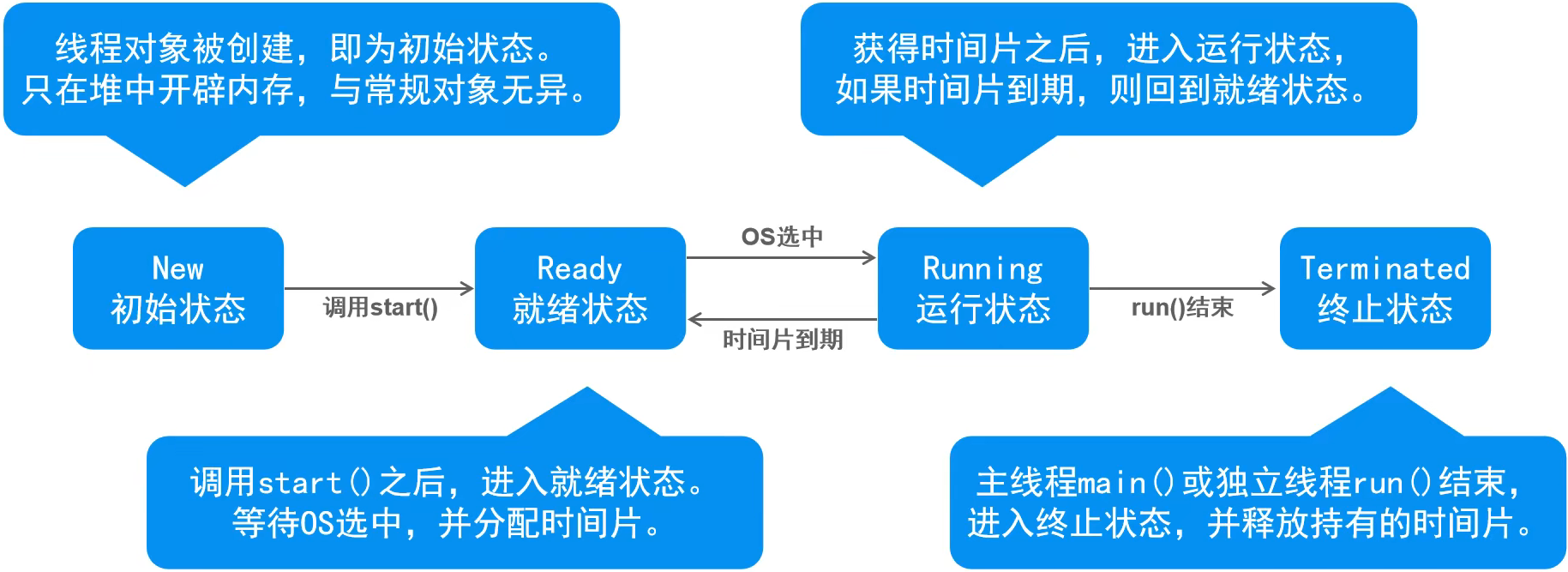
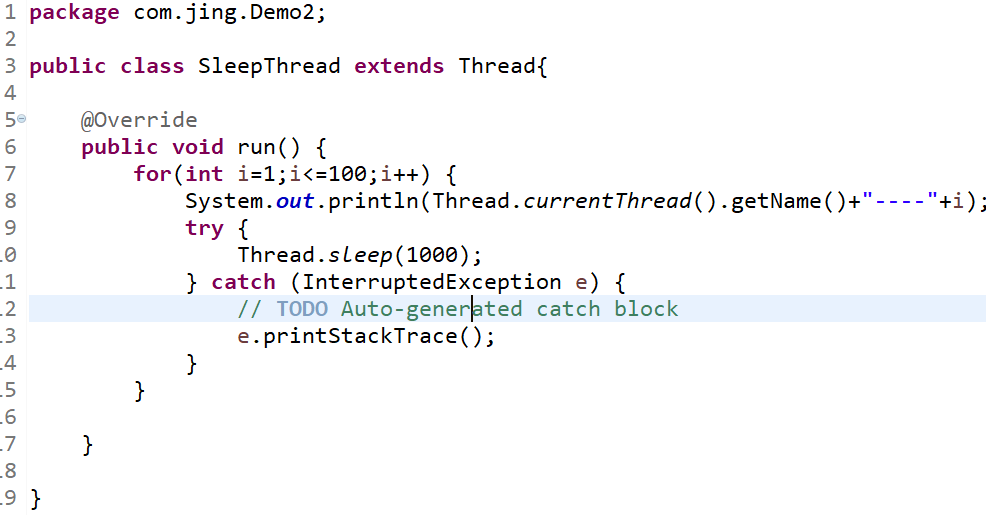
今日完成任务5

