public class CircularLinkedList {

static class Node {

int data;

Node next;

Node() {}

Node(int data) {

this.data = data;

}

}

private Node head;

private Node tail;

// constructor

public CircularLinkedList() {

this.head = null;

this.tail = null;

}

public boolean isEmpty() {

return head == null;

}

**/\*\***

**\* insertAtFirst**

**\*\*/**

public void insertAtFirst(int data) {

Node newNode = new Node(data);

//Checks if the list is empty.

if (head == null) {

//If list is empty, both head and tail would point to new node.

head = newNode;

tail = newNode;

newNode.next = head;

} else {

//Store data into temporary node

Node temp = head;

//New node will point to temp as next node

newNode.next = temp;

//New node will be the head node

head = newNode;

//Since, it is circular linked list tail will point to head.

tail.next = head;

}

}

**/\*\***

**\* insertAtLast**

**\* \*/**

public void insertAtLast(int data) {

//Create new node

Node newNode = new Node(data);

//Checks if the list is empty.

if (head == null) {

//If list is empty, both head and tail would point to new node.

head = newNode;

tail = newNode;

newNode.next = head;

} else {

//tail will point to new node.

tail.next = newNode;

//New node will become new tail.

tail = newNode;

//Since, it is circular linked list tail will point to head.

tail.next = head;

}

}

**/\*\***

**\***

**\* Insert at specified Position**

**\*/**

public void insertAtIndex(int data, int position) {

Node temp, newNode;

int i, count;

newNode = new Node();

temp = head;

count = size();

if (temp == null || size() < position)

System.out.println("Index is greater than size of the list");

else {

newNode.data = data;

for (i = 1; i < position - 1; i++) {

temp = temp.next;

}

newNode.next = temp.next;

temp.next = newNode;

}

}

**/\*\***

**\* delete the first node.**

**\*/**

public void deleteFirst() {

if (head == null) {

return;

} else {

if (head != tail) {

head = head.next;

tail.next = head;

}

//If the list contains only one element

//then it will remove it and both head and tail will point to null

else {

head = tail = null;

}

}

}

**/\*\***

**\*Delete at Last**

**\*/**

public void deleteLast() {

if (head == null) {

return;

} else {

if (head != tail) {

Node current = head;

//Loop will iterate till the second last element as current.next is pointing to tail

while (current.next != tail) {

current = current.next;

}

//Second last element will be new tail

tail = current;

//Tail will point to head as it is a circular linked list

tail.next = head;

}

//If the list contains only one element

//Then it will remove it and both head and tail will point to null

else {

head = tail = null;

}

}

}

**/\*\***

**\***

**\* Delete at Specified Position**

**\*/**

public void deleteNode(int data) {

if (head == null)

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

// Find the required node

Node currentNode = head;

Node previousNode = new Node();

while (currentNode.data != data) {

if (currentNode.next == head) {

System.out.println("Given node with data " + data + " is not found in the circular linked list.");

break;

}

previousNode = currentNode;

currentNode = currentNode.next;

}

// Check if node is only node

if (currentNode == head && currentNode.next == head) {

head = null;

}

// If more than one node, check if

// it is first node

if (currentNode == head) {

previousNode = head;

while (previousNode.next != head) {

previousNode = previousNode.next;

}

head = currentNode.next;

previousNode.next = head;

}

// check if node is last node

else if (currentNode.next == head) {

previousNode.next = head;

} else {

previousNode.next = currentNode.next;

}

}

**/\*\***

**\* Display the list elements**

**\*/**

public void display() {

Node temp = head;

if (head != null) {

do {

System.out.printf("%d ", temp.data);

temp = temp.next;

} while (temp != head);

}

System.out.printf("\n");

}

public static void main(String[] args) {

CircularLinkedList list = new CircularLinkedList();

list.insertAtFirst(1);

list.display();

list.insertAtFirst(2);

list.display();

list.insertAtLast(3);

list.display();

list.insertAtLast(4);

list.display();

list.insertAtIndex(5, 3);

list.display();

list.deleteNode(8);

list.display();

list.deleteNode(2);

System.out.println("Node with data 2 has been deleted");

list.display();

}

}