Exercise 09: Lists, sets and maps in data processing

In this exercise, you will be using the interfaces `List<T>`, `Set<T>`, `Map<K, V>` as well as the inner interface `Map.Entry<K, V>`, all of which are grouped in `java.util`. Using `Iterator<T>`, which is provided by container classes that implement `Iterable<T>` (e.g. `LinkedList<T>`, `TreeSet<T>`), you will calculate statistics about tweets, using the example of a famous person.

Goal of this task

Initially, tweets (strings) will be indexed one by one, in order to count how often each word occurs. Based on these statistics, iterators will then be created to iterate through

- the entire vocabulary (alphabetically),
- the most common hashtags (in descending order, #...), and
- the most common words (in descending order, as buzzwords).

In a final step, an iterator will be created which iterates through all tweets and sorts them in descending order of "buzzword weighting".

From a technical point of view, the interface `TweetCollection` and the factory method `TweetCollection.create` must be implemented.

Information:

- In the official Java documentation you will find a list of the implementing classes (e.g. `LinkedList<T>` and `ArrayList<T>`, etc.) for each of the interfaces (e.g. `List<T>`).
- It is very informative to also read through the Javadoc on the other methods and interfaces that are used.
- In this exercise you do not (!) have to implement your own iterator; rather, you should create data structures that you sort and filter, and call `.iterator()` once you have the result.

Task 1: Class and factory method

Create a new class that implements `TweetCollection`, and change the static factory method `TweetCollection.create` to return a new instance of this new class.

Task 2: Index tweets

Implement the `ingest(String tweet)` method. This method accepts a string that can be converted to a `List<String>` using the given static method `TweetCollection.tokenise(String)`. The original tweet should then be stored in a list (`List<String>`). For each word, a counter should be maintained with how many times a word has already occurred in total.

You will also need these methods:

- `List<T>.add(T e)`: appends an element to a list
- `Map<K, V>.get(K key)`: returns the value for a key, or `null`
- `Map<K, V>.containsKey(K key)`: indicates whether there is a value for a key
- `Map<K, V>.put(K key, V val)`: sets the value (`val`) for a key (`key`)

Task 3: Simple iterators

Implement the following methods:

- `Iterator<String> vocabIterator()`: returns an iterator across all words, sorted alphabetically. Please note: use `Map.keySet()` and read the corresponding documentation.
- `Iterator<String> topHashTags()`: returns an iterator across all hashtags (i.e. words that begin with #), sorted in descending order of frequency. Please note: depending on the implementation, you will need a "helper list"; to store words and their frequency in such a list, you can use the class `org.apache.commons.lang3.tuple.Pair` (via `FactoryMethod Pair.of()`), for example.
- `Iterator<String> topWords()`: returns an iterator across all the words whose first letter is alphabetical (see `Character.isAlphabetic`), also in descending order of importance.

You will also need these methods:

- `List<T>.sort(Comparator<T> c)`: sorts a list with the given comparator.
- The `Map.Entry<K, V>` interface, which contains a `.getKey()` key and a `.getValue()` value; see also `Map.entrySet()`.
- `Pair.getLeft()` and `Pair.getRight`: returns the left or right element of a data pair respectively; `Pair<K, V>` implements `Map.Entry<K, V>`.

Task 4: Best tweets

Implement the `Iterator<Pair<String, Integer>> topTweets()` method, which returns an iterator that iterates through pairs of tweets and "buzzword frequency" (`Pair<String, Integer>`). The pairs should be sorted in descending order of buzzword frequency, where the buzzword frequency of a tweet is the sum of the frequencies of the words.

Please note: Depending on the implementation, you will need to create a helper list again here, which initially contains pairs of tweets and buzzword frequency (i.e. `Pair<String, Integer>`) that can then be sorted accordingly.

Task 5: Stop words

As you can see from the test results, common everyday words like "so" or "and" come to the fore, although these are obviously unimportant. Implement the `setStopwords(File)` method , which uses a `java.util.Scanner` to read words that should be ignored from a file (see `java.io.FileReader`). Read word for word into a separate `set<String>` and then customise the `ingest` and `topTweets` methods to ignore words contained in `Stopwordset`. To do so, use the `Set<T>.contains(T e)` method.