**Plan:**

1. **Create a New List:** The clone() method should create a new instance of the doubly linked list.
2. **Traverse the Original List:** We will traverse the original list starting from the head and create new nodes for each element in the original list.
3. **Copy Elements:** Each new node in the clone list will contain the same element as the corresponding node in the original list, and the next and prev pointers will be appropriately set.
4. **Return the New List:** Finally, return the cloned list.

**Java Code Implementation:**

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 head; // بداية القائمة

private int size; // حجم القائمة

public DoublyLinkedList() {

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 لاستنساخ القائمة

public DoublyLinkedList clone() {

DoublyLinkedList cloneList = new DoublyLinkedList();

if (size == 0) {

return cloneList; // قائمة فارغة، نعيد قائمة فارغة

}

Node current = head;

do {

cloneList.addLast(current.element); // إضافة العنصر إلى القائمة المستنسخة

current = current.next;

} while (current != head); // التكرار حتى نعود إلى العقدة الأولى

return cloneList;

}

// اختبار الكود

public static void main(String[] args) {

DoublyLinkedList list = new DoublyLinkedList();

// إضافة عناصر إلى القائمة

list.addFirst(1);

list.addFirst(2);

list.addFirst(3);

list.addLast(4);

list.addLast(5);

System.out.println("Original List:");

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

// استنساخ القائمة

DoublyLinkedList clonedList = list.clone();

System.out.println("Cloned List:");

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

// التحقق من استقلالية القائمتين

list.removeFirst(); // إزالة العنصر الأول من القائمة الأصلية

System.out.println("Original List after removal:");

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

System.out.println("Cloned List after removal from original:");

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

}

}