1. Given a singly linked list, delete middle of the linked list. For example, if given linked list is 1->2->3->4->5 then linked list should be modified to 1->2->4->5.If there are even nodes, then there would be two middle nodes, we need to delete the second middle element. For example, if given linked list is 1->2->3->4->5->6 then it should be modified to 1->2->3->5->6.If the input linked list is NULL or has 1 node, then it should return NULL

Ans:

```
import java.io.*;
class deleteLinkedList {
    /* Link list Node */
    static class Node {
        int data;
        Node next;
    }
    // Utility function to create a new node.
    static Node newNode(int data)
    {
        Node temp = new Node();
        temp.data = data;
        temp.next = null;
        return temp;
    }
    // count of nodes
    static int countOfNodes(Node head)
    {
        int count = 0;
        while (head != null) {
            head = head.next;
            count++;
        }

        return count;

    }

    // Deletes middle node and returns

    // head of the modified list

    static Node deleteMid(Node head)

    {

        // Base cases

        if (head == null)

            return null;

        if (head.next == null) {

            return null;

        }

        Node copyHead = head;


        // Find the count of nodes

        int count = countOfNodes(head);


        // Find the middle node

        int mid = count / 2;


        // Delete the middle node

        while (mid-- > 1) {

            head = head.next;

        }
        // Delete the middle node
        head.next = head.next.next;

        return copyHead;
    }

    // A utility function to print
    // a given linked list
    static void printList(Node ptr)
    {
        while (ptr != null) {
            System.out.print(ptr.data + "->");
            ptr = ptr.next;
        }
        System.out.println("NULL");
    }

    /* Driver code*/
    public static void main(String[] args)
    {
        /* Start with the empty list */
        Node head = newNode(1);
        head.next = newNode(2);
        head.next.next = newNode(3);
        head.next.next.next = newNode(4);
        System.out.println("Given Linked List");
        printList(head);

        head = deleteMid(head);

        System.out.println(
            "Linked List after deletion of middle");
        printList(head);
    }
}
```

2. Given a linked list of N nodes. The task is to check if the linked list has a loop. Linked list can contain self loop..

Ans:

```
import java.util.*;


public class LinkedList {
    static Node head; // head of list
    /* Linked list Node*/
    static class Node {
        int data;
        Node next;
        Node(int d)
        {
            data = d;
            next = null;
        }
}
/* Inserts a new Node at front of the list. */
static public void push(int new_data)
{
        /* 1 & 2: Allocate the Node &
                        Put in the data*/
        Node new_node = new Node(new_data);

        /* 3. Make next of new Node as head */
        new_node.next = head;

        /* 4. Move the head to point to new Node */
        head = new_node;
}
// Returns true if there is a loop in linked
// list else returns false.
static boolean detectLoop(Node h)
{
        HashSet<Node> s = new HashSet<Node>();
        while (h != null) {
                // If we have already has this node
                // in hashmap it means there is a cycle
                // (Because you we encountering the
                // node second time).
                if (s.contains(h))
```

```
            return true;

        // If we are seeing the node for
        // the first time, insert it in hash
        s.add(h);

        h = h.next;
    }

    return false;
}

/* Driver program to test above function */
public static void main(String[] args)
{
    LinkedList llist = new LinkedList();

    llist.push(20);
    llist.push(4);
    llist.push(15);
    llist.push(10);
    /*Create loop for testing */
    llist.head.next.next.next.next = llist.head;
    if (detectLoop(head))
        System.out.println("Loop Found");
    else
```

```
            System.out.println("No Loop");
    }
}
```

3. Given a linked list consisting of L nodes and given a number N. The task is to find the Nth node from the end of the linked list.

Ans:

```
import java.io.*;
class LinkedList {
    Node head; // head of the list

    /* Linked List node */
    class Node {
        int data;
        Node next;
        Node(int d)
        {
            data = d;
            next = null;
        }
    }

    /* Function to get the Nth node from the last of a
    linked list */
}

void printNthFromLast(int N)
{
    int len = 0;
    Node temp = head;

    // 1) count the number of nodes in Linked List
    while (temp != null) {
        temp = temp.next;
        len++;
    }

    // check if value of N is not more than length of
    // the linked list
    if (len < N)
        return;

    temp = head;

    // 2) get the (len-N+1)th node from the beginning
    for (int i = 1; i < len - N + 1; i++)
        temp = temp.next;

    System.out.println(temp.data);
}
/* Inserts a new Node at front of the list. */
```

```
    }

    public void push(int new_data)
    {
        /* 1 & 2: Allocate the Node &
                Put in the data*/
        Node new_node = new Node(new_data);

        /* 3. Make next of new Node as head */
        new_node.next = head;
        /* 4. Move the head to point to new Node */
        head = new_node;
    }
    // Driver's code
    public static void main(String[] args)
    {
        LinkedList llist = new LinkedList();
        llist.push(20);
        llist.push(4);
        llist.push(15);
        llist.push(35);

        // Function call
        llist.printNthFromLast(4);
    }
}
```

4. Given a singly linked list of characters, write a function that returns true if the given list is a palindrome, else false.

