# TALLER: LISTAS DOBLEMENTE ENLAZADAS

**ESTEBAN RUIZ VARGAS**

**UNICIENCIA**

**IS0401 - ESTRUCTURA DE DATOS [Gr.2] 20231 - P5061**

**HECTOR FABIO SOTO DURAN**

**27/03/2022**

**Contenido:**

Temas y puntos por resolver

1. Comentario del código propuesto
2. Plantear una clase para administrar una lista genérica doblemente encadenada implementando los siguientes métodos:
    - **2.1.** Insertar un nodo al principio de la lista.
    - **2.2.** Insertar un nodo al final de la lista.
    - **2.3.** Insertar un nodo en la segunda posición. Si la lista está vacía no se inserta el nodo.
    - **2.4.** Insertar un nodo en el ante última posición.
    - **2.5.** Borrar el primer nodo.
    - **2.6.** Borrar el segundo nodo.
    - **2.7.** Borrar el último nodo.
    - **2.8.** Borrar el nodo con información mayor.

***RESPUESTAS***

- **1.** Comentario del código propuesto.

```
public class ListaGenerica {

    // Clase interna Nodo para representar los elementos de la lista
    class Nodo {
        int info; // Valor del nodo
        Nodo ant, sig; // Punteros al nodo anterior y siguiente
    }

    private Nodo raiz; // Puntero al primer nodo de la lista

    // Constructor de la clase ListaGenerica
    public ListaGenerica () {
        raiz = null;
    }

    // Método para insertar un nuevo nodo con el valor x en la posición pos
    void insertar (int pos, int x) {
        // Verifica si la posición es válida
        if (pos <= cantidad () + 1)    {
            Nodo nuevo  = new Nodo (); // Crea un nuevo nodo
            nuevo.info  = x; // Asigna el valor al nodo
            // Caso: insertar al principio de la lista
            if (pos == 1) {
                nuevo.sig  = raiz;
                if (raiz != null)
                    raiz.ant  = nuevo;
                raiz = nuevo;
```

```
        } else {
            // Caso: insertar al final de la lista
            if (pos == cantidad () + 1) {
                Nodo reco = raiz;
                // Encuentra el último nodo
                while (reco.sig != null) {
                    reco = reco.sig;
                }
                // Inserta el nuevo nodo al final
                reco.sig = nuevo;
                nuevo.ant  = reco;
                nuevo.sig  = null;
            } else {
                // Caso: insertar en una posición intermedia
                Nodo reco = raiz;
                // Encuentra el nodo en la posición pos - 2
                for (int f = 1 ; f <= pos - 2 ; f++)
                    reco = reco.sig;
                Nodo siguiente = reco.sig;
                // Inserta el nuevo nodo entre reco y siguiente
                reco.sig = nuevo;
                nuevo.ant  = reco;
                nuevo.sig  = siguiente;
                siguiente.ant = nuevo;
            }
        }
    }
}
```

```
// Método para extraer el nodo en la posición pos y devolver su valor
public int extraer (int pos) {
    // Verifica si la posición es válida
    if (pos <= cantidad ())    {
        int informacion;
        // Caso: extraer el primer nodo
        if (pos == 1) {
            informacion = raiz.info;
            raiz = raiz.sig;
            if (raiz != null)
                raiz.ant = null;
        } else {
            // Caso: extraer un nodo en otra posición
            Nodo reco;
            reco = raiz;
            // Encuentra el nodo en la posición pos - 2
            for (int f = 1 ; f <= pos - 2 ; f++)
                reco = reco.sig;
            Nodo prox = reco.sig;
            // Extrae el nodo y actualiza los punteros
            reco.sig = prox.sig;
            Nodo siguiente = prox.sig;
            if (siguiente != null)
                siguiente.ant = reco;
            informacion = prox.info;
        }
        return informacion;
    } else {
        return Integer.MAX_VALUE;
    }
```

```
}

// Método para borrar el nodo en la posición pos
public void borrar (int pos) {
    // Verifica si la posición es válida
    if (pos <= cantidad ()) {
        // Caso: borrar el primer nodo
        if (pos == 1) {
            raiz = raiz.sig;
            if (raiz != null)
                raiz.ant = null;
        } else {
            // Caso: borrar un nodo en otra posición
            Nodo reco;
            reco = raiz;
            // Encuentra el nodo en la posición pos - 2
            for (int f = 1 ; f <= pos - 2 ; f++)
                reco = reco.sig;
            Nodo prox = reco.sig;
