CMSC214

Project #7

Doubly LinkedList

Derya O. Kurin

**Screen Shot of the running tests with JUnit:**

Tests implemented for the constructors:

new TwoWayLinkedList<Integer>(); // no-arg constructor

new TwoWayLinkedList(Integer[] array); constructor with parameter Object

And all the methods that TwoWayLinkedList class implements

add(int index, element);

getFirst()

getLast()

get(int index)

toString()

addFirst(E e)

addLast(E e)

remove(int index)

removeFirst()

removeLast()

listIterator()

listIterator(int index)

List Iterator methods:

next(), previous()

Text

Description automatically generated

**Lessons Learned:**

In the past, I had experience in implementing a Singly linked List in Javascript. This time I learned to work with Doubly Linked List. I learned that the only difference is that doubly linked list allow us to traverse the linked list to the opposite direction as well with an additional data field prev.

In the exercise, I wanted to implement the tests in a separate JUnit file. I did not check my code while writing it so there were many bugs in my code. I learned that it is a better practice to test the part I wrote so far and test it before moving to next methods. It made me lose some time on debugging. But Also it was informative. While I was suspicious that the toString methods did not work properly, by logging the data step by step I realized that my add(int I, E e) method was the one that not working. For me spotting where the problem occurs was more difficult than actually rewriting the methods. Once I understood the dynamics of linked lists, moving forward or back in my methods was not very hard anymore.

Linking nodes to the next and previous while manipulating the order of the nodes in a list was a great exercise.

I will keep this file as a template for my future studies on Linked Lists.

Creating LinkedListIterator using Iterator class actually made me think that we can implement the iteration in any ways depending on our needs with the program.

Using extended classes and creating custom iterators for lists can provide a good amount of flexibility.

|  |  |  |  |
| --- | --- | --- | --- |
| **#** |  | **Y/N** | **Comments** |
|  | Source java files | **Y** |  |
|  | Files: |  |  |
|  | LastNameFirstInitial\_Project07.zip | **Y** |  |
|  | LastNameFirstInitial\_Project07.doc | **Y** |  |
|  | Program compiles | **Y** |  |
|  | Program runs | **Y** |  |
|  | Checklist is completed and included in the Documentation | **Y** |  |
|  | Documentation file: |  |  |
|  | Comprehensive Test Plan ~~(~~integratedinto ~~the main()~~ **the Junit file test cases** | **Y** |  |
|  | Screenshots of running program | **Y** |  |
|  | ~~UML diagram~~ |  |  |
|  | ~~Algorithm: Pseudo-code or Flowchart~~ |  |  |
|  | Lessons Learned | **Y** |  |
|  | Checklist | **Y** |  |