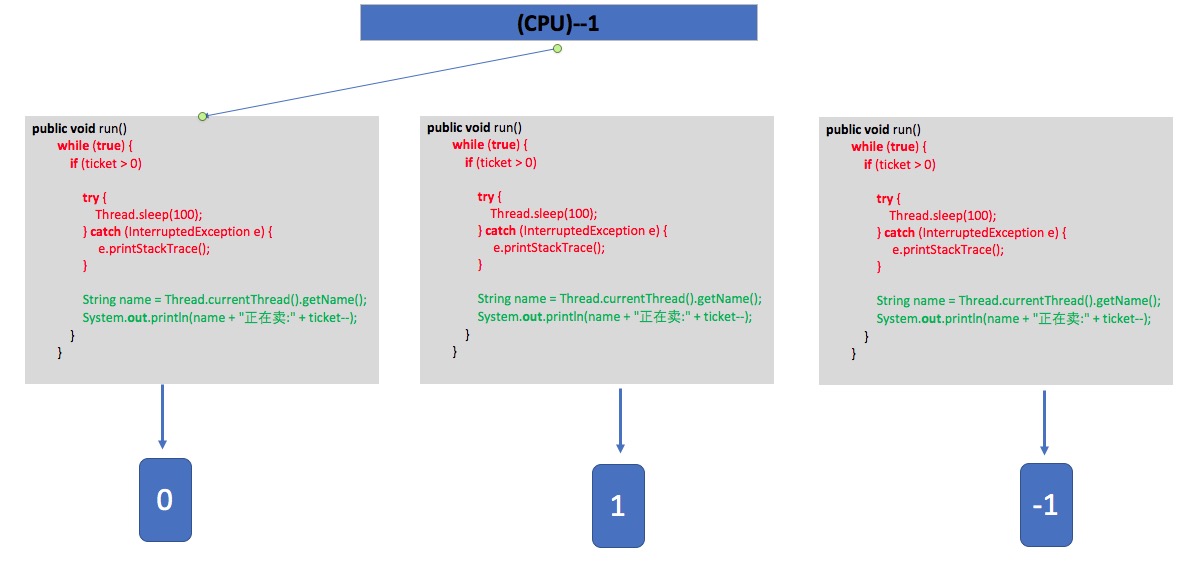
**线程安全问题文档**

什么是线程安全：

有多个线程再同时运行，而这些线程同时运行一段相同代码。程序每次运行结果和单线程运行结果不一样时，就是线程不安全，运行结果是一样的，则代表线程是安全的

出现线程安全问题的条件：

1. 资源共享
2. 多线程
3. 并发访问



上图执行流程：

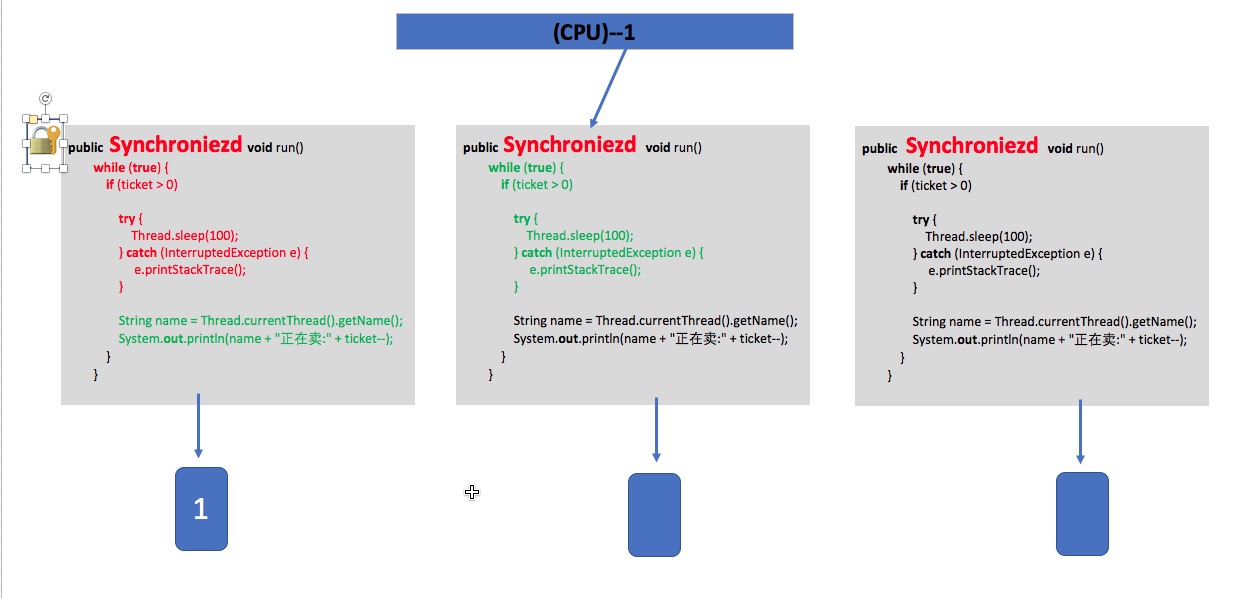
假如当第一个窗口抢到资源，从上往下执行，满足ticket>0，执行到Thread.sleep(100)这，线程进入休眠状态，然后开始重新强资源，当第二个窗口抢到资源，一样从上往下执行，满足ticket>0，执行到Thread.sleep(100)这，线程进入休眠状态，然后又开始重新强资源，当第三个窗口强到资源，从上往下执行，满足ticket>0，执行到Thread.sleep(100)这，线程进入休眠状态，正好第二窗口休眠状态结束，执行休眠下面的代码，这时候卖出第一百张票，然后总票减去1，票已经卖完了，第一个窗口休眠结束，然后执行下面的代码，但是因为是从休眠代码那开始执行，并没有经过if条件判断是否大于0，同理第三个窗口也是一样，所以就会出现卖出第0张和第-1张的情况，这就是线程不安全。

