# Übungszettel 2

Allgemein:
Abgabe via OLAT Upload bis Montag vor der nächsten Übung um 08:00! Für dieses Übungsblatt Abgabe bis 23.11.2014!
Upload muss aus einem File mit Beschreibung der Lösung (pdf oder txt) **und** einem weiteren Archiv das Source-Code enthält (zip oder tar.gz, Eclipse Project exportieren!) bestehen. Punktabzug bei falschem Dateiformat oder mehr oder weniger als 2 Files (kein docx, tex, rar, 7z, Ohne Endung, …) bestehen! Zwei gültige Files z.b. name-u1.txt und name-u1.zip

## 1. Producer / Consumer Advanced: (42%)

Erweitern Sie Aufgabe 4 von Blatt 1 um folgende Funktionalität:

- Mehrere Producer (4) und ein Consumer sollen gleichzeitig gestartet werden. Jeder Producer verfügt über einen eigenen Buffer und der Consumer wartet bis von allen Producern eine Zahl abgeholt werden kann.
- Wenn ein Producer eine 0 liefert wird von diesem Keine weitere Zahl mehr Abgeholt.
- Der Consumer soll nur dann die Buffer abfragen wenn diese Zahlen enthalten und nicht wie in 1) alle 2 Sekunden.
- Erläutern Sie, wie sichergestellt wird, das der Consumer nur dann konsumiert wenn von allen Producern gleichzeitig eine Zahl bereitsteht.
- Wie verhält sich Ihr System wenn es einen weiteren Consumer gibt der nur von der Hälfte der Producer Zahlen abholt? Ist Ihre Lösung Fair (Kann Consumer 1 “verhungern”)? Erläutern Sie mögliche Locking Mechanismen!

## 2. Message Passing: (33%)

Threads sollen über Message Passing kommunizieren. Das File “ex2.java” enthält eine unvollständige Implementierung der Send und Recive Operationen. Die Sendefunktion soll hierbei den weiteren Programmablauf des Senders nicht Blockieren (Asynchron)! Vervollständigen Sie das gegebene Message Passing Programm und implementieren Sie die benötigte Send und Recive Funktion um die Funktion des Programmes zu gewährleisten. Des weiteren ist an einer stelle ein Thread.sleep() durch einen passenderen Synchronisations-Mechanismus zu ersetzen.

# 3.Thread safety: (25%)

Implementieren Sie mithilfe folgendem Codebeispiels ein Szenario in dem 3 Threads ein Objekt via der **getInstance()** Funktion der Klasse LazyInitRace bekommen:

http://jcip.net.s3-website-us-east-1.amazonaws.com/listings/LazyInitRace.java

```
public class LazyInitRaceCondition {
   private ExpensiveObject instance = null;

   public ExpensiveObject getInstance() {
      if (instance == null)
         instance = new ExpensiveObject();
      return instance;
   }
}

class ExpensiveObject { //TODO some memory and time intensive execution here}
```

a) Erklären und implementieren Sie ein Szenario in dem die drei Threads unterschiedliche Objekte der Klasse ExpensiveObject bekommen (ExpensiveObject beötigt einen aufwendigen Konstruktor oder das richtige Timing) bei gleichzeitigem erstmaligem Aufruf der **getInstance()** Funktion.
b) Wie ist es möglich, das Object der Klasse ExpensiveObject erst bei Bedarf anzulegen und trotzdem ThreadSafe zu sein (Schildern Sie mindestens 2 mögliche Ansätze)? Implementieren Sie einen Ihrer Lösungsansätze und testen Sie ihn, mit dem vorhergehenden Testszenario von a) auf Funktionalität.