```java
package linkedlist;

// Java program to rotate a linked list

class LinkedList {
    Node head; // head of list

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

    // This function rotates a linked list counter-clockwise
    // and updates the head. The function assumes that k is
    // smaller than size of linked list. It doesn't modify
    // the list if k is greater than or equal to size
    void rotate(int k)
    {
        if (k == 0)
            return;

        // Let us understand the below code for example k = 4
        // and list = 10->20->30->40->50->60.
        Node current = head;

        // current will either point to kth or NULL after this
        // loop. current will point to node 40 in the above example
        int count = 1;
        while (count < k && current != null) {
            current = current.next;
            count++;
        }

        // If current is NULL, k is greater than or equal to count
        // of nodes in linked list. Don't change the list in this case
        if (current == null)
            return;

        // current points to kth node. Store it in a variable.
        // kthNode points to node 40 in the above example
        Node kthNode = current;

        // current will point to last node after this loop
        // current will point to node 60 in the above example
        while (current.next != null)
            current = current.next;

        // Change next of last node to previous head
        // Next of 60 is now changed to node 10

        current.next = head;

        // Change head to (k+1)th node
        // head is now changed to node 50
        head = kthNode.next;

        // change next of kth node to null
        kthNode.next = null;
    }
    /* Given a reference (pointer to pointer) to the head
        of a list and an int, push a new node on the 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;
    }

    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 llist = new LinkedList();

        // create a list 10->20->30->40->50->60
        for (int i = 60; i >= 10; i -= 10)
            llist.push(i);

        System.out.println("Given list");
        llist.printList();

        llist.rotate(4);

        System.out.println("Rotated Linked List");
        llist.printList();
    }
} /* This code is contributed by Rajat Mishra */
```

```java
package linkedlist;

// Java program to reverse a linked list in groups of
// given size
class LinkedList2
{
    Node head; // head of list

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

    Node reverse(Node head, int k)
    {
        Node current = head;
        Node next = null;
        Node prev = null;

        int count = 0;

        /* Reverse first k nodes of linked list */
        while (count < k && current != null)
        {
            next = current.next;
            current.next = prev;
            prev = current;
            current = next;
            count++;
        }

    /* next is now a pointer to (k+1)th node
        Recursively call for the list starting from current.
        And make rest of the list as next of first node */
        if (next != null)
            head.next = reverse(next, k);

        // prev is now head of input list
        return prev;
    }


    /* Utility functions */

    /* 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;
    }

    /* 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[])
    {
        LinkedList2 llist = new LinkedList2();

        /* Constructed Linked List is 1->2->3->4->5->6->
        7->8->8->9->null */
        llist.push(9);
        llist.push(8);
        llist.push(7);
        llist.push(6);
        llist.push(5);
        llist.push(4);
        llist.push(3);
        llist.push(2);
        llist.push(1);

        System.out.println("Given Linked List");
        llist.printList();

        llist.head = llist.reverse(llist.head, 3);

        System.out.println("Reversed list");
        llist.printList();
    }
}
/* This code is contributed by Rajat Mishra */
```

```java
package linkedlist;

// Java program to get intersection point of two linked list

class LinkedList3 {

    static Node head1, head2;

    static class Node {

        int data;
        Node next;

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

    /*function to get the intersection point of two linked
    lists head1 and head2 */
    int getNode()
    {
        int c1 = getCount(head1);
        int c2 = getCount(head2);
        int d;

        if (c1 > c2) {
            d = c1 - c2;
            return _getIntesectionNode(d, head1, head2);
        }
        else {
            d = c2 - c1;
            return _getIntesectionNode(d, head2, head1);
        }
    }

    /* function to get the intersection point of two linked
    lists head1 and head2 where head1 has d more nodes than
    head2 */
    int _getIntesectionNode(int d, Node node1, Node node2)
    {
        int i;
        Node current1 = node1;
        Node current2 = node2;
        for (i = 0; i < d; i++) {
            if (current1 == null) {
                return -1;
            }
            current1 = current1.next;
        }
        while (current1 != null && current2 != null) {
            if (current1.data == current2.data) {
                return current1.data;
            }
            current1 = current1.next;
            current2 = current2.next;
        }

        return -1;
    }

    /*Takes head pointer of the linked list and
    returns the count of nodes in the list */
    int getCount(Node node)
    {
        Node current = node;
        int count = 0;

        while (current != null) {
            count++;
            current = current.next;
        }

        return count;
    }

    public static void main(String[] args)
    {
        LinkedList3 list = new LinkedList3();

        // creating first linked list
        list.head1 = new Node(3);
        list.head1.next = new Node(6);
        list.head1.next.next = new Node(9);
        list.head1.next.next.next = new Node(15);
        list.head1.next.next.next.next = new Node(30);

        // creating second linked list
        list.head2 = new Node(10);
        list.head2.next = new Node(15);
        list.head2.next.next = new Node(30);

        System.out.println("The node of intersection is " + list.getNode());
    }
}

// This code has been contributed by Mayank Jaiswal
```

```java
package linkedlist;

// Java program to detect loop in a linked list
import java.util.*;

  class LinkedList4 {

    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 their 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)
    {
        LinkedList4 llist = new LinkedList4();

        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");
    }
}

// This code is contributed by Arnav Kr. Mandal.
