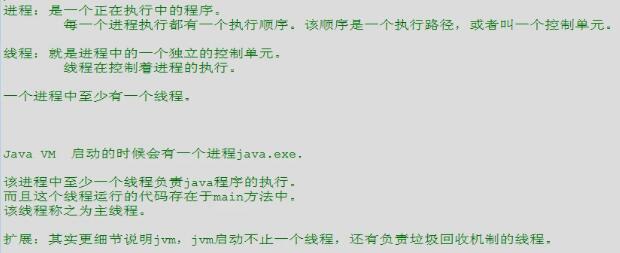
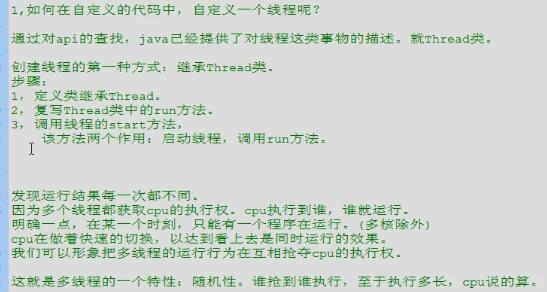
**P130 多线程（概述）**



**P131 多线程（创建线程-继承Thread类）**

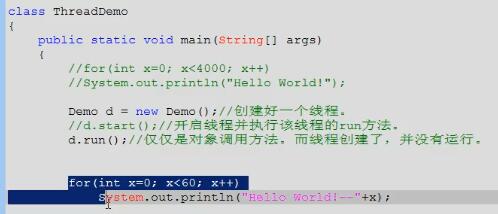


**P132 多线程（创建线程-run和start特点）**

创建线程的步骤：定义类继承Thread 🡪 复写Thread类中的run方法

目的就是将自定义的代码储存在run方法，让线程运行。

其实，程序运行的入口main函数也是一个线程，只不过是主线程。主线程就把自定义的代码储存在main方法中，让主线程运行。所以，要想新建一个线程运行一段程序，该新线程就得有个空间储存新线程的代码，该空间就是run。为什么要新建空间呢？很简单，如果跟主线程一样也放在main方法中，那不就成了主线程中的代码，由主线程来完成运行了吗。



线程创建好了之后要由start函数来开启，如果选择调用run方法，就变成了主线程创立了一个对象，并调用了对象中的run方法，这一系列操作就将由主线程完成，那么就变成了主线程先运行run方法，等run方法全部运行完成以后，再接下来执行下面的循环。

