To reimplement the DoublyLinkedList class using **only one sentinel node** that guards both ends of the list, we can use a single sentinel node placed in the middle of the list. This node will serve as a placeholder to simplify the operations and handle both the head and the tail of the list.

The main idea is that the sentinel node will be the only node that does not store any actual data. It will point to the first and the last elements of the list via its next and prev pointers, respectively.

**Design Changes:**

1. **Single Sentinel Node:** We will have only one sentinel node, which will be placed at the start of the list. The next pointer of the sentinel will point to the first actual element, and the prev pointer will point to the last actual element.
2. **Simplified Operations:** Since there is only one sentinel, all list operations like insertion, deletion, and traversal will be adjusted to work with this central sentinel node.

public class DoublyLinkedList {

// تعريف 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 sentinel; // العقدة السانتيتل الوحيدة

public DoublyLinkedList() {

sentinel = new Node(null); // العقدة السانتيتل التي لا تحتوي على بيانات

sentinel.prev = sentinel; // الرابط إلى نفسه في البداية

sentinel.next = sentinel; // الرابط إلى نفسه في البداية

}

// دالة لإضافة عنصر إلى نهاية القائمة

public void addLast(Object element) {

Node newNode = new Node(element);

// ربط العقدة الجديدة بالنهاية

newNode.prev = sentinel.prev;

newNode.next = sentinel;

// ربط العقدة السابقة بالعقدة الجديدة

sentinel.prev.next = newNode;

sentinel.prev = newNode;

}

// دالة لإضافة عنصر إلى بداية القائمة

public void addFirst(Object element) {

Node newNode = new Node(element);

// ربط العقدة الجديدة بالبداية

newNode.next = sentinel.next;

newNode.prev = sentinel;

// ربط العقدة التالية بالعقدة الجديدة

sentinel.next.prev = newNode;

sentinel.next = newNode;

}

// دالة لإزالة العنصر من بداية القائمة

public Object removeFirst() {

if (sentinel.next == sentinel) {

return null; // القائمة فارغة

}

Node firstNode = sentinel.next;

sentinel.next = firstNode.next;

firstNode.next.prev = sentinel;

return firstNode.element;

}

// دالة لإزالة العنصر من نهاية القائمة

public Object removeLast() {

if (sentinel.prev == sentinel) {

return null; // القائمة فارغة

}

Node lastNode = sentinel.prev;

sentinel.prev = lastNode.prev;

lastNode.prev.next = sentinel;

return lastNode.element;

}

// دالة لإرجاع حجم القائمة

public int size() {

int count = 0;

Node current = sentinel.next;

while (current != sentinel) {

count++;

current = current.next;

}

return count;

}

// دالة لطباعة العناصر في القائمة

public void printList() {

Node current = sentinel.next;

while (current != sentinel) {

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

current = current.next;

}

System.out.println();

}

// اختبار الكود

public static void main(String[] args) {

DoublyLinkedList list = new DoublyLinkedList();

// إضافة عناصر إلى القائمة

list.addLast(1);

list.addLast(2);

list.addLast(3);

list.addFirst(0);

list.addLast(4);

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

list.printList(); // يجب أن تطبع: 0 1 2 3 4

// إزالة عناصر من البداية والنهاية

list.removeFirst();

list.removeLast();

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

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

}

}