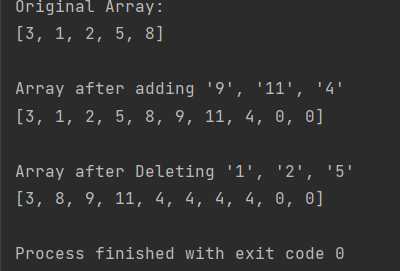
**22K-5159 Amna Mansoor LAB-4 BSE-3B**

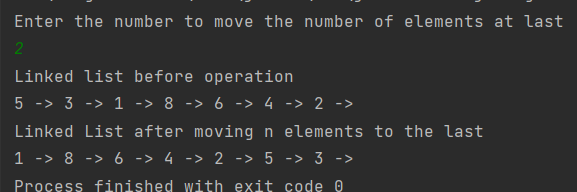
Task1:

import java.util.Arrays;  
  
public class Task1 {  
 public static void main(String[] args) {  
 Array arr=new Array(5);  
 arr.Add(3);  
 arr.Add(1);  
 arr.Add(2);  
 arr.Add(5);  
 arr.Add(8);  
 System.*out*.println("Original Array: ");  
 System.*out*.println(Arrays.*toString*(arr.array));  
  
 arr.Add(9);  
 arr.Add(11);  
 arr.Add(4);  
 System.*out*.println("\nArray after adding '9', '11', '4' ");  
 System.*out*.println(Arrays.*toString*(arr.array));  
  
 arr.Delete(1);  
 arr.Delete(2);  
 arr.Delete(5);  
 System.*out*.println("\nArray after Deleting '1', '2', '5'");  
 System.*out*.println(Arrays.*toString*(arr.array));  
 }  
  
  
  
}  
class Array {  
  
 int[] array;  
 int length;  
  
 Array(int size) {  
 array = new int[size];  
 length = 0;  
 }  
  
 public void Add(int data) {  
 if (length >= array.length) {  
 int[] temp = new int[array.length \* 2];  
 for (int i = 0; i < array.length; i++) {  
 temp[i] = array[i];  
 }  
 array = temp;  
 }  
 array[length] = data;  
 length++;  
 }  
  
 public void Delete(int data) {  
 for (int i = 0; i < length; i++) {  
 if (array[i] == data) {  
 for (int j = i; j < length - 1; j++) {  
 array[j] = array[j + 1];  
 }  
 length--;  
 break;  
 }  
  
 }  
  
 }  
}



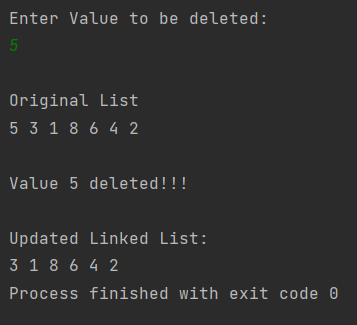
Task 2:

import java.util.Scanner;  
  
public class Task2 {  
 public static void main(String[] args) {  
 LinkedList ll=new LinkedList();  
 Scanner sc=new Scanner(System.*in*);  
 ll.Add(5);  
 ll.Add(3);  
 ll.Add(1);  
 ll.Add(8);  
 ll.Add(6);  
 ll.Add(4);  
 ll.Add(2);  
  
 System.*out*.println("Enter the number to move the number of elements at last");  
 int n= sc.nextInt();  
 System.*out*.println("Linked list before operation ");  
 ll.Display();  
 ll.moveElement(n);  
 System.*out*.println("\nLinked List after moving n elements to the last");  
 ll.Display();  
 }  
}  
  
class LinkedList{  
 Node head;  
 class Node{  
 int data;  
 Node next;  
  
 public Node(int data) {  
 this.data = data;  
 }  
 }  
 public void Add(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 moveElement(int n){  
 if (n <= 0 || head == null || head.next == null) {  
 return;  
 }  
 Node current=head;  
 Node newHead=null;  
 Node NewTail=null;  
 int count = 1;  
  
 while (current != null && count < n) {  
 current = current.next;  
 count++;  
 }  
  
 if (current==null){  
 return;  
 }  
 newHead =current.next;  
 current.next=null;  
  
