

CMSC 315 Project #2: Word Frequency & Sentiment Analysis Program

In this project, you'll implement a set of basic **Natural Language Processing** (NLP) utility methods to analyze a paragraph of text entered by the user.

Starter Code Info

Download `project2_starter.zip` and extract the files. The zip contains two classes, `Main.java` and `NLPUtility.java`. You should be able to create a new Java project and copy the two classes into your project.

The `Main` class is fully implemented and takes care of user input. It calls the methods you'll implement in the `NLPUtility` class.

Predefined sets available in the `main` method:

- `stopWords`: A set of common words to exclude from frequency analysis.
- `positiveWords`: A set of positive sentiment words.
- `negativeWords`: A set of negative sentiment words.

You **should not change** any of the predefined sets. Instead, your task is to **complete all the static methods in `NLPUtility`** by replacing the current placeholder return statements (such as `return null`) with working code based on the method descriptions.

Tasks: Implement the following methods in the `NLPUtility` class

Task 1. `public static String[] splitTextIntoTokens(String text)`

Splits the text it into individual words, treating consecutive whitespace or punctuation characters as a single delimiter.

NOTE: The regular expression given in the Pearson textbook in section 21.6 is incorrect. The plus sign should follow the character class to match 1 or more white space or punctuation characters.

```
text.split("[\\s\\p{P}]+")
```

Example:

```
String[] tokens = NLPUtility.splitTextIntoTokens("WOW!?!    That .?#  
is REALLY(really) amazing!    ");  
System.out.println(Arrays.toString(tokens));  
// [WOW, That, is, REALLY, really, amazing]
```

Task 2. `public static TreeMap<String, Integer>
countFilteredWords(String[] words, Set<String> stopWords)`

Counts the frequency of non-stop words in the given array of words, ignoring case. Returns a **TreeMap** sorted alphabetically by key (i.e. word).

Example:

```
String[] words = {"i", "love", "a", "good", "BOOK", "and", "LOVE", "sad",  
"Book", "book"};  
Set<String> stopWords = new HashSet<>(  
    Arrays.asList("the", "is", "in", "at", "of", "and", "a", "to", "it",  
"or", "was", "so"));  
  
System.out.println(NLPUtility.countFilteredWords(words, stopWords));  
//{book=3, good=1, i=1, love=2, sad=1}
```

Task 3. `public static LinkedHashMap<String, Integer>
sortByValueDescending(Map<String, Integer> map)`

Returns a **LinkedHashMap** sorted by frequency in descending order. For ties, maintains the original order of keys as they appear in the map.

Algorithm:

1. Convert the word map entries to a list for sorting
2. Sort the list of entries in descending order based on value (frequency)
3. Create a LinkedHashMap and insert the sorted entries to maintain their order.

Example:

```
Map<String, Integer> wordMap = new TreeMap<>();  
wordMap.put("book", 3);  
wordMap.put("good", 1);  
wordMap.put("i", 1);  
wordMap.put("love", 2);  
wordMap.put("sad", 1);  
  
System.out.println(wordMap); // {book=3, good=1, i=1, love=2, sad=1}  
  
System.out.println(NLPUtility.sortByValueDescending(wordMap)); // {book=3,  
love=2, good=1, i=1, sad=1}
```

Task 4. `public static String getSentimentFromFrequencies(Map<String,
Integer> wordMap, Set<String> positiveWords, Set<String>`

negativeWords)

Sums the total frequencies of words in the corresponding positive and negative word sets. Returns a summary string in the format "Positive: X, Negative: Y".

Example:

```
Map<String, Integer> wordMap2 = new LinkedHashMap<>();
wordMap2.put("book", 3);
wordMap2.put("love", 2); // positive
wordMap2.put("good", 1); // positive
wordMap2.put("i", 1);
wordMap2.put("sad", 1); // negative
System.out.println(wordMap2); // {book=3, love=2, good=1, i=1, sad=1}

Set<String> positiveWords = new HashSet<>(Arrays.asList("good", "great",
"happy", "love", "like"));
Set<String> negativeWords = new HashSet<>(Arrays.asList("bad", "terrible",
"horrible", "sad", "hate"));

System.out.println(NLPUtility.getSentiment(wordMap2, positiveWords,
negativeWords)); // Positive: 3, Negative: 1
```

Task 5. public static Map<String, Object> getWordsWithMaxFrequency(Map<String, Integer> wordMap)

Returns a map containing an alphabetically sorted list of words that appear most frequently in the given word map, along with the corresponding frequency.

Algorithm:

- Finds the maximum frequency value in the input map
- Collect a list of all words that occur with that frequency
- Sorts the list alphabetically
- Returns a new map with two entries having the following keys:
 - "words": a list of most frequent words, sorted alphabetically
 - "frequency": the maximum frequency as an integer

Note: The returned map contains two entries with **String** keys: "words" and "frequency".

- The value associated with "words" is a **List<String>** containing the most frequently occurring words.
- The value for "frequency" is an **Integer** representing the highest frequency found.

Because the values are of different types (**List<String>** and **Integer**), the method returns a map of type **Map<String, Object>**.

Example:

```
Map<String, Integer> wordMap3 = new LinkedHashMap<>();
wordMap3.put("good", 1);
wordMap3.put("i", 1);
wordMap3.put("love", 3);
wordMap3.put("book", 3);
wordMap3.put("sad", 1);
System.out.println(wordMap3); // {good=1, i=1, love=3, book=3, sad=1}

System.out.println(NLPUtility.getWordsWithMaxFrequency(wordMap3)); //
{words=[book, love], frequency=3}
```

Note that the map passed as a parameter may not be sorted by frequency or word. Your method will have to find the maximum frequency, along with all words that are mapped to that frequency.

Sample Program Flow

```
Enter a paragraph of text:
I really love a good book, and You REALLY love a sad movie. We both
ReALLY LOVE going for a walk!

Tokenized:
[I, really, love, a, good, book, and, You, REALLY, love, a, sad, movie,
We, both, ReALLY, LOVE, going, for, a, walk]

Word map sorted by key ascending:
book:1
both:1
for:1
going:1
good:1
i:1
love:3
movie:1
really:3
sad:1
walk:1
we:1
you:1

Word map sorted by value descending:
love:3
really:3
book:1
both:1
for:1
going:1
good:1
i:1
```

```
movie:1
sad:1
walk:1
we:1
you:1

Sentiment: Positive: 4, Negative: 1

Most frequent word(s): [love, really] (used 3 times)
```

Example: Empty or Non-Meaningful Input

```
Enter a paragraph of text:
S0 is.! It????

Tokenized:
[S0, is, It]

No valid words found.
```

Submitting your solution

You are to submit two files.

1. The first is a **.zip** file that contains all the source code for the project. The **.zip** file should contain only source code and nothing else, which means only the **.java** files. If you elect to use a package the **.java** files should be in a folder whose name is the package name. Every outer class should be in a separate **.java** file with the same name as the class name. Each file should include a comment block at the top containing your name, the project name, the date, and a short description of the class contained in that file.
2. The second is a Word document (PDF or RTF is also acceptable) that contains the documentation for the project, which should include the following:
 - A UML class diagram that includes all classes.
 - A test plan that includes test cases that you have created indicating what aspects of the program each one is testing.
 - Conduct unit tests for each method within the **NLPUtility** class. You may want to develop separate test classes (include "Test" in the class name, and/or place in a separate package) to individually call each method in isolation. Include screenshots that capture the result of your unit tests. Ensure your test cases sufficiently demonstrate each method returns a sorted result when sorting is required.
 - A short paragraph on lessons learned from the project.