Objektorientierte Programmierung Blatt 7

Institut für Softwaretechnik und Programmiersprachen | Sommersemester 2024
Matthias Tichy, Raphael Straub und Florian Sihler

Abgabe (git) bis
16. Juni 2024

Streams, Functional Interfaces, Collections 3 ★ 12 🟦 4 🟢

- Vertieft mit Collections arbeiten
- Streams anwenden

Aufgabe 1: Streams of Primitive Data Types

In dieser Aufgabe lernen wir die Unterschiede zwischen regulären, generischen Streams und Streams für primitive Datentypen kennen. Erstellen Sie für jede Teilaufgabe eine Methode, die die gegebenen Objekte als Parameter akzeptiert und den gesuchten Wert zurückgibt.
Hinweis: Beachten Sie bei jeder Aufgabe ob ein Array oder eine Liste zu verwenden ist.

a) Vorhandensein von negativen Werten 🟦

Gegeben sei eine Liste an `long`-Werten. Überprüfen Sie, ob negative Werte in der Liste enthalten sind. Geben Sie `true` zurück, falls ein negativer Wert gefunden wurde, ansonsten `false`.

b) Zeichenzählung 🟦🟦

Zählen Sie im übergebenen `char`-Array die Anzahl der Vorkommen eines bestimmten Zeichens, welches Sie als 2. Parameter übergeben bekommen. Geben Sie die Anzahl der Vorkommen des Zeichens als `int` zurück.

c) Summe von Integer-Werten 🟦

Gegeben sei ein Array an `int`-Werten. Berechnen Sie die Summe aller Quersummen der Integer im Stream. Sollte das Array leer sein, geben Sie `0` zurück.

d) Durchschnittliche Bewertung 🟦🟦🟢

Gegeben sei eine Liste von `double`-Werten, die Bewertungen von Nutzern repräsentieren. Berechnen Sie die Standardabweichung der Bewertungen. Falls die Liste Leer ist geben Sie `0` zurück. Zur Berechnung der Standardabweichung können Sie die folgende Formel verwenden:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

Dabei ist:

- N die Anzahl der Datenpunkte,
- x_i der i-te Datenpunkt,
- μ der Mittelwert der Datenpunkte, definiert als $\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$.

e) Boolean-Stream 🟦🟢

Gegeben sei ein Array an `boolean`-Werten. Berechnen Sie den logischen ODER-Wert aller Werte im Stream. Falls das Array leer ist, geben Sie `false` zurück.

Aufgabe 2: Functional Interfaces and Foundation

Willkommen auf Terminus! Das Imperium hat beschlossen, eine große Bücherei auf Terminus zu errichten. Sie sind einer der Ingenieure, die für die Implementierung der Bücherei-Software verantwortlich sind.

Gegeben (im vorbereiteten Repository) sind die folgenden Klassen und Interfaces: `Book`, `Microfilm`, `ElectronicBook`, `DataCube`, `LibraryUser`, `LibraryMember`, `Librarian`, `Library`.

Hinweis: Die Functional Interfaces müssen den exakten Namen der Aufgabenstellung haben, damit die Tests funktionieren.

a) Buchbewertung Anwenden 🟦🟦

Erstellen Sie ein Functional Interface `BookRater`, das eine Methode `double rate(Book book)` enthält.

Implementieren Sie die Methode `applyRatings(BookRater rater)` in der Klasse `Library`, die den `BookRater` auf jedes Buch anwendet und das Rating im entsprechenden Buch speichert.

Erweitern Sie hierfür das Interface `Book` um die Methode `void rateBook(BookRater rater)`, die den `BookRater` auf sich selbst anwendet und das Rating speichert. Hierbei stellt `DataCube` eine Ausnahme dar, da es nicht direkt bewertet werden kann. Stattdessen, soll `DataCube` den `BookRater` auf jedes intern gespeicherte `ElectronicBook` anwenden.

b) Bücher Formatieren ⭐🟦🟦

Erstellen Sie ein Functional Interface `BookFormatter`, das eine Methode `String format(Book book)` enthält.

Implementieren Sie die Methode `formatBooks(BookFormatter formatter)` in der Klasse `Library`, die einen `BookFormatter` akzeptiert, auf jedes Buch anwendet und eine Liste der formatierten Bücher zurückgibt.

c) Bücher Finden 🟦🟢

Implementieren Sie die Methode `findBooksByTitle` in der Klasse `Library`, die eine Liste von Büchern zurückgibt, die den gegebenen Titel enthalten. Erweitern Sie die Funktion um einen Parameter `BookComparator sortBy`, welcher es erlauben soll, die Bücher nach einem bestimmten Attribut zu sortieren.

