Lab 3: Doubly Linked List with Performance

```
Location: CourseWeb -> Labs/Recitations -> Lab 3: Doubly Linked List
Download the following files:
  1. DLinkedList.java
  2. DLinkedListTester.java
```

Introduction

Recall the implementation of the ADT List using **singly** linked list discussed in class. In this implementation, the class `LList.java` contains only two instance variables, `firstNode` of type `Node`, and `numberOfEntries` of type `int`. Recall that when we implementation the method `getEntry()`, we use the method `getNodeAt()`. Every time the method `getEntry()` is called, the method `getNodeAt()` has to traverse the link chain from the first node to the node associated with the given position. Imagine if our list contains 100,000 entries and we call the method `getEntry()` 10,000 times, this will take a while because of the method `getNodeAt()`. To improve the performance in this situation, there are various solutions.

- **Solution 1**: We can improve the performance of our original implementation by using **doubly linked list** instead of singly linked list, and adding another instance variable named `lastNode` of type `Node` which always refer to the last node of the link chain. In doing so, it allows us to traverse the link chain from the first node as well as the last node. Imagine if the method `getEntry()` is called with a given position less than the number of entries divided by 2, the method `getNodeAt()` should traverse the link chain from the first node. But if the method `getEntry()` is called with a given position greater than the number of entries divided by 2, the method `getNodeAt()` should traverse the link chain from the last node. In doing so, the total length that the method `getNodeAt()` has to traverse the link chain should be cut in half (in theory). As a result, faster performance compare to the original implementation.

- **Solution 2**: We can further improve the performance of the previous solution by adding two more instance variables to solution 1. The first added instance variable is the instance variable named `middleNode` of type `Node` which always refer to the node in the middle of the link chain. But how do we know which position this middle node is associated to? That is why we need another instance variable. The second added instance variable is the instance variable named `middlePosition` of type `int` which always indicate the position of the node `middleNode` in our list. This solution allows us to traverse the link chain in four ways.

    1. From the first node forward when the given position is less than the number of entries divided by 4.
    2. From the middle node backward when the given position is less than the number of entries divided by 2 but greater than the number of entries divided by 4.
    3. From the middle node forward when the given position is greater than or equal to the number of entries divided by 2 but less than the number of entries times 3 and divided by 4.
    4. From the last node backward when the given position is greater than the number of entries times 3 and divided by 4.

What to do

For this lab, you are going to implement solution 1 and solution 2 explained in previous section. From the starter code (see the code `DLinkedList.java`), this is a very strip down version of our `LList.java` discussed in class. It consists of five instance variables, `firstNode`, `lastNode`, `middleNode`, `numberOfEntries`, and `middlePosition`. It contains only methods `add()` and three methods for getting an entry from a given position named, `getEntry()`, `getEntry1()`, and `getEntry2()`. **Read carefully**. The method `getEntry()` uses the method `getNodeAt()` which is exactly the same implementation in our `LList.java` discussed in class. The method `getEntry1()` uses the method `getNodeAt1()` and the method `getEntry2()` uses the method `getNodeAt2()`. Note that the methods `getNodeAt1()` and the method `getNodeAt2()` simply use the method `getNodeAt()`.

What you need to do is to modify the method `getNodeAt1()` and the method `getNodeAt2()` using the solution 1 and solution 2 discussed in previous section, respectively. Do not modify any other methods or insert any other instance variables. Note that you do not have to worry about setting the value of instance variables `firstNode`, `numberOfEntries`, `lastNode`, `middleNode`, and `middlePosition`. They have been taken care of by the method `add()`. You simply use them to develop solution 1 and solution 2.

Test Class

A test class (`DLinkedListTester.java`) is provided. Simply compile and run this test class. This program will test your `DLinkedList.java` and show how long it takes for each method to complete its task. An example of the output if the program `DLinkedListTester` is shown below:

```
Adding data into the list: DONE
Dry run for the method getEntry(): DONE
Time the method getEntry(): DONE
Dry run for the method getEntry1(): DONE
Time the method getEntry1(): DONE
Dry run for the method getEntry2(): DONE
Time the method getEntry2(): Done

The method getEntry() took 2942.0 millisecond.
The method getEntry1() took 1182.0 millisecond.
The method getEntry2() took 593.0 millisecond.

If getEntry1() took roughly about half the time of getEntry(), you get 5 points.
If getEntry2() took roughly about a quarter the time of getEntry(), you get 5 points.
```

If you implement the method `getEntry1()` and `getEntry2()` correctly, `getEntry1()` should take roughly about half the time of the method `getEntry()` and `getEntry2()` should take roughly about a quarter the time of the method `getEntry()`.

Due Date and Submission

For the due date, please check the lab in the CourseWeb. Submit your `DLinkedList.java` to the CourseWeb under this lab by the due date. **No late submission will be accepted**.