Übung 06: Streams

Abgabetermin: 30. 4. 2015, 8:15

Name: _____ **Matrikelnummer**: _____

Informatik: ☐ G1 (Prähofer) ☐ G2 (Prähofer) ☐ G3 (Grimmer) ☐ G4 (Grimmer)

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

Aufgabe	Punkte	abzugeben schriftlich	abzugeben elektronisch	korr.	Punkte
Übung 6	24	Java-Programm, Ausgabe der Tests	Java-Programm	☐	

Textanalyse (24 Points)

In dieser Übung wollen wir die neuen Streams von Java 8 für die Analyse eines Textes einsetzen. Der Text soll dabei mit der Methode `lines` der Klasse `Files` aus Package `java.nio` von einer Datei eingelesen werden:

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

Zum Beispiel kann man den Text einer Datei mit Namen `"sampletext.txt"` wie folgt lesen:

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

Das heißt, die Methode `lines` liefert einen `Stream<String>` mit den Zeilen der Datei. Ausgehend von diesem Stream sollen Textanalysen wie folgt implementiert werden.

(a) Zeilen, in denen ein Text vorkommt (4 Punkte)

Implementieren Sie eine Methode

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

und eine Methode

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

welche die erste bzw. alle Zeile(n) ermittelt, die den gegebenen Text beinhalten.

(b) Zeilennummern, in denen ein Text vorkommt (4 Punkte)

Analog zu den Methoden aus (a) implementieren Sie nun Suchmethoden

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

die als Ergebnis statt der Zeilen die Zeilennummern ermitteln.

Hinweise:

- Um die Aufgabe zu lösen, müssen Sie die Zeilen nummerieren. Das sollten Sie in einem Mapping (`map`) machen, wobei jede Zeile in ein Paar mit Zeile und Zeilennummer abgebildet wird. Sie können dazu die Klasse `Pair` aus dem Download verwenden.
- Um die Zeilennummern zu generieren, müssen Sie beim Mapping einen Zähler mitführen. Dies würde man normalerweise mit einer lokalen Variablen machen. In Lambda-Expressions können Sie aber lokale veränderliche Variablen nicht verwenden. Überlegen Sie sich einen Workaround.

(c) Alle Wörter (4 Punkte)

Implementieren Sie eine Methode

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

welche eine Liste aller Wörter in der Reihenfolge wie sie vorkommen (auch doppelt) erzeugt. Nicht-Worte, z.B. Zahlen, sollen nicht in die Ergebnisliste übernommen werden.

Hinweis: Sie sollen die Zeilen in Wörter splitten und dann mit `flatMap` zu einer flachen Liste vereinigen.

(d) Vorkommen der Wörter (4 Punkte)

Implementieren Sie eine Methode

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

welche ermittelt, wie oft ein Wort im Text vorkommt, d.h. die Ergebnistabelle stellt für jedes Wort dar, wie oft das Wort im Text vorkommt. Dabei soll Groß- und Kleinschreibung nicht berücksichtigt werden.

Hinweis: Ausgehend vom Stream aller Wörter im Text können Sie die Wörter mit `groupingBy` zusammenfassen und dann die Vorkommen zählen.

(e) Menge aller Wörter (4 Punkte)

Implementieren Sie eine Methode

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

welche die Menge aller Wörter ermittelt, d.h., die Wörter kommen in der Menge nur mehr einmal vor. Groß- und Kleinschreibung soll dabei keinen Unterschied machen. Des Weiteren sollte die Menge der Wörter sortiert sein.

(f) Gruppierung der Wörter (4 Punkte)

Implementieren Sie eine Methode

```
static Map<Character, List<String>> groupByFirstChar(Path path)
```

sowie eine Methode

```
static Map<Character, List<String>> groupByFirstChar(Path path)
```

die die Wörter nach dem ersten Buchstaben bzw. nach der Länge gruppieren.

Testen Sie die Methoden indem Sie die Analysen für den Beispieltext aus der bereitgestellten Datei „sampletext.txt“ testen.

In der Programmvorgabe findet man eine Testapplikation `SampleTextAnalysisApp`, die die Methoden auf die Testdatei `"sampletext.txt"` anwendet. Testen Sie Ihr Programm mit der Applikation. Führen Sie dann einen Test auf eine umfangreiche Textdatei aus, z.B. auf den in der Datei `"faust_1.txt"`.

Allgemeine Hinweise:

- Sie sollen jede Methode möglichst als eine Kaskade von Stream-Methoden implementieren, etwa nach folgendem Schema

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

  wobei am Ende der Kette eine Methode ist, die den Ergebnis-Stream materialisiert, d.h., ein konkretes Ergebnis in der Form einer Collection, Map oder einem skalarem Wert erstellt.
- Jede Methode sollte einzeln implementiert werden und nicht auf bereits vorhandene Methoden zurückgreifen.