Erstellen Sie hierfür ein Functional Interface `BookComparator`, welches die Methode `int compare(Book b1, Book b2)` enthält. Die `compare`-Methode soll eine negative ganze Zahl, null oder eine positive ganze Zahl zurückgeben, je nachdem, ob das erste Argument kleiner als, gleich oder größer als das zweite ist.

d) Medien Konvertieren ⭐🟦🟢

Erstellen Sie ein Functional Interface `MediaConverter`, das eine Methode `R convert(Book media)` enthält. Stellen Sie dazu sicher, dass `R` eine Klasse ist, die `Book` implementiert.

Diese Methode soll benutzt werden können, um alle Medien in der Bibliothek in eins der Formate, die `Book` implementieren (`Microfilm`, `ElectronicBook`, `DataCube`), zu konvertieren.

Implementieren Sie die Methode `convertMedia` in der Klasse `Library`, die einen `MediaConverter` akzeptiert und eine Liste der konvertierten Medien zurückgibt. Passen Sie hierzu den

Rückgabetyp der Methode `convertMedia` an, sodass sie eine Liste des korrekten Typs zurückgibt. Hierbei soll keine `List<Book>` zurückgegeben werden, sondern eine Liste des Typs `R`.

e) Ausleihen Validieren ★★■■●

Erstellen Sie ein Functional Interface `LendingValidator`, welches die folgende Methode enthält: `boolean checkAccess(LibraryMember member, Book book)`. Erstellen Sie einen weiteren Konstruktor für die Klasse `Library`, welcher einen `LendingValidator` akzeptiert und als Attribut speichert.

Nun können Sie die Methode `boolean lendBook(Book book, Member member)` implementieren, welche benutzt werden muss, wenn jemand ein Buch ausleihen möchte. Nutzen Sie hierfür den `LendingValidator`, um zu überprüfen, ob der Nutzer das Buch ausleihen darf. Falls der Validator das Ausleihen erlaubt, geben Sie `true` zurück, ansonsten `false`. Falls das Buch ausgeliehen werden kann, müssen Sie beachten, das Book-Objekt zu updaten.

Auch hier soll `DataCube` eine Ausnahme darstellen. Ein `DataCube` kann genau dann ausgeliehen werden, wenn alle intern gespeicherten `ElectronicBook` ausgeliehen werden können. In diesem Fall gilt der `DataCube` und alle intern gespeicherten `ElectronicBook` Objekte als ausgeliehen. Dementsprechend müssen alle `ElectronicBook` Objekte als ausgeliehen markiert werden.

f) Flexible Bücher Verarbeitung ★★■■■●●●

Erstellen Sie ein Functional Interface `BookProcessor`, das eine Methode `Book process(Book book)` enthält. Darüber hinaus, erstellen Sie im Functional Interface `BookProcessor` eine statische Methode `BookProcessor compose(BookProcessor p1, BookProcessor p2)`, die zwei `BookProcessor` akzeptiert und einen neuen `BookProcessor` zurückgibt, der beide `BookProcessor` nacheinander anwendet.

Implementieren Sie jetzt die Methode `addBookProcessor(BookProcessor processor)` in der Klasse `Library`, die einen `BookProcessor` akzeptiert und automatisch auf jedes Buch anwendet, das über `addBook(Book book)` zur `Library` hinzugefügt wird. Passen Sie hierzu die Methode `addBook` an.

Wenn bereits ein `BookProcessor` in der `Library` vorhanden ist, sollte der neue `BookProcessor` mit dem vorhandenen kombiniert werden. Hierbei soll der neue `BookProcessor` zuletzt angewendet werden.

Aufgabe 3: Collections and How to Use Them

In dieser Aufgabe haben wir verschiedene Collections und wollen diese benutzen, um entweder eine neue Collection zu erstellen, bestimmte Elemente zu finden oder neue Werte basierend auf den Daten der Collection zu berechnen.

Schreiben Sie hierfür immer eine Methode, welche die gegebenen Objekte als Parameter akzeptiert und den gesuchten Wert zurückgibt.

