Individual Project: `txted` Deliverable 1

Project Goals

In this project, you will be developing a simple Java application (txted) using an agile, test-driven process involving multiple deliverables. While you will receive one grade for the entire project, each deliverable must be completed by its own due date, and all deliverables will contribute to the overall project grade.

Specification of the `txted` Utility

`txted` is a simple command-line utility written in Java with the following specification:

Summary

`txted` allows for simple text manipulation of the content of a file.

Syntax

```
txted OPTIONS FILE
```

Description

Program `txted` performs basic text transformations on lines of text from an input FILE. Unless the -f option (see below) is specified, the program writes transformed text to stdout and errors/usage messages to stderr. The FILE parameter is required and must be the last parameter. OPTIONS may be zero or more of the following and may occur in any order:

- **-f**
  Edit file in place. The program overwrites the input file with transformed text instead of writing to stdout.
- **-e** *<string>*
  Exclude any lines containing the given string.
- **-i**
  Used with the -e flag ONLY; applies case insensitive matching.
- **-s** <*integer*>
- Skip either the even or odd lines in a file, with 0 being even and 1 being odd.

- **-x** *<string>*
  Adds string as a suffix to each line.
- **-r**
  Reverse the order of lines in a file -- the last line is first, the first line is last, and so on.
- **-n** <*integer*>
  Add a line number field followed by a single space to the beginning of each line output. The line number field shall be left-padded with 0 or left-truncated as required to the width specified by the integer parameter. Line numbering should start at 1.

NOTES:

- While the last command-line parameter provided is always treated as the filename, OPTIONS flags can be provided in any order and will be applied as follows:
  - Options -f and -i should be processed first, as they determine global parameters of the computation.
  - Options -s, -e, -x, -r, and -n should be processed in this order. That is: (1) if -s is present, then file content is filtered based on the specified parameter; (2) if -e is present, then file content excludes lines that include the specified parameter; (3) if -x is present, then a suffix should be applied; (4) if -r is present, then reversal of line order logic is performed; (5) if -n is present, then padded line numbering should be applied.
- Specifying option -i without having specified option -e should result in an error.
- Specifying option -e or -x with an empty string parameter should result in an error.
- Specifying option -n with an integer <0 should result in an error.
- Specifying option -s with an input parameter not equal to 0 or 1 should result in an error. Each file starts with line 1, which should be considered odd.
- If options are repeated, only their last occurrence is considered.
- All program option parameters are required and will result in an error if omitted.
- You should assume that the <string> parameters will not contain newlines, as the behavior of the program is platform dependent and undefined in those cases.
- You should assume that the last line of the input file will be newline-terminated. Otherwise, program behavior is undefined.
  - The only exception would be an empty input file which should produce an empty output with no options executed.

EXAMPLES OF USAGE

(In the following, "↵" represents a newline character.)

Example 1:

```
txted -f FILE
input FILE:
alphanumeric123foobar↵
↵
edited FILE:
alphanumeric123foobar↵
↵
stdout: nothing sent to stdout
stderr: nothing sent to stderr
```

Example 2:

```
txted -e ABC FILE
input FILE:
01234abc↵
56789def↵
01234ABC↵
56789DEF↵
↵
edited FILE: file not edited
stdout:
01234abc↵
56789def↵
56789DEF↵
↵
stderr: nothing sent to stderr
```

Example 3:

```
txted -e ABC -i FILE
input FILE:
01234abc↵
56789def↵
01234ABC↵
56789DEF↵
↵
edited FILE: file not edited
stdout:
56789def↵
56789DEF↵
↵
stderr: nothing sent to stderr
```

Example 4:
txted -r FILE
input FILE:
01234abc↵
56789def↵
01234ABC↵
56789DEF↵
↵
edited FILE: **file not edited**
stdout:
56789DEF↵
01234ABC↵
56789def↵
01234abc↵
↵
stderr: **nothing sent to stderr**

Example 5:
txted -s 1 FILE
input FILE:
01234abc↵
56789def↵
01234ABC↵
56789DEF↵
↵
edited FILE: **file not edited**
stdout:
56789def↵
56789DEF↵
↵
stderr: **nothing sent to stderr**

