

## 2. Übung „Nebenläufige und verteilte Programmierung“ Besprechung am 14. November

---

### Aufgabe 1

- a) Definieren Sie eine Klasse `Counter`, welche einen sich ständig inkrementierenden Zähler (beginnend bei 0) implementiert. Die Geschwindigkeit mit der der Zähler hochgezählt wird, soll durch einen Zeitparameter (in msec) des Konstruktors festgelegt werden. Diese Zeit soll der Zähler warten, bis er das nächste Mal hochzählt. Außerdem soll der Zähler bei seiner Konstruktion noch einen Namen erhalten. Bei jedem Hochzählen soll der Zähler seinen Namen und den aktuellen Wert auf die Standardausgabe schreiben.

Schreiben Sie ein Programm, mit welchem es möglich ist, beliebig viele Zähler nebenläufig zu starten. Außerdem soll beim Programmstart für jeden gestarteten Zähler die Geschwindigkeit angegeben werden. Drei unterschiedlich schnelle Zähler sollen beispielsweise gestartet werden durch:

```
java Zähler 3 300 500 1000
```

- b) Erweitern Sie Ihren Zähler um eine graphische Ausgabe, d.h. jeder Zähler soll in einem eigenen Fenster angezeigt werden.

### Aufgabe 2

Schreiben Sie ein Java-Programm, welches analog zur Vorlesung eine gemeinsam genutzte Variable nebenläufig inkrementiert und mit zwei multipliziert. Dies ist nicht mit einer Variablen vom Typ `int` möglich. Deshalb sollten Sie eine eigene Klasse definieren, welche Methoden zum Inkrementieren und Verdoppeln zur Verfügung stellt. Überlegen Sie, wie Sie das Ergebnis der nebenläufigen Berechnung nach deren Beendigung ausgeben können.

Modifizieren Sie die Läufe Ihres Programms mit Hilfe von `sleep`-Anweisungen so, dass alle in der Vorlesung diskutierten Ergebnisse ausgegeben werden.

Verwenden Sie `synchronized`, um eine atomare Ausführung des Inkrementierens und des Verdoppelns zu gewährleisten. Die Ausgabe 0 soll also nicht mehr möglich sein.

### Aufgabe 3

Schreiben Sie ein möglichst einfaches Java-Programm, das in einen Deadlock läuft.

#### Aufgabe 4

- a) Definieren Sie eine Klasse **Semaphore**, welche einen abstrakten Datentyp für Semaphore mit den Operationen P und V (und auch L für lookup) zur Verfügung stellt.
- b) Verwenden Sie die Klasse **Semaphore** zur Implementierung der dinierenden Philosophen mit Deadlockvermeidung durch Zurücklegen.