# Linked Lists Implementation in Java

DATE:28/08/2024

REG NO:2022503045

1. Write a Java program to implement the below listed task

CODE :

import java.util.Scanner;

import java.util.HashMap;

import java.util.HashSet;

class Node{

    int data;

    Node next;

    Node(int data){

        this.data=data;

        this.next=null;

    }

}

public class LinkedList\_3045 {

    Node head;

    public LinkedList\_3045(){

        head=null;

    }

    public void insertAtBeginning(int data) {

        Node newNode=new Node(data);

        if(head==null){

            head=newNode;

        }

        else{

            Node current=head;

            newNode.next=current;

            head=newNode;

        }

    }

   public void insertAtEnd(int data) {

        Node newNode=new Node(data);

        if(head==null){

            head=newNode;

        }

        else{

            Node current=head;

            while(current.next!=null){

                current=current.next;

            }

            current.next=newNode;

        }

   }

    public void insertAtPosition(int data, int position) {

        if(position==1)

            insertAtBeginning(data);

        else{

            Node newNode=new Node(data);

            int currentPos=1;

            Node current=head;

            Node prev=null;

            while(currentPos!=position && current.next!=null ){

                prev=current;

                current=current.next;

                currentPos+=1;

            }

            if(currentPos==position) {

                prev.next = newNode;

                newNode.next = current;

            }

            else if(current.next==null && currentPos+1==position){

                current.next=newNode;

            }

            else{

                System.out.println("Position is exceeding the length of linkedlist");

            }

        }

    }

    public void insertAfterNode(int node,int data){

         Node current=head;

         if(head==null){

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

         }

         boolean flag=false;

         while(current.next!=null && !flag){

             if(current.data==node){

                 flag=true;

             }

             else{

                 current=current.next;

             }

         }

         if(flag){

             Node newNode=new Node(data);

             newNode.next=current.next;

             current.next=newNode;

             System.out.println("inserted after the node "+node);

         }

         else{

             System.out.println("the node mentioned is not there in list");

         }

    }

    public void insertBeforeNode(int node,int data){

        if(head!=null){

            Node prev=null;

            Node current=head;

            boolean flag=false;

            while(current!=null && !flag){

                if(current.data==node){

                    flag=true;

                }

                else{

                    prev=current;

                    current=current.next;

                }

            }

            if(flag){

                Node newNode=new Node(data);

                if(prev==null){

                    newNode.next=head;

                    head=newNode;

                }

                else{

                    prev.next=newNode;

                    newNode.next=current;

                }

                System.out.println("inserted before the node");

            }

            else{

                System.out.println("the node is not available");

            }

        }

    }

    public void deleteFromBeginning(){

        if(head==null){

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

        }

        else{

            System.out.println("deleted the node "+head.data);

            head=head.next;

        }

    }

    public void deleteFromEnd(){

        if(head==null){

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

        }

        else{

            Node current=head;

            Node prev=null;

            while(current.next!=null){

                prev=current;

                current=current.next;

            }

            System.out.println("Deleted the node + " +current.data);

            if(prev==null){

                head=null;

            }

            else{

                prev.next=null;

            }

        }

    }

    public void deleteByValue(int value){

        if(head==null){

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

        }

        else{

            Node prev=null;

            Node current=head;

            boolean flag=false;

            while(current!=null && !flag){

                if(current.data==value){

                    flag=true;

                }

                else{

                    prev=current;

                    current=current.next;

                }

            }

            if(flag){

                if(prev==null){

                    head=head.next;

                }

                else{

                    prev.next=current.next;

                }

                System.out.println("Deleted the node + " +current.data);

            }

            else{

                System.out.println("The node is not available");

            }

        }

    }

    public void deleteByPosition(int position){

        if(head==null){

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

        }

        else{

            Node prev=null;

            Node current=head;

            int currentPos=1;

            if(currentPos==position){

                head=head.next;

            }

            else{

                while(current!=null && currentPos!=position){

                    prev=current;

                    current=current.next;

                    currentPos+=1;

