Übungsblatt 2
Java-Concurrency-Framework

Aufgabe 1: Leberkässemmeln (Locks und Conditions)

Der Verkauf von Leberkässemmeln in der Cafeteria soll mit Hilfe von Threads simuliert werden. Studenten, die ständig Hunger haben, können an einer Theke einzeln Leberkässemmeln abholen und verspeisen. Kellner sorgen für ständigen Nachschub. An der Theke kann aber gleichzeitig immer nur eine bestimmte Anzahl an Leberkässemmeln liegen.

Implementieren Sie eine Klasse `KitchenCounter` sowie die aktiven Klassen `Waiter` und `Student`. Die Kapazität des `KitchenCounter` wird über dessen Konstruktor festgelegt. Das Ablegen (durch die Kellner) und Wegnehmen (durch die Studenten) einzelner Leberkässemmeln soll durch die parameterlosen Methoden `put()` und `take()` stattfinden. (Leberkässemmeln werden durch einen `int`-Counter im `KitchenCounter` repräsentiert.)

Verwenden Sie hierzu die Lock- und Condition-Klassen aus dem Java-Concurrency-Framework. Die Methoden `put()` und `take()` müssen „thread safe“ implementiert werden. Für die Zustände „volle Theke“ und „leere Theke“ sind unterschiedliche Conditions zu verwenden.

Testen Sie Ihre Simulation mit acht `Student`-Objekten, zwei `Waiter`-Objekten und einem `KitchenCounter` mit einer Kapazität von vier Leberkässemmeln:

```
public static void main(String[] args) {
      KitchenCounter theke = new KitchenCounter(4);
      new Waiter(theke, "Kellner-1").start();
      new Waiter(theke, "Kellner-2").start();
      for(int i=1; i<=8; i++)
            new Student(theke, "Student-"+i).start();
}
```

Erweitern Sie die Methoden jeweils um geeignete Debug-Ausgaben, um auf der Konsole den jeweiligen Stand Zustand verfolgen zu können.

Aufgabe 2: Downloads (Synchronisations-Hilfsklassen)

Für einen Vergleichstest von Downloadgeschwindigkeiten soll ein Browser entwickelt werden, dessen Downloadmanager eine Anzahl von Downloads durchführt und deren aktuellen Fortschritt in einem Browserfenster zum Vergleich anzeigt.

Für einen Download soll eine aktive Klasse `Download` entwickelt werden. Diese soll einen Download simulieren, indem 100-mal eine zufällige, je Thread unterschiedliche Anzahl an Millisekunden geschlafen wird. Bei jedem der 100-mal „Aufwachen“ soll der zugehörige Fortschrittsbalken im Browser durch den Thread verändert werden (vgl. Abbildung 2).

Die Klasse Browser ist soweit in GRIPS vorgegeben und enthält alle GUI-Elemente. Diese Klasse muss um Synchronisationsmechanismen ergänzt werden:

a) Alle Download-Threads, die bereits durch ihre Methode `start()` gestartet wurden aber durch ein Synchronisationsobjekt blockiert sind, müssen auf den Klick auf den Button „Downloads starten“ warten und anschließend Downloads simulieren (siehe Abbildung 1)

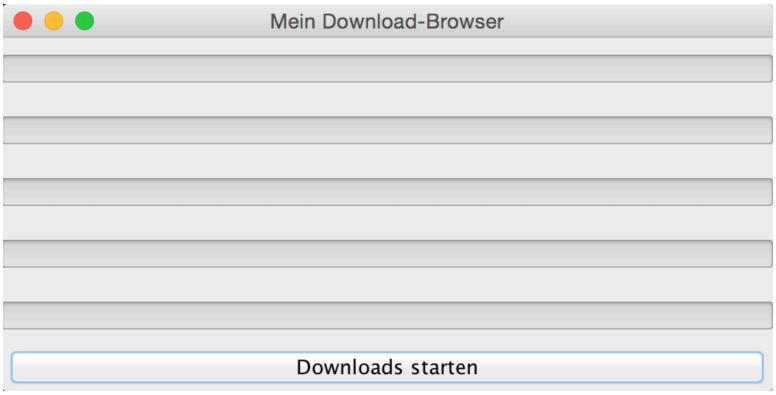

Abbildung 1: Broswerfenster vor Start der Download-Threads

b) Nach Start der `Download`-Threads ändert sich die Beschriftung und das Verhalten des Buttons (Beschriftung „Downloads laufen...“ und Deaktivierung; siehe Abbildung 2)

