Data Structures | **Programming Project 2**

CSC2-242 Fall 2024 (Ali Azhari) | **Due Date: Thursday October 17 2024**

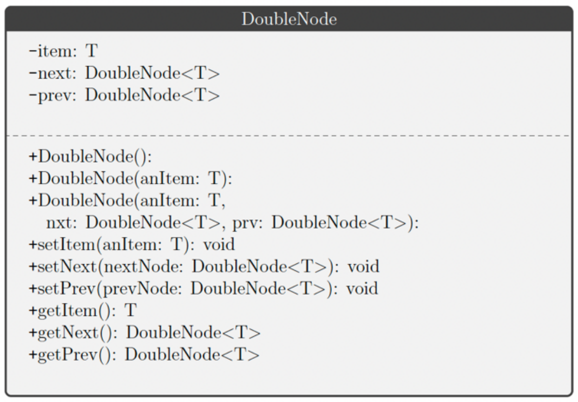
Please solve the problem(s) alone. Remember to test your solution thoroughly. Code that does not work correctly will lose credit. *Code that does not compile will receive no more than 70%.* Please submit a digital copy of your source code on Moodle. The digital copy of your code should be a zip file of the project folder named with your last name followed by the project number. For example, *azhari2.zip* would be my zip file for project 2. Good luck!

In this project you will contemplate an interesting form of the list ADT called a *Doubly Linked*. This is a data structure that is similar to a linked list but, maintains not only a pointer to the next node in the list (the successor) but the previous node in the list (the predecessor). Once you implement your double linked list, DoubleLinkedList, (see phase I) you will enhance it to increase the efficiency of getNodeAt (see phase II). You must complete phase I before attempting phase 1.

Phase I: Double Linked List

In this phase of the project you will implement a doubly linked list as described above. As a hint I would start from the Node.java and LinkedList.java files from class and modify the files.

Rename the file Node.java to DoubleNode.java and modify the file such that it conforms to the following UML:

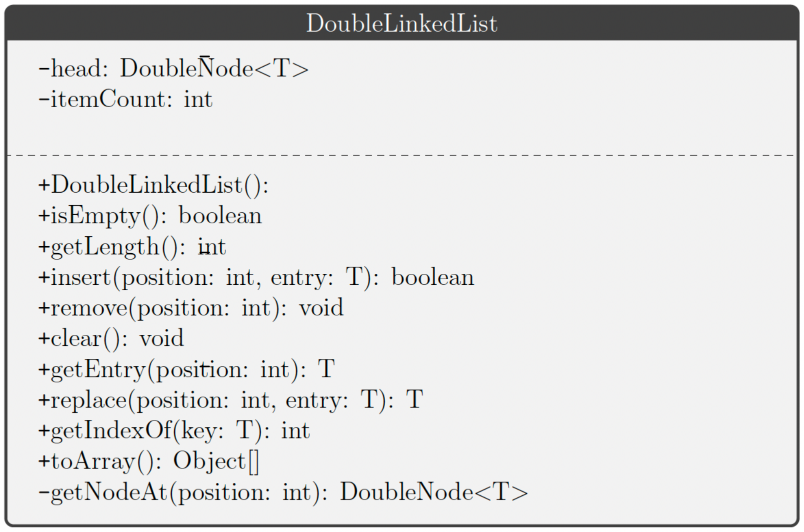

DoubleNode
-item: T
-next: DoubleNode<T>
-prev: DoubleNode<T>
+DoubleNode():
+DoubleNode(anItem: T):
+DoubleNode(anItem: T, nxt: DoubleNode<T>, prv: DoubleNode<T>):
+setItem(anItem: T): void
+setNext(nextNode: DoubleNode<T>): void
+setPrev(prevNode: DoubleNode<T>): void
+getItem(): T
+getNext(): DoubleNode<T>
+getPrev(): DoubleNode<T>

You need to create or modify the following methods:

- Add code to the constructors that properly initializes the prev reference.

- Implement the new accessor methods in the natural way.

Rename the file LinkedList.java to DoubleLinkedList.java and modify the file such that it conforms to the following UML:



```
DoubleLinkedList
-head: DoubleNode<T>
-itemCount: int
-----------------------------------------
+DoubleLinkedList():
+isEmpty(): boolean
+getLength(): int
+insert(position: int, entry: T): boolean
+remove(position: int): void
+clear(): void
+getEntry(position: int): T
+replace(position: int, entry: T): T
+getIndexOf(key: T): int
+toArray(): Object[]
-getNodeAt(position: int): DoubleNode<T>
```

You should be sure to modify all methods such that they properly set the previous pointer. The methods to modify are:

- the constructors,
- insert, and
- remove.

You will need to add a brand new method, getIndexOf. This method should be written such that it returns the index of the key in the list or -1 if the key is not found.

You should override the toString method so that it returns a String that represents the content of the list. For example,

```
[1]   5
[2]   13
[3]   17
[4]   1
```

if the list contains 4 Integers or

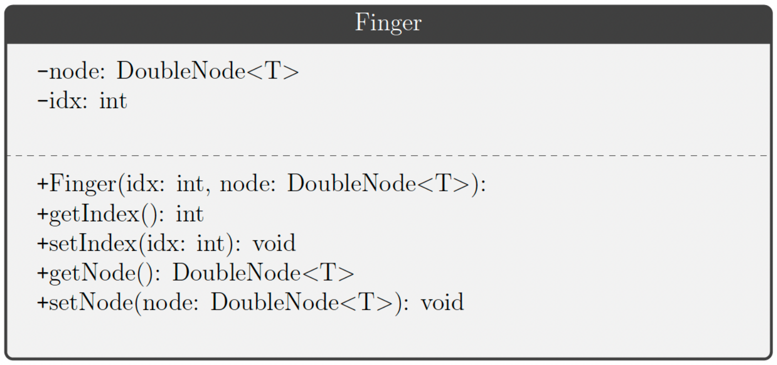
```
Empty List.
```

if the list is empty.

