Übung 1: Multi-Threading Grundlagen

Lernziele:

- Eigene einfache Threads implementieren, starten und „joinen“.
- Nicht-deterministische Thread-Ausführungsgeschwindigkeit nachvollziehen.
- Unterschied zwischen Daemon und normalen Threads verstehen.
- Thread-Kosten einschätzen.

Aufgabe 1: Thread-Implementierungen

In der Vorlage zu den Übungen finden Sie das Programm `ConsoleTicker`, welches wiederholt in bestimmten Zeitabständen ein Zeichen ausgibt.

Starten Sie nun mehrere Threads, die jeweils einen solchen Ticker ausführen. Jeder Thread soll im Ticker sein eigenes Zeichen in seinem eigenen Zeit-Intervall ausgeben.

Probieren Sie Ihre Lösung mit ein paar Threads einmal mit unterschiedlichen und einmal mit gleichen Intervallen aus. Wieso sind die Ausgaben nicht unbedingt immer schön regelmässig?

Aufgabe 2: Daemon-Threads

Markieren Sie die Ticker-Threads in der Lösung zur Aufgabe 1 als Daemon.

a) Was beobachten Sie bei der Ausführung? Die Ausgabe ist “.A” Es ist nicht garantiert, dass aber überhaupt etwas ausgegeben wird.
b) Das Programm soll warten, bis der Benutzer eine Konsoleneingabe tätigt. Realisieren Sie das mit diesen Daemon-Threads.
c) Die Threads sollen sich vor Beendigung noch mit einer Ausgabe verabschieden. Wieso funktioniert das nicht ohne weiteres? Wir wissen noch nicht, wie es geht.

Aufgabe 3: Prozessoren auslasten

Das Ziel dieser Aufgabe ist es, ein Multi-Thread-Programm in Java zu schreiben, das alle Prozessoren voll auslastet.

In der Vorlage finden Sie einen Algorithmus, der leider nur sequentiell läuft. Dieser zählt alle Primzahlen in einem gewissen Zahlenbereich.

Ändern Sie die Lösung, so dass möglichst alle Prozessoren ausgelastet sind.

Vorgehen:

- Zerlegen Sie den Zahlenbereich in Unterbereiche.
- Rechnen Sie jeden Unterbereich mit einem eigenen Thread.
- Jeder Thread ergibt ein Teilresultat.
- Joinen Sie alle Threads.
- Summieren Sie die Teilresultate der Threads.
- Am einfachsten ist es, wenn Sie eine Sub-Klasse von Thread designen.

Führen Sie das Programm aus und konfigurieren Sie es so, dass alle Prozessoren für eine gewisse Zeit voll ausgelastet sind (überprüfen sie die Prozessoren-Auslastung zum Beispiel im Task Manager). Stellen Sie auch sicher, dass das Resultat immer noch stimmt.

Aufgabe 4: Maximale Thread-Anzahl

Finden Sie heraus, wie viele Threads ungefähr maximal in einem Java Programm gleichzeitig laufen können. Was beobachten Sie, wenn das Limit erreicht wird. Wieso tritt der beobachtete Effekt auf?

Tipp: Versuchen Sie fortlaufend neue Threads zu starten, die zum Beispiel folgende Implementierung haben:

```
() -> {
  try {
    while (true) {
      Thread.sleep(1000000);
    }
  } catch (InterruptedException e) {
    e.printStackTrace();
  }
}
```

Aufgabe 5: Thread Debugging

Spielen sie mit dem Debugger von Eclipse bzw. IntelliJ und versuchen sie, ein Multi-Thread-Programm zu debuggen.