                }

                if(currentPos==position){

                    prev.next=current.next;

                    System.out.println("Deleted the node + " +current.data);

                }

                else{

                    System.out.println("No node is not available");

                }

            }

        }

    }

    public int searchByValue(int value){

        if(head==null){

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

        }

        else{

            Node current=head;

            int position=1;

            while(current!=null){

                if(current.data==value)

                    return position;

                current=current.next;

            }

        }

        return -1;

    }

    public void searchByPosition(int position){

        if(head==null){

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

        }

        else{

            int currentPos=1;

            Node current=head;

            boolean flag=false;

            while(current!=null){

                if(currentPos==position){

                    System.out.println("element at position "+position+" is "+current.data);

                    flag=true;

                    break;

                }

                current=current.next;

                currentPos+=1;

            }

            if(!flag){

                System.out.println("position given is not available");

            }

        }

    }

    public void reverse(){

        if(head==null){

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

        }

        else{

            Node curr=head,prev=null,next;

            while(curr!=null){

                next=curr.next;

                curr.next=prev;

                prev=curr;

                curr=next;

            }

            head=prev;

        }

    }

    public int length(){

        int count=0;

        Node current=head;

        while(current!=null){

             count+=1;

             current=current.next;

        }

        return count;

    }

    public void frequencyOfElements(){

        HashMap<Integer,Integer> freq=new HashMap<>();

        if(head==null){

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

        }

        else{

            Node current=head;

            while(current!=null){

                freq.put(current.data,freq.getOrDefault(current.data,0)+1);

                current=current.next;

            }

            System.out.println("Frequency of elements:");

            for(int k:freq.keySet()){

                System.out.println(k+" -- > " +freq.get(k));

            }

        }

    }

    public void concatenate(Node head2){

        if(head==null){

            head=head2;

        }

        else{

            Node current=head;

            while(current.next!=null){

                current=current.next;

            }

            current.next=head2;

        }

    }

    public void sort(){

        if(head==null){

            System.out.println("Empty list !");

        }

        else{

            Node current;

            boolean swapped;

            do{

               swapped=false;

               current=head;

               while(current.next!=null){

                if(current.data>current.next.data){

                    int temp=current.data;

                    current.data=current.next.data;

                    current.next.data=temp;

                    swapped=true;

                }

                current=current.next;

               }

            }while(swapped);

        }

    }

    public void removeDuplicates(){

        if(head==null){

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

        }

        else{

            Node current=head;

            while(current!=null){

                Node current2=current.next;

                Node prev=current;

                while(current2!=null){

                    if(current.data==current2.data){

                        prev.next=current2.next;

                        current2=current2.next;

                    }

                    else{

                        prev=current2;

                        current2=current2.next;

                    }

                }

                current=current.next;

            }

        }

    }

    public void display(){

        if(head==null){

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

        }

        else{

            Node current=head;

            while(current!=null){

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

                current=current.next;

            }

        }

    }

    public static void main(String[] args) {

        System.out.println("\nSivaranjani - 2022503045 ");

        LinkedList\_3045 list = new LinkedList\_3045();

        list.insertAtEnd(10);

        list.insertAtEnd(20);

        list.insertAtEnd(30);

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

        list.display();

        System.out.println();

        Scanner sc = new Scanner(System.in);

        System.out.println("\n1.Insert\n 2.Delete\n 3.Search\n 4.display\n 5.reverse\n6.length\n7.count of each elements\n8.concatenate another list \n9.sort the list\n10.Remove duplicates\n11.exit");

        System.out.println("Enter the option :");

        int opt = sc.nextInt();

        int data;

        int pos;

        while (opt != 11) {

            switch(opt) {

                case 1: {

                    System.out.println("1.Insert At the end\n 2.Insert at the beginning\n 3.Insert at the position\n 4.Insert after a node\n5.Insert before a node\n");

                    System.out.println("Enter the option for insertion :");

                    opt = sc.nextInt();

                    switch (opt) {

                        case 1:

                            System.out.println("enter the data:");

                            data = sc.nextInt();

