**Steps to Clone a Circularly Linked List:**

1. **Check if the list is empty**:
   * If the original list is empty (head == null), return an empty cloned list (i.e., null).
2. **Create a new circular linked list**:
   * Create a new CircularlyLinkedList object to store the cloned list.
3. **Traverse the original list**:
   * Traverse the original circular linked list starting from its head, and for each node, create a new node with the same data value and append it to the cloned list.
4. **Ensure circularity**:
   * Once the new list has been populated with nodes, ensure that the last node in the cloned list points back to its head to maintain the circular structure.

**Pseudocode:**

CloneCircularList(L):

if L is empty:

return empty cloned list

// Create the head for the cloned list

newListHead = new Node(L.head.data)

currentOriginal = L.head.next

currentClone = newListHead

// Traverse the original list and clone the nodes

while currentOriginal != L.head:

newNode = new Node(currentOriginal.data)

currentClone.next = newNode

currentClone = newNode

currentOriginal = currentOriginal.next

// Make the cloned list circular by pointing the last node to the head

currentClone.next = newListHead

return newListHead

**Java Code Implementation:**

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 clone the circularly linked list

public CircularlyLinkedList clone() {

if (head == null) {

return new CircularlyLinkedList(); // If the original list is empty, return an empty list

}

CircularlyLinkedList clonedList = new CircularlyLinkedList();

Node currentOriginal = head;

// Create the first node of the cloned list

clonedList.head = new Node(currentOriginal.data);

Node currentClone = clonedList.head;

// Traverse the original list and clone the nodes

currentOriginal = currentOriginal.next;

while (currentOriginal != head) {

Node newNode = new Node(currentOriginal.data);

currentClone.next = newNode;

currentClone = newNode;

currentOriginal = currentOriginal.next;

}

// Make the cloned list circular by pointing the last node to the head

currentClone.next = clonedList.head;

return clonedList;

}

// Method to print the list (for testing purposes)

public void printList() {

if (head == null) {

System.out.println("The list is empty.");

return;

}

Node current = head;

do {

System.out.print(current.data + " ");

current = current.next;

} while (current != head);

System.out.println();

}

// Method to add nodes for testing purposes

public void addNode(int data) {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

newNode.next = head; // Circular link

} else {

Node current = head;

while (current.next != head) {

current = current.next;

}

current.next = newNode;

newNode.next = head; // Circular link

}

}

}

public class Main {

public static void main(String[] args) {

// Create and populate the original circular linked list

CircularlyLinkedList list = new CircularlyLinkedList();

list.addNode(1);

list.addNode(2);

list.addNode(3);

list.addNode(4);

// Print the original list

System.out.print("Original list: ");

list.printList();

// Clone the list

CircularlyLinkedList clonedList = list.clone();

// Print the cloned list

System.out.print("Cloned list: ");

clonedList.printList();

}

}

**Explanation:**

1. **Cloning Process**:
   * We check if the original list is empty. If it is, we return a new empty circular list.
   * If the list is not empty, we start by creating a new node for the cloned list with the same value as the head of the original list.
   * We then traverse the original list starting from head and for each node, create a new node in the cloned list with the same data and link it to the previous node in the cloned list.
   * After traversing the entire list, we ensure that the last node in the cloned list points back to the head to maintain the circular structure.
2. **Maintaining Circularity**:
   * After all nodes are copied, we ensure the cloned list is circular by setting the next pointer of the last node to point back to the head of the cloned list.
3. **Testing**:
   * We create a CircularlyLinkedList, add a few nodes, and print both the original and cloned lists to verify that the cloning process works correctly.

**Time Complexity:**

* **Time Complexity**: O(n)O(n)O(n), where nnn is the number of nodes in the circular linked list. We traverse the list once to clone it.
* **Space Complexity**: O(n)O(n)O(n), because we are creating a new list with the same number of nodes as the original list.

**Example:**

For the list 1 -> 2 -> 3 -> 4 -> 1 (back to 1):

Original list: 1 2 3 4

Cloned list: 1 2 3 4