hochschule mannheim

FAKULTÄT FÜR INFORMATIK

PP – Parallele Programmierung
Bachelor Informatik (IB)
Sommersemester 2020 [25.06.2020]

Hausarbeit (Prüfungsleistung)

Bearbeitung:	einzeln (Plagiate führen zum Nichtbestehen und werden an den Prüfungsausschuss gemeldet)
Abgabe:	spätestens am 31.07.2020, 16:00 Uhr (harte Deadline, sonst Ausschluss) über Moodle-Upload (max. 20 Dateien, max 20 MB pro Datei)
Format:	Laden Sie einzelne Java-Quelltextdateien und eine PDF-Datei hoch. Quellcodes müssen einheitlich und sinnvoll formatiert sein.

In den folgenden Aufgaben soll der Merge-Sort Algorithmus umgesetzt werden. Es gibt unterschiedliche Varianten, aus denen Sie wählen können. Es ist beispielsweise nicht vorgegeben, ob das Verfahren *stabil* sein muss oder nicht. Verwenden Sie aber in allen Aufgaben dasselbe grundlegende Verfahren.

Die zu sortierenden Daten sollen in jeder Aufgabe jeweils einer von Ihnen zu programmierenden Methode als `LinkedList` (Package `java.util`) übergeben werden. Die Lösung muss generisch sein: Die einzige Annahme über die zu sortierenden Daten, die Sie treffen dürfen, ist, dass die zu sortierenden Daten von einem Typ sind, der das Interface `Comparable` implementiert (Elemente des Inhaltstyps müssen mit sich selbst vergleichbar sein). Nach dem Sortieren soll die als Parameter übergebene `LinkedList` aufsteigend sortiert sein. Das im Sinne von `compareTo` kleinste Element soll also am Anfang der Liste stehen, nachdem sie sortiert wurde.

Zu jeder Programmieraufgabe wird als Abgabe eine Klasse erwartet, die die von Ihnen entworfene Methode enthält. Zu jeder Programmieraufgabe soll es auch jeweils eine Klasse mit allen zur Aufgabe gehörenden Testfällen geben.

Es geht nicht um die Optimierung des Algorithmus, lediglich das Parallelisierungspotenzial soll mit den in jeder Aufgabe vorgegebenen Mitteln maximal ausgeschöpft werden. Dabei dürfen keine Annahmen über die Hardware gemacht werden. Die Ressourcen (CPUs bzw. Cores) der Ablaufumgebung sollen aber bei den Lösungen zu den Aufgaben 2–4 optimal genutzt werden.

Kommentare im Quellcode, die „`TODO`“ (o.ä.) enthalten (insb. automatisch erzeugte), führen zu Punktabzug. Ebenso führen uneinheitliche Formatierung, Vorhandensein von Race-Conditions, unkoordinierter konkurrierender Zugriff auf geteilte Variablen, überflüssige Verwendung von Locks oder des Schlüsselworts `volatile` und potenzielle Verklemmung zu Punktabzug.

Alle nicht privaten Methoden und etwaige nicht privaten Intanz- oder Klassenvariablen müssen mit sinnvollen und verständlichen JavaDoc-Kommentaren (Sprache einheitlich deutsch oder englisch) ausgezeichnet sein. Die Sprache der Kommentare muss auch bei der Benennung von Bezeichnern verwendet werden. Die üblichen Konventionen über die Schreibweise (groß/klein/Binnenmajuskel/Unterstriche) von Bezeichnern bei Java müssen

eingehalten werden. Code-Dopplungen (auch zwischen den Aufgaben) müssen vermieden werden. Alle Klassen sollen im Package `pp` liegen.

Die Java-Quellcodes müssen mit einem Java 14 Compiler übersetzbar sein. Es dürfen keine Methoden oder Typen benutzt werden, die in Java 14 als *deprecated* gekennzeichnet oder bereits entfallen sind.

