Tutorium 3: Threads

Bearbeitung in den Tutorien am 12. und 13. Mai 2020

Themen

- Threads
- Sleep und Interrupt

Abgabe

Die Tutoriumsaufgaben werden im Tutorium gemeinsam bearbeitet, daher ist keine Abgabe erforderlich.
Die Hausaufgaben sind bis zum **19.5.2020 7:00** im **ISIS Quiz #3** zu lösen.

Tutoriumsaufgaben

In diesem Tutorium wollen wir uns weiterhin mit der Implementierung von Threads beschäftigen. Diesmal wollen wir uns Möglichkeiten zur Orchestrierung (oder Abstimmung) von Threads widmen. Betrachten Sie dazu die Vorlage in `tut03/src` im GitLab-Repository `prog2-uebung-20`.

1. Thread-Zustände

a. Benennen Sie die Thread-Zustände
b. Benennen Sie die Java-Methoden mit denen man einen Thread in den jeweiligen Zustand versetzt (sofern eine Methode existiert).
c. Wenn keine Methode existiert: Von wem wird der Thread in den jeweiligen Zustand versetzt?

2. Thread-Ping-Pong

Betrachten Sie hierzu die Vorlagen im git-Repository tut03/src/main/java. Implementieren Sie zwei Threads, die miteinander (unendlich lang) „Ping-Pong“ spielen. Dazu soll ein

Thread immer mit „Pong“ auf die Nachricht „Ping“ eines anderen Threads antworten. Beachten Sie dazu:

- Ein Thread gibt immer nur „Ping“ aus, der andere immer nur „Pong“.
- Die Nachricht wird nur ausgegeben, wenn zuvor die Nachricht des anderen Threads ausgegeben wurde.
- Die Ausgabe wird unendlich lang wiederholt bis das Programm manuell beendet wird.

Wie können Sie die Reihenfolge der Ausgaben sicherstellen ohne:

a. vorangegangene Ausgaben auf der Konsole zu überprüfen
b. ohne das Konzept von Monitors (synchronized) zu benutzen
c. ohne einen dritten Thread, der die anderen beiden Threads steuert

Hausaufgaben

In dieser Hausaufgabe wollen wir das erlangte Wissen aus dem Tutorium nun vertiefen. Betrachten Sie dazu die Vorlage in `ha03/src/main/java` im GitLab-Repository `prog2-uebung-20`. Beachten Sie die folgenden Hinweise:

1. Implementieren Sie hier die Klasse `AppendOrHashThread` (siehe Vorlage: ha03/src/main/java/AppendOrHashThread.java).

   Betrachten Sie dazu auch die Klassen `Hasher` (siehe Vorlage: ha03/src/main/java/Hasher.java) und `Data` (siehe Vorlage: ha03/src/main/java/Data.java), sowie die Klasse `Main` (siehe Vorlage: ha03/src/main/Main.java).

   - Implementieren Sie mittels dieser Klasse einen Thread, der eine ID (Typ String) als Attribut hat. Der Thread soll seine ID an die statische Referenz `word` (Data.java) ranhängen oder aber `word` mittels `Hasher.hash()` hashen. Wurde `word` gehashed soll ebenfalls die statische Referenz `currentIterations` inkrementiert werden. Threads des Typs `AppendOrHashThread` sollen in einer festen Reihenfolge ausgeführt werden: zuerst zwei Threads die ihre ID an `word` ranhängen (append) gefolgt von einem Thread der `word` hashed. Nachdem das `word` gehashed wurde soll eine neue Iteration beginnen.
   - Ein Wort wird in einer Iteration jeweils um die ID von t1 und t2 erweitert, indem die IDs beider Threads mit dem Wort konkateniert werden. Dabei ist streng zu beachten, dass zuerst die ID von t1 und dann die ID von t2 an das Wort gehangen wird. Bevor eine neue Iteration beginnt, wird das Wort mit den IDs der Threads in einem eigenen Thread t3 gehashed (nutzen Sie dazu `Hasher.hash()` aus der Vorlage).
   - Beispiel:

- Sei word="", t1="A" und t2="B"
- Dann passiert folgendes in Iteration 1:
- t1 wird ausgeführt -> word="A"
- t2 wird ausgeführt -> word="A" + "B"
- t3 wird ausgeführt -> word=Hasher.hash(word)
- In Iteration 2 wird dann wie folgt weiter gemacht:
- t1 wird ausgeführt -> word=Hasher.hash(word)+"A"
- t2 wird ausgeführt -> word=Hasher.hash(word)+"A" + "B" // * word wie in t3 aus 1. Iteration
- t3 wird ausgeführt -> word=Hasher.word(Hasher.hash(word)+"A" + "B") // * word wie in t3 aus 1. Iteration
- usw.

  - Dieser Ablauf soll `numOfIterations`-mal (siehe Data.java) ausgeführt werden.

2. Implementieren Sie hier die Methode `verify()` (siehe Vorlage: ha03/src/main/java/Data.java).

   Betrachten Sie dazu auch die Klasse `Hasher` (siehe Vorlage: ha03/src/main/java/Hasher.java).

- Implementieren Sie die Methode `verify()`, die als Parameter (i) eine ID eines Threads (t1) und (ii) eine ID eines weiteren Threads (t2) übergeben bekommt, sowie (iii) einen Hash, den es zu überprüfen gilt. Dazu soll geprüft werden, ob die Threads t1 und t2 den übergebenen Hash erzeugt haben oder nicht. Zur Verifizierung verzichten wir für die Erzeugung die Hashes die Verteilung der Rechenarbeit auf mehrere Threads.
- Beachten Sie:
  - Nehmen Sie hier im Quiz für `numOfIterations` fix 250000 Iterationen (verwenden Sie in der Antwort nicht `numOfIterations`)

- Geben Sie ein `boolean` zurück, Sie müssen sich nicht um die Ausgabe kümmern
- Wiederholen Sie nochmal wie man in Java überprüft, ob zwei Strings identisch sind