Übung 9: Semaphore in Java (Standardklasse)

Ziel der Übung:

Nachdem in Übung 9 ein Semaphor mit einfachen Mitteln implementiert wurde, soll nun eine Standard-Semaphor-Implementierung kennengelernt werden, die mit Java ausgeliefert wird. Desweiteren sollen die Grenzen des Systems ausgetestet werden.

Einführung:

In Java wird mit der Klasse `java.util.concurrent.Semaphore` bereits eine Semaphor-Implementierung ausgeliefert. Die wichtigsten Operationen sind:

- `Semaphore(int permits)`: Im Konstruktor wird dem Semaphor mitgeteilt, wie viele Threads sich gleichzeitig im kritischen Abschnitt befinden (:= permits) dürfen.
- `void acquire()`: Diese Methode muss vom jeweiligen Thread vor dem Eintritt in den kritischen Abschnitt aufgerufen werden. Wenn die Anzahl an Threads, die sich im kritischen Abschnitt befinden, bereits die Obergrenze erreicht hat, muss der Thread warten. Andernfalls kann er in den kritischen Abschnitt eintreten und die Anzahl der Threads, die sich im kritischen Abschnitt befinden, wird um eins reduziert.
- `void release()`: Wird beim Verlassen des kritischen Abschnitts aufgerufen, damit die Erlaubnis zurückgegeben wird und diese dann von anderen Threads genutzt werden kann.

Daneben gibt es einige weitere interessante Methoden, die von der Klasse angeboten werden. Dies sind unter anderem:

- `boolean hasQueuedThreads()`: Gibt zurück, ob sich gerade ein Thread in der Warteschlange befindet.
- `int getQueueLength()`: Gibt eine Schätzung zurück, wieviele Threads auf Einlass in den kritschen Abschnitt warten.
- `Collection <Thread> getQueuedThreads()`: Liefert die wartenden Threads in einer Collection[1] zurück

Aufgabenstellung:

Die Klasse `Semaphore` soll genutzt werden. Gehen Sie dazu folgendermaßen vor:

1. Machen Sie sich mit dem Umgang dieser Standardklasse vertraut. Literatur findet sich dazu im Internet. Gute Einstiegspunkte sind beispielsweise die bereits bekannte Java API, oder der „Semaphore“-Abschnitt im Openbook “Java ist auch eine Insel” von Christian Ullenboom (zu finden über Google).
2. Importieren Sie das Java-Projekt (WiInf_Übung9_Angabe) und synchronisieren Sie den Zähler (Counter) mithilfe eines Standard-Semaphors (bekannt aus Übung 6).

   Wie viele Threads werden, in diesem Fall sinnvollerweise gleichzeitig im kritischen Abschnitt zugelassen?

[1] Collections dienen dazu, Java-Objekte in Gruppen zu bearbeiten, siehe Package java.util.Collections

3. Versuchen Sie das Programm so zu ändern, dass ausgegeben wird, welcher Thread in den kritischen Abschnitt rein will, welcher Thread sich im kritischen Abschnitt befindet und wann der Thread den kritischen Abschnitt verlässt. Überprüfen Sie nun, anhand der Programmausgaben, ob es sich hierbei um eine faire Lösung handelt!

   (Hinweis: Je mehr Threads initalisiert und gestartet werden, desto höher steigt die Komplexität des Programms und die Konsolenausgaben werden unvorhersebar.)

4. Suchen Sie in der Java-API nach einer Methode, die überprüft, ob die implementierte Lösung fair ist. Sollte dies nicht der Fall sein, modifizieren Sie das Programm so, dass ein fairer Ablauf gewährleistet ist. Was bedeutet „fair“ in diesem Zusammenhang?

5. Erhöhen Sie die Threadanzahl z.B. auf 10.000[2]. Bringen Sie das Programm zum Ablauf. Was passiert und welche Exception wird geworfen? Welche Methode verursacht den Wurf dieser Exception? Versuchen Sie diese Exception abzufangen, um das Programm ordnungsgemäß zu beenden!

[2] Je nachdem wie viel Speicher der JVM zugewiesen ist, muss der Wert deutlich erhöht werden. Testen Sie dies entsprechend mit Ihrem System.