1 Merge-Sort (sequenziell)

Aufgabe 1: **von 20 P**

Aufgabe 1

Der Merge-Sort Algorithmus soll sequenziell umgesetzt werden. Implementieren Sie eine öffentliche Methode, die eine übergebene `LinkedList` eines vergleichbaren Inhaltstyps aufsteigend sortiert. Das Ergebnis soll sein, dass die übergebene Liste nach dem Aufruf der Methode sortiert ist.

Implementieren Sie zusätzlich die folgenden drei JUnit-Testfälle für Ihre Sortiermethode:

- leere Liste eines frei gewählten Typs sortieren
- eine Liste mit 32 Strings, die anfangs absteigend (also falsch herum) sortiert ist, sortieren
- eine Liste mit 128 Zufallszahlen, die in zufälliger Reihenfolge angeordnet sind, sortieren (Sie können die Sortiermethode `java.util.Collections.sort`[1] verwenden, um zu prüfen, ob die Sortierung Ihrer Implementierung korrekt ist.)

Bei diesen drei Testfällen muss geprüft werden, ob alle Elemente der jeweiligen Liste richtig sortiert wurden. Die Daten für die Testfälle in den folgenden Programmieraufgaben, mit denen die dort implementierten Sortiermethoden geprüft werden, sollen dann dieselben Eigenschaften wie hier haben (auch dort muss nur die Reihenfolge geprüft werden, keine Eigenschaften der Parallelverarbeitung).

2 Merge-Sort (`Thread`-Klasse mit `Runnable` als anonyme innere Klasse)

Aufgabe 2: **von 20 P**

Aufgabe 2

Der Merge-Sort Algorithmus soll mit Nebenläufigkeit umgesetzt werden. Implementieren Sie eine öffentliche Methode, die eine übergebene `LinkedList` eines vergleichbaren Inhaltstyps aufsteigend sortiert. Das Ergebnis soll sein, dass die übergebene Liste nach dem Aufruf der Methode sortiert ist. Der Code soll folgende Eigenschaften erfüllen:

- Es soll die Klasse Thread verwendet werden, damit Teile des Programms nebenläufig sind und parallel ausgeführt werden können.
- Legen Sie eine maximale Anzahl von Threads in Abhängigkeit der Anzahl der verfügbaren Prozessoren bzw. Cores fest.

[1] `https://docs.oracle.com/javase/7/docs/api/java/util/Collections.html#sort(java.util.List)`

- Das Sortieren jeder einzelnen Teilliste soll in einem eigenen getrennten Thread passieren, solange die von Ihnen festgelegt maximale Anzahl noch nicht erreicht ist (Sie brauchen dabei nur die von Ihnen zum Sortieren gestarteten Threads berücksichtigen). Wenn die maximale Anzahl bereits erreicht ist, dürfen keine Threads mehr gestartet werden. Wird ein Thread beendet, zählt er nicht mehr zu der Anzahl der gestarteten Threads.
- Threads sollen mit dem Konstruktor von `Thread` erzeugt werden, der ein `Runnable` als Parameter erwartet.
- Das `Runnable` soll mit einer anonymen inneren Klasse erzeugt werden.
- Threads müssen über den Aufruf einer öffentlichen *Setter*- bzw. *Mutator*-Methode des `Runnable`, die eine private Instanzvariable vom Typ `boolean` ändert, beendet werden können.
- Wenn die Sortierung fertig ist, dürfen keine zum Sortieren gestarteten Threads mehr „`isAlive()`“ sein.

Es soll eine Klasse mit drei JUnit-Testfällen geben, die wie in Aufgabe 1 prüfen, ob das Ergebnis der Sortierung richtig ist. Die Testdaten sollen die gleichen Eigenschaften wie bei Aufgabe 1 haben.

