Individual Project:
TextSummary Utility -- Deliverable 1

Project Goals

- Develop a Java application for analyzing text within a file.
- Get experience with an agile, test-driven process.

Details

For this project, you must develop, using the Java language, a simple **command-line** utility called *textsummary*, which is the same utility for which you developed test frames in the category-partition assignment. A concise specification for the `textsummary` utility was provided with that assignment and is repeated here for your convenience:

Concise Specification of the `textsummary` Utility

- NAME:
  `textsummary` - helps to analyze a text file.

- SYNOPSIS
  `textsummary OPT <filename>`

  where `OPT` can be **zero or more** of
  - `-c <string>`
  - `-d [integer]`
  - `(-l|-s) [integer]`
- COMMAND-LINE ARGUMENTS AND OPTIONS

  `<filename>`: the file on which the textsummary operation has to be performed.

  `-c <string>`: if specified, the `textsummary` utility will add the number of times the provided `<string>` appears in the line to the start of each line in the file.

  `-d [integer]`: if specified, the `textsummary` utility will output the words (where a word is any sequence of alphanumeric characters) which are duplicated the most times throughout the file. By default it will output the single most common word. If a positive integer is provided in the optional parameter [integer], the utility will output that number of the most common words, starting

from the most common. This option will always be executed first.

`(-l|-s) [integer]`**:** if specified, the `textsummary` utility will keep only the longest(-l) or shortest(-s) [integer](a required positive integer) lines in the file. Options -l and -s are mutually exclusive.

If **none** of the OPT flags is specified, `textsummary` will simply output the longest line in the file.

- NOTES
  - While the last command-line parameter provided is always treated as the filename, OPT flags can be provided in any order; though no matter the order of the parameters, if provided, -d will be applied first.

- EXAMPLES OF USAGE

  Example 1:
  `textsummary file1.txt`
  File content:
  3 dogs
  a cat

  Outputs "3 dogs".

  Example 2:
  `textsummary -d 2 -c "d" file1.txt`
  File content:
  dog bird cat cat
  cat dog fish
  Result file:
  2 dog bird cat cat
  1 cat dog fish

  Outputs "cat dog"

  Example 3:
  `textsummary -l 1 file1.txt`
  File content:
  dog bird cat cat
  cat dog fish
  Result file:
  dog bird cat cat

  Example 4:
  `textsummary -d -s 2 file1.txt`
  File content:
  dog bird cat cat

cat dog fish cat
dog
bird fish
Result file:
dog
bird fish

Outputs “cat”

You must develop the `textsummary` utility using a test-driven development approach such as the one we saw in P4L4. Specifically, you will perform two activities within this project:

- For Deliverable 1, you will extend the set of test cases that you created for Assignment 6; that is, you will use your test specifications to prepare 15 additional JUnit tests (i.e., in addition to the 15 you developed for Assignment 6, for a total of 30 test cases). Then, you will implement the actual `textsummary` utility and make sure that all of the test cases that you developed for Assignment 6 and Deliverable 1, plus the test cases provided by us, pass.
- Deliverable 2 will be revealed in due time.

Deliverables:

DELIVERABLE 1 (this deliverable)

- **Provided (in Assignment 6):**
  - Skeleton of the main class for `textsummary` (`Main.java` we provided in A6)
  - Initial set of JUnit test cases
    - Tests we provided (`MainTest.java`)
    - Tests you created for A6 (`MyMainTest.java`)
- **expected:**
  - 15 **additional** JUnit test cases
  - Implementation of textsummary that makes **all** test cases pass

DELIVERABLE 2

- **provided:** TBD
- **expected:** TBD

Deliverable 1: Instructions

Setup

1. In the root directory of the **personal GitHub repository that we assigned to you**, create a directory called "IndividualProject". Hereafter, we will refer to this directory as `<dir>`.
2. Copy your `MyMainTest.java` file from your Assignment6 directory to a corresponding directory in `<dir>`:
   `<dir>/textsummary/test/edu/gatech/seclass/textsummary/MyMainTest.java`
   This is the test class with your original 15 test cases (to which you will add the additional 15 tests you create for Deliverable 1).
3. Unpack IndividualProject into `<dir>`. It includes **updated** copies of the following files:
   - **`<dir>/textsummary/src/edu/gatech/seclass/textsummary/Main.java`**
     This is the skeleton of the `Main` class of the `textsummary` utility, which you will have to complete for Deliverable 1 **after creating the additional test cases**.
   - **`<dir>/textsummary/test/edu/gatech/seclass/textsummary/MainTest.java`**
     This initial set of test cases for the `textsummary` utility must all pass, **unmodified**, on the implementation you create. The test cases include examples of the normal usage of `textsummary` as well as an example of invoking the `usage` message for an invalid invocation.

Test generation

4. Generate 15 or more JUnit test cases for the `textsummary` utility (in addition to the ones we provided and your original 15 from Assignment 6) and put them in class `MyMainTest`.
   - As you did for Assignment 6, you should add a concise comment to each test that states the test case's purpose and which test frame it implements,using the following format:

     ```
     // Purpose: <concise description of the purpose of the test>
     // Frame #: <test case number in the catpart.txt.tsl of A6>
     ```

   - As a reminder, you can reuse and adapt, when creating your test cases, some of the code, such as the utility methods, which we provided in the `MainTest` class (**without copying the provided test cases verbatim, of course**). You should also feel also free to implement your test cases differently.

Implementation

5. Implement the `textsummary` utility by completing the `Main` class, whose skeleton we provided in Assignment 6, and adding any other class you may need. The requirements for your code are that it passes all the test cases that you created and the ones that we provided, and that it can be executed as follows:

   ```
   java -cp <classpath> edu.gatech.seclass.textsummary.Main <arguments>
   ```

Submission

6. Commit and push your code. You should commit, at a minimum, the content of directory `<dir>/textsummary/test and <dir>/textsummary/src`. As for previous assignments and projects, committing the whole IntelliJ IDEA project, in case you used this IDE, is fine.
7. Paste the commit ID for your submission on Canvas, as usual.

Notes

- After completing your implementation and running your tests, you may find mistakes in the oracles within your tests. You should correct these, **even if they are within tests that you created for A6**. In other words, your implementation should pass all tests, both new and from A6. While you should keep the **frames** you implemented in A6, making corrections to the **tests** is expected.
- If you did not create 15 tests for Assignment 6, you should create additional tests so that you still submit 30 of your own tests overall. For these additional tests, you should add a comment in the same format that was requested for A6:

```
// Purpose: <concise description of the purpose of the test>
// Frame #: <test case number in the catpart.txt.tsl of Part I>
```

- If for any reason you *cannot* create all the required tests using your A6 test frames (e.g., because of severe errors in the frames or missing frames), create additional tests unrelated to the frames and add to each such test a comment "`// not from frame: <reason>`", with a short explanation of why the tests frames cannot be used. Please note, however, that you can and should **make tests from your test frames even if they are not ideal** (and optionally create additional "`not from frame`", better tests). You can contact us privately on Piazza in case of doubts.