Individual Project:
Encode Utility -- Deliverable 2

Project Goals

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

- Refactor a Java application for encoding strings within a file.
- Get experience with an agile, test-driven process.
- Evaluate your current set of tests on an alternative implementation of `encode`.
- 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 `encode`, 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 `encode` that makes your initial set of test cases pass.
For this this deliverable, you have to (1) modify your implementation to account for a slight update in the specification for `encode` requested by your customer and (2) use your tests to debug a provided implementation of encode. The updated specification is below, with the changed parts marked in red:

Concise Specification of the `encode` Utility

- NAME:

  `encode` - encodes words in a file.

- SYNOPSIS

  `encode OPT <filename>`

  where `OPT` can be **zero or more** of

  - `-c <integer>`
  - `-r`
  - `-R`
  - `-d <characters>`

- COMMAND-LINE ARGUMENTS AND OPTIONS

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

  `-c <integer>`: if specified, the utility will apply a caesar cipher to all alphabetic and numerical characters (other characters remain unchanged and

the original capitalization is preserved), incrementing letters and numbers by the integer passed in. Letters will wrap back to 'z' or 'a' at either end of the alphabet, and numbers will similarly wrap back to '0' or '9' at either end of the single digit numerals.

`-r`: if specified, the utility reverse the order of characters in every word, with a word being any set of non-whitespace characters, including non-alphabetic characters, separated from others by any whitespace character.

-R: if specified, the utility reverse the order of the words in the file, with a word being any set of non-whitespace characters, including non-alphabetic characters, separated from others by any whitespace character.

`-d <characters>`: if specified, the `encode` utility will remove all characters from the file which match any of the characters passed into this option, before applying any other transformation. Letters will be deleted in a case-insensitive way (e.g. 'a' will remove 'a' and 'A').

If **none** of the OPT flags is specified, `encode` will default applying -c <length of file>, a caesar cipher incrementing by the number of characters in the input file.

- NOTES
  - While the last command-line parameter provided is always treated as the filename, OPT flags can be provided in any order (-d is always applied first).

- EXAMPLES OF USAGE

Example 1:
```
encode file1.txt
```
(where the content of the file is "abc123")
Increments all letters and numbers by the length of the input file, 6 (resulting in "ghi789").

Example 2:
```
encode -r file1.txt
```
Reverses the characters in every whitespace delimited word.

Example 3:
```
encode -d "aeiou" -c 3 file1.txt
```
Removes all 'a', 'e', 'i', 'o' and 'u' characters in lower or upper case, then increments all remaining letters and numbers by 3.

Example 4:
```
encode -c 2 file1.txt
```
Increments all letters and numbers by 2.

Example 5:
```
encode -R file1.txt
```
Reverses the words in the file.

You must develop the `encode` 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 `encode` 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 `encode` to meet the updated specifications and pass the additional provided tests. Then you will use your tests to debug a provided implementation of encode.

Deliverables:

DELIVERABLE 1 (done)

DELIVERABLE 2 (this deliverable)

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

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

Please note that these tests clarify `encode`'s behavior and also make some assumptions on the way `encode` 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, but they might in some cases. If they do, **please adapt your test cases (and possibly your code) to match the assumptions of the provided tests**, 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 encode.

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 `encode` 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>/encode/test` and `<dir>/encode/src`. As for previous assignments and projects, committing the whole IntelliJ IDEA project, in case you used this IDE, is fine.

Task 2

In this task, you will be provided with a precompiled, alternative version of the `encode` utility that was developed by one of your colleagues in parallel to yours. The goal of this task is to use this version of `encode` 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 `encode` and measure the coverage they achieve and (2) extend your set of tests to find bugs in this version of `encode`. There will be 10 bugs in the precompiled `encode` utility, none of which are revealed by the test cases **we** provided. You must find **at least eight 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 bug will throw an SDPBugException with a message labeling it as a discovered bug** with a unique identifying number (if another test outputs the same identifying bug number, it is the same 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 `encode` 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)
You should see the following output:

```
<prompt>compileAndRunTests.bat
<prompt>javac -cp lib\encode.jar;lib\junit-
4.12.jar;lib\hamcrest-core-1.3.jar -d testclasses
testsrc\edu\gatech\seclass\encode\*
```

```
<prompt>cd testclasses
<prompt>\testclasses>jar -cf tests.jar
edu\gatech\seclass\encode\MyMainTest.class
<prompt>\testclasses>cd ..
<prompt>copy testclasses\tests.jar lib\tests.jar
 1 file(s) copied.
<prompt>del testclasses\tests.jar
<prompt>java -cp
lib\tests.jar;lib\encode.jar;lib\junit-
4.12.jar;lib\hamcrest-core-1.3.jar;
org.junit.runner.JUnitCore
edu.gatech.seclass.encode.MyMainTest  1>report.txt
```

- Check the content of file `report.txt`, which should be as follows (time may vary):

```
JUnit version 4.12
.............................
Time: 0.215

OK (6 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/.../encode/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 `encode` 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 `encode` 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 encode 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 `encode` 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 it's 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.

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

```
edu.gatech.seclass.encode.SDPBugException: You
found Bug #0.
Arguments used: [-r, -R, -c, 1, -d, "aeiou",
C:\Temp\junit23859131\junit44022402542.tmp]
File: n/a
Encode result: throws an
ArrayIndexOutOfBoundsException
```

You might label the test with:

```
// Failure Type: BUG: encode fails when passed
more than 3 options, probably due to the storing
of the options in a fixed-size array
```

Note: the fixed-size array example is not a hint...

7. If you found at least eight unique bugs in the version of `encode` we provided, you are done.

8. Otherwise, you should add test cases to your test suite (`.../D2/testsrc/edu/.../encode/MyMainTest.java`) to make our version of `encode` fail and throw a SDPBugException. If one of the new tests you add does cause a failure in our version of `encode`, 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.

9. This task will be completed when either your tests cause at least eight 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.)

10. 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 a thrown SDPBugException).**

11. Commit and push at a minimum the following files (in addition to the files from Task 1):
    - `.../D2/testsrc/edu/gatech/seclass/encode/MyMainTest.java`
    - `.../D2/report-initial.txt`
    - `.../D2/report-final.txt` (unless you didn't have to modify your test suite, so you only have the initial report)
    - 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.

12. After committing your work for Task 1 and Task 2, paste this last commit ID on Canvas, as usual.