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.

# Instructions

To complete the assignment, you must complete the following tasks:

* Clone your individual GitHub repository in your local workspace. This is the same repository that you used for the previous assignment and will use for all future individual assignments:  
  https://github.gatech.edu/gt-omscs-se-2023spring/6300Spring23<GT username>.git
* Download the archive [assignment 3.tar.gz](https://drive.google.com/file/d/1OiCdhUUCuQlTwRgcjcOCNijXc2OFKKOe/view?usp=sharing)
* Extract 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>.  
  (If you do not know how to extract a “tar.gz” archive, you should be able to find plenty of resources on how to do that online--ask on Ed Discussion otherwise.)
  + Directory <dir>/src contains, in a suitable directory, the Java interface edu.gatech.seclass.MyStringInterface. It also contains an exception class edu.gatech.seclass.MyIndexOutOfBoundsException, which is used by the interface.
  + Directory <dir>/test contains, in a suitable directory, a template JUnit test class edu.gatech.seclass.MyStringTest.
  + File <dir>/lib/junit-platform-console-standalone-1.9.0.jar, a JUnit library to be used for the assignment
* Your **first task is to develop a Java class called MyString** that suitably implements the MyStringInterface interface that we provided. (The semantics of the methods in the interface should be obvious from their name, the JavaDoc comments in the code, and the test examples in class MyStringTest--ask on Ed Discussion otherwise.) Class MyString should be in the same package as the interface and should also be saved under <dir>/src/edu/gatech/seclass.
* Your **second task is to develop a set of JUnit5 test cases** for class MyString **by completing the provided template test class MyStringTest**, which contains 3 complete test cases and 13 to-be-completed test cases. For each test case in class MyStringTest whose body simply consists of the placeholder “fail("Not yet implemented");”, you must do the following:
  + **Replace the placeholder instruction with a meaningful test case for the corresponding method**, which is indicated in the comments and name of the test.
  + **Replace the text “<Add test description here>” in the test comment with a concise description of the purpose of the test** (e.g., “Count strings in an empty string”) **without modifying anything else in the comment and making sure that your comment consists of a single line** (i.e., does not contain newlines) **and does not contain quotes or other special characters.**
  + **Make sure that every test method has a suitable oracle** (i.e., either an assertion or an expected exception) and that the tests are not trivial (i.e., are not a copy of the provided one and have a specific purpose). In other words, each test should (1) test a specific piece of the functionality, and (2) check that such a piece of functionality behaves as expected.
  + It should be obvious, but please also make sure that all the test cases you created pass when run against your code.
  + In addition, at least **two of the tests that you develop must result in an expected exception** (e.g., NullPointerException). When testing for an expected exception, make sure to assert as follow:

assertThrows(<exception\_class\_name>.class, () -> mystring.<method\_name>(..args))

* + Consider, for example, test case testCountStringsS2():  
    @Test

@Timeout(value = 5000, unit = TimeUnit.MILLISECONDS)  
// Description: <Add test description here>  
public void testCountStringsS2() {  
 fail("Not yet implemented");  
}  
In your submission, you should have a corresponding, complete test for method countStrings in class MyString, together with a concise description of the purpose of the method, such as:  
@Test

@Timeout(value = 5000, unit = TimeUnit.MILLISECONDS)  
// Description: Count strings in an empty string  
public void testCountStringsS2() {  
 <actual test, including the oracle>  
}

# Submission

To submit your solution, you should:

* **Commit dir <dir> and push it to the individual remote GitHub repository we assigned to you**. Make sure not to push your work to any other repo, especially public ones. In the best case, you may end up submitting a commit ID we cannot find. In the worst case, you may be violating the honor code.
* Make 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.
* **Submit on Gradescope a file, called submission.txt that contains, in two separate 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, your assignment will be graded** by compiling your code and running it against both your test cases and a set of test cases, written by the instructors. All tests must pass.[[1]](#footnote-0) After that, which should take a few minutes at most, you will see a report with your grade and some corresponding feedback. The feedback should be self-explanatory (e.g., “ERROR: You have 16 tests instead of 15.”), but please let us know otherwise.

**The grade you will see on Gradescope is your actual grade for the assignment**, unless we find some issues with your submission (e.g., hardcoding of the results, identical tests, or similar).  
**You can resubmit as many times as you want before the deadline**, so you have a chance to address issues with your assignments if Gradescope finds an issue with your submission.

