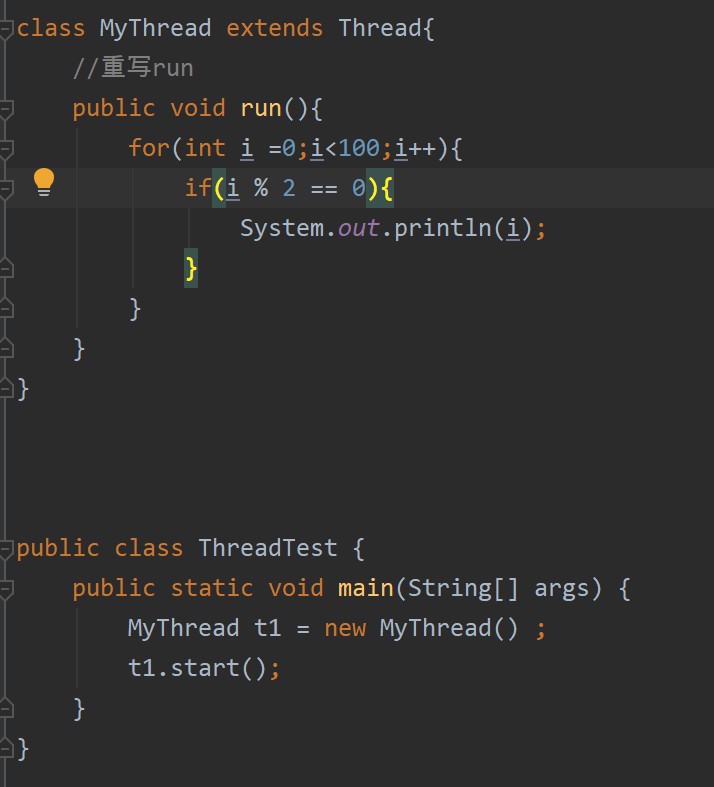
并行：多个CPU做多个事情

并发：一个CPU同时执行多项任务。

多线程的创建： 方式一：继承于Thread类

1. 创建一个继承于Thread类的子类
2. 重写Thread类的run()------------将此线程执行的操作声明在run()中
3. 创建Thread类的子类的对象
4. 通过对象调用start（）



我们不能通过直接调用run()的方式启动线程。

不可以还让已经start（）的线程去执行。

测试Thead中常用方法：

//1. start（）:启动当前线程：调用当前线程的run()

//2. run (): 通常需要重写Thread类中的此方法，将创建的线程要执行的操作声明在此方法中

//3. currentThread（）：静态方法，返回执行当前代码的线程

//4. getName(): 获取当前线程的名字

//5. setName（）： 设置当前线程的名字

//6. yield() : 释放当前CPU的执行权。  
 //7. join(): 在线程A中调用线程b的join()，此时线程a进入阻塞状态。直到线程b完全结束之后，线程a才再开始运行。

//8. stop（）：已过时，当执行此方法时，强制结束当前线程。

//9. sleep（long millitime）: 让当前线程“睡眠“指定的millitime毫秒，在指定的millitime毫秒时间内，当前线程是阻塞状态。

