线程同步

1. 线程的概念及创建线程

一个进程中可以具有多个线程，每个线程执行不同的任务，而这些线程也可以像多个进程一样并发执行。能够同时运行两个或两个以上线程的程序叫做多线程程序。线程同进程一样也有不同的状态（runable，running，block）

1. 线程的创建及启动

两种方法创建线程，继承Thread类和实现Runable接口。

1）通过扩展Thread类创建线程

* 1. 创建线程类： class 新类名 extends Thread, 重写run方法
  2. 在主函数中，创建该类对象：Thread thread1 = new 新类名();
  3. 调用 thread1.start(); 启动线程

例：

public class MySimpleThread extends Thread {

public void run(){

for(int i=0; i<5; i++){

for(int j=0; j<8; j++){

System.out.print(getName()+"["+j+"] ");

}

System.out.println();

}

System.out.println("-----" + getName() + " ends-----");

}

public static void main(String [] args){

Thread thread1 = new MySimpleThread();

thread1.setName("T1");

Thread thread2 = new MySimpleThread();

thread2.setName("T2");

thread1.start();

thread2.start();

System.out.println("====="+Thread.currentThread().getName()+" ends=====");

}

}

2）通过实现Runnable接口创建线程

1. 创建线程类： class 新类名 implements Runnable，重写run方法
2. 在主函数中，创建该类对象：

Thread T1 = new Thread(new 新类名());

T1.start()

例：public class MySimpleRunnable implements Runnable {

public void run(){

for(int i=0; i<5; i++){

for(int j=0; j<8; j++){

System.out.print(Thread.currentThread().getName()+"["+j+"] ");

}

System.out.println();

}

System.out.println("-----" + Thread.currentThread().getName() + " ends-----");

}

public static void main(String [] args){

MySimpleRunnable mysr= new MySimpleRunnable ();

Thread T1 = new Thread(mysr);

Thread T2 = new Thread(new MySimpleRunnable());

T1.start();

T2.start();

System.out.println("====="+Thread.currentThread().getName()+" ends=====");

}

}

1. 线程同步与互斥

1）互斥资源的使用

在很多实际应用中，两个或两个以上的线程需要共享相同的数据和资源，这时，线程就需要考虑到其他线程的状态和行为，否则可能产生意想不到的结果。

要想解决 线程安全问题，必须得保证下面用于处理共享资源的代码在任何时刻只能有一个线程访问。 为了实现这种限制，Java中提供了同步机制。当多个线程使用同一个共享资源时，可以将处理共享资源的代码放置在一个代码块中，**使用synchronized关键字来修饰，被称作同步代码块，其语法格式如下：**

同步代码块可以有效解决线程的安全问题，当把共享资源的操作放在synchronized定义的区域内时，便为这些操作加了同步锁, lock是锁对象，同时只允许一个线程执行这些操作。

在方法前面同样可以使用synchronized关键字来修饰，被修饰的方法为同步方法。 被synchronized修饰的方法在某一时刻只允许一个线程访问，访问该方法的其它线程都会发生阻塞，直到当前线程访问完毕后，其它线程才有机会执行方法。

**修饰符 synchronized 返回值类型 方法名（形参列表）{**

**//同步代码**

**}**

同步方法也有锁，它的锁就是当前调用该方法的对象，也就是this指向的对象。

2） 线程的合作关系

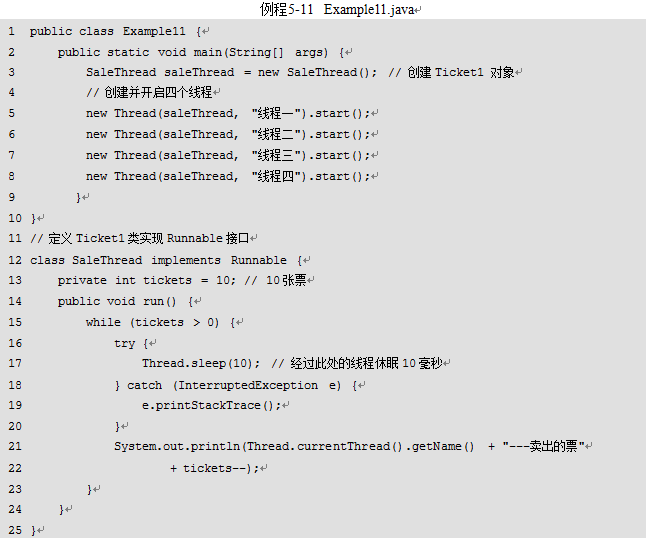
多个相互合作的线程彼此间需要交换数据，则必须保证各线程运行步调一致，如一个线程在没有得到与其合作的线程发来的信息之前，处于等待状态，一直到信息到来时才被唤醒继续执行。如果相互合作的程序配合得不好，程序运行结果将会产生问题。