线程状态（基本）：

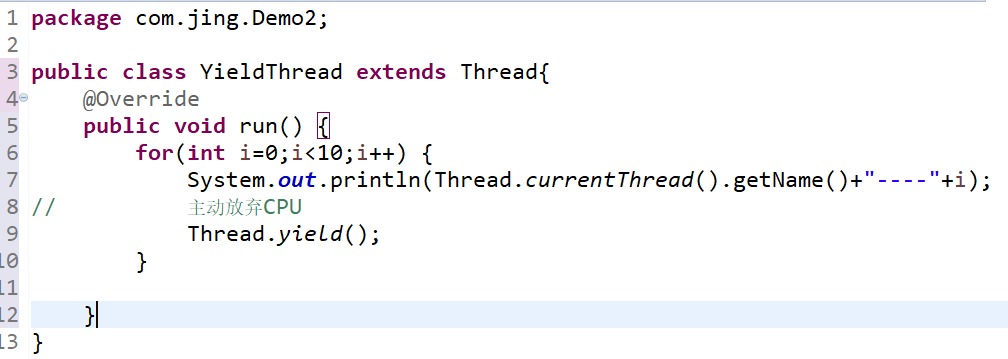


线程休眠

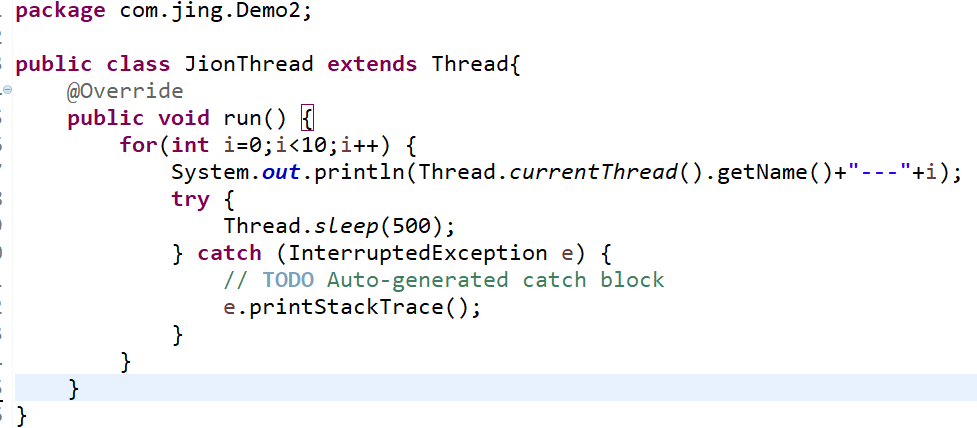
public static void sleep(long millis)//当前线程主动休眠milli毫秒