```

```
package linkedlist;

// Java program to detect and remove loop in linked list

class LinkedList32 {

    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 curr)
    {
        Node ptr1 = null, ptr2 = null;

        /* Set a pointer to the beginning of the Linked List
        and move it one by one to find the first node which
        is part of the Linked List */
        ptr1 = curr;
        while (1 == 1) {

            /* Now start a pointer from loop_node and check
            if it ever reaches ptr2 */
            ptr2 = loop;
            while (ptr2.next != loop && ptr2.next != ptr1) {
                ptr2 = ptr2.next;
            }

            /* If ptr2 reahced ptr1 then there is a loop. So
            break the loop */
            if (ptr2.next == ptr1) {
                break;
            }

            /* If ptr2 did't reach ptr1 then try the next
             * node after ptr1 */
            ptr1 = ptr1.next;
```

```java
        }

        /* After the end of loop ptr2 is the last node of
        the loop. So make next of ptr2 as NULL */
        ptr2.next = null;
    }

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

    // Driver code
    public static void main(String[] args)
    {
        LinkedList32 list = new LinkedList32();
        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);
    }
}

// This code has been contributed by Mayank Jaiswal
```

```java
package linkedlist;

// Java program to detect and remove loop in linked list

class LinkedList11 {

    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;
        while (ptr1.next != ptr2) {
            ptr1 = ptr1.next;
            k++;
        }

        // Fix one pointer to head
        ptr1 = head;

        // And the other pointer to k nodes after head
        ptr2 = head;
        for (i = 0; i < k; i++) {
            ptr2 = ptr2.next;
        }

        /* Move both pointers at the same pace,
        they will meet at loop starting node */
        while (ptr2 != ptr1) {
            ptr1 = ptr1.next;
            ptr2 = ptr2.next;
        }
        // Get pointer to the last node
        while (ptr2.next != ptr1) {
            ptr2 = ptr2.next;
        }

        /* Set the next node of the loop ending node
        to fix the loop */
        ptr2.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)
    {
        LinkedList11 list = new LinkedList11();
        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);
    }
}

// This code has been contributed by Mayank Jaiswal
```

```java
package linkedlist;

// Java program to find middle of linked list
class LinkedList44
{
    Node head; // head of linked list

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

    /* Function to print middle of linked list */
    void printMiddle()
    {
        Node slow_ptr = head;
        Node fast_ptr = head;
        if (head != null)
        {
            while (fast_ptr != null && fast_ptr.next != null)
            {
                fast_ptr = fast_ptr.next.next;
                slow_ptr = slow_ptr.next;
            }
            System.out.println("The middle element is [" +
                    slow_ptr.data + "] \n");
        }
    }

    /* 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;
    }

    /* This function prints contents of linked list
    starting from the given node */
    public void printList()
    {
        Node tnode = head;
        while (tnode != null)
        {
            System.out.print(tnode.data+"->");
            tnode = tnode.next;
        }
        System.out.println("NULL");
    }

    public static void main(String [] args)
    {
        LinkedList44 llist = new LinkedList44();
        for (int i=5; i>0; --i)
        {
            llist.push(i);
            llist.printList();
            llist.printMiddle();
        }
    }
}
// This code is contributed by Rajat Mishra
```

```java
package linkedlist;

// Java program for reversing the linked list

class LinkedList71 {

    static Node head;

    static class Node {

        int data;
        Node next;

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

    /* Function to reverse the linked list */
    Node reverse(Node node)
    {
        Node prev = null;
        Node current = node;
        Node next = null;
        while (current != null) {
            next = current.next;
            current.next = prev;
            prev = current;
            current = next;
        }
        node = prev;
        return node;
    }

    // prints content of double linked list
    void printList(Node node)
    {
        while (node != null) {
            System.out.print(node.data + " ");
            node = node.next;
        }
    }

    // Driver Code