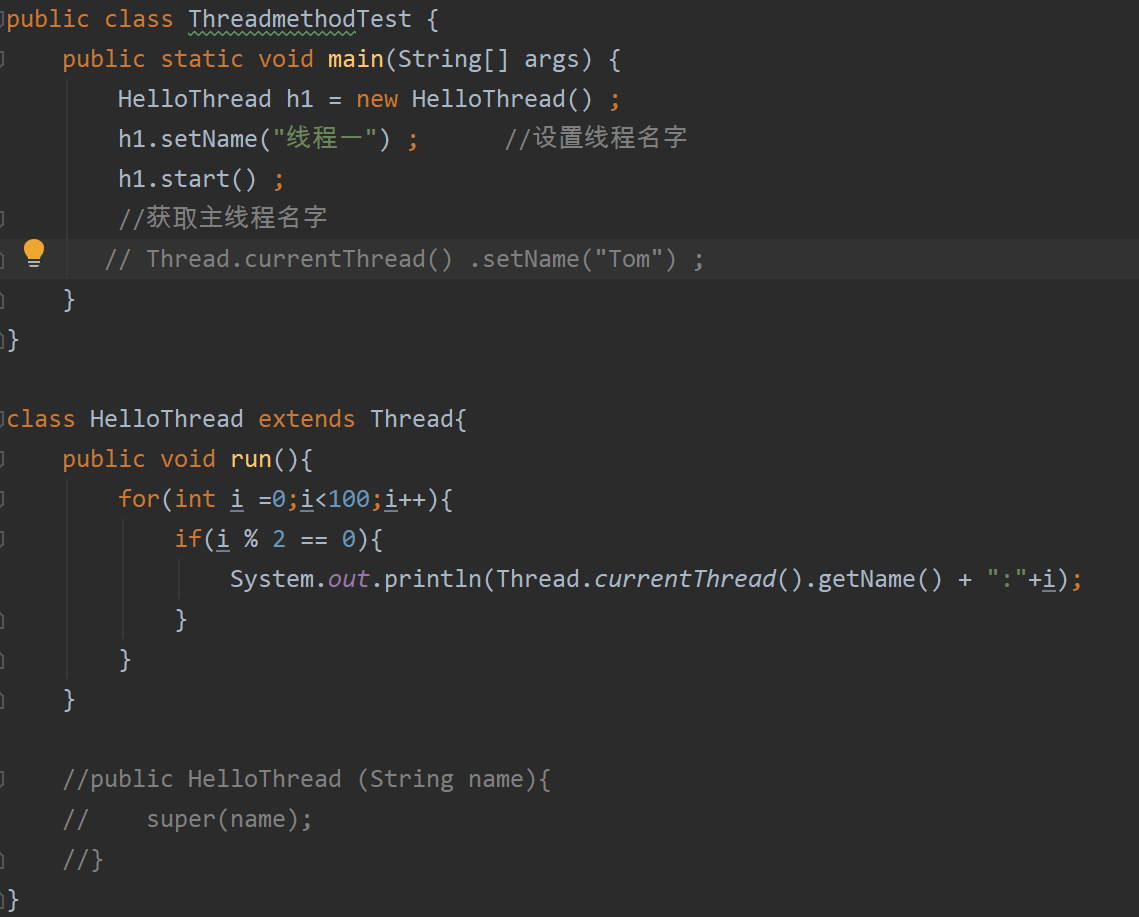
* //10. isAlive(): 判断当前线程是否存活。



结果显示：



//方法1：获取线程名字 “线程一”



