java中共享变量的内存可见性问题

代码示例

```
package threadlocal;

import java.util.Random;

public class M2 {

    static class T1 implements Runnable {
        public static int sum = 0;

        @Override
        public void run() {
            addSum();
        }

        private static void addSum(){
            for (int i = 0; i < 100; i++) {
                try {
                    Thread.sleep(new Random().nextInt(100));
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                sum++;
            }
        }
    }

    public static void main(String[] args) throws InterruptedException {
        for (int i = 0; i < 5; i++) {
            new Thread(new T1()).start();
        }

        while (true) {
            Thread.sleep(1000);
            System.out.println("sum is : " + T1.sum);
        }
    }
}
```

代码输出结果小于500，每次都不太一样

因为多个线程同时操作的共享变量sum，而由于不可见性导致的最终结果小于 预期结果500

Java中的synchronized关键字

介绍

synchronized块是Java提供的一种原子性内置锁， Java中的每个对象都可以把它当作 一个同步锁来使用， 这些 Java 内置的使用者看不到

线程的执行代码在进入 synchronized 代码块前会自动获取内部锁，这时候其他线程访问该同步代码块时会被阻塞挂起 。拿到内部锁的线程
后 或者在同步块内调用了该内置锁资源的wait系列方法时释放该内置锁。 内置锁是排它锁， 也就是当一个线程获取这个锁后， 其他线程必
锁，这是一个很耗时的操作，会引起线程上下文的切换

代码示例

```
package threadlocal;

import java.util.Random;

public class M2 {

    static class T1 implements Runnable {
        public static int sum = 0;

        @Override
        public void run() {
            addSum();
        }

        private static synchronized void addSum(){
            for (int i = 0; i < 100; i++) {
                try {
                    Thread.sleep(new Random().nextInt(100));
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                sum++;
            }
        }
    }

    public static void main(String[] args) throws InterruptedException {
        for (int i = 0; i < 5; i++) {
            new Thread(new T1()).start();
        }

        while (true) {
            Thread.sleep(1000);
            System.out.println("sum is : " + T1.sum);
        }
    }
}
```

注意这里的 addSum方法是静态的，也必须是静态的，如果不是静态的 synchronized 不会生效。

由于调用的地方 是多个 T1 对象， 而synchronized锁住的是括号里的对象，而不是代码。对于非static的synchronized方法，锁的就是对象