    public static void main(String[] args)
    {
        LinkedList71 list = new LinkedList71();
        list.head = new Node(85);
        list.head.next = new Node(15);
        list.head.next.next = new Node(4);
        list.head.next.next.next = new Node(20);

        System.out.println("Given Linked list");
        list.printList(head);
        head = list.reverse(head);
        System.out.println("");
        System.out.println("Reversed linked list ");
        list.printList(head);
    }
}

// This code has been contributed by Mayank Jaiswal
```

```java
package linkedlist;

// Java program for flattening a Linked List
class LinkedList22
{
    Node head; // head of list

    /* Linked list Node*/
    class Node
    {
        int data;
        Node right, down;
        Node(int data)
        {
            this.data = data;
            right = null;
            down = null;
        }
    }

    // An utility function to merge two sorted linked lists
    Node merge(Node a, Node b)
    {
        // if first linked list is empty then second
        // is the answer
        if (a == null)   return b;

        // if second linked list is empty then first
        // is the result
        if (b == null)   return a;

        // compare the data members of the two linked lists
        // and put the larger one in the result
        Node result;

        if (a.data < b.data)
        {
            result = a;
            result.down = merge(a.down, b);
        }

        else
        {
            result = b;
            result.down = merge(a, b.down);
        }

        result.right = null;
        return result;
    }

    Node flatten(Node root)
    {
        // Base Cases
        if (root == null || root.right == null)
            return root;

        // recur for list on right
        root.right = flatten(root.right);

        // now merge
        root = merge(root, root.right);

        // return the root
        // it will be in turn merged with its left
        return root;
    }

    /* Utility function to insert a node at beginning of the
    linked list */
    Node push(Node head_ref, int data)
    {
        /* 1 & 2: Allocate the Node &
                Put in the data*/
        Node new_node = new Node(data);

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

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

        /*5. return to link it back */
        return head_ref;
    }

    void printList()
    {
        Node temp = head;
        while (temp != null)
        {
            System.out.print(temp.data + " ");
            temp = temp.down;
        }
        System.out.println();
    }

    /* Driver program to test above functions */
    public static void main(String args[])
    {
        LinkedList22 L = new LinkedList22();

        /* Let us create the following linked list
            5 -> 10 -> 19 -> 28
            | |   |    |
            V V   V    V
            7 20 22 35
            |        |    |
            V        V    V
            8        50  40
            |             |
            V             V
            30            45
        */

        L.head = L.push(L.head, 30);
        L.head = L.push(L.head, 8);
        L.head = L.push(L.head, 7);
        L.head = L.push(L.head, 5);

        L.head.right = L.push(L.head.right, 20);
        L.head.right = L.push(L.head.right, 10);

        L.head.right.right = L.push(L.head.right.right, 50);
        L.head.right.right = L.push(L.head.right.right, 22);
        L.head.right.right = L.push(L.head.right.right, 19);

        L.head.right.right.right = L.push(L.head.right.right.right, 45);
        L.head.right.right.right = L.push(L.head.right.right.right, 40);
        L.head.right.right.right = L.push(L.head.right.right.right, 35);
        L.head.right.right.right = L.push(L.head.right.right.right, 20);
        // flatten the list
        L.head = L.flatten(L.head);

        L.printList();
    }
} /* This code is contributed by Rajat Mishra */
```

```java
package linkedlist;

// Java program to add two numbers
// represented by linked list

class LinkedList75 {

    static Node head1, head2;

    static class Node {

        int data;
        Node next;

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

    /* Adds contents of two linked
lists and return the head node
of resultant list */
    Node addTwoLists(Node first, Node second)
    {
        // res is head node of the resultant list
        Node res = null;
        Node prev = null;
        Node temp = null;
        int carry = 0, sum;

        // while both lists exist
        while (first != null || second != null) {
            // Calculate value of next
            // digit in resultant list.
