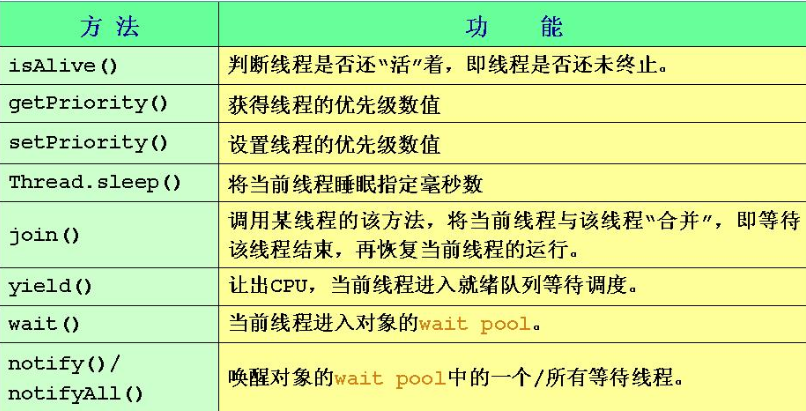
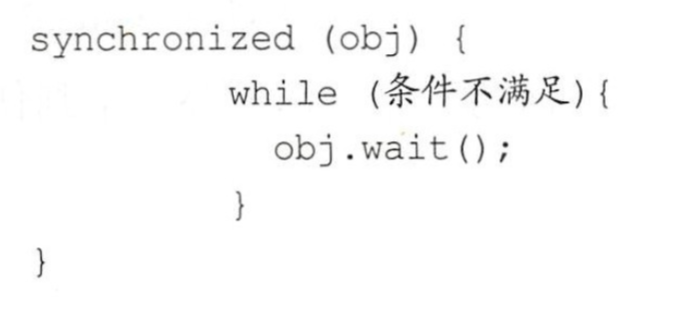
1. 线程控制基本方法：



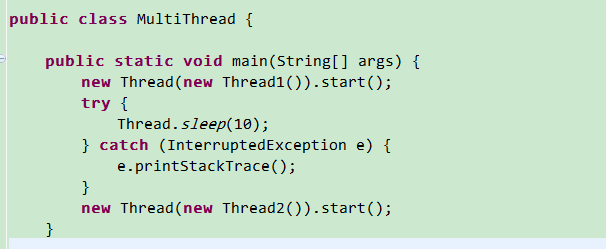
1. 等待wait与通知notify/notifyAll
   1. wait()方法
      1. wait是Object类里的方法，当一个线程执行到wait()方法时，它就进入到一个和该对象相关的等待池中，同时失去(释放)了对象的机锁(暂时失去机锁，wait(long timeout)超时时间到后还需要返还对象锁)；其他线程可以访问
      2. wait()必须放在synchronized block中，否则会在program runtime时扔出”java.lang.IllegalMonitorStateException“异常。
      3. 当一个线程调用一个共享变量的wait()方法时，该调用线程会被阻塞挂起，直到发生下面几件事情之一才返回
         1. 其他线程调用了该共享对象的 notify()或者notifyAll()方法
         2. 其他线程调用了该线程的interrupt()方法，该线程抛出InterruptedException异常返回
      4. 一个线程可以从挂起状态变为可以运行状态(也就是被唤醒)， 即使该线程没有被其他线程调用 notify()、 notifyAll()方法进行通知，或者被中断，或者等待超时，这就是所谓的**虚假唤醒**。经典的防范虚假唤醒的代码如下(while循环)

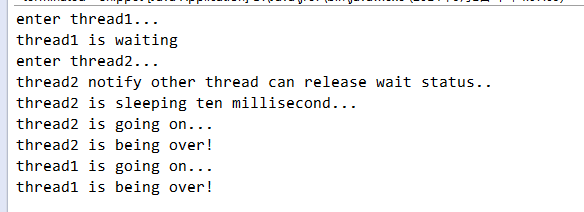


* 1. notify/notifyAll方法
     1. 一个线程调用共享对象的notify()方法后，会唤醒一个在该共享变量上调用wait系列方法后被挂起的线程。一个共享变量上可能会有多个线程在等待，具体唤醒哪个等待的线程是随机的。
     2. 被唤醒的线程不能马上从 wait 方法返回并继续执行，它必须在获取了共享对象的监视器锁后才可以返回。也就是唤醒它的线程释放了共享变量上的监视器锁后，被唤醒的线程也不一定会获取到共享对象的监视器锁，这是因为该线程还需要和其他线程一起竞争该锁， 只有该线程竞争到了共享变量的监视器锁后才可以继续执行
     3. notify()并不会释放锁