 Node temp=newHead;  
 while (temp.next != null) {  
 temp = temp.next;  
 }  
  
 temp.next = head;  
 head = newHead;  
 }  
 public void Display(){  
 Node current =head;  
 while (current!=null){  
 System.*out*.print(current.data+" -> ");  
 current=current.next;  
 }  
 }  
  
}

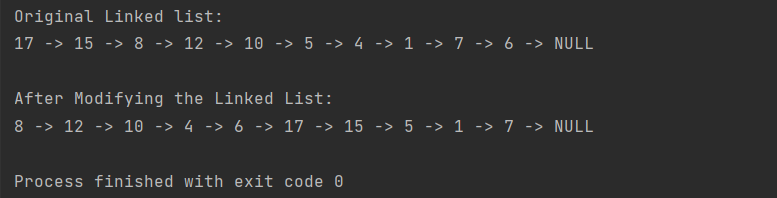


*Task 3:*

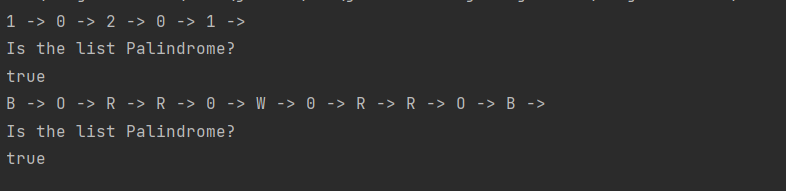
import java.util.Scanner;  
  
public class Task3 {  
 public static void main(String[] args) {  
 Linked\_List ll=new Linked\_List();  
 Scanner sc=new Scanner(System.*in*);  
 ll.Add(5);  
 ll.Add(3);  
 ll.Add(1);  
 ll.Add(8);  
 ll.Add(6);  
 ll.Add(4);  
 ll.Add(2);  
  
 System.*out*.println("Enter Value to be deleted: ");  
 int value=sc.nextInt();  
 System.*out*.println("\nOriginal List ");  
 ll.Display();  
  
 boolean deleted=ll.Delete(value);  
  
 if (deleted){  
 System.*out*.println("\n\nValue "+value+" deleted!!!");  
 }  
 else{  
 System.*out*.println("\n\nValue "+value+" not present!!!");  
 }  
 System.*out*.println("\nUpdated Linked List: ");  
 ll.Display();  
 }  
}  
class Linked\_List{  
 Node head;  
 class Node{  
 int data;  
 Node next;  
  
 public Node(int data) {  
 this.data = data;  
 }  
 }  
 public void Add(int data){  
 Node newNode=new Node(data);  
 if (head==null){  
 head=newNode;  
 }  
 else {  
 Linked\_List.Node current=head;  
 while (current.next!=null){  
 current=current.next;  
 }  
 current.next=newNode;  
 }  
 }  
 public void Display(){  
 Linked\_List.Node current =head;  
 while (current!=null){  
 System.*out*.print(current.data+" ");  
 current=current.next;  
 }  
 }  
  
 public boolean Delete(int value){  
 if (head==null){  
 return false;  
 }  
 if (head.data==value){  
 head=head.next;  
 return true;  
 }  
 Node current=head;  
 while (current.next!=null && current.next.data!=value){  
 current=current.next;  
 }  
 if (current.next!=null){  
 current.next=current.next.next;  
 return true;  
 }  
 return false;  
 }  
  
}



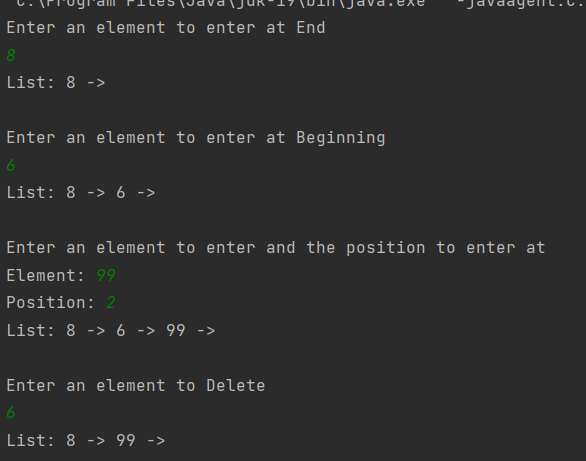
Task 4:  
import java.util.Scanner;  
  