            // The next digit is sum
            // of following things
            // (i) Carry
            // (ii) Next digit of first
            // list (if there is a next digit)
            // (ii) Next digit of second
            // list (if there is a next digit)
            sum = carry + (first != null ? first.data : 0)
                    + (second != null ? second.data : 0);

            // update carry for next calulation
            carry = (sum >= 10) ? 1 : 0;

            // update sum if it is greater than 10
            sum = sum % 10;

            // Create a new node with sum as data
            temp = new Node(sum);

            // if this is the first node then set
            // it as head of the resultant list
            if (res == null) {
                res = temp;
            }

            // If this is not the first
            // node then connect it to the rest.
            else {
                prev.next = temp;
            }
            // Set prev for next insertion
            prev = temp;

            // Move first and second pointers
            // to next nodes
            if (first != null) {
                first = first.next;
            }
            if (second != null) {
                second = second.next;
            }
        }

        if (carry > 0) {
            temp.next = new Node(carry);
        }

        // return head of the resultant list
        return res;
    }
    /* Utility function to print a linked list */

    void printList(Node head)
    {
        while (head != null) {
            System.out.print(head.data + " ");
            head = head.next;
        }
        System.out.println("");
    }

    // Driver Code
    public static void main(String[] args)
    {
        LinkedList75 list = new LinkedList75();

        // creating first list
        list.head1 = new Node(7);
        list.head1.next = new Node(5);
        list.head1.next.next = new Node(9);
        list.head1.next.next.next = new Node(4);
        list.head1.next.next.next.next = new Node(6);
        System.out.print("First List is ");
        list.printList(head1);

        // creating seconnd list
        list.head2 = new Node(8);
        list.head2.next = new Node(4);
        System.out.print("Second List is ");
        list.printList(head2);

        // add the two lists and see the result
        Node rs = list.addTwoLists(head1, head2);
        System.out.print("Resultant List is ");
        list.printList(rs);
    }
}

// this code has been contributed by Mayank Jaiswal
```

```java
package linkedlist;

/* Java program to check if linked list is palindrome recursively */
import java.util.*;

class linkeList {
    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;
    }
}
```

```java
package linkedlist;

// Java program for linked-list implementation of queue

// A linked list (LL) node to store a queue entry
class QNode {
    int key;
    QNode next;

    // constructor to create a new linked list node
    public QNode(int key)
    {
        this.key = key;
        this.next = null;
    }
}

// A class to represent a queue
// The queue, front stores the front node of LL and rear stores the
// last node of LL
class Queue {
    QNode front, rear;

    public Queue()
    {
        this.front = this.rear = null;
    }

    // Method to add an key to the queue.
    void enqueue(int key)
    {

        // Create a new LL node
        QNode temp = new QNode(key);

        // If queue is empty, then new node is front and rear both
        if (this.rear == null) {
            this.front = this.rear = temp;
            return;
        }

        // Add the new node at the end of queue and change rear
        this.rear.next = temp;
        this.rear = temp;
    }

    // Method to remove an key from queue.
    void dequeue()
    {
        // If queue is empty, return NULL.
