Übungen

Gerne erstelle ich für dich einige Übungsbeispiele, die funktionale Programmierung und den Einsatz von Streams in Java abdecken. Diese Übungen sollen dir dabei helfen, ein tieferes Verständnis für funktionale Konzepte in Java und die mächtigen Operationen, die Streams bieten, zu entwickeln.

Übung 1: Grundlagen der Streams

Erzeuge eine Liste von Zahlen und verwende einen Stream, um die Summe der ungeraden Zahlen zu berechnen.

Beispiel:

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
// Berechne die Summe der ungeraden Zahlen
```

Übung 2: Map und Filter mit Streams

Gegeben sei eine Liste von Strings. Verwende Streams, um zunächst alle Elemente zu filtern, die länger als drei Zeichen sind, und wandle diese dann in Großbuchstaben um.

Beispiel:

```
List<String> words = Arrays.asList("Java", "Kotlin", "Scala", "Clojure");
// Filter und Map Operationen anwenden
```

Übung 3: Finden von Elementen

Nutze Streams, um das erste Element in einer Liste von Ganzzahlen zu finden, das größer als 100 ist.

Beispiel:

```
List<Integer> bigNumbers = Arrays.asList(55, 60, 95, 105, 110, 120);
// Finde das erste Element größer als 100
```

Übung 4: Gruppierung von Daten

Verwende Streams, um eine Liste von Strings nach ihrer Länge zu gruppieren.

Beispiel:

```
List<String> names = Arrays.asList("Alice", "Bob", "Charlie", "David");
// Gruppieren nach Länge der Strings
```

Übung 5: Reduzierungen

Berechne die durchschnittliche Länge der Strings in einer Liste mithilfe von Streams.

Beispiel:

```
List<String> names = Arrays.asList("Java", "Python", "JavaScript");
// Berechne die durchschnittliche Länge
```

Lösungsansätze

Um diese Übungen zu lösen, solltest du dich mit den folgenden Stream-Operationen vertraut machen: `filter`, `map`, `reduce`, `collect`, `findFirst`, und `groupBy`. Diese Methoden ermöglichen es dir, komplexe Datenverarbeitungsoperationen auf eine deklarative Weise auszudrücken.

Tipps

- Nutze `.stream()` um eine Kollektion in einen Stream zu konvertieren.
- `filter` nimmt ein Prädikat (eine Funktion, die ein Boolean zurückgibt) und behält nur die Elemente bei, für die das Prädikat `true` ergibt.
- `map` wendet eine Funktion auf jedes Element an und transformiert es dabei.
- `reduce` kombiniert alle Elemente des Streams zu einem einzigen Ergebnis.
- `collect(Collectors.groupingBy(...))` kann genutzt werden, um Elemente nach einem bestimmten Kriterium zu gruppieren.

Versuche, diese Übungen selbst zu lösen, um dein Verständnis für funktionale Programmierung und Streams in Java zu vertiefen.

Lösungen

Natürlich, ich werde dir nun Lösungen für die zuvor genannten Übungsbeispiele liefern. Diese Lösungen helfen dir, ein besseres Verständnis dafür zu entwickeln, wie du funktionale Konzepte und Streams in Java praktisch anwenden kannst.

Lösung zu Übung 1: Summe der ungeraden Zahlen

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
int sum = numbers.stream()
                 .filter(n -> n % 2 != 0)
                 .reduce(0, Integer::sum);
System.out.println("Summe der ungeraden Zahlen: " + sum);
```

Lösung zu Übung 2: Filter und Map

```
List<String> words = Arrays.asList("Java", "Kotlin", "Scala", "Clojure");
List<String> filteredAndUppercased = words.stream()
                                          .filter(word -> word.length() > 3)
                                          .map(String::toUpperCase)
                                          .collect(Collectors.toList());
System.out.println(filteredAndUppercased);
```

Lösung zu Übung 3: Finden von Elementen

```
List<Integer> bigNumbers = Arrays.asList(55, 60, 95, 105, 110, 120);
Integer firstOver100 = bigNumbers.stream()
                                 .filter(n -> n > 100)
                                 .findFirst()
                                 .orElse(null); // Verwende orElse, um einen Standar
System.out.println("Erstes Element größer als 100: " + firstOver100);
```

Lösung zu Übung 4: Gruppierung von Daten

```
List<String> names = Arrays.asList("Alice", "Bob", "Charlie", "David");
Map<Integer, List<String>> groupedByLength = names.stream()
                                                  .collect(Collectors.groupingBy(St
System.out.println(groupedByLength);
```

Lösung zu Übung 5: Durchschnittliche Länge der Strings

```
List<String> names = Arrays.asList("Java", "Python", "JavaScript");
double averageLength = names.stream()
                            .mapToInt(String::length)
                            .average()
                            .orElse(0.0); // Verwende orElse, um einen Standardwert
System.out.println("Durchschnittliche Länge: " + averageLength);
```

Diese Lösungen illustrieren die Grundprinzipien und einige fortgeschrittene Techniken der funktionalen Programmierung und des Stream-APIs in Java. Es ist wichtig, mit diesen Konzepten

zu experimentieren und sie in verschiedenen Kontexten anzuwenden, um ihre volle Kraft und Flexibilität zu erfassen.