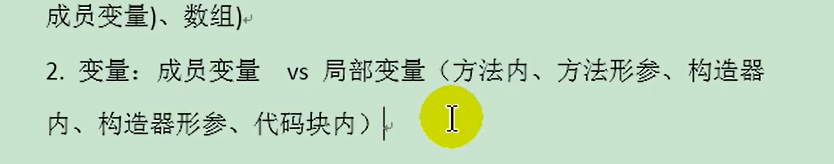
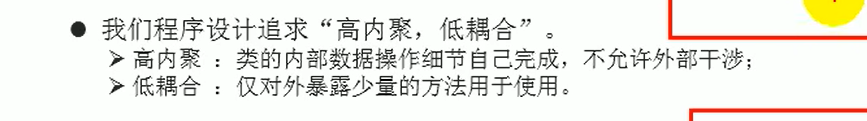


注意是new出来的结构,对象名还在栈



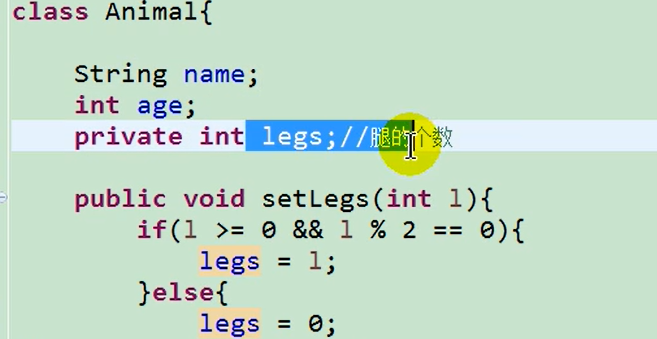
Static修饰的对象在方法区的静态域中

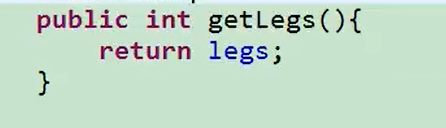
String是比较特殊的类,定义的对象赋值时储存在字符串常量池中,实际上是char[]组成的而且一旦定赋值了就不能改变,而当我们重新赋值时是在常量池中新造一个值然后把地址赋给对象

****

**属性的封装性, S s=new S(); s.name=xxx; 这时如果要对赋值进行制约,要写一个setName方法(),同时禁用s.name方式对name属性赋值,禁用方法为在S class中private String name;加一个权限修饰符,设置为私有的,不可对外直接可见的,不能在其他类中直接对象.name调用,但是可以在同类的方法中进行直接调用修改.**

