**Steps to Implement equals():**

1. **Check for equality of list sizes**:
   * If the two lists have different sizes, they are not equal.
2. **Compare corresponding elements**:
   * Starting from the head of each list, we compare the elements in the lists one by one.
   * Since both lists are circular, we will traverse each list once and check if the nodes in the two lists have the same values at each corresponding position.
3. **Consider circularity**:
   * In a circular linked list, once you reach the end of the list, you will circle back to the beginning. This is handled by comparing nodes in a circular manner.

**Implementation in Java:**

class Node {

int data;

Node next;

public Node(int data) {

this.data = data;

this.next = null;

}

}

class CircularlyLinkedList {

Node head;

public CircularlyLinkedList() {

head = null;

}

// Method to check if two circularly linked lists are equal

public boolean equals(CircularlyLinkedList other) {

if (other == null) {

return false; // If the other list is null, they are not equal

}

// Check if the sizes are different

if (this.size() != other.size()) {

return false;

}

// If both lists are empty, they are considered equal

if (this.head == null && other.head == null) {

return true;

}

Node currentThis = this.head;

Node currentOther = other.head;

// Traverse both lists and compare corresponding elements

do {

if (currentThis.data != currentOther.data) {

return false; // If any corresponding elements are different, the lists are not equal

}

currentThis = currentThis.next;

currentOther = currentOther.next;

} while (currentThis != this.head); // Stop when we circle back to the head of the first list

// If we didn't find any unequal elements, the lists are equal

return true;

}

// Method to get the size of the circular linked list

public int size() {

if (head == null) {

return 0;

}

Node current = head;

int count = 0;

do {

count++;

current = current.next;

} while (current != head);

return count;

}

// Other methods like insert, delete, etc., can be added here

}

**Explanation:**

1. **Equality Check**:
   * First, we check if the other list is null. If it is, the lists are not equal.
   * Next, we compare the sizes of the two lists. If they are not equal, we return false immediately.
   * If both lists are empty (head == null), we return true because two empty circular linked lists are considered equal.
2. **Traverse and Compare Nodes**:
   * We traverse both lists simultaneously starting from their head nodes.
   * For each node in the lists, we check if the data values (currentThis.data and currentOther.data) are the same.
   * If we find any node where the data values don't match, we return false because the lists are not equal.
   * The traversal continues until we have circled back to the head of the first list (currentThis != this.head).
3. **Return True if Equal**:
   * If we successfully traverse both lists and find no mismatched elements, we return true, indicating the lists are equal.

**Example:**

public static void main(String[] args) {

CircularlyLinkedList list1 = new CircularlyLinkedList();

CircularlyLinkedList list2 = new CircularlyLinkedList();

// Add elements to the lists (methods to add elements are assumed to be implemented)

// For example, list1: 1 -> 2 -> 3 -> 1 (circular)

// list2: 1 -> 2 -> 3 -> 1 (circular)

if (list1.equals(list2)) {

System.out.println("The lists are equal.");

} else {

System.out.println("The lists are not equal.");

}

}

**Time Complexity:**

* The time complexity of the equals() method is O(n)O(n)O(n), where nnn is the number of nodes in the list. This is because we traverse both lists once, comparing the data of each node.

**Edge Cases:**

* **Both lists are empty**: The method will return true because two empty lists are considered equal.
* **Lists with different sizes**: If the sizes are different, the method will return false immediately.
* **Lists with the same data but different node arrangements**: Since we're considering the circular nature of the list and the order of the nodes, the lists will only be equal if the elements appear in the same order starting from the head.