Phase II: Finger Searching

As we saw in class, searching through a linked list is a time-consuming process. To speed up search we can use *finger searching*. In a finger search, extra references called the fingers(finger attribute below) are maintained in such a way that they refer to the nodes in a proportional way. To successfully implement finger searching you will need to implement a Finger class and update the DoubleLinkedList class.

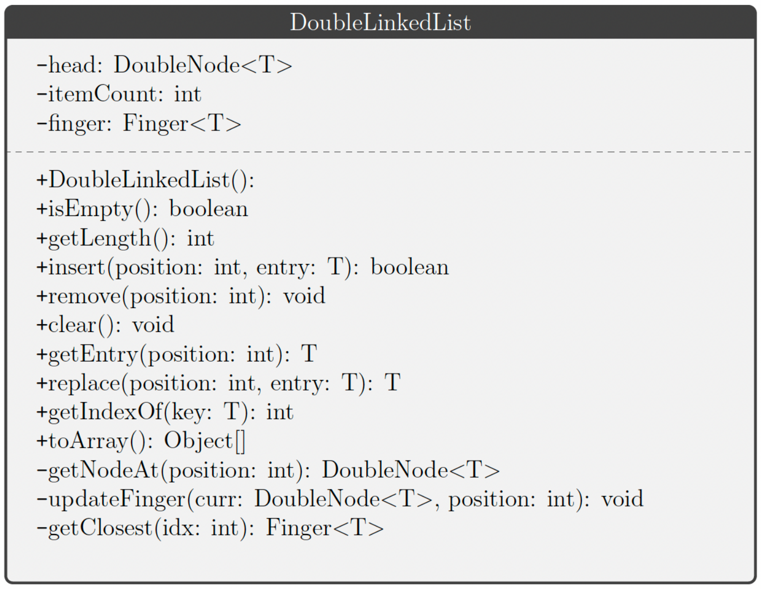
The Finger class which is in package list has the following UML class diagram.



```
Finger
-node: DoubleNode<T>
-idx: int
+Finger(idx: int, node: DoubleNode<T>):
+getIndex(): int
+setIndex(idx: int): void
+getNode(): DoubleNode<T>
+setNode(node: DoubleNode<T>): void
```

- The constructor should set this.node and this.idx appropriately.
- The accessor methods behave as you would expect (idx is called index for the purpose of accessors).

Next, you must modify DoubleLinkedList so that it conforms to the update UML:



DoubleLinkedList

```
-head: DoubleNode<T>
-itemCount: int
-finger: Finger<T>
-----------------------------------------
+DoubleLinkedList():
+isEmpty(): boolean
+getLength(): int
+insert(position: int, entry: T): boolean
+remove(position: int): void
+clear(): void
+getEntry(position: int): T
+replace(position: int, entry: T): T
+getIndexOf(key: T): int
+toArray(): Object[]
-getNodeAt(position: int): DoubleNode<T>
-updateFinger(curr: DoubleNode<T>, position: int): void
-getClosest(idx: int): Finger<T>
```

Specific updates are:

- Add a new attribute called fingers of type Finger (This is an array)
- The default constructor should be modified to allocate an array fingers (default) 4 fingers
- The update̅Finger method is responsible for updating the fingers by distributing all the fingers throughout the list.
- The getClosest takes an index idx and finds the closest reference whether that is head or the finger. The method returns a Finger that contains the index and node associated with the closest reference. Recall the linear distance formula is defined as $|x_2 - x_1|$
- insert, remove, and replace must be modified to call updateFinger after modifying itemCount. When removing a node, you should call updateFinger again

T

- Modify getNodeAt so that is uses getClosest to find the closest finger and use the finger to locate the node of interest. As a note, you may need to follow previous pointers if the position is closest to finger pointer but, the index of the finger is greater than position.

+

Phase III: Simulator

In this phase you will create a simulator that reads from the file commands.csv, and executes the commands read. Each row starts with the command and followed by either the index or the item.
Print command is followed by either. 0 (List is not sorted), or by 1 (sorted list).
You need to implement a sort method in the DoubleLinkedList.java using a Comparator as an argument.

General Reminders

- If you desire, you may add *private methods* to help you with writing the code. You may *not* however add or subtract from any public interface nor are you allowed to remove private methods.
- **Start early**
- Make sure to check your code against the rubric *before* submission.
- You are encouraged to see me for help with the project.

Submission

Submissions in this class come in two parts:

1. **Digital Submission**: Since our projects now have multiple files that need to be graded, I ask that you submit a zip file containing the project folder associated with the assignment. Failure to follow these directions will result in a deduction of up to **5 points from your grade**.

Grading

Your grade on this assignment will be defined as follows:

- Program follows course style guidelines and has good comments (*20 points*).
- Program passes tests (*10 points*).
- The doubly linked list has been correctly implemented (*20 points*)
- The finger search has been correctly implemented (*20 points*)

- The main method is correctly implemented (*10 points*)
- Comparator and Sort implementations (*20 points*)

Your code will be reviewed against various AI resources to detect any instances of plagiarism.

Good luck!