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”);

name.stream.startsWith(„K“)

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.

Functions with return value
Functions without return value
Functions with arguments
Functions without arguments

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

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

Das Problem liegt darin, dass bei parallelstreams bei sort eine falsche Reihenfolge herauskommen kann, da die parallel laufenden Streams zu unterschiedlichen Zeiten fertig werden können.

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; })

Problem: Ergebnis könnte falsch sein, da map parallel aufgerufen wird

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

lostupdate

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)
```

1A, 2B, 3C, 4D

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

Anstatt alles durchzuarbeiten, brechen Streams beim ersten Ergebnis ab, für die Performance

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?
```

Die Ergebnisse der Threads müssen wieder zusammengeführt werden

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

Seq: es kommt immer eine andere Reihenfolge raus

Par: ebenso

9. Fallen

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

bricht bei der ersten foreach schon ab

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

hört nicht auf, da .limit() fehlt

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

gibt immer nur 0 und 1 raus

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

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

peek() löscht alles, was es sieht gleich raus

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