我们看一个例子：

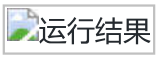
```
class Demo {
    public synchronized void test() {
        System.out.println("test方法开始执行，当前线程为："+Thread.currentThread().getName());
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        System.out.println("test方法执行完毕，当前线程为："+Thread.currentThread().getName());
    }
}


class MyThread implements Runnable {

    @Override
    public void run() {
        Demo demo = new Demo();
        demo.test();
    }

}
public class Test {
    public static void main(String[] args) {
        MyThread myThread = new MyThread();
        new Thread(myThread,"子线程A").start();
        new Thread(myThread,"子线程B").start();
        new Thread(myThread,"子线程C").start();
    }
}
```

运行结果：



从运行结果我们可以看出，Demo类提供的test同步方法好像并没有起作用，这是怎么一回事。

实际上，synchronized(this) 以及非 static 的 synchronized 方法，只能防止多个线程同时执行同一个对象的同步代码块。即 synchronized 锁住的是括号里的对象，而不是代码块

所以说 synchronized 是一个对象锁。

当 synchronized 锁住一个对象后，别的线程如果也想拿到这个对象的锁，就必须等待这个线程执行完成释放锁，才能再次给对象加锁，这样才能达到线程同步的目的。所以即使两个不同的代码块都要锁住同一个对象，那么这两个代码段也不能在多线程环境下同时运行，必须等其中一个现将对象锁释放掉，另一个才能给对象上锁。

所以在上例中，MyThread线程类启动三次也创建了三个Demo类，并且对其调用，三个不同的对象进入了同步方法中，所以显示如上结果。

当一个线程A 进入到同步方法所在的类中，其他线程不能进入该类中的其他类中，因为锁住的是对象。类比：厕所里有个电视机，某人上厕所时关上了锁，其他人也不能进来看电视。

那我们如果想将一段代码锁住，使同时有且只有一个对象能访问该代码块应该如何操作。

这种锁住代码块的的操作叫做全局锁，可以通过以下两种途径实现：

1.1 锁住同一个对象

```
class Demo {
    public void test() {
        // 锁住进入的方法的对象
        synchronized(this) {
            System.out.println("test方法开始执行，当前线程为："+Thread.currentThread().getName());
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            System.out.println("test方法执行完毕，当前线程为："+Thread.currentThread().getName());
        }
    }
}


class MyThread implements Runnable {
    // 为了防止多个线程创建多个对象，所以在类中自己创建一个对象
    private Demo demo;
    // 在构造方MyThread时将真正的对象传入
    public MyThread(Demo demo) {
        this.demo = demo;
    }

    @Override
    public void run() {
        this.demo.test();
    }

}
public class Test {
    public static void main(String[] args) {
        // 实际上，整个程序只有这一个对象
        // 锁住了该对象就相当于将 Demo类中的test方法代码锁住了，曲线救国实现全局锁
        Demo demo = new Demo();
        MyThread myThread = new MyThread(demo);
        new Thread(myThread,"子线程A").start();
        new Thread(myThread,"子线程B").start();
        new Thread(myThread,"子线程C").start();
    }
}
```

1.2 锁住整个类

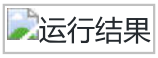
```
class Demo {
    public void test() {
        // 将 Demo类 作为锁定的对象，每次只能有一个对象进入该类
        synchronized(Demo.class) {
            System.out.println("test方法开始执行，当前线程为："+Thread.currentThread().getName());
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            System.out.println("test方法执行完毕，当前线程为："+Thread.currentThread().getName());
        }
    }
}


class MyThread implements Runnable {

    @Override
    public void run() {
        // 虽然这里还是存在创建多个对象的问题
        // 但是由于test方法这次锁住了整个类，所以同时有且仅有一个对象能够进入Demo类中
        Demo demo = new Demo();
        demo.test();
    }

}
public class Test {
    public static void main(String[] args) {
        MyThread myThread = new MyThread();
        new Thread(myThread,"子线程A").start();
        new Thread(myThread,"子线程B").start();
        new Thread(myThread,"子线程C").start();
    }
}
```

运行结果：



当然，使用静态同步方法也可以实现锁住整个类的效果。

```
public static synchronized test() {
    // statement
}
```