

June 20<sup>th</sup>, 2022

## Sheet 09 – IO-Streams, Refactoring

### Exercise 1 (Duplicate Detection)

[4 Points]

Duplicate detection<sup>1</sup>, also known as record linkage, is a still challenging research topic. The task is how to identify pairs of records that are duplicates, and how to do it effectively and efficiently. To face this challenge you have to write a program.

- a) Write a program that detects duplicate restaurants in the `restaurantOLAT.csv` file, which provides a list of restaurants with their attributes, such as id, name, address, etc. The similarity score between two restaurants (A and B) can be computed by calculating the string similarity between each pair of string-attributes (i.e. do not compare the restaurants id), for instance, by comparing `restaurantA.name` and `restaurantB.name`, and finally normalizing the value over the number of compared pairs. For this exercise, implement the Levenshtein Distance<sup>2</sup> function.

#### Hint



For instance, for calculating the similarity score between the SVEN and JEAN values, you should count the minimum number of edits, which can be replace, insert, or delete a character, needed to match SVEN to JEAN.

Source (s):	S	V	E	N
Target (t):	J	E	A	N

Edit :            Replace   Replace   Replace   No edit

Levenshtein Distance (d): 3 (Number of edits)

$$\text{sim}(s, t) = 1 - \frac{d}{\text{maxLength}(s, t)} = 1 - \frac{3}{4} = 1 - 0.75 = 0.25$$

- b) The program should compute a similarity score between 0 and 1 for the compared restaurants. The total similarity score is computed by computing the arithmetic mean of all the attributes similarity scores. Furthermore, the program should define a threshold above which detected duplicates are considered "real" duplicates, e.g., threshold: 0.75. Then, the program filters and stores the resulting duplicates into the `Result.csv` file in the format `<restaurantId1>, <restaurantId2>, <totalSimilarityScore>`.

<sup>1</sup>[https://en.wikipedia.org/wiki/Record\\_linkage](https://en.wikipedia.org/wiki/Record_linkage)

<sup>2</sup>[https://en.wikipedia.org/wiki/Levenshtein\\_distance](https://en.wikipedia.org/wiki/Levenshtein_distance)

- c) Now check your implementation of the method that computes the similarity score by providing adequate JUnit-tests.

### Submit



```
at/ac/uibk/pm/gXX/zidUsername/s09/e01/Application.java
at/ac/uibk/pm/gXX/zidUsername/s09/e01/Test.java
at/ac/uibk/pm/gXX/zidUsername/s09/e01/Result.csv
at/ac/uibk/pm/gXX/zidUsername/s09/e01/*.java
```

## Exercise 2 (Detect the language of a given text)

[4 Points]

The idea of detecting the language of an input text is based on the detection of the words in the text. One main task is to detect commonly used or distinctive words in a specific language. In this context, Google has been creating a repository that contains wordlists of more than 1000 language varieties, and each of them contains 1000 distinctive words. Read more at Google Research Dataset<sup>3</sup>.

- a) Write a program that, given an input text via command line, computes the probability of whether the text is of a certain language. For doing that, the program should load the `qu.txt` and `ay.txt` <sup>OLAT</sup> wordlists files that represent the Quechua<sup>4</sup> and Aymara<sup>5</sup> languages respectively.
- b) Then, via the command line, the program asks for an input text, and the user enters a text, e.g., "quechua rimayta pisi wayna yachan". Afterwards, the program evaluates the input text against the loaded language wordlists, e.g., checking whether the words are contained within the different language wordlists.

### Hint



Calculate the probability of the words belonging to a language-wordlist, for instance, given the string "quechua rimayta pisi wayna yachan":

Input	quechua	rimayta	pisi	wayna	yachan
Quechua (QU)	1	1	1	1	1
Aymara (AY)	0	0	1	1	0

QU: 5/5 words are quechua words => 1.00



AY: 2/5 words are aymara words => 0.40

- c) Finally, the program shows via command line a value between 0 to 1, which is the confidence score obtained for the specific languages, e.g., QU: 1.00, AY: 0.40. Note that your program should handle the errors and exceptions adequately.

<sup>3</sup><https://github.com/google-research-datasets/TF-IDF-IIF-top100-wordlists>

<sup>4</sup>[https://en.wikipedia.org/wiki/Quechuan\\_languages](https://en.wikipedia.org/wiki/Quechuan_languages)




<sup>5</sup>[https://en.wikipedia.org/wiki/Aymara\\_language](https://en.wikipedia.org/wiki/Aymara_language)

**Submit** at/ac/uibk/pm/gXX/zidUsername/s09/e02/Application.java at/ac/uibk/pm/gXX/zidUsername/s09/e02/\*.java

### Exercise 3 (Refactoring)

**[2 Points]**

The goal of these exercises is to apply your gained knowledge about refactoring. Get familiar with the code provided in the Game.java<sup>OLAT</sup> and Player.java<sup>OLAT</sup> files (i.e. a simple rock-paper-scissors<sup>6</sup> game is implemented.) and refactor them, note that the methods should not be longer than 15 lines of code.

**Submit** at/ac/uibk/pm/gXX/zidUsername/s09/e03/Game.java at/ac/uibk/pm/gXX/zidUsername/s09/e03/Player.java at/ac/uibk/pm/gXX/zidUsername/s09/e03/\*.java

**Important:** Submit your solution to OLAT and mark your solved exercises with the provided checkboxes. The deadline ends at 6:00 pm (18:00) on the day before the discussion.

<sup>6</sup>[https://en.wikipedia.org/wiki/Rock\\_paper\\_scissors](https://en.wikipedia.org/wiki/Rock_paper_scissors)