CS 245 - Lab Assignment 5 Fall, 2020

Lab Assignment 5 - Linked Lists

The goal of this assignment is to demonstrate your mastery of Linked Lists implementing your own Linked Lists. **READ THE SUBMISSION DETAILS**

Background

Linked Lists are common interview talking points for questions as there are many things you can ask with them! So here is the assignment:

Create a Linked List data structure using your own custom Node(which should have getters and setters of its own). The data structure should have a `constructor`, `add`, `get`, `remove`, and `reverse` function.

Here are some examples of how the functions should work conceptually :

Add

```
ADD 6
Given List: 1->2->3->4->5->NULL
Output: 1->2->3->4->5->6->NULL
```

Add (with position)

```
ADD 6 POSITION 0
Given List: 1->2->3->4->5->NULL
 Output: 6->1->2->3->4->NULL
```

Get

```
GET POSITION 2
Given List: 1->2->3->4->5->NULL
          Output: 3
```

Remove

```
REMOVE POSITION 4
Given List: 1->2->3->4->5->NULL
   Output: 1->2->3->4->NULL
```

Reverse

```
Given List: 1->2->3->4->5->NULL
 Output: 5->4->3->2->1->NULL
```

Requirements

- Make a LinkedList Data Structure. This should exist in a class called `LinkedList`
- The Data Structure must include ***6 Functions .*** Each of which will be described more in detail below:

    - **Constructor**: Your list must have a constructor which will make an instance of your linked list. It will take in a variable of type `<E>`.

      Function Signature: `public LinkedList();`

    - **Add**: Your code must have **TWO add** functions. One of which will take in `Node of type E` and add it to the end of the LinkedList. The other one takes in both the `Node of type E` and the `Position` of which it is to be added into. This should not return anything.

      Function Signatures:
      ```
      public void add(E item);
      public void add(E item, int position);
      ```

    - **Get** : Your list must have a get function. This takes in an integer `position` which then returns the node at that position in the list. If the position is invalid, return null.

      Function Signature: `public E get(int position);`

    - **Remove:** Your list must have a remove function. The function takes in an integer `position`, which it then both removes and returns the `Node of type E` at that position.

      Function Signature: `public E remove(int position);`

    - **Reverse:** Your list must have a reverse function. The function reverses the whole linked list from head to tail. It should take in a parameter `Node of type E named head.` And return back the new head once the function is finished.

      Function Signature: `public E reverse(E head);`

- When complete your code should only have one file, `LinkedList.java.` Store this in a github repo titled “Lab_Assignment 4”. The repo **SHOULD ONLY HAVE ONE FILE.**

Submission

Submit the GitHub repository link for your implementation on Canvas.

THIS REPO SHOULD ONLY HAVE ONE FILE AND YOUR READ ME. ANY DEVIATION OF EXTRA FILES WILL RESULT IN A 0.

Your readme should have any additional information that the grader may need.

Grading

Your grade for this assignment will be determined as follows:

- 75% = Implementation: your class implementations must run successfully with the function signatures and data provided. It must produce the expected results, a sample of which appears in the Implementation section above. Any deviation from the expected results results in 0 credit for implementation.
    - 25% LinkedList works as expected. All 5 functions work and produce the expected output
    - 50% Reversing the LinkedList
- 15% = Decomposition: in the eyes of the grader, your solution follows the suggestions above or otherwise must represent a reasonable object-oriented and procedural decomposition to this problem.
- 10% = Style: your code must be readable to the point of being self-documenting; in other words, it must have consistent comments describing the purpose of each class and the purpose of each function within a class. Names for variables and functions must be descriptive, and any code which is not straightforward or is in any way difficult to understand must be described with comments.