                            list.insertAtEnd(data);

                            break;

                        case 2:

                            System.out.println("enter the data:");

                            data = sc.nextInt();

                            list.insertAtBeginning(data);

                            break;

                        case 3:

                            System.out.println("enter the data:");

                            data = sc.nextInt();

                            System.out.println("enter the position(1-based):");

                            pos = sc.nextInt();

                            list.insertAtPosition(data, pos);

                            break;

                        case 4:

                            System.out.println("enter the data:");

                            data = sc.nextInt();

                            System.out.println("enter the node to insert after:");

                            pos = sc.nextInt();

                            list.insertAfterNode(pos, data);

                            break;

                        case 5:

                            System.out.println("enter the data:");

                            data = sc.nextInt();

                            System.out.println("enter the node to insert before:");

                            pos = sc.nextInt();

                            list.insertBeforeNode(pos, data);

                            break;

                    }

                    break;

                }

                case 2: {

                    System.out.println("1.Delete At the end\n 2 Delete  from the beginning\n 3.Delete By position\n 4.Delete By Value\n");

                    System.out.println("Enter the option for deletion :");

                    opt = sc.nextInt();

                    switch (opt) {

                        case 1:

                            list.deleteFromEnd();

                            break;

                        case 2:

                            list.deleteFromBeginning();

                            break;

                        case 3:

                            System.out.println("enter the position:");

                            pos = sc.nextInt();

                            list.deleteByPosition(pos);

                            break;

                        case 4:

                            System.out.println("enter the data:");

                            data = sc.nextInt();

                            list.deleteByValue(data);

                            break;

                    }

                    break;

                }

                case 3: {

                    System.out.println("1.Search By Value\n 2 Search By positon\n");

                    System.out.println("Enter the option for searching :");

                    opt = sc.nextInt();

                    switch (opt) {

                        case 1:

                            System.out.println("enter the data:");

                            data = sc.nextInt();

                            int index=list.searchByValue(data);

                            if(index==-1)

                                System.out.println("element not present");

                            else

                                System.out.println("element present at position "+index );

                            break;

                        case 2:

                            System.out.println("enter the position:");

                            pos = sc.nextInt();

                            list.searchByPosition(pos);

                            break;

                    }

                    break;

                }

                case 4:

                      list.display();

                      break;

                case 5:

                     list.reverse();

                     break;

                case 6:

                     System.out.println("The length of the list is : "+list.length());

                     break;

                case 7:

                    list.frequencyOfElements();

                    break;

                case 8:{

                    System.out.println("Enter the elements for linkedlist( enter N to stop adding ):");

                    String reply=sc.next();

                    LinkedList\_3045 list2=new LinkedList\_3045();

                    while(!reply.equals("N")){

                        data=Integer.parseInt(reply);

                        list2.insertAtEnd(data);

                        reply=sc.next();

                    }

                    list.concatenate(list2.head);

                    break;

                   }

                case 9:

                    list.sort();

                    break;

                case 10:

                    list.removeDuplicates();

                    break;

                }

                System.out.println("\n List after modification:");

            list.display();

            System.out.println("\n1.Insert\n 2.Delete\n 3.Search\n 4.display\n 5.reverse\n6.length\n7.count of each elements\n8.concatenate another list\n9.sort the list\n10.remove duplicates\n11.exit");

            System.out.println("\nEnter the option :");

            opt = sc.nextInt();

            }

            System.out.println("exited..");

