Java Day 17

线程安全

线程安全问题：-->TicketThread.java 、TicketRunnable.java

多线程并发操作同一数据时，就有可能出现线程安全问题

同步技术可以解决这种问题，把操作数据的代码进行同步，不要多个线程一起操作

线程安全-->Account.java,AccountThread.java

锁的粒度：太细，并发性降低；太粗，数据可能不安全

一般将多线程操作的共享对象，作为锁对象

对共享数据进行操作的，可能引起数据不一致的代码，放到同步代码块中

“临界区”:对共享资源进行操作的代码区域

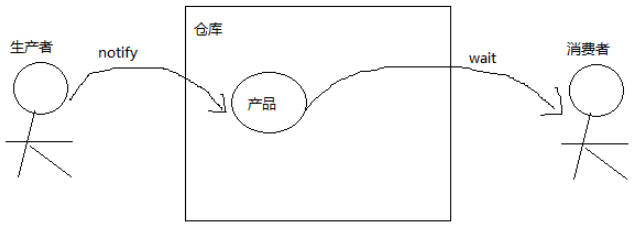
线程间的通信-->WaitNotifyTest.java

1）什么时候需要通信

多个线程并发执行时，在默认情况下 CPU 是随机切换线程的

如果我们希望他们有规律地执行，就可以使用线程通信

生产者消费者问题?



2）怎么通信

如果希望线程等待，就调用 java.lang.Object.wait()

如果希望唤醒等待的线程，就调用 java.lang.Object.notify();

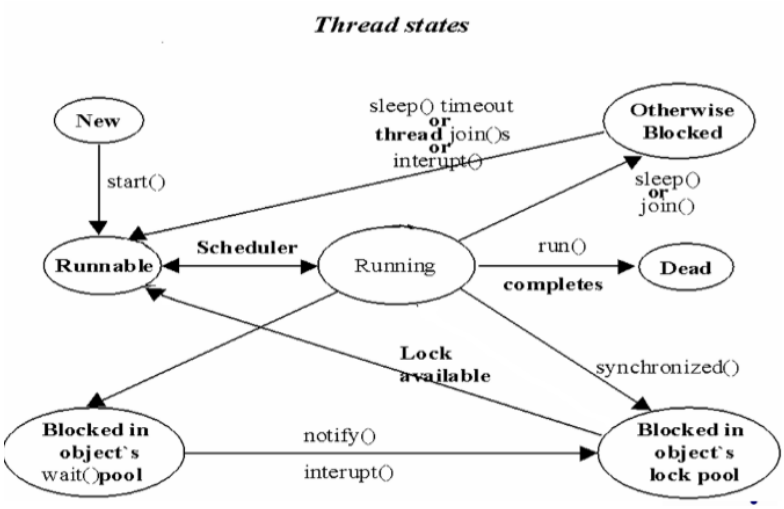
这两个方法必须在同步代码中执行，并且使用同步锁对象来调用

（synchronized 锁对象和调用 wait()/notify()的对象必须相同）

如果就执行一次，确保 wait() 在 notify() 之前

-->WaitNotifyTest2.java

线程状态转换图



sleep 方法和 wait 方法的区别?

1）这两个方法来自不同的类分别是 Thread 和 Object

2）sleep 方法在同步方法或同步代码块中,不释放锁；wait 方法在同步方法或者同步代码块中,释放锁

死锁:-->DeadLockTest.java

“哲学家就餐问题”

两个或两个以上的线程在执行过程中，因争夺资源而造成的一种互相等待的现象

多线程同步的时候，如果同步代码嵌套，使用相同锁，就有可能出现死锁

同步代码块尽量不要嵌套使用

死锁的发生必须具备以下四个必要条件：

1）互斥条件：指线程对所分配到的资源进行排它性使用，即在一段时间内某资源只由一个线程占用。如果此时还有其它线程请求资源，则请求者只能等待，直至占有资源的线程用毕释放。

2）请求和保持条件：指线程已经保持至少一个资源，但又提出了新的资源请求，而该资源已被其它线程占有，此时请求线程阻塞，但又对自己已获得的其它资源保持不放。

3）不剥夺条件：指线程已获得的资源，在未使用完之前，不能被剥夺，只能在使用完时由自己释放。

4）环路等待条件：指在发生死锁时，必然存在一个线程资源的环形链，即线程集合{P0，P1，P2,；Pn}中的 P0 正在等待一个 P1 占用的资源；P1 正在等待 P2 占用的资源，……，Pn 正在等待已被 P0 占用的资源。

通过给共享资源编号，顺序使用共享资源，解决死锁

线程优先级

---

yield()：当前线程让出 CPU，让其它线程有机会运行

setPrioirity(int)：设置线程的优先级，java 中线程有 1~10 个优先级，数值越大优先级别越高