

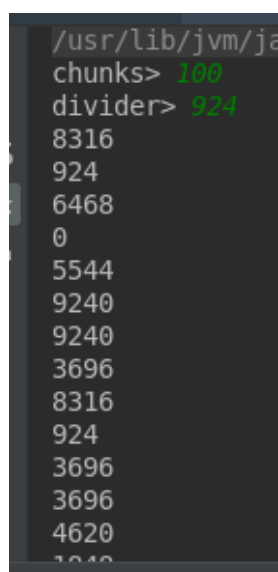
1 Beispiel

Sie finden eine Datei namens „numbers.csv“ im Moodle. Diese Datei enthält Zahlen getrennt durch einen Doppelpunkt. Leider sind durch einen Übertragungsfehler einige Zahlen verloren gegangen und es haben sich auch ein paar Buchstaben eingeschlichen. Beim Lesen der Datei sollen Sie nur die erfolgreich übertragenen Zahlen lesen.

Schreiben Sie nun ein Programm, welches alle Zahlen aus der Datei in eine `ArrayList` einliest. Nun haben Sie eine Liste von x Zahlen. Der Benutzer gibt über die Konsole einen Teiler ein. Es soll nun jede Zahl der Liste überprüft werden, ob die eingegebene Zahl ein Teiler dieser Zahl ist.

Wir erinnern uns: Eine Zahl $a \in \mathbb{Z}$ teilt eine Zahl $b \in \mathbb{Z}$ dann und nur dann, wenn es eine Zahl n gibt für die gilt $a \cdot n = b$. Oder anders ausgedrückt: Eine ganze Zahl ist durch eine andere ganze Zahl teilbar, wenn bei der Division kein Rest verbleibt, also die „Geteilt-Rechnung aufgeht“.

Die Überprüfung der Teilbarkeit der Zahlen der Liste mit der eingegebenen Zahl soll parallel mithilfe der Java Concurrency Api erfolgen. Der Benutzer gibt über die Konsole ein in wie viele Teile die Ursprungsliste geteilt werden soll. Teilen Sie die ursprüngliche Liste von Zahlen nun in die angegebene Menge an Teilen und überprüfen Sie in einem Thread jeden Teil separat. Wenn ein Thread eine Zahl findet die durch die vom Benutzer eingegebene Zahl teilbar ist, geben Sie diese einfach auf die Konsole aus.



```
/usr/lib/jvm/ja
chunks> 100
divider> 924
8316
924
6468
0
5544
9240
9240
3696
8316
924
3696
3696
4620
```

Abbildung 1: Screenshot der fertigen Anwendung

2 Beispiel

Die gaußsche Summenformel ist eine Formel für die Summe der ersten n aufeinanderfolgenden natürlichen Zahlen:

$$1 + 2 + 3 + \dots + n = \sum_{k=1}^n k = \frac{n^2 + n}{2}$$

Beispiel:

$$1 + 2 + 3 + \dots + 210 = \sum_{k=1}^{210} k = \frac{210^2 + 210}{2} = 22155$$

Sie sollen nun mithilfe der Java Concurrency Api ein Programm entwickeln, welches diese Summe berechnen kann und zwar parallel. Teilen Sie dazu die gesamte Summe in Summen zu je 100 Zahlen auf:

Summe der Zahlen 1 - 100: 5050

Summe der Zahlen 101 - 200: 15050

Summe der Zahlen 201 - 210: 2055

Gesamtsumme: $5050 + 15050 + 2055 = 22155$

Der Benutzer kann die Zahl n eingeben und es wird die Summe, welche parallel berechnet wurde ausgegeben.

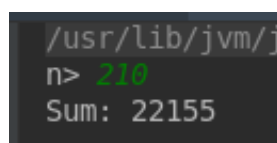


Abbildung 2: Screenshot der fertigen Anwendung

Sie können Ihre Ergebnisse mit der oben angegebenen Formel überprüfen. Aufgrund des Kommutativgesetzes ist es egal welcher Thread welche Summe zuerst berechnet!

3 Beispiel

Programmieren Sie die Lösung wie in das im Moodle bereitgestellte StudentStub Projekt!

Sudoku ist ein Zahlenrätsel: „Schreiben Sie einfach in jedes freie Feld eine Zahl von 1 bis 9, so dass jede Zeile, jede Spalte und jedes 3x3-Kästchen alle Zahlen von 1 bis 9 enthält. Das heißt, dass pro Zeile, Spalte und 3x3-Kästchen keine Zahl doppelt vorkommen darf.“ Quelle: Puzzler Media Ltd.

In Abbildung 3 sehen Sie ein Beispiel für ein ungelöstes Sudoku. In Abbildung 4 sehen Sie das gelöste Sudoku.

	3							
			1	9	5			
		8					6	
8				6				
4			8					1
				2				
	6					2	8	
			4	1	9			5
							7	

Abbildung 3: ungelöstes Sudoku

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Abbildung 4: gelöstes Sudoku

3.1 Aufgabe

Öffnen Sie die Klasse `SudokuSolver`, die wenigstens die Methoden des Interface `ISudokuSolver` implementiert (siehe `Student_Stub` im Moodle). Die Methoden sollen dabei folgende Funktionalität bereitstellen:

- `SudokuSolver::readSudoku`: Liest eine Sudoku-Datei (csv-File siehe Anhang) ein. Verwenden Sie dafür eine Stream-Lambda-Kombination. Die eingelesene Datei wird in ein 9x9 2D-int-Array umgewandelt und zurückgegeben.
- `SudokuSolver::checkSudoku`: Überprüfen Sie für das als Parameter übergebene Sudoku-Array, ob in jeder Zeile, Spalte und in jedem 3x3-Kästchen alle neun Zahlen nur genau einmal vor-

kommen. In diesem Fall wird `true` zurückgegeben, sonst `false`.

