Assignment 3: Basic Java coding and JUnit

This assignment assesses your basic knowledge of Java and JUnit, which you will need for future assignments and projects.

To complete the assignment you must complete the following tasks:

1. Clone your individual GitHub repository in your local workspace. This is the same repository that you used for the previous assignment (i.e., `https://github.com/qc-se-fall25/370Fall25<firstname_lastname>.git`).
2. Download the archive Assignment3.zip
3. Unzip the archive in the root directory of the repository, which will create a directory called `Assignment3` and several subdirectories. Hereafter, we will refer to the directory `Assignment3` in your local repo as `<dir>`.
4. Directory `<dir>/src` contains, in a suitable directory, Java interface `edu.qc.seclass.MyCustomStringInterface`. It also contains exception `edu.qc.seclass.MyIndexOutOfBoundsException`, which is used by the interface.
5. Your **first task** is to develop a Java class called `MyCustomString` that suitably implements the `MyCustomStringInterface` that we provided. (The semantics of the methods in the interface should be obvious from their name and from the JavaDoc comments in the code. If not, please ask on Piazza.) Class `MyCustomString` should be in the same package as the interface and should also be saved under `<dir>/src/edu/qc/seclass`.
6. Your **second task** is to develop a set of JUnit 4.0 test cases for class `MyCustomString`.
    - You should create several test cases for each method that implements a method in the interface, except for getters and setters (i.e., the first two methods). **Make sure that every test method has a suitable oracle** (i.e., either an assertion or an expected exception), that the tests are not trivial (i.e., have a specific purpose). In other words, each test should (1) test a specific piece of functionality and (2) check that such piece of functionality behaves as expected.
    - In addition, for each exception that can be thrown by a method M and is explicitly mentioned in M's documentation, there should be at least one test for M that results in that expected exception. For example, at least three of the tests for method `convertDigitsToNamesInSubstring` should result in expected exceptions:
        - One for `IllegalArgumentException`
        - One for `MyIndexOutOfBoundsException`
        - One for `NullPointerException`
    - When testing for an expected exception, make sure to use the `@Test(expected = <exception class>)` notation. For example:

```
@Test(expected = MyIndexOutOfBoundsException.class)
public void testConvertDigitsToNamesInSubstring8() {
  ...
}
```

    - To make your job a little easier, the archive also contains, in directory `<dir>/test`, a test class `edu.qc.seclass.MyCustomStringTest`, which provides a skeleton for the

test cases that you need to implement and a complete implementation for a few of them. Your job is to write the body of the test cases that are not implemented (i.e., the ones that simply fail with a "Not yet implemented" message). Make sure that the test cases you write are not just a trivial variation of the ones we provide. To this end, **make sure to add a concise comment to each test that you implement to clarify its rationale** (e.g.,. "This test checks whether the method `convertDigitsToNamesInSubstring` suitably throws an IllegalArgumentException if startPosition is greater than endPosition"). Finally, creating more than the required test cases will not be rewarded nor penalized; in other words, feel free to write more tests if you are so inclined, but **you will have to provide at least as many tests as there are skeletons**.

7. Submit your solution by
    - Pushing `<dir>` and all the files and directories underneath it (except for the files excluded by .gitignore, if you have one, such as all class files) to the individual remote GitHub repository we assigned to you. You can decide whether to commit Eclipse or IDEA project related files, as this does not make a difference for us, but the repo **must necessarily** contain the two files:
        - `Assignment3/src/edu/qc/seclass/MyCustomString.java`
        - `Assignment3/test/edu/qc/seclass/MyCustomStringTest.java`
    - Submitting the commit ID for your submission on Brightspace. You can get your commit ID by running "`git log -1`".

Notes:

- **You cannot modify the provided interface (`MyCustomStringInterface`), nor the already provided test cases (except for those that you are supposed to implement, obviously, whose body is simply `"fail("Not yet implemented")"`).**
- With the term "*digits*", we mean '0' through '9'.
- We will also run your code against our set of test cases to make sure that you implemented the functionality of the required methods correctly. Note that our test cases are fairly simple and are not trying to exercise arcane corner cases, so as long as you implement the interface as described, you will be fine.
- Although it is not mandatory, we recommend that you use an IDE to complete the assignment, so that you can use the assignment to also get familiar with this IDE (in case you are not already familiar with it). If you do so, and as we said above, feel free to commit the IDE's related files to the repo as well.
- Please also note that using an IDE should also make it easier to create skeleton code for the class that implements the interface, create and run the JUnit test cases, and so on.
- Before submitting, make sure to compile and run your test cases and to check that they all pass--something else that you can do at the push of a button within most IDEs.
- **You can perform multiple commits as you produce your solution. This is not only fine, but actually very much encouraged.**
- We also encourage you to use a "development" branch and merge your stable version(s) into the "master" branch. If interested, see this site for an example of a possible branching model of this kind.