**P133 多线程（线程练习）**

//

**P134 多线程（线程运行状态）**

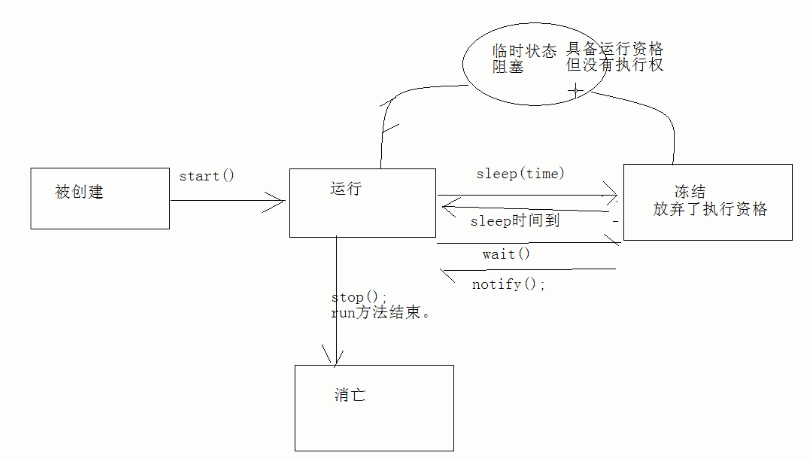
从冻结状态恢复后，该线程可能直接进入运行，也可能先进入临时状态。

运行状态：既有运行资格，又有执行权

临时状态：有运行资格，但没有执行权

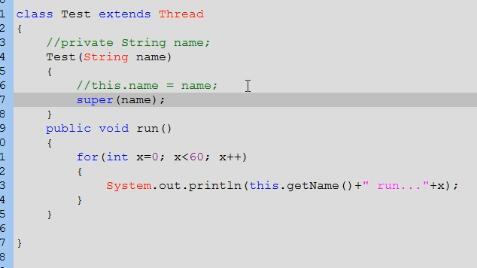
冻结状态：放弃了执行资格

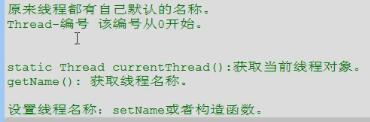
--wait和sleep的区别：sleep是冻结你预先设定好的时间，然后自动恢复；wait是无限制的冻结下去直至你手动唤醒。



