

Stream-API

Carsten Gips (HSBI)

Unless otherwise noted, this work is licensed under CC BY-SA 4.0.

Motivation

```
public record Studi(String name, int credits) {}  
public record Studiengang(String name, List<Studi> studis) {}  
public record Fachbereich(String name, List<Studiengang> studiengaenge) {}  
  
private static long getCountFB(Fachbereich fb) {  
    long count = 0;  
    for (Studiengang sg : fb.studiengaenge()) {  
        for (Studi s : sg.studis()) {  
            if (s.credits() > 100) count += 1;  
        }  
    }  
    return count;  
}
```

Innere Schleife mit Streams umgeschrieben

```
private static long getCountSG(Studiengang sg) {  
    return sg.studis().stream()  
        .map(Studi::credits)  
        .filter(c -> c > 100)  
        .count();  
}
```

```
private static long getCountFB2(Fachbereich fb) {  
    long count = 0;  
    for (Studiengang sg : fb.studiengaenge()) {  
        count += getCountSG(sg);  
    }  
    return count;  
}
```

Erzeugen von Streams

```
List<String> l1 = List.of("Hello", "World", "foo", "bar", "wuppie");  
Stream<String> s1 = l1.stream();  
  
Stream<String> s2 = Stream.of("Hello", "World", "foo", "bar", "wuppie");  
  
Random random = new Random();  
Stream<Integer> s3 = Stream.generate(random::nextInt);  
  
Pattern pattern = Pattern.compile(" ");  
Stream<String> s4 = pattern.splitAsStream("Hello world! foo bar wuppie!");
```

Intermediäre Operationen auf Streams

```
private static void dummy(Studiengang sg) {  
    sg.studis().stream()  
        .peek(s -> System.out.println("Looking at: " + s.name()))  
        .map(Studi::credits)  
        .peek(c -> System.out.println("This one has: " + c + " ECTS"))  
        .filter(c -> c > 5)  
        .peek(c -> System.out.println("Filtered: " + c))  
        .sorted()  
        .forEach(System.out::println);  
}
```

Was tun, wenn eine Methode Streams zurückliefert

```
private static long getCountSG(Studiengang sg) {  
    return sg.studis().stream().map(Studi::credits).filter(c -> c > 100).count();  
}  
  
private static long getCountFB2(Fachbereich fb) {  
    long count = 0;  
    for (Studiengang sg : fb.studiengaenge()) {  
        count += getCountSG(sg);  
    }  
    return count;  
}
```

```
private static long getCountFB3(Fachbereich fb) {  
    return fb.studiengaenge().stream()  
        .flatMap(sg -> sg.studis().stream())  
        .map(Studi::credits)  
        .filter(c -> c > 100)  
        .count();  
}
```

Streams abschließen: Terminale Operationen

```
Stream<String> s = Stream.of("Hello", "World", "foo", "bar", "wuppie");

long count = s.count();
s.forEach(System.out::println);
String first = s.findFirst().get();
Boolean b = s.anyMatch(e -> e.length() > 3);

List<String> s1 = s.collect(Collectors.toList());
List<String> s2 = s.toList();    // ab Java16
Set<String> s3 = s.collect(Collectors.toSet());
List<String> s4 = s.collect(Collectors.toCollection(LinkedList::new));
```

- Operationen dürfen nicht die Stream-Quelle modifizieren
- Operationen können die Werte im Stream ändern (`map`) oder die Anzahl (`filter`)
- Keine Streams in Attributen/Variablen speichern oder als Argumente übergeben: Sie könnten bereits “gebraucht” sein!

=> Ein Stream sollte immer sofort nach der Erzeugung benutzt werden
- Operationen auf einem Stream sollten keine Seiteneffekte (Veränderungen von Variablen/Attributen außerhalb des Streams) haben

`Stream<T>`: Folge von Objekten vom Typ `T`, Verarbeitung “lazy”

- Neuen Stream anlegen: `Collection#stream()` oder `Stream.of()` ...
- Intermediäre Operationen: `peek()`, `map()`, `flatMap()`, `filter()`, `sorted()` ...
- Terminale Operationen: `count()`, `forEach()`, `allMatch()`, `collect()` ...
 - `collect(Collectors.toList())`
 - `collect(Collectors.toSet())`
 - `collect(Collectors.toCollection())` (mit `Supplier<T>`)
- Streams speichern keine Daten
- Intermediäre Operationen laufen erst bei Abschluss des Streams los
- Terminale Operation führt zur Verarbeitung und Abschluss des Streams

LICENSE



Unless otherwise noted, this work is licensed under CC BY-SA 4.0.