# 多线程

## 一、进程和线程

### 1.进程

**每一个运行的程序都是一个进程，进程与进程之间相互独立，独立分配资源**。**进程是一个在内存中运行的程序**，如一个运行的xxx.exe就是一个进程。每个进程都有自己独立的一块内存空间。如：电脑上的Eclipse或者QQ并不是进程，当双击运行Eclipse或者QQ时，Eclipse或QQ运行之后才能称为一个进程。

一个进程至少有一个线程，一个进程可以运行多个线程，多个线程可共享数据。**线程共享进程的资源**。

**进程是程序（任务）的执行过程，它持有资源（共享内存，共享文件）和线程。**

进程是具有一定独立功能的程序在一个数据集上的一次动态执行的过程，是应用程序运行的载体。

进程一般由**程序**、**数据集合**和**进程控制块**三部分组成。程序用于描述进程要完成的功能，是控制进程执行的指令集；数据集合是程序在执行时所需要的数据和工作区；程序控制块(Program Control Block，简称PCB)，包含进程的描述信息和控制信息，是进程存在的唯一标志。

**进程的特征**：

①动态性：进程是程序的一次执行过程，是临时的，有生命期的，是动态产生，动态消亡的；

②并发性：任何进程都可以同其他进程一起并发执行；

③独立性：进程是系统进行资源分配和调度的一个独立单位；

④结构性：进程由程序、数据集合和进程控制块三部分组成。

### 2.线程

#### ⑴线程

**线程**：是组成进程的基本单位。线程之间也是相互独立的，占用同一资源。线程是进程中的一个控制单元，负责当前进程中程序的执行。**线程是(cpu)处理器任务调度和执行的基本单位，是程序执行的最小单位，也是任务执行的最小单位**，**而进程是操作系统资源分配的最小单位**。

同类的多个线程共享进程的**堆**和**方法区资源**，但每个线程都有自己的**程序计数器(Program Counter Register)**、**虚拟机栈(VM Stack)**和**本地方法栈(Native Method Stack)**。

**一个java程序的运行是main线程和多个其它线程共同运行的**。

#### ⑵程序计数器

**程序计数器**：在多线程情况下程序计数器用于记录当前线程执行的位置，从而当线程被切换回来的时候能够知道该线程上次运行到哪儿了。注意如果执行的是 native 方法，那么程序计数器记录的是 undefined 地址，只有执行的是 Java 代码时程序计数器记录的才是下一条指令的地址。

程序计数器私有主要是为了**线程切换后能恢复到正确的执行位置**。

#### ⑶虚拟机栈

**虚拟机栈**：每个 Java 方法在执行的同时会创建一个栈帧用于存储局部变量表、操作数栈、常量池引用等信息。从方法调用直至执行完成的过程，就对应着一个栈帧在 Java 虚拟机栈中入栈和出栈的过程。

#### ⑷本地方法栈

**本地方法栈**：和虚拟机栈所发挥的作用非常相似，区别是： **虚拟机栈为虚拟机执行 Java 方法 (也就是字节码)服务，而本地方法栈则为虚拟机使用到的 Native 方法服务。**

为了**保证线程中的局部变量不被别的线程访问到**，虚拟机栈和本地方法栈是线程私有的。

### 3.线程和进程的区别

线程具有许多传统进程所具有的特征，故也被称为轻型进程(Light-Weight Process)；把传统的进程称为重型进程(Heavy-Weight Process)。

