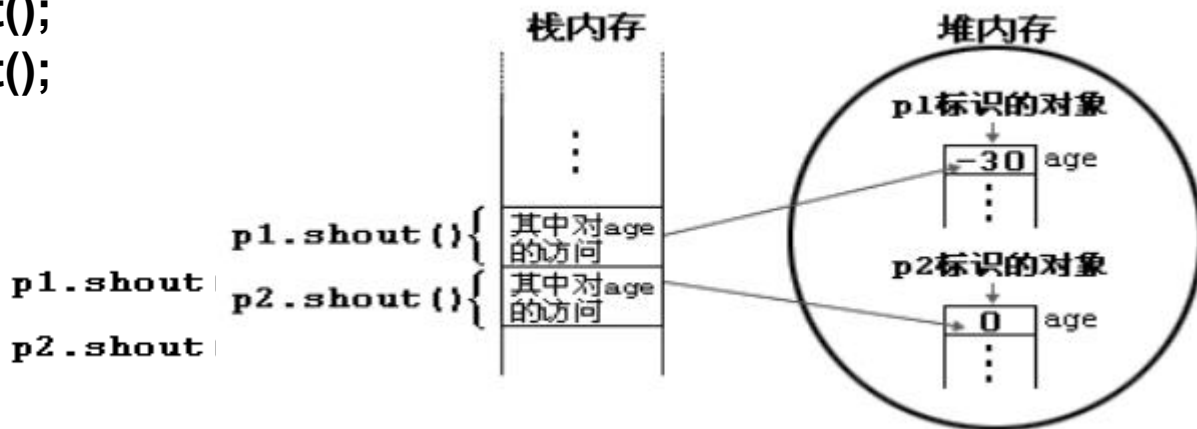




4.3 对象的创建和使用

对象的使用

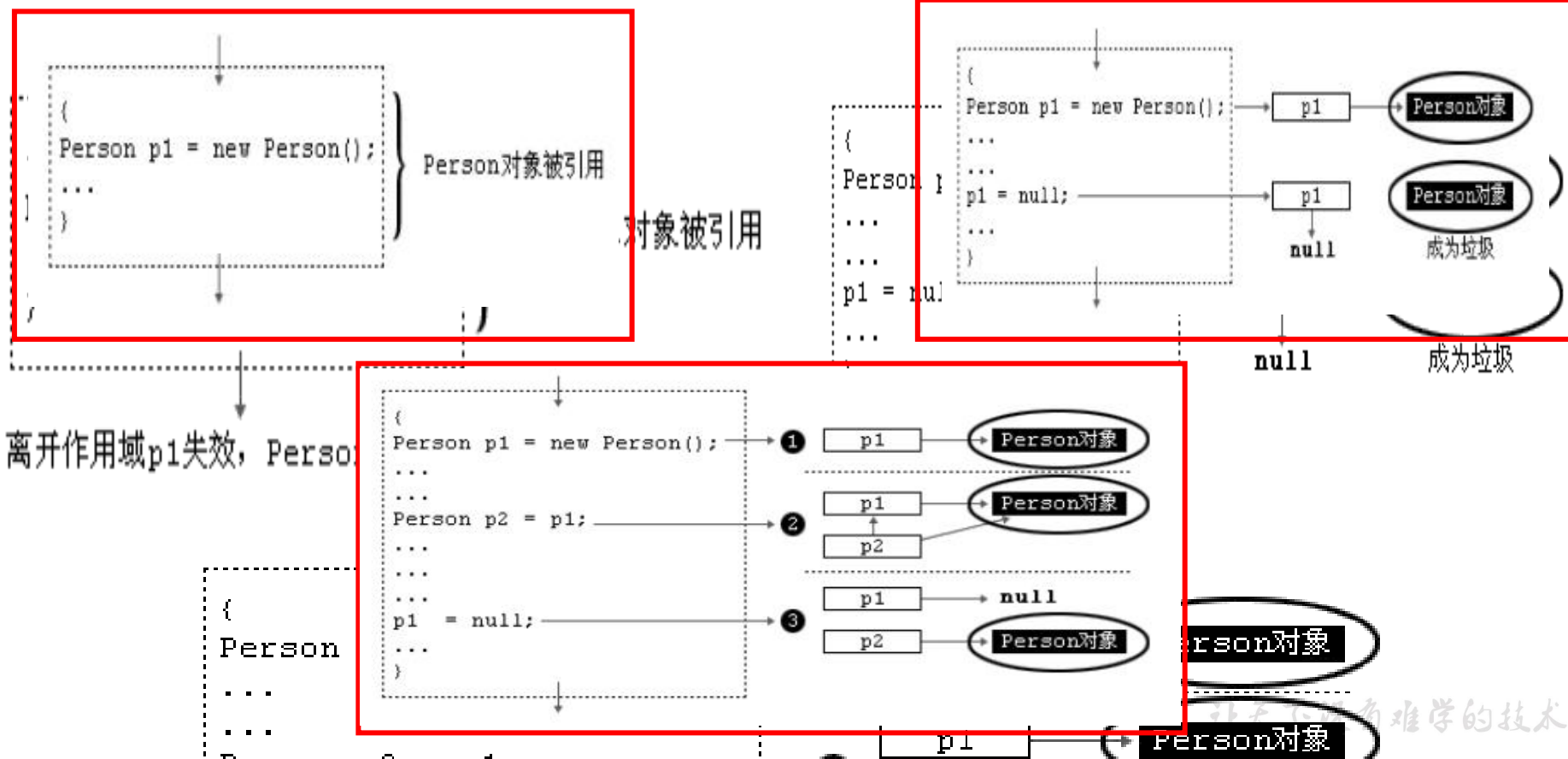
```
class PersonTest{  
    public static void main(String[] args) { //程序运行的内存布局如下图  
        Person p1 = new Person();  
        Person p2 =new Person();  
        p1.age = -30;  
        p1.shout();  
        p2.shout();  
    }  
}
```





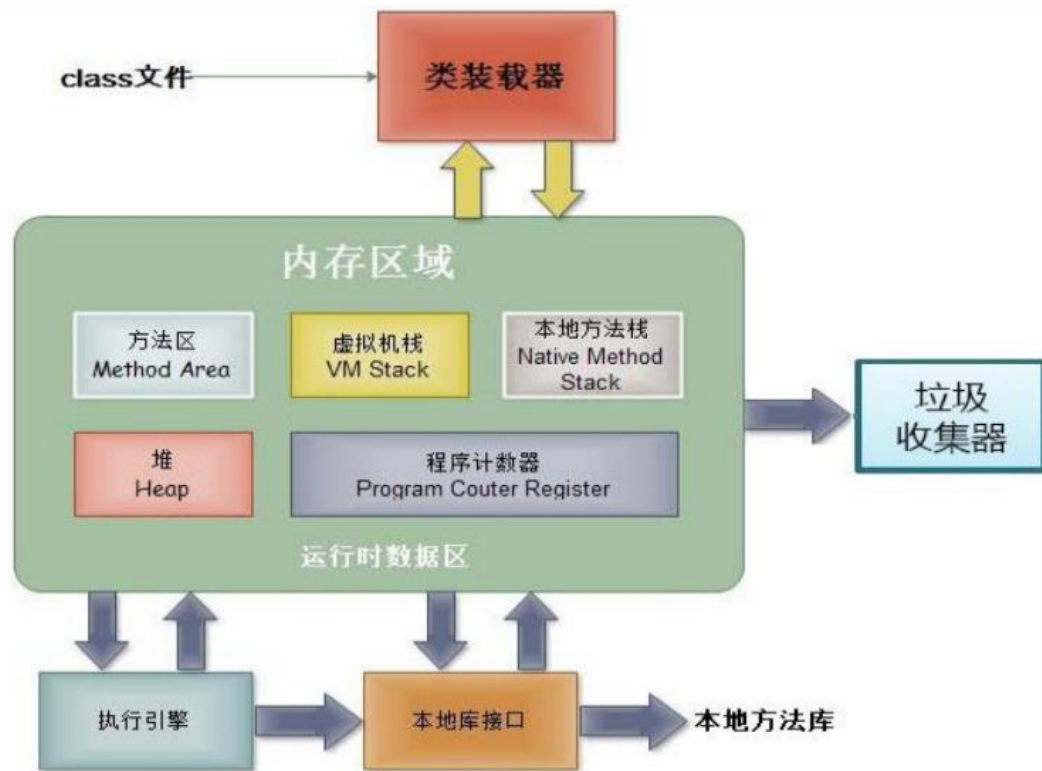
4.3 对象的创建和使用

对象的生命周期





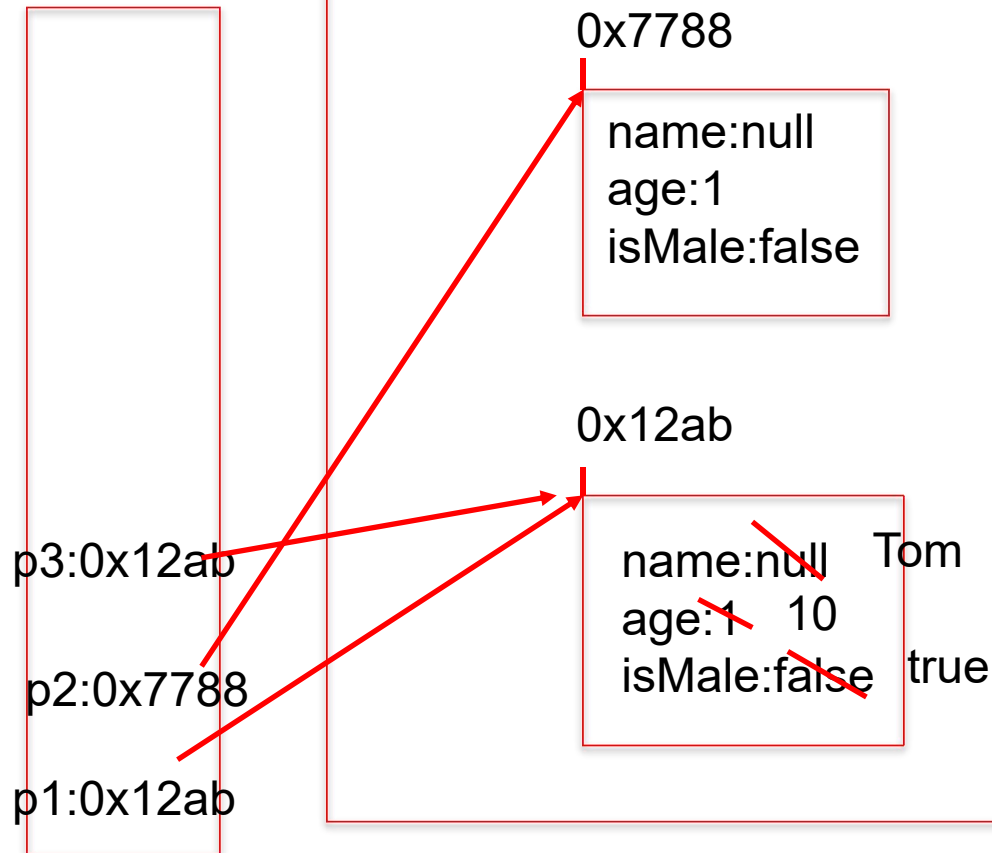
4.3 对象的创建和使用：内存解析



- **堆 (Heap)**，此内存区域的唯一目的就是存放对象实例，几乎所有的对象实例都在这里分配内存。这一点在Java虚拟机规范中的描述是：所有的对象实例以及数组都要在堆上分配。
- 通常所说的**栈 (Stack)**，是指虚拟机栈。虚拟机栈用于存储局部变量等。局部变量表存放了编译期可知长度的各种基本数据类型（boolean、byte、char、short、int、float、long、double）、对象引用（reference类型，它不等同于对象本身，是对象在堆内存的首地址）。方法执行完，自动释放。
- **方法区 (Method Area)**，用于存储已被虚拟机加载的类信息、常量、静态变量、即时编译器编译后的代码等数据。