Ans:

```
import java.util.*;

class linkedList {

    public static void main(String args[])

    {

        Node one = new Node(1);

        Node two = new Node(2);

        Node three = new Node(3);

        Node four = new Node(4);

        Node five = new Node(3);

        Node six = new Node(2);

        Node seven = new Node(1);

        one.ptr = two;

        two.ptr = three;

        three.ptr = four;

        four.ptr = five;

        five.ptr = six;

        six.ptr = seven;

        boolean condition = isPalindrome(one);

        System.out.println("isPalidrome :" + condition);

    }
    }

static boolean isPalindrome(Node head)
{

    Node slow = head;
    boolean ispalin = true;
    Stack<Integer> stack = new Stack<Integer>();

    while (slow != null) {
        stack.push(slow.data);
        slow = slow.ptr;
    }

    while (head != null) {

        int i = stack.pop();
        if (head.data == i) {
            ispalin = true;
        }
        else {
            ispalin = false;
            break;
        }
        head = head.ptr;
    }
    return ispalin;
    }
}


class Node {
    int data;
    Node ptr;
    Node(int d)
    {
        ptr = null;
        data = d;
    }
}
```

5. Given a linked list of **N** nodes such that it may contain a loop.

A loop here means that the last node of the link list is connected to the node at position X(1-based index). If the link list does not have any loop, X=0.

Remove the loop from the linked list, if it is present, i.e. unlink the last node which is forming the loop.

Ans:

```
class LinkedList {
```

```
static Node head;

static class Node {

	int data;
	Node next;

	Node(int d)
	{
		data = d;
		next = null;
	}
}

// Function that detects loop in the list
int detectAndRemoveLoop(Node node)
{
	Node slow = node, fast = node;
	while (slow != null && fast != null
		&& fast.next != null) {
		slow = slow.next;
		fast = fast.next.next;

		// If slow and fast meet at same point then loop
```

```
            // is present
            if (slow == fast) {
                removeLoop(slow, node);
                return 1;
            }
        }
        return 0;
    }

    // Function to remove loop
    void removeLoop(Node loop, Node head)
    {
        Node ptr1 = loop;
        Node ptr2 = loop;

        // Count the number of nodes in loop
        int k = 1, i;
        Node prevNode = ptr1;
        while (ptr1.next != ptr2) {
            // keeping track beforeing moving next
            prevNode = ptr1;
            ptr1 = ptr1.next;
            k++;
        }
        prevNode.next = null;
```

```
}

// Function to print the linked list
void printList(Node node)
{
        while (node != null) {
                System.out.print(node.data + " ");
                node = node.next;
        }
}

// Driver program to test above functions
public static void main(String[] args)
{
        LinkedList list = new LinkedList();
        list.head = new Node(50);
        list.head.next = new Node(20);
        list.head.next.next = new Node(15);
        list.head.next.next.next = new Node(4);
        list.head.next.next.next.next = new Node(10);

        // Creating a loop for testing
        head.next.next.next.next.next = head.next.next;
        list.detectAndRemoveLoop(head);
        System.out.println(
                              "Linked List after removing loop : ");
                    list.printList(head);
          }
}
```

6. Given a linked list and two integers M and N. Traverse the linked list such that you retain M nodes then delete next N nodes, continue the same till end of the linked list.

Ans:

```
import java.util.*;
class LinkedList
{

// A linked list node
static class Node
{
          int data;
          Node next;
};

/* Function to insert a node at the beginning */
static Node push( Node head_ref, int new_data)
{
          /* allocate node */
          Node new_node = new Node();
```

```
    /* put in the data */
    new_node.data = new_data;

    /* link the old list of the new node */
    new_node.next = (head_ref);

    /* move the head to point to the new node */
    (head_ref) = new_node;

    return head_ref;
}

/* Function to print linked list */
static void printList( Node head)
{
    Node temp = head;
    while (temp != null)
    {
        System.out.printf("%d ", temp.data);
        temp = temp.next;
    }
    System.out.printf("\n");
}
```

```
// Function to skip M nodes and then
// delete N nodes of the linked list.
static void skipMdeleteN( Node head, int M, int N)
{
    Node curr = head, t;
    int count;

    // The main loop that traverses
    // through the whole list
    while (curr!=null)
    {
        // Skip M nodes
        for (count = 1; count < M && curr != null; count++)
            curr = curr.next;

        // If we reached end of list, then return
        if (curr == null)
            return;

        // Start from next node and delete N nodes
        t = curr.next;
        for (count = 1; count <= N && t != null; count++)
        {
            Node temp = t;
            t = t.next;
        }

        // Link the previous list with remaining nodes
        curr.next = t;

        // Set current pointer for next iteration
        curr = t;
    }
}

// Driver code
public static void main(String args[])
{
    /* Create following linked list
    1.2.3.4.5.6.7.8.9.10 */
    Node head = null;
    int M=2, N=3;
    head=push(head, 10);
    head=push(head, 9);
    head=push(head, 8);
    head=push(head, 7);
    head=push(head, 6);
    head=push(head, 5);
    head=push(head, 4);
    head=push(head, 3);
```

```
        head=push(head, 2);

        head=push(head, 1);


        System.out.printf("M = %d, N = %d \nGiven" +

                                                "Linked list is :\n", M, N);

        printList(head);


        skipMdeleteN(head, M, N);


        System.out.printf("\nLinked list after deletion is :\n");

        printList(head);

    }

    }
```

