Tutorium 2: Threads und Runnables

Bearbeitung in den Tutorien am 5. und 6. Mai 2020

Themen

- Threads und Runnables
- Singletons

Abgabe

Die Tutoriumsaufgaben werden im Tutorium gemeinsam bearbeitet, daher ist keine Abgabe erforderlich.
Die Hausaufgaben sind bis zum **12.5.2020 7:00** im **ISIS Quiz #2** zu lösen.

Tutoriumsaufgaben

In diesem Tutorium wollen wir uns mit der Implementierung von Threads beschäftigen. Dazu wollen wir zunächst zwei Schleifen mittels der Java-Klasse Thread und dem Java-Interface Runnable parallelisieren. Außerdem wollen wir die Grenzen der parallelen Ausführung austesten und überprüfen, inwiefern das Amdahl'sche Gesetz korrekt ist. Betrachten Sie dazu die Vorlage in `tut02/src` im GitLab-Repository `prog2-uebung-20`.

1. Extends Thread

1. Erstellen Sie zwei for-Schleifen (A und B), die nacheinander ausgeführt werden und in jedem Durchlauf ihren Namen (A bzw. B) und die entsprechende Nummer des Durchlaufs ausgeben. **Optional:** Können Sie hier auch eine for-each-Schleife nutzen?
2. Parallelisieren Sie nun die beiden Schleifen, indem Sie diese in zwei Threads abarbeiten. Schreiben Sie dafür eine Klasse, die von Thread erbt (extends Thread).

3. Frage: Können Sie auch die `run()` Methode aufrufen? Begründen Sie Ihre Antwort!

2. Implements Runnable

1. Schreiben Sie eine Klasse Logger, die eine Methode `log()` implementiert. Diese Methode bekommt zwei Parameter `id` und `number`, die auf der Konsole mit `System.out.println()` ausgegeben werden sollen.

2. Nutzen Sie nun die Logger-Klasse für die Ausgabe der zwei for-Schleifen aus **Aufgabe 1.1**, indem die von Ihnen erstellte Klasse aus **Aufgabe 1.2** von der Logger-Klasse erbt. **Frage:** Treten Probleme auf? Wenn ja, wieso?

3. Lösen Sie das Problem aus **Aufgabe 2.2** mit dem Runnable-Interface.

3. Grenzen von Threads

1. **Frage:** Was ist eine Singleton-Klasse? Welche besonderen Eigenschaften hat diese? Warum sind diese Eigenschaften besonders wichtig, wenn Sie mit Threads arbeiten?

2. Implementieren Sie nun eine Singleton-Klasse Queue, die ein Array mit **N** Elementen erzeugt. Füllen Sie das Array mit beliebigen Großbuchstaben. **Hinweis:** Hier können Sie auch immer denselben Großbuchstaben verwenden. **Optional:** Wählen Sie einen zufälligen Großbuchstaben.

3. Schreiben Sie nun ein Programm, das eine Instanz der Queue-Klasse mit **N** Elementen erstellt und mit **M** Threads alle Großbuchstaben der Queue in Kleinbuchstaben umwandelt. Messen Sie die Zeit, die das Programm für diese Aufgabe benötigt hat.

4. Führen Sie das Programm aus **Aufgabe 3.3** mit **N=25.000.000** und jeweils **M= 1/2/4/8/16/32**. **Frage:** Was fällt auf? Bleibt der Performance-Gewinn konstant?

