Come abbiamo visto, il meccanismo dei lock in Java è molto primitivo: quando un thread possiede il lock di un oggetto, nessun altro thread può accedere a quell'oggetto in modo sincronizzato.

In molti casi questo comportamento è inaccettabile: esistono oggetti cui devono poter accedere thread in lettura e in scrittura. Naturalmente, molti thread lettori devono poter accedere a un oggetto, ma un solo thread scrittore deve poterlo fare. Se uno scrittore sta accedendo, tutti gli altri rimangono in attesa.

Realizzeremo una classe, detta `RWLock` (sta per Read/Write Lock). Il comportamento di questa classe è il seguente:

- prima di accedere in lettura a una risorsa, un thread deve invocare il metodo `startReading()`; al termine della lettura, invocherà il metodo `stopReading()`;
- prima di accedere in lettura a una risorsa, un thread deve invocare il metodo `startWriting()`; al termine della lettura, invocherà il metodo `stopWriting()`;
- un numero arbitrario di thread possono essere in lettura; se però un thread è in scrittura, nessun altro thread può contemporaneamente essere in scrittura/lettura.

Cosa succede quando ci sono thread in lettura e uno scrittore decide di scrivere? Esistono vari approcci possibili. Quello che noi adotteremo è il seguente (detto *Writers Preferred*):

- prima di tutto lasciamo che i lettori attualmente in lettura terminino di leggere;
- eventuali lettori che si dovessero aggiungere, però, saranno messi in attesa;
- quando i lettori hanno finito di leggere, lo scrittore può accedere.

Manteniamo le seguenti variabili di istanza:

- `reqR` numero di lettori che hanno fatto richiesta, e che sono attivi o in attesa;
- `grantedR` numero di lettori attivi;
- `reqW` numero di scrittori che hanno fatto richiesta, e che sono attivi o in attesa;
- `grantedW` numero di scrittori attivi (0 o 1).

```
public synchronized void startReading()
throws InterruptedException {
    reqR++;
    try {
        while ( reqW != 0 ) wait();
    }
    catch( InterruptedException e ) {
        reqR--;
        throw e;
    }
    grantedR++;
}

public synchronized void stopReading() {
    grantedR--;
    reqR--;
    if ( grantedR + grantedW == 0 ) notifyAll();
}
```

```
public synchronized void startWriting()
throws InterruptedException {
    reqW++;
    try {
        while ( grantedR+grantedW != 0 ) wait();
    }
    catch( InterruptedException e ) {
        reqW--;
        throw e;
    }
    grantedW = 1;
}
public synchronized void stopWriting() {
    grantedW--;
    reqW--;
    if ( grantedR + grantedW == 0 ) notifyAll();
}
```

Questa implementazione funziona correttamente, ma assume che nessun lettore richieda il lock in scrittura prima di liberare il lock in lettura. Se ciò avvenisse, si verificherebbe un deadlock: uno scrittore pronto a scrivere deve aspettare che tutti i lettori liberino il lock!

Come evitare questo problema? Si tiene traccia di quale thread possiede attualmente il lock in scrittura: a quel thread vengono concessi liberamente lock in lettura e scrittura, senza farlo aspettare.