Dominik Gruntz
Daniel Kröni

Reparatur: CollectionTest

Sie haben festgestellt, dass die LinkedList nicht threadsafe programmiert ist. In dieser Aufgabe suchen Sie nach Lösungen, bzw. alternativen Datenstrukturen, die auch dann funktionieren wenn parallel darauf zugegriffen wird.

Aufgaben:

1. Verwenden Sie java.util.Collections.synchronizedList() um die LinkedList threadsafe zu machen. Funktioniert Ihre Lösung? Sind nun alle Elemente in der Liste?

2. Inspizieren Sie die Klasse, die Ihr Problem gelöst hat. Welches Objekt wird als Lock verwendet? Machen Sie eine kleine Skizze wie damit Threadsicherheit erreicht wird.
   Hinweis: LinkedList ist kein Subtyp von RandomAccess.

3. In Aufgabe 2 haben Sie gesehen, dass die Collections-Lösung zwar thread-safe ist, aber mittels Synchronisation tatsächlich alle Zugriffe serialisiert. Im Package java.util.concurrent finden Sie Collections, die nicht nur threadsafe, sondern performanceoptimiert für multi-threaded Zugriffe sind. Eine ConcurrentList gibt es nicht – verwenden Sie ConcurrentLinkedQueue und überprüfen Sie, ob der Test damit durchläuft.

4. (Bonus) Werfen Sie noch einen Blick auf das Interface java.util.concurrent.ConcurrentMap. Es leitet ab von java.util.Map und bietet zusätzliche atomare Operationen.

5. (Bonus) Inspizieren Sie den Code der Klasse ConcurrentLinkedQueue. Ziemlich abgefahren oder? CAS (compareAndSwap) werden wir im Unterricht noch kennenlernen.

Dominik Gruntz
Daniel Kröni

Weitere Locking Probleme

Im Unterricht haben Sie gesehen, wie mittels *synchronized* (und den entsprechenden Locks) der Zustand eines Objektes geschützt werden kann. Finden Sie die Probleme der folgenden Klassen mit der Annahme, dass alle public Methoden von mehreren Threads gleichzeitig zugegriffen werden können. Schlagen Sie Modifikationen vor um die Klassen threadsicher zu machen.

```
class LockProblem1 {
    private List<String> list = new LinkedList<String>();

    public synchronized void add(String s) {
        list.add(s);
    }

    public synchronized void remove(String s) {
        list.remove(s);
    }

    public String get() {
        if(!list.isEmpty()) {
            return list.get(0);
        } else {
            return null;
        }
    }
}
```

```
class LockProblem2 {
    private List<String> list = new ArrayList<String>();

    public synchronized void add(String s) {
        list.add(s);
    }

    public void addTwo(String a, String b) {
        synchronized (list) {
            list.add(a);
            list.add(b);
        }
    }
}
```