# Trabajo sobre LISTAS DOBLEMENTE ENLAZADAS

Autor: Jonathan Andres Rosero Soto

Presente un menú que permita escoger y realizar una de las siguientes opciones:
* Insertar un nuevo nodo al inicio de la lista
* Insertar un nuevo nodo al final de la lista
* Insertar un nuevo nodo en la posición correspondiente (para obtener una lista ordenada de forma ascendente)
* Eliminar un elemento determinado de la lista
* Recorrer la lista desde Head —> Tail (de la cabeza hacia la cola)
* Recorrer la lista desde Tail —> Head (de la cola hacia la cabeza)

## La clase Nodo

```
package trabajo_estructuras;
/**
 *
 * @author JRosero
 */

public class Nodo {
    int valor;
    Nodo sig;
    Nodo act;

    public Nodo(int valor) {
        this.valor = valor;
        this.sig = null;
        this.act = null;
    }



}
```

## Clase Principal(ejecutora) y metodos de ingreso, ordenamiento, eliminación y visualización.

```
package trabajo_estructuras;

import java.util.Scanner;

/**
 *
 * @author JRosero
 */
public class Principal {

    private Nodo head;
    private Nodo cola;

    public Principal() {
        this.head = null;
        this.cola = null;
    }

    public void add_Inicio(int valor) {
        Nodo nuevo = new Nodo(valor);
```

```
    if (head == null) {
        head = nuevo;
        cola = nuevo;
    } else {
        nuevo.sig = head;
        head.act = nuevo;
        head = nuevo;
    }
}

public void add_Final(int valor) {
    Nodo nuevo = new Nodo(valor);

    if (head == null) {
        head = nuevo;
        cola = nuevo;
    } else {
        nuevo.act = cola;
        cola.sig = nuevo;
        cola = nuevo;
    }
}

public void add_Orden(int valor) {
    Nodo nuevo = new Nodo(valor);
    Nodo actual;

    if (this.head == null) {
        this.head = nuevo;
        this.cola = nuevo;
    } else if (this.head.valor >= nuevo.valor) {

        nuevo.sig = this.head;
        this.head.act = nuevo;
        this.head = nuevo;
    } else {
        actual = this.head;
        while (actual.sig != null && actual.sig.valor < nuevo.valor) {
            actual = actual.sig;
        }
        nuevo.sig = actual.sig;
        if (actual.sig != null) {
            nuevo.sig.act = nuevo;
        } else {
            this.cola = nuevo;
        }
        actual.sig = nuevo;
        nuevo.act = actual;
    }
}

public void eliminar(int valor) {
    Nodo actual = this.head;
    if (this.head.valor == valor) {
        if (this.head.sig == null) {
            this.head = this.cola = null;
        } else {
            this.head = this.head.sig;
            this.head.act = null;
        }
    } else if (this.cola.valor == valor) {
        this.cola = this.cola.act;
        this.cola.sig = null;
    } else {
        while (actual.sig != this.cola) {
            if (actual.sig.valor == valor) {
                actual.sig = actual.sig.sig;
                actual.sig.act = actual;
            }
```

```
            actual = actual.sig;
        }
    }
}

public void imprimir_Cabeza() {
    Nodo actual = this.head;
    System.out.print("NULL");
    while (actual != null) {
        System.out.print("<- [" + actual.valor + "] ->");
        actual = actual.sig;
    }
    System.out.println("NULL");
}

public void imprimir_Cola() {
    Nodo actual = this.cola;
    System.out.print("NULL");
    while (actual != null) {
        System.out.print("<- [" + actual.valor + "] ->");
        actual = actual.act;
    }
    System.out.println("NULL");
}




public static void main(String[] args) {
    Scanner scan = new Scanner(System.in);
    int op, elem, num;
    Principal miLista = new Principal();
    do{
        System.out.printf("1.Ingresar un nuevo Nodo al inicio"
                + "\n2.Ingresar un nuevo Nodo al final"
                + "\n3.Insertar Nodo y obtener lista ordenada"
                + "\n4.Eliminar Nodo"
                + "\n5.Presentar desde el inicio"
                + "\n6.Presentar desde el final\n0.Salir\n");
        op = scan.nextInt();

        switch (op) {
            case 1:
                System.out.println("ELemento:");
                elem = scan.nextInt();
                miLista.add_Inicio(elem);
                break;
            case 2:
                System.out.println("Elemento:");
                elem = scan.nextInt();
                miLista.add_Final(elem);
                break;
            case 3:
                System.out.println("Elemento:");
                elem = scan.nextInt();
                miLista.add_Orden(elem);
                break;
            case 4:
                System.out.println("Ingrese el elemento a eliminar:");
                elem = scan.nextInt();
                miLista.eliminar(elem);
                break;
            case 5:
                System.out.println("Presentacion desde la CABEZA -> COLA");
                miLista.imprimir_Cabeza();
                break;
            case 6:
                System.out.println("Presentacion desde la COLA -> CABEZA");
                miLista.imprimir_Cola();
```

```
                    break;

                default: break;
            }
        } while (op != 0);


    }

}
```

