SLList Completion
Due: Day of lab at 11:59PM

1 SLList Completion Specification

1.1 Available Resources

- Lecture slides
- Other sections of the provided code
- me
- The textbook
- **DO NOT** refer to or use online implementations

1.2 Lab Instructions

- This is an *individual* lab.
- Make sure to read through all of the specifications so your submission is complete.
- Follow all the submission steps in the Setup document by the lab deadline.

1.3 Lab Link

The skeleton code for the lab is available at `https://classroom.github.com/a/H5zqL5Vg`.

1.4 Implementation

One of the two goals of this lab is to practice writing Java code that implements operations for a single-linked list data structure. In the lecture, we saw private helper methods and we saw public methods for a few of the operations we want our `SLList` to perform.

I have provided those methods in the **edu.wit.cs.comp2000** package. You will implement the remaining two methods – `remove` and `size`. Descriptions of their expected behavior are included in comments above the methods. For your implementation, consider what helper methods are available, and how they may aid in the work that your methods must do.

1.5 Testing

The other goal of this lab is to gain familiarity with complete code testing, and the syntax of the JUnit testing framework. We will use this framework throughout the course to verify that our data structure implementations work the way we expect them to.

In addition to the `SLList` code, JUnit tests are provided in the **edu.wit.cs.comp2000.tests** package. You can run these tests to see if the `SLList` implementation is performing correctly. The tests that I have provided make sure that **add**, **get**, **set**, and **remove** are behaving as expected. They also provide examples of how the JUnit methods **assertTrue** and **assertEquals** work and examples of how expected exceptions can be detected.

For the rest of the provided test methods (the ones that fail as not implemented), implement tests that check if that method works with the list implementation. You can follow the steps that other test methods take for each test – create a list, modify it, and then assert it behaves the way you expect. Make sure that you actually call the method that you are testing. (You should also delete the `Test not implemented` placeholder lines in the tests you implement.)

1.6 JUnit Assertion Syntax

For each of the JUnit assert* methods, the first argument is the string that prints out if the test fails. Use any of these four assert methods in your tests to confirm the functionality of the `SLList` methods:

- assertTrue
- assertEquals
- assertNotEquals
- assertNull

1.7 Considerations For Each Method

Be considerate in testing your ADT. Your goal is to test all of the possible cases for each `SLList` method. Each test method may have several assert method calls depending on how many possibilities you test for. In future labs, you will be writing more complete test cases like these to test your own code.

1.7.1 Common Edge Cases

How might our list be structured when we run an operation? Consider all the different possibilities for each operation. For each case that applies, we will want to write some code that sets up a list, followed by at least one assert statement.

- Normal results (no bad inputs, no special cases in the operation)
- Bad input index
- Adding/removing data from the front or end
- Expecting an exception

2 Double Check:

- Have you implemented two list operations?
- Have you written three JUnit tests?
- Have you committed/pushed your code from the two files?
- Do you have the correct github username?

3 Grading

Each of the 5 **TODO** methods is worth $\frac{1}{5}$ of the lab grade.

Grades and any comments for the lab will be posted to your project on github. Grades will also be posted to Brightspace, eventually.