Circular Linked List:

In a **circular doubly linked list**, each node has two pointers (`next` and `prev`), just like in a regular doubly linked list, but the difference is that:

1. The `next` pointer of the last node points to the head node.
2. The `prev` pointer of the head node points to the last node.

This creates a circular structure, allowing traversal from any node in a continuous loop in either direction.

Key Concepts of Circular Doubly Linked List

1. **Circular Structure**: The `next` of the last node points back to the head, and the `prev` of the head points to the last node.
2. **Traversal in Both Directions**: Nodes can be traversed in both forward and backward directions continuously.
3. **Efficient for Cyclic Data Processing**: Useful in applications that require wrapping around, like task scheduling.

Circular Doubly Linked List Implementation in Java

Here's how to modify your `DoublyLinkedList` class to make it circular.

```
public class CircularDoublyLinkedList {

    private Node head;
    private Node tail;

    public CircularDoublyLinkedList() {
        head = null;
        tail = null;
    }
```

```
    // Add data to the front (head)
    public void addFront(int data) {
        Node newNode = new Node(data);

        if (head == null) {
            head = tail = newNode;
            head.next = head;
            head.prev = head;
        } else {
            newNode.next = head;
            newNode.prev = tail;
            head.prev = newNode;
            tail.next = newNode;
            head = newNode;
        }
    }

    // Add data to the end (tail)
    public void addEnd(int data) {
        Node newNode = new Node(data);

        if (tail == null) {
            head = tail = newNode;
            head.next = head;
            head.prev = head;
        } else {
            newNode.prev = tail;
            newNode.next = head;
            tail.next = newNode;
            head.prev = newNode;
            tail = newNode;
        }
    }

    // Delete from front (head)
```

```
    public void deleteFront() {
        if (head == null) {
            System.out.println("List is empty!");
            return;
        }

        if (head == tail) {  // Only one node
            head = tail = null;
        } else {
            head = head.next;
            head.prev = tail;
            tail.next = head;
        }
    }

    // Delete from end (tail)
    public void deleteEnd() {
        if (tail == null) {
            System.out.println("List is empty!");
            return;
        }

        if (head == tail) {  // Only one node
            head = tail = null;
        } else {
            tail = tail.prev;
            tail.next = head;
            head.prev = tail;
        }
    }

    // Display forward
    public void displayForward() {
        if (head == null) return;

        Node current = head;
```

```
        do {
            System.out.print(current.data + "\t");
            current = current.next;
        } while (current != head);
        System.out.println();
    }

    // Display backward
    public void displayBackward() {
        if (tail == null) return;

        Node current = tail;
        do {
            System.out.print(current.data + "\t");
            current = current.prev;
        } while (current != tail);
        System.out.println();
    }

    // Node class
    private static class Node {
        int data;
        Node next;
        Node prev;

        public Node(int data) {
            this.data = data;
            next = null;
            prev = null;
        }
    }
}
```

Key Operations

- **Insertion at the beginning and end**: Adjusts `head` and `tail` pointers and sets up circular links.
- **Deletion from the beginning and end**: Updates `head`, `tail`, and adjusts the circular references.
- **Display Forward/Backward**: Traverse until reaching the starting node again.

Important Interview Questions on Circular Doubly Linked Lists

1. **What are the differences between circular singly, doubly, and non-circular linked lists?**
2. **How would you detect if a linked list is circular (circular singly or doubly)?**
3. **When would you use a circular doubly linked list over other linked list types?**
4. **How can you implement a circular linked list in a real-world application, like a playlist or a round-robin scheduler?**
5. **How would you find the middle element in a circular linked list?**
6. **Can you implement a queue using a circular doubly linked list? How?**

This code and explanation provide a strong foundation to understand and implement circular doubly linked lists in various scenarios!