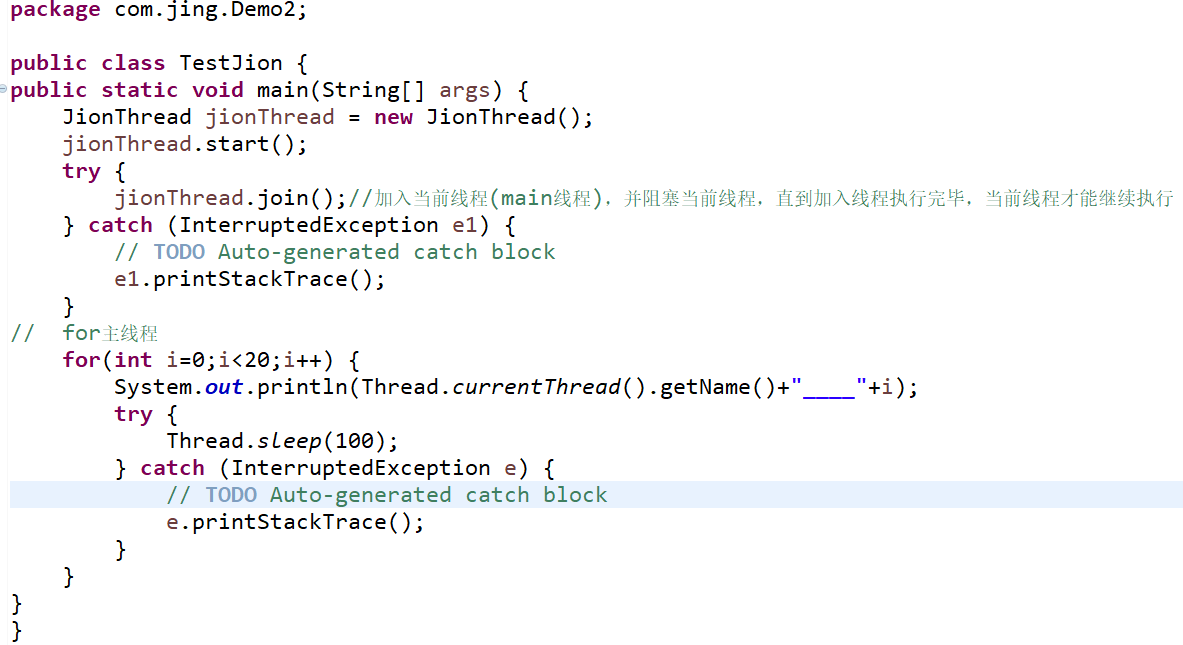


线程放弃：public static void yield()//当前线程主动放弃时间片，回到就绪状态，竞争下一次时间片。



线程加入：public final void jion()//允许其他线程加入到当前线程中







设置线程优先级（常用方法）

1. 优先级

线程对象.setPriority().