public class Task4 {  
 public static void main(String[] args) {  
 \_LinkedList ll=new \_LinkedList();  
 ll.Add(17);  
 ll.Add(15);  
 ll.Add(8);  
 ll.Add(12);  
 ll.Add(10);  
 ll.Add(5);  
 ll.Add(4);  
 ll.Add(1);  
 ll.Add(7);  
 ll.Add(6);  
 System.*out*.println("Original Linked list:");  
 ll.Display();  
 ll.Modify();  
 System.*out*.println("\nAfter Modifying the Linked List: ");  
 ll.Display();  
 }  
}  
class \_LinkedList{  
 Node head;  
 Node tail;  
 class Node{  
 int data;  
 Node next;  
  
 public Node(int data) {  
 this.data = data;  
 }  
 }  
 public void Add(int data){  
 Node newNode=new Node(data);  
 if (head==null){  
 head=newNode;  
 tail=newNode;  
 }  
 else {  
 tail.next=newNode;  
 tail=newNode;  
 }  
 }  
 public void Display(){  
 Node current=head;  
 while (current!=null){  
 System.*out*.print(current.data+" -> ");  
 current=current.next;  
 }  
 System.*out*.println("NULL");  
 }  
  
 public void Modify(){  
 if (head==null || head.next==null){  
 return;  
 }  
  
 Node evenHead=null;  
 Node oddHead=null;  
 Node evenTail=null;  
 Node oddTail=null;  
  
 Node current=head;  
 while (current!=null){  
 if(current.data%2==0){  
 if (evenHead==null){  
 evenHead=current;  
 evenTail=current;  
 }  
 else{  
 evenTail.next=current;  
 evenTail=current;  
 }  
 } else{  
 if (oddHead==null){  
 oddHead=current;  
 oddTail=current;  
 }  
 else{  
 oddTail.next=current;  
 oddTail=current;  
 }  
 }  
 current=current.next;  
 }  
 if (oddTail != null) {  
 oddTail.next = null;  
 }  
 if (evenTail!=null){  
 evenTail.next=oddHead;  
 }  
 if (evenHead != null) {  
 head = evenHead;  
 tail = evenTail;  
 } else {  
 head = oddHead;  
 tail = oddTail;  
 }  
 }  
  
}



Task 5:  
public class Task5 {  
 public static void main(String[] args) {  
 Palindrome list1 = new Palindrome();  
 list1.Add('1');  
 list1.Add('0');  
 list1.Add('2');  
 list1.Add('0');  
 list1.Add('1');  
 list1.display();  
 System.*out*.println("\nIs the list Palindrome?\n" + list1.isPalindrome());  
  
 Palindrome list2 = new Palindrome();  
 list2.Add('B');  
 list2.Add('O');  
 list2.Add('R');  
 list2.Add('R');  
 list2.Add('0');  
 list2.Add('W');  
 list2.Add('0');  
 list2.Add('R');  
 list2.Add('R');  
 list2.Add('O');  
 list2.Add('B');  
 list2.display();  
 System.*out*.println("\nIs the list Palindrome?\n" + list2.isPalindrome());  
 }  
}  
  
  
class Palindrome{  
  
 CharNode head;  
  
 class CharNode{  
 char data;  
 CharNode next;  
  
 public CharNode(char data) {  
 this.data = data;  
 }  
 }  
 public void Add(char data){  
 if (head==null){  
 head=new CharNode(data);  
 }  
 else {  
 CharNode newNode = new CharNode(data);  
 CharNode currentNode = head;  
 while (currentNode.next != null) {  
 currentNode = currentNode.next;  
 }  
 currentNode.next = newNode;  
 }  
 }  
  
  
 public boolean isPalindrome() {  
 if (head == null) {  
 return true;  
 }  
  
 CharNode slow = head;  
 CharNode fast = head;  
 CharNode prevSlow = head;  
  
 while (fast != null && fast.next != null) {  
 fast = fast.next.next;  
 prevSlow = slow;  
 slow = slow.next;  
 }  
 prevSlow.next = null;  
 CharNode reversedSecondHalf = reverse(slow);  
  