**每一个运行的程序都是一个进程，进程与进程之间相互独立，独立分配资源**。一个进程至少有一个线程，一个进程可以运行多个线程，多个**线程共享进程的资源。**

线程是组成进程的基本单位。线程之间也是相互独立的。线程负责当前进程中程序的执行。

**①根本区别**：**进程是操作系统资源分配的基本(或最小)单位；而线程是(cpu)处理器任务调度和执行的基本单位，是程序执行的最小单位，也是任务执行的最小单位**。

**②上下文切换**：**线程上下文切换比进程上下文切换要快的多**。

**③包含关系：若一个进程中包含多个线程，则执行过程不是一条线，而是多条线(多个线程)共同完成。线程是组成进程的基本单位，线程也称为轻量级进程**。

**④内存分配：同一进程的多个线程之间共享本进程的地址空间和资源，而多个进程之间的地址空间和资源是相互独立的**。

**⑤影响关系**：一个进程崩溃后，在保护模式下不会对其他进程产生影响，但是一个线程崩溃整个进程都死掉。所以多进程要比多线程健壮。

早期的操作系统中并没有线程的概念，早期的进程是能拥有资源和独立运行的最小单位，也是程序执行的最小单位。它相当于一个进程里只有一个线程，进程本身就是线程。

### 4.多线程

**多进程：操作系统中同时运行的多个程序。**

**多线程：在同一个进程中同时运行的多个任务。**

多线程具有**随机性**，造成的原因：CPU在瞬间不断切换去处理各个线程而导致的，可以理解成多个线程在抢CPU资源**。**

多线程不能提高运行速度，但能提高运行效率，使CPU的使用率更高。若多线程有安全问题或者进行频繁的上下文切换，则运行速度反而更低。

**同一时刻CPU只能执行一个线程。**说多线程是同时执行是因为多个线程之间在CPU中的切换速度极快，用户无法通过肉眼观察到切换，看上去像多个线程在同时执行。

**多线程分类**：**抢占式**、**非抢占式**。抢占式是根据线程的优先级或时间来分配资源。非抢占式是等到运行中的线程执行完之后再去占资源。**Java中多线程的方式是抢占式的**。

**多线程的好处**：改善了系统资源的利用率，提高系统的处理能力。

### 5.线程的同步和线程的异步

**线程同步**：就是多个线程同时访问同一个资源，等待资源访问结束。这样比较浪费时间、效率低。

**线程异步**：线程在访问资源时，在空闲等待时间同时访问其他资源，从而实现多线程机制。

线程同步的好处和弊端：解决了线程安全问题，但是降低了程序的运行效率。

**实现线程同步的方式**：

**①同步代码块**：被synchronized关键字修饰的代码块就是同步代码块。

**②同步方法**：被synchronized关键字修饰的方法就是同步方法。

单线程才是一个线程执行完之后才继续执行下一个线程。而多线程则是多个线程同时执行。

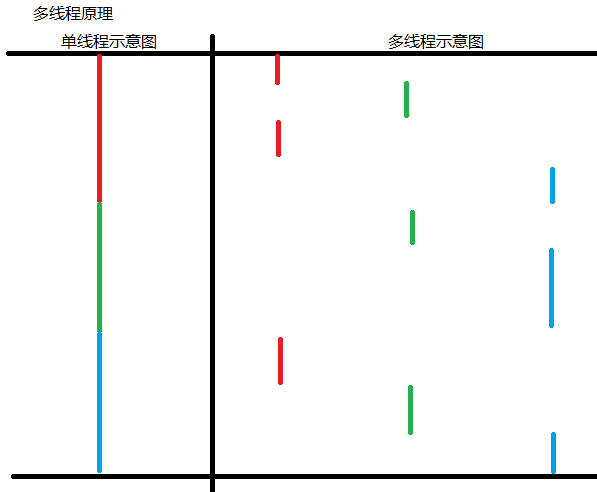
### 6.任务调度(知识扩展)