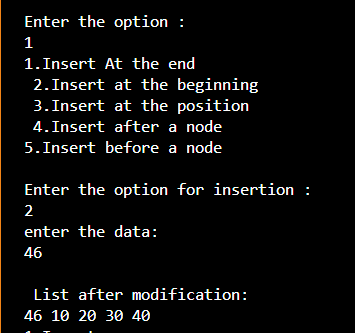
        }

}

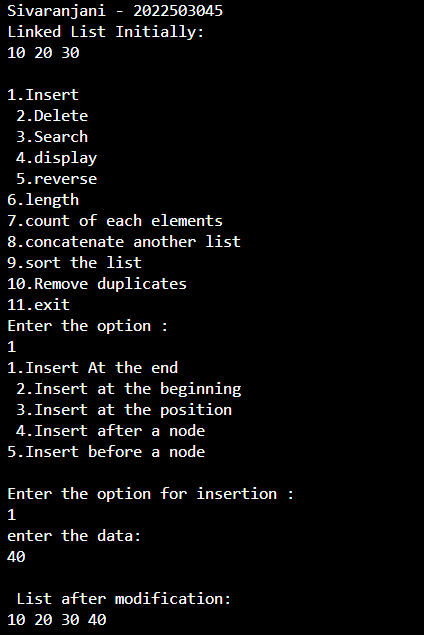
OUTPUT:

**1.Insertion:**

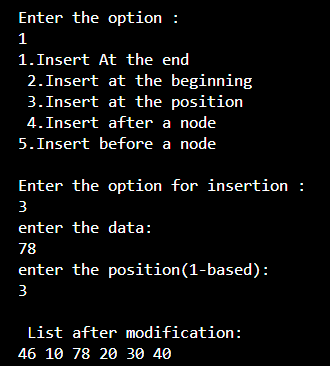
Insert at the beginning.



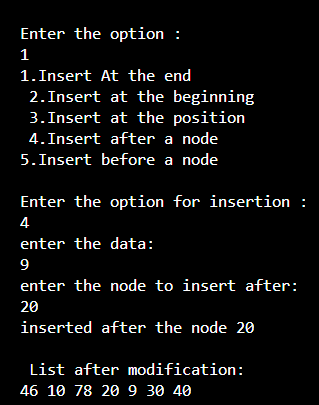
Insert at the end.



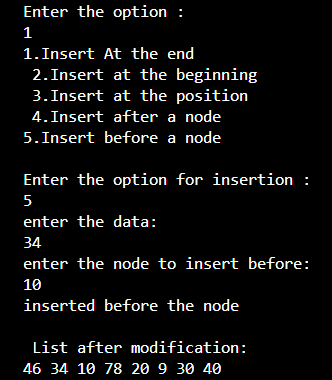
Insert at a specific position.



Insert after a specific node.

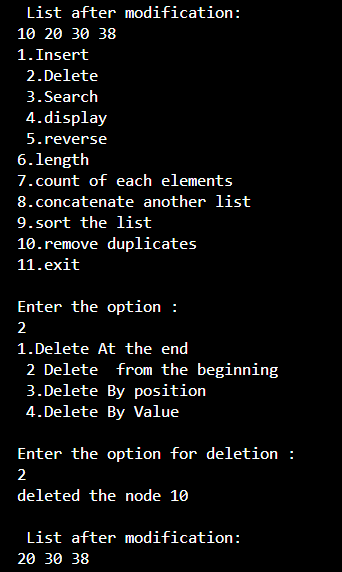


Insert before a specific node.

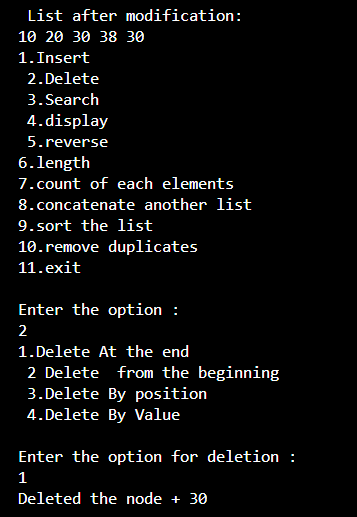


**2.Deletion:**

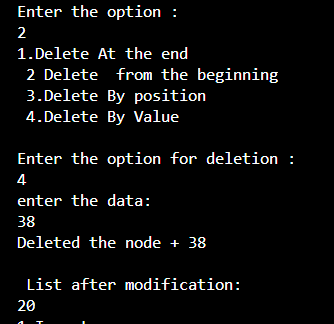
Delete from the beginning.



