Exercises:

1. Explain following code snippet. How does the combination of *Stream.of()* and *Stream.flatMap()* work in this snippet?

```
Stream<String> namesStream = Stream.of("Ashley", "Lisa", "Samuel", "Pat", "Marion");
Stream<String> namesStream2 = Stream.of("Alex", "Peter", "Julian", "Martin", "Ruth");
Stream<String> namesStream3 = Stream.of("Felix", "Olivia", "Phillip", "Viola", "David");
Stream<String> allNames = Stream.of(namesStream, namesStream2, namesStream3).flatMap(Function.identity());
```

2. Create an own *Collector*, which processes *String*s.
   a) It should return an object of the new UDT *StringStatistics*.
   b) *StringStatistics* should hold the count of encountered *String*s and store the min and max *String*.
   c) The way *String*s are compared (What is min and max?) should be configurable from outside.
   d) Provide example-usages with Unit Tests.
      1. Also add tests with text files (e.g. from https://introcs.cs.princeton.edu/java/data/)!
         Play around with very long text files and parallelization (Check the results for plausibility!). When is there a performance gain?
3. Rewrite the *String* processing explained above but use *Stream.reduce()*, i.e. no *Collector*.
   a) Write some words (also your personal opinion is appropriate to mention) about the differences between using *Stream.collect()* and *Stream.reduce()*.

Remarks:

- As always.