线程优先级为1-10，默认为5，优先级越高，表示获取CPU机会越多

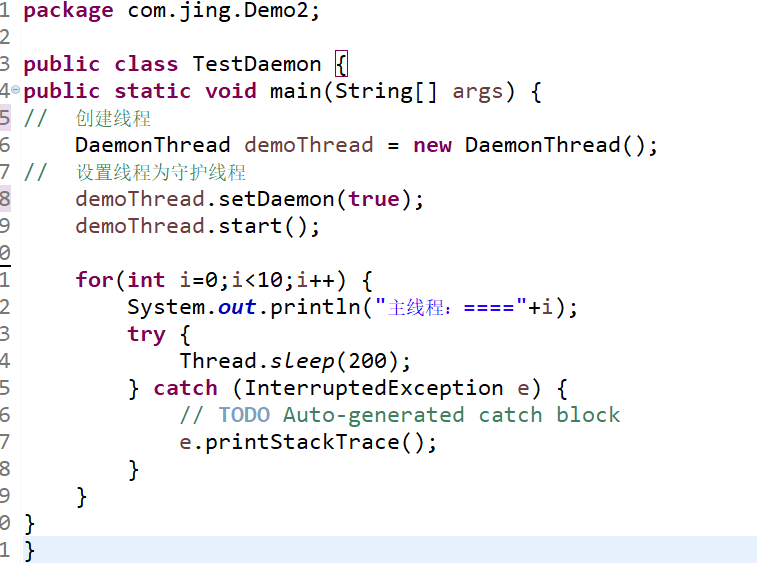
1. 守护线程

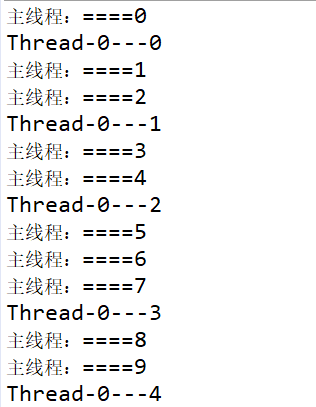
线程对象.setDaemon(true);设置为守护线程

线程有两类：用户线程(前台线程)，守护线程(后台线程)