Example 6:
txted -x ! FILE
input FILE:
01234abc↵
56789def↵
01234ABC↵
56789DEF↵
↵
edited FILE: **file not edited**
stdout:
01234abc!↵
56789def!↵
01234ABC!↵

56789DEF!↵
↵
stderr: **nothing sent to stderr**

Example 7:
txted -n 3 FILE
input FILE:
01234abc↵
56789def↵
01234ABC↵
56789DEF↵
↵
edited FILE: **file not edited**
stdout:
001 01234abc↵
002 56789def↵
003 01234ABC↵
004 56789DEF↵
↵
stderr: **nothing sent to stderr**

Example 8:
txted -n 4 -r -x !!! FILE
input FILE:
01234abc↵
56789def↵
01234ABC↵
56789DEF↵
↵
edited FILE: **file not edited**
stdout:
0001 56789DEF!!!↵
0002 01234ABC!!!↵
0003 56789def!!!↵
0004 01234abc!!!↵
↵
stderr: **nothing sent to stderr**

Example 9:

```
txted -n 3 -r -e Bar -s 0 -n 2 -x ! -f FILE
input FILE:
alphanumeric123foo↵
alphanumeric123Foo↵
alphanumeric123FOO↵
alphanumeric123bar↵
alphanumeric123Bar↵
alphanumeric123BAR↵
alphanumeric123foobar↵
alphanumeric123Foobar↵
alphanumeric123fooBar↵
alphanumeric123FooBar↵
alphanumeric123FOOBar↵
alphanumeric123FooBAR↵
alphanumeric123FOOBAR
↵
edited FILE:
01 alphanumeric123FOOBAR!↵
02 alphanumeric123foobar!↵
03 alphanumeric123FOO!↵
04 alphanumeric123foo!↵
↵
stdout: nothing sent to stdout
stderr: nothing sent to stderr
```

Example 10:

```
txted -i FILE
input FILE:
01234abc↵
56789def↵
01234ABC↵
56789DEF↵
↵
edited FILE: file not edited
stdout: nothing sent to stdout
stderr:
Usage: txted [ -f | -i | -s integer | -e string | -r | -x string | -n
integer ] FILE
```

Deliverables Summary

This part of the document is provided to help you keep track of where you are in the individual project and will be updated in future deliverables.

DELIVERABLE 1 (this deliverable, see below for details)

- **Provided:**
  - `txted` specification
  - Skeleton of the main class for `txted`
  - Example tests and skeleton of the test class to submit
  - JUnit libraries
- **Expected:**
  - Part I (Category Partition)
    - `catpart.txt`: TSL file you created
    - `catpart.txt.tsl`: test specifications generated by the `TSLgenerator` tool when run on your TSL file.
  - Part II (Junit Tests)
    - Junit tests derived from your category partition test frames (`MyMainTest.java`)

DELIVERABLE 2

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

DELIVERABLE 3

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

DELIVERABLE 4

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

Deliverable 1: Instructions

Part I

Generate **between 50 and 90 test-case specifications** (i.e., generated test frames) for the `txted` utility using the category-partition method presented in lesson P4L2. **Make sure to watch the lesson and demo before getting started**.

When defining your test specifications, your goal is to suitably cover the domain of the application under test, including **relevant erroneous inputs and input combinations**. Just to give you an example, if you were testing a calculator, you may want to cover the case of a division by zero.

Do not manually generate combinations of inputs as single choices. Instead, use multiple categories and choices with necessary constraints to cause the tool to generate meaningful combinations. Using the calculator example again, you should not offer choices *"add"*, *"multiply"*, and also *"add and multiply"* in a single category.

In particular, make sure to use constraints (error and single), selector expression (if), and properties appropriately, rather than eliminating choices, to keep the number of test cases within the specified thresholds.

Note that **the domain is that of the java application under test**, so you can assume that anything the shell would reject (e.g., unmatched double quotes) will not reach the application. In other words, **you must test for invalid input arguments, but do not need to test for errors involving parsing the command-line arguments before they are sent to the java application**. You can find more details about command-line argument parsing at this link. To illustrate, the sample tests in Part II will demonstrate how input arguments would be sent to your application.

