Übung 6: Synchronisation von Java-Threads

Ziel der Übung:

Diese Übung dient dazu, den Umgang mit der Synchronisationsprimitive „synchronized“ bei nebenläufigen Threads in der Programmiersprache Java kennenzulernen. Ein einfaches Java-Programm, in dem drei Threads nebenläufig einen einfachen Zähler hochzählen, soll zum Ablauf gebracht werden. Dabei erfolgt das Hochzählen zuerst ohne Synchronisation und anschließend durch Markierung des kritischen Abschnitts mit „synchronized“.

Aufgabenstellung:

Importieren Sie zunächst die zur Übung mitgelieferten Source-Files über die Eclipse-Import-Funktion (siehe *Übung 05, Seite 1*). In dem Programm sind vier Java-Klassen ausprogrammiert. Die erste Klasse namens *CounterObject* stellt ein einfaches Zählerobjekt dar. Die Klassen *countThread1* und *countThread2* dienen dem unsynchronisierten bzw. synchronisierten Hochzählen des Zählers und die Klasse *countTest* führt den gesamten Versuch durch.

1. Versuchen Sie zunächst die vier Java-Klassen zu verstehen und überprüfen Sie, ob sie richtig übersetzt werden. Es müssen in Eclipse vier *.class*-Dateien erzeugt werden.

   Was macht das Programm genau?

2. Bringen Sie das Programm zum Ablauf. Hierzu müssen Sie den Eclipse-Menüpunkt „Run“ benutzen.

   Prüfen Sie die Ausgaben des Programms auf der Konsole!

   Was wird ausgegeben? Welche Summen ergeben sich mit und ohne Synchronisation?

   Wie weichen die Ergebnisse voneinander ab?

   Ändern Sie das Programm so, dass es möglich ist, die Threadanzahl über eine lokale Variable festzulegen! (*Hinweis*: Es werden ein Thread-Array und einige Schleifen benötigt)

   Exkurs: Erzeugen und Nutzen von Arrays: Ein Array von einem beliebigen Typ x kann mittels der Java-Anweisung

   ```
   x[] <name> = new x[<size>];
   ```

   erzeugt werden. Der Name (name) des Arrays sowie die Größe (size) sind frei wählbar. Die Größe muss jedoch einer natürlichen Ganzzahl entsprechen. Der Zugriff auf eine Position in diesem Array ist folgendermaßen möglich:

   ```
   <name>[i] = …;
   ```

   Hierbei ist zu beachten, dass die erste Position im Array den Wert 0 hat. Zum Iterieren über jede Array-Position eignet sich z.B. die for-Schleife:

   ```
   for(int i=0; i < <array>.length; i++) {…}
   ```

3. Erhöhen Sie nun die Threadanzahl auf 10 und lassen Sie das Programm 10 mal ablaufen. Notieren Sie sich dabei die verschiedenen Ergebnisse! Worin unterscheiden sie sich und warum ist das so?

4. Versuchen Sie das Programm so zu modifizieren, dass die Dauer vom Starten der Threads bis zu deren Beendigung ausgegeben wird. Lassen Sie sich nun die Dauer für den Vorgang der Threads mit Synchronisation und ohne Synchronisation ausgeben. Was fällt Ihnen dabei auf?

   Hinweis: Die Methode `System.currentTimeMillis()` könnte in diesem Zusammenhang hilfreich sein.

5. Erhöhen Sie die künstliche Verzögerung in der *set*-Methode der Klasse *CounterObject* und übersetzen Sie das Programm erneut. Starten Sie dann das neu übersetzte Programm und betrachten Sie parallel dazu den Windows-Taskmanager.

   Hinweis: Mithilfe des Process Explorer (siehe Übung 2) sind die Informationen deutlicher und besser zu erkennen.

   Suchen Sie im Taskmanager (oder im Process Explorer) den Java-Prozess, in dem das Programm abläuft (Name = *javaw.exe*) und stellen Sie die Anzahl der Threads während des Ablaufs fest. Hierzu müssen Sie den Taskmanager entsprechend einstellen (Spalte *Threadanzahl* ergänzen über *Ansicht->Spalten auswählen*).

6. Wie viele Threads benutzt der Prozess und wie verändert sich die Anzahl der Threads zur Laufzeit?

7. Welche Priorität haben die Threads?
   Hinweis: es gibt eine Priorität auf Betriebsystem-Ebene und eine Priorität auf JVM-Ebene)

8. Wie viel CPU-Zeit und wie viel Hauptspeicher verbraucht das Programm?

Fortgeschrittene Übung: Synchronisation mit dem Schlüsselwort „volatile“

Gegeben sei folgender Code-Ausschnitt (in Anlehnung an [1], S. 21). Gehen Sie davon aus, dass jede Methode in einem eigenen Thread ausgeführt wird:

```
public class Processor {
      private boolean connectionPrepared = false;

      public void prepareConnection() {
            // … open connection …
            connectionPrepared = true;
      }
      public void start() throws InterruptedException {
            // … various initializations …
```

```
            while (! connectionPrepared) {
                  Thread.sleep(500)
            }
            // … start actual processing …
      }
}
```

1. Auf einem Single-Core-System ist dieser Code ohne Synchronisation fehlerfrei lauffähig, wohingegen er auf einem Multi-Core-System Probleme verursachen kann. Finden Sie heraus aus welchem Grund?
2. Informieren Sie sich über das Schlüsselwort „volatile“. An welcher Codestelle ist dieses in o.g. Code einzusetzen?
3. Welche Technik der Synchronisation halten Sie hier für besser geeignet - „synchronized“ oder „volatile“? Begründung!

Quelle:

[1] Java Core Programmierung - Memory Model und Garbage Collection; A. Langer, K. Kreft