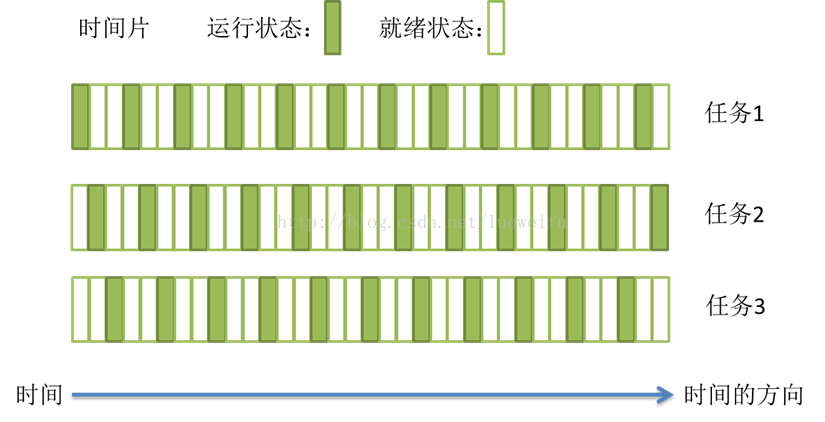
大部分操作系统(如Windows、Linux)的**任务调度**是采用**时间片轮转的抢占式调度方式**，也就是说**一个任务执行一小段时间后强制暂停去执行下一个任务，每个任务轮流执行**。任务执行的一小段时间叫做**时间片**，任务正在执行时的状态叫**运行状态**，任务执行一段时间后强制暂停去执行下一个任务，被暂停的任务就处于**就绪状态**，被暂停的任务然后等待下一个属于它的时间片的到来。这样每个任务都能得到执行，由于CPU的执行效率非常高，时间片非常短，在各个任务之间快速地切换，给人的感觉就是多个任务在“同时进行”，这也就是我们所说的并发。

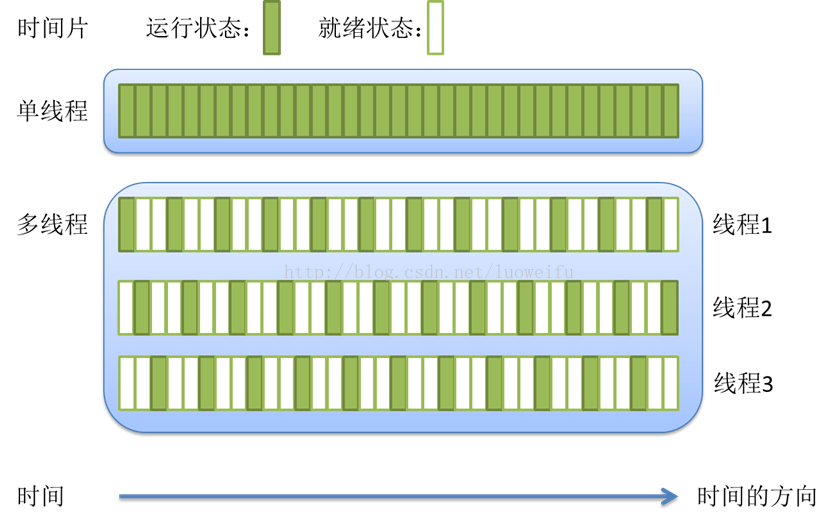
**并发就是多个任务同时执行**。

计算机的核心是**CPU，它承担了所有的计算任务**。操作系统是计算机的管理者，它**负责任务的调度、资源的分配和管理**，统领整个计算机硬件。

应用程序是具有某种功能的程序。







## 二、线程应用

### 1.线程的优先级

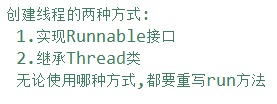
线程默认的优先级是**5**。main函数是一个主线程。线程的优先级是1~10，优先级最高的是10。优先级越高，获得的资源的概率就越大，但是不一定能获得资源。Thread类中三个优先级的常量如下：