Please also keep in mind that **you are only required to specify test inputs, but you do not have to specify the expected outcome for such inputs in Part I**. It is therefore OK if you do not know how the system would behave for a specific input. Using once more the calculator example, you could test the case of a division by zero even if you did not know how exactly the calculator would behave for that input.

Tools and Useful Files

You will use the `TSLgenerator` tool to generate test frames starting from a TSL file, just like we did in the demo for lesson P4L2. Versions of the `TSLgenerator` for Linux, Mac OS X, and Windows, together with a user manual, are available at:

- TSLgenerator-manual.txt
- TSL generator for Linux
- TSL generator for Mac OS
- TSL generator for Windows 8 and newer

- TSL generator for Windows XP and earlier

We are also providing the TSL file for the example we used in the lesson, cp-example.txt, for your reference.

Important:

- **These are command-line tools**, which means that **you have to run them from the command line**, as we do in the video demo, rather than by clicking on them.

- On Linux and Mac systems, you may need to change the permissions of the files to make them executable using the `chmod` utility. To run the tool on a Mac, for instance, you should do the following, from a terminal:

```
chmod +x TSLgenerator-mac
./TSLgenerator-mac <command line arguments>
```

- You can run the `TSLgenerator` as follows:
`<tool> [--manpage] [-cs] infile [-o outfile]`

  Where `<tool>` is the specific tool for your architecture, and the command-line flags have the following meaning:

`--manpage`	`Prints the man page for the tool.`
`-c`	`Reports the number of test frames that would be generated, without` **`actually producing them.`**
`-s`	`Outputs the result to standard output.`
`-o outfile`	`Outputs the result to file outfile,` **`unless the -s option is also used.`**

- If you encounter issues while using the tool, please post a public question on Ed Discussion and consider running the tool on the VM provided for the class or on a different platform (if you have the option to do so). Gradescope will execute the tool on a Linux platform.

Committing Part I

- Create a directory `"IndividualProject"` **in the personal GitHub repo we assigned to you**.
- Add to this new directory two text files:
  - `catpart.txt`: TSL file you created.
  - `catpart.txt.tsl`: test specifications generated by the `TSLgenerator` tool when it processes your TSL file.
- Commit and push your files to GitHub. (You can also do this only at the end of Part II, but it is always safer to have intermediate commits.)

Part II

In this second part of the deliverable, you will create actual test cases implementing the test specifications you created in Part I. (As discussed in the lesson on the category-partition method, each test frame is a test spec that can be instantiated as an individual concrete test case). To do so, you should perform the following steps:

- Download archive `individualproject-d1.tar.gz`
- Unpack the archive in the directory "`IndividualProject`", which you created in Part I of the deliverable. Hereafter, we will refer to this directory as `<dir>`.
- After unpacking, you should see the following structure:
  - `<dir>/txted/src/edu/gatech/seclass/txted/Main.java`
    This is a skeleton of the `Main` class of the `txted` utility, which we provide so that the test cases for `txted` can be compiled. It contains an empty main method and a method usage, which prints on standard error a usage message and should be called when the program is invoked incorrectly. In case you wonder, this method is provided for consistency in test results.
  - `<dir>/txted/test/edu/gatech/seclass/txted/MainTest.java`
    This is a test class with a few test cases for the `txted` utility that you can use as an example and that correspond to the examples of usage of `txted` that we provided. In addition to providing this initial set of tests, class MainTest also provides some utility methods that you can leverage/adapt and that may help you implement your own test cases:
    - `File createTmpFile()`
      Creates a File object for a new temporary file in a platform-independent way.
    - `File createInputFile*()`
      Examples of how to create, leveraging method `createTmpFile`, input files with given contents as inputs for your test cases.
  - `<dir>/txted/test/edu/gatech/seclass/txted/MyMainTest.java`
    This is an empty test class in which you will add your test cases, provided for your convenience.
  - `<dir>/txted/lib/junit-4.12.jar`
    `<dir>/txted/lib/hamcrest-core-1.3.jar`
    JUnit and Hamcrest libraries to be used for the assignment.
- Use the test frames from Part I to generate additional JUnit test cases for the `txted` utility, one per frame, and put them in the test class `MyMainTest` (i.e., **do not add your test cases to class `MainTest`**). For ease of grading, please name your test cases `txtedTest1`, `txtedTest2`, and so on. Each test should contain a concise comment that indicates which test frame the test case implements. Use the following format for your comments, before each test:
  `// Frame #: <test case number in file catpart.txt.tsl>`