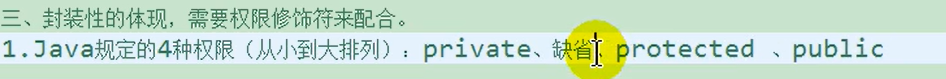
****

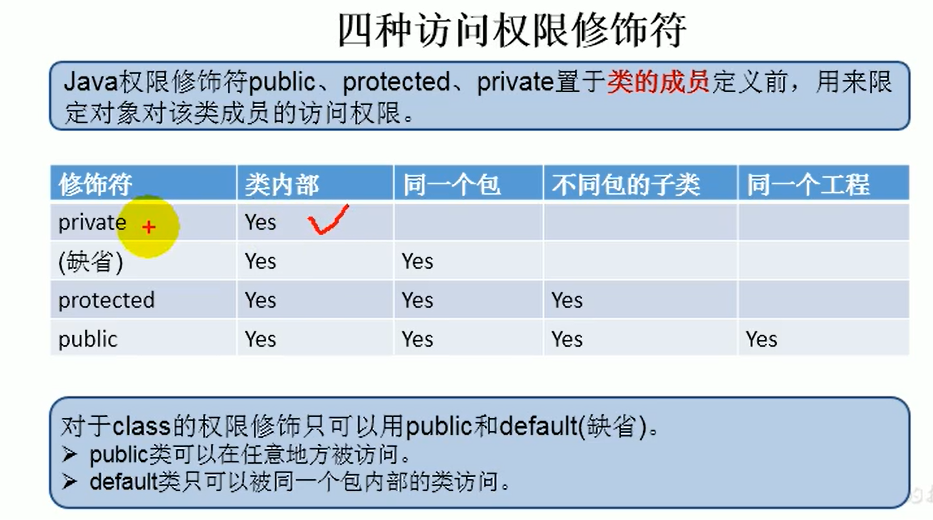
****

****

**同理同一个类中的方法也可以加private,只能在此类中其他方法使用**

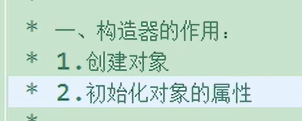
**封装性的体现就体现在设置了权限修饰符**

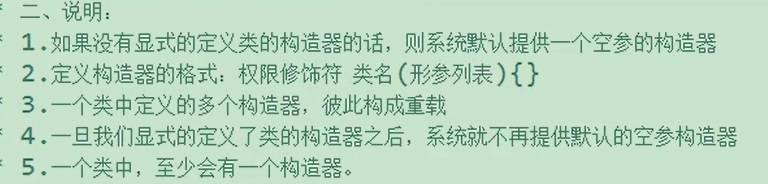
****

****

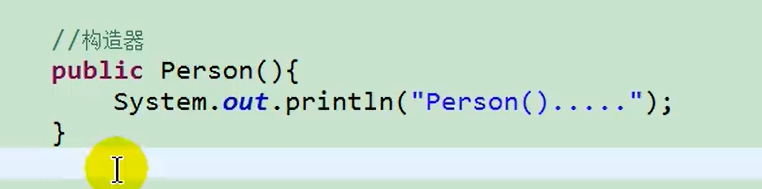
****

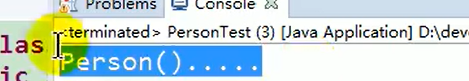
**注意，默认提供的构造器权限类型跟定义的类的权限类型一样**

****

****

****

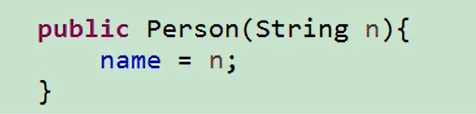
