Individual Project:
Replace Utility -- Deliverable 2

Project Goals

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

Details

For this project you must develop, using Java, a simple **command-line** utility called `replace`, 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 `replace` that makes your initial set of test cases pass.
For this second deliverable, you have to modify your implementation to account for a slight update in the specification for `replace` requested by your customer. The updated specification is below, with the changed parts marked in red:

Concise (Updated) Specification of the `replace` Utility

- NAME:
  `replace` - looks for all occurrences of string *from* in a file and replaces it with string *to*.

- SYNOPSIS
  `replace OPT <from> <to> <filename>`

  where `OPT` can be **zero or more** of
  - `-f`
  - `-i`
  - `-w [char]`
  - `-x [char]`

- COMMAND-LINE ARGUMENTS AND OPTIONS

  `from`: string to be replaced with string `to`.

  `to`: string that will replace string `from`.

  `filename`: the file on which the replace operation has to be performed.

  `-f`: if specified, the `replace` utility only replaces the first occurrence of string `from` in the file.

  `-i`: if specified, the `replace` utility performs the search for string `from` in a case insensitive way.

`-w [char]`: if specified, the `replace` utility performs the search for string `from` only matching whole words or phrases that are delimited either by a whitespace (if no character is specified) or by the character provided as an additional argument.

`-x [char]`: if specified, the `replace` utility matches string `from` by treating as a wildcard either 'x' (if no character is specified) or the character provided as an additional argument.

- EXAMPLES OF USAGE

Example 1:
```
replace -i Howdy Hello file1.txt
```
would replace all occurrences of "Howdy" with "Hello" in the file specified. Because the "`-i`" option is specified, occurrences of "howdy", "HOwdy", "HOWDY", and so on would also be replaced.

Example 2:
```
replace -w -f Bill William file1.txt
```
would replace the first whitespace delimited occurrence of "Bill" with "William" in the file specified. It would not replace non-whitespace delimited occurrences, such as "Billy".

Example 3:
```
replace -w "Bill is" "William is" file1.txt
```
would replace all whitespace delimited occurrences of "Bill is" with "William is" in the file specified. It would not replace non-whitespace delimited strings, such as the occurrence of "Bill is" in the text "Bill is, in my opinion, …".

Example 4:
```
replace -w -i abc ABC file1.txt
```
would replace all whitespace delimited occurrences of "abc", where the letters in "abc" can be either uppercase or lowercase, with "ABC" in the file specified. This would include "Abc", or "aBc", but not "abc!" or "Abc's".

Example 5:
```
replace -x -i xill Jill file1.txt
```
would replace all occurrences of any character plus "ill", where the letters in "ill" can be either uppercase or lowercase, with "Jill" in the file specified. This would include, for instance, "WILL" or "Bill".

Example 6:
```
replace -x q qlex Alex file1.txt
```
would replace all occurrences of any character plus "lex" with "Alex" in the file specified.

You must develop the `replace` utility using a test-driven development approach such as the one we saw in P4L4. Specifically, you will perform three 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 `replace` 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 `replace` to meet the updated specifications and pass the additional provided tests.
- Deliverable 3 will be revealed in due time.

Deliverables:

<u>DELIVERABLE 1</u> (this deliverable)

- **Provided (in Assignment 6):**
  - ~~Skeleton of the main class for `replace`~~
  - ~~Initial set of JUnit test cases (yours + ours)~~
  - ~~Your A6 submission~~
- **expected:**
  - ~~15 **additional** JUnit test cases~~
  - ~~Implementation of replace that makes **all** test cases pass~~

<u>DELIVERABLE 2</u>

- **provided:**
  - Additional set of JUnit test cases (to be run in addition to yours)
  - Updated replace specification
- **expected:**
  - Implementation of replace that makes **all** test cases pass

<u>DELIVERABLE 3</u>

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

Deliverable 2: 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/.../replace/MainTest.java`
   - `<root>/IndividualProject/.../replace/MainTestSuite.java`
3. Class `MainTest` is the originally provided test class with both updated and additional test cases for the `replace` 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 `replace` based on (1) updated customer requirements and (2) some of the discussion about the semantics of `replace` 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 `replace`.

   Please note that these tests clarify `replace`'s behavior and also make some assumptions on the way `replace` works. You should use the test cases to refactor 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, but they might in some cases. If they do, please adapt your test cases (and possibly your code) accordingly, which 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 replace.

   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. 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 `replace` at once and check that they all pass. This is how we will run all the test cases on your submission.
5. Commit and push your code. You should commit, at a minimum, the content of directory `<dir>/replace/test` and `<dir>/replace/src`. As for previous assignments and projects, committing the whole IntelliJ IDEA project, in case you used this IDE, is fine.
6. Paste the commit ID for your submission on T-Square, as usual.