Java提供了一个精心设计的线程间通信机制，使用wait()、notify()方法解决线程间合作的问题。这些方法是作为 Object 类中的 final 方法实现的。

* 当线程的执行条件不满足时通过调用this.wait()方法阻塞自己。wait方法是线程在获取对象锁后，主动释放对象锁，同时本线程休眠。直到有其它线程调用this.notify()唤醒该线程，才能继续获取对象锁，并继续执行。
* notify()就是唤醒操作，在当前操作能使某些线程不再阻塞，可以执行时调用，通常放在在线程完成的工作最后。但有一点需要注意的是notify()调用后，并不是马上就释放对象锁的，而是在相应的synchronized(){}语句块执行结束，自动释放锁后，JVM会在wait()对象锁的线程中随机选取一线程，赋予其对象锁，唤醒线程，继续执行。

这两个方法仅在 synchronized 方法或同步(synchronized)语句块中才能被调用，也就是要与synchronized(Obj)一起使用，即wait,与notify是针对已经获取了当前对象（this）锁进行操作。

1. 实验要求



1）上例是模拟订票系统，有四个线程模拟四个售票窗口进行售票，运行结果，分析结果是否正确，若不正确，原因是什么？ 并对程序进行修改（同步代码块和同步方法两种方式修改，实现互斥），使得其正确运行。

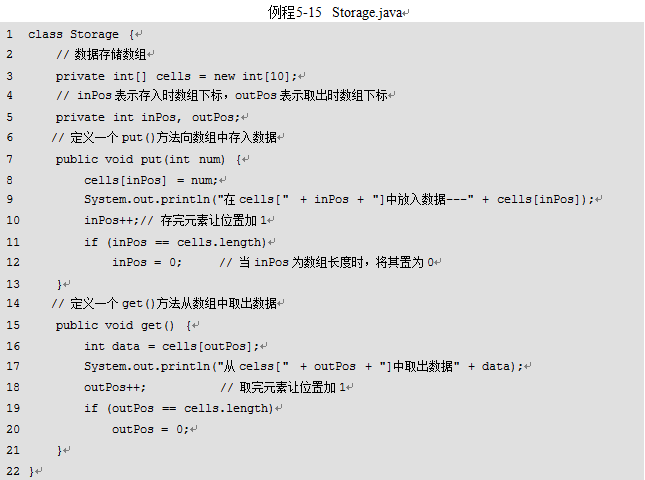
2）下例中两个线程同时去操作同一个存储空间，其中一个线程负责向存储空间中存入数据，另一个线程负责取出数据（即达到通信目的），即生产者与消费者问题。要求：

* 请参照Input类写出Output类程序。
* 编写程序测试这两个类（即写main函数），观察运行结果，放数据和读数据是否正确？
* 如果不正确，请修改程序，在适当的地方添加synchronized、this.wait()和 this.notify()进行同步。

synchronized、this.wait()和 this.notify()使用方法：

当存入数据时，如果count的值与cells数组的长度相同，说明数组已经添满，此时就需要调用同步锁的wait()方法（this.wait()）使存入数据的线程进入等待状态。同理，取出数据时如果count的值为0，说明数组已被取空，此时就需要调用同步锁的wait()方法，使取出数据的线程进入等待状态。

**每操作一次数据，便调用一次notify()方法唤醒对应同步锁上等待的线程**



接下来实现两个线程同时访问上例中的共享数据，这两个线程都需要实现Runnable接口。请参照Input类写出Output类。（此input线程程序向共享存储空间放置50个数，实验室也可以调整数量）

class Input implements Runnable{ // 输入线程类

private Storage st ;

private int num=1; // 向共享存储空间放置的数据num，也控制循环次数

Input(Storage st){ // 通过构造方法接收一个Storage对象，此对象与output共享

this.st = st;

}

public void run(){

while(num<50){

st.put(num++); // 将num存入数组，每次存入后num自增

}

}

}