TP08

Static/Inner Nested class in java: The objective of this particular task is to understand the concept of static/inner nested classes in java . In the TP assets folder, you will find the solution of **TP06 Part One** (classes) your task is to transfer them in a single java file. What is your remarks?

Note: Do not omitted this part

To learn more about the concept of Static/Inner classes:
https://docs.oracle.com/javase/tutorial/java/javaOO/nested.html

Java Collection

Part One (Basics):

1- Create a method that takes an **ArrayList** of integers as input and returns the sum of all even numbers in the list. Example Input: [1, 2, 3, 4, 5, 6] Example Output: 12
2- Create a method that takes a **LinkedList** of strings as input and returns a new **LinkedList** with all strings of length less than 5 removed. Example Input: ["Hello", "there", "world", "!", "Java"] Example Output: ["Hello", "world", "Java"]
3- Create a method that takes two **sets** of integers as input and returns a new **Set** containing all the common elements in both sets. Example Input: {1, 2, 3, 4, 5}, {4, 5, 6, 7, 8} Example Output: {4, 5}
4- Create a method that takes an array of integers as input and returns a **Map** containing the count of each integer in the array. Example Input: [1, 2, 3, 4, 5, 2, 3, 4, 5, 5] Example Output: {1=1, 2=2, 3=2, 4=2, 5=3}.

Part Two (Intermediate):

1. In the assets folder, there are two classes one contains a static table of the most used English words. The second class contains a method that read the above words and adds them to a list while removing the duplicates. The program also counts and prints the exclusion time, your objective is to improve it.
2. Write a method **reverseLinkedList** that takes a **LinkedList** of Integers and returns a new **LinkedList** with the elements in reverse order. You should not modify the original **LinkedList**.
3. Write a method **countWords** that takes words from English words class and returns a **Map** where each key is a unique word, and the value is the number of times that word appears. You should ignore case and punctuation when counting words.

Dr. Oussama SERHANE (o.serhane@esi-sba.dz)

4. Implement a method that receives a **Map** of Integers to Strings and returns a new **Map** with the same key-value pairs, but with the values sorted in alphabetical order.
5. Implement a method that returns the n most frequently occurring words in a given string. Use a Map to count the number of occurrences of each word in the string, and a **PriorityQueue** to efficiently get the top n elements by their count.
6. Implement a method that takes an **ArrayList** of integers and rearranges the elements so that all the even numbers come before the odd numbers. Use two **LinkedLists** to store the even and odd numbers, respectively, and then combine them into a single **ArrayList**.
7. Implement a method that takes a Set of strings and returns a **Map** where each key is a character and each value is a List of strings that start with that character. Use a **TreeMap** to automatically sort the keys alphabetically, and a **HashMap** to efficiently store the List values.

For each taks, you should implement a solution that uses the appropriate collection type (e.g., ArrayList, LinkedList, Set, or Map). You should also measure the execution time of your solution using the System.currentTimeMillis().

Part Three (Advanced**):** Building a Word Spelling Checker

In this part, you will build a word spelling checker with Java that checks if a word is spelled correctly or not. You will implement the spelling checker using the words provided in EnglishWords assets folder.

Step one: Check if a Word is Spelled Correctly

1. Create a **Dictionary** class that maps words to their definitions using a **HashMap**.
2. Create a **SpellChecker** class that checks if a word is spelled correctly or not by looking it up in the **Dictionary**.
3. Create a **Main** class to test your implementation by checking the spelling of various words taped by a user.

Step two: Find the Closest Matching Word for Misspelled Words <<**Optional**>>

1. Update the **SpellChecker** class to include a method that finds the closest matching word for misspelled words using the **Levenshtein distance algorithm** (the algorithm implementation is provide in assets).
2. Update the **Main** class to use the **getClosestMatch** method for each misspelled word to get the closest matching word from the **dictionary** (saying to the use do you mean this+ the closest word).

***Important*:**

The objective of this TP is not to solve all the tasks the objective is two folds:

- Be familiar with the collection framework syntax definition.
- Get the best practices of using which collection for a given problem.