COMP 305 -Programming Assignment 1

Part 1, Part 2 & Part 3

Outline

Create a Java application that reads a paragraph of text from an input file, counts the number of occurrences of each word in the paragraph, and produces an html file displaying a table of words and word counts.

Step 1 – Create Java Project

- Use VS Code to create a Java project. Use your last name as the project name.
- Create a **res** folder in the main project folder.

Step 2 – Create Input File

- Create a text file named **words.txt** in the **res** folder. Add a paragraph of text to the file.
- To quickly create sample text, install a **Lorem Ipsum Generator** extension to VS Code. Once installed, the extension can be used to create one or more paragraphs of Lorem Ipsum text. (To generate the text, open View Menu, select Command Palette and type Lorem.) Alternatively, there are many Lorem Ipsum text generators available online.

Step 3 – Initialize Git Repo and Commit

- After the project and text file have been created, initialize a local Git repo.
- Create a **.gitignore** file in the main project folder and add the line ***.class** to the file.
- Commit project files to repo with "Initial Commit" message.

Step 4 – Read Input File

- Add code to read the contents of the **words.txt** file, separate it into words (removing all blanks and punctuation), and store the words in an ArrayList.

Step 5 – Count Word Occurrences

- Add code to create a HashMap that stores the number of occurrences of each word that appears in the original text file. The keys to the HashMap should be the words from the text file. The values returned should be the number of occurrences of the key word in the text file.

Step 6 – Create Output HTML File

- Add code to create an HTML file named **words.html** that contains a table consisting of a word column and a wordcount column.
- To complete this step you will need to iterate through the HashMap. Assuming the HashMap is named **wordCount**, the following code will iterate through all the keys.

```
for (String key : wordCounter.keySet()) {

        // Code to populate the table goes here

}
```

Step 7 – Test and Commit

- Test your code.
- At the end of the lab, commit your project files to the local repo with a message listing which of the above steps were completed successfully.

Sample Input and Output

<table>
<tr><th>Text Input</th><th>Wordcount HTML Output</th></tr>
<tr>
<td>Velit fugiat eu do aliqua est dolore excepteur. Ad anim exercitation magna commodo non exercitation qui amet et. Est laboris ad laborum Lorem. Veniam consequat culpa labore occaecat non id labore excepteur quis.
Ea incididunt sit Lorem aute. Commodo Lorem ullamco occaecat minim qui minim officia ut nisi consequat minim enim. Ullamco irure do reprehenderit occaecat anim elit eu fugiat reprehenderit excepteur eu et commodo. Irure elit mollit esse Lorem officia elit dolore mollit eiusmod.
Laborum occaecat non ullamco ullamco mollit ex nulla et enim eiusmod et culpa commodo. Aliquip voluptate anim anim in consequat. Cillum esse commodo proident consequat elit culpa aliquip eiusmod. Lorem labore nostrud irure adipisicing id eu aute nisi veniam tempor sint eiusmod sunt. Fugiat deserunt fugiat est cupidatat sunt excepteur. Dolore cillum ut non est anim Lorem. Est officia ut velit qui esse commodo cupidatat labore excepteur non tempor ea proident reprehenderit. Sint minim labore consectetur non ad qui dolor Lorem non ullamco.</td>
<td>
<table>
<tr><td>culpa</td><td>3</td></tr>
<tr><td>incididunt</td><td>1</td></tr>
<tr><td>mollit</td><td>3</td></tr>
<tr><td>tempor</td><td>2</td></tr>
<tr><td>est</td><td>5</td></tr>
<tr><td>aute</td><td>2</td></tr>
<tr><td>commodo</td><td>6</td></tr>
<tr><td>laborum</td><td>2</td></tr>
<tr><td>cupidatat</td><td>2</td></tr>
<tr><td>proident</td><td>2</td></tr>
<tr><td>do</td><td>2</td></tr>
<tr><td>lorem</td><td>7</td></tr>
<tr><td>officia</td><td>3</td></tr>
<tr><td>Cillum</td><td>2</td></tr>
<tr><td>Id</td><td>2</td></tr>
<tr><td>ea</td><td>2</td></tr>
<tr><td>sunt</td><td>2</td></tr>
<tr><td>ut</td><td>3</td></tr>
<tr><td>enim</td><td>2</td></tr>
<tr><td>occaecat</td><td>4</td></tr>
<tr><td>ad</td><td>3</td></tr>
<tr><td>consequat</td><td>4</td></tr>
<tr><td>velit</td><td>2</td></tr>
<tr><td>eiusmod</td><td>4</td></tr>
<tr><td>sint</td><td>2</td></tr>
<tr><td>esse</td><td>3</td></tr>
<tr><td>et</td><td>4</td></tr>
<tr><td>eu</td><td>4</td></tr>
</table>
Note: Table truncated
</td>
</tr>
</table>

Part 2

Step 8 – Create a WordFrequency Class

- Create a new class named WordFrequency to represent a word and its word count. Add a constructor, get and set methods, and a toString method.

- Based on the example given in Lab 6, modify the class so that it implements the **Comparable** interface. The class should support comparisons based on the word count.

Step 9

- Based on the sample code given in Lab 6, add code to the App class to create an ArrayList of WordFrequency objects.Populate the ArrayList with the data stored in the HashMap created in Step 5.

Step 10

- Sort the contents of the ArrayList in increasing order of the wordcount. Create a second HTML file called **sortedWords.html** that consists of a table of words and word counts, arranged in ascending wordcount order.

Step 11

- Add Javadoc comments describing all classes and methods in your project and their functionality.

Step 12

- At the end of part 2, commit your project files to the local repo with a message.

Part 3

Step 13 – Create Input File

- Create a text file named **paragraph.txt** in the **res** folder. Add a paragraph of text to the file.

Step 14 – Read Input File

- Add code to read the contents of the **paragraph.txt** file, separate it into words (removing all blanks and punctuation), and store the words in an ArrayList.

Step 15 – Count Word Occurrences

- Add code to create a HashMap that stores the number of occurrences of each word that appears in the original text file. The keys to the HashMap should be the words from the text file. The values returned should be the number of occurrences of the key word in the text file.

Step 16 – Create Output HTML File

- Add code to create an HTML file named **paragraph.html** that contains a table consisting of a two columns, one that has the word and the other that lists the count for that word

Step 17 – Create a ParagraphFrequency Class

- Create a new class named ParagraphFrequency to represent a word and its word count. Add a constructor, get and set methods, and a toString method. Modify the class so that it implements the **Comparable** interface. The class should support comparisons based on the word count.

Step 18

- Add code to the App class to create an ArrayList of ParagraphFrequency objects.Populate the ArrayList with the data stored in the HashMap created in Step 15.

Step 19

- Sort the contents of the ArrayList in increasing order of the wordcount. Create a second HTML file called **sortedParagraphWords.html** that consists of a table of words and word counts, arranged in ascending wordcount order.

Step 20

- Add Javadoc comments describing all the functionality you added in part 3.

Step 21

- At the end of part 3, commit your project files to the local repo with an appropriate message.

add note about how i made a new repo

Note:

The assignment is due by March 10, 2023. Submit a zip folder onto blackboard by the mentioned due date.

Before submitting, check that all the requirements are met. If you are not able to finish the assignment by the due date, still submit what you have for partial marks and leave a message with the submission.