  Your test frames should contain enough information to create relevant test cases. **If you**

cannot implement your test frames as useful JUnit tests (e.g., because the test frames do not provide enough information), you should revisit Part I. Extending the calculator example, if your test frame specified a numerical input, and you realized that you should use both negative and positive numbers in your actual test case, you should revise your categories and choices so that this is reflected in your test frames

- **If you are uncertain what the result should be for a test, you may make a reasonable assumption on what to use for your test oracle.** While you should include a test oracle, we will not grade the accuracy of the test oracle itself.

  Feel free to reuse and adapt, when creating your test cases, some of the code we provided in the `MainTest` class. Feel also free to implement your test cases differently. Basically, class `MainTest` is provided for your convenience and to help you get started. Whether you leverage class `MainTest` or not, your test cases should assume (just like the test cases in `MainTest` do) that the `txted` utility will be executed from the command line, as follows:

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

- **Make sure not to make calls to System.exit() within your tests, as that creates problems for JUnit.**
- **For this deliverable, do not implement the `txted` utility, but only the test cases for it. This means that most, if not all of your test cases will fail, which is fine.**

Committing Part II and Submitting the Deliverable

- As usual, commit and push your code to your individual, assigned private repository.
- Make sure that all Java files are committed and pushed, including the ones we provided.
- Make also sure to commit and push the provided libraries (`lib` directory). To do so, you may need to force add the jar files (i.e., "`git add -f lib/*`"), which are typically excluded by the "`.gitignore`" file.
- You can check that you committed and pushed all the files you needed by doing the following:
  - Clone a fresh copy of your personal repo in another directory
  - Go to directory `IndividualProject/txted` in this fresh clone
  - Compile your code. One way to do is to run, from a Unix-like shell:
    ```
    javac -cp lib/\* -d classes src/edu/gatech/seclass/txted/*.java
    test/edu/gatech/seclass/txted/*.java
    ```
    (on some platforms, you may need to first create directory "`classes`")
  - Run your tests. Again, from a Unix-like shell, you can run:
    ```
    java -cp classes:lib/\* org.junit.runner.JUnitCore
    edu.gatech.seclass.txted.MyMainTest[1]
    ```
- **Submit on Gradescope a file, called submission.txt that contains, in two separate**

[1] If using a Windows-based system, you may need to run `java -cp "classes;lib/*" org.junit.runner.JUnitCore edu.gatech.seclass.txted.MyMainTest` instead.

lines (1) your GT username and (2) the commit ID for your submission. For example, the content of file submission.txt for George P. Burdell could look something like the following:

submission.txt

```
gpburdell1
81b2f59
```

- **As soon as you submit, Gradescope will verify your submission** by making sure that your files are present and in the correct location, as well as a few additional minor checks. If you pass all these checks, you will see a placeholder grade of 10 and a positive message from Gradescope. Otherwise, you will see a grade of 0 and an error message with some diagnostic information. Please note that:
  - **a positive response from Gradescope only indicates that you passed the initial checks and is meant to prevent a number of trivial errors**;
  - **if your submission does not pass the Gradescope checks, it will not be graded and will receive a 0**, so please make sure to pay attention to the feedback you receive when you submit and keep in mind that **you can resubmit as many times as you want before the deadline.**[2]

Gradescope Queries

If you need clarification or have questions regarding Gradescope output, please post privately on Ed Discussion (we will make it public if appropriate) and make sure to add, when it applies:

- a link to the Gradescope results, and
- any information that may be relevant.

The bottom line is that, to make the interaction efficient, you should make your posts as self-contained and easy-to-check as possible. The faster we can respond to the posts, the more students we can help.

[2] Although we tested the checker, it is possible that it might not handle correctly some corner cases. If you receive feedback that seems to be incorrect, please contact us on Ed Discussion.