**public** **final** **static** **int** ***MIN\_PRIORITY*** = 1;

**public** **final** **static** **int** ***NORM\_PRIORITY*** = 5;

**public** **final** **static** **int** ***MAX\_PRIORITY*** = 10;

### 2.创建线程的方式



**法一：是继承Thread类，这种启动线程的方式是直接创建该类，然后直接调用start()方法。**

**法二：是实现了Runnable接口。这种启动线程要先创建该类对象并把对象作为参数传到Thread对象中。**

#### ⑴继承Thread类

java中的Thread类是一个实现了Runnable接口的类。Thread类一些常用的构造器和方法如下：

**public** **class** Thread **implements** Runnable {...}

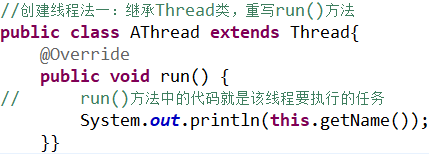
**①常用的构造器**：

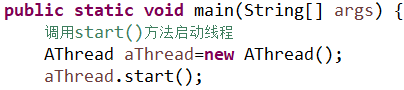
**public** Thread(){...}

**public** Thread(String name)

**public** Thread(Runnable target){...}

**public** Thread(Runnable target, String name){...}





**②常用的方法：**

还没补充

#### ⑵实现Runnable接口

还没补充

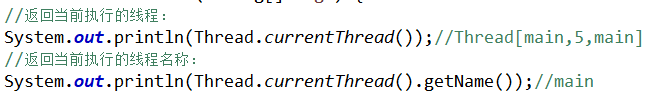
创建线程的方法中，继承Thread类有一个弊端，由于java中类之间没有多继承，因此当一个类继承了Thread类之后，则不能再继承其他类了。而实现Runnable接口的方法创建线程则没有这个弊端。

注意当一个类是通过实现Runnable接口，再重写run()方法的方式创建线程时，要在run()方法内获得线程的名称时只能通过Thread.currentThread().getName()的方法获得线程名。而通过继承Thread类，并重写run()方法创建线程时，在run()方法内获得线程名可以通过this.getName()或Thread.currentThread().getName()的方法获得线程名。

### 2.常用的方法

Thread**.currentThread()**：返回值为当前执行的线程，返回的类型为Thread类型。

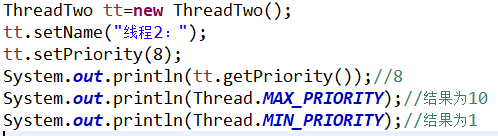
Thread**.getName()**：返回值为String类型的线程名称。



Tread**.setName()**：为线程设置新名称。

Thread**.setPriority(int newPriority)**：为线程设置优先级。

Thread**.getPriority()**：获得线程的优先级，返回值为int类型。



Thread**.start()**：启动线程调用的是start()方法，而不是run()方法。

Thread**.join()**：将该线程加入某个线程中，只有等该线程结束才会继续执行其他线程。

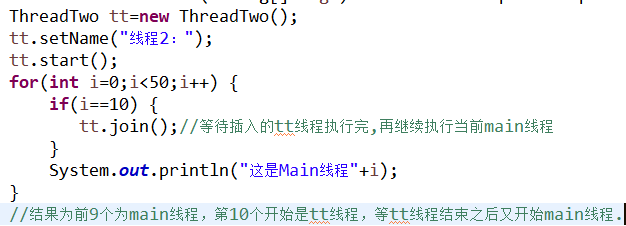
Thread**.sleep(long millis)**：线程的休眠，括号内的参数为休眠的毫秒数。在自定义的线程中写这个会报错，要try...catch，而在main方法中写这个则不会报错。**注意在线程的休眠过程中不会释放资源**。

