Es passieren komische Dinge

Streams besitzen teilweise ein Eigenleben und verhalten sich anders als erwartet. Dies ist auf ihre Faulheit zurückzuführen. Als Faulheit (Laziness) wird die (vielleicht unerwartete) Art des Verhaltens von Streams bei der Abarbeitung seiner Elemente bezeichnet.

Schaut man sich die Methode filter() etwas genauer an, so erkennt man, dass sie als Argument ein Predicate erwartet. Die Methode dieses funktionalen Interfaces liefert, wie Sie bereits wissen, einen boolschen Wert zurück. Der folgende Quelltext liefert somit einen Fehler, da kein Rückgabewert erzeugt wird. :

```
Stream.of(1, 68, 17, 104, 15)
      .filter(i -> System.out.println("filter: " + i); // Fehler

//Mit Rückgabewert:
Stream.of(1, 68, 17, 104, 15)
      .filter(i -> { System.out.println("filter: " + i);
                     return true;});
```

Abbildung 30: Beispiel für falsche Verwendung

Fügt man einen Rückgabewert hinzu, so wird der Quelltext zwar akzeptiert, erzeugt erstaunlicherweise jedoch keine Ausgabe. Die Ursache besteht darin, dass intermediäre Methoden nur ausgeführt werden, wenn eine terminale Operation vorhanden ist.
Damit nicht genug, auch die Reihenfolge der Abarbeitung ist erstaunlich. Variieren und erweitern wir den Quelltext etwas und fügen eine zweite filter()-Methode hinzu.

```
Stream.of(1, 68, 9, 104, 15)
      .filter(i -> {System.out.println("filter 1: " + i); return i > 10;})
      .filter(i -> {System.out.println("filter 2: " + i); return i % 3== 0;})
      .forEach(i -> System.out.println("forEach: " + i));
```

Abbildung 29: Beispiel mit Filtern

Ausgabe:

```
filter 1: 1
filter 1: 68
filter 2: 68
filter 1: 9
filter 1: 104
filter 2: 104
filter 1: 15
filter 2: 15
forEach: 15
```

Anders als man vielleicht erwarten würde, wird nicht zunächst die erste Methode in der Kette für alle Werte ausgeführt, dann die zweite, etc., sondern nacheinander wird die gesamte Pipeline für jeden einzelnen Wert durchlaufen. Wird eine Bedingung nicht erfüllt, wie es hier beim ersten filter() für die Werte 1 und 9 der Fall ist, so werden die Folgemethoden gar nicht erst in Angriff genommen.
Entsprechend wird die terminale Operation nur dann ausgeführt, wenn die Kette der zuvor durchlaufenen filter()-Methoden jeweils true ergeben hat. Dies ist hier nur für den letzten Wert, 15, der Fall.