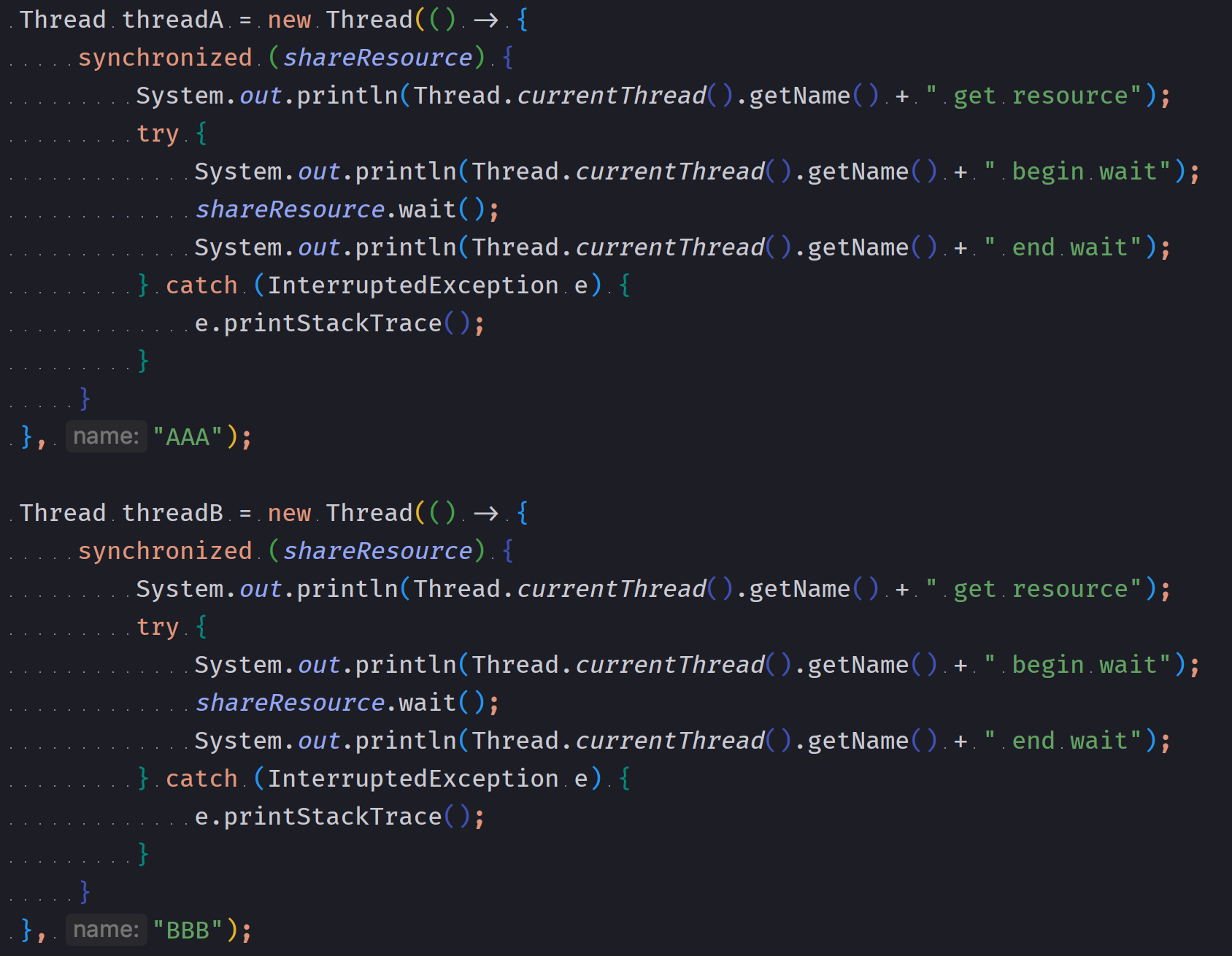


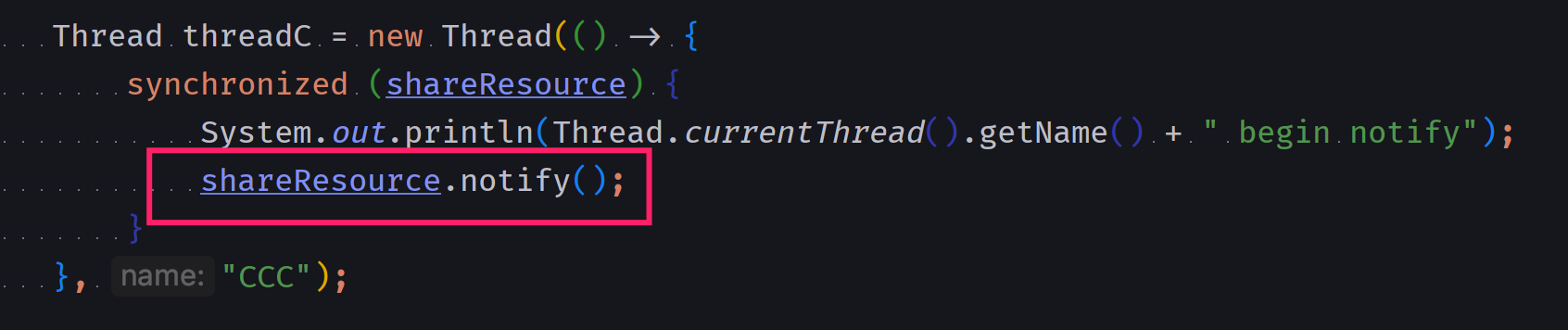


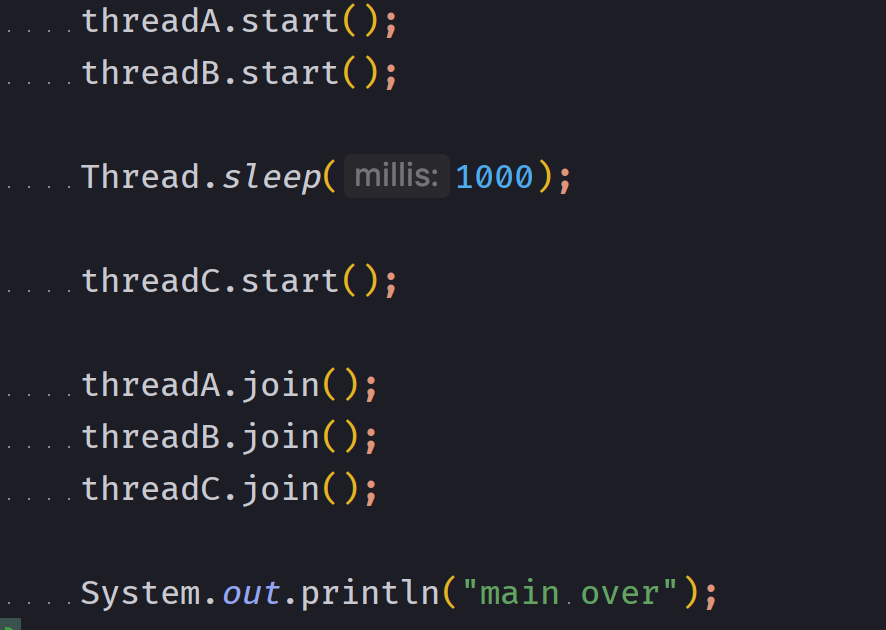




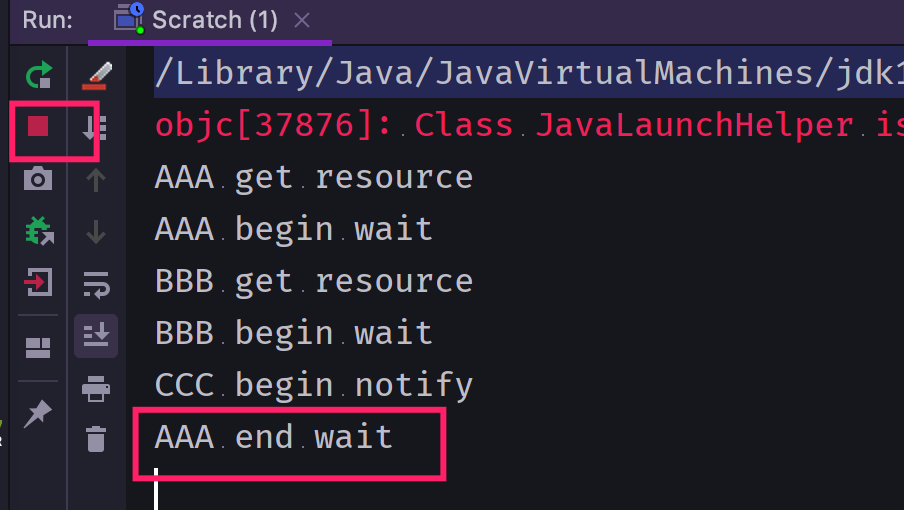
* + 1. notifyAll(): 唤醒所有在该共享变量上由于调用 wait 系列方法而被挂起的线程
    2. notifyAll()方法只会唤醒调用这个方法前调用了 wait系列函数而被放入共享变量等待集合里面的线程。如果调用notifyAll()方法后一个线程调用了该共享变量的 wait()方法而被放入阻塞集合，则该线程是不会被唤醒的



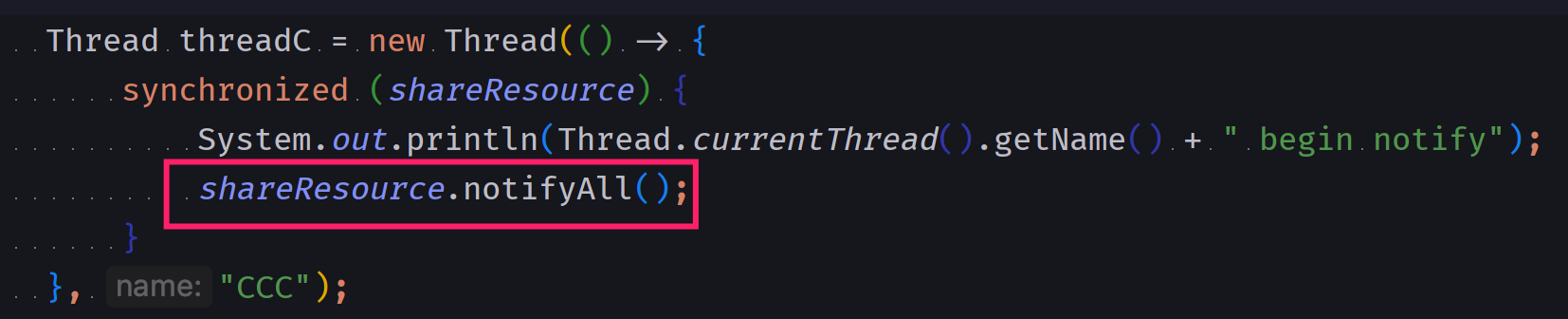


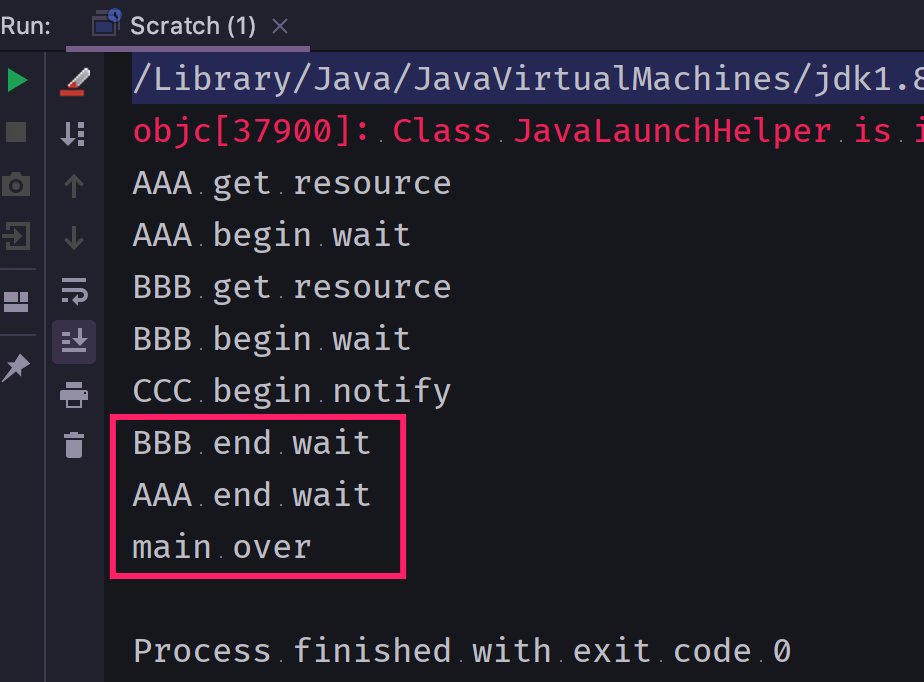


notify效果

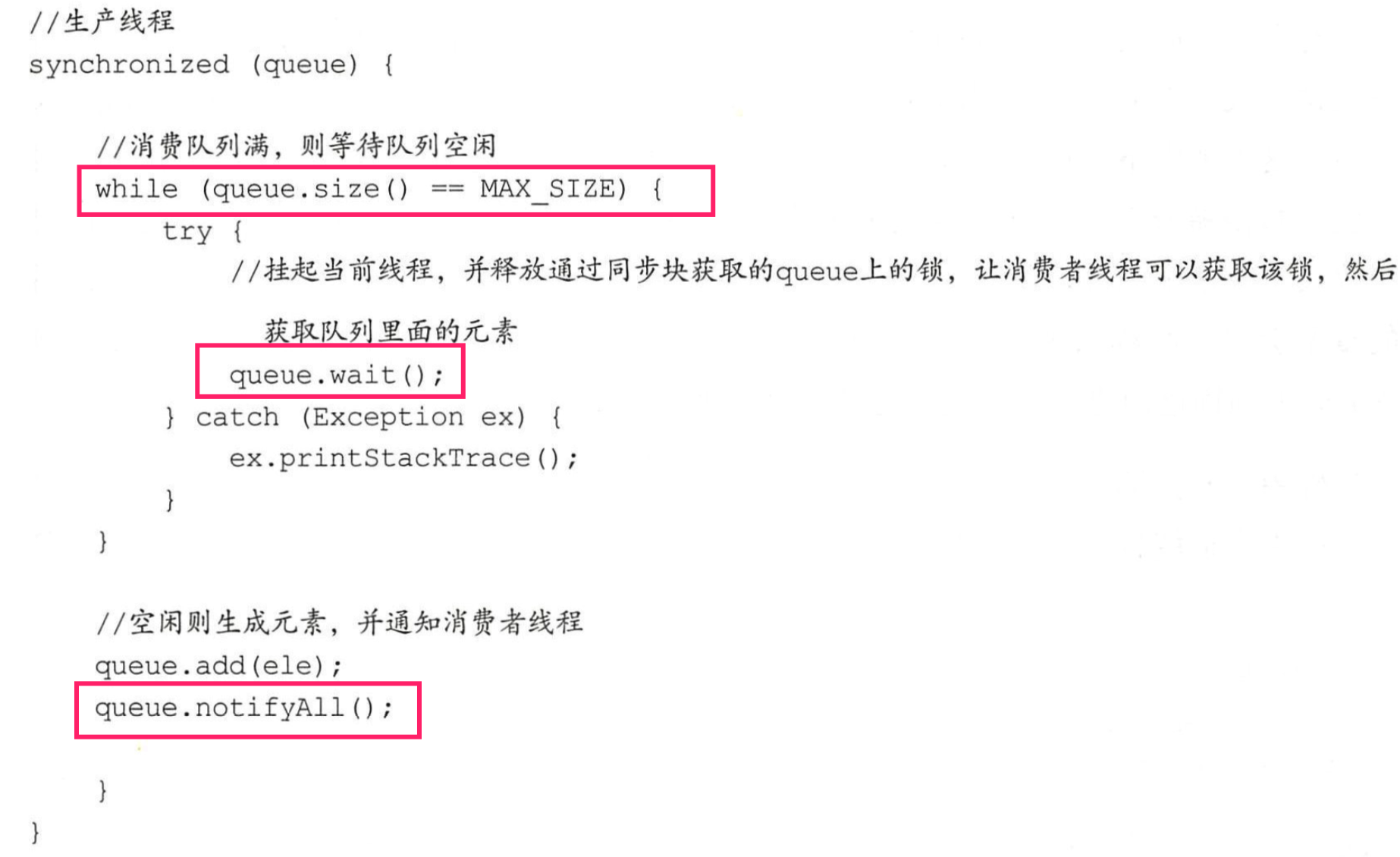


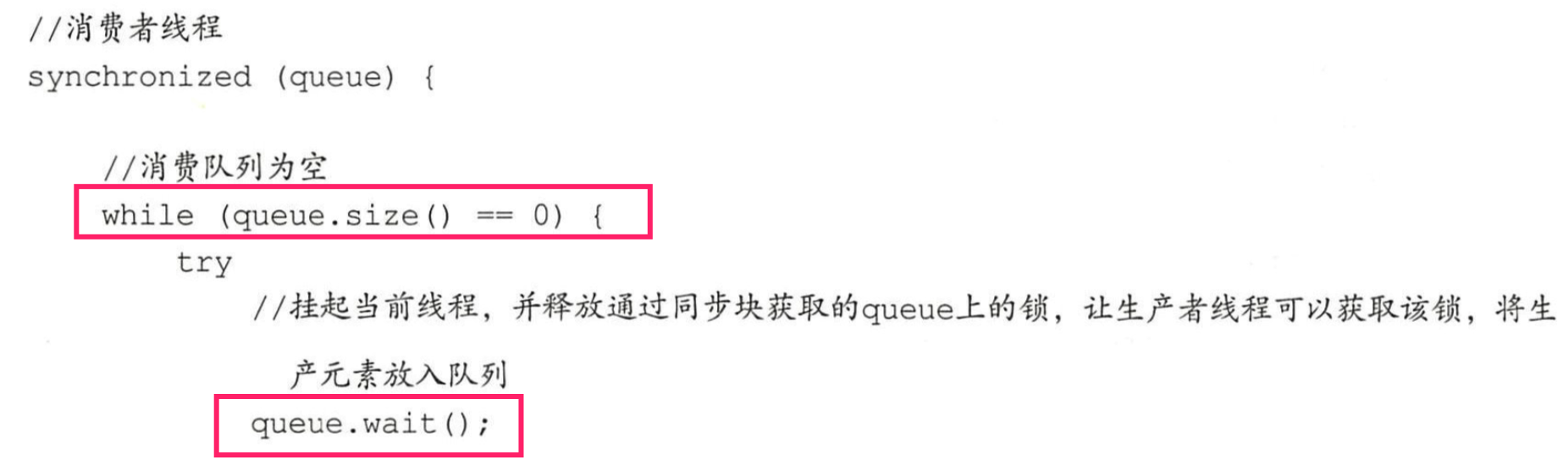
notifyAll效果

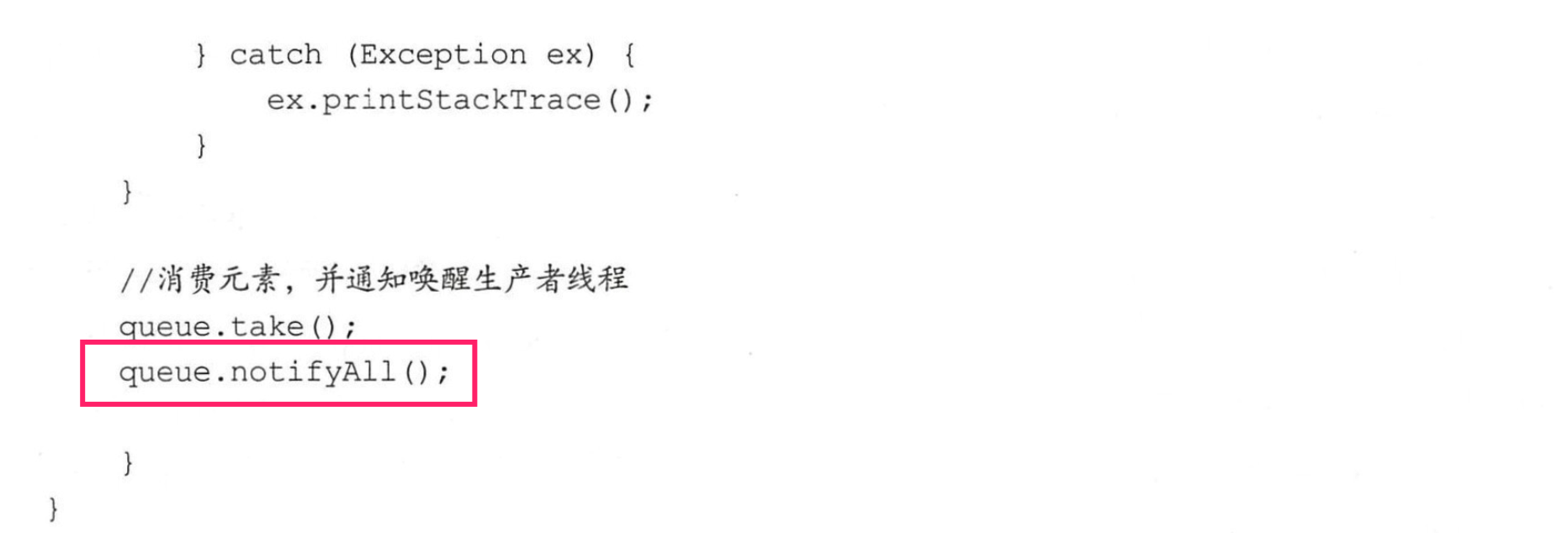




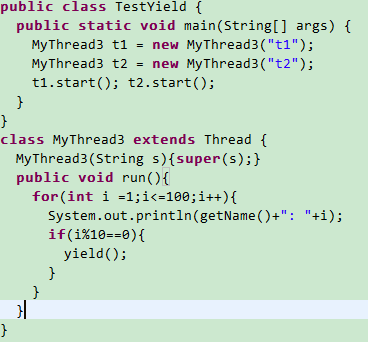
经典的生产者/消费者模式示例