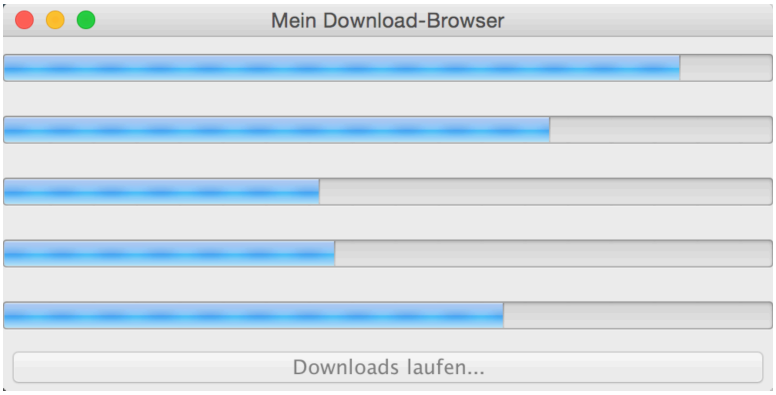

Abbildung 2: Browserfenster während der aktiven Downloads

c) Sobald der letzte Download abgeschlossen ist, ändert sich die Beschriftung des Buttons erneut zu „ENDE“ (siehe Abbildung 3)

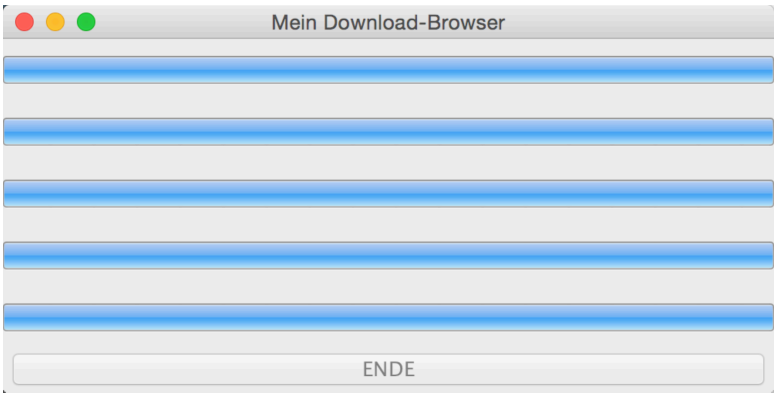

Abbildung 3: Browserfenster nachdem alle Download-Threads abgeschlossen sind

Wählen Sie aus den in der Vorlesung vorgestellten Synchronisationsklassen eine oder mehrere Möglichkeiten aus, um das gewünschte Verhalten zu implementieren.

Die Synchronisationsklassen werden in der Klasse `Browser` initialisiert und in beiden Klassen, `Browser` sowie `Download`, verwendet.

Zusatz-Aufgabe 3: Dining Philosophers

Versuchen Sie das aus der Vorlesung bekannte „Dining Philosophers"-Problem mit Hilfe von Locks und Conditions „thread safe" und verklemmungsfrei zu implementieren.

Tipps:

- Implementieren Sie eine aktive Klasse `Philosoph`, mit Methoden `essen()` und `denken()`
  - `essen()` bedeutet, es wird versucht, die beiden Gabeln links und rechts zu nehmen und für eine zufällige Anzahl an Sekunden zu essen (`sleep`)
  - `denken()` bedeutet, dass der `Philosoph`-Thread eine zufällige Anzahl an Sekunden schläft (`sleep`)
- Jeder Philosoph will endlos denken, essen, denken, essen, denken, essen, ...
- Implementieren Sie eine Klasse `Tisch` mit N Philosophen, die
  - 0 .. N-1 `Philosoph`-Objekte erzeugt und startet
  - die Gabeln als ein `boolean`-Array darstellt (`true` = Gabel in Benutzung, `false` = Gabel frei)
- Jeder Philosoph hat eine Nummer (0 .. N-1); seine linke Gabel hat die selbe Nummer, die Rechte + 1 bzw. 0

Zusatz-Aufgabe 4: Deadlock-Detection

Um Verklemmungen („Deadlocks") zur Laufzeit feststellen zu können, haben OAKS UND WONG zu Testzwecken als Erweiterung des `ReentrantLock` die Klasse `DeadlockDetectingLock` implementiert (`http://www.onjava.com/pub/a/onjava/2004/10/20/threads2.html?page=2`, verfügbar in GRIPS).

Machen Sie sich mit den Grundsätzen von „Deadlocks" vertraut (was sie sind, wie sie entstehen) und analysieren Sie die Klasse `DeadlockDetectingLock` (insbesondere dessen Methode `canThreadWaitOnLock`). Führen Sie die Klasse `TestDeadlockDetectionLock` aus

und versuchen Sie, die Deadlock-Situation in der Methode `sumArrays` sowie die Vorgänge im `DeadlockDetectionLock` zu verstehen und zu erklären.

Ersetzen Sie die Reentrant-Locks aus Aufgabe 4 durch `DeadlockDetectingLock`-Objekte und testen Sie Ihre Implementierung des „Dining Philosophers“-Problems auf Verklemmungen.