Assignment 3

Doubly-Linked Lists

Introduction

In this assignment, you will write and test non-static methods in the A3LinkedList to implement the basic operations of a linked list for Part 1. In Part 2, you will implement two advanced linked list operations, `removeMiddle` and `interleave`.

The automated grading of your assignment will include some different and additional tests to those found in the `A3Tester.java` file, as it does not include a comprehensive set of sets for each method. You are expected to write additional tests until you are convinced each method has full test coverage. The displayResults and test coverage videos provide more information about code testing.

Objectives

Upon finishing this assignment, you should be able to:
- Draw a complete memory trace for a doubly-linked list
- Write a reference-based implementation of the List ADT
- Write your own tests using the displayResults method

Submission and Grading

Attach `A3LinkedList.java` to the BrightSpace assignment page. Remember to click **submit** afterward. You should receive a notification that your assignment was successfully submitted.

If you chose not to complete some of the methods required, you **must** provide a stub for the incomplete method(s) in order for our tester to compile. If you submit files that do not compile with our tester, you will receive a zero grade for the assignment. It is your responsibility to ensure you follow the specification and submit the correct files. Additionally, your code must not be written to specifically pass the test cases in the tester, instead, it must work on all valid inputs. We may change the input values during grading and we will inspect your code for hard-coded solutions. This video explains stubs.

Be sure you submit your assignment, not just save a draft. All late and incorrect submissions will be given a zero grade. A reminder that it is OK to talk about your assignment with your classmates, but not to share code electronically or visually (on a display screen or paper). Plagiarism detection software will be run on all submissions.

Instructions

Part 1:

1. Download all of the `.java` files found in the *Assignments ¿ Assignment 3* page on BrightSpace.
2. Part 1 uses files `A3List.java` (the interface where all the documentation is found), `A3Node.java` (as you will be completing a reference-based implementation), and `A3LinkedList.java`, (the class that implements the A3List interface).
3. Read through the documetation provided in the `A3List.java` interface. There is a lot of information there that will help you with your implementation of each method.
4. Compile and run `A3Tester.java`. Work through implementing each method one at a time. Debug the method until all of the tests path for that method before proceeding to the next method.

Part 2:

1. For `removeMiddle` and `interleave`, make sure you come up with a strategy before writing any code.
2. For `removeMiddle`, think about what scenarios would require one node to be removed, two nodes to be removed, or no nodes to be removed. These are the edge cases you need to think about. Remember to provide an implementation for each of them!
3. For `interleave`, I made a short video to help you understand what the method does. This method also has a number of edge cases you will need to consider!
4. When solving these problems, I strongly recommend drawing out your strategy on paper similar to how we have drawn out linked lists with boxes and arrows in the lectures and labs. Think about what reference arrows need to change to solve the problem, if any temporary reference variables will be needed, and about the order of operations required for your strategy to work. You will save a lot of time if you draw out your strategy *before* writing any code. I recommend drawing out how the reference arrows are updated after each line of code you write, this will help you keep track of everything as you progress through the problem.
5. Remember to test that the reference arrows have been correcly updated after calling the methods. The linked lists for this assignments can be traversed forward from front to back or backward from back to front.

CRITICAL: Any compile or runtime errors will result in a **zero grade** (if the tester crashes it will not be able to award you any points for any previous tests that may have passed). Make sure to compile and run your program before submitting it!