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



堆: heap

栈: stack



4.3 对象的创建和使用：内存解析

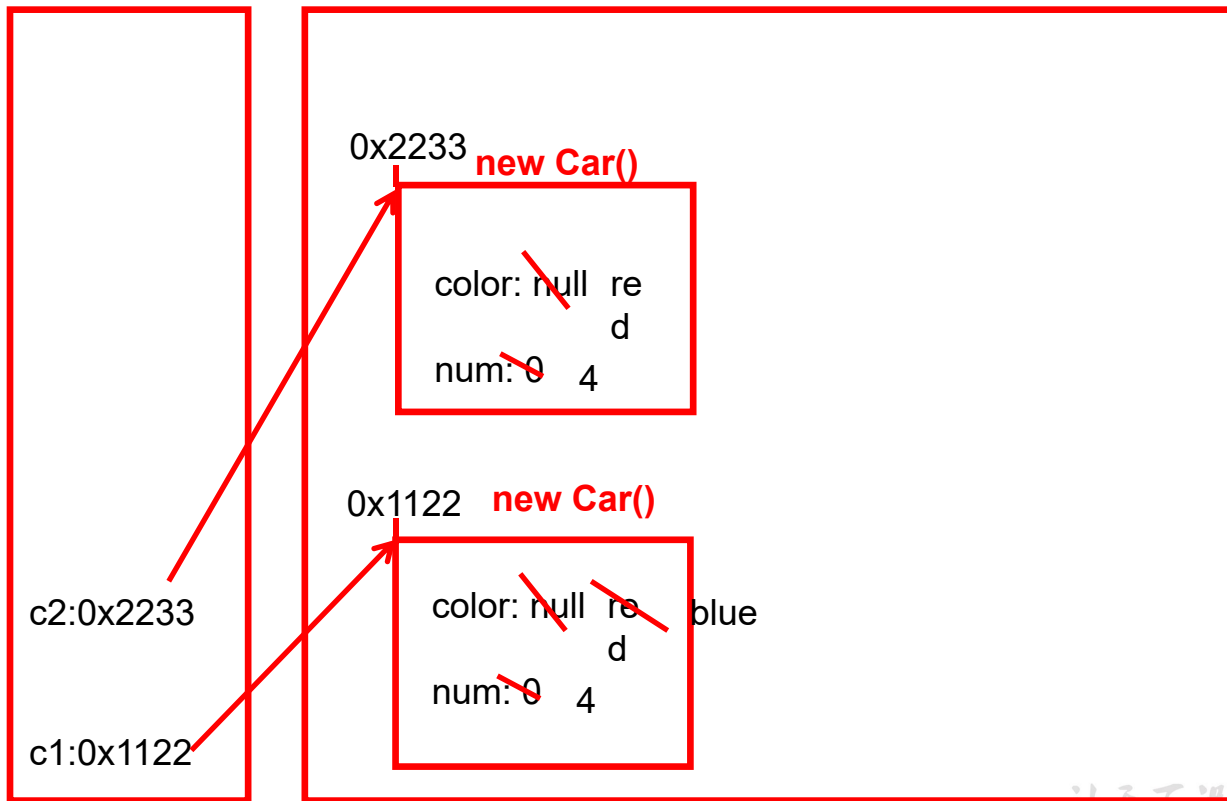
根据代码，画出内存图

```
class Car{
    String color = "red";
    int num = 4;
    void show(){
        System.out.println("color="+color+"..num="+num);
    }
}
class CarTest {
    public static void main(String[] args) {
        Car c1 = new Car(); //建立对象c1
        Car c2 = new Car(); //建立对象c2
        c1.color = "blue"; //对对象的属性进行修改
        c1.show(); //使用对象的方法
        c2.show();
    } }
```



4.3 对象的创建和使用：内存解析

对象内存结构

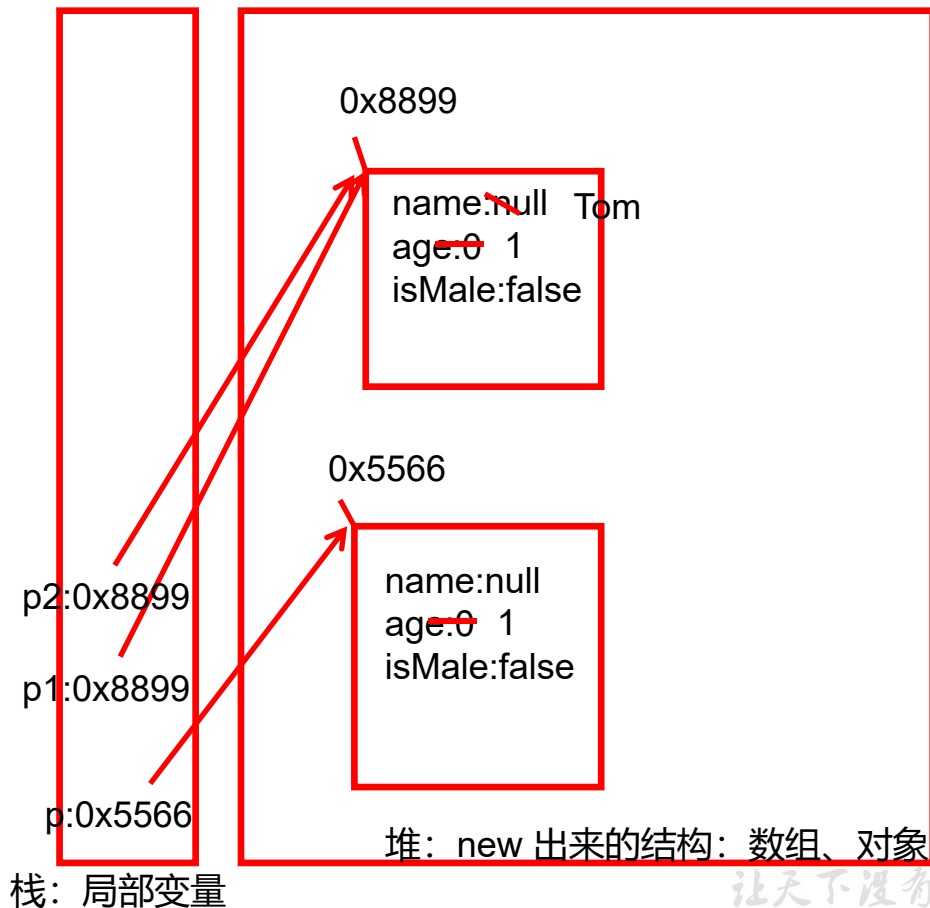




4.3 对象的创建和使用：内存解析

```
class Person{//人类
//1.属性
String name;//姓名
int age = 1;//年龄
boolean isMale;//是否是男性
}
```

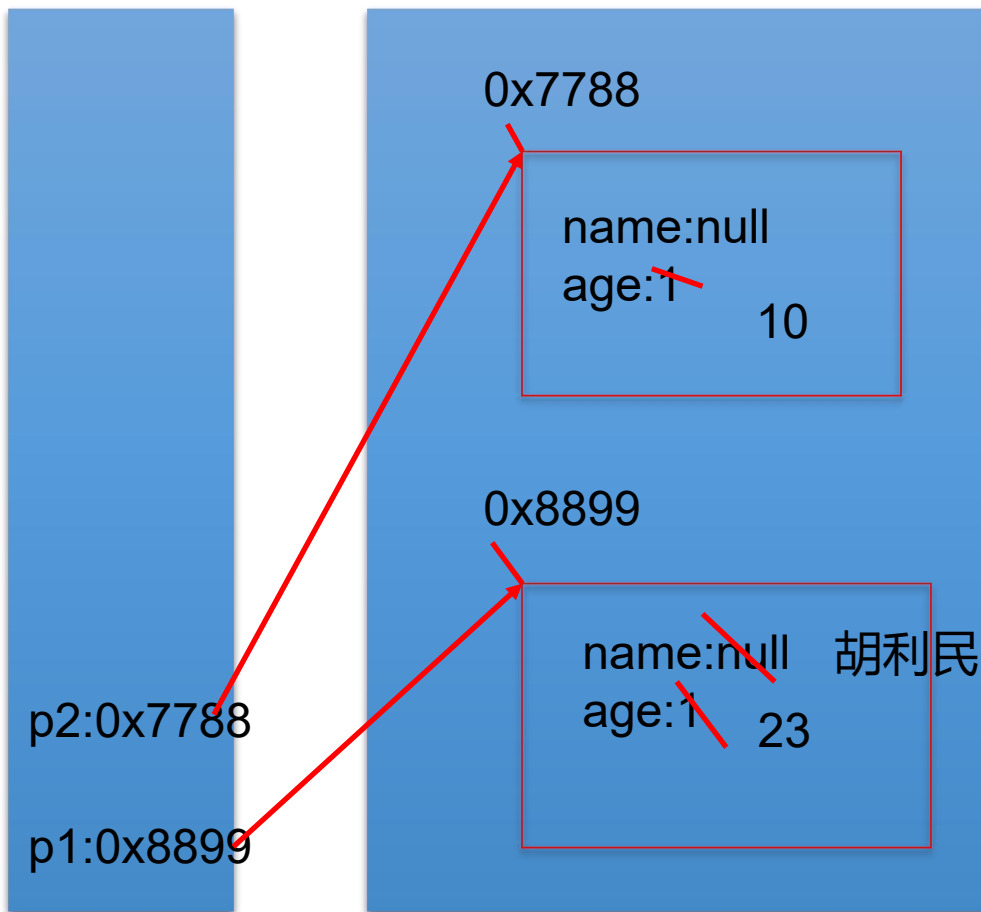
```
class PersonTest{
    main(){
        Person p = new Person();
        Person p1 = new Person();
        p1.name = "Tom";
        Person p2 = p1;
    }
}
```





4.3 对象的创建和使用：内存解析

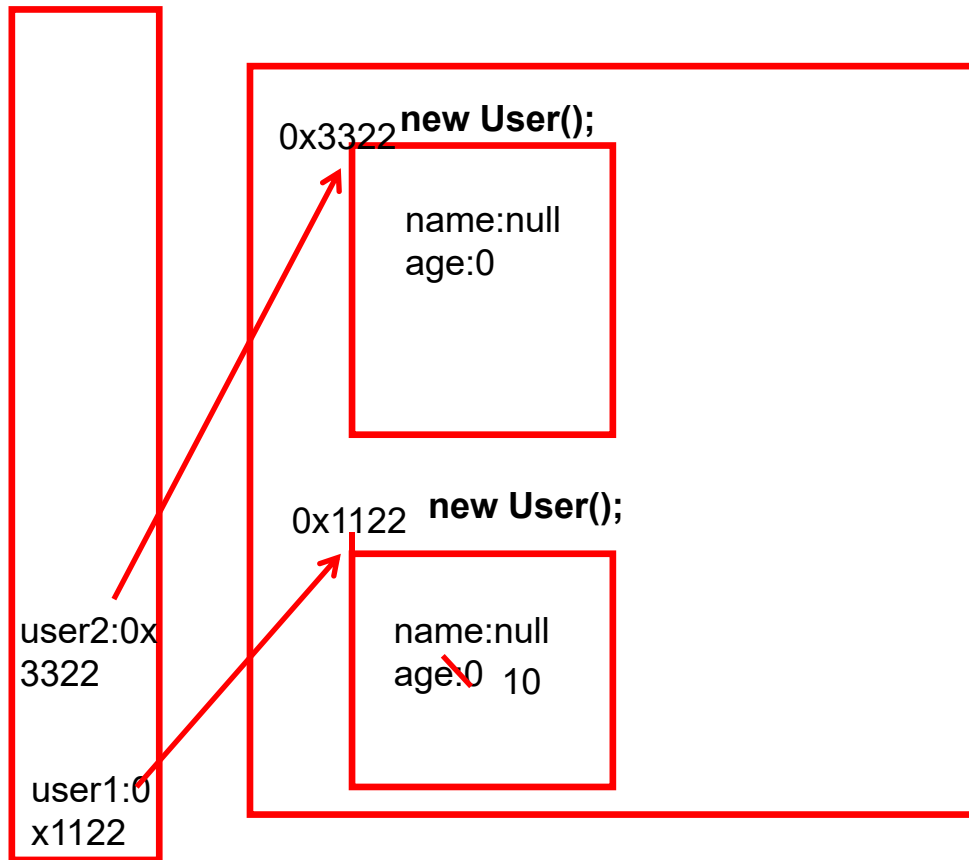
```
Person p1 = new Person();  
p1.name = "胡利民";  
p1.age = 23;  
Person p2 = new Person();  
p2.age = 10;
```





4.3 对象的创建和使用：内存解析

```
class UserTest{
    main(){
        User user1 = new
        User();
        user1.age = 10;
        User user2 = new
        User();
        sysout(user2.age);//0
    }
}
class User{
    //属性（或成员变量）
    String name;
    int age ;
}
```





4.3 对象的创建和使用：匿名对象

- 我们也可以不定义对象的句柄，而直接调用这个方法。这样的对象叫做匿名对象。
 - 如：**`new Person().shout();`**
- 使用情况
 - 如果对一个对象只需要进行一次方法调用，那么就可以使用匿名对象。
 - 我们经常将匿名对象作为实参传递给一个方法调用。



4-4 类的成员之一： 属性(field)



4.4 类的成员之一：属性

- 语法格式：

修饰符 数据类型 属性名 = 初始化值 ;

- 说明1：修饰符

- ✓ 常用的权限修饰符有：private、缺省、protected、public
- ✓ 其他修饰符：static、final (暂不考虑)

