import java.util.concurrent.Semaphore;

/* Reader/Writer Lock
 * This class implements the classic many-readers, one-writer
 * blocking
 * semantics.
 */

public class RWLock {

    Semaphore read = new Semaphore(0);
    Semaphore write = new Semaphore(0);
    Semaphore lock = new Semaphore(1);
    int r = 0;
    int rw = 0;
    int w = 0;
    int ww = 0;

    /* Acquire a reader lock */
    public void rlock() throws InterruptedException {
        lock.acquire();
        if ((w+ww) != 0) {
            rw++;
            lock.release();
            read.acquire();
        } else {
            r++;
            lock.release();
        }
    }

    /* Release a reader lock */
    public void runlock() throws InterruptedException {
        lock.acquire();
        r--;
        if ((r == 0) && (ww > 0)) {
            write.release();
            w++;
            ww--;
        }
        lock.release();
    }
}
/* Acquire the only writer lock */
public void wlock() throws InterruptedException {
    lock.acquire();
    if ((w + r + ww) != 0) {
        ww++;
        lock.release();
        write.acquire();
    } else {
        w++;
        lock.release();
    }
}

/* Release the only writer lock */
public void wunlock() throws InterruptedException {
    lock.acquire();
    w--;
    if (ww>0) {
        write.release();
        w++;
        ww--;
    } else {
        while (rw>0) {
            read.release();
            r++;
            rw--;
        }
    }
    lock.release();
}