 CharNode current1 = head;  
 CharNode current2 = reversedSecondHalf;  
 while (current1 != null && current2 != null) {  
 if (current1.data != current2.data) {  
 return false;  
 }  
 current1 = current1.next;  
 current2 = current2.next;  
 }  
  
 return true;  
 }  
 public CharNode reverse(CharNode startNode) {  
 CharNode prev = null;  
 CharNode current = startNode;  
 while (current != null) {  
 CharNode next = current.next;  
 current.next = prev;  
 prev = current;  
 current = next;  
 }  
 return prev;  
 }  
 public void display(){  
 CharNode current =head;  
 while (current!=null){  
 System.*out*.print(current.data+" -> ");  
 current=current.next;  
 }  
  
 }  
}

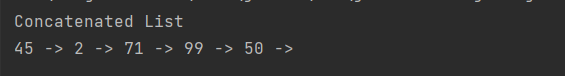


Task 6:  
import java.util.Scanner;  
  
public class Task6 {  
 public static void main(String[] args) {  
 Circular cll=new Circular();  
  
 Scanner sc=new Scanner(System.*in*);  
 System.*out*.println("Enter an element to enter at End");  
 int end= sc.nextInt();  
 cll.insertEnd(end);  
 cll.Display();  
  
 System.*out*.println("\nEnter an element to enter at Beginning");  
 int start= sc.nextInt();  
 cll.insertEnd(start);  
 cll.Display();  
  
 System.*out*.println("\nEnter an element to enter and the position to enter at");  
 System.*out*.print("Element: ");  
 int elem=sc.nextInt();  
 System.*out*.print("Position: ");  
 int pos= sc.nextInt();  
 cll.insertPosition(elem,pos);  
 cll.Display();  
  
 System.*out*.println("\nEnter an element to Delete");  
 int del= sc.nextInt();  
 cll.delete(del);  
 cll.Display();  
 }  
}  
class Circular{  
 class Node{  
 int data;  
 Node next;  
  
 Node (int data){  
 this.data=data;  
 this.next=null;  
 }  
  
 }  
 Node head;  
 int size;  
  
// public Circular(Node head, int size) {  
// head = null;  
// size = 0;  
// }  
 public boolean isEmpty() {  
 return head == null;  
 }  
 public void insertEnd(int data){  
 Node newNode=new Node(data);  
 if (isEmpty()){  
 head=newNode;  
 head.next=head;  
 }  
 else{  
 Node current=head;  
 while(current.next!=head){  
 current=current.next;  
 }  
 current.next=newNode;  
 newNode.next=head;  
 }  
 size++;  
 }  
 public void insertBeginning(int data){  
 Node newNode=new Node(data);  
 if (isEmpty()){  
 head=newNode;  
 head.next=head;  
 }  
 else{  
 Node current=head;  
 while(current.next!=head){  
 current=current.next;  
 }  
 current.next=newNode;  
 newNode.next=head;  
 head=newNode;  
 }  
 size++;  
 }  
 public void insertPosition(int data, int position){  
 if (position<0 || position >size){  
 System.*out*.println("Invalid position");  
 return;  
 }  
 if (position==0){  
 insertBeginning(data);  
 } else if (position==size) {  
 insertEnd(data);  
 }else{  
 Node newNode=new Node(data);  
 Node current=head;  
 for (int i=0; i<position-1; i++) {  
 current = current.next;  
 }  
 newNode.next=current.next;  
 current.next=newNode;  
 size++;  
 }  
 }  
 public void delete(int data) {  
 if (isEmpty()) {  
 System.*out*.println("List is empty");  
 return;  
 }  
 if (head.data==data) {  
 if (size==1) {  
 head=null;  
 } else {  
 Node current = head;  
 while (current.next != head) {  
 current = current.next;  
 }  
 current.next = head.next;  
 head = head.next;  
 }  
 size--;  
 return;  
 }  
 Node current= head;  
 Node prev= null;  
 do {  
 prev= current;  
 current= current.next;  
 } while (current!=head && current.data != data);  
  
 if (current!=head) {  
 prev.next=current.next;  
 size--;  
 } else {  
 System.*out*.println("Element not found");  
 }  
 }  
 public void Display(){  
 if (isEmpty()){  
 System.*out*.println("List is empty");  
 return;  
 }  
 Node current = head;  
 System.*out*.print("List: ");  
 do{  
 System.*out*.print(current.data+" -> ");  
 current= current.next;  
 } while(current != head);  
 System.*out*.println();  
 }  
  