- 说明2：数据类型

- ✓ 任何基本数据类型(如int、Boolean) 或 任何引用数据类型。

- 说明3：属性名

- ✓ 属于标识符，符合命名规则和规范即可。

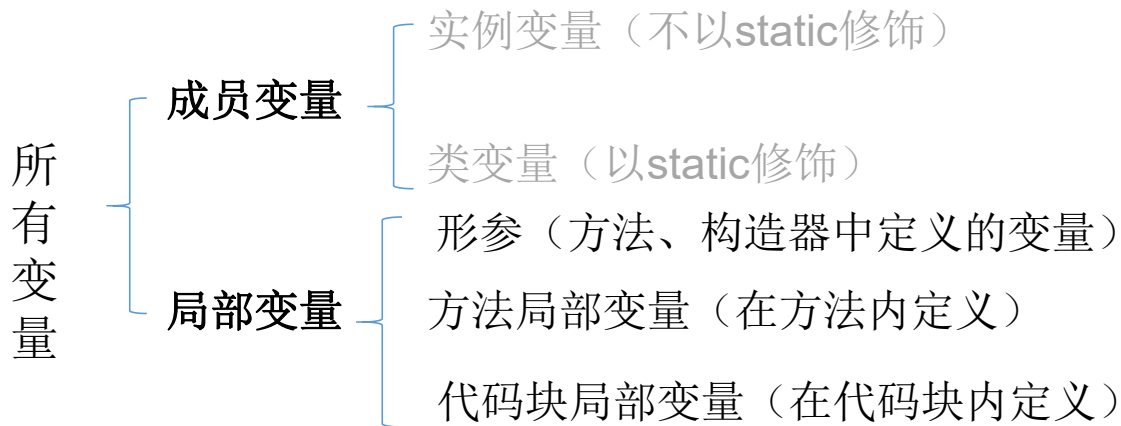
- 举例：

```
public class Person{  
    private int age;           //声明private变量 age  
    public String name = "Lila"; //声明public变量 name  
}
```



变量的分类：成员变量与局部变量

- 在方法体外，类体内声明的变量称为成员变量。
- 在方法体内部声明的变量称为局部变量。



- 注意：二者在初始化值方面的异同：

同：都有生命周期

异：局部变量除形参外，均需显式初始化。



成员变量（属性）和局部变量的区别？

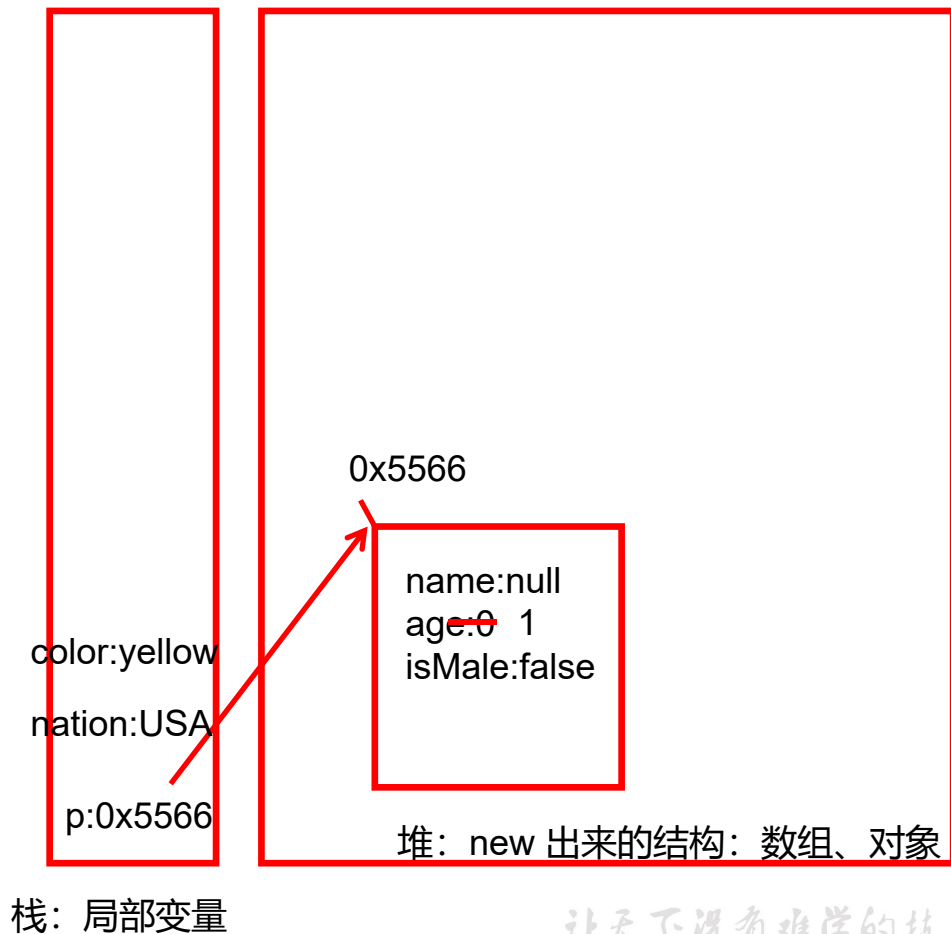
	成员变量	局部变量
声明的位置	直接声明在类中	方法形参或内部、代码块内、构造器内等
修饰符	private、public、static、final等	不能用权限修饰符修饰，可以用final修饰
初始化值	有默认初始化值	没有默认初始化值，必须显式赋值，方可使用
内存加载位置	堆空间 或 静态域内	栈空间



```
class Person{//人类
    //1.属性
    String name;//姓名
    int age = 1;//年龄
    boolean isMale;//是否是男性

    public void show(String nation){
        //nation:局部变量
        String color;//color:局部变量
        color = "yellow";
    }
}

//测试类
class PersonTest{
    public static void main(String[] args){
        Person p = new Person();
        p.show("USA");
    }
}
```





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，如上面的Person及前面讲过的数组。

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



4-5 类的成员之二： 方法(method)



4.5 类的成员之二：方法(method)

什么是方法(method、函数):

- 方法是类或对象行为特征的抽象，用来完成某个功能操作。在某些语言中也称为函数或过程。
- 将功能封装为方法的目的是，可以实现代码重用，简化代码
- **Java**里的方法不能独立存在，所有的方法必须定义在类里。

举例:

```
public class Person{  
    private int age;  
    public int getAge() { //声明方法getAge()  
        return age;  
    }  
    public void setAge(int i) { //声明方法setAge  
        age = i;    //将参数i的值赋给类的成员变量age  
    }  
}
```




方法的声明格式：

```
修饰符 返回值类型 方法名 (参数类型 形参1, 参数类型 形参2, ....) {  
    方法体程序代码  
    return 返回值;  
}
```

其中：

修饰符：**public**,缺省,**private**, **protected**等

返回值类型：

➤ 没有返回值：**void**。

➤ 有返回值，声明出返回值的类型。与方法体中“**return 返回值**”搭配使用
方法名：属于标识符，命名时遵循标识符命名规则和规范，“见名知意”

形参列表：可以包含零个，一个或多个参数。多个参数时，中间用“,” 隔开

返回值：方法在执行完毕后返还给调用它的程序的数据。

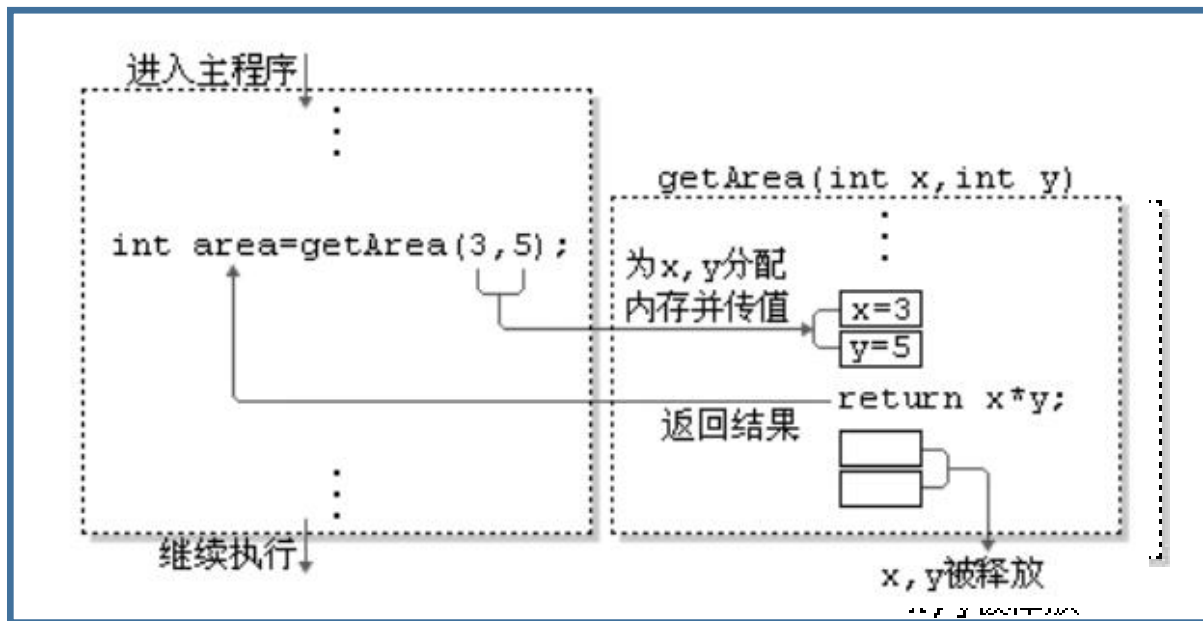
◆ 如何理解方法返回值类型为**void**的情况？



方法的分类：按照是否有形参及返回值

	无返回值	有返回值
无形参	<code>void 方法名 () {}</code>	<code>返回值的类型 方法名 () {}</code>
有形参	<code>void 方法名 (形参列表) {}</code>	<code>返回值的类型 方法名 (形参列表) {}</code>

- 方法的调用
 - 方法通过方法名被调用，且只有被调用才会执行。
- 方法调用的过程分析





●注 意：

- 方法被调用一次，就会执行一次
- 没有具体返回值的情况，返回值类型用关键字**void**表示，那么方法体中可以不必使用**return**语句。如果使用，仅用来结束方法。
- 定义方法时，方法的结果应该返回给调用者，交由调用者处理。
- 方法中只能调用方法或属性，不可以在方法内部定义方法。



练习3

1. 创建一个Person类，其定义如下：

Person
name:String
age:int
sex:int
+study():void
+showAge():void
+addAge(int i):int

要求：(1)创建Person类的对象，设置该对象的name、age和sex属性，调用study方法，输出字符串

“studying”，调用showAge()方法显示age值，调用addAge()方法给对象的age属性值增加2岁。

(2)创建第二个对象，执行上述操作，体会同一个类的不同对象之间的关系。

2. 利用面向对象的编程方法，设计类Circle计算圆的面积。



练习3

- 3.1 编写程序，声明一个method方法，在方法中打印一个10*8 的*型矩形，在main方法中调用该方法。
- 3.2 修改上一个程序，在method方法中，除打印一个10*8的*型矩形外，再计算该矩形的面积，并将其作为方法返回值。在main方法中调用该方法，接收返回的面积值并打印。
- 3.3 修改上一个程序，在method方法提供m和n两个参数，方法中打印一个m*n的*型矩形，并计算该矩形的面积， 将其作为方法返回值。在main方法中调用该方法，接收返回的面积值并打印。



练习3

4. 对象数组题目：

定义类**Student**，包含三个属性：学号**number(int)**，年级**state(int)**，成绩**score(int)**。创建20个学生对象，学号为1到20，年级和成绩都由随机数确定。

