Exercise 07: Streams

Deadline: 12. 05. 2016, 10:00

Name: _____ **Student number**: _____

CS: ☐ G1 (Grimmer) ☐ G2 (Prähofer) ☐ G3 (Prähofer)

WIN: ☐ G1 (Khalil) ☐ G2 (Khalil) ☐ G3 (Hummel)

Exercise	**Points**				
Übung	24	Java program, Output of Tests		☐	

Text Analysis (24 points)

In this exercise, we want to use the new Streams of Java 8 for the analysis of a text. The text should be read from a file with the method `lines` of the class `Files` from package `java.nio`:

```
static Stream<String> lines(Path path) throws IOException
```

For example, you can read the text of a file with name `"sampletext.txt"` as follows:

```
java.nio.Files.lines(java.nio.Paths.path("sampletext.txt"))
```

In other words, the `lines` method returns a `Stream<String>` containing the lines of the file. Starting from this Stream, text analysis should be implemented as follows.

(a) Lines, where a text is present (4 points)

Implement a method

```
static Optional<String> findLine(String filename, String text)
```

and a method

```
static List <String> findLines(String filename, String text)
```

which determines the first or all rows that contain the specified text.

(b) Line numbers where a text is present (4 points)

Analogous to the methods from (a) you now implement the search methods

```
static Optional<Integer> findLineNumber (String filename, String word)
static List<Integer> findLineNumbers (String filename, String word)
```

which determine the line numbers (instead of rows).

Notes:

- To solve the problem, you need to give numbers to rows. You should do this with a mapping (`map`), where each line is mapped to a pair of line and line number. You can do this using the class `Pair` from the download.
- To generate the line numbers, you need a counter. Usually you would do this with a local variable. However, in Lambda expressions you cannot use local variables. Consider a workaround.

(c) All words (4 points)

Implement a method

```
static List<String> words(String filename)
```

which creates a list of all words in the order as they occur (duplicates allowed). Others ("no-words"), e.g. numbers, should not be taken into the result list.

Note: Split the line into words and then unite to a flat list with `flatMap`. Use `string.split ("[ .,;?!.:()]'').`

(d) Occurrences of words (4 points)

Implement a method

```
static Map<String,Integer> wordOcurrences(String fileName)
```

which determines how often a word occurs in the text. In the result table, for each word the number of times the word is found in the text should be stated. The analysis should be case insensitive (upper/lower case letters are dealt with similarly).

Note: Starting from the stream of all the words in the text you can summarize the words with `groupingBy` and then count the occurrences.

(e) Set of words (4 points)

Implement a method

```
static Set<String> setOfWords (String fileName)
```

which determines the set of words. This means that each word occurs only once (in the set), again case-insensitive. In addition, the set of words should be sorted.

Test the methods with the text "sampletext.txt".

There's a test application in the download we provides named `SampleTextAnalysisApp` that applies the methods to the testfile `"sampletext.txt"` . Test your program with this application. Then run a test on a large text file, for example on the file `"faust_1.txt"`.

General notes:

- You should implement each method as a cascade of stream methods whenever possible, roughly according to the following scheme

  ```
  Files.lines(...)
        .streamMethod1(...)
        .streamMethode2(...)
        . …
        .materializeMethod(...);
  ```

  where at the end of the chain a method is called that materializes the result stream, i.e., it creates a concrete result in the form of a Collection, Map or a scalar value.
- Each method should be implemented individually and not rely on existing methods.