            prox = prox.sig;
            // Borra el nodo y actualiza los punteros
            reco.sig = prox;
            if (prox != null)
                prox.ant = reco;
        }
    }
}

// Método para intercambiar los valores de los nodos en las posiciones pos1 y pos2
public void intercambiar (int pos1, int pos2) {
```

```
    // Verifica si las posiciones son válidas
    if (pos1 <= cantidad () && pos2 <= cantidad ())    {
        Nodo reco1 = raiz;
        // Encuentra el nodo en la posición pos1
        for (int f = 1 ; f < pos1 ; f++)
            reco1 = reco1.sig;
        Nodo reco2 = raiz;
        // Encuentra el nodo en la posición pos2
        for (int f = 1 ; f < pos2 ; f++)
            reco2 = reco2.sig;
        // Intercambia los valores de los nodos
        int aux = reco1.info;
        reco1.info = reco2.info;
        reco2.info = aux;
    }
}

// Método para encontrar el valor del mayor elemento en la lista
public int mayor () {
    if (!vacia ()) {
        int may = raiz.info;
        Nodo reco = raiz.sig;
        while (reco != null) {
            if (reco.info > may)
                may = reco.info;
            reco = reco.sig;
        }
        return may;
    }
    else
```

```
        return Integer.MAX_VALUE;
}

// Método para encontrar la posición del mayor elemento en la lista
public int posMayor() {
    if (!vacia ())    {
        int may = raiz.info;
        int x = 1;
        int pos = x;
        Nodo reco = raiz.sig;
        while (reco != null){
            if (reco.info > may) {
                may = reco.info;
                pos = x;
            }
            reco = reco.sig;
            x++;
        }
        return pos;
    }
    else
        return Integer.MAX_VALUE;
}

// Método para contar la cantidad de elementos en la lista
public int cantidad ()
{
    int cant = 0;
    Nodo reco = raiz;
    while (reco != null) {
```

```
        reco = reco.sig;
        cant++;
    }
    return cant;
}

// Método para verificar si la lista está ordenada de menor a mayor
public boolean ordenada() {
    if (cantidad() > 1) {
        Nodo reco1 = raiz;
        Nodo reco2 = raiz.sig;
        while (reco2 != null) {
            if (reco2.info < reco1.info) {
                return false;
            }
            reco2 = reco2.sig;
            reco1 = reco1.sig;
        }
    }
    return true;
}

// Método para verificar si existe un elemento con el valor x en la lista
public boolean existe(int x) {
    Nodo reco = raiz;
    while (reco != null) {
        if (reco.info == x)
            return true;
        reco = reco.sig;
    }
```

```
    return false;
}

// Método para verificar si la lista está vacía
public boolean vacia ()
{
    if (raiz == null)
        return true;
    else
        return false;
}

// Método para imprimir los elementos de la lista
public void imprimir ()
{
    Nodo reco = raiz;
    while (reco != null) {
        System.out.print  (reco.info + "-");
        reco = reco.sig;
    }
    System.out.println();
}

// Método principal para probar la clase ListaGenerica
public static void main(String[] ar) {
    ListaGenerica lg = new ListaGenerica();
    lg.insertar (1, 10);
    lg.insertar (2, 20);
    lg.insertar (3, 30);
    lg.insertar (2, 15);
```

```
        lg.insertar (1, 115);
        lg.imprimir ();
        System.out.println ("Luego de Borrar el primero");
        lg.borrar (1);
        lg.imprimir ();
        System.out.println ("Luego de Extraer el segundo");
        lg.extraer (2);
        lg.imprimir ();
        System.out.println ("Luego de Intercambiar el primero con el tercero");
        lg.intercambiar (1, 3);
        lg.imprimir ();
        if (lg.existe(10))
           System.out.println("Se encuentra el 20 en la lista");
        else
           System.out.println("No se encuentra el 20 en la lista");
        System.out.println("La posicion del mayor es:" + lg.posMayor());
        if (lg.ordenada())
           System.out.println("La lista esta ordenada de menor a mayor");
        else
           System.out.println("La lista no esta ordenada de menor a mayor");
    }
}
```

- **2.** Plantear una clase para administrar una lista genérica doblemente encadenada implementando los siguientes métodos.

```
public class ListaGenerica {

    class Nodo {
        int info;
        Nodo ant, sig;
```

```
    }

    private Nodo raiz;

    public ListaGenerica() {
        raiz = null;
    }
```

