```
public static void main(String[] args) {
    final ReentrantLock reentrantLock = new ReentrantLock();
    final Condition condition = reentrantLock.newCondition();

    Thread thread = new Thread((Runnable) () -> {
        reentrantLock.lock();  //获取锁(一般写在try上面)
        System.out.println("thread获取锁!");
        try {
            System.out.println("thread要等一个新信号!");
            //await会释放锁,直到(1.被signal, 2.signal的线程释放锁)两个条件后又获取
锁
            //即thread1的condition.signalAll()和reentrantLock.unlock()
            condition.await();
        }
        catch (InterruptedException e) {
            e.printStackTrace();
        } finally{
            System.out.println("thread拿到一个信号!");
            reentrantLock.unlock();  //释放锁(一般写在finally里)
            System.out.println("thread释放锁!");
        }
    }, "waitThread1");

    thread.start();

    Thread thread1 = new Thread((Runnable) () -> {
        reentrantLock.lock();  //获取锁
        System.out.println("thread1拿到锁了");
        try {
            Thread.sleep(3000);
        }
        catch (InterruptedException e) {
            e.printStackTrace();
        }
        condition.signalAll();  //通知await的线程
        System.out.println("thread1发了一个信号！！");
        reentrantLock.unlock();  //释放锁(如果不释放,其他await的线程也不能拿到锁,
会一直处于阻塞状态)
```

```
            System.out.println("thread1释放锁!");
        }, "signalThread");

        thread1.start();
    }
```

输出:

thread获取锁!
thread要等一个新信号!
thread1拿到锁了
thread1发了一个信号！！
thread1释放锁!
thread拿到一个信号!
thread释放锁!

wait()和lock()的区别

在使用Lock之前，我们都使用Object 的wait和notify实现同步的。举例来说，一个producer和consumer，consumer发现没有东西了，等待，produer生成东西了，唤醒。

线程consumer	线程producer

synchronize(obj){
obj.wait();//没东西了，等待
}
	synchronize(obj){
	obj.notify();//有东西了，唤醒
	}

有了lock后，世道变了，现在是:

lock.lock();
condition.await();
lock.unlock();
	lock.lock();
	condition.signal();
	lock.unlock();

为了突出区别，省略了若干细节。区别有三点:

1. lock不再用synchronize把同步代码包装起来;
2. 阻塞需要另外一个对象condition;
3. 同步和唤醒的对象是condition而不是lock，对应的方法是await和signal，而不是wait和notify。

```
Lock lock  = new ReentrantLock();
lock.lock();   //相当于synchronize块/synchronize方法块的开始
try{
```

```
    //可能会出现线程安全的操作
}finally{
    //一定在finally中释放锁
    //也不能把获取锁在try中进行，因为有可能在获取锁的时候抛出异常
    lock.ublock();    //相当于synchronize块/synchronize方法块的结束
}
```