如果程序中所有前台线程都执行完毕了，后台线程会自动结束

垃圾回收器线程属于守护线程





多线程安全问题：

多线程并发访问临界资源，如果破坏原子操作，可能造成数据的不一致

临界资源：共享资源(同一个对象)，一次仅允许一个线程使用才可保证其正确性。

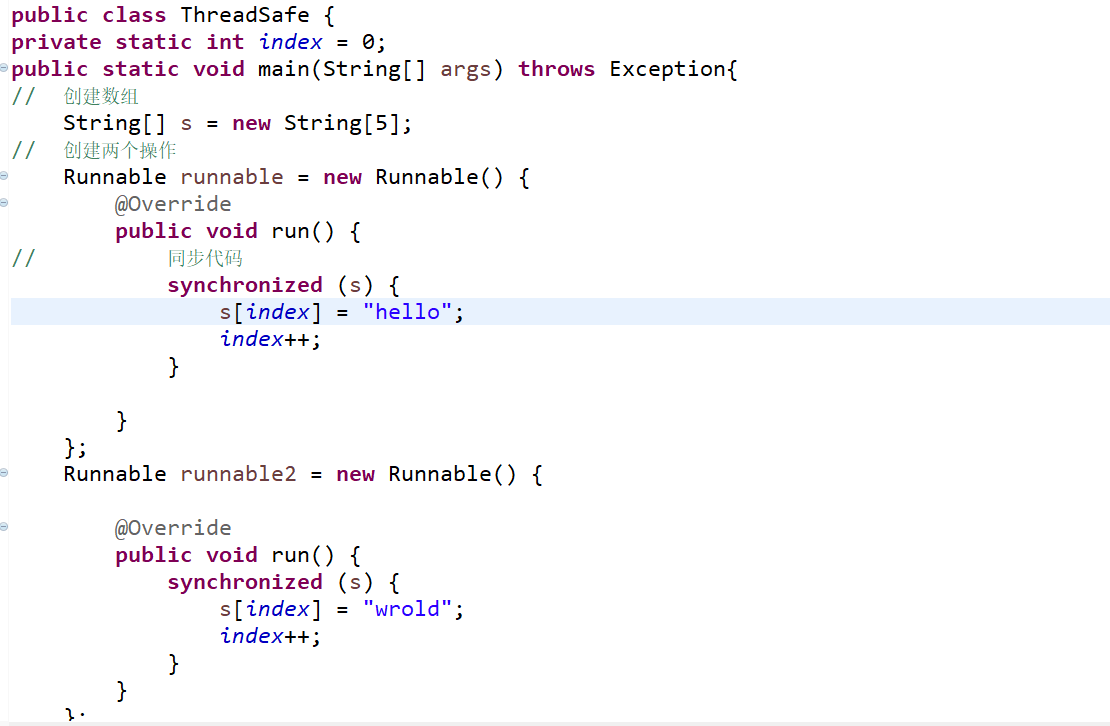
原子操作：不可分割的多步操作，被视作一个整体，顺序和步骤不可打乱或省略

同步代码块：synchronized(临界资源对象){//(代码（原子操作）)}

注：每个对象都有一个互斥锁标记，用来分配给线程的。

只有拥有对象互斥锁标记的线程，才能进入对该对象加锁的同步代码块。

线程退出同步代码块时，会释放相应的互斥锁标记。

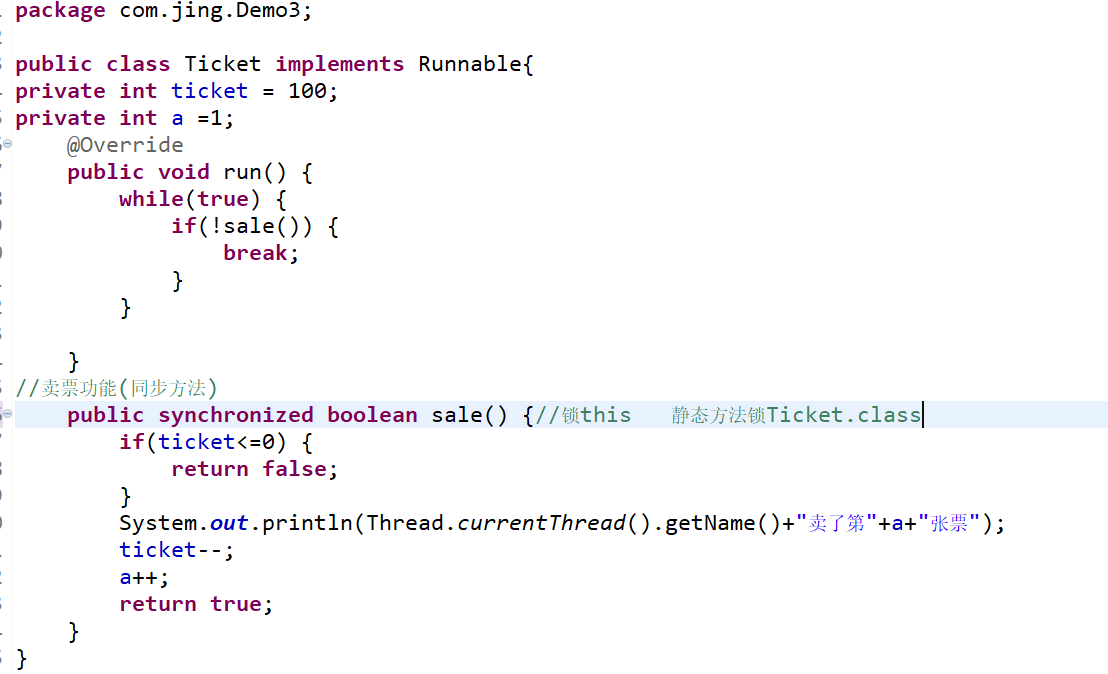


同步方法：synchronized 返回值类型 方法名称(形参列表0){

//对当前对象(this)加锁

//代码（原子操作）

}



同步规则：

注意：1.只有在调用包含同步代码块的方法，或者同步方法时，才需要对象的锁标记

2.如调用不包含同步代码块的方法，或者普通方法时，则不需要锁标记，可直接调用

已知JDK中线程安全的类

StringBuffer、Vector、Hashtable-以上类中的公开方法，均为synchonized修饰的同步方法

死锁：

1.当第一个线程拥有A对象锁标记，并等待B对象锁标记，同时第二的线程拥有B对象的锁标记，并等待A对象锁标记时，产生死锁。

2.一个线程可以同时拥有多个对象的锁标记，当线程阻塞时，不会释放已经拥有的锁标记，由此可能造成死锁

线程通信：

等待：public final void wait() public final void wait(long timeout)

必须在对obj加锁的同步代码块中。在一个线程中，调用obj.wait()时，此线程会释放其拥有的所有锁标记。同时此时阻塞在obj的等待队列中。释放锁，进入等待队列。

通知:public final void notify() public final void notifyAll()