Hausaufgaben

In dieser Hausaufgabe wollen wir das erlangte Wissen aus dem Tutorium nun vertiefen. Betrachten Sie dazu die Vorlage in `ha02/src/main/java` im GitLab-Repository `prog2-uebung-20`. Beachten Sie die folgenden Hinweise:

1. Implementieren Sie die Methode `rightPad()` aus der Klasse Data (siehe Vorlage: `ha02/src/main/java/Data.java`).
    - Die Methode bekommt einen String übergeben und soll diesen auf eine Länge **l** Zeichen reduzieren.
    - Wurde ein String mit einer Länge kleiner **l** übergeben, so soll der Rest mit einem Zeichen **r** aufgefüllt werden (z.B. `"123456789" -> "123456789000"`, für **l=12** und **r="0"**).
    - Wurde ein String mit einer Länge größer l übergeben, so sollen allen Zeichen nach dem l-ten Zeichen "abgeschnitten" werden (z.B. `"123456789012345" -> "123456789012"`, für **l=12** und **r="0"**).
    - Wird null übergeben, so soll null als String `null` interpretiert werden (z.B. `null -> "null00000000"`, für **l=12** und **r="0"**)
    - **l** und **r** sollen ebenfalls als Parameter übergeben werden.
2. Implementieren Sie hier die Klasse WordProcessor (siehe Vorlage: `ha02/src/main/java/WordProcessor.java`). Diese Klasse soll in einem separaten Thread eine bestimme Anzahl an Elementen aus dem words-Array der Klasse Data (siehe Vorlage: `ha02/src/main/java/Data.java`) lesen und alle gelesenen Wörter wie folgt transformieren:

- alle Großbuchstaben sollen durch Kleinbuchstaben ersetzt werden
- alle Wörter sollen exakt 12 Zeichen lang sein; ist ein gelesenes Wort kürzer, so wird der Rest mit "0" gefüllt (z.B. 123456789000)
- Nutzen Sie hierzu die `rightPad()` Funktion aus **Aufgabe 1**.

- Beachten Sie bei der Implementierung die folgenden Anforderungen:
  - Die Klasse soll von der Klasse Thread erben.
  - Bei der Instanziierung eines WordProcessor-Objekts muss ein `offset` und ein `limit` angegeben werden.
  - Der Parameter `offset` gibt an ab welcher Stelle vom `words`-Array aus der Klasse Data gelesen werden soll. Ist der übergebene `offset` negativ, so soll das Vorzeichen ignoriert werden. Übersteigt der `offset` die Anzahl an Elementen im `words`-Array, soll "wieder von vorne begonnen werden", d.h. (sei **N** die Anzahl an Elementen im `words`-Array, dann gilt, dass das ein `offset = 0` und ein `offset = n` von derselben Stelle beginnen.
  - Der Parameter `limit` gibt an wie viele Elemente gelesen werden sollen. Ist das übergebene `limit` negativ, so soll das Vorzeichen ignoriert werden. Übersteigt das `limit` die Anzahl an Elementen im `words`-Array, soll nur bis zum Ende des `words`-Array gelesen werden. Das gilt auch, wenn das `limit` zwar kleiner ist als die Länge des `words`-Array, aber in Kombination mit dem `offset` zu einer Übersteigung der Elemente im `words`-Array führen würde.

3. Implementieren Sie hier die Klasse WordProcessorRunnable (siehe Vorlage: `ha02/src/main/java/WordProcessorRunnable.java`). Betrachten Sie dazu auch die Klasse WordLogger (siehe Vorlage: `ha02/src/main/java/WordLogger.java`).

    - Nun wollen wir das Logging (die Ausgabe) der Wörter anpassen, dazu haben wir einen speziellen (sehr simplen) Logger geschrieben. (In der Realität werden an dieser Stelle Logger eingesetzt, die Logs sammeln, komprimieren, an einen externen Log-Server senden, etc., - wir wollen es hier einfach halten). Dieser Logger gibt neben dem verarbeiteten Wort auch den Hash-Code des Wortes aus.
    - Allerdings müssen Sie von der Klasse WordLogger erben, damit Sie die `log()`-Methode nutzen können. Daher müssen Sie Ihre WordProcessor-Klasse nun umschreiben, damit diese das Interface Runnable implementiert, anstatt von der Klasse Thread zu ergeben.
    - Ansonsten soll die Funktionalität erhalten bleiben.