Philipps-Universität Marburg
Fachbereich Mathematik und Informatik
Arbeitsgruppe Datenbanksysteme

Sommersemester 2023
21. 06. 2023

Prof. Dr. Bernhard Seeger
Amir El-Shaikh, M.Sc.

Programmierpraktikum

AG Datenbanksysteme
Tag 1

Aufgabe 1: Einarbeitung in die Java Streaming API **(4 Punkte)**

Stellen Sie sicher, dass Sie die Java-20 Version verwenden. Sie können diese **hier**, oder über Ihre Arbeitsumgebung **Intellij IDEA** automatisch herunterladen (empfohlen). Arbeiten Sie sich für folgende Aufgaben in die `Stream`-API von Java 20 ein. Lesen Sie dazu folgende Dokumentationen durch und machen Sie sich mit den Funktionalitäten vertraut:

- Das `Stream` Interface.
- Die `StreamSupport` Klasse.
- Die `Collectors` Klasse.
- Non-Blocking I/O (nio) Paket Übersicht.

Schauen Sie sich auch die in der Dokumentation beschriebenen Beispiele an. Ihr Tutor wird Ihnen zu jedem Punkt drei Fragen stellen, um sicherzustellen, dass Sie die Grundlagen kennen. Eine Frage könnte z.B. sein: Was ist ein `IntStream`?

Weiter ist für die Bearbeitung der Aufgaben folgendes zu beachten:

1. Laden Sie sich das Klassen-Gerüst aus dem Ilias herunter und bearbeiten Sie darin die folgenden Aufgaben.
2. Es gibt insgesamt 30 Punkte heute. Davon müssen Sie mindestens **15** Punkte sammeln, um zu bestehen. Am Tag 2 werden wieder insgesamt 30 Punkte zu vergeben sein, hier sind ebenfalls mindestens **50%** der Punkte zu erzielen, um zu bestehen.

Viel Erfolg bei der Bearbeitung!

Aufgabe 2: Einführung in die Java Streaming API (1+1+1+1+1+1) (6 Punkte)

Bearbeiten Sie folgende Aufgaben in der Klasse `StreamingJava`. Verlassen Sie für die Bearbeitung dieser Teil-Aufgaben **nicht** die `Stream`-API.

a) Vervollständigen Sie die Methode `<E> Stream<E> flatStreamOf(List<List<E>> list)`, um die Eingabe `list` zu einem `Stream<E>` zu transformieren.
Hinweis: Nutzen Sie die Stream-Methode `flatMap`.

b) Vervollständigen Sie die Methode `<E> Stream<E> mergeStreamsOf(Stream<Stream<E>> stream)`, um die Eingabe `stream` zu einem `Stream<E>` zu transformieren. Dabei soll die Reihenfolge der zusammengesteckten Streams keine Rolle spielen.
Hinweis: Nutzen Sie die `Stream`-Methode `reduce`.

c) Vervollständigen Sie die Methode `<E extends Comparable<? super E>> E minOf(List<List<E>> list)`, um aus der übergebenen `list` das Minimum zu ermitteln. Nutzen Sie einen parallelen `Stream` und für die Ordnung die natürliche Ordnung der Elemente. Gibt es kein Minimum, soll eine Exception geworfen werden.
Hinweis: Nutzen Sie die `Stream` Methode `min`.

d) Vervollständigen Sie die Methode `<E> E lastWithOf(Stream<E> stream, Predicate<? super E> predicate)`, um das letzte Element zu finden, welches das Prädikat `predicate` erfüllt.
Hinweis: Nutzen Sie die `Stream` Methode `filter`.

e) Vervollständigen Sie die Methode `<E> Set<E> findOfCount(Stream<E> stream, int count)`, um ein `Set<E>` zurückzugeben, die alle Elemente aus `stream` enthält, die genau `count` oft in `stream` vorkommen.
Hinweis: Nutzen Sie die `Stream` Collector-Methode `Collectors.groupingBy()`.

f) Vervollständigen Sie die Methode `IntStream makeStreamOf(String[] strings)`, um das übergebene String-Array in einen `IntStream` zu transformieren. Dabei sollen die einzelnen Buchstaben der Zeichenketten in dem zurückgegebenen `Stream` enthalten sein.
Hinweis: Nutzen Sie die `Stream`-Methode `IntStream.flatMapToInt()`.

