**Task1:**

Consider the Java implementation of the Singly Linked List (SLL) shared with you on Moodle:

1. Rewrite the methods for inserting in the middle of the list, deleting the tail, and deleting from the middle using a for loop instead of a while loop.
2. Run and test your code. Attach screenshots of modified code and output.

class LinkedList {

  Node head;

  // Create a node

  class Node {

    int data;

    Node next;

    Node(int d) {

      data = d;

      next = null;

    }

  }

  // Insert at the beginning

  public void insertAtBeginning(int new\_data) {

    // insert the data

    Node new\_node = new Node(new\_data);

    new\_node.next = head;

    head = new\_node;

  }

  // Insert after a node

  public void insertionAtMiddle(Node prev\_node, int new\_data) {

    if (prev\_node == null) {

      System.out.println("The given previous node cannot be null");

      return;

    }

    Node new\_node = new Node(new\_data);

    new\_node.next = prev\_node.next;

    prev\_node.next = new\_node;

  }

  // Insert at the end

  public void insertAtEnd(int new\_data) {

    Node new\_node = new Node(new\_data);

    if (head == null) {

      head = new Node(new\_data);

      return;

    }

    new\_node.next = null;

    Node last = head;

    while (last.next != null)

      last = last.next;

    last.next = new\_node;

    return;

  }

  // Delete a node

  void deleteNode(int position) {

    if (head == null)

      return;

    Node temp = head;

    if (position == 0) {

      head = temp.next;

      return;

    }

    // Find the key to be deleted

    for (int i = 0; temp != null && i < position - 1; i++)

      temp = temp.next;

    // If the key is not present

    if (temp == null || temp.next == null)

    {

       // System.out.println("the element at given position is "+temp.data);

      return;

    }

    // Remove the node

    Node next = temp.next.next;

    temp.next = next;

  }

  // Print the linked list

  public void printList() {

    Node tnode = head;

    while (tnode != null) {

      System.out.print(tnode.data + " ");

      tnode = tnode.next;

    }

  }

  public static void main(String[] args) {

    LinkedList obj = new LinkedList();

    obj.insertAtEnd(1);

    obj.insertAtBeginning(2);

    obj.insertAtBeginning(3);

    obj.insertAtEnd(4);

    System.out.println("Inserting an element at middle");

    obj.insertionAtMiddle(obj.head.next, 5);

    System.out.println("Linked list: ");

    obj.printList();

    System.out.println();

    System.out.println("Deleting Tail element");

    obj.deleteNode(4);

    obj.printList();

    System.out.println();

    System.out.println("\nAfter deleting a middle element: ");

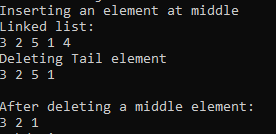
    obj.deleteNode(2);

    obj.printList();

  }

}

**Output**

****

**TASK2:**

Based on your understanding of the doubly linked list (DLL) concepts and algorithms, use the SLL Java implementation as a basis to create a Java program to implement a doubly linked list of Strings. Your program should contain the following methods:

1. Insert at the beginning of the DLL
2. Insert at the end of the DLL
3. Insert in the middle of the DLL
4. Delete the first node
5. Delete the last node
6. Delete the middle node
7. Display the list
8. Return the size of the DLL

Run and test your code. Attach screenshots of code alongside the output.

For both parts of this assignment, each pair group member is asked to specify which parts they have completed. Failure to do so will result in loss of marks. Your final deliverable should include:

* This file updated with screenshots of operational code with output for both parts. Your file should also contain an account of each member’s contribution. Please save the file as a PDF. Word documents will not be considered a submission.
* Java files for both parts.

import java.util.\*;

import java.io.\*;

class node{

  int num;

  node next;

  node prev;

  public node(int num)

  {

    this.num=num;

    next=null;

    prev=null;

    //size=0;

  }

}

class list{

  //int size=0;

 node head=null;

 node tail=null;

  boolean isEmpty()

  {

    if(head==null)

    {

      return true;

    }

    else

    return false;

  }

  void insertAtBeginning(node n)

  {

     if(isEmpty())

     {

         head=n;

     }

else{

    n.next=head;

    head.prev=n;

    head=n;

}

  }

  /\*void insertAtBeginning(String name)

  {

node n=new node(name);

