需求：写一个学生类，使用一个学生对象让两个线程通信，一个输入数据的线程、一个输出数据的线程，当数据输入好后让该线程进入等待，通知让输出线程进行输出，输出完后通知输入线程输入，输出线程进入等待。

代码：

**package** xhu.edu.net.communication;

**public** **class** Student {

**private** String nameString;

**private** **boolean** flag = **false**;

**public** String getNameString() {

**return** nameString;

}

**public** **void** setNameString(String nameString) {

**this**.nameString = nameString;

}

**public** **boolean** isFlag() {

**return** flag;

}

**public** **void** setFlag(**boolean** flag) {

**this**.flag = flag;

}

}

**package** xhu.edu.net.communication;

**public** **class** InStudent **implements** Runnable {

**private** Student student;

String[] string1 = { "李湘", "大海", "少林", "久俺", "偶昂", "变量" };

**public** InStudent(Student student) {

**super**();

**this**.student = student;

}

@Override

**public** **void** run() {

**while** (**true**) {

**synchronized** (student) {

// 如果学生数据标识为假，就设置数据随机生成一个数据，并设置，然后唤醒输出线程，并让该线程进入等待

**if** (student.isFlag() == **false**) {

student.setNameString(string1[randomName()]);

System.***out***.println("in" + student.getNameString());

student.setFlag(**true**);

}

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e1) {

// **TODO** Auto-generated catch block

e1.printStackTrace();

}

student.notify();

**try** {

student.wait();

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

}

**public** **int** randomName() {

**int** ranNum = (**int**) (Math.*random*() \* 5);// 注意要先用括号把后面的括起来，不然强制转化的是前面的，就全是0

**return** ranNum;

}

}

**package** xhu.edu.net.communication;

/\*

\* 问题1：如果我们有多个生产线程，消费线程，而只有一个物品对象，

\* 那么，如果是消费线程陷入阻塞，我们进行唤醒的时候，是并不能指定到一定是生产线程抢到锁，

\* 那么这中间损失的效率应该怎么办？或者说如何来安排线程的数量和线程被唤醒的条件？

\*/

**public** **class** OutStudent **implements** Runnable {

**private** Student student;

**public** OutStudent(Student student) {

**super**();

**this**.student = student;

}

@Override

**public** **void** run() {

**while** (**true**) {

**synchronized** (student) {

// 如果学生数据设置完成，就输出，并且将标识设置为假

**if** (student.isFlag()) {

System.***out***.println("out" + student.getNameString());

student.setFlag(**false**);

}

/\*

\* 如果没有数据，或者数据已经被打印了，这时候就需要唤醒设置数据的线程， 让它准备好将去抢占锁对象的监听器，并且让当前线程进入等待状态，

\* 直到被通知有数据时才被唤醒 （notify（）方法仅仅只是通知，而且并不会立即释放当前线程的锁，会将当前线程代码执行完）

\* 而wait()方法会直接释放锁让当前获取到锁对象的监听器的线程进入阻塞状态。 所以我们这里可以选择先通知被阻塞线程，再让该线程进入阻塞状态。

\*/

student.notify();

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e1) {

// **TODO** Auto-generated catch block

e1.printStackTrace();

}

**try** {

student.wait();

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

}

}

**package** xhu.edu.net.communication;

//实现Runnable接口和继承Thread有所不同，前者可以通过一个实例来创建多个线程，后者通过多个实例来创建多个线程，

//所以前者不用Static便可实现变量共享

**public** **class** Test {

**public** **static** **void** main(String[] args) {

Student student = **new** Student();

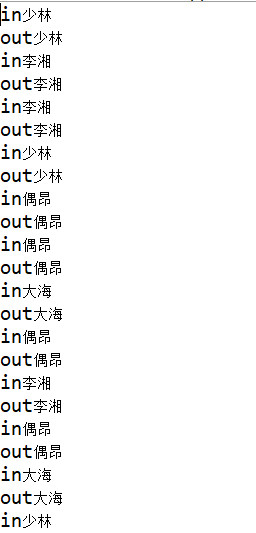
**new** Thread(**new** InStudent(student)).start();

**new** Thread(**new** OutStudent(student)).start();

}

}

运行截图：



总结：

关于等待阻塞与同步阻塞：

等待阻塞就是因为某个线程获取了某个对象的锁，而这个时候，该对象调用了wait方法，让这个线程进入阻塞状态这个状态是一般是主动进入，需要使用notify进行唤醒，当线程之间进行通信时，协作做事时，就会出现等待阻塞（可能某些条件需要其他线程进行给予或者某些事情需要其他线程完成）。等待阻塞需要用notify（）或者notifyall()进行唤醒。（有点自愿的意味在里面）

同步阻塞：是因为使用了synchronized,使得线程之间在抢占同一个资源时，只有抢占成功的线程才能使用，而抢占失败的线程就被迫进入阻塞状态。（有点被迫的意味）

关于notify与wait方法：

Wait方法是让获得该对象监听器的线程进入阻塞状态，notify方法是让调用了该对象的wait方法的线程中的某个线程被唤醒，唤醒只是叫它进入抢占资源的准备，而调用notify方法的线程继续执行。