[GK] 8a: Grand Prix

✔ Done: View ✔ Done: Make a submission ✔ Done: Receive a grade

Wir wollen ein Rennen mit mehreren Runden simulieren, in dem mehrere Threads gegeneinander laufen. Neben drei Läufer-Threads fungiert der Main-Thread als Schiedsrichter, der das Startsignal gibt und die Zeiten der Threads auf der Konsole ausgibt.

Die Ausgabe eines Wettrennens könnte zum Beispiel so aussehen:

```
Die Läufer sind bereit...
Start!
Thread 1 hat Runde 1 nach 200ms abgeschlossen!
Thread 2 hat Runde 1 nach 240ms abgeschlossen!
Thread 3 hat Runde 1 nach 330ms abgeschlossen!
Thread 1 hat Runde 2 nach 390ms abgeschlossen!
Thread 2 hat Runde 2 nach 450ms abgeschlossen!
Thread 1 hat Runde 3 nach 560ms abgeschlossen! Platz 1!
Thread 3 hat Runde 2 nach 590ms abgeschlossen!
Thread 2 hat Runde 3 nach 620ms abgeschlossen! Platz 2!
Thread 3 hat Runde 3 nach 890ms abgeschlossen! Platz 3!
```

Verwende dafür (gemeinsam mit dem EK-Teil) dieses Github-Repository: https://classroom.github.com/a/bfbyL2wA

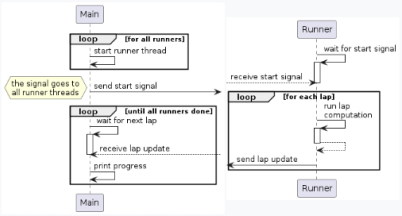
Aufgabenstellung

Implementiere die Anwendung, in der mehrere Threads wie folgend beschrieben kommunizieren und so eine Ausgabe wie oben gezeigt zustandekommt. Die geforderten Ausgaben erfolgen dabei alle durch den Main-Thread.

> Beurteilt wird nicht der abgegebene Code, sondern das Abgabegespräch. **Zu einem Abgabegespräch muss trotzdem eine funktionierende Lösung vorliegen!** Im Abgabegespräch werden Fragen zum Stoffgebiet und *detaillierte Fragen* zur abgegebenen Lösung gestellt. Der Maßstab ist: **du musst wissen, was jede Codezeile deiner Lösung tut und warum es sie gibt.** Wirf einen Blick auf die Bewertungsrubrik, um einen Überblick über die Kriterien zu bekommen.

Grundstruktur Multithreading

Es gibt insgesamt vier Threads: der Main-Thread ist der Schiedsrichter, zusätzlich werden drei Läufer-Threads gestartet. Die Threads sollen mittels der Klassen in `java.util.concurrent`, insbesondere einer der `BlockingQueue`s und einem `CountDownLatch`, miteinander kommunizieren/synchronisiert werden. Der Ablauf soll den folgenden Sequenzdiagrammen entsprechen:



Im Main-/Schiedsrichter-Thread passiert also folgendes:

- Die Läufer-Threads werden gestartet
- Es wird an die jetzt laufenden Threads ein Startsignal gesendet.
- In einer Schleife:
  - Warten auf das Update zur nächsten abgeschlossenen Runde
  - Ausgeben der Details zur Runde
- Die Schleife endet, wenn alle Runden Abgeschlossen sind (Hinweis: die Anzahl der Läufer-Threads und der Runden pro Läufer ist bekannt.)

In jedem Läufer-Thread passiert folgendes:

- Bevor es wirklich losgeht wird auf das Startsignal gewartet
- Dreimal in einer Schleife:
  - Die Berechnung für die aktuelle Runde durchführen (jedes mal eine andere)
  - Ein Update zur abgeschlossenen Runde an dem Main-Thread schicken

Die Threads müssen also auf zwei Arten miteinander kommunizieren:

- Der Schiedsrichter-Thread gibt das Startsignal, alle Läufer-Threads warten bis das Signal kommt. Dabei ist wichtig, dass alle Läufer-Thread das Signal *gleichzeitig* bekommen.
- Die Läufer-Threads schicken Runden-Updates, der Schiedsrichter-Thread empfängt die Updates der Reihe nach. Tipp: das lässt sich als ein Producer-Consumer-Problem auffassen.

Beachte, dass alle geforderten **Ausgaben** durch den **Main-Thread** erfolgen.

Hinweis: Thread-Synchronisation ist berüchtigt dafür, dass sich dabei leicht Fehler einschleichen können; was einmal funktioniert, kann beim nächsten mal schief gehen. Deswegen ist es hier *besonders* wichtig, Code verständlich zu gestalten - Stichwort Kommentare.

Berechnungen der Läufer-Threads

In einem echten Rennen besteht die Leistung darin, die Rundendistanz zurückzulegen. Hier wird stattdessen Rechenleistung erbracht. In jeder Runde müssen die Threads daher eine einigermaßen zeitaufwendige Berechnung durchführen, z.B.

- für eine bestimmte Zeit schlafen (`Thread.sleep()`, nur GKÜ)
- rekursiv die n-te Fibonacci-Zahl berechnen
- Quadratwurzeln mit der babylonischen Methode berechnen
- proof-of-work-artige Berechnung
- ...

