

Nachdenkzettel: Software-Entwicklung 2, Streams processing

1. Filtern sie die folgende Liste mit Hilfe eines Streams nach Namen mit „K“ am Anfang:
final List<String> names = Arrays.asList("John", "Karl", "Steve", "Ashley", "Kate");

```
final List<String> names = Arrays.asList("John", "Karl", "Steve", "Ashley", "Kate");
final List<String> filteredNames = names.stream().filter(name -> name.startsWith("K")).toList();
```

2. Welche 4 Typen von Functions gibt es in Java8 und wie heisst ihre Access-Methode?
Tipp: Stellen Sie sich eine echte Funktion vor (keine Seiteneffekte) und variieren Sie die verschiedenen Teile der Funktion.

Suppliers: functions that supplies data (no argument)
Consumers: functions that consumes data (no return value)
Predicates: boolean-valued function (one argument)
Operators: functions that receive and return the same value type

~ Stream()
~ println()
~ filter()

3. forEach() und peek() operieren nur über Seiteneffekte. Wieso?

They have no return value but perform an action (→ side effect)

example: set-methods ~ changing the object's internal state

4. sort() ist eine interessante Funktion in Streams. Vor allem wenn es ein paralleler Stream ist. Worin liegt das Problem?

*Parallel Streams need to be sorted, if array too long
it is sorted differently which might cause unexpected behavior*

5. Achtung: Erklären Sie was falsch oder problematisch ist und warum.

a) Set<Integer> seen = new HashSet<>();

someCollection.parallel().map(e -> { if (seen.add(e)) return 0; else return e; })

*→ The function has a side effect
→ could lead to concurrency issues*

b) Set<Integer> seen = Collections.synchronizedSet(new HashSet<>());

someCollection.parallel().map(e -> { if (seen.add(e)) return 0; else return e; })

bad style

6. Ergebnis?

List<String> names = Arrays.asList("1a", "2b", "3c", "4d", "5e");

names.stream()

.map(x → x.toUpperCase())

No output

.mapToInt(x → x.toInt(1))

.filter(x → x < 5)

Wenn Sie schon am Grübeln sind, erklären Sie doch bei der Gelegenheit warum es gut ist, dass Streams „faul“ sind.

The processing ends once the terminal function is done

→ else it would continue after finding the first match

(which we don't want) → no unnecessary iterations after task is fulfilled

7. Wieso braucht es das 3. Argument in der reduce Methode?

```
List<Person> persons = Arrays.asList(  
    new Person("Max", 18, 4000),  
    new Person("Peter", 23, 5000),  
    new Person("Pamela", 23, 6000),  
    new Person("David", 12, 7000));  
  
int money = persons  
    .parallelStream()  
    .filter(p -> p.salary > 5000)  
    .reduce(0, (p1, p2) -> (p1 + p2.salary), (s1, s2) -> (s1 + s2));  
  
log.debug("salaries: " + money);
```

→ to access the actual salary of the two objects we're comparing.
(else we would just be comparing the objects themselves ↗ error)

Tipp: Stellen Sie sich eine Streamsarchitektur vor (schauen Sie meine Slides an). Am Anfang ist eine Collection. Sie haben mehrere Threads zur Verfügung. Mit was fangen Sie an? Dann haben die Threads gearbeitet. Was muss dann passieren?

8. Was ist der Effekt von stream.unordered() bei sequentiellen Streams und bei parallelen streams?

9. Fallen

a) IntStream stream = IntStream.of(1, 2);
stream.forEach(System.out::println);
stream.forEach(System.out::println);

parallel streams would change the order of the list while sequential keeps the initial order

b) IntStream.iterate(0, i -> i + 1)
.forEach(System.out::println); ↗ eternal Stream

Stream has already been used and can't be called another time

c) IntStream.iterate(0, i -> (i + 1) % 2)
.distinct()
.limit(10)
.forEach(System.out::println);

//.parallel()?

d) List<Integer> list = IntStream.range(0, 10)
.boxed()
.collect(Collectors.toList());

list.stream()
.peek(list::remove)
.forEach(System.out::println); ~ concurrent modification exception

from: Java 8 Friday: <http://blog.jooq.org/2014/06/13/java-8-friday-10-subtle-mistakes-when-using-the-streams-api/>