**P62 面向对象（概述）**

**//**

**P63 面向对象（举例）**

**//**

**P64 面向对象（类与对象的关系）**

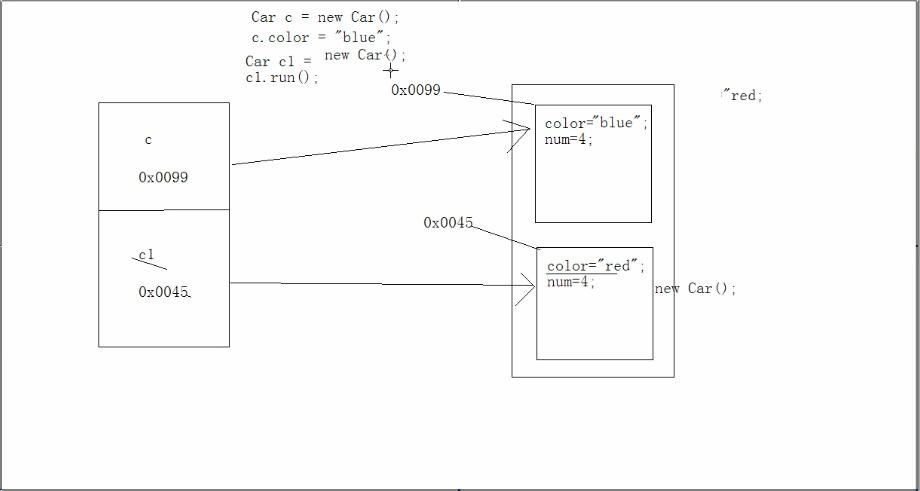
--类与对象的关系：

类就是对现实生活中事物的描述；

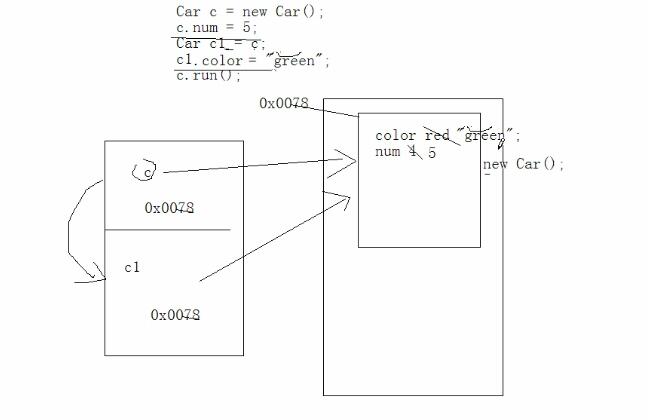
对象就是这类事物，实实在在存在的个体。

--根据class Car创建一个对象：

Car c = new Car(); //c称为引用型变量，是一个类类型变量，类类型为Car。用关键字new在堆内存中生成一个对象。可以看到，类中的实例变量，随着对象的产生而产生，并且和对象一起储存在堆内存中。



--多个引用指向同一个对象：



**P65 面向对象（成员变量和局部变量）**

--成员变量和局部变量：

作用范围：成员变量作用与整个类中，局部变量作用在函数中或语句中。

在内存中的位置：成员变量在堆内存中，因为对象的存在，才在内存中存在；局部变量存在栈内存中

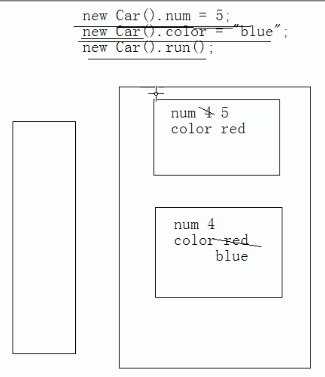
**P66 面向对象（匿名对象的应用）**

--不匿名的写法：Car c = new Car();

c.num = 5;

匿名的写法：new Car().name = 5;

--



第一句在堆内存中开辟了一个空间，第二句在堆内存中又开辟了一个新的空间，而且这两句一执行完，对应的对象就变成了垃圾。所以前两句话没意义。但是第三局话调用了方法，可以有一个运行内容在里面，所以说匿名对象，调用方法有意义，调用属性没意义。

