1. java.lang.Math类和java.lang.StrictMath类是否是单例模式？

否，在单例模式中，某个只能有一个实例，它必须自行创建这个实例并向整个系统提供一个全局访问点。对于java.lang.Math类和java.lang.StrictMath类来说，用户不需要关心这种类有几个、是怎么创建、通过什么方式访问，直接使用类名.方法名()调用需要的方法接口即可。因此，可以说这两个类不必保证实例的唯一性，或者向整个系统提供某个“唯一实例”的访问方法。另外，它们不需要维护任何状态，仅仅是提供全局访问的方法接口。这两个类是静态类，即它们所有的方法都是静态方法，在编译期绑定，不可更改，这样和数学方法的固定和不可变是一致的，且在使用中具有更好的性能。

1. 单例模式有三种实现方式，分别是饿汉式，懒汉式，以及静态内部类实现。
2. 对于懒汉式实现方式，请列举可以保证线程安全的两种方法；

一是锁方法，即对单例类的getInstance()方法使用synchronized关键字加锁，二是双重检查锁定，即在程序经过第一重if (instance==null)判断进入锁定代码块后，在创建单例实例之前必须进行第二重if (instance==null)判断。

1. 请比较三种实现方式的差异；

饿汉式单例类：缺点是在类加载时就进行初始化，资源利用效率较差，且系统加载时间可能会比较长；优点是没有加锁，无须考虑多个线程同时访问的问题，因此调用速度和反应时间更好，执行效率更高。

懒汉式单例类：优点是实现了延迟加载，减少内存浪费；缺点是必须通过双重检查锁定等机制处理多线程同时访问问题，系统性能受到一定影响。

静态内部类实现：内部类只有在外部类被调用才加载，综合了前两者的优点并回避了缺点，既可以做到延迟加载，也不必担心多线程同步问题。

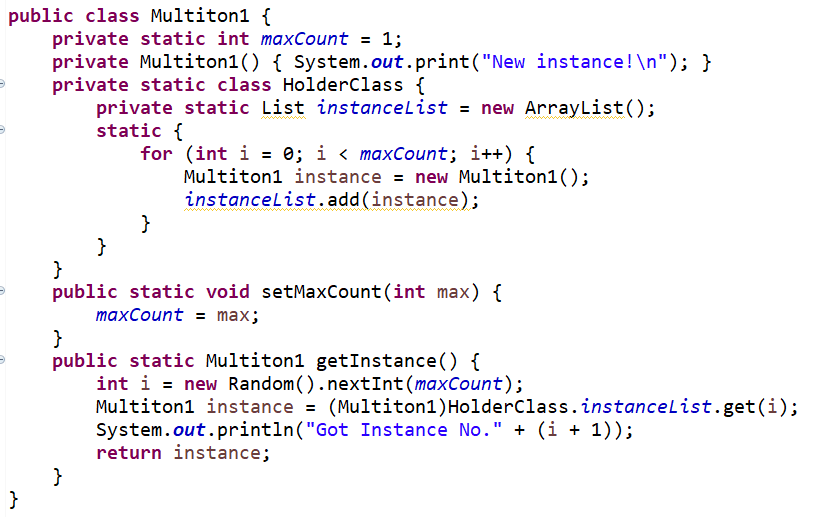
1. 请分析上述三种实现方式下，单例实例进入内存空间的时间点。

饿汉式：单例实例在类加载时就进入内存空间。

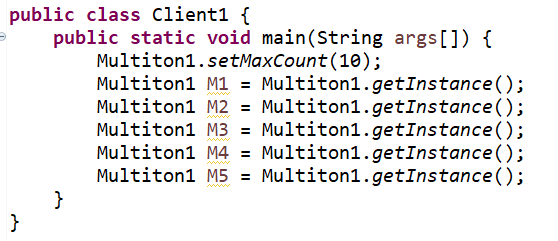
懒汉式：在程序第一次调用类的getInstance()并创建单例实例时进入。

静态内部类：当程序第一次调用类的getInstance()，HolderClass被加载，静态对象instance真正被创建时进入。

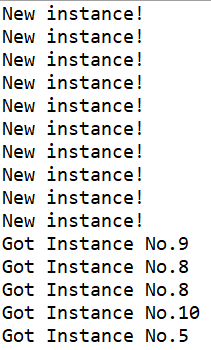
第一种实现：



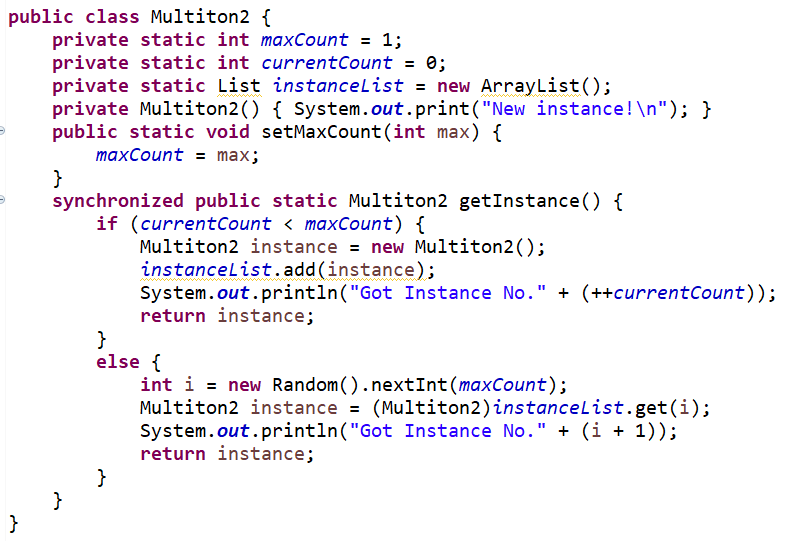
思路：多例类default实例个数为1，如果用户自行指定，则必须在第一次调用getInstance()之前直接对Multiton类调用setMaxCount(max)指定，否则一旦调用过getInstance()就不能再次更改。具体实现模仿静态内部类方法，在首次调用getInstance()时，通过内部类HolderClass一次性创建maxCount个实例储存在列表中，每次程序调用getInstance()，便随机返回其中一个。



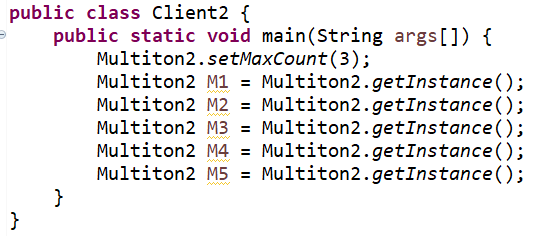
测试：让默认构造函数Multiton()每次被调用时输出一句New Instance!，获取实例的函数getInstance()每次输出Got Instance + 获取第几个实例的编号，注意到这里一次性输出了10次New Instance!，然后是随机获取的五个实例。



第二种实现：



思路：实例个数指定方法同上。具体实现模仿懒汉式单例类与锁方法，每次调用getInstance()时，检查当前已经创建的实例个数currentCount是否小于最大允许创建的实例个数maxCount，若是，创建新的实例存入instanceList并返回，同时更新currentCount；若不是，随机返回List中的一个实例。



测试：原理同上，前三次调用getInstance()每次输出一句New Instance!然后返回刚刚创建的实例（No.1, No.2, No.3），后两次没有New Instance!而且返回的是之前三次创建的实例，可见两种实现方法的效果都符合设想。

