实验内容

将实验6文件夹中所附的源程序“RWthread.java”改为用线程联合实现。

实现方法

1. 源程序分析

源程序采用同步线程的方式实现了读一个学生信息和写一个学生信息交替进行的操作。具体如下：

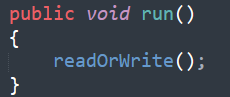
1. 两个线程的创建和启动：

在 RWthread 类的 main 方法中，创建了两个线程，分别代表读线程和写线程，并启动它们。这两个线程共享同一个 ReadWrite 类对象。



1. 读线程与写线程的交替执行：

Runnable接口的run方法中实现了readOrWrite()同步方法（synchronized）。

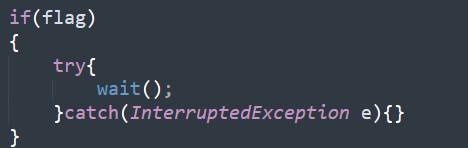




在实现Runnable接口的ReadWrite类中定义成员布尔变量flag（初始化为false，表示待读入信息）来标识此时是否有学生信息输入，以此来控制线程的切换。

读线程负责从用户输入读取学号和姓名，如果flag为true（已经读入了学生信息但是还没有写），使用 wait() 来释放锁，暂停自己的执行，把线程让给写线程，直到写线程发出通知，才能继续排队来抢线程。

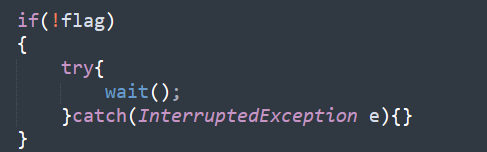




写线程当收到读线程的通知后，打印读线程读取的学号和姓名，设置flag为false，并通过 notify() 方法通知读线程可以继续执行。

此时线程还没有切换，写进程进入下一轮循环时，检测到flag为false，使用 wait() 来释放锁，暂停自己的执行，此时正在排队的读线程抢到线程继续工作。





1. 改用线程联合实现方法

线程联合是一种用于控制线程执行顺序的机制。当一个线程需要等待另一个线程执行完成后再继续执行时，可以使用线程联合。主要用到的方法是 join()。

join() 方法允许一个线程等待另一个线程的结束。调用该方法的线程会被阻塞，直到被调用 join() 的线程执行完成。如果线程 A 调用了线程 B 的 join() 方法，那么线程 A 将会被阻塞，直到线程 B 执行完毕，或者指定的时间到达。通常情况下，join() 方法用于主线程等待其他线程执行完成。例如，主线程需要等待所有子线程执行完毕才能结束。

根据上述的思路，要用线程联合来实现读一个学生信息和写一个学生信息交替进行的操作，需要一个主线程和一个联合线程。这里的思路是，用读线程作为主线程，在下一次读取之前，需要被打断，这里用写线程为联合进程，被打断的时候执行写线程的输出方法，执行完毕后，主线程继续读取下一个学生信息。

ReadWrite 类实现了 Runnable 接口，包含了两个成员变量 read（读线程）和 write（写线程）。



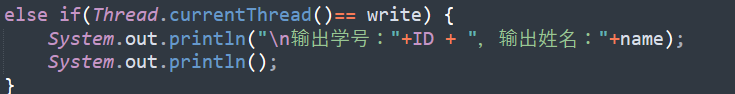
在 main() 方法中，创建了一个 ReadWrite 对象 rw，并启动了读线程 read，设定读线程为贯穿程序的主线程。

run() 方法中，通过判断当前线程是读线程还是写线程，来执行不同的逻辑。

若当前线程是读线程，通过一个循环不断获取用户输入的学号和姓名，并创建一个新的写线程来输出这些信息。每次循环结束后，使用 write.join() 等待写线程执行完成后再继续循环。