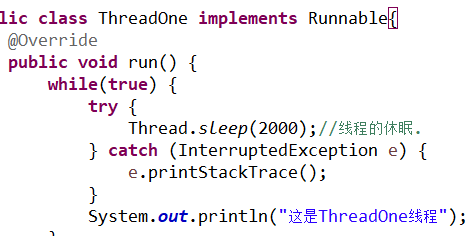
**sleep()和waite()的区别**：

sleep()方法是Thread的方法，是让线程休眠，同时不会释放资源。

waite()方法是Object的方法，让占有当前资源的线程等待,同时释放资源。





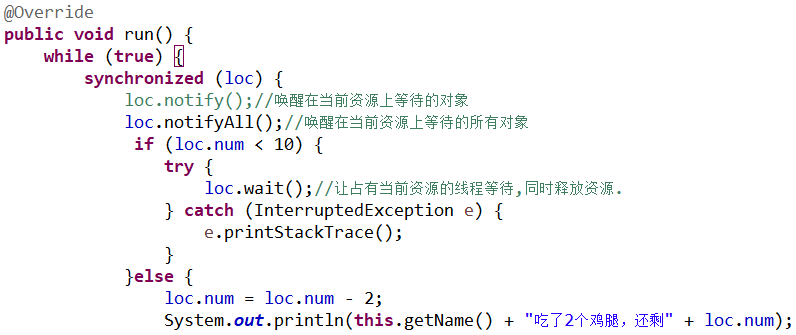


注意下面这三个方法都属于Object对象的方法，而不是Thread的方法。

Object**.notify()**：唤醒在当前资源上等待的对象

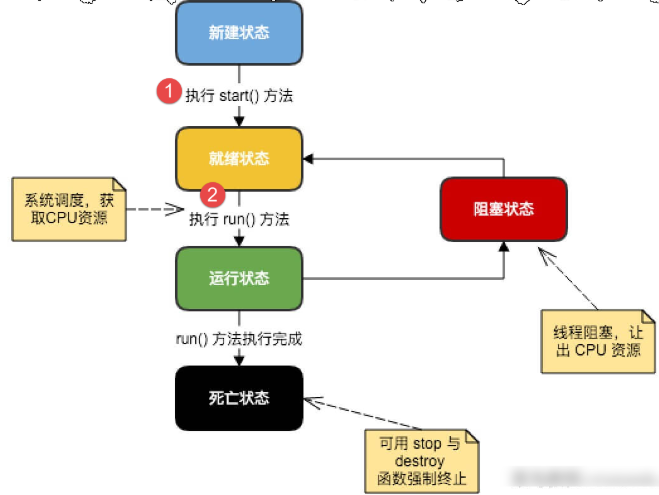
Object**.notifyAll()**：唤醒在当前资源上等待的所有对象

Object**.wait()**：让占有当前资源的线程等待,同时释放资源。



### 3.线程的生命周期

线程的生命周期示意图：



**①新建状态(New)**：使用new创建一个线程对象之后，该线程对象就处于新建状态。直到调用start()方法启动该线程之前，该线程对象一直处于新建状态。

**②就绪状态(Runnable)**：当线程对象调用start()方法启动线程，该线程就处于就绪状态，该状态的线程处于可运行线程池中，变得可运行。处于就绪状态的线程需要等待JVM里线程调度器的调度。

**③运行状态(Running)**：处于就绪状态的线程若获得了CPU资源则可以执行run()方法，此时线程便处于运行状态。处于运行状态的线程可以变为就绪状态、阻塞状态、死亡状态。

**④阻塞状态(Blocked)**：线程因为某种原因放弃了CPU的使用权，暂时停止运行，线程就处于阻塞状态。

阻塞得情况分为三种：

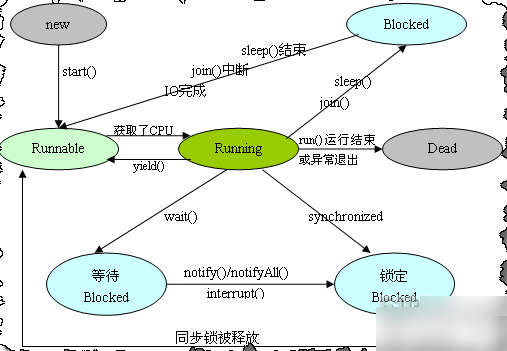
**等待阻塞**：运行得线程执行wait()方法，JVM会把该线程放在等待池中。