Aufgabe 3: Datenverarbeitung (1+1+1+3+4+3+4+3) **(20 Punkte)**

In dieser Aufgabe sollen Sie Daten aus einer **CSV**-Datei verarbeiten. Um bestens für die Bearbeitung folgender Aufgaben vorbereitet zu sein, machen Sie sich zuerst mit folgenden Funktionalitäten aus dem **Non-Blocking I/O (nio) Paket** vertraut:

1. Die Klasse `Files`.
2. Das Interface `Path`.

Bearbeiten Sie folgende Teil-Aufgaben in der Klasse `StreamingJava`. Weiter sollen Sie die Datei names **NaturalGasBilling.csv** aus der Vorlage laden können. Dazu sollen Sie Ihr Projekt-Verzeichnis kennen, um in den Teil-Aufgaben auf diese Datei zugreifen zu können.

Bemerkung: Für diese Teil-Aufgaben gibt es keine Beschränkungen. Sie können die Aufgaben auch vollständig ohne die `Stream`-API lösen.

a) Vervollständigen Sie die Methode `Stream<String> fileLines(String path)`, um die Datei names **NaturalGasBilling.csv** zu laden. Dabei sollen Sie einen **BufferedReader** nutzen und es sollen Zeilen des Inhalts über den `Stream<String>` zurückgegeben werden. Achten Sie dabei darauf, dass die erste Zeile übersprungen wird und dass der zurückgegebene `Stream<String>` sequentiell ist. Fügen Sie noch eine Konsolen-Ausgabe hinzu, sobald der zurückgegebene `Stream<String>` geschlossen wurde.

b) Vervollständigen Sie die Methode `double averageCost(Stream<String> lines)`, um die durchschnittlichen Kosten (Spalte **Amount**) aus allen Einträgen aus `lines` (Übergabe aus a)) zu berechnen.

c) Vervollständigen Sie die Methode `long countCleanEnergyLevy(Stream<String> lines)`, um die Einträge `lines` (Übergabe aus a)) zu zählen, die keinen (leer oder 0) `Clean energy levy` Betrag gezahlt haben.

d) Erstellen Sie einen `record` namens `NaturalGasBilling`, um Datensätze aus der Datei **NaturalGasBilling.csv** darzustellen. Dazu sollen Sie jedes Attribut als Feld definieren. Weiter sollen Sie die Methode `Stream<NaturalGasBilling> orderByInvoiceDateDesc(Stream<String> stream)` erzeugen, um aus dem übergeben `stream`, einen nach **Invoice Date** absteigenden sortierten `Stream<NaturalGasBilling>` zu erhalten.

e) Fügen Sie dem `record` names `NaturalGasBilling` die Methode `Stream<Byte> toBytes()` hinzu, die den Datensatz in ein `Stream<Byte>` zu transformieren und dabei das Format bzw. die Reihenfolge der **CSV**-Datei **NaturalGasBilling.csv** verfolgt. Diese Methode soll nicht die Attribut-Namen (erste Zeile der **NaturalGasBilling.csv** Datei), sondern nur den Datensatz des `record` umwandeln.

f) Erstellen Sie die Methode `Stream<Byte> serialize(Stream<NaturalGasBilling> stream)`, um den übergebenen `stream` in einen `Stream<Byte>` zu transformieren. Dieser zurückgegebene `Stream` soll dazu verwendet werden, den `Datenbestand` erneut in eine **CSV**-Datei zu schreiben. Daher achten Sie dabei, dass die erste Zeile die Attribute beinhaltet, gefolgt von den serialisierten Datensätzen. Nutzen Sie nun Ihre `serialize()` und `orderByInvoiceDateDesc()` Methoden und speichern Sie die Datensätze aus `NaturalGasBilling.csv` absteigend sortiert nach **Invoice Date** in eine neue **CSV**-Datei.

g) Erzeugen Sie die Methode `Stream<NaturalGasBilling> deserialize(Stream<Byte> stream)`, um einen serialisierten `Stream<Byte>` aus f) zurück in einen `Stream<NaturalGasBilling>` zu überführen. Machen Sie den Aufruf: `deserialize(serialize(orderByInvoiceDateDesc(fileLines(Datei aus f))))` und schreiben Sie das Resultat auf die Konsole.

h) Vervollständigen Sie die Methode `Stream<File> findFilesWith(String dir, String st, String ed, int maxFiles)`, um das Verzeichnis `dir` nach Dateien zu durchsuchen. Es sollen alle Dateien (auch in Unterverzeichnissen), die mit `st` beginnen und mit `ed` enden gefunden werden. Weiter sollen die gefundenen Dateien nach Größe absteigend sortiert zurückgegeben werden. Es dürfen dabei maximal `maxFiles` viele Dateien zurückgegeben werden. Machen Sie einen Aufruf Ihrer Methode, um alle Java-Source-Code Dateien aus Ihrem Java-Projekt zurück zu bekommen.