****

****

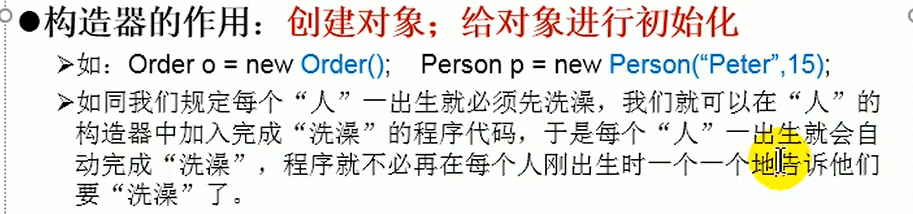
**实例化对象时会自动触发构造器里面的方法**

**构造器虽然和方法很相似但是不是一回事,构造器是用来创造对象的**

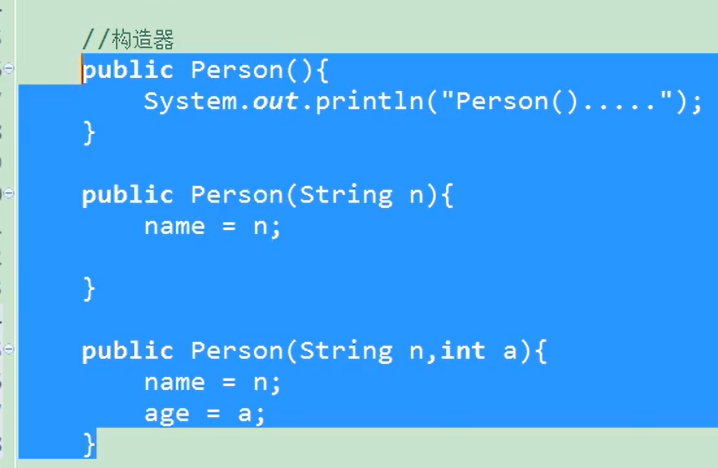
**同一个类中可以定义多个构造器,即可以传入不同类型的形参**

****

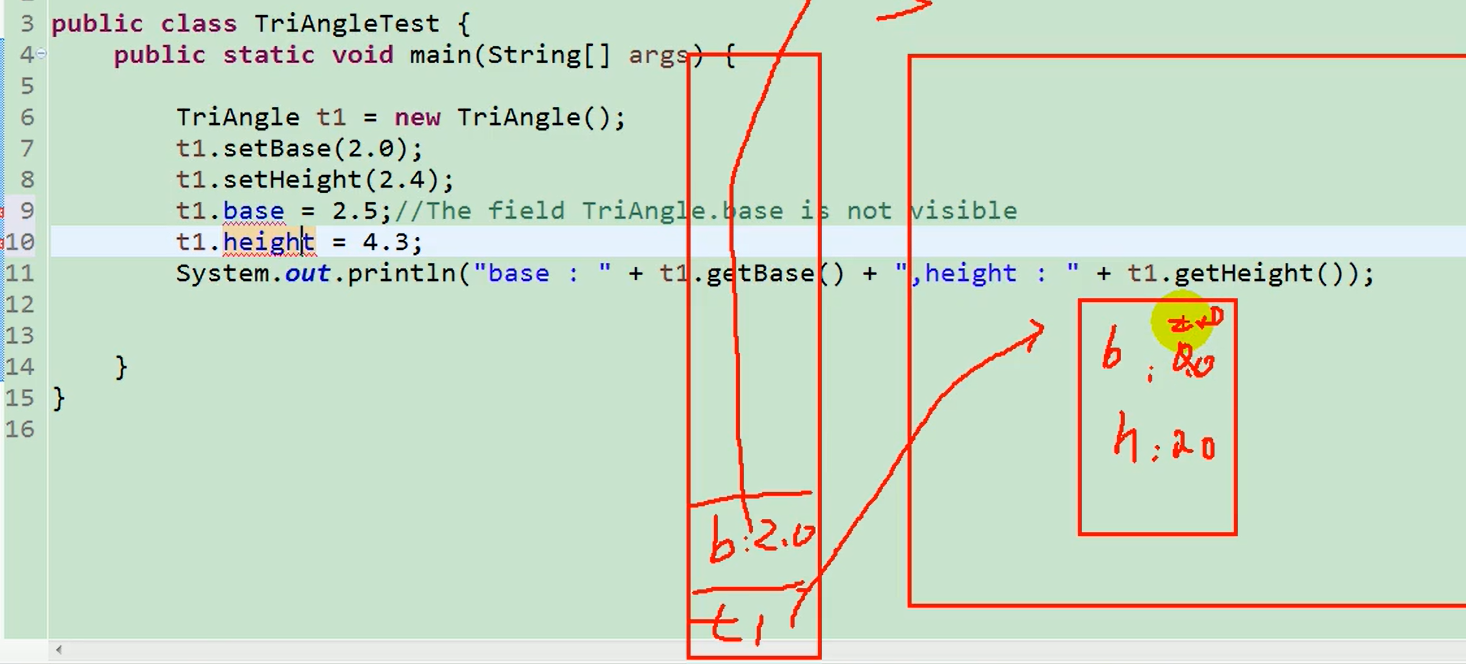
**此name即为创建的p1对象堆里面的name属性**

****

**多个构造器也可以重载.这样创建对象的时候就可以自由重载过的各种参数类型,甚至不写参数,比如API中的Sring类,就已经使用构造器重载很多类型了,所以可以输入数字,字母,字符串等**

****

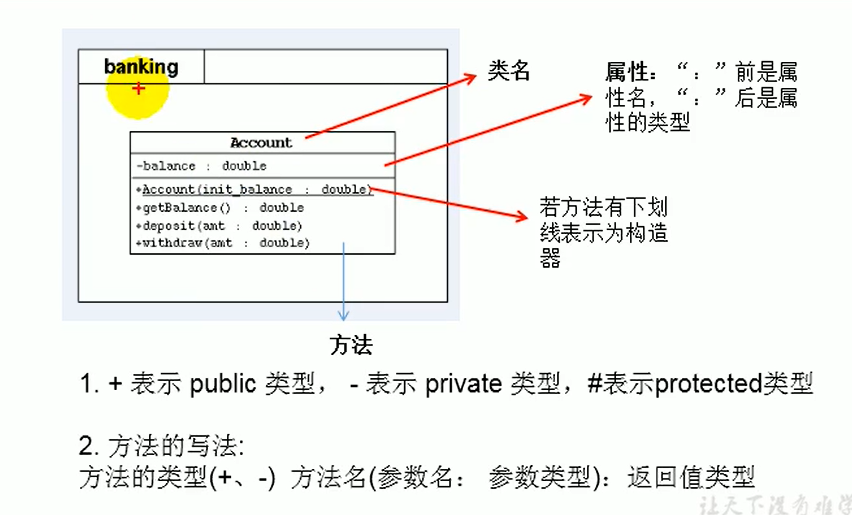
**构造器中也可以调同一个类中的方法**

****

**扩展之javabean**

****

**拓展之uml类图**

****