}



Task 7:

public class Task7 {  
 public static void main(String[] args) {  
 Doubly list=new Doubly();  
 Node L=new Node(45);  
 L.next=new Node(2);  
 L.next.prev=L;  
 L.next.next=new Node(71);  
 L.next.next.prev=L.next;  
 Node M=new Node(99);  
 M.next=new Node(50);  
 M.next.prev=M;  
  
 Node concatenate=list.concatenate(L,M);  
 System.*out*.println("Concatenated List");  
 list.display(concatenate);  
 }  
}  
class Node{  
 int data;  
 Node next;  
 Node prev;  
  
 public Node(int data) {  
 this.data = data;  
 }  
}  
class Doubly{  
 public Node concatenate(Node L, Node M){  
 if (L==null){  
 return M;  
 } else if (M==null) {  
 return L;  
 }  
 Node tail\_L=L;  
 while (tail\_L.next!=null){  
 tail\_L=tail\_L.next;  
 }  
 tail\_L.next=M;  
 M.prev=tail\_L;  
 return L;  
 }  
 public void display(Node head) {  
 Node current = head;  
 while (current != null) {  
 System.*out*.print(current.data + " -> ");  
 current = current.next;  
 }  
 System.*out*.println();  
 }  
}



Task 8:

public class Task8 {  
 public static void main(String[] args) {  
 singlyLinked sl=new singlyLinked();  
 sl.Add(67);  
 sl.Add(24);  
 sl.Add(31);  
 sl.Add(8);  
 sl.Add(199);  
 sl.Add(206);  
 sl.Add(51);  
 sl.Add(74);  
 sl.Add(777);  
 sl.Add(82);  
 sl.Add(151);  
 System.*out*.println("List: ");  
 sl.display();  
 singlyLinked alternate=sl.getAlternates();  
 System.*out*.println("Alternative Nodes: ");  
 alternate.display();  
 alternate.reverse();  
 System.*out*.println("List After Separating Alternative Nodes:");  
 sl.display();  
 sl.append(alternate);  
 System.*out*.println("Appending the Alternatives to Original List:");  
 sl.display();  
 }  
}  
class singlyLinked {  
 Node head;  
 Node next;  
 class Node {  
 int data;  
 Node next;  
  
 public Node(int data) {  
 this.data = data;  
 }  
 }  
 public void Add(int data) {  
 if (head == null) {  
 head = new Node(data);  
 }  
 else{  
 Node newNode=new Node(data);  
 Node current=head;  
 while (current.next!=null){  
 current=current.next;  
 }  
 current.next=newNode;  
 }  
 }  
 public singlyLinked getAlternates(){  
 if (head==null || head.next==null){  
 return null;  
 }  
 Node currentNode = head;  
 singlyLinked alternateList = new singlyLinked();  
 while (currentNode != null && currentNode.next != null) {  
 alternateList.Add(currentNode.next.data);  
 currentNode.next = currentNode.next.next;  
 if (currentNode.next == null) {  
 break;  
 }  
 currentNode = currentNode.next;  
 }  
 return alternateList;  
 }  
 public void reverse() {  
 Node prev = head;  
 Node curr = head.next;  
 while (curr != null) {  
 Node next = curr.next;  
 curr.next = prev;  
 prev = curr;  
 curr = next;  
 }  
 head.next = null;  
 head = prev;  
 }  
 public void append(singlyLinked list) {  
 Node current = list.head;  
 while (current != null) {  
 Add(current.data);  
 current = current.next;  
 }  
 }  
 public void display() {  
 Node current = head;  
 while (current != null) {  
 System.*out*.print(current.data + " -> ");  
 current = current.next;  
 }  
 System.*out*.println();  
 System.*out*.println();  
 }  
}