若当前线程是写线程，就打印读线程获取的学号和姓名。

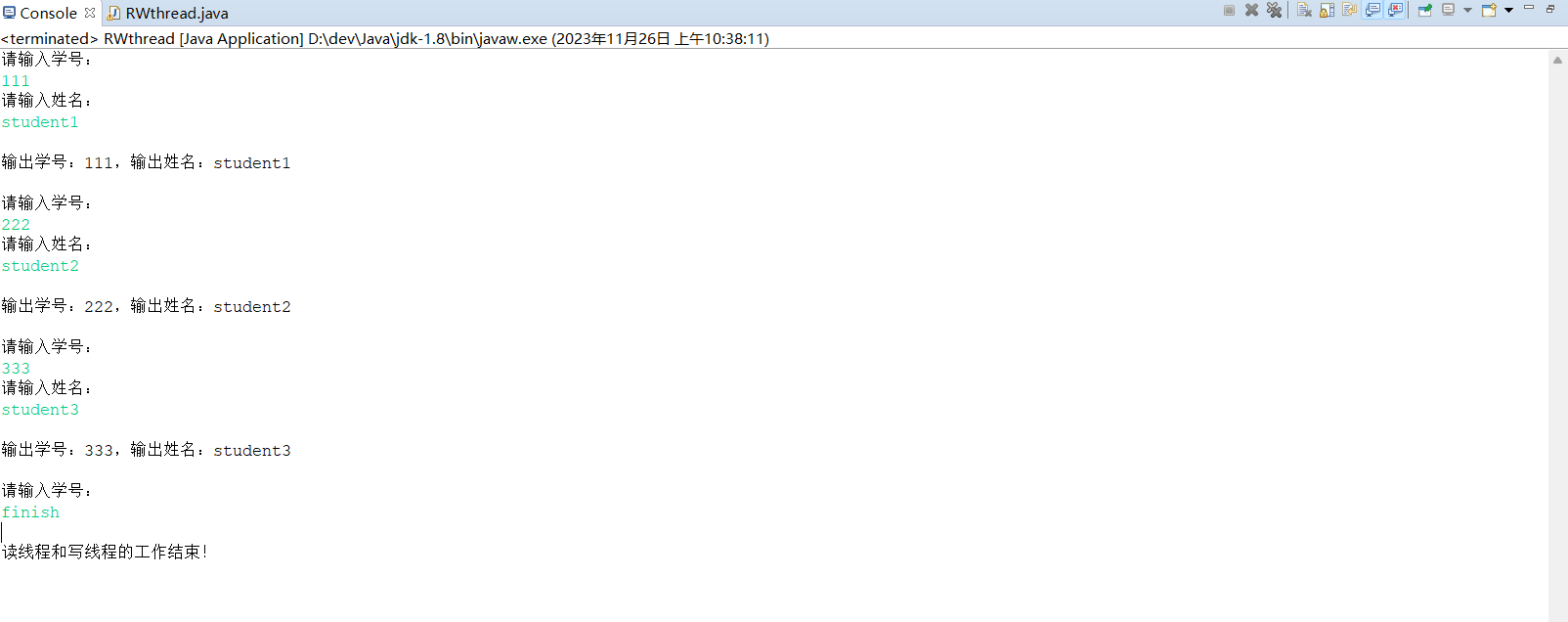


每次循环创建新的写线程的目的是确保写线程只完成一次打印操作，然后退出。这种设计可以让每个写线程只处理一次读线程获取的学号和姓名，然后在打印完成后自动结束线程生命周期。需要注意的是，每次都要创建一个新的写线程，因为联合线程要执行完所有任务后退出，当退出的时候该写线程被消灭，下次无法被调用。



实验结果

运行程序结果如图所示，输入一个学生的学号和姓名，输出该学生的学号和姓名，直到输入的学生学号为finish时完成读线程和写线程的任务，程序退出。



结论分析

在多线程编程中，同步线程和线程联合是两种用于控制线程执行顺序和数据共享的重要机制。通过这两种方式，可以确保多个线程之间的协调和安全执行。

同步线程使用 synchronized 关键字确保多个线程在访问共享资源时的安全性，避免数据竞争和冲突。使用 wait() 和 notify() 方法实现线程间的等待和唤醒，实现线程间的通信和同步。对于大量线程或频繁访问共享资源的情况，可能造成性能问题。

线程联合使用 join() 方法实现一个线程等待另一个线程执行完成后再继续执行。

主要用于控制线程的执行顺序，确保特定线程在其他线程执行完毕后再执行。提供了简单的线程顺序控制方式，适用于需要线程顺序执行的场景。