**- 2.1. RTA:**

Insertar un nodo al principio de la lista

```
    public void insertarAlPrincipio(int x) {
        Nodo nuevo = new Nodo();
        nuevo.info = x;
        nuevo.sig = raiz;
        if (raiz != null) {
            raiz.ant = nuevo;
        }
        raiz = nuevo;
    }
```

**- 2.2. RTA:**

Insertar un nodo al final de la lista

```
    public void insertarAlFinal(int x) {
        Nodo nuevo = new Nodo();
        nuevo.info = x;
        if (raiz == null) {
            raiz = nuevo;
        } else {
            Nodo reco = raiz;
            while (reco.sig != null) {
```

```
            reco = reco.sig;
        }
        reco.sig = nuevo;
        nuevo.ant  = reco;
    }
}
```

**- 2.3. RTA:**

Insertar un nodo en la segunda posición. Si la lista está vacía no se inserta el nodo.

```
public void insertarEnSegundaPosicion(int x) {
    if (raiz != null && raiz.sig != null) {
        Nodo nuevo = new Nodo();
        nuevo.info  = x;
        nuevo.sig  = raiz.sig;
        nuevo.ant  = raiz;
        raiz.sig.ant = nuevo;
        raiz.sig = nuevo;
    }
}
```

**- 2.4. RTA:**

Insertar un nodo en la ante última posición

```
public void insertarEnAnteUltimaPosicion(int x) {
    if (raiz != null) {
        Nodo nuevo = new Nodo();
        nuevo.info  = x;
        Nodo reco = raiz;
        while (reco.sig != null) {
            reco = reco.sig;
        }
```

```
            if (reco.ant != null) {
                nuevo.sig = reco;
                nuevo.ant = reco.ant;
                reco.ant.sig = nuevo;
                reco.ant = nuevo;
            } else {
                raiz.sig = nuevo;
                nuevo.ant = raiz;
            }
        }
    }
```

**- 2.5. RTA:**

Borrar el primer nodo

```
    public void borrarPrimerNodo() {
        if (raiz != null) {
            raiz = raiz.sig;
            if (raiz != null) {
                raiz.ant = null;
            }
        }
    }
```

**- 2.6. RTA:**

Borrar el segundo nodo

```
    public void borrarSegundoNodo() {
        if (raiz != null && raiz.sig != null) {
            raiz.sig = raiz.sig.sig;
            if (raiz.sig != null) {
                raiz.sig.ant = raiz;
```

```
        }
    }
}
```

**- 2.7. RTA:**

Borrar el último nodo

```
public void borrarUltimoNodo() {
    if (raiz != null) {
        Nodo reco = raiz;
        while (reco.sig != null) {
            reco = reco.sig;
        }
        if (reco.ant != null) {
            reco.ant.sig = null;
        } else {
            raiz = null;
        }
    }
}
```

**- 2.8. RTA:**

Borrar el nodo con información mayor

```
public void borrarNodoMayor() {
    if (raiz != null) {
        Nodo reco = raiz;
        Nodo mayor = raiz;

        // Encontrar el nodo con información mayor
        while (reco != null
        {
```

```
            if (reco.info > mayor.info) {
                mayor = reco;
            }
            reco = reco.sig;
        }

        // Borrar el nodo con información mayor
        if (mayor.ant == null) { // Si el nodo mayor es el primero
            raiz = mayor.sig;
            if (raiz != null) {
                raiz.ant = null;
            }
        } else if (mayor.sig == null) { // Si el nodo mayor es el último
            mayor.ant.sig = null;
        } else { // Si el nodo mayor está en el medio
            mayor.ant.sig = mayor.sig;
            mayor.sig.ant = mayor.ant;
        }
    }
}

// Método para imprimir los elementos de la lista
public void imprimir() {
    Nodo reco = raiz;
    while (reco != null) {
        System.out.print(reco.info + "-");
        reco = reco.sig;
    }
    System.out.println();
}
```

```
    public static void main(String[] ar) {
        ListaGenerica lg = new ListaGenerica();
        lg.insertarAlPrincipio(10);
        lg.insertarAlPrincipio(20);
        lg.insertarAlFinal(30);
        lg.insertarEnSegundaPosicion(15);
        lg.insertarEnAnteUltimaPosicion(25);
        lg.imprimir();
        lg.borrarPrimerNodo();
        lg.imprimir();
        lg.borrarSegundoNodo();
        lg.imprimir();
        lg.borrarUltimoNodo();
        lg.imprimir();
        lg.borrarNodoMayor();
        lg.imprimir();
    }
}
```