        if (this.front == null)
            return;

        // Store previous front and move front one node ahead
        QNode temp = this.front;
        this.front = this.front.next;

        // If front becomes NULL, then change rear also as NULL
        if (this.front == null)
            this.rear = null;
    }
}

// Driver class
  class Test33 {
    public static void main(String[] args)
    {
        Queue q = new Queue();
        q.enqueue(10);
        q.enqueue(20);
        q.dequeue();
        q.dequeue();
        q.enqueue(30);
        q.enqueue(40);
        q.enqueue(50);
        q.dequeue();
        System.out.println("Queue Front : " + q.front.key);
        System.out.println("Queue Rear : " + q.rear.key);
    }
}
// This code is contributed by Gaurav Miglani
```

```java
package linkedlist;

// Java program to Implement a stack
// using singly linked list
// import package
import static java.lang.System.exit;

// Create Stack Using Linked list
class StackUsingLinkedlist {

    // A linked list node
    private class Node {

        int data; // integer data
        Node link; // reference variable Node type
    }
    // create global top reference variable global
    Node top;
    // Constructor
    StackUsingLinkedlist()
    {
        this.top = null;
    }

    // Utility function to add an element x in the stack
    public void push(int x) // insert at the beginning
    {
        // create new node temp and allocate memory
        Node temp = new Node();

        // check if stack (heap) is full. Then inserting an
        // element would lead to stack overflow
        if (temp == null) {
            System.out.print("\nHeap Overflow");
            return;
        }

        // initialize data into temp data field
        temp.data = x;

        // put top reference into temp link
        temp.link = top;

        // update top reference
        top = temp;
    }

    // Utility function to check if the stack is empty or not
    public boolean isEmpty()
    {
        return top == null;
    }

    // Utility function to return top element in a stack
    public int peek()
    {
        // check for empty stack
        if (!isEmpty()) {
            return top.data;
        }
        else {
            System.out.println("Stack is empty");
            return -1;
        }
    }
    // Utility function to pop top element from the stack
    public void pop() // remove at the beginning
    {
        // check for stack underflow
        if (top == null) {
            System.out.print("\nStack Underflow");
            return;
        }

        // update the top pointer to point to the next node
        top = (top).link;
    }

    public void display()
    {
        // check for stack underflow
        if (top == null) {
            System.out.printf("\nStack Underflow");
            exit(1);
        }
        else {
            Node temp = top;
            while (temp != null) {

                // print node data
                System.out.printf("%d->", temp.data);

                // assign temp link to temp
                temp = temp.link;
            }
        }
    }
}
// main class
  class GFG {
    public static void main(String[] args)
    {
        // create Object of Implementing class
        StackUsingLinkedlist obj = new StackUsingLinkedlist();
        // insert Stack value
        obj.push(11);
        obj.push(22);
        obj.push(33);
        obj.push(44);

        // print Stack elements
        obj.display();

        // print Top element of Stack
        System.out.printf("\nTop element is %d\n", obj.peek());

        // Delete top element of Stack
        obj.pop();
        obj.pop();

        // print Stack elements
        obj.display();

        // print Top element of Stack
        System.out.printf("\nTop element is %d\n", obj.peek());
    }
}
```

```
package linkedlist;

// Java program to sort a linked list of 0, 1 and 2
class LinkedList35
{
    Node head; // head of list

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

    void sortList()
    {
        // initialise count of 0 1 and 2 as 0
        int count[] = {0, 0, 0};

        Node ptr = head;

        /* count total number of '0', '1' and '2'
         * count[0] will store total number of '0's
         * count[1] will store total number of '1's
         * count[2] will store total number of '2's */
        while (ptr != null)
        {
            count[ptr.data]++;
            ptr = ptr.next;
        }

        int i = 0;
        ptr = head;

        /* Let say count[0] = n1, count[1] = n2 and count[2] = n3
         * now start traversing list from head node,
         * 1) fill the list with 0, till n1 > 0
         * 2) fill the list with 1, till n2 > 0
         * 3) fill the list with 2, till n3 > 0 */
        while (ptr != null)
        {
            if (count[i] == 0)
                i++;
            else
            {
                ptr.data= i;
                --count[i];
                ptr = ptr.next;
            }
        }
    }


    /* Utility functions */

    /* 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;
    }

    /* 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[])
    {
        LinkedList35 llist = new LinkedList35();

        /* Constructed Linked List is 1->2->3->4->5->6->7->
        8->8->9->null */
        llist.push(0);
        llist.push(1);
        llist.push(0);
        llist.push(2);
        llist.push(1);
        llist.push(1);
        llist.push(2);
        llist.push(1);
        llist.push(2);

        System.out.println("Linked List before sorting");
        llist.printList();

        llist.sortList();

        System.out.println("Linked List after sorting");
        llist.printList();
    }
}
/* This code is contributed by Rajat Mishra */
```

```java
package linkedlist;

// Simple Java program to find n'th node from end of linked list
class LinkedList67 {
    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 program to test above methods */
    public static void main(String[] args)
    {
        LinkedList67 llist = new LinkedList67();
        llist.push(20);
        llist.push(4);
        llist.push(15);
        llist.push(35);
        llist.printNthFromLast(4);
    }
} // This code is contributed by Rajat Mishra
```