Variiere die genaue Aufgabe (z.B. anderes `n`, andere Präzision für die Wurzelberechnung) pro Thread und pro Runde, damit nicht alle Threads fast genau gleich schnell sind.

Die Berechnungen, die ein Läufer in den drei Runden absolviert, sollen nicht "hard-coded" sein. Der Läufer bekommt seine Berechnungen im Konstruktor übergeben, also Array oder Collection. (Wenn für GKÜ z.B. nur `Thread.sleep()` benutzt wird, reicht es aus ein Array mit drei Zahlen anzugeben. In jeder Runde wird für die angegebene Zeit geschlafen.)

Anforderungen GK überwiegend

- Die Ausgabe ist ähnlich dem Beispiel, mit mindestens drei Läufern und drei Runden.
- Die Threads sind entsprechend den Sequenzdiagrammen und der Beschreibung synchronisiert, kein busy waiting o.Ä.
- Mindestens eine Berechnung ist umgesetzt. Die Berechnungen, die ein Thread in den einzelnen Runden durchführen soll, werden als Array oder Liste angegeben.
- Der Sourcecode ist sinnvoll dokumentiert.

Zusätzliche Anforderungen GK vollständig

- Es sind mindestens zwei verschiedene Berechnungen implementiert, die für die Runden der Läufer verwendet werden, kein `sleep`.
  - Ein Läufer soll in verschiedenen Runden verschiedene Berechnungen durchführen können. Trotzdem sollen die Beechnungen als Array oder Liste angegeben werden können.
- Die Kommunikations-Logik ist **in einer eigenen Klasse gekapselt**, sodass die `main`-/`run`-Methoden der Threads nicht direkt die Klassen in `java.util.concurrent` benutzen. (`BlockingQueue` und `CountDownLatch` **dürfen** verwendet werden, nur nicht direkt in den run-Methoden der Thread-Klassen bzw. der main-Methode.)

Abgabe

Trage zur Abgabe einen direkten Github-Link auf den letzten **Commit** in deinem Repository ein. Dieser Link müsste also so aussehen: `https://github.com/TGM-HIT/sew8-2324-grand-prix-<github-username>/commit/<commit-id>`;

Dein abgegebener Commit muss den Sourcecode beinhalten. Die Beurteilung erfolgt in einem Abgabegespräch.

Edit submission | Remove submission

Submission status

Submission status	Submitted for grading
Grading status	Graded
Last modified	Wednesday, 3 April 2024, 1:59 PM
Online text	+ https://github.com/TGM-HIT/sew8-2324-grand-prix-kaperbm/commit/7563d2cc08edc10ef8e0d4613bbd77d43af96504
Submission comments	Comments (0)

Grading criteria

Codeverständnis GKÜ-Teil	Mangelhaftes Verständnis der abgegebenen Lösung *0 points*	*1 points*	Fragen und Änderungen im Programm konnten mit leichten Unsicherheiten beantwortet werden. *2 points*	*3 points*	Fragen zum "Was" und "Warum" sowie Änderungen zur abgegebenen Lösung konnten gut beantwortet werden. *4 points*
Codeverständnis GKV-Teil	Mangelhaftes Verständnis der abgegebenen Lösung		Fragen und Änderungen im Programm konnten mit		Fragen zum "Was" und "Warum" sowie Änderungen

	0 points		beantwortet werden. *1 points*		zur abgegebenen Lösung konnten gut beantwortet werden. *2 points*
Multithreading	Mangelhaftes Verständnis des zugrundeliegenden Stoffgebiets *0 points*	*1 points*	Fragen zum zugrundeliegenden Stoffgebiet konnten mit leichten Unsicherheiten beantwortet werden. *2 points*	*3 points*	Fragen zum zugrundeliegenden Stoffgebiet konnten gut beantwortet werden. *4 points*
Pünktlichkeit	Nachfrist *0 points*		Knapp verspätetes Abgabegespräch *1 points*		Rechtzeitiges Abgabegespräch *2 points*

Feedback

Grade	12.00 / 12.00
Graded on	Wednesday, 3 April 2024, 2:15 PM
Graded by	SM Schabel Markus

Grade breakdown

Codeverständnis GKÜ-Teil	Mangelhaftes Verständnis der abgegebenen Lösung *0 points*	*1 points*	Fragen und Änderungen im Programm konnten mit leichten Unsicherheiten beantwortet werden. *2 points*	*3 points*	Fragen zum "Was" und "Warum" sowie Änderungen zur abgegebenen Lösung konnten gut beantwortet werden. *4 points*
Codeverständnis GKV-Teil	Mangelhaftes Verständnis der abgegebenen Lösung *0 points*		Fragen und Änderungen im Programm konnten mit leichten Unsicherheiten beantwortet werden. *1 points*		Fragen zum "Was" und "Warum" sowie Änderungen zur abgegebenen Lösung konnten gut beantwortet werden. *2 points*
Multithreading	Mangelhaftes Verständnis des zugrundeliegenden Stoffgebiets *0 points*	*1 points*	Fragen zum zugrundeliegenden Stoffgebiet konnten mit leichten Unsicherheiten beantwortet werden. *2 points*	*3 points*	Fragen zum zugrundeliegenden Stoffgebiet konnten gut beantwortet werden. *4 points*
Pünktlichkeit	Nachfrist *0 points*		Knapp verspätetes Abgabegespräch *1 points*		Rechtzeitiges Abgabegespräch *2 points*