Hinweis: Da dies eine sehr wichtige Aufgabe ist, mit der Sie auch in der Realität häufig konfrontiert werden, haben wir Ihnen eine große Zahl an Übungsaufgaben bereitgestellt. Die Aufgaben beginnen simpel und werden mit der Zeit komplexer. Wir empfehlen, zu versuchen, die Aufgaben zunächst mit der Verwendung von Streams zu lösen. Eine weitere Lösung mit Schleifen kann ebenfalls sinnvoll sein, um die Ergebnisse miteinander zu vergleichen.

a) Sum of Integers ■

Aufgabe: Summieren Sie alle Integer in der Liste auf. Eine Leere liste gibt `0` zurück.
Gegeben: `List<Integer>`
Gesucht: `Integer`

b) Concatenating String

Aufgabe: Konkatenieren Sie alle Strings in der Liste.
Gegeben: `List<String>`
Gesucht: `String`

c) Find Minimum

Aufgabe: Finden Sie das Minimum der Liste im Bezug auf den Preis. Eine Leere liste gibt `null` zurück.
Gegeben: `List<ValuableItem>`
Gesucht: `Integer`

```
public record ValuableItem(String name, int price, String category) {}
```

d) Find Average

Aufgabe: Finden Sie den durchschnittlichen Preis aller Items in der Liste. Eine Leere liste gibt `0` zurück.
Gegeben: `List<ValuableItem>`
Gesucht: `Double`

e) Find Median

Aufgabe: Finden Sie den Median der Liste von Integern.Eine Leere liste gibt `0` zurück.
Gegeben: `List<Integer>`
Gesucht: `Double`

f) Collect to List

Aufgabe: Gegeben sei eine Map, die Namen auf Preise abbildet, und eine Map, die Namen auf Kategorien abbildet. Nutzen Sie diese Einträge, um eine Liste an `ValuableItem` zu erzeugen. Wenn einer oder beide der Werte in der Map fehlen, dann nutzen sie `0` oder den leeren String.
Gegeben: `Map<String, Integer>` und `Map<String, String>`
Gesucht: `List<ValuableItem>`

g) Find Most Frequent Element

Aufgabe: Finden Sie das am häufigsten vorkommende Element in der Liste.
Gegeben: `List<String>`
Gesucht: `String`

h) Partition by Threshold

Aufgabe: Partitionieren Sie die Liste von Items in zwei Listen: eine Liste mit Items, deren Preis über dem Schwellenwert liegt, und eine Liste mit Items, deren Preis unter oder gleich dem Schwellenwert liegt.
Gegeben: `List<ValuableItem>` und `int threshold`
Gesucht: `Map<String, List<ValuableItem>>`
Hinweis: Die Schlüssel in der Map sollten Äboveünd "BelowOrEqualßein.

i) Count Items by Price Range

Aufgabe: Zählen Sie die Anzahl der Items in verschiedenen Preisbereichen (z.B. 0-100, 101-200, etc.) und geben Sie das Ergebnis als Map zurück, wobei der Schlüssel der Preisbereich und der Wert die Anzahl der Items in diesem Bereich ist. Erzeugen Sie selbst die notwendige anzahl an Preisbereichen.
Gegeben: `List<ValuableItem>`
Gesucht: `Map<String, Integer>`

j) Compute Weighted Average Price

Aufgabe: Berechnen Sie den gewichteten Durchschnittspreis der Items basierend auf einer Gewichtung, die durch eine zusätzliche Map angegeben wird. Die Map enthält die Namen der Items als Schlüssel und die Gewichte als Werte.
Gegeben: `List<ValuableItem>` und `Map<String, Double>`
Gesucht: `Double`

k) K-th Largest Element in Stream

Aufgabe: Implementieren Sie eine Klasse, die den k-größten Wert effizient verfolgt. Es soll eine Methode geben, um einen neuen Wert zur Collection hinzuzuzufügen, und eine Methode, um den aktuellen k-größten Wert zu erhalten.
Gegeben: `int k`
Gesucht: Eine Klasse mit den Methoden `void add(int value)` und `int getKthLargest()`

l) Group by First Letter and Sort

Aufgabe: Gruppieren Sie die Items nach dem ersten Buchstaben ihres Namens und sortieren Sie innerhalb jeder Gruppe die Items aufsteigend nach Preis.
Gegeben: `List<ValuableItem>`
Gesucht: `Map<Character, List<ValuableItem>>`

m) Find the Closest Pair of Points

Aufgabe: Finden Sie das Paar von Punkten mit dem kleinsten Abstand in einer Liste von Punkten. Jeder Punkt ist als `Point` gegeben, der x- und y-Koordinaten hat.
Gegeben: `List<Point>`
Gesucht: `Pair<Point, Point>`

```
public record Point(int x, int y) {}
public record Pair<T, U>(T first, U second) {}
```

n) Merge Intervals

Aufgabe: Geben Sie eine Liste von Intervallen (mit Start- und Endpunkten) so zurück, dass alle überlappenden Intervalle zusammengeführt werden. Gehen Sie davon aus, dass alle Intervalle inklusive ihrer Enden sind.
Gegeben: `List<Interval>`
Gesucht: `List<Interval>`

```
public record Interval(int start, int end) {}
```