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”);

```
public class Exercise1 {

    public static void main(String[] args) {
        final List<String> names = Arrays.asList("John", "Karl", "Steve", "Ashley", "Kate");
        names.stream().filter(s -> s.startsWith("K")).forEach(s -> System.out.println(s));
    }
}
```

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.

Predicate, Access-Method: boolean valued function

consumer , Access-Method: function that consums data

supplier, Access-Method: function that supplys data

operators, Access-Method: receives and returns the same data type

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

Both methods return void. If they want to operate in some way they have to create some sort of side effect. They are meant to produce side effects.

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

You need to use forEachOrdered() at the end instead of sort() if it is a prallel stream because forEachOrdered() will process the elements in the order specified by the stream. There is no guaranteed order in a parallel stream.

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; })

HashSet is not thread safe.

b) Set<Integer> seen = Collections.synchronizedSet(new HashSet<>());
someCollection.parallel().map(e -> { if (seen.add(e)) return 0; else return e; })

Is threadsafe, but the method is unclear for what it is intended for.

6. Ergebnis?
List<String> names = Arrays.asList(“1a”, “2b”, “3c”, “4d”, “5e”);
names.stream()
.map(x → x.toUppercase())
.mapToInt(x → x.pos(1)
.filter(x → x < 5)

Output: 1 2 3 4

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

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);
```

```
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?
```

You need the 3. argument is needed to combine the aggregated values.

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

There is no difference in the effect if the number of objects is small.

9. Fallen

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

Creates an error: You need to create a new stream

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

Its a infinite stream. To solve this you need to use skip or limit.

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

infinite stream → It tries to produce more elements then it can. It can only produce 0 or 1.

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

        list.stream()
        .peek(list::remove)
        .forEach(System.out::println);
```

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

There is a side effect and its list::remove.