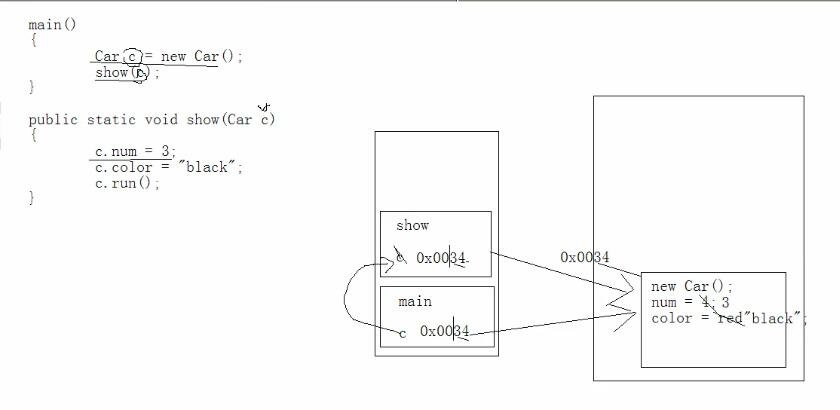
--匿名对象使用方式一：当对对象的方法只调用一次时，可以用匿名对象来完成，这样写比较简化。但是如果堆一个对象进行多个成员调用，必须给这个对象起一个名字；

匿名对象使用方式二：可以将匿名对象作为实际参数进行传递。例如：

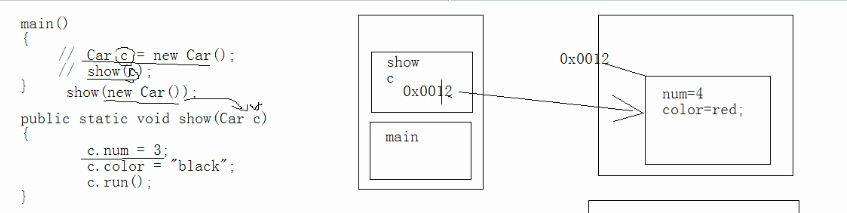
不匿名的写法：Car q = new Car();

show(q);

匿名的写法：show(new Car());



在调用show()方法时，方法以及里面的局部变量都是存放在栈内存中。还可以注意到：当show()方法执行完后，上图的堆内对象由于还有main中的c指向，所以还存在，但是下图的对象就随着show方法的结束变成了垃圾。



