Übungen zu Verteilte Systeme

Ostbayerische Technische Hochschule Regensburg

04.11.2013, Übung 4

Universitätsstraße 31, 93053 Regensburg

Prof. Dr. Jan Dünnweber

Züge auf einer eingleisigen Bahnstrecke

Übung 1: Das Gleis kann zu jedem Zeitpunkt nur durch einen Zug belegt sein. Zusammenstöße werden mittels `wait` und `notify` vermieden

```
public class SingleTrack {
  private boolean isFree = true;
  public void enter () throws InterruptedException {
    synchronized (this) {
      while (! this.isFree)
        this.wait ();
      this.isFree = false;
    }
  }
  public void leave () throws InterruptedException {
    synchronized (this) {
      this.isFree = true;
      this.notifyAll ();
    }
  }
```

Jeder Zug wird durch einen Thread repräsentiert

Unser Modell für Züge:

```
public class Train extends Thread {

  private String label;
  private SingleTrack track;

  public Train (String label, SingleTrack track) {
    this.label = label;
    this.track = track;
  }

// ...
```

- Den vollständigen Code zu dieser Übung finden Sie auf:

  `http://www.dpunkt.de/buecher/3213/fortgeschrittene-programmierung-mit-java`

Die run-Methode

```
  public void run () {
    try {
      for (int i = 0; i < 2; i++) {
        System.out.println (this.label + "running");
        Thread.sleep (1000);  }
      System.out.println (this.label + "try to enter...");
      this.track.enter ( );
      System.out.println (this.label + "entered!");
      for (int i = 0; i < 3; i++) {
        System.out.println (this.label + "running on SingleTrack");
        Thread.sleep (1000);  }
      System.out.println (this.label + "exiting");
      this.track.leave ();
      for (int i = 0; i < 2; i++) {
        System.out.println (this.label + "running");
        Thread.sleep (1000);  }  }
    catch (InterruptedException e) {
      System.out.println (e);  }
  }
}
```

Experimente mit der Bahnstrecke

Eine Testanwendung könnte z. B. so aussehen:

```
public class Application {

  public static void main (String [] args) {
    SingleTrack track = new SingleTrack  ();
    Train t0 = new Train ("", track);
    Train t1 = new Train ("\t\t", track);
    t0.start ();
    try { Thread.sleep (500); } catch (InterruptedException ignored) { }
    t1.start ();
  }
}
```

- Könnte man auch `Locks` und `Conditions` anstelle von `wait` & `notify` verwenden?
- Finden Sie auch eine Lösung die Semaphore verwendet?

DeadlockDetectingLock

- **Übung 2:** Die Method `sumArray` bewirkt **Deadlocks**

```
public int sumArrays(int [ ] a1, int [ ] a2)  {
    int value = 0;
    int size = a1.length;
    if (size == a2.length) {
        synchronized(a1) {
            synchronized(a2) {
                for (int i = 0;  i < size;  ++i)
                    value += a1 [ i ] + a2 [ i ];
            }  }
    }
    return value;
}
```

- Implementieren Sie die Klasse `ArrayWithDeadlockDetectingLock` die Zugriffe mittels eines `DeadlockDetectingLock` koordiniert (→ `http://www.onjava.com/pub/a/onjava/2004/10/20/threads2.html?page=2`)
- Entfernen Sie die Rekursion aus dem `DeadlockDetectingLock` indem Sie den Lock-Tree per *Breitensuche* (mit einer *Queue*) anstelle der Tiefsuche durchlaufen. Sind Deadlocks noch immer ausgeschlossen?

Knack den Safe

Übung 3:

- Laden Sie sich den Java-Code der Klasse `Safe` aus dem Moodle
- Die Kombinationen werden in der Form `x-y-z` eingegeben (z. B. , `45-90` $\Rightarrow$ Drehung um 45° und 90°)
- Erweitern Sie das Programm, so dass sich der Safe mit der Kombination `04-11-13` öffnet
- Nach drei Fehlversuchen soll der Alarm ausgelöst werden

Die Methoen `open` und `alarm` sind bereits implementiert.

- Verwenden Sie ein `CountDownLatch`, um die Fehlversuche zu registrieren