## Salida del programa

```
run:
1.Ingresar un nuevo Nodo al inicio
2.Ingresar un nuevo Nodo al final
3.Insertar Nodo y obtener lista ordenada
4.Eliminar Nodo
5.Presentar desde el inicio
6.Presentar desde el final
0.Salir
1
ELemento:
1
1.Ingresar un nuevo Nodo al inicio
2.Ingresar un nuevo Nodo al final
3.Insertar Nodo y obtener lista ordenada
4.Eliminar Nodo
5.Presentar desde el inicio
6.Presentar desde el final
0.Salir
2
Elemento:
2
1.Ingresar un nuevo Nodo al inicio
2.Ingresar un nuevo Nodo al final
3.Insertar Nodo y obtener lista ordenada
4.Eliminar Nodo
5.Presentar desde el inicio
6.Presentar desde el final
0.Salir
1
ELemento:
0
1.Ingresar un nuevo Nodo al inicio
2.Ingresar un nuevo Nodo al final
3.Insertar Nodo y obtener lista ordenada
4.Eliminar Nodo
5.Presentar desde el inicio
6.Presentar desde el final
0.Salir
5
Presentacion desde la CABEZA -> COLA
NULL<- [0] -><- [1] -><- [2] ->NULL
1.Ingresar un nuevo Nodo al inicio
2.Ingresar un nuevo Nodo al final
3.Insertar Nodo y obtener lista ordenada
4.Eliminar Nodo
5.Presentar desde el inicio
6.Presentar desde el final
0.Salir
2
Elemento:
8
1.Ingresar un nuevo Nodo al inicio
2.Ingresar un nuevo Nodo al final
3.Insertar Nodo y obtener lista ordenada
4.Eliminar Nodo
```

```
5.Presentar desde el inicio
6.Presentar desde el final
0.Salir
3
Elemento:
9
1.Ingresar un nuevo Nodo al inicio
2.Ingresar un nuevo Nodo al final
3.Insertar Nodo y obtener lista ordenada
4.Eliminar Nodo
5.Presentar desde el inicio
6.Presentar desde el final
0.Salir
6
Presentaicon desde la COLA -> CABEZA
NULL<- [9] -><- [8] -><- [2] -><- [1] -><- [0] ->NULL
1.Ingresar un nuevo Nodo al inicio
2.Ingresar un nuevo Nodo al final
3.Insertar Nodo y obtener lista ordenada
4.Eliminar Nodo
5.Presentar desde el inicio
6.Presentar desde el final
0.Salir
4
Ingrese el elemento a eliminar:
0
1.Ingresar un nuevo Nodo al inicio
2.Ingresar un nuevo Nodo al final
3.Insertar Nodo y obtener lista ordenada
4.Eliminar Nodo
5.Presentar desde el inicio
6.Presentar desde el final
0.Salir
5
Presentacion desde la CABEZA -> COLA
NULL<- [1] -><- [2] -><- [8] -><- [9] ->NULL
1.Ingresar un nuevo Nodo al inicio
2.Ingresar un nuevo Nodo al final
3.Insertar Nodo y obtener lista ordenada
4.Eliminar Nodo
5.Presentar desde el inicio
6.Presentar desde el final
0.Salir
6
Presentacion desde la COLA -> CABEZA
NULL<- [9] -><- [8] -><- [2] -><- [1] ->NULL
1.Ingresar un nuevo Nodo al inicio
2.Ingresar un nuevo Nodo al final
3.Insertar Nodo y obtener lista ordenada
4.Eliminar Nodo
5.Presentar desde el inicio
6.Presentar desde el final
0.Salir
0
BUILD SUCCESSFUL (total time: 2 minutes 22 seconds)
```