解决方法：

第一种、同步方法锁

同步方法就是使用synchronized关键字修饰某个方法，这个方法就是同步方法。这个同步方法(非static方法)无须显式指定同步监视器，同步方法的同步监视器是this，也就是调用该方法的对象。通过同步方法可以非常方便的实现线程安全的类

如下图 所示

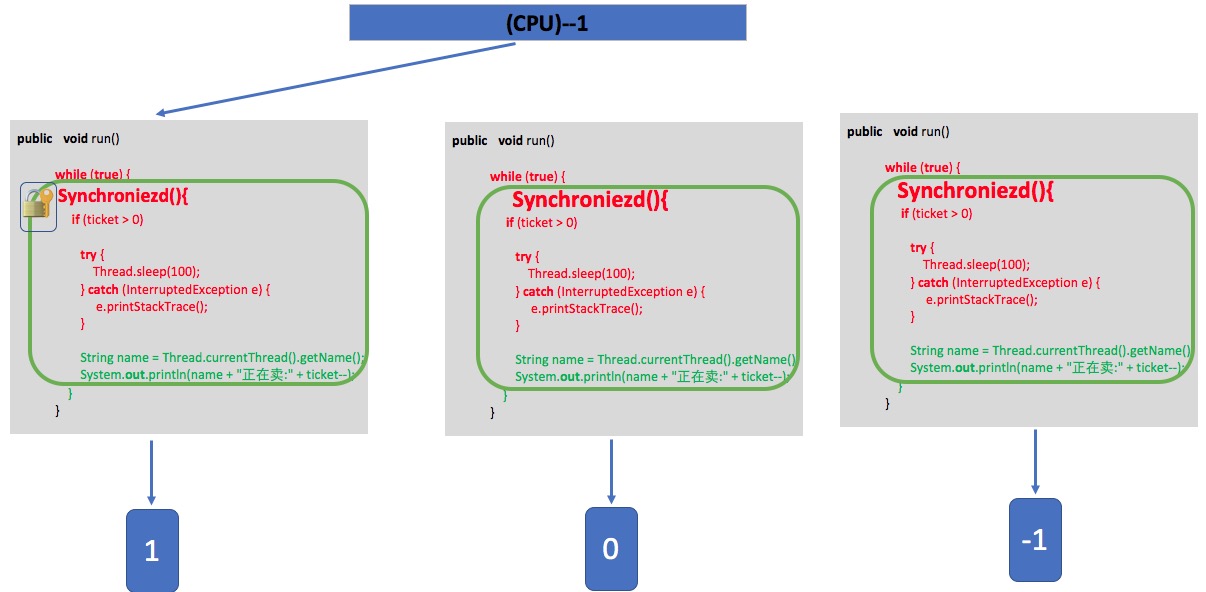


当第一个窗口抢到资源时，执行这个方法时，就会给这个方法上一把锁，如果不执行完这个方法，其他窗口就不能争抢资源，所以当休眠的时候，其他窗口也要等待，因为是while死循环，所以所有的票全部都会由第一口卖完，也就避免了出现线程安全问题，因为相当于单线程在执行，所以缺点很明显，其他资源等于没有事做。

第二种、同步代码块锁

线程开始执行同步代码块之前，必须先获得对同步监视器的锁定。任何时刻只能有一个线程可以获得对同步监视器的锁定，当同步代码块执行完成后，该线程会释放对该同步监视器的锁定。

如下图 所示



同步块锁就避免了上一个解决方法的弊端，能三个窗口卖票，如上图所示，当抢到资源的窗口运行到Synchroniezd下的代码时，就会给这个代码块上锁，所以休眠的时候，其他窗口也不能抢资源，只能等这一段代码块执行完之后，才能重新抢资源。这个方法的缺点也很大，就是你同一时间执行的还是只有一段代码，也就是一个窗口，其他窗口无法卖票，也就是耗时比单线程还慢，因为三个窗口还要切换。

第三种方法：分段锁

因为没有试过，所以只有个大概理论，分段锁，就是给一个范围性上锁，也就相当于给票上锁，加入当给1到10号上锁，第一个窗口就卖1-10号票，其他窗口不能卖1-10，但是可以卖20-30号票，同理三窗口也一样，这样就解决了上诉两个方法的缺点，可以三个同时执行。