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”);

List<String>knames = names.stream().filter(s -> return s.startsWith('K')).collect(Collectors.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.

Predicate: Parameter liefert True oder False
Funktion: Parameter liefert Wert
Consumer: Parameter keine Rückgabe
Producer: Keine Parameter, gibt Wert zurück

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

weil es immer genutzt wird um neue Listen oder Prints zu machen

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

Man muss immer auf alle Daten warten, kein Durchfluss möglich -> stau

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 ist nicht Threadsafe (hat Seiteneffekt und)

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

synchronised macht den Stream langsamer
eigentlich dann nicht parallel

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

keine Terminal Op -> kein Ergebnis weil noch im Stream

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

Wenn es keine Terminal Operation gibt, wird nichts gemacht. Also wird Rechenzeit gespart und Fehler schneller erkannt

Einzelne Werte werden durchgereicht. Hat man eine Funktion wie findFirst, kann der Stream den Rest einfach nicht bearbeiten und ist so schneller.

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 parallelen Streams müssen wieder zusammengefügt werden und das dritte Argument ist das Pattern dafür

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

Die Reihenfolge Unterscheidet sich bei Parallelen stark. (Bei mehrmaligen Ausführen kann das Ergebnis abweichen)

9. Fallen

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

nach dem ersten ForEach ist der Stream leer

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

dank IntStream wird es zur Endlosschleife, da einfach alle Interger genommen werden und auch bei Overflow nicht gestoppt wird.

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

//.parallel()?

Es kann beim iterate nur 1 und 0 zurückgegeben werden
distinct filtert dann jede 1 und 0 raus.
Das Limit 10 wird nie erreicht
=> endlos

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

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

Seiteneffect im debug. Greifen direkt auf die Ausgangscollection zu

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