问题一：打印出3年级(**state**值为3)的学生信息。

问题二：使用冒泡排序按学生成绩排序，并遍历所有学生信息

提示：

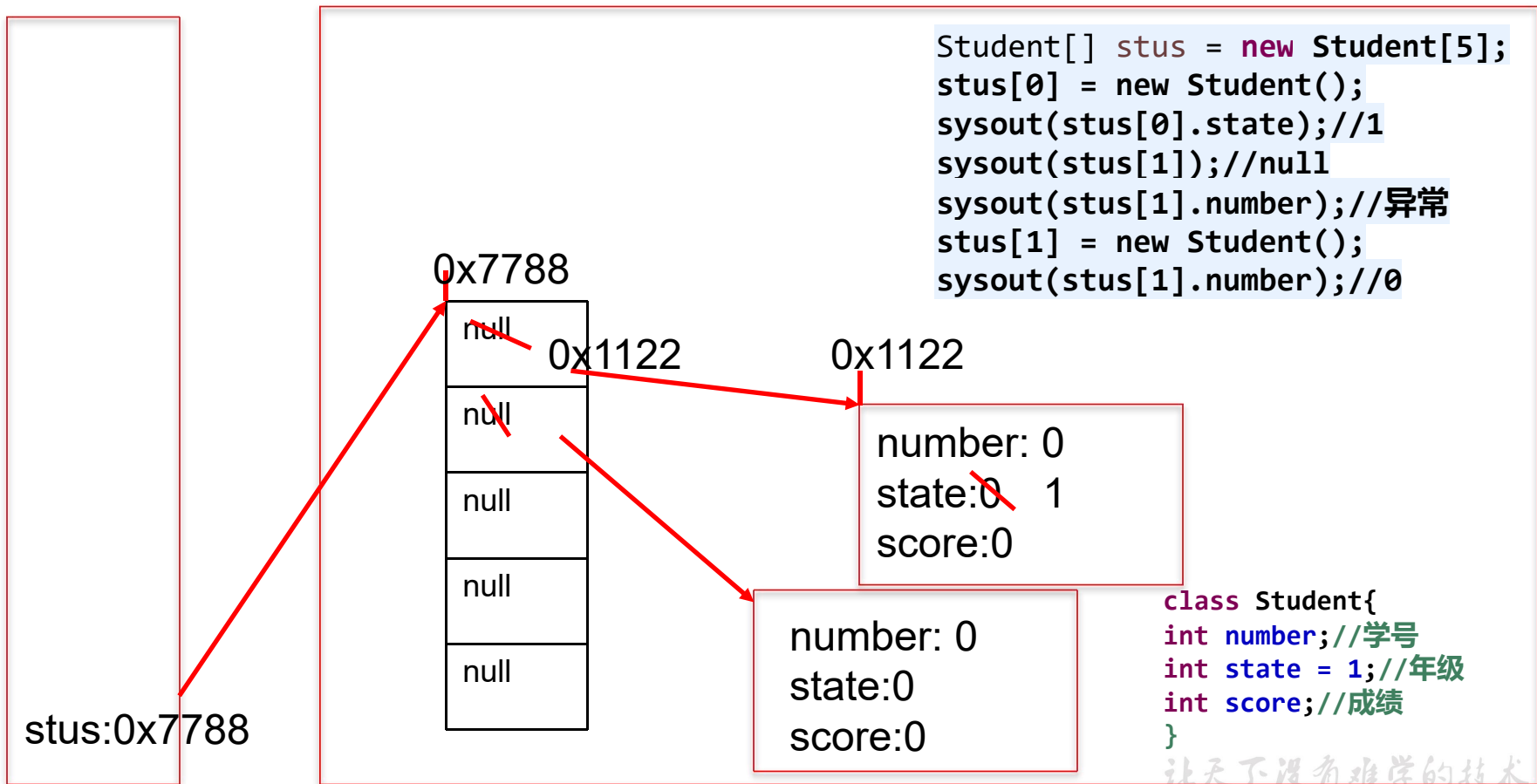
1) 生成随机数：**Math.random()**，返回值类型**double**；

2) 四舍五入取整：**Math.round(double d)**，返回值类型**long**。

5.声明一个日期类型**MyDate**：有属性：年**year**,月**month**，日**day**。创建2个日期对象，分别赋值为：你的出生日期，你对象的出生日期，并显示信息。



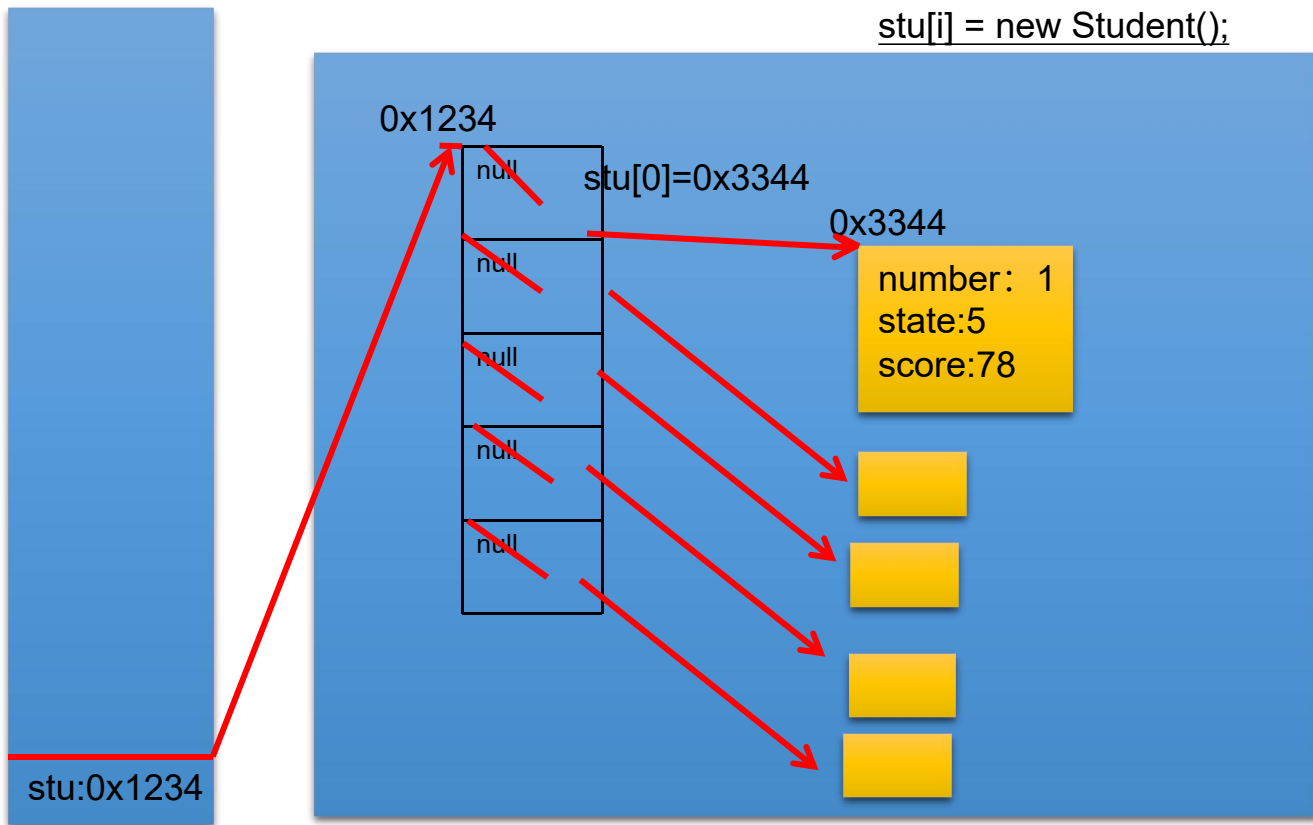
```
Student[] stus = new Student[5];  
stus[0] = new Student();  
sysout(stus[0].state); // 1  
sysout(stus[1]); // null  
sysout(stus[1].number); // 异常  
stus[1] = new Student();  
sysout(stus[1].number); // 0
```





```
Student[] stu = new Student[5];
```

```
stu[i] = new Student();
```



heap:new 出来的东西

让天下没有难学的技术



4-6 再谈方法

4.6.1 方法的重载

4.6.2 可变形参的方法

4.6.3 方法参数的值传递机制

4.6.4 递归方法



重载的概念

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。

重载的特点：

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。

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;}
```



4.6 再谈方法1：方法的重载(overload)

```
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!");  
    }  
}
```



4.6 再谈方法1：方法的重载(overload)

- 使用重载方法，可以为编程带来方便。
- 例如，`System.out.println()`方法就是典型的重载方法，其内部的声明形式如下：

```
public void println(byte x)
public void println(short x)
public void println(int x)
public void println(long x)
public void println(float x)
public void println(double x)
public void println(char x)
public void println(double x)
public void println()
.....
```



练习4

1.判断：

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



练习4

2.编写程序，定义三个重载方法并调用。方法名为mOL。

- 三个方法分别接收一个int参数、两个int参数、一个字符串参数。分别执行平方运算并输出结果，相乘并输出结果，输出字符串信息。
- 在主类的main ()方法中分别用参数区别调用三个方法。

3.定义三个重载方法max()，第一个方法求两个int值中的最大值，第二个方法求两个double值中的最大值，第三个方法求三个double值中的最大值，并分别调用三个方法。



JavaSE 5.0 中提供了**Varargs(variable number of arguments)**机制，允许直接定义能和多个实参相匹配的形参。从而，可以用一种更简单的方式，来传递个数可变的实参。

//JDK 5.0以前：采用数组形参来定义方法，传入多个同一类型变量

```
public static void test(int a ,String[] books);
```

//JDK5.0：采用可变个数形参来定义方法，传入多个同一类型变量

```
public static void test(int a ,String...books);
```




说明：

1. 声明格式：方法名(参数的类型名 ...参数名)
2. 可变参数：方法参数部分指定类型的参数个数是可变多个：0个，1个或多个
3. 可变个数形参的方法与同名的方法之间，彼此构成重载
4. 可变参数方法的使用与方法参数部分使用数组是一致的
5. 方法的参数部分有可变形参，需要放在形参声明的最后
6. 在一个方法的形参位置，最多只能声明一个可变个数形参



4.6 再谈方法2：可变个数的形参

```
public void test(String[] msg){
    System.out.println("含字符串数组参数的test方法 ");
}
public void test1(String book){
    System.out.println("****与可变形参方法构成重载的test1方法****");
}
public void test1(String ... books){
    System.out.println("****形参长度可变的test1方法****");
}
public static void main(String[] args){
    TestOverload to = new TestOverload();
    //下面两次调用将执行第二个test方法
    to.test1();
    to.test1("aa" , "bb");
    //下面将执行第一个test方法
    to.test(new String[]{"aa"});
}
```



- 方法，必须由其所在类或对象调用才有意义。若方法含有参数：

- 形参：方法声明时的参数

- 实参：方法调用时实际传给形参的参数值

- Java的实参值如何传入方法呢？

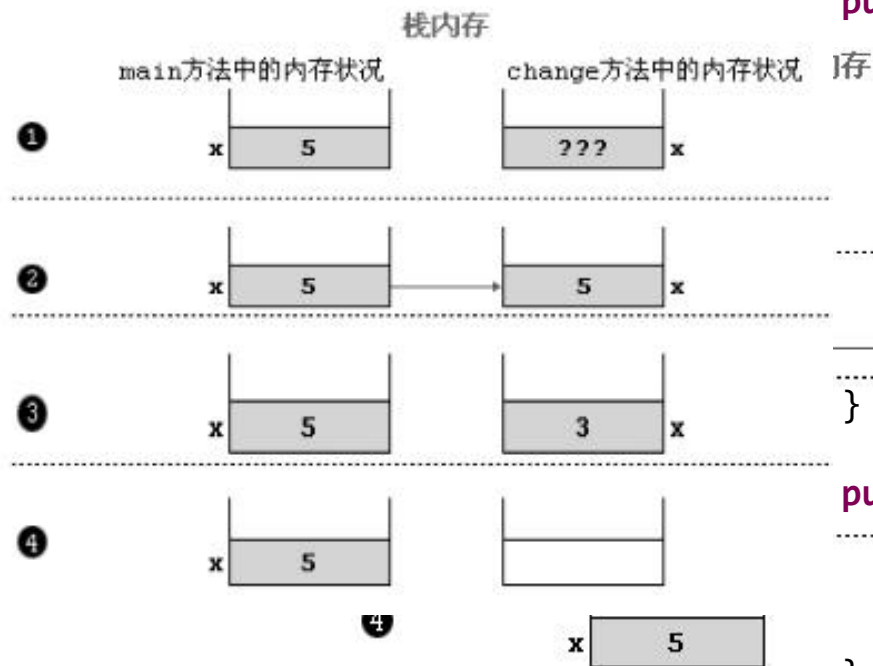
Java里方法的参数传递方式只有一种：**值传递**。即将实际参数值的副本（复制品）传入方法内，而参数本身不受影响。

- 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



—基本数据类型的参数传递



```
public static void main(String[] args) {
    int x = 5;

    change方法中的内存状况
    System.out.println("修改之前x = " + x); // 5
    // x是实参
    change(x);
    System.out.println("修改之后x = " + x); // 5
}

public static void change(int x) {
    System.out.println("change:修改之前x = " + x);
    x = 3;
    System.out.println("change:修改之后x = " + x);
}
```



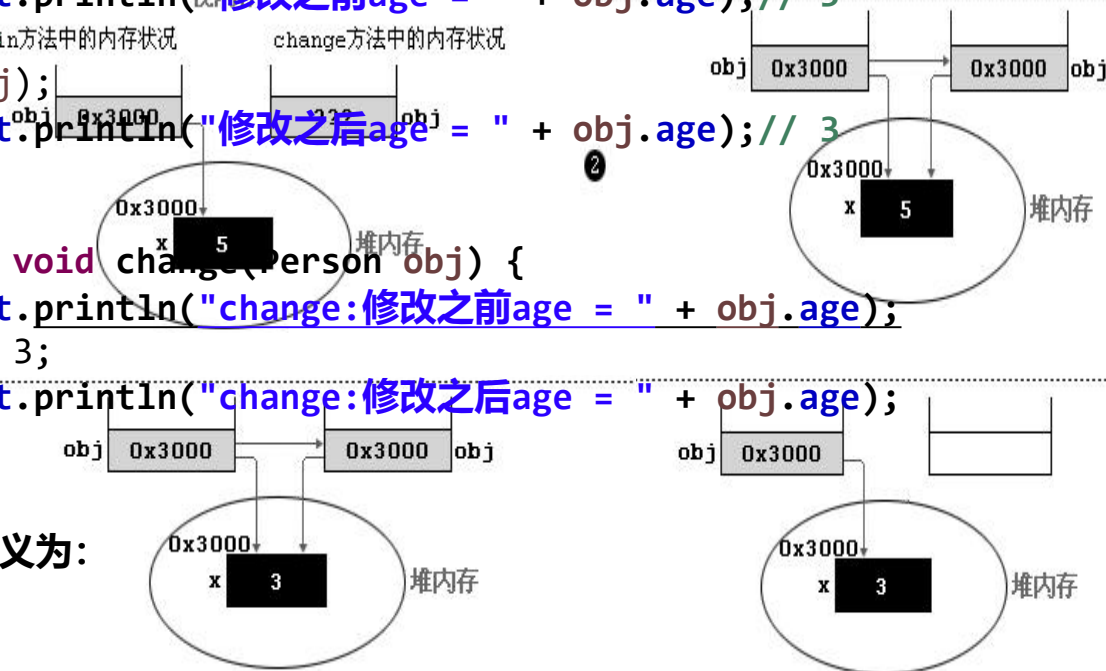
— 引用数据类型的参数传递