7. Given two linked lists, insert nodes of second list into first list at alternate positions of first list.

For example, if first list is 5->7->17->13->11 and second is 12->10->2->4->6, the first list should become 5->12->7->10->17->2->13->4->11->6 and second list should become empty. The nodes of second list should only be inserted when there are positions available. For example, if the first list is 1->2->3 and second list is 4->5->6->7->8, then first list should become 1->4->2->5->3->6 and second list to 7->8.

Use of extra space is not allowed (Not allowed to create additional nodes), i.e., insertion must be done in-place. Expected time complexity is O(n) where n is number of nodes in first list.

Ans:

```
class LinkedList
{
    Node head; // head of list

    /* Linked list Node*/
    class Node
    {
        int data;
        Node next;
        Node(int d) {data = d; next = null; }
    }

    /* Inserts a new Node at front of the list. */
    void push(int new_data)
    {
        /* 1 & 2: Allocate the Node &
                    Put in the data*/
        Node new_node = new Node(new_data);

        /* 3. Make next of new Node as head */
        new_node.next = head;

        /* 4. Move the head to point to new Node */
        head = new_node;
}

// Main function that inserts nodes of linked list q into p at
// alternate positions. Since head of first list never changes
// and head of second list/ may change, we need single pointer
// for first list and double pointer for second list.
void merge(LinkedList q)
{
        Node p_curr = head, q_curr = q.head;
        Node p_next, q_next;

        // While there are available positions in p;
        while (p_curr != null && q_curr != null) {

                // Save next pointers
                p_next = p_curr.next;
                q_next = q_curr.next;
```

```
                    // make q_curr as next of p_curr
                    q_curr.next = p_next; // change next pointer of q_curr
                    p_curr.next = q_curr; // change next pointer of p_curr

                    // update current pointers for next iteration
                    p_curr = p_next;
                    q_curr = q_next;
            }
            q.head = q_curr;
    }

    /* Function to print linked list */
    void printList()
    {
            Node temp = head;
            while (temp != null)
            {
            System.out.print(temp.data+" ");
            temp = temp.next;
            }
            System.out.println();
    }

    /* Driver program to test above functions */
    public static void main(String args[])
```

```
{
        LinkedList llist1 = new LinkedList();

        LinkedList llist2 = new LinkedList();

        llist1.push(3);

        llist1.push(2);

        llist1.push(1);


        System.out.println("First Linked List:");

        llist1.printList();


        llist2.push(8);

        llist2.push(7);

        llist2.push(6);

        llist2.push(5);

        llist2.push(4);


        System.out.println("Second Linked List:");


        llist1.merge(llist2);


        System.out.println("Modified first linked list:");

        llist1.printList();


        System.out.println("Modified second linked list:");

        llist2.printList();
```

```
    }
}
```

8.Given a singly linked list, find if the linked list is

A linked list is called circular if it is not NULL-terminated and all nodes are connected in the form of a cycle. Below is an example of a circular linked list.

Ans:

```
import java.util.*;


class LinkedList {
    /* Link list Node */
    static class Node {
        int data;
        Node next;
    }
    /*This function returns true if given linked
    list is circular, else false. */
    static boolean isCircular(Node head)
    {
        // An empty linked list is circular
        if (head == null)
            return true;
        // Next of head
        Node node = head.next;

        // This loop would stop in both cases (1) If
        // Circular (2) Not circular
        while (node != null && node != head)
            node = node.next;

        // If loop stopped because of circular
        // condition
        return (node == head);
    }

    // Utility function to create a new node.
    static Node newNode(int data)
    {
        Node temp = new Node();
        temp.data = data;
        temp.next = null;
        return temp;
    }

    /* Driver code*/
    public static void main(String args[])
    {
        /* Start with the empty list */
        Node head = newNode(1);
        head.next = newNode(2);
        head.next.next = newNode(3);
        head.next.next.next = newNode(4);

        System.out.print(isCircular(head) ? "Yes\n": "No\n");

        // Making linked list circular
        head.next.next.next.next = head;

        System.out.print(isCircular(head) ? "Yes\n": "No\n");
    }
}
```