Homework 3 – Word Counting

Authors: Melanie, Anupama, SeoHyun, David, Nicolas, Lucas, Dongjae, Michael

Problem Description

This assignment will test your knowledge of static variables & methods, wrapper classes, String parsing, and JavaDocs. It is up to you how to achieve the functionality we ask for. Feel free to create all the helper methods that you need. Your program must utilize good **design principles**, **JavaDocs**, as well as **Checkstyle**.

In this HW, you will write a program that can help give statistics about text and writing. You'll write code to help count the total number of initial words that appear in sentences, count the length of the longest sentence in a short paragraph, and calculate the age of a certain piece of text.

Solution Description

Note: you must follow the **rules of encapsulation for all instance fields** of all classes you write.

`WordStatistics.java`

You will create a class called `WordStatistics.java`. You should be able to create many instances of this class, and **each** instance will have:

- its own `text` attribute, which will be of type `String`. This variable, `text`, will store a few sentences that will be parsed in other methods within `WordStatistics`.
- its own `history` attribute, which will be of type `String`. `history` stores the date in which this text was published in the format "`Published_18700118`" meaning year 1870, January 18.

Across **all** instances of this class, we also want to store/record:

- the total number of words ever counted by a `WordStatistics` instance via the method `countInitWords()`, which is described in more detail below. Name this variable `totalWordCount` and declare it with the correct modifier to ensure it will update for all instances of `WordStatistics`.
- the length of the longest sentence ever recorded by a `WordStatistics` instance. Name this variable `longestSentenceLength`. The length of a sentence will be defined by the number of words it contains. If the longest sentence ever found was "`The little dog laughed to see such sport, and the dish ran away with the spoon.`" Then `longestSentenceLength` should be set to `16` for **all** instances of `WordStatistics`.

`WordStatistics.java` will have the following constructor:

- two-argument constructor, that takes in `text` as the first parameter and `history` as the second parameter. Both are of type `String`.

`WordStatistics.java` will have the following methods:

(For all methods within `WordStatistics.java`, you may assume that sentences will **always** be separated by a period ('.'). That is, no other punctuation ending a sentence will appear, such as '?' or '!'. You can also assume that any character within a sentence that is not a letter is a comma (',').)

- `countInitWords()`
  - **takes in** an integer as a parameter that indicates how many initial unique words to count and **return** a `String[][]` where each subarray holds the word that was counted and its frequency.
  - The `int` parameter stores the number of unique initial words in `text` that will have their frequencies counted throughout `text`. The case (upper or lower) of the text should not differentiate one word from another. For example, if `text = "Hey diddle, diddle, the`

cat and the fiddle. The cow jumped over the moon. The little dog laughed to see such sport, and the dish ran away with the spoon." And the integer passed in was 5. Then `countInitWords()` should return the following array:

`[["hey", "1"], ["diddle", "2"], ["the", "7"], ["cat", "1"], ["and", "2"]]`

The initially recorded word should be stored in the returned array as its lowercase version. Make sure you are updating appropriate instance variables. The total words counted within this method in the example above is 1 + 2 + 7 + 1 + 2 = **13**. `countInitWords()` should **not print** anything.

- `longestSentence()`
  - should **return the length** of the longest sentence within that specific instance's `text` and **print** the longest sentence.
  - The length of a sentence will be defined by the numbers of words it contains. For example, if `text` = "Hey diddle, diddle, the cat and the fiddle. The cow jumped over the moon. The little dog laughed to see such sport, and the dish ran away with the spoon." Then `longestSentence()` should return `16` and print:

    "The little dog laughed to see such sport, and the dish ran away with the spoon."

    Make sure you are updating appropriate instance variables.

- `textAge()`
  - calculate and **return the age** of the text (in years) by subtracting the year in which it was published from 2022 and adding 1.
  - For example, if `history` = "`Published_18700101`", then `textAge()` should return `154` because 2023 – 1870 + 1. `textAge()` should **not print** anything.

`Driver.java`

You will also create a `Driver.java` file to test your code. Within the `main` method, you will need to use the `Scanner` class to get user input to set the `text` of a `WordStatistics` instance. Use the constructor you've created in `WordStatistics.java` to initialize a `WordStatistics` instance with the user inputted text. The output of the `main` method in `Driver.java` should be the following (assuming `text` is the same as above):

```
Set text: {user inputted text}              *{} should be omitted from the actual output
Enter the initial number of words: 5        *5 is a user inputted number
Word Counts:
hey 1
diddle 2
the 7
cat 1
and 2
Longest Sentence: The little dog laughed to see such sport, and the dish ran
away with the spoon.
Age of text: 154
```

You will need to call the methods you created in a specific order to get the desired output, as well as some iterative logic to correctly print a `String[][]`. You should also make sure you are checking the values of your `static` variables and that they only change when you expect them too. You are free to add additional output lines to facilitate testing all of your methods and variables, as long as it includes the output shown above.

Here are some methods that you may find useful throughout your code:

- `int Integer.parseInt(String s);`
- `String String.valueOf(int i);`
- In the `Scanner` class:
  - `String next();`
  - `boolean hasNext()`
- In the `String` Class:
  - `String toLowerCase();`

Even though you are free to implement your own program style, you should think carefully about the design of your program and its classes. To achieve the best score, **you should follow the principles of modularity and encapsulation in your design**. Try not to overload any one class or method with too much to do.

Submitting

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- `WordStatistics.java`
- `Driver.java`
- Additional classes that you created (if any)

Make sure you see the message stating, "Homework 3 submitted successfully." We will only grade your last submission be sure to **submit *every file* each time you resubmit.**

Checkstyle and Javadoc

You must run Checkstyle on your submission. (To learn more about Checkstyle, examine our course's external website Java Resources page under "CS 1331 Style Guide".) The Checkstyle cap for this assignment is **15 points**. If you don't have Checkstyle yet, download it from Canvas →Files →Resources →checkstyle-8.28.jar. You can also find it on the course website under "Java Info." Place it in the same folder as the files you want to run Checkstyle on. Run checkstyle on your code like so:

```
$ java -jar checkstyle-8.28.jar yourFileName.java Starting
audit...
Audit done.
```

The message above means there were no Checkstyle errors. If you had any errors, they would show up above this message, and the number at the end would be the points we would take off (limited by the Checkstyle cap mentioned above). The Java source files we provide contain no Checkstyle errors. In future assignments, we will be increasing this cap, so get into the habit of fixing these style errors early!

Additionally, you must Javadoc your code.
Run the following to only check your Javadocs:

```
$ java -jar checkstyle-8.28.jar -j yourFileName.java
```

Run the following to check both Javadocs and Checkstyle:

```
$ java -jar checkstyle-8.28.jar -a yourFileName.java
```

Import Restrictions

You may only import `Scanner`.

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our auto grader. For that reason, do not use any of the following in your final submission:

- `var` (the reserved keyword)
- `System.exit`

Collaboration

No collaboration is allowed on this assignment. See syllabus for more details.

In addition, note that it is not allowed to upload your code to any sort of public repository. This could be considered an Honor Code violation, even if it is after the homework is due. Only post code on Ed Discussion in a **private** post.

Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items
- Do not submit `.class` files.
- Test your code in addition to the basic checks on Gradescope
- Submit every file each time you resubmit
- Read the "Import Restrictions" and “Feature Restrictions" to avoid losing points
- Check on Ed Discussion for all official clarifications
- Make sure to test your program manually based on the assignment instructions and expected outputs.

The Gradescopeautograder visible to you is **NOT** comprehensive. A few test cases will be hidden when you submit. This is done intentionally so that you learn to write your own test cases for your assignments.