Erick Gonzalez Parada #178145

Andre Francois Duhamel Gutiérrez #177315

Alejandro González Díaz #178645

# Activity #8 Double Circular Linked List

[[1]](#_Bibliography)

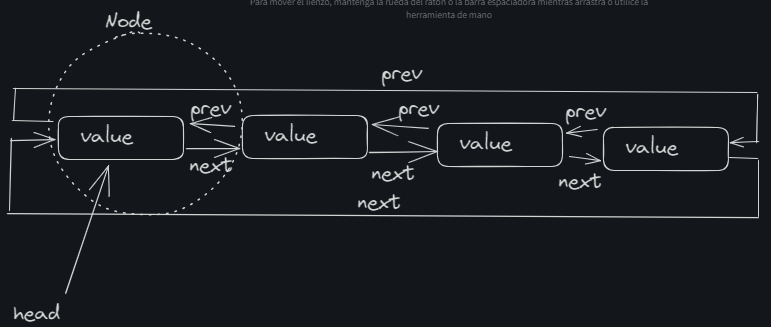
As we already know a NORMAL CIRCULAR linked list links with the objective to form a circle making it so that there is no NULL at the end after an element was inserter in the list, therefore a doubly circular linked list contains now two pointers, one pointing to the previous and the other one to the next node, with this approach we can traverse this list as we wish.

## Project Objective

The objective of this activity is to know and domain how a doubly circular linked list works NOT ONLY in C language, we need to understand the workflow of the doubly circular linked list in general.

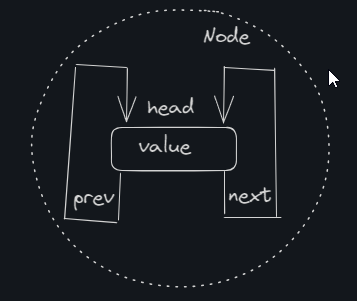
## Solution Design

As a team we decided to have all the time our graphic representation of a linked list (in general), with a graphical representation is easier to understand how we need to achieve the algorithms of any function for the circular linked list.

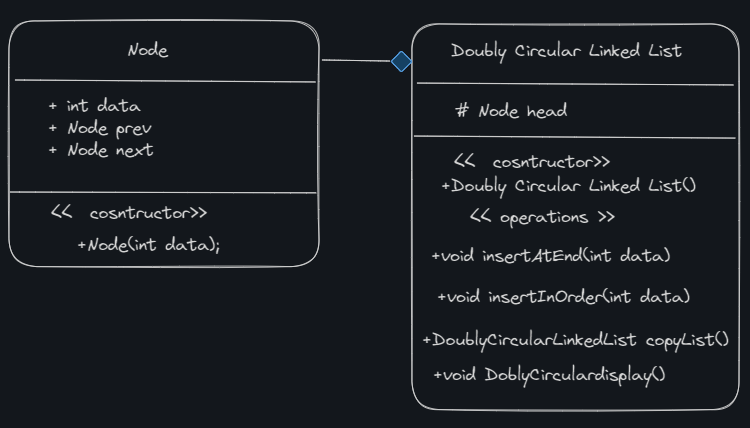


### Main Steps

The big deal in order to convert our graphic ideas into a real working algorithm is to focus our attention in the manipulation of the next pointer and the previous pointer.



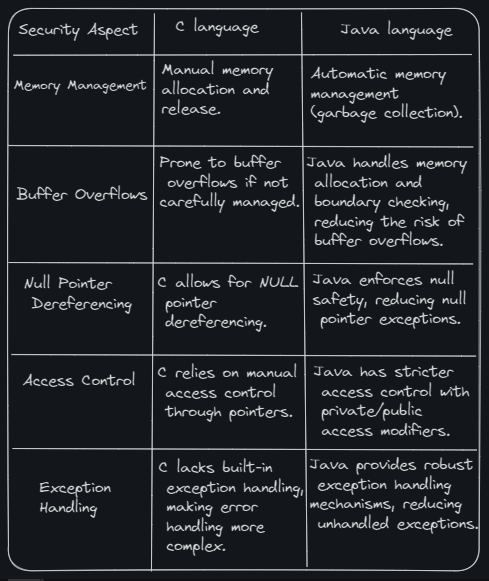
Both of this need to be updated every time we add or delete an element, here is the UML Diagram that help us a lot when using java:



## Security Aspects

Our most critical security aspect to always care is the **Buffer Overflow**, if we set a limit of nodes that could be a solution to the Buffer Overflow draining all the memory nevertheless there are ways to prevent this memory error and this one will be implemented in all the functions asked.

Also **input handling** is always need it when we let the user/client enter any char data (specially strings) input(s) to set the value of any node or something else.



## Analysis

We are going to have all in one main and go through the 3 methods one by one:

### Main code

public class Main {

public static void main(String[] args) {

DoublyCircularLinkedList list = new DoublyCircularLinkedList();

System.out.println("Testing insertAtEnd method:");

list.insertAtEnd(10);

list.insertAtEnd(30);

list.insertAtEnd(20);

list.insertAtEnd(40);

list.DoblyCirculardisplay();

System.out.println("Testing insertInOrder method:");

DoublyCircularLinkedList orderedList = new DoublyCircularLinkedList();

orderedList.insertInOrder(30);

orderedList.insertInOrder(10);

orderedList.insertInOrder(40);

orderedList.insertInOrder(20);

orderedList.DoblyCirculardisplay();

System.out.println("Testing copyList method:");

DoublyCircularLinkedList copiedList = orderedList.copyList();

copiedList.DoblyCirculardisplay();

}

}

As u could see we implemented another function to display the list more quickly instead of doing a mess in the main, let’s start with the first part.

