Infozettel: Tipps und Tricks

Matthias Eurich – 2017-04-04

Table of Contents

Einleitung

In den vorhergehenden Zetteln wurden die Grundlagen, Operationen, Reduktion, Evaluation und Parallelisierung von Streams behandelt. In diesem Zettel sollen Tipps und Tricks zu Streams vorgestellt werden.

Arbeiten mit unendlichen Streams

Streams erlauben durch die *lazy Evaluation* das Verarbeiten von unendlichen Datenmengen. Für bestimmte Anwendungsfälle kann durch dieses Feature der Code stark vereinfacht werden.

Im nachfolgenden Beispiel wird ein unendlicher Stream von Ganzzahlen nach Primzahlen gefiltert und das Ergebnis anschließend ausgegeben.

Zuerst wird ein `Predicate` angelegt, mit dem nach Primzahlen gefiltert werden kann. Da nachfolgend ein `IntStream` verwendet wird, muss für die Filterung ein Objekt vom Typ `IntPredicate` verwendet werden.

```
jshell> IntPredicate isPrime = number ->
   ...> number > 1 &&
   ...> IntStream.range(2, number).noneMatch(index -> number % index == 0);
isPrime ==> $Lambda$37/1735934726@335eadca
```

Danach wird eine Methode `primeNumbers` mit dem Rückgabetyp `IntStream` angelegt, die als Filteranweisung das zuvor erzeugte `Predicate` verwendet.

```
jshell> IntStream primeNumbers() {return IntStream.iterate(0, i -> i +
1).filter(isPrime);}
|  created method primeNumbers()
```

Die Methode kann nun an verschiedenen Stellen im Code unterschiedlich genutzt werden, ohne geändert werden zu müssen. Durch die Rückgabe eines Objekts vom Typ `IntStream` kann der Verwender der Methode selbst entscheiden, wie der Stream abgeschlossen werden soll. Nachfolgend zwei Beispiele zum unterschiedlichen Abschluss des Streams.

```
jshell> primeNumbers().limit(2).forEach(System.out::println)
2
3

jshell> primeNumbers().limit(5).sum()
$8 ==> 28
```

Optional

Der Container-Typ `Optional` hat großes Potenzial und kann z.B. viele (unvorhergesehene) `NullPointerExceptions` vermeiden. Es bietet sich daher an, `Optional` nicht nur als möglichen Rückgabewert einer *terminalen Operation* eines Streams zu sehen, sondern `Optional` auch im eigenen Code zu verwenden.

peek()

In manchen Situationen wünscht man sich, man könnte während der Verarbeitung in den Stream hineinschauen. Mit Hilfe der Methode `peek()` ist das möglich.

Im nachfolgenden Beispiel wird das Ergebnis von `limit(5)` mit Hilfe von `peek()` ausgegeben.

```
jshell> primeNumbers().limit(5).peek(System.out::println).sum()
2
3
5
7
11
$10 ==> 28
```

Ohne `peek()` sieht die Ausgabe so aus:

```
jshell> primeNumbers().limit(5).sum()
$11 ==> 28
```

> Die Methode `peek()` ist eine Hilfe für die Entwicklung und das Debuggen von Streams. Sie sollte nicht zur Erzeugung zustandsbehafteter Seiteneffekte missbraucht werden.

Last updated 2017-06-20 21:41:20 CEST