**P67 面向对象（封装概述）**

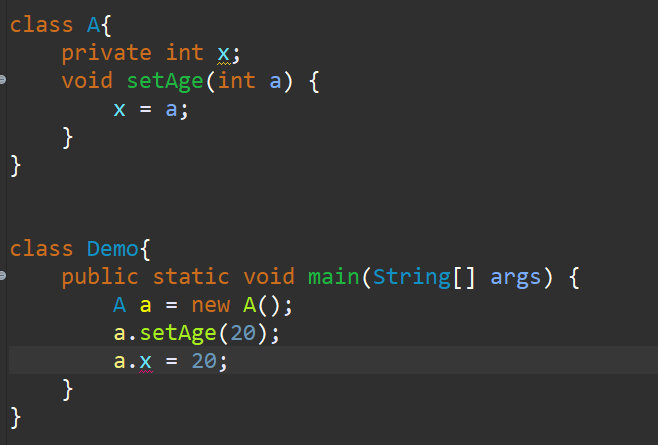
//

**P68 面向对象（封装private）**

--private：私有，权限修饰符：用于修饰类中的成员（成员变量，成员函数）。私有只在本类中有效。将变量私有化后，类以外即使建立了对象也不能直接访问。

--私有仅仅是封装的一种表现形式。只要对方的权限在你之外，对你而言都是封装的。

--



如何看待a.setAge()和a.x的区别？

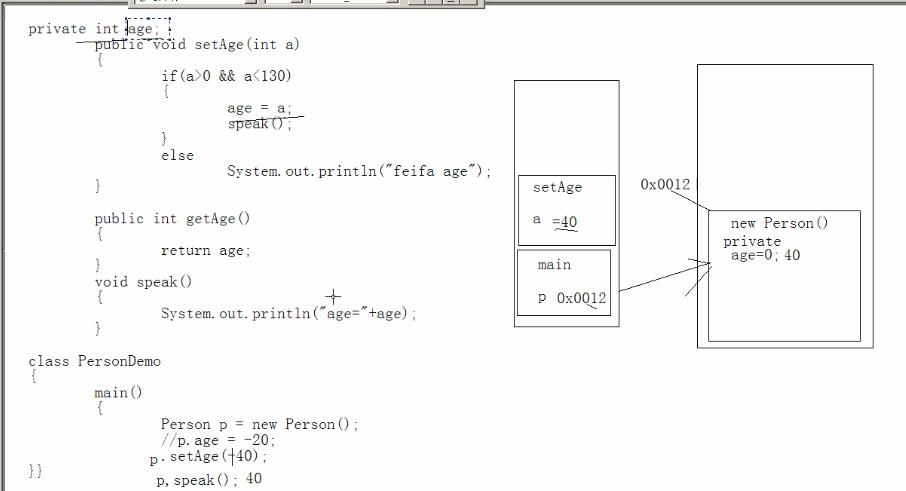
a.x因为A中对变量x进行了私有，所以无法直接进行访问。这很好理解。

而a.setAge(20)的流程是a.setAge()先进入栈内存，然后将20赋给形参int a，最后对x进行了修改。

--调用非静态函数时,（肯定就会用到成员变量），那么该函数在内存空间是如何运行的？

该函数在类建立时开始储存在方法区，当执行该方法语句时，进入栈。然后局部变量也在栈中，成员变量就在对象所在的堆内存中。接下来碰到赋值给局部变量的，该局部变量也在栈中；碰到赋值给成员变量的，直接对堆内存里的对象的对应变量进行修改；碰到调用其他方法的，一如该方法一开始被执行到的情况。

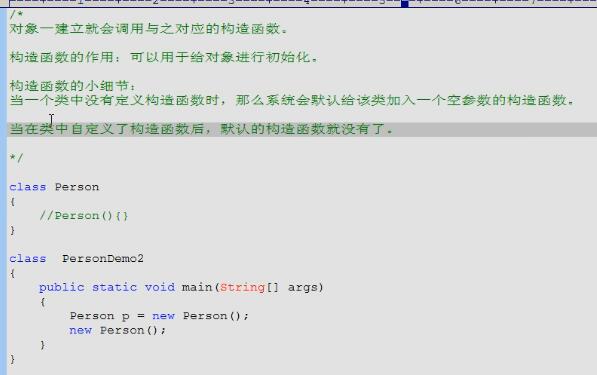
--

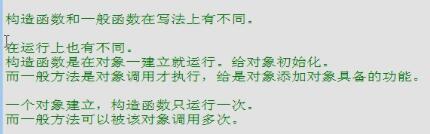


成员变量有默认初始化值，因为它在堆内存中；而局部变量没有初始化值，因为它在栈内存中。图中a是局部变量，并没有指向对象。

**P69 面向对象（构造函数）**

--



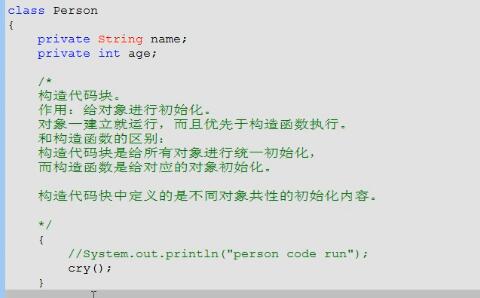


那么什么时候定义构造函数呢？

当分析事物时，该事物一存在（或者说一初始化）就具备一些特性或者行为，那么将这些内容定义在构造函数中。

构造函数在进行定义的时候需不需要未知内容参与运算？需要就定义参数，然后初始化的时候往里面传就可以了；不需要就不用。

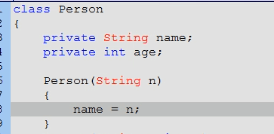
**P70 面向对象（构造代码块）**



由此可见，它主要的用处就是将不同对应对象的构造函数中共同的部份提取出来放在构造代码块中，以避免代码的重复，提高效率。

**P71 面向对象（this关键字）**

--

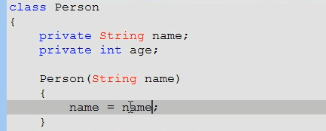


此时，我输入

Person p = new Person(“lisi”);

输出的name结果就是lisi。

而略微修改



此时，我输入

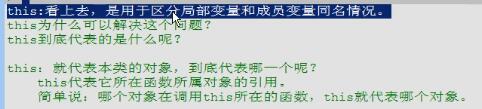
Person p = new Person(“lisi”);

输出的name结果就是null。

因为图一的构造函数中，name是成员变量，n是局部变量；图二的构造函数中，两个name都是是局部变量。

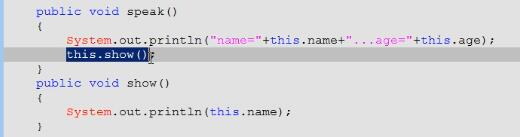
为什么会这样呢？因为在调用函数的时候，碰到变量，它会先检查是否为局部变量，如果没找到，就会去找成员变量。再没有就报错了hh

解决方法：将左边的name改为this.name



--

像下图中的this都可以省略，加上去只是为了更好的阅读性。

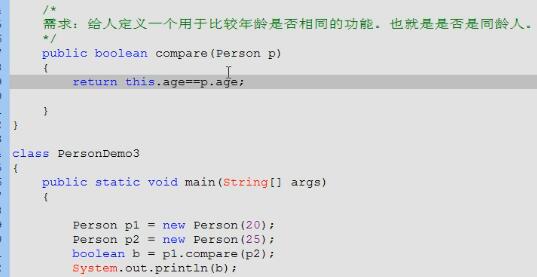


**P72 面向对象（this关键字的应用，this用法一）**

--

因为在定义方法（同义词：功能、函数）时，还没有对象，这时用this来表示，那么后期谁调用我了，我代表谁。

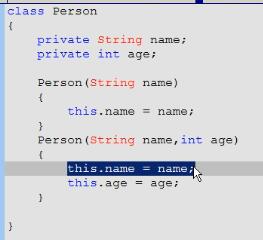
如下图的实例



p1就代表this，而p2就传给了p

**P73 面向对象（this关键字在构造函数间调用，this用法二）**

--实例如下



有一个Person类的构造函数是如上定义的，发现蓝色部分跟前面有重复

改为 this.name = name; 🡪 Person(name); 是错误的，虽然一般函数就是这么调用的

应该为 🡪 this(name);

当执行 Person p = new Person(“lisi”, 30);

程序是这样执行的：先将”lisi”,30传入第二个构造函数，然后读到this(name);，就将“lisi”传入第一个构造函数，第一个构造函数就将“lisi”成功的传入到对象所在的堆内存。最后读this.age=age;， 就将30传入对象所在的堆内存。整个完成。

进一步理解this(name);

在执行 Person p = new Person(“lisi”, 30);时，对象就是p，所以就相当于p(name);， 就相当于new Person(name);, 此时传入“lisi”，就按第一个构造函数完成了name的初始化。目的是提高代码的复用。

--



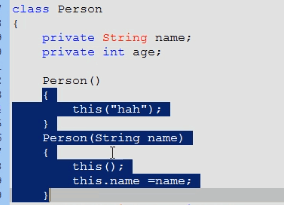
为什么this语句只能放在第一行？



不放第一行，可以看到最后的name是“haha”，而不是“lisi”。

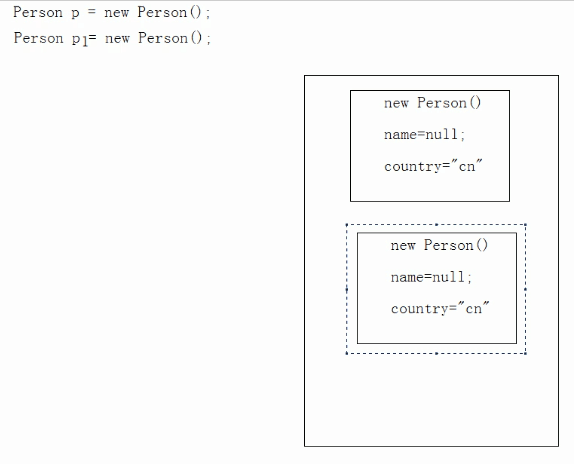
所以说，初始化动作中还有其他的初始化动作（也就是构造函数中还调用到了其他的构造函数），其他的初始化要先执行，然后再执行自己的初始化。

--死循环的反例



**P74 面向对象（Static关键字）**

--可以看到，例如成员变量country，往往都是“cn”，把它提出来作为静态变量，能够节省内存空间。



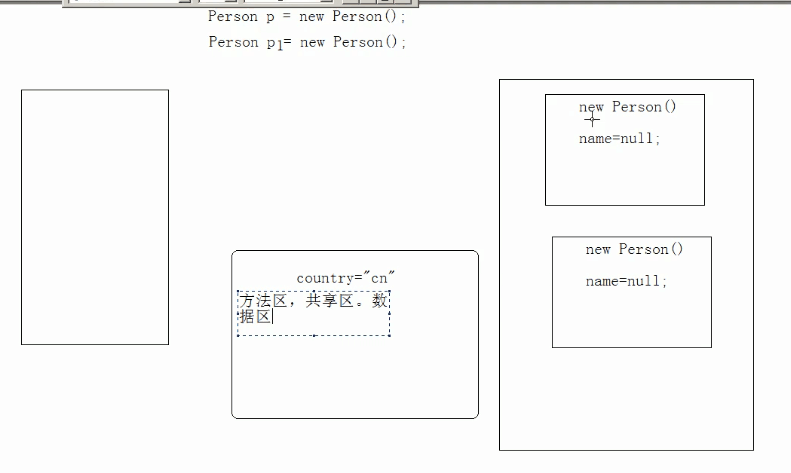
--静态修饰内容被对象所共享。

--



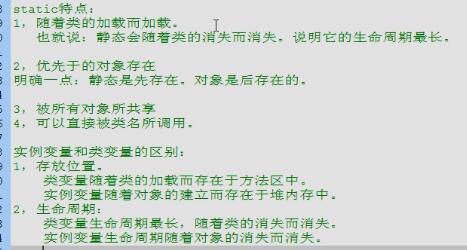
--在定义事物的时候，哪些是共性的，哪些是特有的，要学会区分，特有的内容随着对象存储。

--



静态变量，函数（不一定要静态）存储在方法区中。

--



注意特点1，其实很多时候，类及其对象都是一起的。

String name; //成员变量，实例变量

Static String country = “CN”; //静态的成员变量，类变量。

类为什么不都定义成静态的？ 1.不符合实际；2.因为生命周期长，所以占内存。

--静态使用注意事项：

1.静态方法只能访问静态成员，非静态变量，非静态方法都不行，因为它们都随对象的，在代码中其实隐含着this.，而静态先于对象存在。反例如下：



* 1. 非静态方法既可以访问静态，也可以访问非静态

2.静态方法中不可以定义this，super关键字，因为静态优先于对象存在。

3.主函数是静态的

可以发现，函数，成员变量其实都隐含着this.；如果在该静态函数下没有新建一个对象，那么这些this就没有指向，就会报错，所以静态方法中不能有非静态成员。

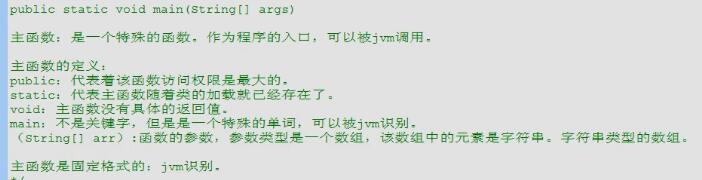
--



关于访问出现局限性整个弊端的解决方法：其实main函数就是这么做的，就是在静态方法中，new一个对象就ok了。

**P75 面向对象（main函数）**

--



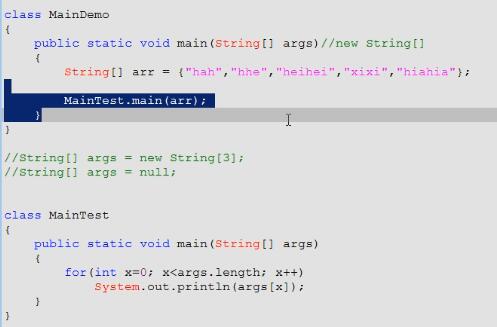
像 public static void main(int x); 虽然属于重载，不报错，但是虚拟机不调用这个函数。

主函数被jvm直接调用：类名.main(new String[0]); //注意是0个元素

当然也可以像下图一样传入[“haha”,”hehe”,”heihei”]这一数组。



--

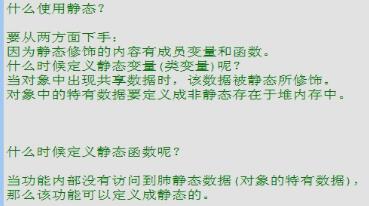


图中代码也能正常运行，但是没啥实际意义。因为只要一个输入入口就够了

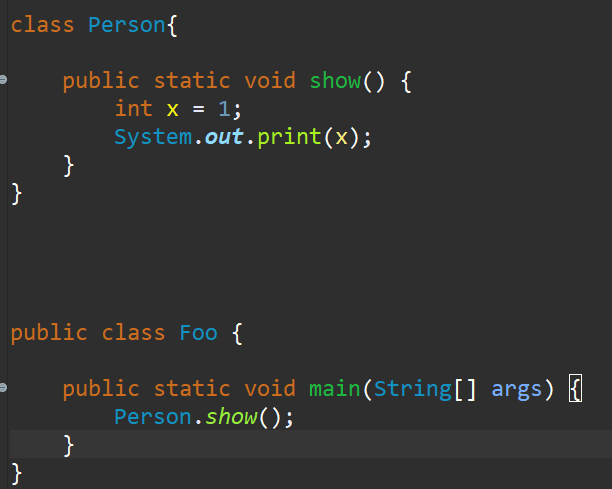
**P76 面向对象（静态什么时候使用）**

--共享数据不等于共享属性。比如一个人叫张三，一个人叫李四，张三和李四属于共享属性—name，但这两个并不是共享数据，属于对象特有数据。

建立对象的目的：就是为了封装数据，将特有数据分开进行存放。反过来，特别是如果调用一个方法，这个方法没有用到特有数据，那么就没有创建对象的必要。



注意，功能内部不能有非静态数据，但是能有局部变量（因为局部变量就不是非静态数据），例如下面的arr数组



**P77 面向对象（静态的应用-工具类）**

//

**P78 面向对象（帮助文档的制作javadoc）**

--当你使用别人现成的class工具时，如果没有和你的class文件保存在同一个目录下，如何解决？实例如下：



我自己写了一个ArrayToolDemo.java，保存至D:\java0217\day06下，用到了别人的ArrayTool.class方法，但是这个class文件保存在C:\myclass下。所以，我们先要设置classpath，set classpath=.;c:\myclass 。.表示当前目录，也就是D:\java0217\day06，然后用；隔开。

如果这样设置set classpath=c:\myclass，编译能通过，但运行时会找不到ArrayToolDemo.class而报错。

**P79 面向对象（静态代码块）**

随着类的加载而加载，只执行一次，后面再反复new对象也不执行了。

顺序：静态代码块 🡪 构造代码块 🡪 构造函数

**P80 面向对象（对象的初始化过程）**

第一步，因为new用到了Person.class，所以将Person.class文件通过java虚拟机，从硬盘中加载进内存。注意，如果是Person p = null，则不会加载。

第二步，静态代码块执行

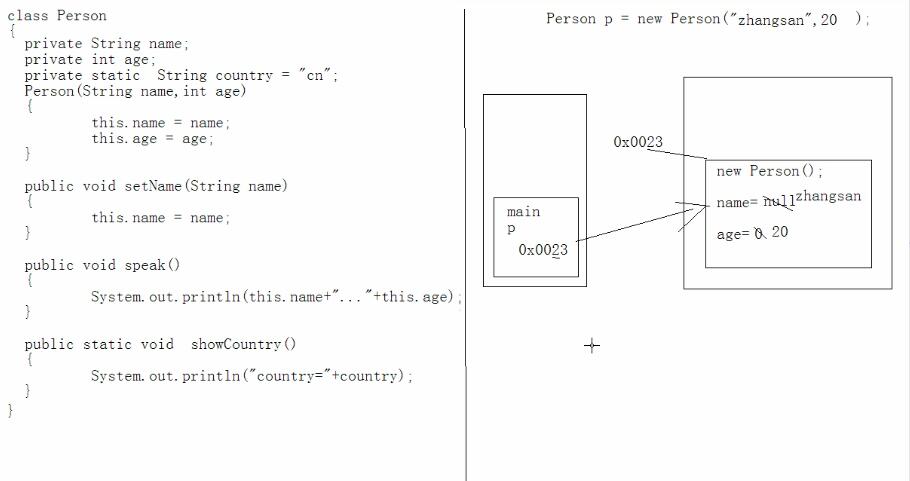
第三步，开辟堆内存空间，分配内存地址，在堆内存中建立对象的特有属性，并进行默认初始化

第四步，对属性进行显式初始化

第五步，对对象进行构造代码块初始化

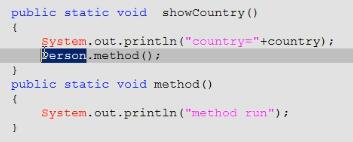
第六步，对对象进行对应的构造函数初始化

第七步，将内存地址赋给栈内存中的p变量

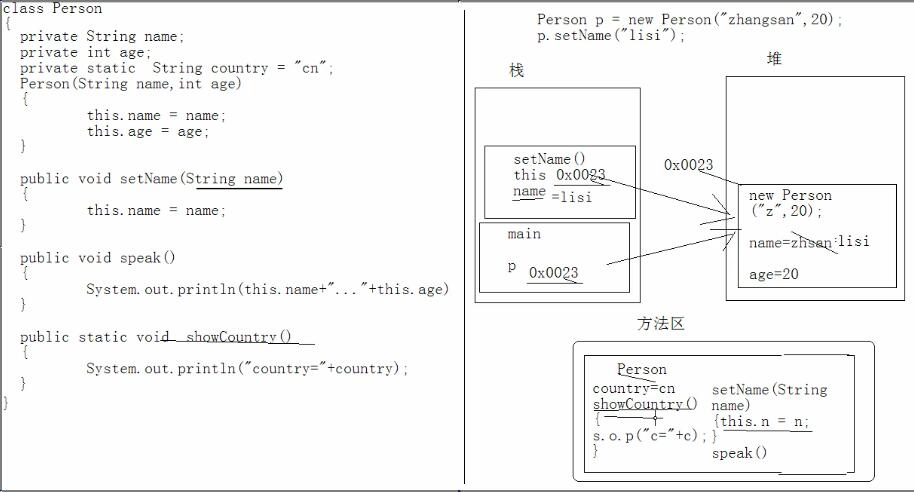


**P81 面向对象（对象调用成员过程）**

--静态变量，静态方法及其方法体，非静态方法及其方法体，其实都随着类的加载，而优先于对象存储在内存中的方法区。（特别注意非静态方法也是如此）。静态的在方法区中的静态区，调用它们时省略的是类名点，而非静态在方法区中的非静态区，调用它们时省略的是对象点。



