**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 get the size of the circular linked list

public int size() {

if (head == null) {

return 0; // If the list is empty, size is 0

}

Node current = head;

int count = 0;

// Traverse the list and count the nodes

do {

count++;

current = current.next;

} while (current != head); // Stop when we circle back to the head

return count;

}

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

}

**Explanation:**

1. **Check if the list is empty**:
   * If head == null, the list is empty, so we return a size of 0.
2. **Traverse the list**:
   * We start from the head and use a current pointer to iterate through the nodes.
   * Since it's a circular linked list, we use a do-while loop. This loop ensures we visit every node, and the loop stops once we circle back to the head.
3. **Counting nodes**:
   * We increment a count variable each time we visit a node. This keeps track of the number of nodes in the list.
4. **Return the size**:
   * After completing the traversal and counting all nodes, we return the count, which represents the size of the circular linked list.

**Example:**

For a circularly linked list with nodes 1 -> 2 -> 3 -> 1 (back to the head):

1. Start at head (node with value 1).
2. Traverse through nodes: 1 → 2 → 3 → (back to 1).
3. The size of the list is 3.

**Time Complexity:**

* The time complexity is O(n)O(n)O(n), where nnn is the number of nodes in the circular linked list. This is because we traverse each node exactly once.

**Edge Cases:**

* **Empty list**: If the list is empty (head == null), the size will be 0.
* **Single-node circular list**: If the list contains only one node, the size will be 1, and the loop will stop when it circles back to the head.