- **SudokuSolver::solveSudoku**: Löst das als Parameter übergebene Sudoku und gibt das gelöste Sudoku zurück. Für die Sudokus mit den Namesendungen `*_level1.csv` ist folgende Strategie ausreichend:

1. Wrap/Initialize: Speichere für jede der 9x9 Zellen alle neun Zahlen, die als Einträge in Frage kommen können. Hat eine Zelle einen vorgegebenen/fixen Eintrag, so speichere nur diesen.
2. Reduce: Verwerfe nun für jede Zelle all jene der neun Zahlen, die aufgrund der ganz oben genannten Sudoku-Regel nicht in der Zelle stehen dürfen, weil sie bereits in einer anderen Zelle in der gleichen Zeile, Spalte oder dem gleichen 3x3-Kästchen fixiert sind.
3. Select: In allen Zellen, in denen nun nur noch eine Zahl übrig bleibt, wird diese als Eintrag für diese Zelle fixiert.
4. Führe alle Schritte ab 3. erneut aus, solange mindestens ein weiterer Eintrag in einer Zelle fixiert werden kann.
5. Das Sudoku ist gelöst.

3.2 Aufgabe

Verwenden Sie soweit wie möglich Streams/Lambdas.

3.3 Aufgabe

Erstellen Sie die Methode `public long benchmark(int[][] rawSudoku)`, welche den Mittelwert der Laufzeit von 10 Durchläufen der drei in Aufgabe 1 genannten Methoden - entweder mit `System.currentTimeMillis()` oder mit `System.nanoTime()` und `TimeUnit.MILLISECONDS.convert(nanoTime, TimeUnit.NANOSECONDS)` - berechnet und zurückgibt.

3.4 Aufgabe

Parallelisiere die Methode `SudokuSolver::checkSudoku` mit Hilfe von `ExecutorServices`.

3.5 Aufgabe

Parallelisiere die Methode `SudokuSolver::solveSudoku` mit Hilfe von `ExecutorServices`.

3.6 Aufgabe

Lassen Sie den Benchmark erneut laufen. Wie viel schneller ist der Benchmark mit den parallelisierten Methoden? Entspricht die Änderung euren Erwartungen. Wenn ja/nein: warum (als Kommentar)?

3.7 Aufgabe

Programmieren Sie ihre Lösung so um, dass sie auch die Datei mit der Namensendung *_level2.csv lösen kann. Hier passiert es, dass nach dem Schritt 3. Reduce nicht keine einzelne Zahl innerhalb einer Zelle stehen bleibt. Stattdessen kommt innerhalb einer Zeile, Spalte oder eines 3x3-Kästchens eine Zahl nur einmal vor. Diese Zahl muss dann in der dazugehörigen Zelle fixiert werden.

subsubsectionMögliches UML-Diagramm

Ein MÖGLICHES UML-Diagramm könnte wie folgt aussehen, wobei ihr bis auf das vorgegebene Interface in der Gestaltung Ihrer Klassen und Programmierung völlig freie Hand habt:

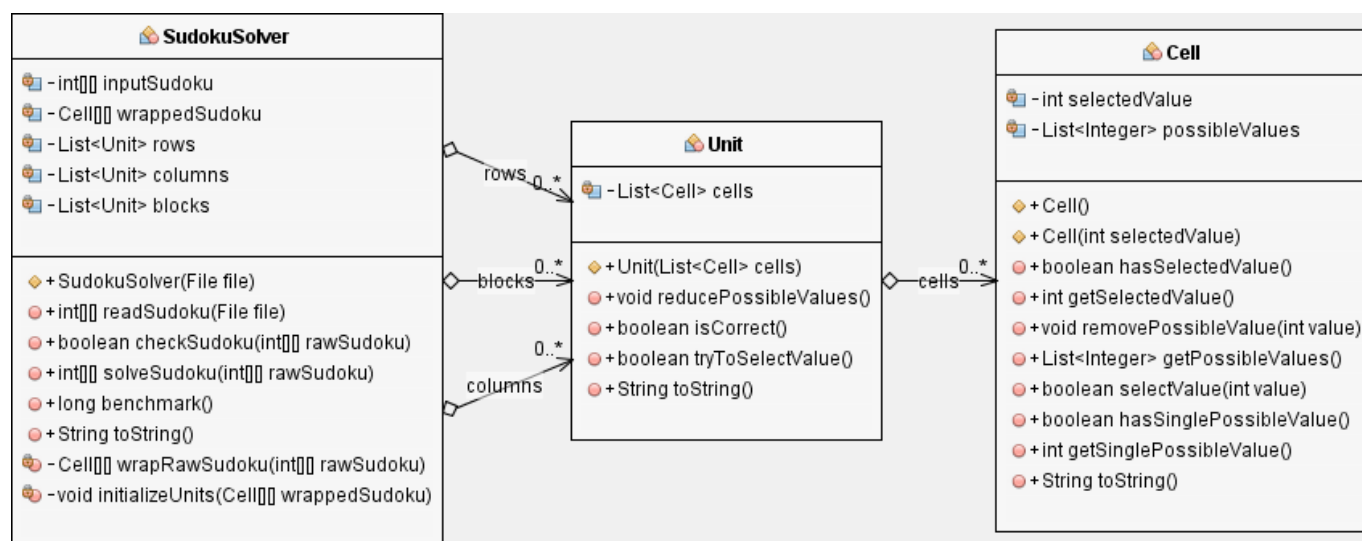


Abbildung 5: mögliches UML-Diagramm