

Aufgabe 1 (15 Punkte)

Für den Leser/Schreiber-Ausschluss sollen geeignete Sperroperationen bereitgestellt werden, mit denen man eine Lese- bzw. Schreibsperre setzen und wieder aufheben kann, z.B. so:

```
lock1.Rlock();
... // read data protected by lock1
lock1.unlock();
lock2.Wlock();
... // modify data protected by lock2
lock2.unlock();
```

Für Objekte wie das hier verwendete `lock` ist eine Klasse `RWlockImpl` **implements** `RWlock` mit folgender Spezifikation zu entwickeln:

```
interface RWlock { // für Lese/Schreib-Sperrobjekte
void Rlock() throws LockErrorException;
    // setzt Lesesperre, sobald keine Schreibsperre gesetzt ist;
    // Ausnahmemeldung, wenn laufender Faden bereits eine Sperre gesetzt hat.
void Wlock() throws LockErrorException;
    // setzt Schreibsperre, sobald keine Lese/Schreibsperre gesetzt ist;
    // Ausnahmemeldung, wenn laufender Faden bereits eine Sperre gesetzt hat.
void unlock() throws LockErrorException;
    // löscht die vom laufenden Faden gesetzte Sperre;
    // Ausnahmemeldung, wenn laufender Faden keine Sperre gesetzt hat.
}
```

a) (3 Punkte) Implementieren Sie `RWlockImpl` unter Einsatz von Wachen (**WHEN**)! Welche Fairness-Eigenschaften hat Ihre Implementierung?

b) (5 Punkte) Implementieren Sie `RWlockImpl` unter Einsatz von Ereignisobjekten (**EVENT**) mit einer *signal-and-return*-Semantik! Welche Fairness-Eigenschaften hat Ihre Implementierung?

c) (7 Punkte) Implementieren Sie `RWlockImpl` in Java (mit `wait/notify/notifyAll`)! Welche Fairness-Eigenschaften hat Ihre Implementierung?

[Achtung: Lassen Sie sich nicht zu sehr von `java.util.concurrent.locks` beeinflussen! Die dortigen Methoden haben eine etwas andere Semantik.]

Aufgabe 2 (10 Punkte)

Vorgegeben ist die folgende, etwas eigenwillige Spezifikation von Multimengen, auf die wir im nächsten Aufgabenblatt zurückkommen werden:

```
interface Bag<E> {  
void add(E x);  
    // add x to bag  
E remove();  
    // remove and deliver arbitrary element from bag,  
    // as soon as bag is not empty (!)  
}
```

(Offensichtlich kann `remove` blockieren.)

- a) (6 P.) Geben Sie eine effiziente Implementierung an, die möglichst wenige und kurze kritische Abschnitte hat und `wait/notify/notifyAll` verwendet!
- b) (4 P.) Lösen Sie das Problem unter ausschließlicher Verwendung von Semaphoren der Klasse `java.util.concurrent.Semaphore`!