To implement a **circular doubly linked list** without any sentinel nodes, we need to adjust the structure of the doubly linked list so that the next pointer of the last node points to the first node, and the prev pointer of the first node points to the last node. This creates a circular structure where there is no distinct "beginning" or "end" of the list, and the list can be navigated indefinitely in both directions.

**Requirements:**

1. **No Sentinels:** The list should not have any sentinel nodes, and the list will be circular, meaning the first node's prev will point to the last node, and the last node's next will point to the first node.
2. **Public Behaviors:** The list should support behaviors like adding/removing elements from both ends, checking size, and printing the list.
3. **Two New Methods:**
   * **rotate()**: This method will rotate the list to the right (move the last node to the front).
   * **rotateBackward()**: This method will rotate the list to the left (move the first node to the end).

**Circular Doubly Linked List Design:**

* The circular doubly linked list will be implemented without any sentinel nodes.
* We will maintain a reference to the head of the list (head), which is the starting point for all operations.
* The next pointer of the last node will point to the head, and the prev pointer of the head will point to the last node.

**Java Code Implementation:**

public class CircularDoublyLinkedList {

// الكلاس Node لتمثيل العقدة

private static class Node {

Object element;

Node prev;

Node next;

public Node(Object element) {

this.element = element;

this.prev = null;

this.next = null;

}

}

private Node head; // بداية القائمة

private int size; // حجم القائمة

public CircularDoublyLinkedList() {

head = null;

size = 0;

}

// دالة لإضافة عنصر إلى بداية القائمة

public void addFirst(Object element) {

Node newNode = new Node(element);

if (size == 0) {

head = newNode;

newNode.next = head;

newNode.prev = head;

} else {

Node last = head.prev;

newNode.next = head;

newNode.prev = last;

last.next = newNode;

head.prev = newNode;

head = newNode;

}

size++;

}

// دالة لإضافة عنصر إلى نهاية القائمة

public void addLast(Object element) {

Node newNode = new Node(element);

if (size == 0) {

head = newNode;

newNode.next = head;

newNode.prev = head;

} else {

Node last = head.prev;

newNode.next = head;

newNode.prev = last;

last.next = newNode;

head.prev = newNode;

}

size++;

}

// دالة لإزالة العنصر الأول من القائمة

public Object removeFirst() {

if (size == 0) return null;

Object removedElement = head.element;

if (size == 1) {

head = null;

} else {

Node last = head.prev;

head = head.next;

head.prev = last;

last.next = head;

}

size--;

return removedElement;

}

// دالة لإزالة العنصر الأخير من القائمة

public Object removeLast() {

if (size == 0) return null;

Node last = head.prev;

Object removedElement = last.element;

if (size == 1) {

head = null;

} else {

Node secondLast = last.prev;

secondLast.next = head;

head.prev = secondLast;

}

size--;

return removedElement;

}

// دالة لإرجاع حجم القائمة

public int size() {

return size;

}

// دالة لطباعة القائمة

public void printList() {

if (size == 0) {

System.out.println("List is empty.");

return;

}

Node current = head;

do {

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

current = current.next;

} while (current != head); // التكرار حتى نصل إلى العقدة التي تشبه الرأس

System.out.println();

}

// دالة لتدوير القائمة إلى اليمين (التحريك إلى الأمام)

public void rotate() {

if (size > 1) {

head = head.prev; // تحريك العقدة الأخيرة إلى الأمام

}

}

// دالة لتدوير القائمة إلى اليسار (التحريك إلى الخلف)

public void rotateBackward() {

if (size > 1) {

head = head.next; // تحريك العقدة الأولى إلى الخلف

}

}

// اختبار الكود

public static void main(String[] args) {

CircularDoublyLinkedList list = new CircularDoublyLinkedList();

// إضافة عناصر إلى القائمة

list.addFirst(1);

list.addFirst(2);

list.addFirst(3);

list.addLast(4);

list.addLast(5);

System.out.println("List after additions:");

list.printList(); // يجب أن تطبع: 3 2 1 4 5

// تدوير القائمة إلى اليمين

list.rotate();

System.out.println("List after rotate:");

list.printList(); // يجب أن تطبع: 5 3 2 1 4

// تدوير القائمة إلى اليسار

list.rotateBackward();

System.out.println("List after rotateBackward:");

list.printList(); // يجب أن تطبع: 3 2 1 4 5

// إزالة عناصر

list.removeFirst();

list.removeLast();

System.out.println("List after removals:");

list.printList(); // يجب أن تطبع: 2 1 4

}

}

**شرح الكود:**

1. **Node Class:**
   * يحتوي على المتغيرات element (العنصر المخزن في العقدة)، و prev و next (المؤشرات إلى العقد السابقة والتالية).
2. **CircularDoublyLinkedList Class:**
   * **addFirst(Object element)**: يضيف العنصر إلى بداية القائمة.
   * **addLast(Object element)**: يضيف العنصر إلى نهاية القائمة.
   * **removeFirst()**: يزيل العنصر من بداية القائمة.
   * **removeLast()**: يزيل العنصر من نهاية القائمة.
   * **size()**: يعيد حجم القائمة.
   * **printList()**: يطبع العناصر في القائمة.
   * **rotate()**: يقوم بتدوير القائمة إلى اليمين (حركة العقدة الأخيرة إلى الأمام).
   * **rotateBackward()**: يقوم بتدوير القائمة إلى اليسار (حركة العقدة الأولى إلى الخلف).

**النتائج المتوقعة:**

List after additions:

3 2 1 4 5

List after rotate:

5 3 2 1 4

List after rotateBackward:

3 2 1 4 5

List after removals:

2 1 4

**التعقيد الزمني:**

* **الزمن:** جميع العمليات (الإضافة، الإزالة، الطباعة) تعمل في O(1)O(1)O(1) لأننا نقوم بتعديل المؤشرات مباشرة.
* **الفضاء:** O(n)O(n)O(n) حيث أن nnn هو عدد العناصر في القائمة.