```
public static void main(String[] args) {  
    Person obj = new Person();  
    obj.age = 5;  
    System.out.println("修改之前age = " + obj.age); // 5  
    // x是实参  
    change(obj);  
    System.out.println("修改之后age = " + obj.age); // 3  
}
```

```
public static void change(Person obj) {  
    System.out.println("change:修改之前age = " + obj.age);  
    obj.age = 3;  
    System.out.println("change:修改之后age = " + obj.age);  
}
```

其中Person类定义为:

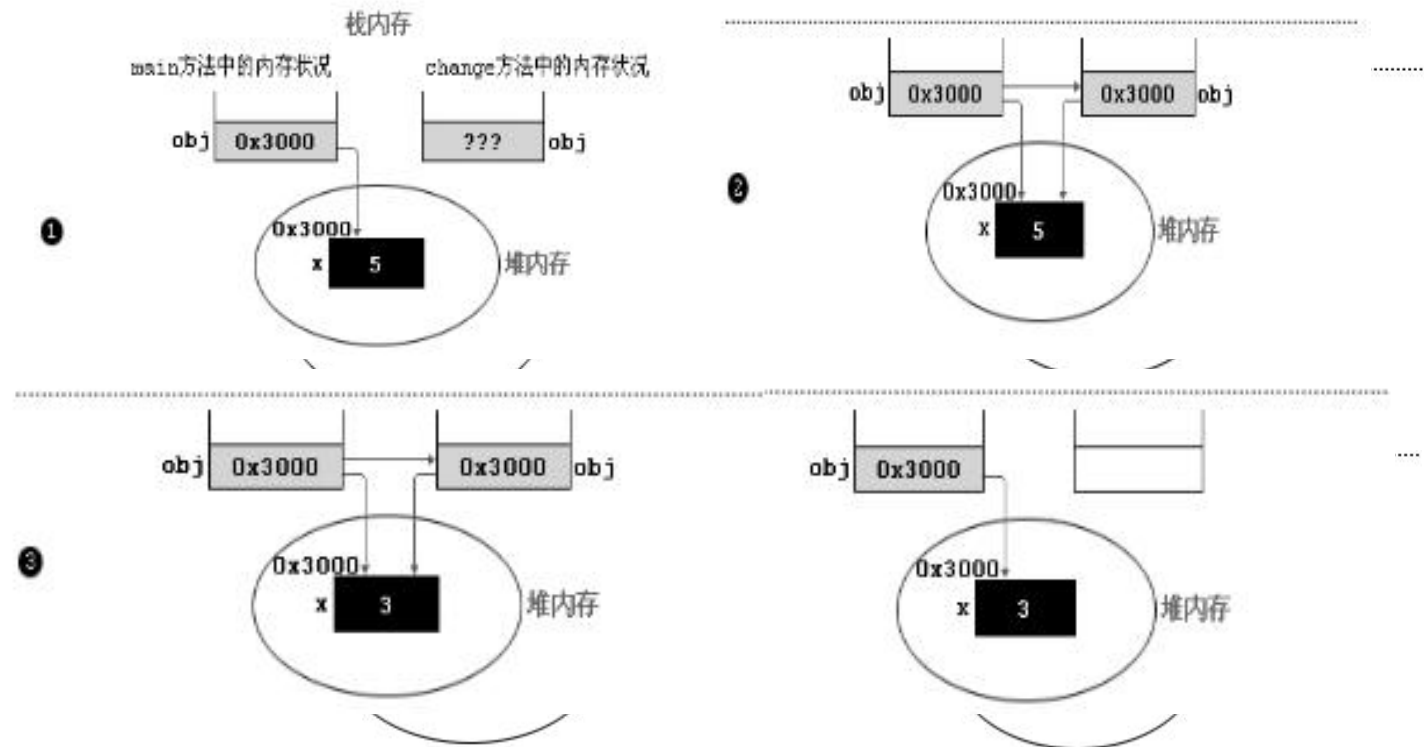
```
class Person{  
    int age;  
}
```



图示见下图



— 引用数据类型的参数传递





——引用数据类型的参数传递

```

public static void main(String[] args) {
    Person obj = new Person();
    obj.age = 5;
    System.out.println("修改之前age = " + obj.age); // 5
    // x是实参
    change(obj);
    System.out.println("修改之后age = " + obj.age); // 5
}

```

```

public static void change(Person obj) {
    obj = new Person();
    System.out.println("change:修改之前age = " + obj.age);
    obj.age = 3;
    System.out.println("change:修改之后age = " + obj.age);
}

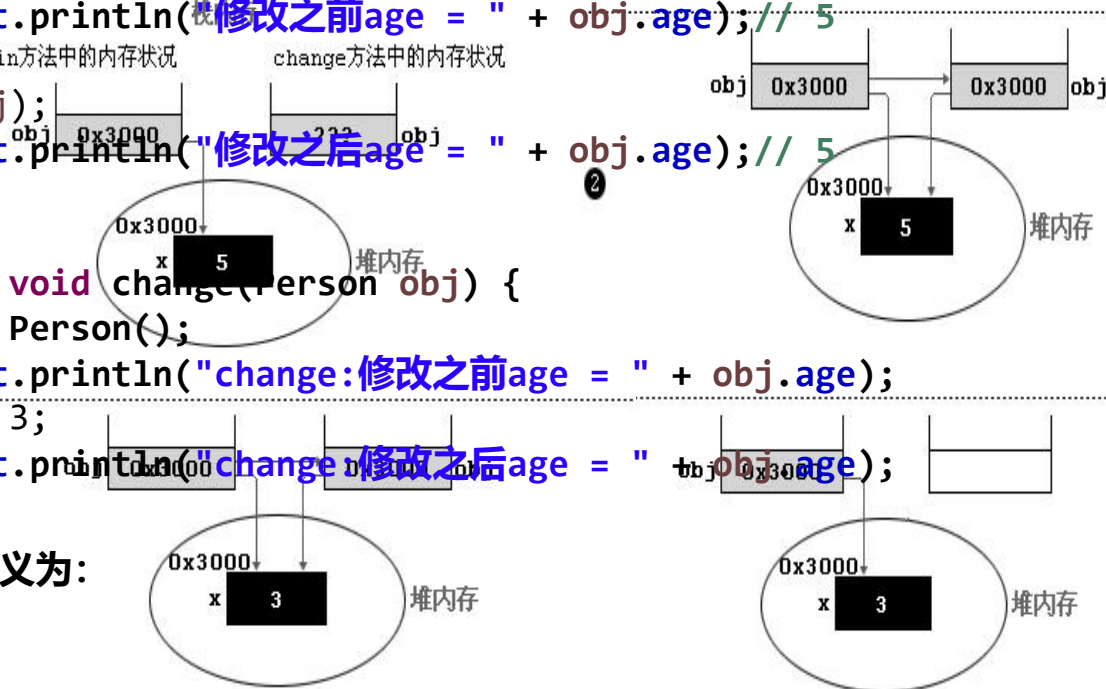
```

其中Person类定义为:

```

class Person{
    int age;
}

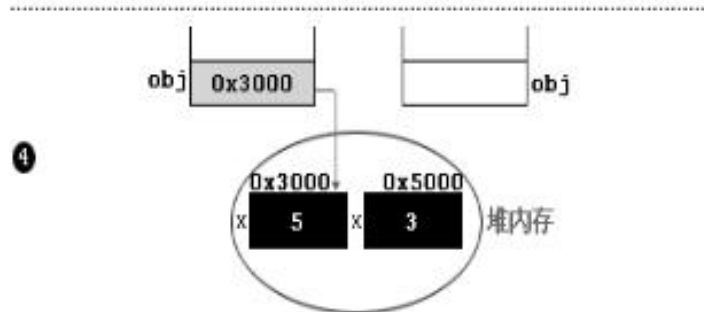
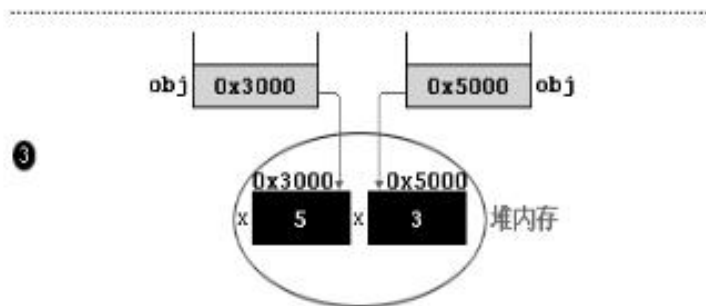
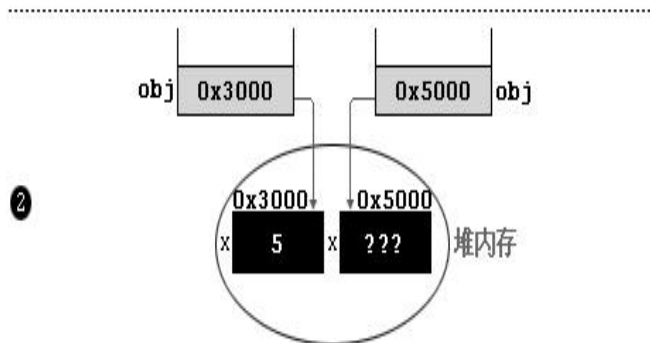
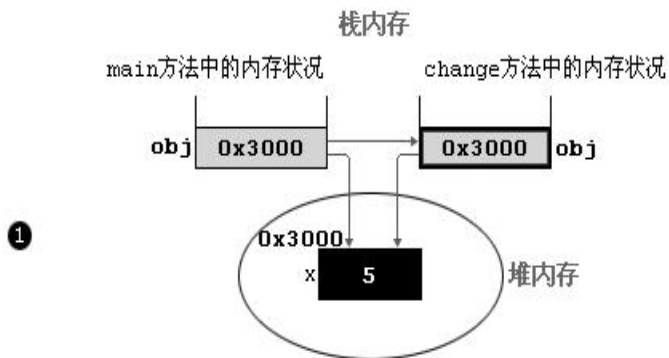
```



图示见下图



— 引用数据类型的参数传递





例题1：方法的参数传递

```
public class TransferTest1 {  
    public void swap(int a, int b) {  
        int tmp = a;  
        a = b;  
        b = tmp;  
        System.out.println("swap方法里, a的值是" + a + "; b的值是" + b);  
    }  
  
    public static void main(String[] args) {  
        TransferTest1 test = new TransferTest1();  
        int a = 5;  
        int b = 10;  
        test.swap(a, b);  
        System.out.println("交换结束后, 变量a的值是" + a + "; 变量b的值是" + b);  
    }  
}
```

请输出结果



```
main(){  
    int m = 10;  
    int n = 20;  
    v.swap(m,n);  
    sysout(m,n);  
}
```