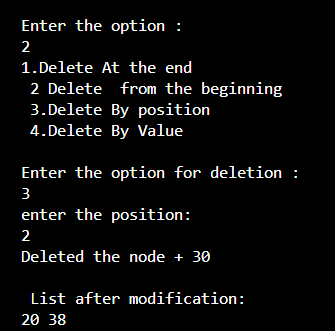
Delete from the end.



Delete a specific element by value.

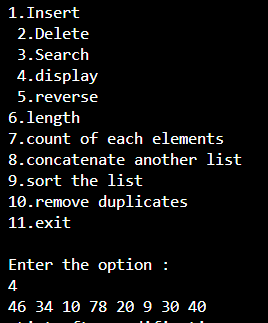


Delete a specific element by position.

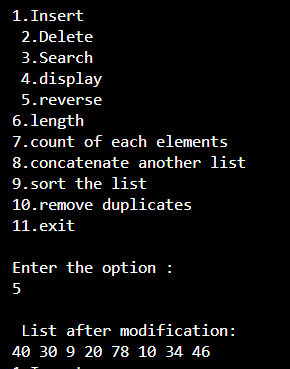


**Traversal and Display:**

Traverse and print the elements in the linked list.

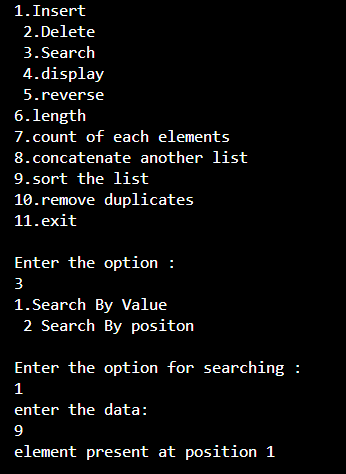


Reverse and print the elements in the linked list.

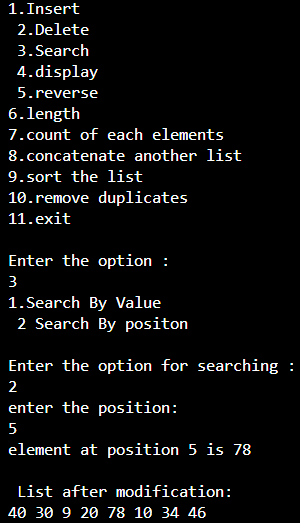


**Search and Access:**

Search for an element by value.

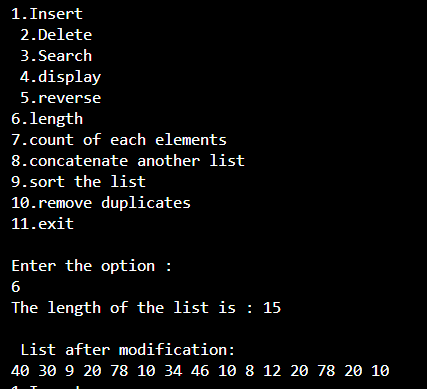


Access an element by position.

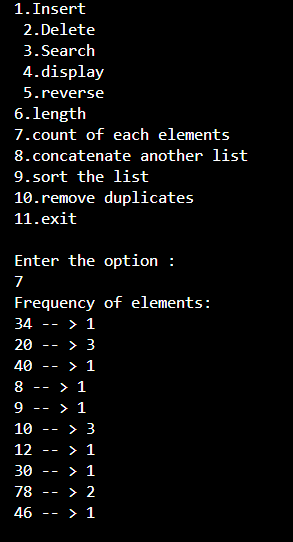


**Length and Counting:**

Find the length (number of nodes) of the linked list.

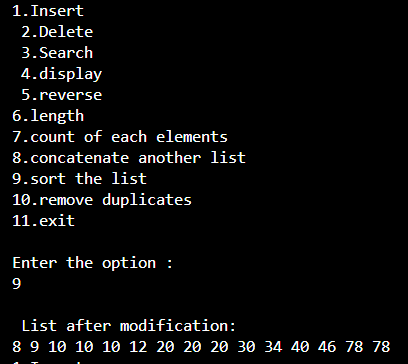


Count the occurrences of a specific value in the list.



