小程序：

class Singleton  
{  
 private static Singleton *singleton* = new Singleton(); //第一种：放在这儿运行结果是counter1=1,counter2=0  
 public int counter1;  
 public int counter2=0;  
 //private static Singleton singleton = new Singleton(); //第二种：放在这儿运行结果是counter1=1,counter2=1  
  
 private Singleton()  
 {  
 counter1++;  
 counter2++;  
 }  
 public static Singleton getInstance()  
 {  
 return *singleton*;  
 }  
}  
  
public class MyTest  
{  
 public static void main(String[] args)  
 {  
 Singleton singleton = Singleton.*getInstance*();  
 System.*out*.println("counter1="+singleton.counter1);  
 System.*out*.println("counter2="+singleton.counter2);  
 }  
}

**运行方式**

大家可以跟着我实践一下，顺便告诉大家最简单的方式，只要你装了jdk便可以得到答案。

首先进入文件所在目录，然后输入命令javac MyTest.java，这个命令的作用是将java编译成可执行的class文件，也可以称为二进制文件。然后再输入命令java MyTest，就能得到结果。

**分析结果**

然后我们思考一下，为什么实例化对象这句代码放的位置不对，结果就不一样呢。

下面跟着我一起来从java程序对类的使用方式分析一下，我尽可能地扩展使读者能够对jvm类的加载、连接与使用这块多了解一些：

**知识点：**

Java程序对类的使用方式可分为两种：

主动使用和被动使用

所有的Java虚拟机实现必须在每个类或接口被Java程序"首次主动使用"时才初始化他们。

主动使用有六种：

1、创建类的实例

例：new Test();

2、访问某个类或接口的静态变量，或者对该静态变量赋值

例：int b=Test.a;

Test.a=b;

3、调用类的静态方法

例：Test.doSomething();

4、反射

例：Class.forName("come.tgb.Test");

5、初始化一个类的子类

例：class Parent{}

class child extends Parent

{public static int a =3;}

Child.a=4;

6、Java虚拟机启动时被标明为启动类的类

例：Test.java可能被编译成多个类文件，Test.class, Parent.class, Child.class，最终运行java Test,那么Test.class就是启动类。

除了以上6种情况，其他使用Java类的方式都被看看作是对类的被动使用，都不会导致类的初始化。

**干货来了！！！**

第一种输出结果为counter1=1, counter2= 0

**[java]** [view plain](http://blog.csdn.net/qwlzxx/article/details/71172491) [copy](http://blog.csdn.net/qwlzxx/article/details/71172491)

1. Singleton singleton = Singleton.getInstance();

这一句使用了主动使用的第三种，调用类的静态方法。然后就会加载Singleton类。

**[java]** [view plain](http://blog.csdn.net/qwlzxx/article/details/71172491) [copy](http://blog.csdn.net/qwlzxx/article/details/71172491)

1. private static Singleton singleton = new Singleton();

首先执行这一句，new Singleton会调用构造方法。

counter1和counter2都是int型，默认初始值为0，重要！所以执行完构造方法后都变为1。

然后才执行。

**[java]** [view plain](http://blog.csdn.net/qwlzxx/article/details/71172491) [copy](http://blog.csdn.net/qwlzxx/article/details/71172491)

1. public static int counter1;
2. public static int counter2=0;

第一行没给赋值，所以还是1；第二行赋值为0，所以counter2又由1变为0。

最后输出结果为counter1=1, counter2= 0.

第二种输出结果为counter1=1, counter2=1

加载类同第一种，首先执行

**[java]** [view plain](http://blog.csdn.net/qwlzxx/article/details/71172491) [copy](http://blog.csdn.net/qwlzxx/article/details/71172491)

1. public static int counter1;
2. public static int counter2=0;

counter1的值是int型默认值0，第二个counter2的值被赋为0.

然后再执行

**[java]** [view plain](http://blog.csdn.net/qwlzxx/article/details/71172491) [copy](http://blog.csdn.net/qwlzxx/article/details/71172491)

1. private static Singleton singleton = new Singleton();

与第一种类似，调用构造方法。

counter1和counter2的值都加1变为1.

最后输出结果为counter1=1, counter2=1

**小结**

都这里一切都真相大白了！JVM东西千千万，这里展示的只是其中一部分，更多关于JVM的干货请期待后面的博文！

# 问题总结：关键点

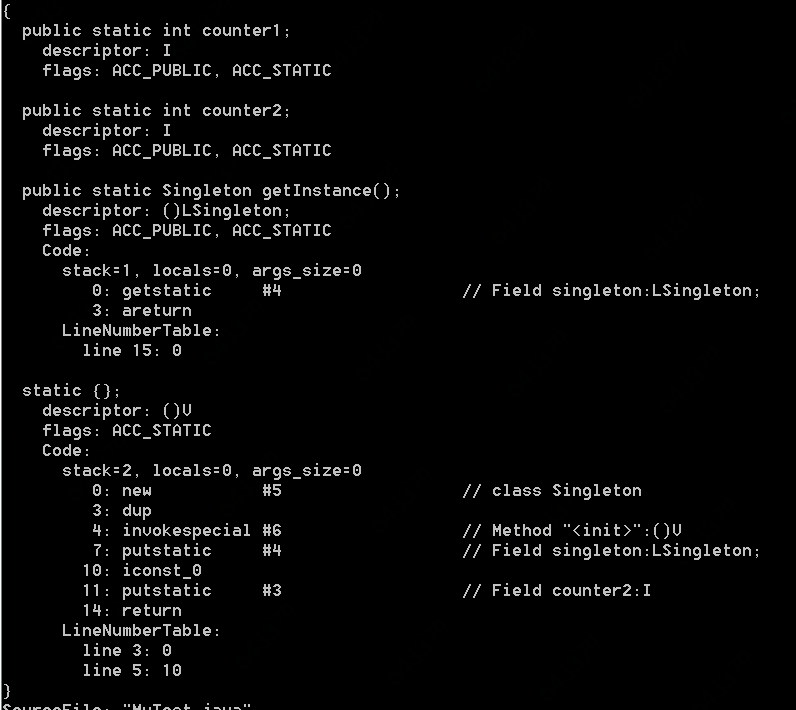
对于类的初始化：类的初始化一般只初始化一次，并且对静态变量进行初始化。

类的编译结果决定了类的初始化过程。编译器生成的class文件主要对定义在源文件中的类进行了如下的更改：

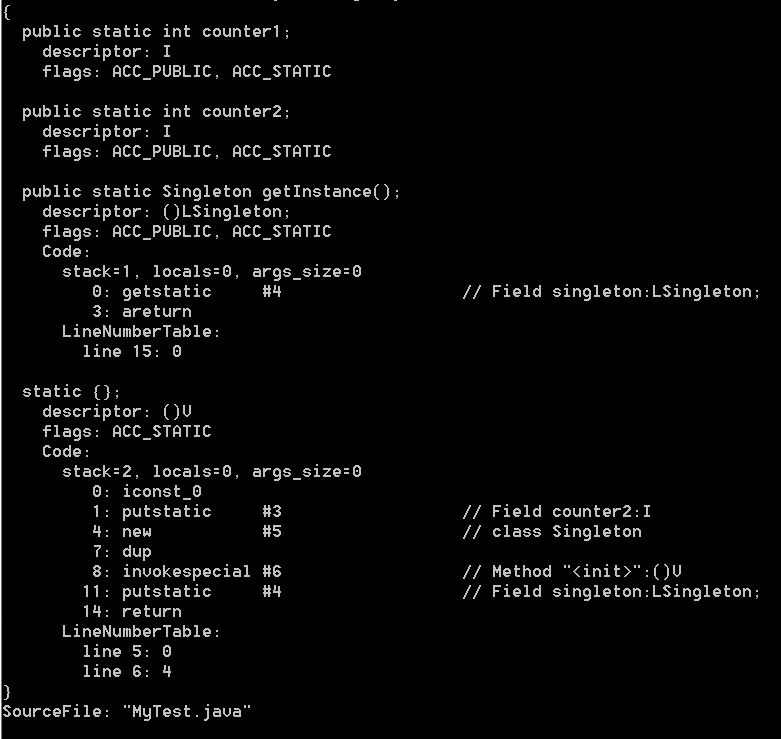
1)先按照静态成员变量的定义顺序在类内部声明成员变量。基本类型带有默认值

2)再按照原java类中对成员变量的初始化顺序进行初始化。

Counter赋值在后，会将之前的值覆盖，所以输出结果为0,1



Counter的赋值在前，所以构造方法就是直接累加了，输出结果为1,1



首先是静态变量的声明；

然后是方法的声明；

最后静态代码块对变量进行复制

代码块赋值的顺序和源码的声明顺序一致。

# 代码的执行顺序

1. 主调类的静态代码块
2. 对象父类的静态代码块
3. 对象的静态代码块
4. 对象父类的非静态代码块
5. 对象父类的构造函数
6. 对象的非静态代码块
7. 对象的构造函数

**无父哪来子，静完才非静**

# 常量的初始化顺序如何呢？