

# **Optimization Assignment Thing**

Henning W.

14/04/2021

# *Contents*

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Bottlenecks and fixes</b>	<b>5</b>

# 1. *Introduction*

This small revolves around the optimization of self-chosen project. One also had the option of choosing a project, which one of the teachers had chosen. Due to the fact, that most of my projects utilize a multitude of libraries, in which the logic is implemented inside said libraries, the actual potential of me personally optimizing those projects is limited. Because of this, the project "letterfrequencies", which was supplied by the teachers, has been chosen for this assignment.

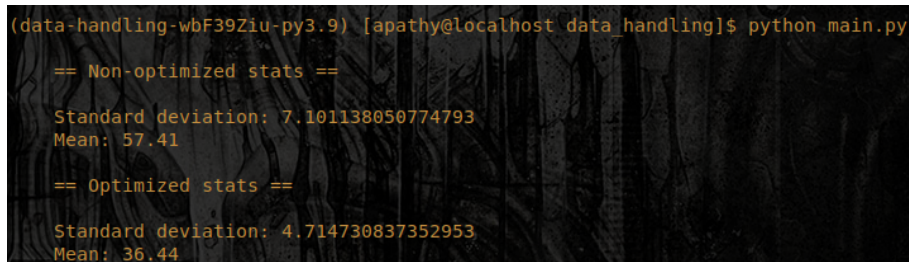
This program was designed to iterate through a file, and print out the sum of each occurrence of each separate letter.

Since an explanation of which part was to be optimized in the code, I should clarify, that most of the code has been changed in more ways, than just speed. Therefore the program will have more than a few changes. But the main changes revolves around changing the `FileReader` to a `BufferedReader`, and to limit the amount of times the program iterates through hashmaps. Disclaimer - The old Java file is not entirely pristine. This is largely due to the fact, that I had to print the csv files.

Documentation of old and new performance can be found in `old_data.csv` and `data.csv` respectively - Both inside the resources folder.

I didn't use any special software for determining the problems. Just my brain.

Since documentation of mean and standard deviation values were also recommended, a small python program has been created. The program converts both files into pandas dataframes and utilizes the inbuilt pandas functions to find the mean and standard deviation values of both the optimized and non-optimized versions. This program can be run by the user if wanted, or a screenshot can be found in `./resources/results_screenshot.png`



```
(data-handling-wbF39Ziu-py3.9) [apathy@localhost data_handling]$ python main.py

== Non-optimized stats ==

Standard deviation: 7.101138050774793
Mean: 57.41

== Optimized stats ==

Standard deviation: 4.714730837352953
Mean: 36.44
```

Figure 1.1: Screenshot of results

## 2. *Bottlenecks and fixes*

The largest change in speed can be simply explained by this quote from <https://www.geeksforgeeks.org/difference-between-bufferedReader-and-FileReader-in-java/>:

A buffer is a small portion of the device memory storage used to temporarily store some amount of data. Usually, Buffers use the RAM of the device to store the temporary data, and hence, accessing data from the buffer is much faster than accessing the same amount of data from the hard drive.

As such the FileReader was changed

```
1 Reader reader = new FileReader(RESOURCES_DIRECTORY+FILE_NAME);
```

To

```
1 private static BufferedReader getFileFromResources(String fileName)
  throws FileNotFoundException {
2     return new BufferedReader(new FileReader(RESOURCES_DIRECTORY +
  fileName));
3 }
```

Largest change code wise was deleting this method entirely:

```
1 private static void print_tally(Map<Integer, Long> freq) {
2     int dist = 'a' - 'A';
3     Map<Character, Long> upperAndlower = new LinkedHashMap();
4     for (Character c = 'A'; c <= 'Z'; c++) {
5         upperAndlower.put(c, freq.getDefault(c, 0L)
6             + freq.getDefault(c + dist, 0L));
7     }
8     Map<Character, Long> sorted = upperAndlower.entrySet()
9         .stream()
10        .sorted(Collections.reverseOrder
11            (Map.Entry.comparingByValue()))
12        .collect(
13            toMap(Map.Entry::getKey, Map.Entry::getValue,
14                (e1, e2) ->
15                e2, LinkedHashMap::new));
16    for (Character c : sorted.keySet()) {
17        System.out.println(" " + c + ": " + sorted.get(c));
18    }
19 }
20 }
```

And changing this method:

```
1 private static void tallyChars(Reader reader, Map<Integer, Long>
2   freq) throws IOException {
3     int b;
4     while ((b = reader.read()) != -1) {
5       try {
6         freq.put(b, freq.get(b) + 1);
7       } catch (NullPointerException np) {
8         freq.put(b, 1L);
9       }
10    }
11 }
```

To this:

```
1 private static Map<Character, Integer> createMapOfCharacters
2   (Reader reader) throws IOException {
3     Map<Character, Integer> mapOfCharacters = new HashMap<>();
4     int b;
5     while ((b = reader.read()) != -1) {
6       Character letter = Character.toUpperCase((char) b);
7       if (letter >= 'A' && letter <= 'Z')
8         if (mapOfCharacters.containsKey(letter)) {
9           mapOfCharacters.put
10             (letter, mapOfCharacters.get(letter) + 1);
11         } else {
12           mapOfCharacters.put(letter, 0);
13         }
14     }
15     return mapOfCharacters;
16 }
```

This means, that the program no longer populates the hashmap with values and then iterates through it to reverse the order and collects it in a new hashmap. Instead it just puts the values in in the correct order to begin with.

Note that we no longer use longs. Integers in Java are 32bit. That's  $2^{32}$  or 4294967296. There won't be that many instances of the same character at all in this document. Longs are 64bit. In other words completely unnecessary.

Apart from that, optimizations which aren't related to speed were made. As a rule of thumb code is for humans. Because of this, all my variables and method names are self-explanatory.