方法二：获取“To

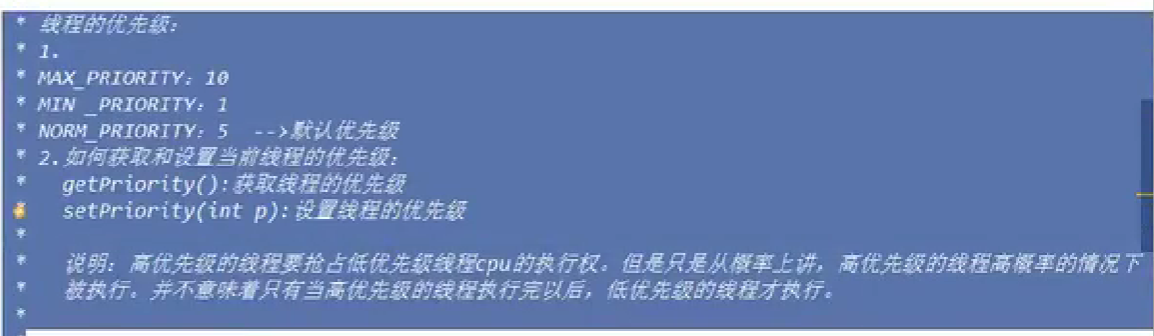


设置主线程名字 （在主线程运行）



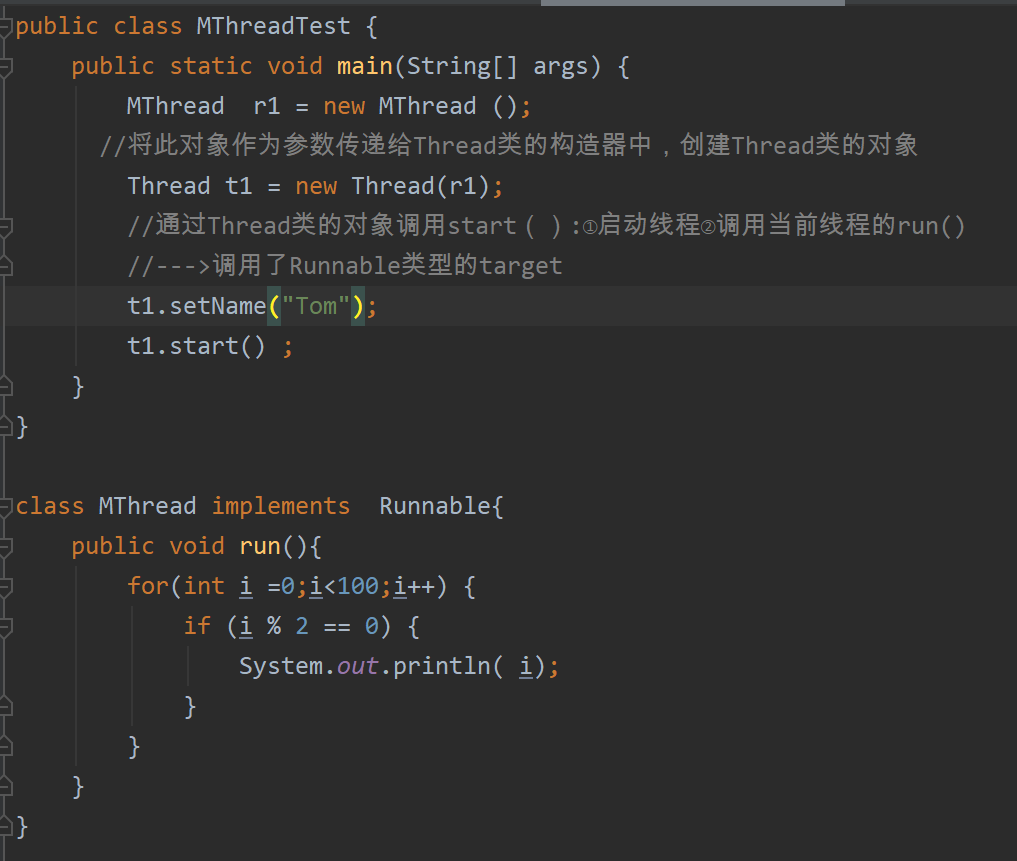
此时有两个线程：“线程一”与“Tom“显示结果顺序是随意的，属于并发式

//线程的优先级



创建多线程的方式二; 实现Runnable接口

1. 创建一个实现了Runnable接口的类
2. 实现类去实现Runnable中的抽象方法：run()
3. 创建实现类的对象
4. 将此对象作为参数传递给Thread类的构造器中，创建Thread类的对象
5. 通过Thread类的对象调用start（）



比较创建线程的两种方式：

开发中：优先选择：实现Runnable接口的方式

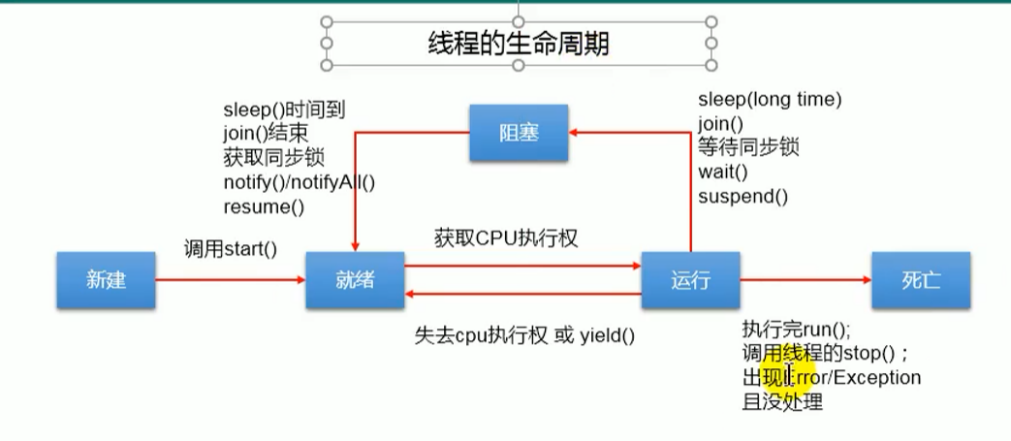
原因：1. 实现的方式没有类的单继承性的局限性

2. 实现的方式更适合来处理多个线程有共享数据的情况。

联系：public class Thread implementd Runnable

相同点：两种方式都需要重写run()，将线程要执行的逻辑声明在run（）中。

线程的生命周期



例子：创建三个窗口卖票，总数为100张，使用实现Runnable接口的方式

1. 问题：卖票过程中，出现了重票、错票 -🡪出现了线程安全问题
2. 问题出现的原因：当某个线程操作车票的过程中，尚未完成操作时，其他线程参与进来，也操作车票。
3. 如何解决：当一个线程a在操作ticket时，其他线程不能参与进来，直到线程a操作完ticket时，其他线程才可以操作ticket，这种情况即使线程a出现了阻塞，也不能改变。
4. 在Java中，我们通过同步机制，来解决线程的安全问题。

方式一：同步代码块



说明：操作共享数据的代码 ，即为需要被同步的代码

共享数据：多个线程共同操作的变量。比如：ticket就是共享数据。

同步监视器：俗称：锁。任何一个类的对象 ，都可以充当锁。

要求：多个线程必须要共用同一把锁。

补充：在实现Runnable接口创建多线程的方式中，我们可以考虑使用this充当同步监视器。

方式二：同步方法

如果操作共享数据的代码完整的声明在一个方法中，我们不妨将此方法声明为同步的。

关于同步方法的总结：

1. 同步方法仍然涉及到同步监视器，只是不需要我们显式的声明
2. 非静态的同步方法，同步监视器是：this
3. 静态的同步方法，同步监视器是：当前类本身
4. 同步的方式，解决了线程的安全问题。-----好处

操作同步代码时，只能有一个线程参与，其他线程等待。相当于是一个单线程的过程，效率低。------

线程死锁

