一个class有且仅有一个instance/object

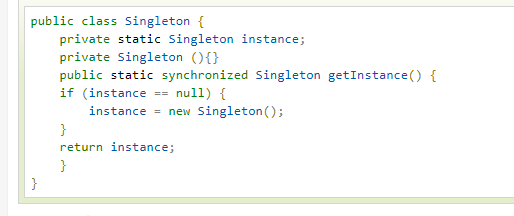
同时这个class必须给所有其他对象提供这一实例

**意图：**保证一个类仅有一个实例，并提供一个访问它的全局访问点。

特点，constructor是私有的，这样别人就不能重复构造。有一个初始参数，且是private static的，保证共有

实现1：

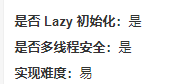
懒汉式



在getInstance的时候如果不存在，会new一个

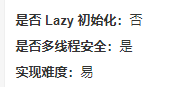
为了保证线程安全整个synchronized，

因为如果不synchronized，在一开始还没有new Singleton()的时候，两个线程同时访问，会有两个instance



缺陷在于我们就一开始new instance的时候需要synchronized保证单例，但是后来再synchronized影响效率

实现2：饿汉式，用的多

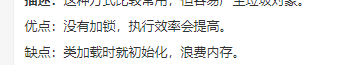




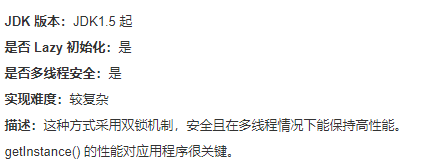
两种都行，一个是在静态代码块new,一个是一开始就new,他用classLoader机制避免了多线程的同步问题，

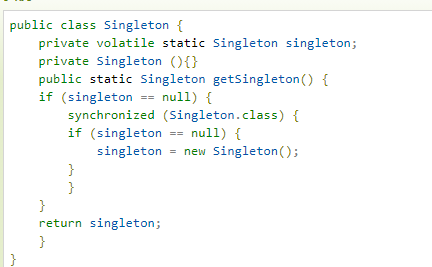
没有synchronized效率高

但有时候我们并不需要初始化Instance，这里会默认初始化，多了一个object占用内存



3.DCL double check locking双重锁





和懒汉式不同在于，,给singleton加了一个volatile关键字

synchronized写在代码块内

**修饰代码块**

在某些情况下，我们编写的方法体可能比较大，同时存在一些比较耗时的操作，而需要同步的代码又只有一小部分，如果直接对整个方法进行同步操作，可能会得不偿失，此时我们可以使用同步代码块的方法对需要同步的代码进行包裹，这样就无需对整个方法进行同步操作了。

synchronized(xx.class)  
可以让这一个代码块synchronized

synchronized(this)都行

面试官：synchronized里你传Singleton.class这个参数，起到什么作用，换成别的行不行？

面试者：对当前类加锁，使得这个代码块一次只能被一个线程访问。这里Singleton.class可以换成一个常量字符串或者自己定义一个内部静态Object。

volatile关键字

1.保证线程间变量的可见性，简单地说就是当线程A对变量X进行了修改后，在线程A后面执行的其他线程能看到变量X的变动，更详细地说是要符合以下两个规则：

线程对变量进行修改之后，要立刻回写到主内存。

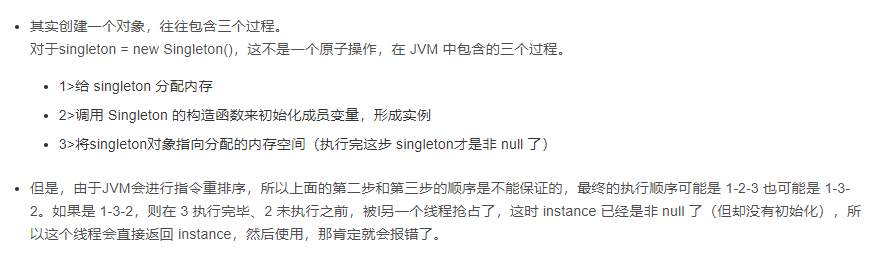
线程对变量读取的时候，要从主内存中读，而不是缓存。

换成人话来说，对于线程，他原来的变量读取是在缓存里的，而变量加了volatile关键字以后，这个变量就不会写在缓存里，而是写在主内存里，如果线程A对他进行改动，那么线程B就会立刻知道

面试者：volatile这个英文单词意思是易变的，用在多线程中来同步变量。Java的对象都是在内存堆中分配空间。但是Java有主内存和线程自己独有的内存拷贝。对于没有volatile修饰的局部变量，线程在运行过程中访问的是工作内存中的变量值，其修改对于主内存不是立即可见。而volatile修饰的值在线程独有的工作内存中无副本，线程直接和主内存交互，修改对主内存立即可见。

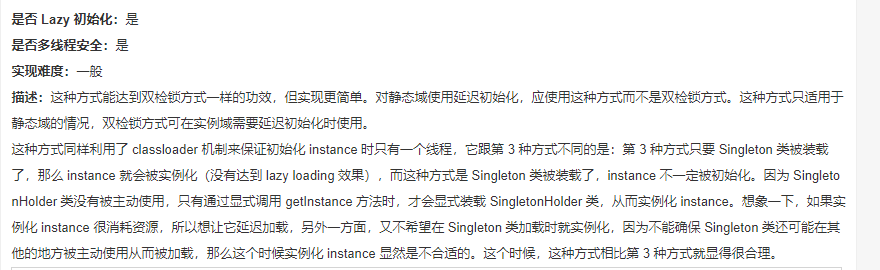
第一次检查可能会同时漏入多个第一次运行的进程，这时synchronized保证只有一个进，初始化，然后第二个进程进入，如果没有volatile，第二个进程B有可能还是副本内的singleton,而有了volatile，他就意识到singleton已经被刷新了。

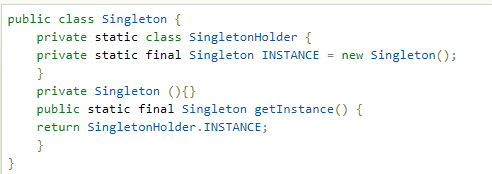
2.保证禁止指令重排序



在以后的getInstance第一次check就行了

4.静态内部类，登记式





singleton被类装载的时候不一定初始化，因为SingletonHolder没有被主动使用，只有使用getInstance的时候，会装载SingleHolder

不强制用final

