Assignment 6: Category-Partition

This individual assignment has **two parts**. For Part I, you must generate test case specifications for a simplified version of the `replace` utility, whose specs are provided below, using the category-partition method. For Part II, you will demonstrate how some of your test frames may be used, by developing test cases that implement them.

Concise 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`
- 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`: if specified, the `replace` utility performs the search for string `from` only matching whole words or phrases that are whitespace delimited.
- 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".

Part I

Generate **between 40 and 80 test-case specifications** (i.e., generated test frames) for the `replace` utility using the category-partition method that we saw in lesson P4L2. **Make sure to watch the included demo and read the instructor notes for the lesson**. Make also sure, when defining your test specifications, to suitably cover the domain of the application under test. This also includes possibly erroneous inputs. Just to give you an example, if you had to test a calculator, you may want to cover the case of a division by zero. **Use constraints appropriately (rather than eliminating choices) to keep the number of test cases within the specified thresholds**.

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. A version of the `TSLgenerator` tool for Linux, Mac OS X, and Windows (two versions), together with a user manual, are available at:

- TSLgenerator-manual.txt
- TSLgenerator.linux
- TSLgenerator-mac
- TSLgenerator-win32.exe
- TSLgenerator-win8.exe

We are also providing the TSL file for the example we saw 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>
```

- If you are running a modern (i.e., newer than XP) version of Windows, you may want to try the `TSLgenerator-win8.exe` version of the tool. It should work better on these versions than `TSLgenerator-win32.exe`. If you encounter issues, please post a public question on Piazza 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).

Instructions to commit Part I

- Create a directory "`Assignment6`" in the personal GitHub repo we assigned to you.
- Add to this new directory two text files for Part I:
  - `catpart.txt`: the TSL file you created.
  - `catpart.txt.tsl`: the test specifications generated by the `TSLgenerator` tool when run on 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

To demonstrate how your test frames resulting from the category partition may be used to cover the domain of the application with appropriate tests, you will use your test specifications and a provided interface to prepare **15 actual JUnit tests** (as discussed in the lesson on the category-partition method, each test frame is a test spec that can be instantiated as a concrete test case). To do so, you should perform the following steps:

- Download archive Assignment6.tar.gz
- Unpack the archive in directory "Assignment6", which you created when completing Part I of the assignment. Hereafter, we will refer to this directory as `<dir>`. After unpacking, you should see the following structure:
  - `<dir>/replace/src/edu/gatech/seclass/replace/Main.java`
    This is a skeleton of the Main class of the replace utility, which is provided so that the test cases for replace 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.
  - `<dir>/replace/test/edu/gatech/seclass/replace/MainTest.java`
    This is a test class with a few test cases for the replace utility that you can use

as an example and that correspond to the examples of usage of `replace` 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 a given content as inputs for your test cases.
    - `String getFileContent(String filename)`
      Returns a `String` object with the content of the specified file, which is useful for checking the outcome of a test case.

  - `<dir>/replace/test/edu/gatech/seclass/replace/MyMainTest.java`
    This is an empty test class in which you will add your test cases, provided for your convenience.

- Generate 15 additional JUnit test cases for the replace utility 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 `replaceTest1`, `replaceTest2`, and so on. Each test should contain a concise comment that indicates its purpose and which test frame the test case implements. Use the following format for your comments:
  ```
  // Purpose: <concise description of the purpose of the test>
  // Frame #: <test case number in the catpart.txt.tsl of Part I>
  ```
  Each test should obviously suitably implement the referenced test frame.

  Feel free to reuse and adapt, when creating your test cases, some of the code we provided in the `MainTest` class (without copying the provided test cases verbatim, of course, which also implies that you should not implement test frames that correspond exactly to the test cases we provided). But 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 replace utility will be executed from the command line, as follows:
  ```
  java -cp <classpath> edu.gatech.seclass.replace.Main <arguments>
  ```

  Important: Do not implement the replace utility, but only the test cases for it. This also means that most, if not all of your test cases will fail, which is fine.

Instructions to commit Part II and submit the assignment

- Commit and push your code (i.e., the Java files in directory <dir>/replace).
- Paste the final commit ID for your submission on T-Square, as usual.