# Notes

* Make sure to submit only the test cases in the template and do not add extra tests. If you feel compelled to add additional tests, feel free to do so in a separate test file, which you can add to your repo but won’t be graded.
* Do not use any **external** library other than the provided libraries. You can obviously use all the standard JDK libraries,[[2]](#footnote-1) and you really should not need any external one.
* The only files that we will use to grade your submissions are:
  + Assignment3/src/edu/gatech/seclass/MyString.java
  + Assignment3/test/edu/gatech/seclass/MyStringTest.java
  + Assignment3/lib/junit-platform-console-standalone-1.9.0.jar

Feel free to commit extra files (e.g., IDEA’s project-related files), as this does not make a difference for your grade.

* You **cannot** modify any of the following:
  + The provided interface (MyStringInterface)
  + The already provided test cases (except for those that you are supposed to implement, obviously, whose body is simply "fail("Not yet implemented")").
  + The test class name, the names of the test cases, and the test comments, except for replacing “<Add test description here>” with a concise description of the purpose of the test. (We understand that it may be advisable to use more meaningful names for the tests, but having numbered tests helps the grading.)
  + The declaration of mystring in the test class.
* You should use Java version 11 to solve the assignment.
* Before submitting, make sure to compile and run your test cases as a collective suite (not only individually) and to check that they all pass. Although gradescope performs the same checks, it is much faster to do that locally, on your machine, beforehand.
* Verify that your final commit ID contains what you intend, and is pushed to the repository, by cloning it in a separate location or viewing it on github.gatech.edu.
* More specifically, 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 Assignment3 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/\* test/edu/gatech/seclass/MyStringTest.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.platform.console.ConsoleLauncher --select-class edu.gatech.seclass.MyStringTest[[3]](#footnote-2)
* 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 “main” branch. If interested, see [this site](http://nvie.com/posts/a-successful-git-branching-model/) for an example of a possible branching model of this kind. (There are many others.)
* From this point on in the course, you should NOT run the git repository reset commands given in Assignment 2, any version of force --push, or any commands deleting branches, tags, or prior commits on your repository, as these commands could cause you to lose prior work.

# Gradescope Feedback/Requests for Clarifications

**Gradescope Feedback:** The test case results that you see in Gradescope tell you whether a given test passed or not. If the test didn't pass, Gradescope should show the difference between the expected and actual output. For ease of consumption, that difference may contain ellipses ("...") to make the feedback more compact by omitting common parts of expected and actual output, and use square brackets to map corresponding parts of the output that differ in the expected and actual output.

**Requests for clarifications:** If you need clarifications on a specific test or 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,
* an inlined, complete copy of the test(s) you reference, if yours (i.e., no screenshots or commit IDs, please), 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.

# Using Intellij IDEA

Although it is not mandatory, we recommend that you use IntelliJ IDEA to complete the assignment, so that you can also get familiar with this IDE (if you aren’t already). To do so, you should open IDEA and do the following (notice the initial, alternative steps):

* If you are in the initial IntelliJ IDEA screen (see image below), do the following:

Graphical user interface, application, Teams

Description automatically generated

* + Click “Open”
  + Select <dir> as the directory to import
* Otherwise, if you are already in a project window, do the following
  + Click menu “File” => “New” => “Project from Existing Sources…

Graphical user interface, text, application, email

Description automatically generated

* + Select <dir> as the project directory  
    A screenshot of a computer

    Description automatically generated with medium confidence
  + Select “Create project from existing sources”  
    Graphical user interface, text, application

    Description automatically generated
  + Click “Next” accepting the default choices (make sure to include the provided libraries and to select Java SDK 11) and then “Finish”:  
    Graphical user interface, text, application

    Description automatically generated  
    Graphical user interface, text, application

    Description automatically generated  
    Graphical user interface, text, website

    Description automatically generated  
    Graphical user interface, text

    Description automatically generated  
    Graphical user interface, text, application

    Description automatically generated
* Please 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.

1. Our test cases make sure that you implemented the functionality of the required methods correctly. They are fairly simple and are not trying to exercise arcane corner cases. [↑](#footnote-ref-0)
2. Internal Java libraries are those documented in the [Java Platform Standard Edition API Specification](https://docs.oracle.com/en/java/javase/11/docs/api/index.html#JavaSE). [↑](#footnote-ref-1)
3. If using a Windows-based system, you may need to run java -cp "classes;lib/\*" org.junit.platform.console.ConsoleLauncher --select-class edu.gatech.seclass.MyStringTest instead. [↑](#footnote-ref-2)