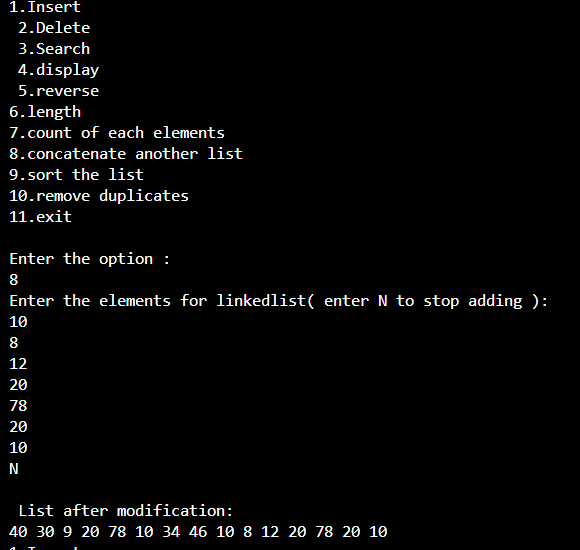
**Sorting :**

Sort the linked list (best sort).



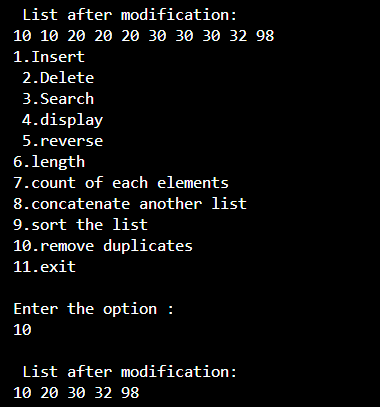
**Concatenation:**

Concatenate (combine) two linked lists together.



**Duplicate Removal:**

Remove duplicate elements from a linked list.



1. Implement Stack using LinkedList:

Code:

import java.util.Scanner;

class Node{

    int data;

    Node next;

    Node(int data){

        this.data=data;

        this.next=null;

    }

}

public class Stack\_3045 {

    Node top;

    public Stack\_3045(){

        top=null;

    }

    public boolean isEmpty(){

        if(top==null)

            return true;

        return false;

    }

    public void push(int data,Stack\_3045 stack){

        Node newNode=new Node(data);

        if(stack.isEmpty()){

            top=newNode;

        }

        else{

            Node current=top;

            newNode.next=current;

            top=newNode;

        }

    }

    public void pop(Stack\_3045 stack){

        if(stack.isEmpty()){

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

        }

        else{

            System.out.println("popped  the element "+top.data);

            top=top.next;

        }

    }

    public void display(Stack\_3045 stack){

        if(stack.isEmpty()){

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

        }

        else{

            Node current=top;

            while(current!=null){

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

                current=current.next;

            }

        }

    }

    public void peek(Stack\_3045 stack){

        if(!stack.isEmpty()){

            System.out.println(top.data+" is the peek element");

        }

        else{

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

        }

    }

    public static void main(String[] args){

        Stack\_3045 stack =new Stack\_3045();

        System.out.println("\nSivaranjani - 2022503045 ");

        System.out.println("1.push\n2.pop\n3.display\n4.peek element\n5exit");

        Scanner sc=new Scanner(System.in);

        System.out.println("enter the option:");

        int opt=sc.nextInt();

        int data;

        while(opt!=5){

            switch(opt){

                case 1:{

                    System.out.println("enter the data:");

                    data=sc.nextInt();

                    stack.push(data,stack);

                    System.out.println(data+" is pushed");

                    break;

                }

                case 2:{

                   stack.pop(stack);

                   break;

                }

                case 3:{

                    stack.display(stack);

                    break;

                }

                case 4:

                    stack.peek(stack);

                    break;

            }

            System.out.println("\n1.push\n2.pop\n3.display\n4.peek element\n5.exit");

            System.out.println("\nenter the option:");

            opt=sc.nextInt();

        }

        System.out.println("exited..");

        sc.close();

    }

}

OUTPUT:

