Individual Project:
TextSummary Utility -- Deliverable 2

Project Goals

For the last deliverable of this project, your goals are to

- Refactor a Java application for analyzing text within a file.
- Get experience with an agile, test-driven process.
- Evaluate your current set of tests on an alternative implementation of `textsummary`.
- Extend the test set in a black-box fashion to debug the alternative implementation.

Details

For this project you must develop, using Java, a simple **command-line** utility called `textsummary`, which is the same utility for which you developed test frames in the category-partition assignment. For Deliverable 1, you have developed a first implementation of `textsummary` that makes your initial set of test cases pass.
For this deliverable, you have to (1) modify your implementation to account for a slight update in the specification for `textsummary` requested by your customer and (2) use your tests to debug a provided implementation of textsummary. The updated specification is below, with the changed parts marked in red:

Concise (Updated) 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>[integer]`
  - `-d [integer]`
  - `(-l|-s) [integer]`
  - `-u`
- COMMAND-LINE ARGUMENTS AND OPTIONS

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

`-c <string>[integer]`: 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. If the optional positive `[integer]` is provided, then all lines with less than that many copies of the string will be removed.

`-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, and the number of times each appears. 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, -s, and -u are mutually exclusive. This option is executed last.

-u: if specified, the `textsummary` utility will keep only the first instance of any unique word (where a word is any sequence of alphanumeric characters) in the file. All other characters remain. Options -l, -s, and -u are mutually exclusive. This option is executed last.

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” 2 file1.txt
```
File content:
dog bird cat cat
cat dog fish

Result file:
2 dog bird cat cat

Outputs "cat 3 dog 2"

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 4"

Example 5:
`textsummary -u file1.txt`
File content:
dog bird cat cat
cat dog fish cat
Result file:
dog bird cat
 fish

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 extended the set of test cases that you created for Assignment 6; that is, you used your test specifications to prepare 15 additional JUnit tests (i.e., in addition to the 15 you developed for Assignment 6). Then, you implemented the actual `textsummary` utility and made sure that all of the test cases that you developed for Assignment 6 and Deliverable 1, plus the initial test cases provided by us, passed.

- Deliverable 2 will continue the test-driven development approach, extending and refactoring `textsummary` to meet the updated specifications and pass the additional provided tests. Then you will use your tests to debug a provided implementation of textsummary.

Deliverables:

DELIVERABLE 1 (done)

DELIVERABLE 2 (this deliverable)

- **provided:**
  - Additional set of JUnit test cases (to be run in addition to yours)
  - Updated textsummary specification
  - Blackbox test implementation of textsummary
- **expected:**
  - Implementation of textsummary that makes **all** test cases pass
  - Test cases for debugging provided implementation of textsummary

Task 1

Instructions

1. Download archive individualProject-d2.tar.gz
2. Unpack the archive in the root directory of the **personal GitHub repo that we assigned to you**. After unpacking, you should see the following files:
   - `<root>/IndividualProject/.../textsummary/MainTest.java`
   - `<root>/IndividualProject/.../textsummary/MainTestSuite.java`

   There are other files for Task 2; ignore them for now.
   If you have difficulty unpacking the archive on Mac use the command:

   ```
   tar -zxvf IndividualProject-d2.tar.gz
   ```

   You may need to move files manually if they will not unpack to the correct directory on your machine.
3. Class `MainTest` consist of the originally provided test class with both updated and additional test cases for the `textsummary` utility. **It should replace your original `MainTest.java` file**. Imagine that these are test cases developed by coworkers who were also working on the project and developed tests for `textsummary` based on (1) updated customer requirements and (2) some of the discussion about the semantics of `textsummary` that took place on Piazza during the week. As was the case for the test cases we provided for Deliverable 1, **all these tests must pass, unmodified, on your implementation of `textsummary`.**

Please note that these tests clarify `textsummary`'s behavior and also make some assumptions on the way `textsummary` works. You should use the test cases to guide the refactoring of your code.

In *most* cases, we expect that these assumptions will not violate the assumptions you made when developing your test cases for Deliverable 1. However, for any tests which do violate your prior assumptions, please **adapt your test cases (and possibly your code) accordingly**. This may also give you an additional opportunity to get some experience with refactoring.

To summarize: all the test cases in the new `MainTest` class should pass (unmodified) and all the test cases in the `MyMainTest` class should pass (possibly after modifying them) on your implementation of textsummary.

Please ask privately on Piazza if you have doubts about specific tests. We will make your question public if it is OK to do so.

4. As usual, develop your code using Java 8 and ensure that any libraries used other than the standard jdk and junit are committed to your git repository.
5. Class `MainTestSuite` is a simple test suite that invokes all the test cases in test classes `MainTest` and `MyMainTest`. You can use this class to run all the test cases for `textsummary` at once and check that they all pass. This is how we will run all the test cases on your submission.
6. Commit and push your code to your assigned individual private repository. You should commit, at a minimum, the content of directory `<dir>/textsummary/test` and `<dir>/textsummary/src`. As usual, committing the whole IntelliJ IDEA project, in case you used this IDE, is fine.

Task 2

In this deliverable, you will be provided with a precompiled, alternative version of the `textsummary` utility that was developed by one of your colleagues in parallel to yours. The goal of this task is to use this version of `textsummary` to evaluate the tests that you developed for the previous deliverables. To complete this task you must (1) run the tests you defined (class `MyMainTest`) against the provided version of `textsummary` and review the report on their results and (2) extend your set of tests to find bugs in this version of `textsummary`. There are 7 labeled bugs in the provided `textsummary` utility, none of which are revealed by the test cases **we** provided. You must find **at least five unique bugs** to get full points for this task, and finding the two remaining bugs will qualify you for up to 10 extra points. To remove ambiguity in identifying the bugs, **each of the**

labeled bugs will throw an `SDPBugException` with a message labeling it as a discovered bug and a unique identifying number (i.e., if another test outputs the same identifying bug number, it is the same bug). Each test will only be able to throw one exception, representing the first bug found, even if it triggers more than one bug.

Instructions

Setup

1. After unpacking the archive in Task 1, you should also see the following files:

Under `<root>/IndividualProject/D2`:

- **`compileAndRunTests.bat`** and **`compileAndRunTests.sh`**
  These scripts for Windows and Unix/Mac, respectively, will compile and run against the provided version of `textsummary` the set of tests in `.../D2/testsrc/edu/.../MyMainTest.java` and save the console output for your review.
- **`lib/*`**
  Various libraries used to run the tests. You can safely ignore these files.
- **`.../D2/testsrc/edu/.../MyMainTest.java`**
  This is a placeholder that you will have to replace with your own version of this test class.
- **`testclasses/*`**
  This is the directory where the compiled version of your tests (i.e., class `MyMainTest`) will be saved. You can safely ignore this directory.

Testing

2. Before beginning this task you should make sure that the provided files work as expected, by doing the following:
   - Open a command shell
   - Go to directory `.../D2`
   - Run `./compileAndRunTests.sh`

   (if on Windows, run the corresponding bat file)
   - Check the content of file `report.txt`, which should be as follows (time may vary):

```
JUnit version 4.12
```

```
.............................
Time: 0.215

OK (9 tests)
```

- **If you have not set up java for use from the command line, you will need to do so.**
- If some of these steps do not work as expected, please post a public question on Piazza, as other student may have solved similar issues and may be able to help.

3. Copy your latest version of `MyMainTest.java` to `.../D2/testsrc/edu/.../textsummary/MyMainTest.java`, thus replacing the placeholder file currently there.
4. Run your test suite by executing `compileAndRunTests` (`.sh` or `.bat`, depending on your platform).
5. Rename your initial junit report from `report.txt` to `report-initial.txt` under directory `.../D2`, so that it will remain saved.
6. Review the junit report. If one of your test cases fails, it could be for one of several reasons:
   - Your test case hits a corner case that is not defined in the requirements and for which your version of `textsummary` makes different assumptions than the version we provided. You should (1) modify the test case (**in this copy of `MyMainTest.java` only**) so that it **passes** and (2) add the following comment right before the `@test` annotation for that test: "// Failure Type: Corner Case".
   - Your test behaves incorrectly, which means that there is a bug in your version of `textsummary` that was not caught by our test suite (class `MainTest`). You should (1) modify the test case (**in this copy of `MyMainTest.java` only**) so that it passes and (2) add the following comment right before the `@test` annotation for that test: "// Failure Type: Test Failure". **Note**: You will not be penalized for this, so there is no need to fix the bug in your own textsummary implementation; we are actually interested in seeing what kind of bugs our test suite missed in your code.
   - Your test triggered a bug in our version of `textsummary` and caused a SDPBugException: good job! In this case, you should leave the test case as is but add the following comment right before the

`@test` annotation for that test: "// Failure Type: BUG <short explanation of the failure and what you think is the corresponding bug>". Do not worry too much about the explanation and just make your best guess as you will not be penalized for getting a reasonable, but incorrect, explanation. If you trigger duplicates of the same bug, you may label them with a copy of the same explanation or simply "Duplicates <testname>".

As an example, if the output of your test was:

```
edu.gatech.seclass.textsummary.SDPBugException: You
found Bug #0.
Arguments used: [-c, "cat", -d, 2, -l, 4,
C:\Temp\junit23859131\junit44022402542.tmp]
File: n/a
Encode result: throws an
ArrayIndexOutOfBoundsException
```

You might label the test with:

```
// Failure Type: BUG: textsummary fails when passed
more than 2 options, probably due to the storing
of the options in a fixed-size array
```

Note: This example bug is not one of the bugs in the provided implementation.

If you found at least five unique bugs in the version of `textsummary` we provided, you are done.

- Otherwise, you should add test cases to your test suite (`.../`**`D2`**`/testsrc/edu/.../textsummary/MyMainTest.java`) to make our version of `textsummary` fail and throw a SDPBugException. If one of the new tests you add does cause a failure in our version of `textsummary`, you should add the following comment right before the `@test` annotation for that test: "// New Failure Type: BUG: <short explanation of the failure and what you think it's the corresponding bug>".
  Note: To extend your test suite, you should add new tests. You may use test cases from the test suite we provided (`MainTest`), but our tests will not reveal any bug in our code.

- This task will be completed when either your tests cause at least five failures or you are stuck and cannot find further bugs. (In this latter case, you will still get partial credit depending on how much you accomplished.)

7. Generate the junit report for the final version of your test suite and save it as `report-final.txt` under directory `.../D2`. **All tests in this version should pass, except for the labelled bugs (which should all fail with an uncaught SDPBugException).**

Note: **All valid bugs may be found with junit tests and from the command line**. They do not include execution time, null input, extremely large or non-text files, or extended character sets.

8. Commit and push at a minimum the following files (in addition to the files from Task 1):
    - `.../D2/testsrc/edu/gatech/seclass/textsummary/MyMainTest.java`
    - `.../D2/report-initial.txt`
    - `.../D2/report-final.txt`
    - There is no need to push any other file, as none of the other existing files under `D2` is supposed to be modified, but as usual, it will not be penalized if you do push all related project files.

9. After committing your work for Task 1 and Task 2, paste this last commit ID on Canvas, as usual. You should only submit one commit ID, after both tasks are done.