3 Merge-Sort (Thread-Pool mit Lambda-Ausdruck)

Aufgabe 3 | **Aufgabe 3: von 20 P**

Der Merge-Sort Algorithmus soll mit Nebenläufigkeit umgesetzt werden. Implementieren Sie eine öffentliche Methode, die eine übergebene `LinkedList` eines vergleichbaren Inhaltstyps aufsteigend sortiert. Das Ergebnis soll sein, dass die übergebene Liste nach dem Aufruf der Methode sortiert ist. Der Code soll folgende Eigenschaften erfüllen:

- Es soll ein Thread-Pool verwendet werden, damit Teile des Programms nebenläufig sind und parallel ausgeführt werden können.
- Wählen Sie einen zur Aufgabenstellung passenden Thread-Pool-Typ aus und erzeugen Sie ihn.
- Das Sortieren jeder einzelnen Teilliste soll getrennt im Thread-Pool passieren.
- Die Aufgaben, die im Thread-Pool nebenläufig bearbeitet werden, sollen in Form von Lambda-Ausdrücken übergeben werden.
- Der Thread-Pool muss am Ende der Sortierung beendet und seine Ressourcen wieder freigegeben werden.

Es soll eine Klasse mit drei JUnit-Testfällen geben, die wie in Aufgabe 1 prüfen, ob das Ergebnis der Sortierung richtig ist. Die Testdaten sollen die gleichen Eigenschaften wie bei Aufgabe 1 haben.

4 Merge-Sort (Fork-Join-Framework)

Aufgabe 4 | **Aufgabe 4: von 20 P**

Der Merge-Sort Algorithmus soll mit Nebenläufigkeit umgesetzt werden. Implementie-

ren Sie eine öffentliche Methode, die eine übergebene `LinkedList` eines vergleichbaren Inhaltstyps aufsteigend sortiert. Das Ergebnis soll sein, dass die übergebene Liste nach dem Aufruf der Methode sortiert ist.

Es soll das Fork-Join-Framework verwendet werden, damit Teile des Programms nebenläufig sind und parallel ausgeführt werden können.

Es soll eine Klasse mit drei JUnit-Testfällen geben, die wie in Aufgabe 1 prüfen, ob das Ergebnis der Sortierung richtig ist. Die Testdaten sollen die gleichen Eigenschaften wie bei Aufgabe 1 haben.

5 Experiment

Aufgabe 5: **von 20 P**

Aufgabe 5

Entwerfen Sie ein Experiment, mit dem Ihre vier Implementierungen (das ist die erste unabhängige Variable) des Merge-Sort-Algorithmus verglichen werden können, führen Sie es durch und werten Sie die Ergebnisse aus.

- Wählen Sie für das Experimental-Design eine zweite unabhängige Variable und eine geeignete abhängige Variable.
- Entwerfen Sie einen Experimentalplan mit mindestens 12 Messpunkten und schreiben Sie ein Hauptprogramm, das diesen Plan umsetzt. Der Plan soll nicht die JUnit-Testfälle aus den vorigen Aufgaben enthalten oder darauf aufbauen. Geben Sie auch den Quellcode dieses Programms ab.
- Lassen Sie das Programm laufen und protokollieren Sie die Messungen.
- Wiederholen Sie die Messungen, so dass Sie für jeden Messpunkt (Kombination aus den zwei unabhängigen Variablen) insgesamt drei Werte der abhängigen Variable gemessen haben und bilden Sie den Mittelwert der drei Messungen.
- Das Protokoll der Messungen und die errechneten Mittelwerte müssen in dem abzugebenen PDF-Dokument enthalten sein. Die Berechnung der Mittelwerte muss nicht selbst programmiert werden, sondern Sie können bspw. ein Tabellenkalkulationsprogramm verwenden.
- Visualisieren Sie Ihre Ergebnisse und schreiben Sie einen kurzen Text, in dem Sie die vier Implementierungen anhand Ihres Experiments vergleichen. Visualisierung und Text müssen auch im abzugebenen PDF-Dokument enthalten sein.