### insertAtEnd Method

public void insertAtEnd(int data) {

CHNode newCHNode = new CHNode(data);

if (head == null) {

insertAtStart(data);

} else {

newCHNode.next = head;

newCHNode.prev = head.prev;

head.prev.next = newCHNode;

head.prev = newCHNode;

}

}

### Test in Main

System.out.println("Testing insertAtEnd method:");

list.insertAtEnd(10);

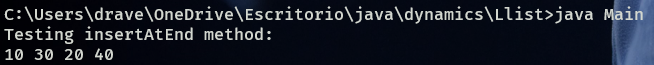
list.insertAtEnd(30);

list.insertAtEnd(20);

list.insertAtEnd(40);

list.DoblyCirculardisplay();

### Output



### insertInOrder Method

public void insertInOrder(int data) {

CHNode newCHNode = new CHNode(data);

if (head == null) {

head = newCHNode;

newCHNode.next = newCHNode;

newCHNode.prev = newCHNode;

} else {

CHNode current = head;

do {

if (current.data >= data) {

break;

}

current = current.next;

} while (current != head);

newCHNode.next = current;

newCHNode.prev = current.prev;

current.prev.next = newCHNode;

current.prev = newCHNode;

if (current == head && data < head.data) {

head = newCHNode;

}

}

}

### Test in Main

System.out.println("Testing insertInOrder method:");

DoublyCircularLinkedList orderedList = new DoublyCircularLinkedList();

orderedList.insertInOrder(30);

orderedList.insertInOrder(10);

orderedList.insertInOrder(40);

orderedList.insertInOrder(20);

orderedList.DoblyCirculardisplay();

### Output



### copyLink Method

public DoublyCircularLinkedList copyList() {

DoublyCircularLinkedList newList = new DoublyCircularLinkedList();

if (head != null) {

CHNode current = head;

do {

newList.insertAtEnd(current.data);

current = current.next;

} while (current != head);

}

return newList;

}

### Test in Main

System.out.println("Testing copyList method:");

DoublyCircularLinkedList copiedList = orderedList.copyList();

copiedList.DoblyCirculardisplay();

### Output



Here is another set of tests:

Main:

public class Main {

public static void main(String[] args) {

DoublyCircularLinkedList list = new DoublyCircularLinkedList();

System.out.println("Testing insertAtEnd method:");

list.insertAtEnd(5);

list.insertAtEnd(3);

list.insertAtEnd(2);

list.insertAtEnd(4);

list.DoblyCirculardisplay();

System.out.println("Testing insertInOrder method:");

DoublyCircularLinkedList orderedList = new DoublyCircularLinkedList();

orderedList.insertInOrder(5);

orderedList.insertInOrder(1);

orderedList.insertInOrder(4);

orderedList.insertInOrder(2);

orderedList.DoblyCirculardisplay();

System.out.println("Testing copyList method:");

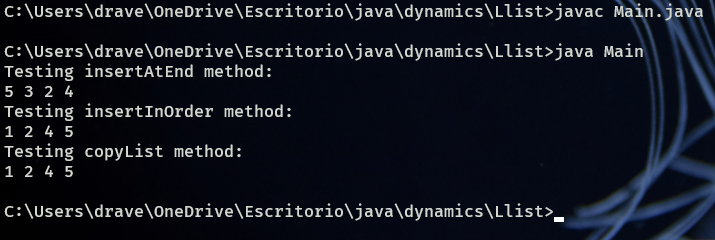
DoublyCircularLinkedList copiedList = orderedList.copyList();

copiedList.DoblyCirculardisplay();

}

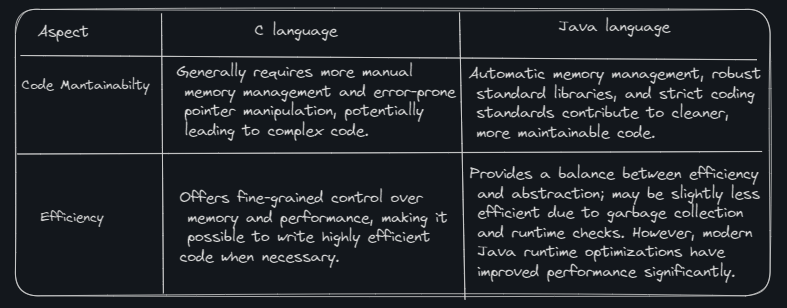
}

Output:



Note that this list is circular, meaning that when an element is inserted the new Null value becomes the head, now it’s important to also mention that these methods in particular don’t or can’t prove the veracity of traversing in both ways that would need more fixing in the code if we want the client to perceive that e.g. in the console or the interface that we have had built for him.

## Conclusion



Concluding one real-world application where a doubly circular linked list can be used is in the management of playlists in a music player. In a music player application, you often need to create playlists that allow for both forward and backward traversal of songs. A doubly circular linked list is a suitable data structure for this purpose because it enables seamless looping between the first and last songs in the playlist.

## Bibliography

Botman. (2023, 30 junio). Doubly Circular Linked List |C++ Implementation - ProgrammerCave - Medium. *Medium*. https://medium.com/programmercave/doubly-circular-linked-list-c-implementation-850165897321#:~:text=A%20node%20in%20a%20doubly,can%20traverse%20in%20both%20direction.&text=Here%20is%20a%20meme%20to,last%20node%20is%20the%20tail.