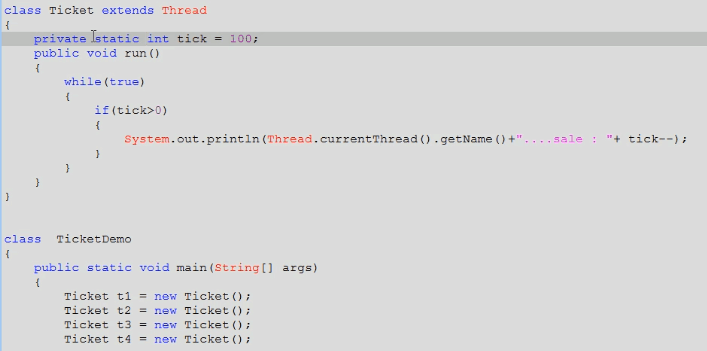
**P135 多线程（获取线程对象以及名称）**

想自定义线程名称时。绿色部份的方法传递不进去，要么用父类的构造函数，由父类定义好的一系列功能传递进去（推荐），要么就this.setName(name)传进去。

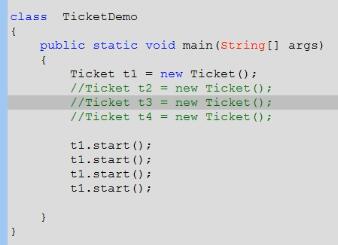




**P136 多线程（售票的例子）**



如果不想把tick定义为静态，能否创建4个相同的线程？



不行，程序会报错：线程状态出问题了

因为通过第一个start，你已经让线程从创建状态进入了运行状态，后面的start就没意义了

**P137 多线程（创建线程-实现Runable接口）**

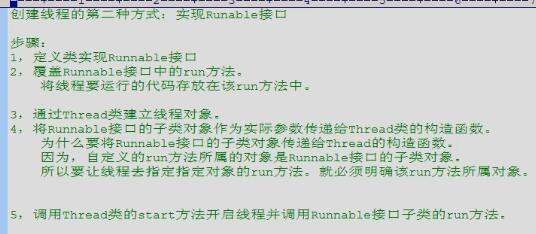
前面的例子，我们都是使用创建线程的第一种方法

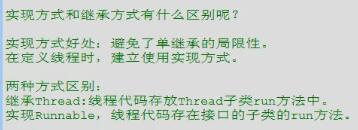




另一个方法就是使用Runable接口

Runable接口应该由那些打算通过某一线程执行其实例的类来实现。类必须定义一个称为run的无参数方法。

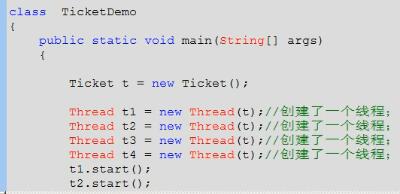




而且，实现方式的话，变量不需要static也能全局共享。如下例，因为你只建立了一个Ticket对象。

再次强调下接口子类的定义

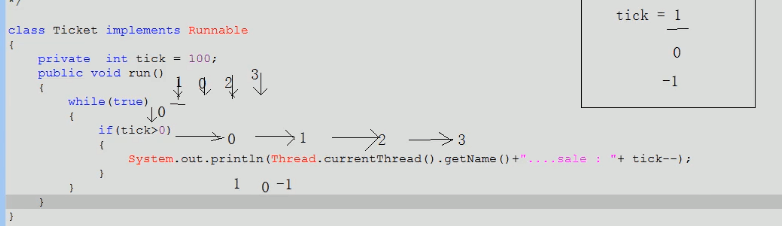




该类该怎么定义怎么定义，该怎么继承怎么继承。但该类设计完成后，加上Runable接口，并额外加一个run方法，把想要让线程运行的程序放进去即可。

Thread类的构造函数接受Runable类型的接口。这也是多态的体现。

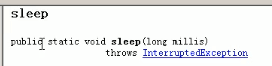
**P138 多线程（多线程的安全问题）**



举一个特殊例子来说明安全问题

比如要卖100张tick，当tick=1时，线程0进入if判断并通过，此时cpu切换到线程1；线程1进入if判断并通过，此时cpu切换到线程2；线程2进入if判断并通过，此时cpu切换到线程3。此时切换回线程0，输出tick=1并且tick值减1；此时切换回线程1，输出tick=0并且tick值减1；此时切换回线程2，输出tick=-1并且tick值减1；此时切换回线程3，输出tick=-2并且tick值减1。

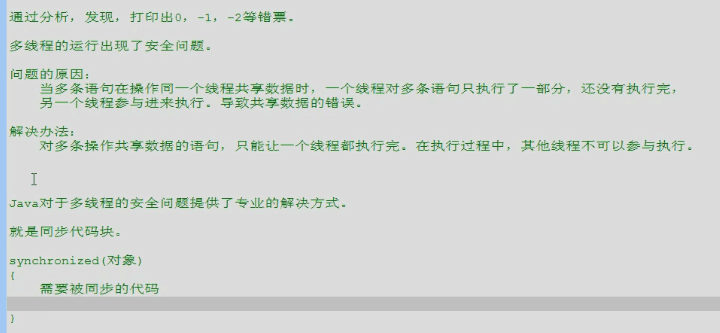
我们调用sleep方法来实现这个安全问题



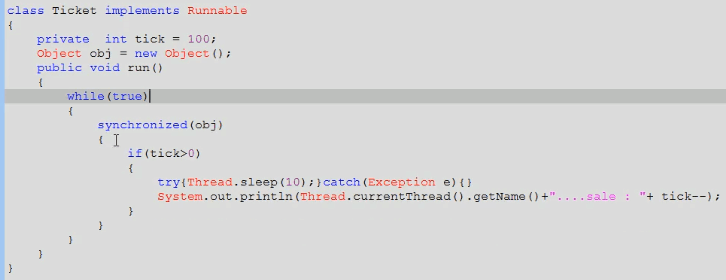
可以看到，调用sleep方法会抛出一个异常。

一般异常的解决有两种：继续抛出（因为我们定义的类实际上是Runable接口的子类，接口并没有处理异常的方法，所以该解决方法不可行）或者就地处理

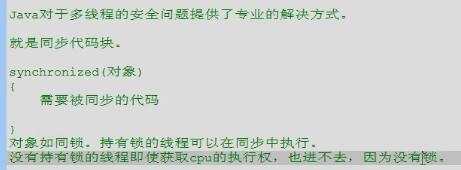




解决方法

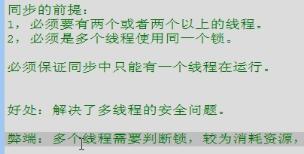


**P139 多线程（多线程同步代码块）**

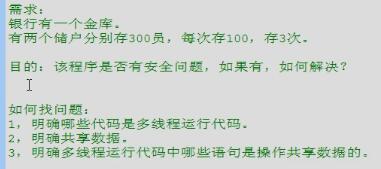


这么理解：这个对象相当于0/1标识符.当线程0读到synchronize(对象)时，先判断对象的状态，当对象为1时，线程0进入代码下一行，同时将对象置为0.这时如果cpu切换到其他线程，其他线程看到对象值为0，只能在外面等着。过了一会线程0重新那会执行权，执行完所有的同步代码后离开，并且将对象值重新置为1。

