CIRCULAR LINKED LIST

```
class CNode {
    int data;
    CNode next;

    CNode(int data) {
        this.data = data;
    }
}

public class CircularLinkedList {
    CNode head = null;

    public void insertAtFirst(int data) {
        CNode newnode = new CNode(data);
        if (head == null) {
            newnode.next = newnode;
            head = newnode;
        } else {
            CNode temp = head;
            while (temp.next != head)
                temp = temp.next;
            newnode.next = head;
            temp.next = newnode;
            head = newnode;
        }
    }

    public void insertAtLast(int data) {
        CNode newnode = new CNode(data);
        if (head == null) {
            newnode.next = newnode;
            head = newnode;
        } else {
            CNode temp = head;
            while (temp.next != head)
                temp = temp.next;
            temp.next = newnode;
            newnode.next = head;
        }
    }
```

```
public void deleteFirst() {
    if (head == null)
        return;

    if (head.next == head) {
        head = null;
    } else {
        CNode temp = head;
        while (temp.next != head)
            temp = temp.next;
        CNode toDelete = head;
        head = head.next;
        temp.next = head;
        toDelete.next = null;
        toDelete = null;
    }
}

public void deleteLast() {
    if (head == null)
        return;

    if (head.next == head) {
        head = null;
    } else {
        CNode temp = head;
        CNode prev = null;
        while (temp.next != head) {
            prev = temp;
            temp = temp.next;
        }
        prev.next = head;
        temp.next = null;
        temp = null;
    }
}

public void insertBeforeValue(int target, int data) {
    if (head == null)
        return;
    if (head.data == target) {
        insertAtFirst(data);
        return;
    }
```

```
        CNode temp = head;
        do {
            if (temp.next.data == target) {
                CNode newnode = new CNode(data);
                newnode.next = temp.next;
                temp.next = newnode;
                return;
            }
            temp = temp.next;
        } while (temp != head);
    }

    public void deleteValue(int target) {
        if (head == null)
            return;
        if (head.data == target) {
            deleteFirst();
            return;
        }
        CNode temp = head;
        do {
            if (temp.next.data == target) {
                CNode toDelete = temp.next;
                temp.next = toDelete.next;
                toDelete.next = null;
                toDelete = null;
                return;
            }
            temp = temp.next;
        } while (temp != head);
    }

    public void display() {
        if (head == null) {
            System.out.println("CLL is empty");
            return;
        }

        CNode temp = head;
        do {
            System.out.print(temp.data + " -> ");
            temp = temp.next;
        } while (temp != head);
        System.out.println("(head)");
    }
}
```

```java
public void displayReverse() {
        if (head == null) {
            System.out.println("null");
            return;
        }

        Node temp = head;
        while (temp.next != null)
            temp = temp.next;

        while (temp != null) {
            System.out.print(temp.data + " <-> ");
            temp = temp.prev;
        }
        System.out.println("null");
    }


public class MainCLL {
    public static void main(String[] args) {
        CircularLinkedList cll = new CircularLinkedList();
        cll.insertAtFirst(30);
        cll.insertAtFirst(20);
        cll.insertAtFirst(10);
        cll.display();  // 10 -> 20 -> 30 -> (head)
        cll.insertAtLast(40);
        cll.insertAtLast(50);
        cll.display();  // 10 -> 20 -> 30 -> 40 -> 50 -> (head)
        cll.insertBeforeValue(30, 25);
        cll.display();  // 10 -> 20 -> 25 -> 30 -> 40 -> 50 -> (head)
        cll.deleteFirst();
        cll.display();  // 20 -> 25 -> 30 -> 40 -> 50 -> (head)
        cll.deleteLast();
        cll.display();  // 20 -> 25 -> 30 -> 40 -> (head)
        cll.deleteValue(25);
        cll.display();  // 20 -> 30 -> 40 -> (head)
    }
}
```

PRACTICE METHODS- CLL

```
public void deleteOddPositionedNodes() {
    if (head == null || head.next == head) {
        head = null;
        return;
    }

    int index = 1;
    CNode curr = head;
    CNode prev = null;

    do {
        if (index % 2 != 0) {  // Odd position
            CNode toDelete = curr;
            if (curr == head) {
                head = head.next;
                // Find the last node
                CNode last = head;
                while (last.next != toDelete)
                    last = last.next;
                last.next = head;
                curr = head;
            } else {
                prev.next = curr.next;
                curr = curr.next;
            }
            toDelete.next = null;
            toDelete = null;
        } else {
            prev = curr;
            curr = curr.next;
        }
        index++;
    } while (curr != head);
}
```

```
public static CLL addTwoCLL(CLL l1, CLL l2) {
        CLL result = new CLL();

        if (l1.head == null || l2.head == null)
            return result;

        CNode p1 = l1.head;
        CNode p2 = l2.head;

        do {
            int sum = p1.data + p2.data;
            result.insertAtLast(sum);

            p1 = p1.next;
            p2 = p2.next;
        } while (p1 != l1.head && p2 != l2.head);

        return result;
    }
-----------------------------------------------------
public class MainCLLAdd {
    public static void main(String[] args) {
        CLL l1 = new CLL();
        CLL l2 = new CLL();

        l1.insertAtLast(2);
        l1.insertAtLast(4);
        l1.insertAtLast(6);
        l1.insertAtLast(7);
        l1.insertAtLast(9);

        l2.insertAtLast(5);
        l2.insertAtLast(3);
        l2.insertAtLast(12);
        l2.insertAtLast(11);
        l2.insertAtLast(45);

        CLL result = CLL.addTwoCLL(l1, l2);

        System.out.print("Resultant CLL: ");
        result.display();  // Output: 7 -> 7 -> 18 -> 18 -> 54 -> (head)
    }
}
```