**同步阻塞**：运行的线程在获取对象的同步锁时，若该同步锁被别的线程占用，则JVM会把该线程放入锁池中。

**其他阻塞**：运行的线程执行sleep()或join()方法，或者发出了I/O请求时，JVM会把该线程置为阻塞状态。当sleep()状态超时、join()等待线程终止或者超时、或者I/O处理完毕时，线程重新转入就绪状态。

**⑤死亡状态(Dead)**：一个运行状态的线程完成任务(执行完run()方法)或者异常终止时，该线程就进入了死亡状态。

线程的状态转换图：



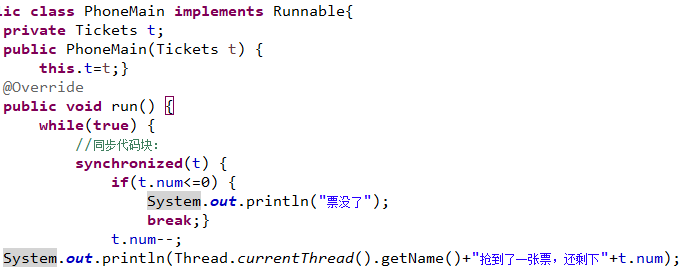
### 4.同步代码块和同步方法

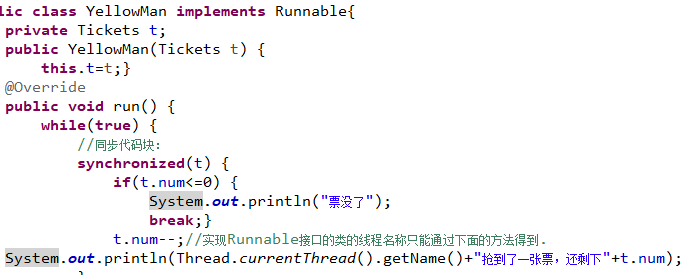
当多个线程对于同一个资源进行修改时，可能会产生问题，解决办法就是利用同步的方式。

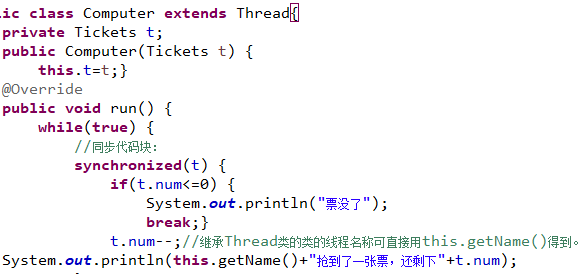
**同步代码块**，使用**synchronized**关键字修饰，需要写在线程的run方法中，synchronized()括号中为要同步的资源。指的是，当前线程在执行任务时，给自己锁上，锁就是资源对象，锁住的过程中，其他线程不会来抢夺资源。

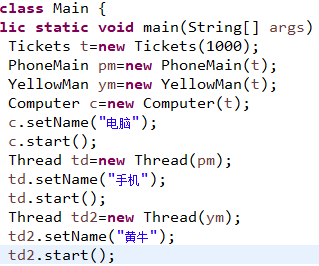
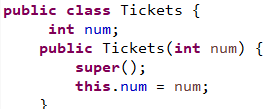
**同步方法**，使用**synchronized**关键字修饰方法，需要写在资源类中，**普通同步方法的锁为当前方法所属的那个对象，静态方法的锁是当前类的class对象**。