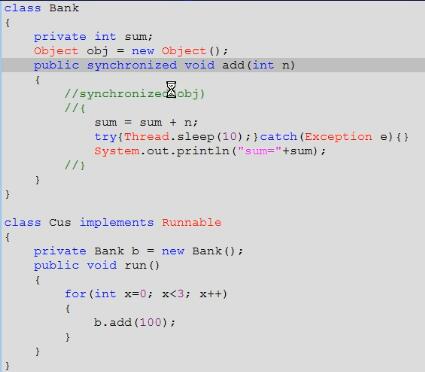
原理相当于很多人上同一个公共厕所。



**P140 多线程（多线程-同步函数）**



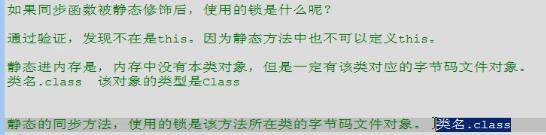
问题：有可能一个客户存完100后，打印之前切换到其他线程，其他线程也存入100.这时打印就会是200，200而不是100，200



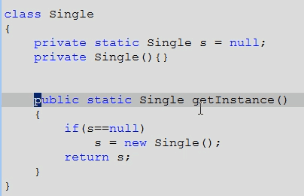
**P141 多线程（多线程-同步函数的锁是this）**

同步函数的锁是this

**P142 多线程（多线程-静态同步函数的锁是Class对象）**

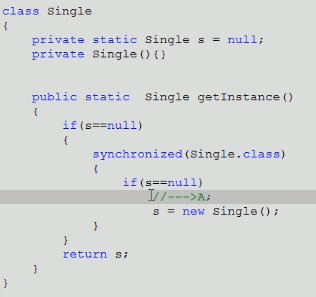


**P143 多线程（多线程-单例设计模式-懒汉式）**



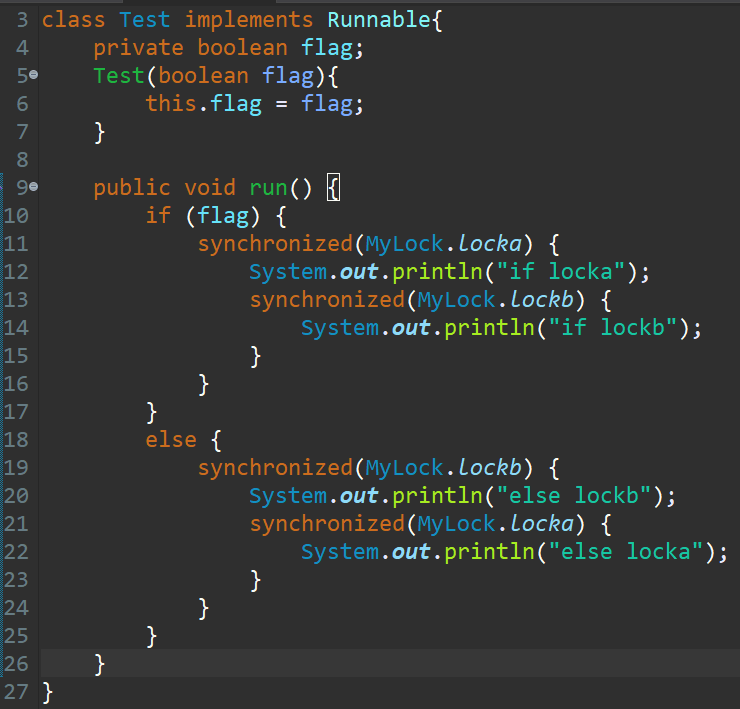
有多条语句在操作共享数据s，就可能new多个对象，造成安全隐患

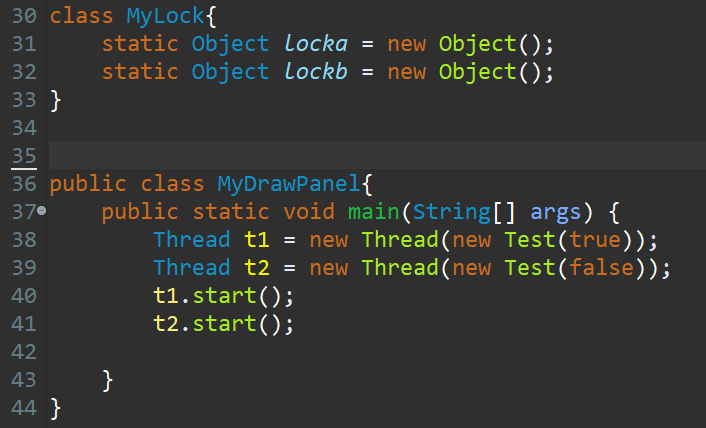
改进：



**P144 多线程（多线程-死锁）**

死锁的例子