```
swap(int m ,in n){  
    int temp = m;  
    m = n;  
    n = temp;  
    sysout(m,n);  
}
```

swap()

main()

temp:10

~~n:20~~ 10

~~m:10~~ 20

n:20

m: 10



例题2：方法的参数传递

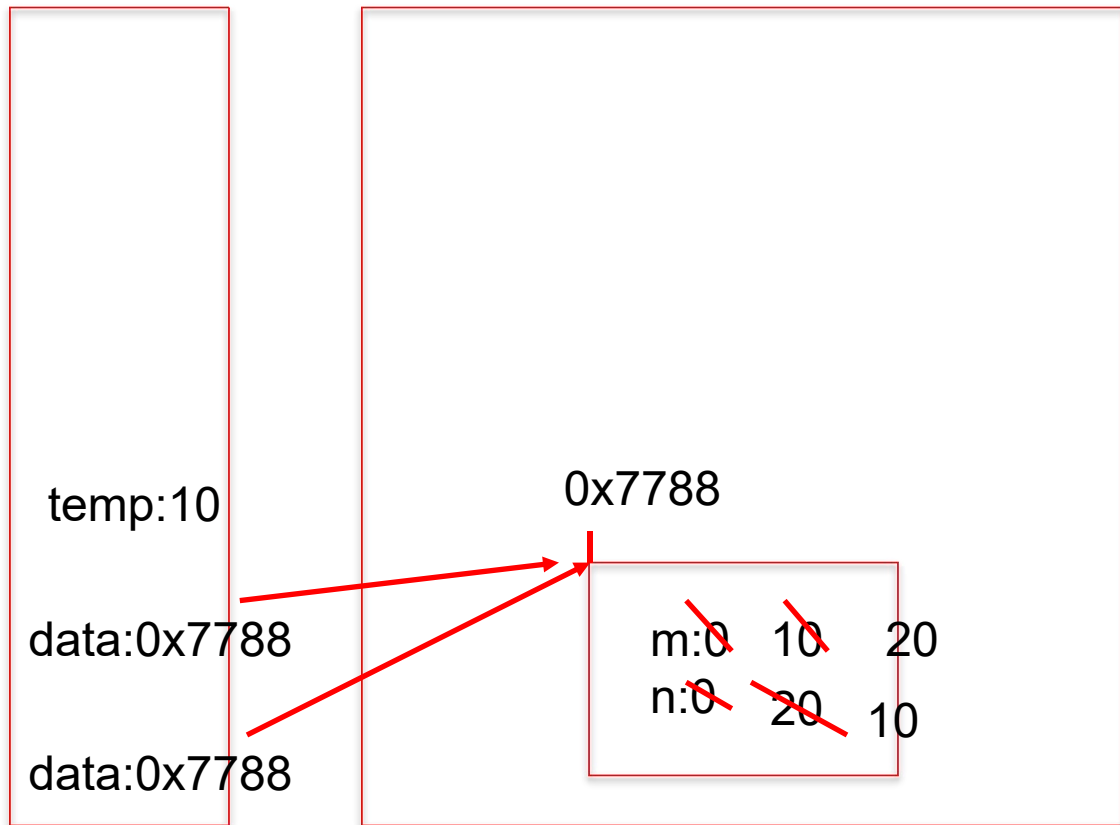
```
class DataSwap {  
    public int a;  
    public int b;  
}  
  
public class TransferTest2 {  
    public static void swap(DataSwap ds) {  
        int temp = ds.a;  
        ds.a = ds.b;  
        ds.b = temp;  
        System.out.println("swap方法里, a Field的值是" + ds.a + ";b Field的值是" + ds.b);  
    }  
  
    public static void main(String[] args) {  
        DataSwap ds = new DataSwap();  
        ds.a = 5;  
        ds.b = 10;  
        swap(ds);  
        System.out.println("交换结束后, a Field的值是" + ds.a + ";b Field的值是" + ds.b);  
    }  
}
```

请输出结果



```
class Data{  
    int m;  
    int n;  
}
```

```
main(){  
    Data data = new Data();  
    data.m = 10;  
    data.n = 20;  
    v.swap(data);  
    sysout(data.m,data.n);  
}  
swap(Data data){  
    int temp = data.m;  
    data.m = data.n;  
    data.n = temp;  
}
```





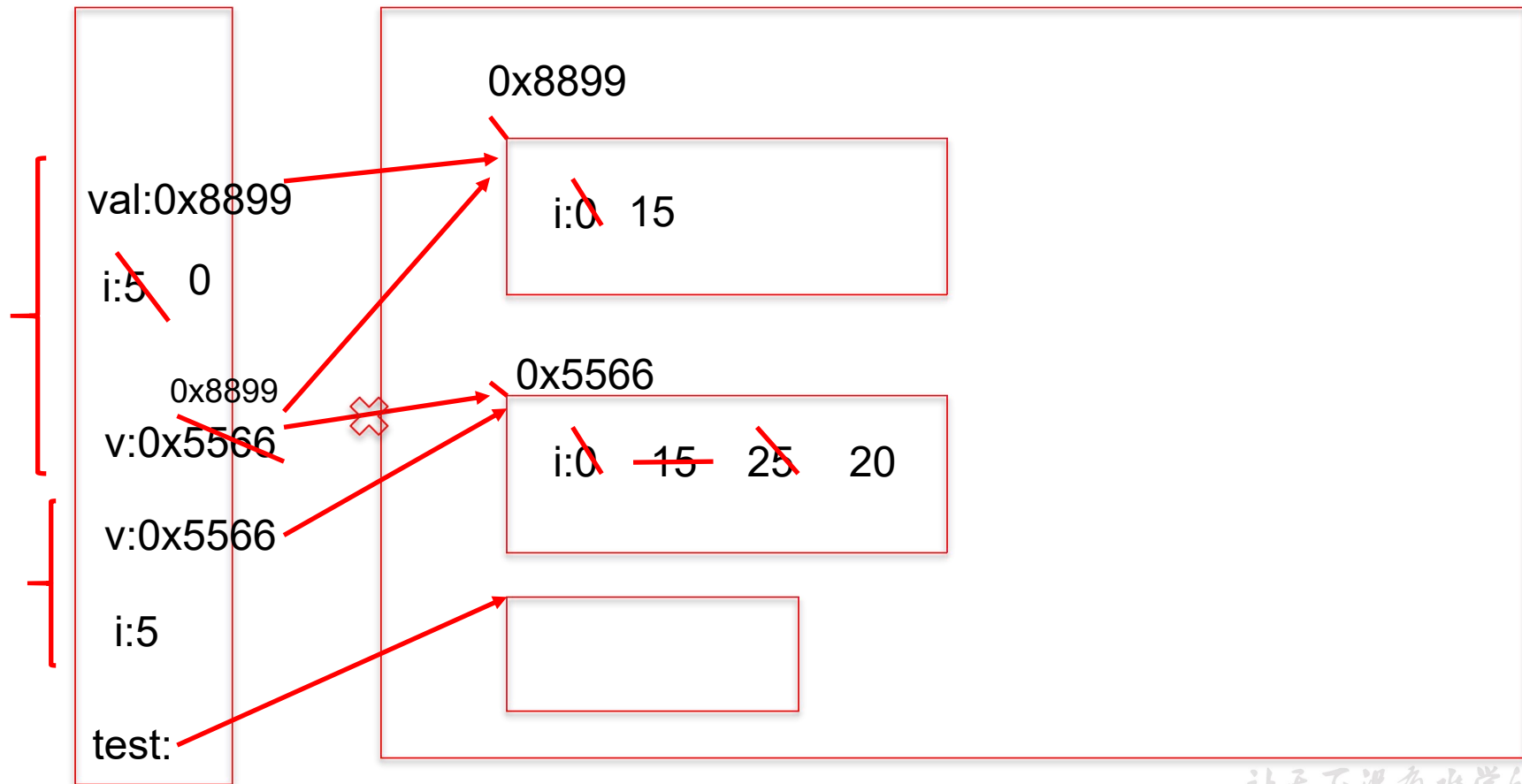
例题3：方法的参数传递

```
public class TransferTest3 {  
    public static void main(String args[]) {  
        TransferTest3 test = new TransferTest3();  
        test.first();  
    }  
    public void first() {  
        int i = 5;  
        Value v = new Value();  
        v.i = 25;  
        second(v, i);  
        System.out.println(v.i);  
    }  
    public void second(Value v, int i) {  
        i = 0;  
        v.i = 20;  
        Value val = new Value();  
        v = val;  
        System.out.println(v.i + " " + i);  
    }  
}
```

```
class Value {  
    int i = 15;  
}
```



例题3：方法的参数传递图示





练习5：貌似是考查方法的参数传递

附加题：(每题 10 分)

1. `public class Test {`
 `public static void main(String[] args) {`
 `int a=10;`
 `int b=10;`

`method(a,b);` //需要在method方法被调用之后，仅打印出a=100,b=200。请写出method方法的代码

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

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

`}`

 //代码编写处



微软:

定义一个int型的数组: `int[] arr = new int[]{12,3,3,34,56,77,432};`
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。

答案:

//错误写法

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

//正确写法1

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

//正确写法2

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




```
int[] arr = new int[10];  
System.out.println(arr); //地址值?
```

```
char[] arr1 = new char[10];  
System.out.println(arr1); //地址值?
```



练习6：将对象作为参数传递给方法

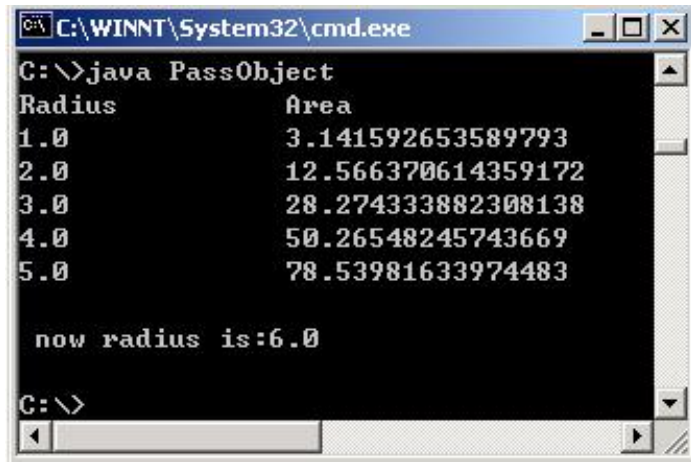
(1) 定义一个Circle类，包含一个double型的radius属性代表圆的半径，一个findArea()方法返回圆的面积。

(2) 定义一个类PassObject，在类中定义一个方法printAreas()，该方法的定义如下：**public void printAreas(Circle c, int time)**

在printAreas方法中打印输出1到time之间的每个整数半径值，以及对应的面积。

例如，times为5，则输出半径1，2，3，4，5，以及对应的圆面积。

(3) 在main方法中调用printAreas()方法，调用完毕后输出当前半径值。程序运行结果如图所示。



```
C:\WINNT\System32\cmd.exe
C:\>java PassObject
Radius      Area
1.0         3.141592653589793
2.0         12.566370614359172
3.0         28.274333882308138
4.0         50.26548245743669
5.0         78.53981633974483

now radius is:6.0
C:\>
```



- **递归方法：一个方法体内调用它自身。**
- 方法递归包含了一种隐式的循环，它会重复执行某段代码，但这种重复执行无须循环控制。
- 递归一定要向已知方向递归，否则这种递归就变成了无穷递归，类似于死循环。

//计算1-100之间所有自然数的和

```
public int sum(int num){  
    if(num == 1){  
        return 1;  
    }else{  
        return num + sum(num - 1);  
    }  
}
```



练习 7

练习7.1: 请用Java写出递归求阶乘($n!$)的算法

练习7.2: 已知有一个数列: $f(0) = 1, f(1) = 4, f(n+2) = 2 * f(n+1) + f(n)$, 其中 n 是大于0的整数, 求 $f(10)$ 的值。

练习7.3: 已知一个数列: $f(20) = 1, f(21) = 4, f(n+2) = 2 * f(n+1) + f(n)$, 其中 n 是大于0的整数, 求 $f(10)$ 的值。

练习7.4: 输入一个数据 n , 计算斐波那契数列(Fibonacci)的第 n 个值

1 1 2 3 5 8 13 21 34 55

规律: 一个数等于前两个数之和

要求: 计算斐波那契数列(Fibonacci)的第 n 个值, 并将整个数列打印出来



拓展:

宋老师，我今天去百度面试，遇到一个一个双重递归调用的问题，我琢磨了一下，完全不知道为什么。打断点了，也还是没看懂为什么程序会那样走。您有空可以看一下，求指教

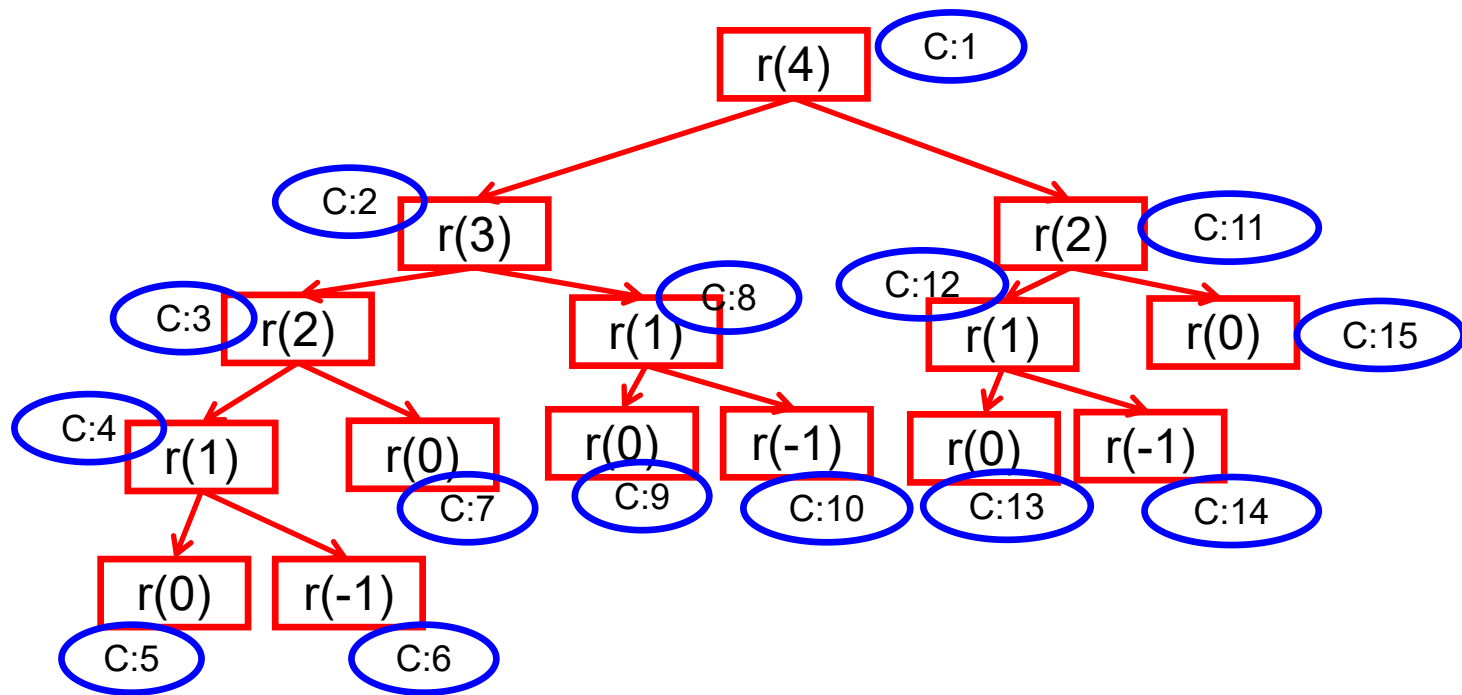
```
/**
 * 测试递归调用的次数
 */
@Test
public void binomial() { recursion(10); }

private static int count = 0 ;
public static int recursion(int k){
    count++;
    System.out.println("count1:"+count + "    k:" + k);
    if(k<=0){
        return 0 ;
    }
    return recursion(k-1) + recursion(k-2);//287
    //return recursion(k-1);//11
    //return recursion(k-1) + recursion(k-1);//2047
}
```

1 test passed - 8ms

```
count1:274 k:3
count1:275 k:2
count1:276 k:1
count1:277 k:0
count1:278 k:-1
count1:279 k:0
count1:280 k:1
count1:281 k:0
count1:282 k:-1
count1:283 k:2
count1:284 k:1
count1:285 k:0
count1:286 k:-1
count1:287 k:0

Process finished with exit code 0
```





4-7 面向对象特征之一： 封装与隐藏



- 为什么需要封装？封装的作用和含义？
 - 我要用洗衣机，只需要按一下开关和洗涤模式就可以了。有必要了解洗衣机内部的结构吗？有必要碰电动机吗？
 - 我要开车，...
- 我们程序设计追求“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。**



4.7 面向对象特征之一：封装和隐藏

使用者对类内部定义的属性(对象的成员变量)的直接操作会导致数据的错误、混乱或安全性问题。

```
class Animal {  
    public int legs;  
    public void eat(){  
        System.out.println("Eating");  
    }  
    public void move(){  
        System.out.println("Moving.");  
    }  
}  
  
public class Zoo {  
    public static void main(String args[]) {  
        Animal xb = new Animal();  
        xb.legs = 4;  
        System.out.println(xb.legs);  
        xb.eat();  
        xb.move();  
    }  
}
```

应该将**legs**属性保护起来，防止乱用。

保护的方式：信息隐藏

问题：**xb.legs = -1000;**



信息的封装和隐藏

Java中通过将数据声明为私有的(private)，再提供公共的 (public) 方法:**getXxx()**和**setXxx()**实现对该属性的操作，以实现下述目的：

- **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；



4.7 面向对象特征之一：封装和隐藏

```
class Animal {  
    private int legs; // 将属性legs定义为private, 只能被Animal类内部访问  
    public void setLegs(int i) { // 在这里定义方法 eat() 和 move()  
        if (i != 0 && i != 2 && i != 4) {  
            System.out.println("Wrong number of legs!");  
            return;  
        }  
        legs = i;  
    }  
    public int getLegs() {  
        return legs;  
    }  
}  
public class Zoo {  
    public static void main(String args[]) {  
        Animal xb = new Animal();  
        xb.setLegs(4); // xb.setLegs(-1000);  
        //xb.legs = -1000; // 非法  
        System.out.println(xb.getLegs());  
    }  
}
```



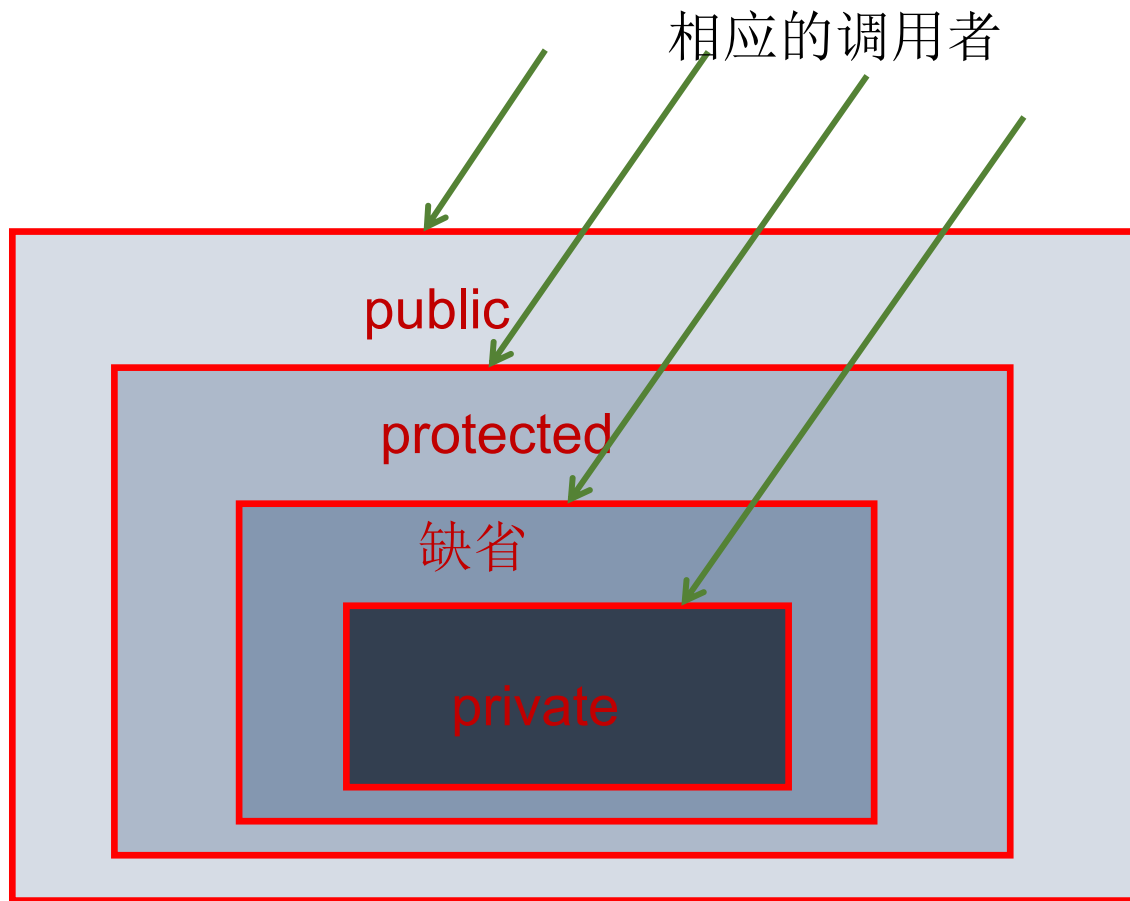
四种访问权限修饰符

Java权限修饰符public、protected、(缺省)、private置于**类的成员**定义前，用来限定对象对该类成员的访问权限。

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。





练习8

1.创建程序,在其中定义两个类：**Person**和**PersonTest**类。定义如下：
用**setAge()**设置人的合法年龄(0~130)，用**getAge()**返回人的年龄。
在**PersonTest**类中实例化**Person**类的对象b，调用**setAge()**和**getAge()**方法，体会Java的封装性。

Person
-age:int
+setAge(i: int)
+getAge(): int



4-8 类的成员之三： 构造器(或构造方法)



●构造器的特征

- 它具有与类相同的名称
- 它不声明返回值类型。（与声明为void不同）
- 不能被static、final、synchronized、abstract、native修饰，不能有return语句返回值

●构造器的作用：创建对象；给对象进行初始化

- 如：Order o = new Order(); Person p = new Person("Peter",15);
- 如同我们规定每个“人”一出生就必须先洗澡，我们就可以在“人”的构造器中加入完成“洗澡”的程序代码，于是每个“人”一出生就会自动完成“洗澡”，程序就不必再在每个人刚出生时一个一个地告诉他们要“洗澡”了。



4.8 类的成员之三：构造器(构造方法)

- 语法格式：

修饰符 类名 (参数列表) {
 初始化语句;

}

- 举 例：

```
public class Animal {  
    private int legs;  
    // 构造器  
    public Animal() {  
        legs = 4;  
    }  
    public void setLegs(int i) {  
        legs = i;  
    }  
    public int getLegs() {  
        return legs;  
    }  
}
```

创建Animal类的实例：**Animal a = new Animal();**
调用构造器，将legs初始化为4。



4.8 类的成员之三：构造器(构造方法)

- 根据参数不同，构造器可以分为如下两类：
 - 隐式无参构造器（系统默认提供）
 - 显式定义一个或多个构造器（无参、有参）
- 注意：
 - Java语言中，每个类都至少有一个构造器
 - 默认构造器的修饰符与所属类的修饰符一致
 - 一旦显式定义了构造器，则系统不再提供默认构造器
 - 一个类可以创建多个重载的构造器
 - 父类的构造器不可被子类继承



练习9

1. 在前面定义的**Person**类中添加构造器，利用构造器设置所有人的**age**属性初始值都为18。
2. 修改上题中类和构造器，增加**name**属性,使得每次创建**Person**对象的同时初始化对象的**age**属性值和**name**属性值。

Person
-name:String
+setName(i: String)
+getName(): String



练习9

3.编写两个类，TriAngle和TriAngleTest，其中TriAngle类中声明私有的底边长**base**和高**height**，同时声明公共方法访问私有变量。此外，提供类必要的构造器。另一个类中使用这些公共方法，计算三角形的面积。



构造器重载

- 构造器一般用来创建对象的同时初始化对象。如

```
class Person{  
    String name;  
    int age;  
    public Person(String n , int a){ name=n; age=a;}  
}
```

- 构造器重载使得对象的创建更加灵活，方便创建各种不同的对象。

构造器重载举例：

```
public class Person{  
    public Person(String name, int age, Date d) {this(name,age);...}  
    public Person(String name, int age) {...}  
    public Person(String name, Date d) {...}  
    public Person(){...}  
}
```

- 构造器重载，参数列表**必须**不同



构造器重载举例

```
public class Person {  
    private String name;  
    private int age;  
    private Date birthDate;  
    public Person(String n, int a, Date d) {  
        name = n;  
        age = a;  
        birthDate = d;  
    }  
    public Person(String n, int a) {  
        name = n;  
        age = a;  
    }  
    public Person(String n, Date d) {  
        name = n;  
        birthDate = d;  
    }  
    public Person(String n) {  
        name = n;  
        age = 30;  
    }  
}
```



练习10

(1)定义Student类,有4个属性:

String name;

int age;

String school;

String major;

(2)定义Student类的3个构造器:

➤ 第一个构造器Student(String n, int a)设置类的name和age属性;

➤ 第二个构造器Student(String n, int a, String s)设置类的name, age 和school属性;

➤ 第三个构造器Student(String n, int a, String s, String m)设置类的name, age ,school和major属性;

(3)在main方法中分别调用不同的构造器创建的对象, 并输出其属性值。



截止到目前，我们讲到了很多位置都可以对类的属性赋值。现总结这几个位置，并指明赋值的先后顺序。

- 赋值的位置：

- ① 默认初始化
- ② 显式初始化
- ③ 构造器中初始化
- ④ 通过“对象.属性”或“对象.方法”的方式赋值

- 赋值的先后顺序：

- ① - ② - ③ - ④



JavaBean

- JavaBean是一种Java语言写成的可重用组件。
- 所谓javaBean，是指符合如下标准的Java类：
 - 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。用户可以认为JavaBean提供了一种随时随地的复制和粘贴的功能，而不用担心任何改变。

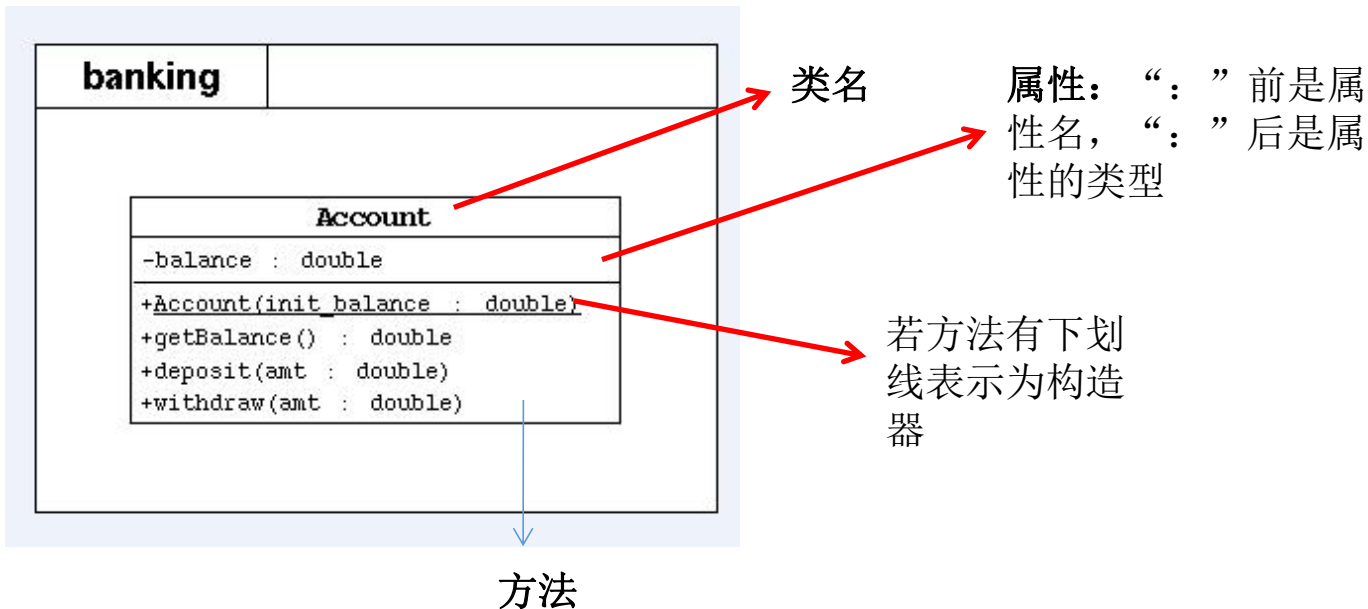


JavaBean示例

```
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;  
    }  
}
```



4.8 拓展知识：UML类图



1. + 表示 public 类型, - 表示 private 类型, #表示protected类型

2. 方法的写法:

方法的类型(+、-) 方法名(参数名: 参数类型): 返回值类型



4-9 关键字：this的使用



this是什么？

- 在Java中，**this**关键字比较难理解，它的作用和其词义很接近。
 - 它在方法内部使用，即这个方法所属对象的引用；
 - 它在构造器内部使用，表示该构造器正在初始化的对象。
- **this** 可以调用类的属性、方法和构造器
- 什么时候使用**this**关键字呢？
 - 当在方法内需要用到调用该方法的对象时，就用**this**。
具体的：我们可以用**this**来区分**属性**和**局部变量**。
比如：**this.name** = **name**;



● 使用this，调用属性、方法

```
class Person{                                // 定义Person类
    private String name ;
    private int age ;
    public Person(String name,int age){
        this.name = name ;
        this.age = age ; }
    public void getInfo(){
        System.out.println("姓名： " + name) ;
        this.speak();
    }
    public void speak(){
        System.out.println("年龄： " + this.age);
    }
}
```

1. 在任意方法或构造器内，如果使用当前类的成员变量或成员方法可以在其前面添加this，增强程序的阅读性。不过，通常我们都习惯省略this。
2. 当形参与成员变量同名时，如果在方法内或构造器内需要使用成员变量，必须添加this来表明该变量是类的成员变量
3. 使用this访问属性和方法时，如果在本类中未找到，会从父类中查找



4.9 关键字—this

```
class Person{ // 定义Person类
    String name;
    Person(String name){
        this.name = name;}
    public void getInfo(){
        System.out.println("Person类 --> " + this.name); }
    public boolean compare(Person p){
        return this.name==p.name;
    } }
}
```

当前正在操作本方法的对象称为当前对象。

```
public class PersonTest{
    public static void main(String args[]){
        Person per1 = new Person("张三");
        Person per2 = new Person("李四");
        per1.getInfo();    // 当前调用getInfo()方法的对象是per1
        per2.getInfo();    // 当前调用getInfo()方法的对象是per2
        boolean b = per1.compare(per2);
    } }
}
```



● 使用this调用本类的构造器

```
class Person{                                // 定义Person类
    private String name ;
    private int age ;
    public Person(){                          // 无参构造器
        System.out.println("新对象实例化");
    }
    public Person(String name){
        this();    // 调用本类中的无参构造器
        this.name = name ;
    }
    public Person(String name,int age){
        this(name); // 调用有一个参数的构造器
        this.age = age;
    }
    public String getInfo(){
        return "姓名: " + name + ", 年龄: " + age ;
    }
}
```

4.this可以作为一个类中构造器相互调用的特殊格式



注意：

- 可以在类的构造器中使用"**this(形参列表)**"的方式，调用本类中重载的其他的构造器！
- 明确：构造器中不能通过"**this(形参列表)**"的方式调用自身构造器
- 如果一个类中声明了n个构造器，则最多有 $n - 1$ 个构造器中使用了"**this(形参列表)**"
- "**this(形参列表)**"必须声明在类的构造器的首行！
- 在类的一个构造器中，最多只能声明一个"**this(形参列表)**"



练习11

添加必要的构造器，综合应用构造器的重载，**this**关键字。

Boy

-name:String

-age:int

+setName(i: String)

+getName(): String

+setAge(i: int)

+getAge(): int

+marry(girl:Girl)

+shout():void

Girl

-name:String

-age:int

+setName(i: String)

+getName(): String

+marry(boy:Boy)

+compare(girl:Girl)



练习12

实验1: Account_Customer.doc

实验2: Account_Customer_Bank.doc



4-10 关键字：package、import的使用



4.10 关键字—package

- **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 ;

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()");
    }
}
```

- 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；
- 包通常用小写单词标识。通常使用所在公司域名的倒置：**com.atguigu.xxx**



4.10 关键字—package

源文件布局：

- Java 源文件的基本语法：

```
[<包声明>]  
    [<导入声明>]  
    <类声明>+
```

- 示例，VehicleCapacityReport.java 文件：

```
package shipping.reports;  
  
import shipping.domain.*;  
import java.util.List;  
import java.io.*;  
  
public class VehicleCapacityReport {  
    private List  vehicles;  
    public void generateReport(Writer output) {...}  
}
```

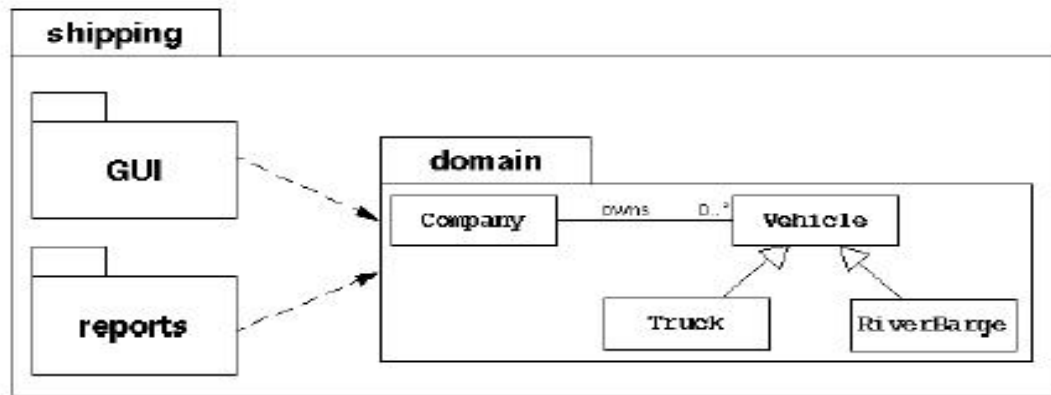


4.10 关键字—package

包的作用：

- 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。比如：MVC的设计模式
- 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
- 控制访问权限

例：某航运软件系统包括：一组域对象、GUI和reports子系统





MVC设计模式

MVC是常用的设计模式之一，将整个程序分为三个层次：**视图模型层**，**控制器层**，与**数据模型层**。这种将程序输入输出、数据处理，以及数据的展示分离开来的设计模式使程序结构变的灵活而且清晰，同时也描述了程序各个对象间的通信方式，降低了程序的耦合性。

模型层 **model** 主要处理数据

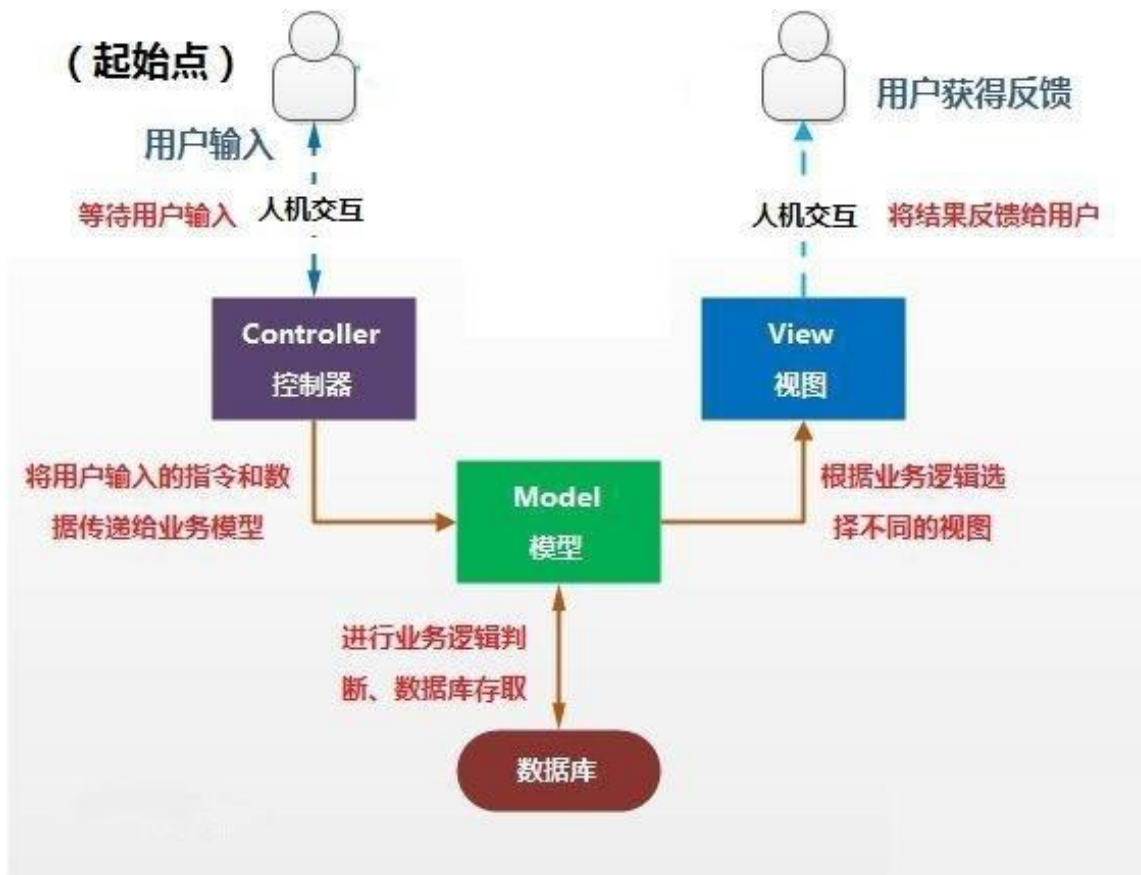
- >数据对象封装 `model.bean/domain`
- >数据库操作类 `model.dao`
- >数据库 `model.db`

控制层 **controller** 处理业务逻辑

- >应用界面相关 `controller.activity`
- >存放fragment `controller.fragment`
- >显示列表的适配器 `controller.adapter`
- >服务相关的 `controller.service`
- >抽取的基类 `controller.base`

视图层 **view** 显示数据

- >相关工具类 `view.utils`
- >自定义view `view.ui`





JDK中主要的包介绍

1. **java.lang**----包含一些Java语言的核心类，如String、Math、Integer、System和Thread，提供常用功能
2. **java.net**----包含执行与网络相关的操作的类和接口。
3. **java.io** ----包含能提供多种输入/输出功能的类。
4. **java.util**----包含一些实用工具类，如定义系统特性、接口的集合框架类、使用与日期日历相关的函数。
5. **java.text**----包含了一些java格式化相关的类
6. **java.sql**----包含了java进行JDBC数据库编程的相关类/接口
7. **java.awt**----包含了构成抽象窗口工具集（abstract window toolkits）的多个类，这些类被用来构建和管理应用程序的图形用户界面(GUI)。 B/S C/S



4.10 关键字—import

- 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。

- 语法格式：

import 包名. 类名;

- 应用举例：

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



●注意：

1. 在源文件中使用import显式的导入指定包下的类或接口
2. 声明在包的声明和类的声明之间。
3. 如果需要导入多个类或接口，那么就并列显式多个import语句即可
4. 举例：可以使用java.util.*的方式，一次性导入util包下所有的类或接口。
5. 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。
6. 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。
7. 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。
8. import static组合的使用：调用指定类或接口下的静态的属性或方法

让天下没有难学的技术



尚硅谷