**同步代码块例子**：

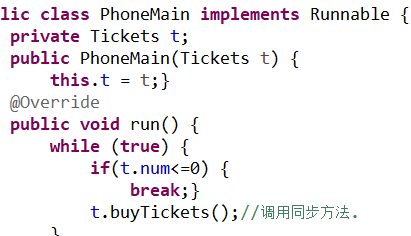


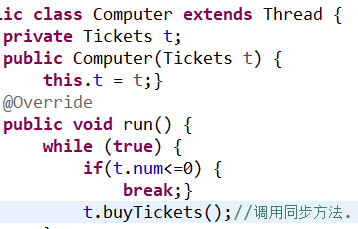


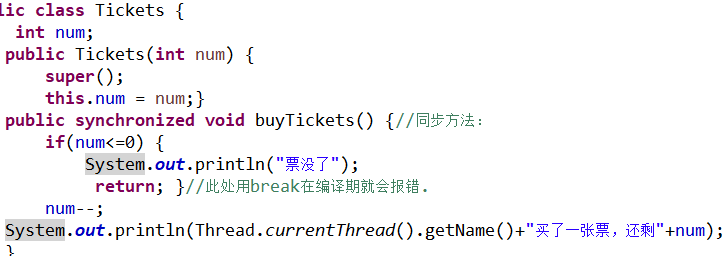


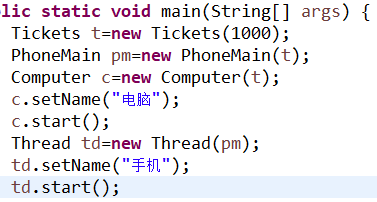


**同步方法的例子：**









注意：**break语句只能用于循环或者switch语句中**。用于循环中表示跳出整个循环。break语句不能用于单独的if语句中，会出现编译错误。上面代码中if语句内写了break语句可以正常运行是因为if语句外面还有一层while循环。break语句是跳出的整个while循环，而不是跳出if语句。

在抢资源的时候，可以一个用同步代码块，一个用同步方法，只要保证抢夺的是同一个对象即可。

## 三、死锁：（看书签）

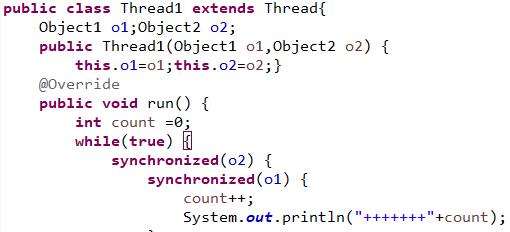
**死锁：**多个线程因竞争资源而造成的一种僵局(相互等待)，若无外力作用，这些进程都将无法向前推进。

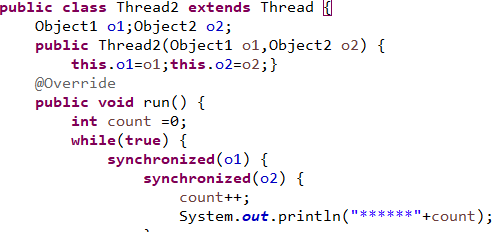
**死锁产生的原因**：

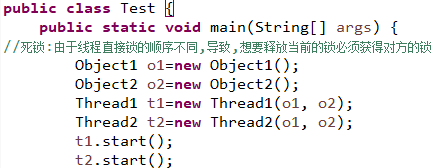
**死锁产生的条件**：

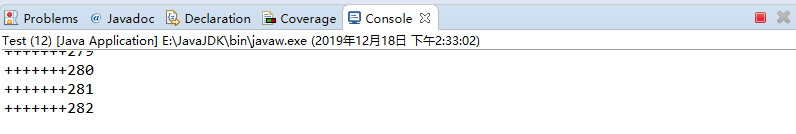
**死锁解决的办法**：调整锁的顺序。

死锁例子：









这个程序不再继续执行了，但是程序还没有退出。

**类的对象实例可以有很多个，但是每个类只有一个class对象**，所以不同对象实例的对象锁是互不干扰的，但是每个类只有一个类锁

**例题：设计一个show方法，只能调用一次，否则抛出异常。**(思路：同一个线程只能启动一次，即Thread**.start()**方法只能调用一次，再次调用会报错。将该方法定义在一个类中，该类实现Runnable方法或继承Thread接口，再重写run()方法，则该类就是一个线程了。该线程只能调用一次start()方法，即在调用show()方法的同时也调用start()方法就可以实现show()方法只能调用一次，即在show()方法里面用**this.start()**调用start()方法，相当于每调用一次show()方法的同时也在调用start()方法。)。