    {

     if(isEmpty())

     {

         head=n;

     }

else{

    n.next=head;

    head.prev=n;

    head=n;

}

  }

  }\*/

void insertAtEnd(node n)

{

 if(isEmpty()){

 head=n;

 //size++;

 }

 else{

  node curr=head;

while(curr.next!=null)

{

  curr=curr.next;

}

curr.next=n;

n.prev=curr;

 }

}

void display()

{

  node curr=head;

  while(curr!=null)

  {

    System.out.print(curr.num+" ");

    curr=curr.next;

  }

  System.out.println();

 // System.out.print(curr.data);

}

void size()

{

  int count=0;A

  node curr=head;

  while(curr!=null)

  {

    ++count;

    curr=curr.next;

  }

  System.out.println("The Size of this Linked List is "+ count);

 // System.out.print(curr.data);

}

void insertAtMid(node n,int num)

{

  if(isEmpty())

  System.out.println("list is empty");

  else{

        node curr=head;

        node back=null;

        while(curr.num!=num)

        {

            back=curr;

            curr=curr.next;

        }

    node  temp=back.next;

      back.next=n;

      curr.prev=back;

      n.next=temp;

      curr.prev=n;

//      size++;

  }

}

 public void deleteFromStart() {

        //Checks whether list is empty

        if(head == null) {

            return;

        }

        else {

            //Checks whether the list contains only one element

            if(head != tail) {

                //head will point to next node in the list

                head = head.next;

                //Previous node to current head will be made null

                head.prev = null;

            }

            else {

                head = tail = null;

            }

        }

    }

 public void deleteFromLast() {

     node curr=head;

        //Checks whether list is empty

        if(head == null) {

            return;

        }

        else {

            //Checks whether the list contains only one element

            while(curr.next.next != null)

                //head will point to next node in the list

                curr = curr.next;

                //Previous node to current head will be made null

               node t=curr.next.next;

              // t.prev=null;

               curr.next=null;

        }

    }

    public void deleteFromMid(int value){

    node curr=head;

node temp=null;

    while(curr.num!=value)

    {

      temp=curr;

      curr=curr.next;

    }

    temp.next=curr.next;

    curr.next.prev=temp;

    //curr.next=null;

    //curr.prev=null;

    }

public static void main(String[] args)

{

  node node1 =new node(1);

  node node2 =new node(2);

  node node3 =new node(3);

  node node4 =new node(4);

  node node5 =new node(5);

  node node6 =new node(6);

  list obj=new list();

 // obj.insert(node1);

  obj.insertAtEnd(node2);

   obj.insertAtEnd(node3);

    obj.insertAtEnd(node4);

    System.out.println("the linked list is: ");

    obj.display();

  System.out.println();

  System.out.println("Insertion at the Tail:");

obj.insertAtEnd(node5);

     obj.display();

     System.out.println();

      System.out.println("Insertion at the Mid:");

     obj.insertAtMid(node6,3);

     obj.display();

     System.out.println();

      System.out.println("Insertion at the Beginning:");

     obj.insertAtBeginning(node1);

     obj.display();

     //Deletion

obj.deleteFromStart();

System.out.println("After deleting node from start");

obj.display();

  System.out.println();

obj.deleteFromLast();

System.out.println("After deleting node from tail");

obj.display();

  System.out.println();

obj.deleteFromMid(3);

System.out.println("After deleting node from middle");

obj.display();

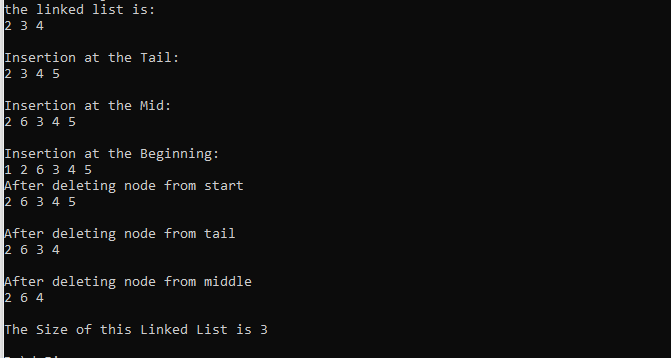
  System.out.println();

  obj.size();

}

}

**OUTPUT:**

****