

### 3. Übung „Nebenläufige und verteilte Programmierung“ Abgabe am 21. November, Besprechung am 22. November

---

#### Aufgabe 1

In der Vorlesung haben wir eine Implementierung der Klasse `MVar` zur Kommunikation mittels Message Passing schrittweise entwickelt.

- a) In der letzten entwickelten Variante der `MVar` wurde gezielt nur noch ein relevanter Thread erweckt. Dieser überprüft aber dennoch ein weiteres Mal in der `while`-Schleife den Wert der Variablen `empty`. Ist dieser Test wirklich notwendig, oder können Sie wegen der Verwendung von `notify` anstelle von `notifyAll` einfach auch nur ein `if` verwenden? Begründen Sie Ihre Antwort ausführlich.
- b) Erweitern Sie die letzte in der Vorlesung vorgestellte `MVar`-Implementierung um folgende Methoden:
  - `boolean tryPut (T o)`, welche nicht blockiert, sondern `false` liefert, wenn ein `put` nicht möglich ist.
  - `void write (T o)`, welche nicht blockiert und den aktuellen Wert einfach überschreibt.
  - `T swap (T o)`, welche einen ggf. vorhandenen Wert ersetzt und den alten Wert als Ergebnis zurück liefert. Falls die `MVar` leer ist soll die Methode suspendieren.
  - Stellen Sie außerdem Varianten zur Verfügung, welche zusätzlich einen Timeout anbieten. Nach dieser Zeit soll die Suspension automatisch aufgehoben werden.
- c) Java stellt in der neuesten Version ähnliche Kommunikationsabstraktionen wie die `MVar` im Paket `java.util.concurrent` zur Verfügung. Suchen Sie eine Klasse die sich genau wie die von uns definierte `MVar` verwenden lässt. Vergleichen Sie auch die Implementierungen.
- d) Implementieren Sie eine Klasse `SyncVar`, welche analog zur `MVar` einen Null-elementigen Puffer zur Verfügung stellt. Da kein Wert gepuffert werden kann, entspricht eine Kommunikation über diese `SyncVar` einer synchronen oder Hand-Shake-Kommunikation. Diskutieren Sie die Vor- und Nachteile bei einer Verwendung beim Producer-Consumer-Problem, insbesondere im Vergleich zur `MVar`

## Aufgabe 2

Erweitern Sie den graphischen Zähler aus Übung 2, Aufgabe 1 um folgende Buttons:

- Stop: der Zähler bleibt beim aktuellen Wert stehen.
- Start: der Zähler zählt mit seiner vorgegebenen Geschwindigkeit weiter.
- Close: die Darstellung des Zählers wird geschlossen.
- Copy: eine Kopie des Zählers wird angelegt, welche sich zunächst identisch wie das Original verhält. Alle weiteren Aktionen des kopierten bzw. des Originalzählers haben allerdings keine Auswirkungen auf den jeweils anderen Zähler.
- Clone: zunächst geschieht das gleiche wie bei Copy, allerdings verändern alle Aktionen des Klons bzw. des Originals auch das Original bzw. den Klon. Ausnahmen sollen der Close-Button und der Copy-Button sein: nur der Klon bzw. das Original wird beendet bzw. kopiert.

Beachten Sie, dass die Zähler auch mehrfach (und auch rekursiv) geklont werden können. Alle Klone sollen sich identisch verhalten.

Achten Sie auch darauf, dass Ihr Programm terminiert, falls kein Zähler mehr dargestellt wird.