AZHAR ALI
023-23-0314
Sec A

LAB 3

DSA 2024

1. Understand provided code and implement all required methods (with all possible exceptions) in DoubleLinkedList.

DoubleLinkedList.java > Node

```java
class Node {
    String name;
    Node prev, next;

    Node(String name) {
        this.name = name;
        this.prev = null;
        this.next = null;
    }
}
public class DoubleLinkedList {
    Node head;


    public void insertAtBeginning(String name) {
        Node n1 = new Node(name);
        if (head == null) {
            head = n1;
        } else {
            n1.next=head;
            head.prev=n1;
            head=n1;
        }
        head.prev=null;
    }
    public void insertAtBeginning(Node node){
        if (head == null){
            head=node;
        }
        else{
            node.next=head;
            head.prev=node;
            head=node;
        }
    }


    public void insertAtEnd(String name) {
        Node n1 = new Node(name);
        if (head == null) {
            head = n1;
        } else {
            Node temp = head;
            while(temp.next!=null){
```

```java
    public void insertAtEnd(String name) {
            while(temp.next!=null){
                temp=temp.next;
            }
            temp.next=n1;
            n1.prev=temp;
        }
    }
    public void insertAtEnd(Node node){
        if (head == null) {
            head = node;
            head.prev=null;
        } else {
            Node temp = head;
            while(temp.next!=null){
                temp=temp.next;
            }
            temp.next=node;
            node.prev=temp;
            node.next=null;
        }
    }

     void insertBeforeName(String name, Node node){
        if(head==null){
            System.out.println("There is No any element");
        }
        else{
            Node temp = head;
            while (temp != null && !temp.name.equals(name)){
               temp=temp.next;
            }
            if(temp.name==null){
                System.out.println("Element Not Found.");
            }
            else{
                Node PreTemp = temp.prev;
                PreTemp.next = node;
                node.prev = PreTemp;
                node.next = temp;
                temp.prev = node;
            }  }}
```

```java
    public void printAll() {
        if (head == null) {
            System.out.println("List Empty!");
        } else {
            Node temp = head;
            while(temp!=null){
                System.out.print("-->"+temp.name);
                temp=temp.next;
            }
            System.out.println();  }
    }
```

```java
public void insertAfterName(String name, Node node) {
    if (head == null) {
        System.out.println("There is no element in the list.");
        return;
    }

    Node temp = head;
    while (temp != null && !temp.name.equals(name)) {
        temp = temp.next;
    }
    if (temp == null) {
        System.out.println("Node with name " + name + " not found.")
        return;
    }
    if (node.prev != null || node.next != null) {
        System.out.println("The node to insert is already part of th
        return;
    }
    node.next = temp.next;
    node.prev = temp;
    if (temp.next != null) {
        temp.next.prev = node;
    }
    temp.next = node;
}
public void makeCircular() {
    if (head != null) {
        Node temp = head;
        while (temp.next != null) {
            temp = temp.next;
        }
        temp.next = head;
        head.prev = temp;
    }
}
```

Main

```java
public class DoubleLinkedList {
    }

    Run | Debug
    public static void main(String[] args) {
        DoubleLinkedList D1 = new DoubleLinkedList();
        Node node = new Node(name:"Azhar1");
        Node node2 = new Node(name:"Azhar2");
        Node node3 = new Node(name:"Azhar3");
        Node node4 = new Node(name:"Azhar4");
        D1.insertAtBeginning(name:"Ali");
        D1.insertAtBeginning(name:"Ahmed");
        D1.insertAtEnd(name:"Sara");
        D1.insertAtEnd(name:"Hyder");
        D1.insertAtBeginning(name:"Khan");
        D1.insertAtBeginning(node);
        D1.insertAtEnd(node2);
        D1.insertAfterName(name:"Khan",node3);
        D1.insertBeforeName(name:"Ali", node4);
        D1.printAll();
    }
}
```

Output

```
-->Azhar1-->Khan-->Azhar3-->Ahmed-->Azhar4-->Ali-->Sara-->Hyder-->Azhar2
```

2. In previous labs, you have designed single linkedlist with all possible common methods with only head. Now your task is to implement following methods (Single/Double LL) but this time you have to make another variable say tail for accessing last element directly. - All types of methods for inserting (Beginning, End) - All types of methods for removing (Beginning, End) Compare these methods with those which were designed without tail.

```java
class Node {
    int data;
    Node next;

    Node(int data) {
        this.data = data;
        this.next = null;
    }
}

class TailTypeLinkedList {
    Node head;
    Node tail;
    int size;

    void addToBack(int data) {
        Node node = new Node(data);
        if (head == null) {
            head = node;
            tail = node;
        } else {
            tail.next = node;
            tail = node;
        }
        size++;
    }

    void addToFront(int data) {
        Node node = new Node(data);
        node.next = head;
        head = node;
        if (tail == null) {
            tail = node;
        }
        size++;
    }
```

```java
    void removeFromFront() {
        if (head == null) {
            System.out.println("List is empty");
            return;
        }
        head = head.next;
        if (head == null) {
            tail = null;
        }
        size--;
    }
    void removeFromBack() {
        if (head == null) {
            System.out.println("List is empty");
            return;
        }
        if (head.next == null) {
            head = null;
            tail = null;
        } else {
            Node n = head;
            while (n.next.next != null) {
                n = n.next;
            }
            n.next = null;
            tail = n;
        }
        size--;
    }
    boolean isEmpty() {
        return head == null;
    }
```

Main Method

```
    void printList() {
        Node node = head;
        while (node != null) {
            System.out.print(node.data + " ");
            node = node.next;
        }
        System.out.println();
    }
    Run | Debug
    public static void main(String[] args) {
        TailTypeLinkedList T1 = new TailTypeLinkedList();
        T1.addToFront(data:10);
        T1.addToBack(data:30);
        T1.addToBack(data:40);
        T1.addToBack(data:20);
        T1.printList();

        T1.removeFromFront();
        T1.removeFromBack();
        T1.printList();

        T1.removeFromFront();
        T1.removeFromBack();
        T1.printList();
    }
}
```

Output

```
10 30 40 20
30 40
```

3. **Design a method that takes head as param and detect whether linked list contains cycle or not? Cycle exists in a linked list if any node is visited twice while traversing whole traversing.**

```java
public class LinkedListCycle {
    public boolean CheckCycle(Node head) {
        if (head == null) {
            return false;
        }
        Node slow = head;
        Node fast = head;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;

            if (slow == fast) {
                return true;
            }
        }
        return false;
    }
    Run | Debug
    public static void main(String[] args) {
        LinkedListCycle detector = new LinkedListCycle();

        //cant written add method so that
        Node head = new Node(data:1);
        head.next = new Node(data:2);
        head.next.next = new Node(data:3);
        head.next.next.next = new Node(data:4);
        head.next.next.next.next = head.next;

        System.out.println("Cycle detected: " + detector.CheckCycle(head));
    }
}
```

Output

```
Cycle detected: true
PS D:\Semester 3\DSA LAB>
```