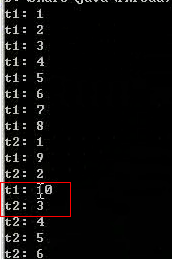




1. join方法: 等待线程执行终止
   1. 合并某个线程(调用完之后，线程的调用就相当于方法调用，必须等待该线程执行完才继续其他线程)
   2. A线程调用B线程的join方法后，A线程会被阻塞，等待B线程执行完成后返回；B线程执行完毕后b.join()方法会返回。A线程等待join返回阻塞状态下，若被中断interrupt，即调用a.interrupt()，会中断A的阻塞抛出InterruptedException
2. sleep方法: 让线程睡眠
   1. sleep()使当前线程进入停滞状态(阻塞当前线程)，让出CPU的使用，目的是不让当前线程独自霸占该线程所获的CPU资源，以留一定时间给其他线程执行的机会
   2. sleep()是Thread类的static(静态)的方法；因此他不能改变对象的机锁，所以当在一个synchronized块中调用sleep()方法是，线程虽然休眠了，但是对象的机锁并没有被释放，其他线程无法访问这个对象(即使睡着也持有对象锁)
   3. 在sleep()休眠时间期满后，该线程不一定会立即执行，这是因为其它线程可能正在运行而且没有被调度为放弃执行
   4. 如果在睡眠期间其他线程调用了该线程的 interrupt()方法中断了该线程，则该线程会在调用 sleep 方法的地方抛出 InterruptedException异常而返回
   5. 在哪个线程里调用的Thread.sleep(long millis)方法，就让哪个线程睡眠
3. yield方法：让出CPU执行权
   1. 建议让出CUP，给其他线程执行的机会，但是只是建议，并没有任何机制保证他将被采纳。线程调度器可以无条件忽略这个建议，因此，对于任何重要的控制或在调整应用时，都不能依赖于yield

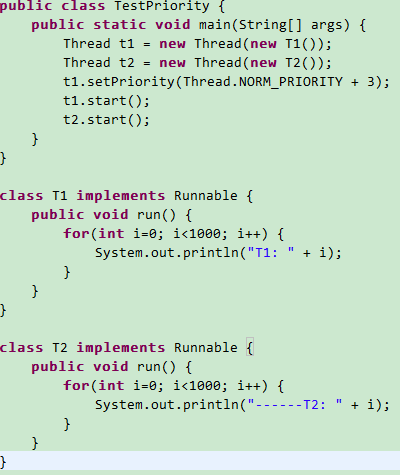


输出：



1. 线程的优先级：优先级越高的，得到的cup的执行时间片就越多。





输出：t1得到的执行时间明显比t2多