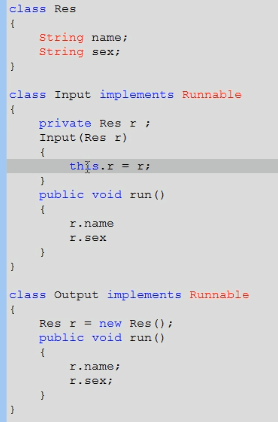


**P145 多线程（线程间通信-示例代码）**

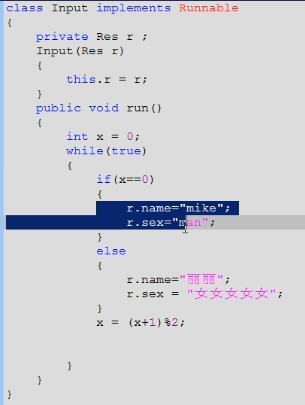


需求：

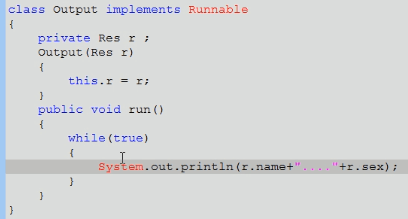
有一堆人物信息，包括姓名和性别，一个线程在存，一个线程在取。

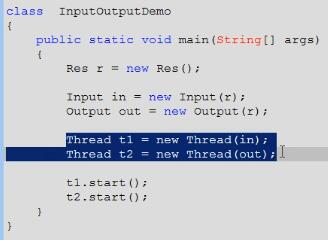


存入



取出

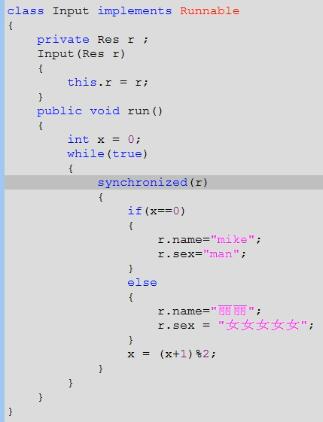


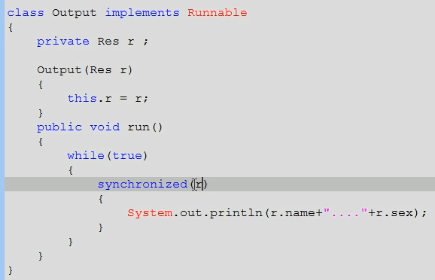


这时会出现一个问题：存入和取出不同步，即取出的姓名和性别不一定对应。

**P146 多线程（线程间通信-解决安全问题）**

在上一 P中问题的解决





这样，就给Input和Output上了同一把锁r。其实只要是同一把锁就可以，现成的同一把锁还有：Input.class;Output.class;Res.class;InputOutputDemo.class;

**P147 多线程（线程间通信-等待唤醒机制）**