3.9 Übungen zu Streams und Filter-Map-Reduce

Lernziel: In diesem Übungsteil wollen wir die Arbeit mit den in JDK 8 neu eingeführten Streams kennenlernen und dabei insbesondere auf das Filter-Map-Reduce-Framework eingehen.

✍ Aufgabe 1 ✍

Aufgabe 1a: Welche Arten von Operationen gibt es für Streams? Was sind typische Vertreter?

Aufgabe 1b: *Create:* Wie erzeugt man einen `Stream<String>`?

```
final String[] namesArray = { "Tim", "Tom", "Andy", "Mike", "Merten" };
final List<String> names = Arrays.asList(namesArray);

final Stream<String> streamFromArray = // ... TODO ...
final Stream<String> streamFromList = // ... TODO ...
final Stream<String> streamFromValues = // ... TODO ...
```

Tipp: Nutzen Sie `Arrays.stream()`, `List.stream()`, `Stream.of()`.

Aufgabe 1c: *Intermediate:* Filtern Sie eine Liste von Namen, z. B. alle beginnend mit "`T`" und führen Sie ein Mapping aus, z. B. auf `UPPERCASE`.

```
final Stream<String> filtered = // ... TODO ...
final Stream<String> mapped = // ... TODO ...
```

Tipp: Nutzen Sie `Stream.filter()` bzw. `Stream.map()`.

Aufgabe 1d: *Terminal:* Geben Sie den Inhalt des Streams auf der Konsole aus.

Tipp: Nutzen Sie `Stream.forEach()`.

✍ Aufgabe 2 ✍

Wandeln Sie den Inhalt eines Streams in ein Array bzw. in eine Liste um:

```
final Object[] contentsAsArray = streamFromArray. // ... TODO ...
final List<String> contentsAsList = streamFromValues. // ... TODO ...
```

Tipp: Verwenden Sie die Vorarbeiten aus Aufgabe 1 sowie die Methoden `toArray()` und `collect(Collectors.toList())`.

✍ Aufgabe 3 ✍

Wenn eine Ausgabe kommaseparierter Werte gewünscht ist, so kann man folgenden Lambda nutzen. Es wird jedoch abschließend ein Komma zu viel ausgegeben.

```
mapped.forEach(str -> System.out.print(str + ", "));
```

Das kann man aufwendig durch eine Spezialbehandlung vermeiden. Dabei stellt sich allerdings die Frage, wie man das letzte Zeichen bei einem parallelen Stream erkennt. Deshalb bietet es sich an, vordefinierte Methoden einzusetzen.

```
final String joined = streamFromArray. // ... TODO ...
System.out.println(joined);
```

Die erwartete Ausgabe ist: `Tim, Tom, Andy, Mike, Merten`

Tipp: Nutzen Sie `collect()` und `Collectors.joining(", ")` zur Ausgabe.

✍ Aufgabe 4 ✍

Aufgabe 4a: Gruppieren Sie den Inhalt nach dem Anfangsbuchstaben, sodass wir folgendes Ergebnis erhalten: `A=[Andy], T=[Tim, Tom], M=[Mike, Merten]`

```
final Stream<String> inputs = Stream.of("Tim", "Tom", "Andy", "Mike", "Merten");
final Map<Character, List<String>> grouped = inputs. // ... TODO ...
System.out.println(grouped);
```

Tipp: Nutzen Sie `collect()` und die `Collectors.groupingBy()`.

Aufgabe 4b: Teilen Sie den Inhalt gemäß einer vorgegebenen Länge, sodass wir folgende Ausgabe erhalten: `false=[Andy, Mike, Merten], true=[Tim, Tom]`

```
final Stream<String> inputs = Stream.of("Tim", "Tom", "Andy", "Mike", "Merten");
final Map<Boolean, List<String>> partioned = inputs. // ... TODO ...
System.out.println(partioned);
```

Tipp: Nutzen Sie `collect()` und die `Collectors.partioningBy()`.

Aufgabe 4c: Gruppieren Sie den Inhalt nach Anfangsbuchstaben und summieren Sie die Anzahl der Vorkommen, sodass wir diese Ausgabe erhalten: `A=1, T=2, M=2`

```
final Stream<String> inputs = Stream.of("Tim", "Tom", "Andy", "Mike", "Merten");
final Map<Character, Long> groupedAndCounted = inputs. // ... TODO ...
System.out.println(groupedAndCounted);
```

Tipp: Nutzen Sie `collect()` sowie die Methoden `Collectors.groupingBy()` und `Collectors.counting()`.

✍ Aufgabe 5 ✍

Gegeben seien die Zahlen von 1 bis 10 als Eingabe. Berechnen Sie das Minimum, Maximum, die Summe und den Durchschnitt dieser Zahlenfolge.

```
final IntStream ints = IntStream.of(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
```

Tipp 1: Verwenden Sie die Stream-Methoden `min()`, `max()`, `sum()` usw.

Tipp 2: Man sieht recht schnell, dass man immer wieder neue Streams erzeugen muss, um die Informationen zu ermitteln. Im JDK findet sich mit der Klasse `IntSummaryStatistics` eine interessante Klasse. Versuchen Sie es mal mit folgender Programmzeile:

```
final IntSummaryStatistics stats = ints.summaryStatistics();
```

✍ Aufgabe 6 ✍

Finden Sie eine grafische Notation für die Stream-Operationen `filter()`, `map()` und `partitioningBy()`. Eine Idee liefert Abbildung 3-4, die zwei Varianten von Gruppierungen (`groupingBy()`) zeigt, einmal nach Form und einmal nach Füllung.

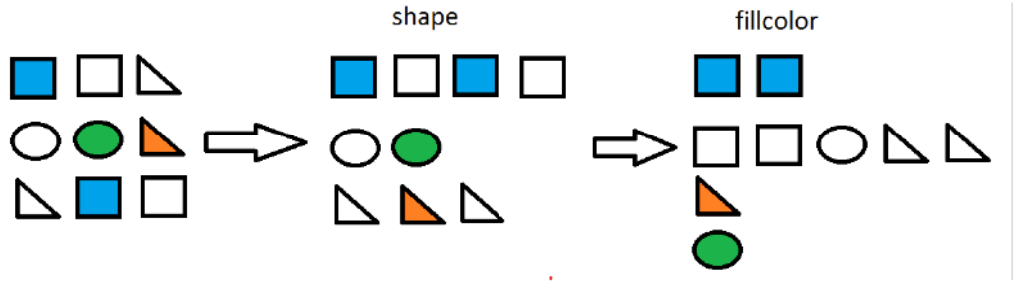

Abbildung 3-4 *Grafische Notation für die Methode* `groupingBy()`