DOUBLY LINKED LIST

```
class Node {
    int data;
    Node prev, next;

    Node(int data) {
        this.data = data;
    }
}

public class DoublyLinkedList{
    Node head;

    public void insertAtFirst(int data) {
        Node newnode = new Node(data);
        if (head == null)
            head = newnode;
        else {
            newnode.next = head;
            head.prev = newnode;
            head = newnode;
        }
    }

    public void insertAtLast(int data) {
        Node newnode = new Node(data);
        if (head == null)
            head = newnode;
        else {
            Node temp = head;
            while (temp.next != null)
                temp = temp.next;
            temp.next = newnode;
            newnode.prev = temp;
        }
    }

    public void deleteFirst() {
        if (head == null)
            return;

        Node toDelete = head;
        head = head.next;
```

```
        if (head != null)
            head.prev = null;

        toDelete.next = null;
        toDelete.prev = null;
        toDelete = null;
    }

    public void deleteLast() {
        if (head == null)
            return;

        if (head.next == null) {
            Node toDelete = head;
            head = null;
            toDelete = null;
        } else {
            Node temp = head;
            while (temp.next != null)
                temp = temp.next;
            Node toDelete = temp;
            temp.prev.next = null;

            toDelete.prev = null;
            toDelete.next = null;
            toDelete = null;
        }
    }

    public void insertBeforeValue(int target, int data) {
        if (head == null)
            return;

        if (head.data == target) {
            insertAtFirst(data);
            return;
        }

        Node temp = head;
        while (temp != null && temp.data != target)
            temp = temp.next;

        if (temp == null)
            return;

        Node newnode = new Node(data);
```

```
        newnode.prev = temp.prev;
        newnode.next = temp;
        temp.prev.next = newnode;
        temp.prev = newnode;
    }



 public void deleteValue(int target) {
        if (head == null)
            return;

        if (head.data == target) {
            deleteFirst();
            return;
        }

        Node temp = head;
        while (temp != null && temp.data != target)
            temp = temp.next;

        if (temp == null)
            return;

        Node toDelete = temp;
        if (temp.next != null)
            temp.next.prev = temp.prev;
        if (temp.prev != null)
            temp.prev.next = temp.next;

        toDelete.prev = null;
        toDelete.next = null;
        toDelete = null;
    }

    public void display() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + " <-> ");
            temp = temp.next;
        }
        System.out.println("null");
    }
}
```

```java
public class MainDLL {
    public static void main(String[] args) {
        DoublyLinkedList dll = new DoublyLinkedList();

        dll.insertAtFirst(30);
        dll.insertAtFirst(20);
        dll.insertAtFirst(10);
        dll.display();  // 10 <-> 20 <-> 30 <-> null

        dll.insertAtLast(40);
        dll.insertAtLast(50);
        dll.display();  // 10 <-> 20 <-> 30 <-> 40 <-> 50 <-> null

        dll.insertBeforeValue(30, 25);
        dll.display();  // 10 <-> 20 <-> 25 <-> 30 <-> 40 <-> 50 <-> null

        dll.deleteFirst();
        dll.display();  // 20 <-> 25 <-> 30 <-> 40 <-> 50 <-> null

        dll.deleteLast();
        dll.display();  // 20 <-> 25 <-> 30 <-> 40 <-> null

        dll.deleteValue(25);
        dll.display();  // 20 <-> 30 <-> 40 <-> null
    }
}
```

PRACTICE METHODS - DLL

```
public void deleteOddNodes() {
    Node temp = head;
    while (temp != null) {
        if (temp.data % 2 != 0) {
            Node toDelete = temp;

            if (temp == head)
                head = head.next;

            if (temp.prev != null)
                temp.prev.next = temp.next;
            if (temp.next != null)
                temp.next.prev = temp.prev;

            temp = temp.next;

            toDelete.prev = null;
            toDelete.next = null;
            toDelete = null;
        } else {
            temp = temp.next;
        }
    }
}
```

```java
public void insertSorted(int data) {
    Node newnode = new Node(data);
    if (head == null || head.data >= data) {
        insertAtFirst(data);
        return;
    }

    Node temp = head;
    while (temp.next != null && temp.next.data < data)
        temp = temp.next;

    newnode.next = temp.next;
    if (temp.next != null)
        temp.next.prev = newnode;

    temp.next = newnode;
    newnode.prev = temp;
}
```