**Banking是包名，第一个double是返回值类型，下面的是形参名和形参类型**

**This(当前对象的意思)**

**This关键字常用于在类中构造方法时形参和属性重名时,在方法里面用this.name=name来区分属性和形参,即this.属性=形参.**

**如果一个类写了多个构造器即构造器重载,在第一个构造器中写了很多初始化对象的代码,而另外重载的构造器也需要使用这些代码时,就不用再复制一份.比如可以在②号构造器中的第一行直接写this();表示调用的是不含任何参数的①号构造器中的代码. 也可以this(age);age是int类型并且在当前构造器的形参列表中已经定义, 来表示调用形参列表也是int类型的构造器**

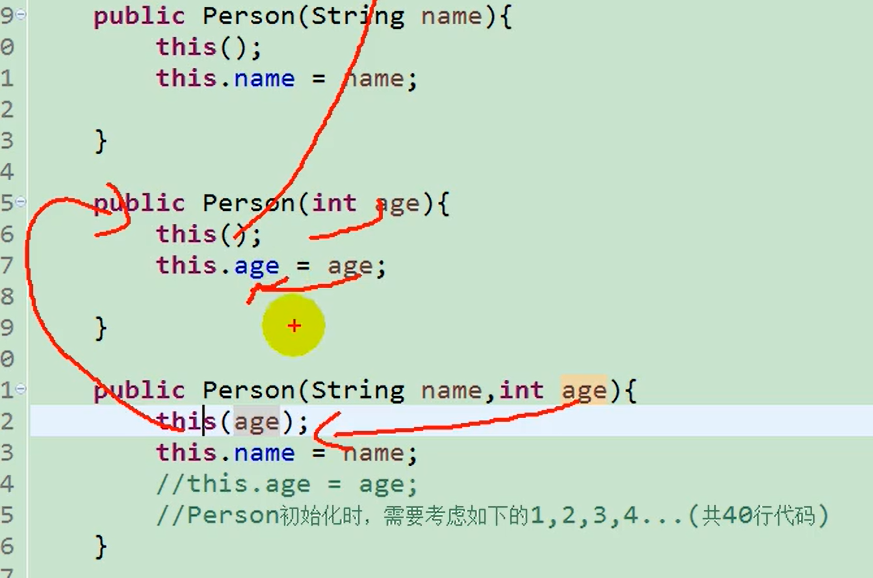
**当调用之后(①调②) 如this(age),相当于将①的age值(在main方法中定义对象时传入的)传入②中的int类型形参,然后调用②中的对此形参类型加工的代码工作,此时还是用的①的age值**

**This另外妙用，比如在A类中有一个方法形参是B类b对象，需要调用B类的一个方法，而B类的这个方法的形参也恰好用A类a对象，这时就要在A类的这个方法中写b.xx（this），**

**这个this就代表当前对象，也就是A类的a对象或者说调用此方法的a对象，因为a对象在main方法中应该会定义，（可能已经用构造器赋值属性），毕竟定义了a对象才能用a.某个方法实现上述的所有操作。**

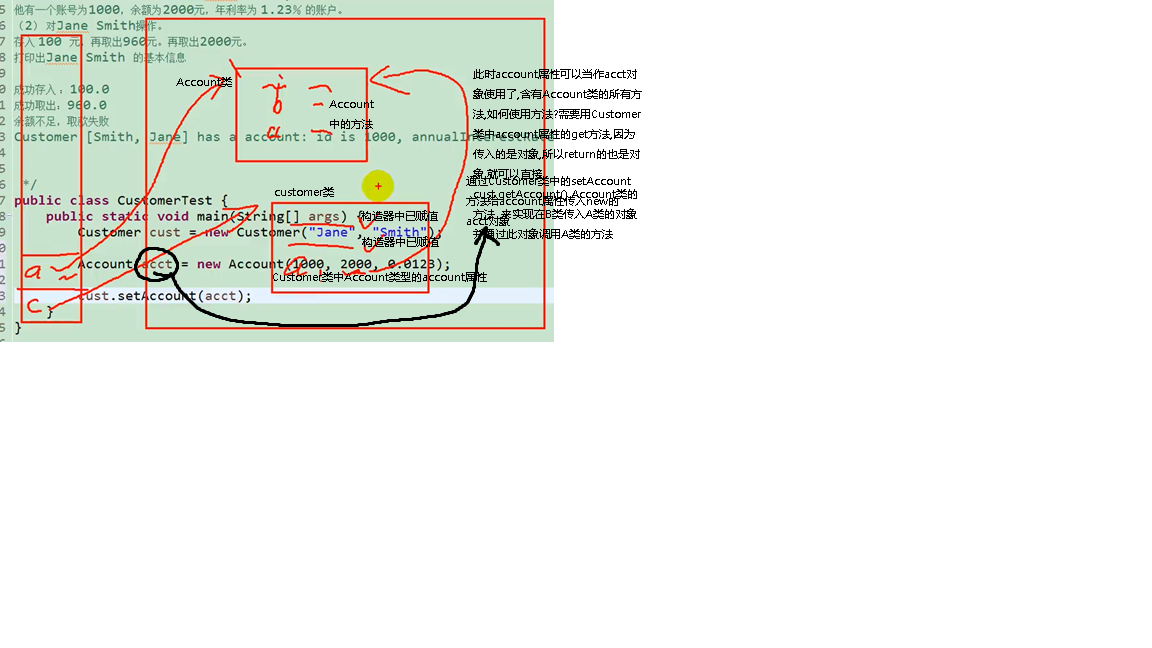
**需要注意: 两个构造器不能相互调用,类似于无限循环.也不能自己调自己,也不能①调②,②调③,③调①,这样又是无限循环,也不行,所以最后的这个构造器不能调任何构造器**

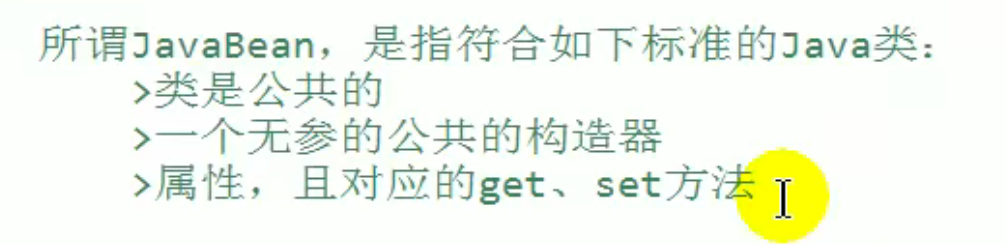
**用this来调用其他构造器时必须将this代码写在第一行,并且只能写一次**

****

**在B类中Private 属性时可以用A类+属性名定义一个A类的属性,类似于String xx;引用变量类型的属性.类就是引用类型.**

**比如A类中有3个方法, 在B类可以定义一个A引用类型的a属性,即A a; 然后对这个a属性设置get和set方法, 此后在mian中调用B类时 B b=new B(); 可以用b对象调用a属性的set方法来传入一个A类的对象存到B类的a属性中,此后可也以通过a属性的get方法来使用A类的3个方法.即通过B类的a属性调用A类的3个方法 a.get().A类的方法 连续点两次**



****

**复习**

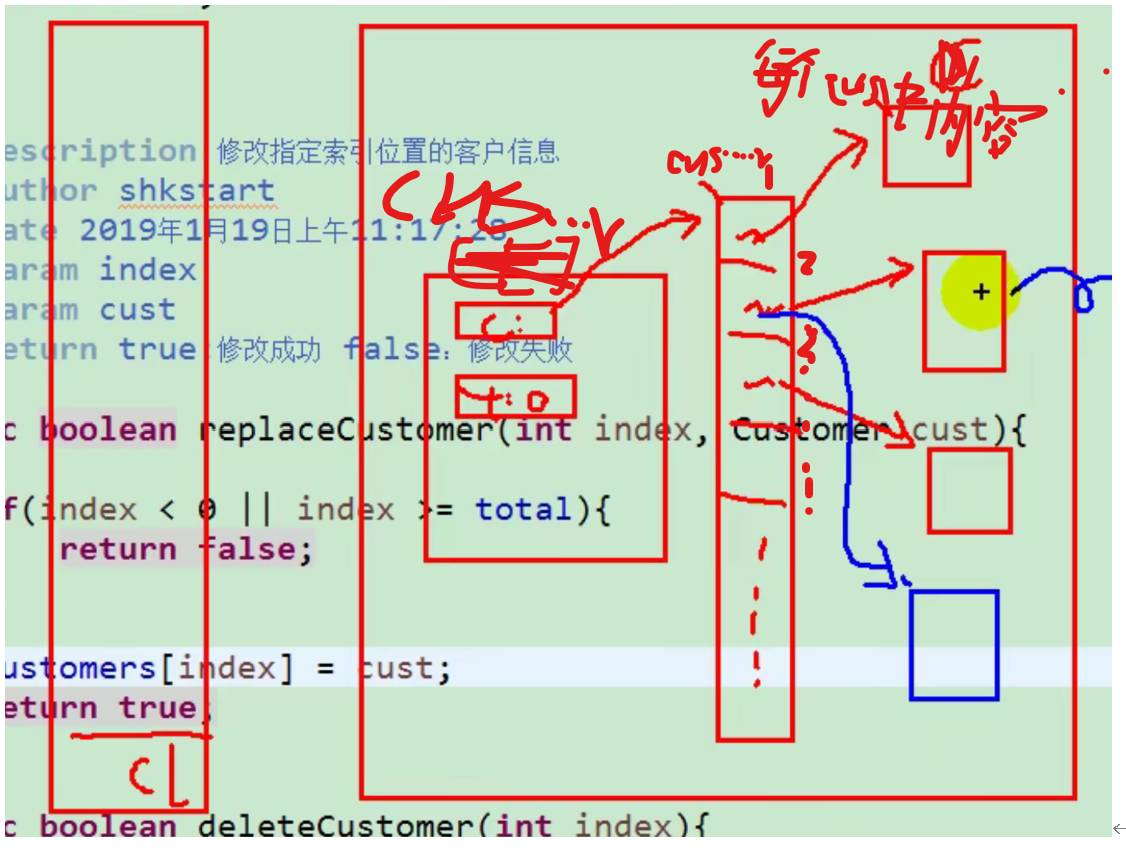
Java中a=b的意思是复制值（都是基本变量类型时）或复制地址（都是引用变量类型的时候，改变b可能会对a也改变）。

