Assignment 1 CS3345

Objectives

The objectives of this assignment are:

- To review using doubly-linked lists
- To implement a List using Generics
- To implementation several methods for a doubly-linked list
- To implement a Java iterator

The Lab

Java Files:

- List.java specifies the interface for a list using Generics and includes the printBackwards method.
- DoublyLinkedListTest.java is a set of tests that test your implementation of the DoublyLinkedList.
- DoublyLinkedList.java is a partially completed implementation of the List interface using doubly-linked lists.

When you examine **`DoublyLinkedList.java`**, you will notice several methods that are not complete (with a comment `//TODO`), so they only compile, but are lacking a proper implementation. You will implement all methods, and test them using functions that you will add to **`DoublyLinkedListTest.java`** main function.

Step 1. Print an empty list.

Implement the **`print`** method in the **`DoublyLinkedList`** class. You should start from the head, and traverse by the next references.

Within the main **`DoublyLinkedListTest.java`**, finish the function called **`testPrintEmptyListForward`** that:

- creates a **`DoublyLinkedList`** object
- calls print on the empty **`DoublyLinkedList`**.

Run your **`main`** method and make sure it does not crash.

Step 2. Print an empty list backwards.

Implement the **`printBackwards`** method in the DoublyLinkedList class. You should start from the tail and traverse by the previous references.

Within the main **`DoublyLinkedListTest.java`**, finish the function called **`testPrintEmptyListForward`** that:

- creates a **`DoublyLinkedList`** object
- calls **`printBackwards`** on the empty **`DoublyLinkedList`**.

Run your **`main`** method and make sure it does not crash.

Step 3. Get from the doubly-linked list

The **`get`** method has an integer parameter that is the *position* in the list. *position* start from 0 for the first element. You should return the T **data** (not the **Node**) at that position. If the position does not exist, you should return **null**.

Begin by handling the cases where you should return null.

After you are sure the position is valid, write a loop that only traverse as far as the *position* parameter. Again, you should return the T **data** (not the **Node**) at that position.

Within the main **`DoublyLinkedListTest.java`**, write a function called **`testEmptyGet`** that works en empty list. Of course, it should return null no matter what *position* you pass as an argument, since the list is empty.

Step 4a. Prepend to the doubly-linked list: drawing and pseudocode

Draw on paper what it will look like to add a new generic object to the beginning of a doubly-linked list. You should draw three examples:

- What happens when you prepend to an empty doubly-linked list?
- What happens when you prepend to a doubly-linked list with one item in it?
- What happens when you prepend to a doubly-linked list with three items in it?

For each example:

1. Draw the original doubly-linked list, including (a) the head of the doubly-linked list, (b) the tail of the doubly-linked list, (c) the number of elements, (d) every node in the doubly-linked list, each with the data, next and previous.
2. Draw the new Node that needs to be created for the new generic object under your original doubly-linked list. Be sure to draw the data, next and previous fields in the Node.
3. Draw the new arrows necessary for the new doubly-linked list after appending the new Node. Number each of these arrows. Each new arrow will require an assignment in your Java code.
4. Double-check that the instance variables (the head, tail and the number of elements) are set correctly. If they aren't, you'll need to modify them.

Step 4b. Prepend to the doubly-linked list: Java code.

Implement the **`addFirst`** method in the **`DoublyLinkedList`** class. The **`addFirst`** method adds a new generic object to the *start* of the doubly linked list.

If you implement **`addFirst`** correctly, you should be able to get the output of **`testPrependAndGet`** correct by uncommenting it in the main function. If passed, uncomment **`testPrependForwards`** and **`testPrependBackwards`** one at a time, to see if you get the list printed in the right order. If not, (or if you get NullPointerException), it is likely that you have corrupted your previous

references.

Step 5a. Append to the doubly-linked list: drawing and pseudocode.

Draw on paper what it will look like to append a new generic object to a doubly-linked list. You should draw three examples:

- What happens when you append to an empty doubly-linked list?
- What happens when you append to a doubly-linked list with one item in it?
- What happens when you append to a doubly-linked list with three items in it?

Step 5b. Append to the doubly-linked list: Java code.

Now implement the **`addLast`** method. Be sure your code will handle all three situations that you drew in the previous example.
Same way as in 5, uncomment **`testAddLastAndGet(), testAddLastForwards(), testAddLastBackwards()`** one at a time, and verify you get the right output.

Step 6. Implement getLength and isEmpty.

Write the **`getLength`**() and **`isEmpty()`** methods. Since these don't modify the doubly-linked list, they should be very similar to the same methods in the linked list seen in class. Uncomment **`testIsEmpty()`** and **`testGetLength()`** to verify your code.

Step 7a. Remove from the doubly-linked list: drawing.

First draw on paper what it will look like to remove an object in a doubly-linked list. You will need to search the entire doubly-linked list for the item you want to remove. If the item is found, you should remove the first version of that item. You should draw five examples (again, have each person in your group should draw at least one example):

- Assuming you have a doubly-linked list with three items in it, what happens when you remove the first item in the doubly-linked list?
- Assuming you have a doubly-linked list with three items in it, what happens when you remove the middle item in the doubly-linked list?
- Assuming you have a doubly-linked list with three items in it, what happens when you remove the last item in the doubly-linked list?
- Assuming you have a doubly-linked list with three items in it, what happens if you do not find the item?
- Assuming you have a doubly-linked list with one item in it, what happens when you remove that item from the doubly-linked list?

Go through the same four-step process as in Step 4a.

Step 7b. Remove from the doubly-linked list: Java code.

Now implement the remove method. Be sure your code will handle all four situations that you drew in the previous example.

Then you can run the following tests:

- `testRemoveFromEmptyList();`
- `testRemoveFromListWithOneElementNegative();`
- `testRemoveFromListWithOneElementPositive();`
- `testRemoveFromListWithTwoElementNegative();`
- `testRemoveFromListWithTwoElementPositive();`
- `testRemoveFromListWithThreeElementNegative();`
- `testRemoveFromListWithThreeElementPositive();`

and see if they pass the tests properly.

Step8: Iterator for DoublyLinkList

Add an iterator class definition for the **`DoublyLinkList`**. Follow the example of the slides. Then finish the **`testIterator`** method that create a list, obtain an iterator, and then uses it to print the content of the list.