调用非静态方法时，省略的是对象.，而调用静态方法时，省略的是类名.。这一原理同样适用于静态变量和非静态变量。

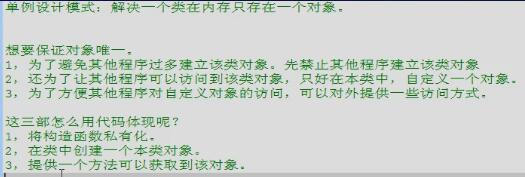


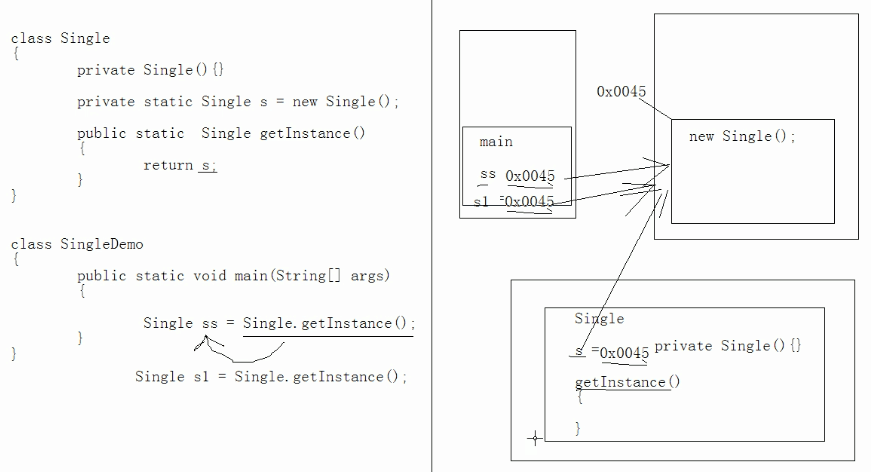
当通过main执行Person p = new Person(“zhangsan”,20);时，和P80一样。

在执行到p.setName(“lisi”);时，因为涉及到了局部变量name，所以又在栈内存中新开辟了一个空间调用的是setName()方法而不是p.setName()，包括了两个局部变量this和name，this由p对象调用，所以等于p对象的地址值。传入的“lisi”传递给name，最后“lisi”传递给了堆内存中的name。

而类似的，静态方法也相似，不过只涉及到栈内存和方法区。

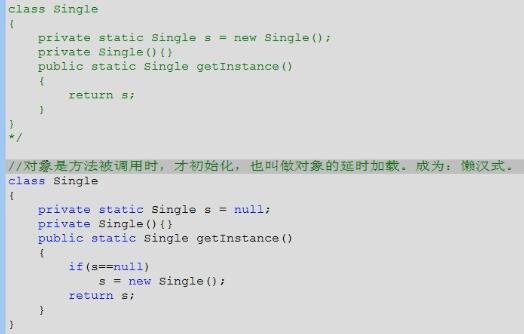
**P82 面向对象（单例设计模式）**





**P83 面向对象（单例设计模式方法二）**

两种单例设计模式方法



实际应用中，往往使用饿汉式，因为懒汉式存在一个问题：比如a，b同时访问，由于cpu的处理机制，有可能a在进行if(s==null);后，s=new Single();之前，cpu暂时中断a的任务，转而执行其他任务，比如b，这时a只是通过判断，但是还没创建对象，所以b也能通过条件判断，进入s=new Single();语句。这样，最后就会创建出两个不同的变量。

解决方法public static Single getInstance() 🡪 public static synchronized Single getInstance()，这样一来，在a进入条件判断以后，直到a出来为止，其他变量都不允许进入条件判断，在外面等着。但是整个程序的效率会变得十分低效。

进一步的解决方法，跟前一种解决方法相比，主要的改进是：锁的判断延后了，之前只要调用该方法，就要进行锁的判断；而现在只有同时满足调用该方法以及s为空，才进行锁的判断。