Customer【】customer，Customer【】表示customer是一个引用类型的数组存放在堆中，使customer只能存储地址，如果是简单的Array【】a，则a是在栈中的，前提是局部变量。

创建一个Customer类 类型的数组 Customer【】customer；

那么在同一个数组中 customer【0】=customer【1】，此时customer【0】所储存的customer对象地址被customer【1】覆盖掉，同时我们将customer【1】所存储的地址变为空，这样就保证了customer【0】的内容不跟customer【1】重复。

如果new了另外一个Customer【】cust，将cust【1】=customer【1】，改变cust中的内容会对customer的显示有影响，通常此操作仅用来当customer定义的太大如5而需要获取动态修改的内容后并返回时返回cust来显示内容不修改内容。



注 在图中customer是数组，数组是引用类型所以存储的是第i个customer对象的地址，而第i个customer也是引用类型（Customer customer），存储customer对象所指向的customer具体属性方法等内容块。

**复习**

基本数据类型是放在栈中还是放在堆中，这取决于基本类型在何处声明，下面对数据类型在内存中的存储问题来解释一下：

   一：**在方法中声明的变量，即该变量是局部变量**，每当程序调用方法时，系统都会为该方法建立一个方法栈，其所在方法中声明的变量就放在方法栈中，当方法结束系统会释放方法栈，其对应在该方法中声明的变量随着栈的销毁而结束，这就局部变量只能在方法中有效的原因

      在方法中声明的变量可以是基本类型的变量，也可以是引用类型的变量。

         （1）当声明是基本类型的变量的时，其变量名及值（变量名及值是两个概念）是放在JAVA虚拟机栈中

         （2）当声明的是引用变量时，所声明的变量（该变量实际上是在方法中存储的是内存地址值）是放在JAVA虚拟机的栈中，该变量所指向的对象是放在堆类存中的。

   二：**在类中声明的变量是成员变量，也叫全局变量，放在堆中**的（因为全局变量不会随着某个方法执行结束而销毁）。

       同样在类中声明的变量即可是基本类型的变量 也可是引用类型的变量

       （1）当声明的是基本类型的变量其变量名及其值放在堆内存中的

       （2）引用类型时，其声明的变量仍然会存储一个内存地址值，该内存地址值指向所引用的对象。引用变量名和对应的对象仍然存储在相应的堆中

此外，为了反驳观点" Java的基本数据类型都是存储在栈的 "，我们也可以随便举出一个反例，例如：

      int[] array=new int[]{1,2};

      由于new了一个对象，所以new int[]{1,2}这个对象时存储在堆中的，也就是说1,2这两个基本数据类型是存储在堆中，

      这也就很有效的反驳了基本数据类型一定是存储在栈中~~

在方法中定义了一个局部变量，当这个方法结束了，这个局部变量会立即被销毁吗？ 为什么？  
注意，我指的的这个局部变量本身，不是这个局部变量指向的对象

答案 栈里面的变量会被立即回收，但堆里面的对象需要GC回收。函数调用返回，对应该函数的栈帧消失，局部变量（基本类型或非基本类型的引用）储存在栈帧中，所以局部变量占用的内存被回收，但是堆中的对象（局部变量引用的）需要等待GC

不过也无妨，因为即使堆中不能及时被CG垃圾回收也没有变量储存他的的地址，干放在那里罢了

项目客户信息管理P259内存解析