

LISTAS ENLAZADAS

Ing. Jaider de la Rosa Bertel

Jaider.delarosa@cecar.edu.co

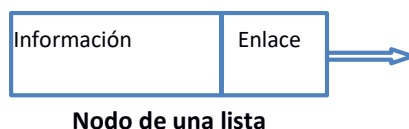
DESARROLLO

CONCEPTOS DE LISTAS ENLAZADAS:

Existen estructuras de datos dinámicas que se utilizan para almacenar datos que están cambiando constantemente. A diferencia de los vectores, las estructuras de datos dinámicas se expanden y se contraen haciéndolas más flexibles a la hora de añadir o eliminar información.

Esto quiere decir que se puede crear un nuevo nodo para insertarlo entre los nodos ya existentes en la lista o eliminar un nodo existente. Entre las estructuras de datos lineales están: Las listas enlazadas, las pilas y colas.

Las listas enlazadas permiten almacenar información en posiciones de memoria que no sean contiguas. Estas listas para almacenar la información, contienen elementos llamados nodos. Estos nodos poseen dos campos, uno para almacenar la información o valor del elemento y otro para el enlace que determina la posición del siguiente elemento o nodo de la lista.



Al insertar o borrar información no es necesario realizar un desplazamiento; para esto los nodos de la lista cuentan con punteros o enlaces que contienen la posición o dirección del otro nodo que le sigue. Por esta razón no es necesario que los elementos de la lista se almacenen en posiciones contiguas.



Cuando en una lista enlazada no hay ningún elemento quiere decir que la lista está vacía, además existe un puntero de cabecera para acceder al primer nodo de la lista y un puntero nulo para determinar el último elemento (nodo) de la lista.

Lo más recomendable y flexible para la creación de un nodo es utilizar un objeto por cada nodo, para ello debe comprender cuatro conceptos fundamentales que son:

- Clase auto-referenciada,
- Nodo,
- Campo de enlace y
- Enlace

Una clase auto-referenciada es una clase con al menos un campo cuyo tipo de referencia es el nombre de la misma clase.

```
public class Nodo  
{  
  
    Object elemento;  
  
    Nodo siguiente;  
  
    //métodos  
  
}
```

OPERACIONES A REALIZAR EN UNA LISTA ENLAZADA:

Cuando se utilizan listas enlazadas podemos realizar las siguientes operaciones:

- Añadir información a la lista insertando un nuevo nodo en un determinado lugar dentro de la lista. Puede ser al inicio de la lista, entre dos nodos existentes y al final de la lista.
- Eliminar un nodo específico dentro de la lista que contenga información.
- Recuperar la información almacenada en un nodo específico o realizar operaciones sobre la información de los nodos.

IMPLEMENTACIÓN DE LISTAS ENLAZADAS EN JAVA:

- **Ejercicio propuesto implementado en Java**

A continuación, se implementa un ejercicio que permite almacenar en una lista enlazada la información de los estudiantes de un curso, correspondiente a su código o identificación, el nombre y las tres notas que se le toman a un estudiante durante un determinado periodo; también se calcula la nota promedio del semestre.

El ejercicio en su implementación permite al usuario las siguientes opciones:

- Agregar nodos a la lista.
- Mostar la información de los nodos almacenados.
- Determinar el número de nodos almacenados en la lista.
- Buscar la información de in estudiante en los nodos.
- Eliminar un determinado nodo de la lista.
- Realizar operaciones sobre la información de la lista.

Para la solución del ejercicio se requiere de la implementación de tres ficheros, en donde se implementan las clases que permitirán almacenar la información de los estudiantes en la lista y facilitarán realizar operaciones sobre los nodos.

El código anterior es una clase auto-referenciada porque su campo siguiente tiene el tipo Nodo.

El nodo es un objeto creado a partir de una clase auto-referenciada.

El campo de enlace es la variable de instancia que contiene el tipo que corresponde con el nombre de la clase (para el caso anterior variable siguiente).

El enlace es el contenido del campo de enlace, que hace referencia (guarda la dirección) a otro nodo.

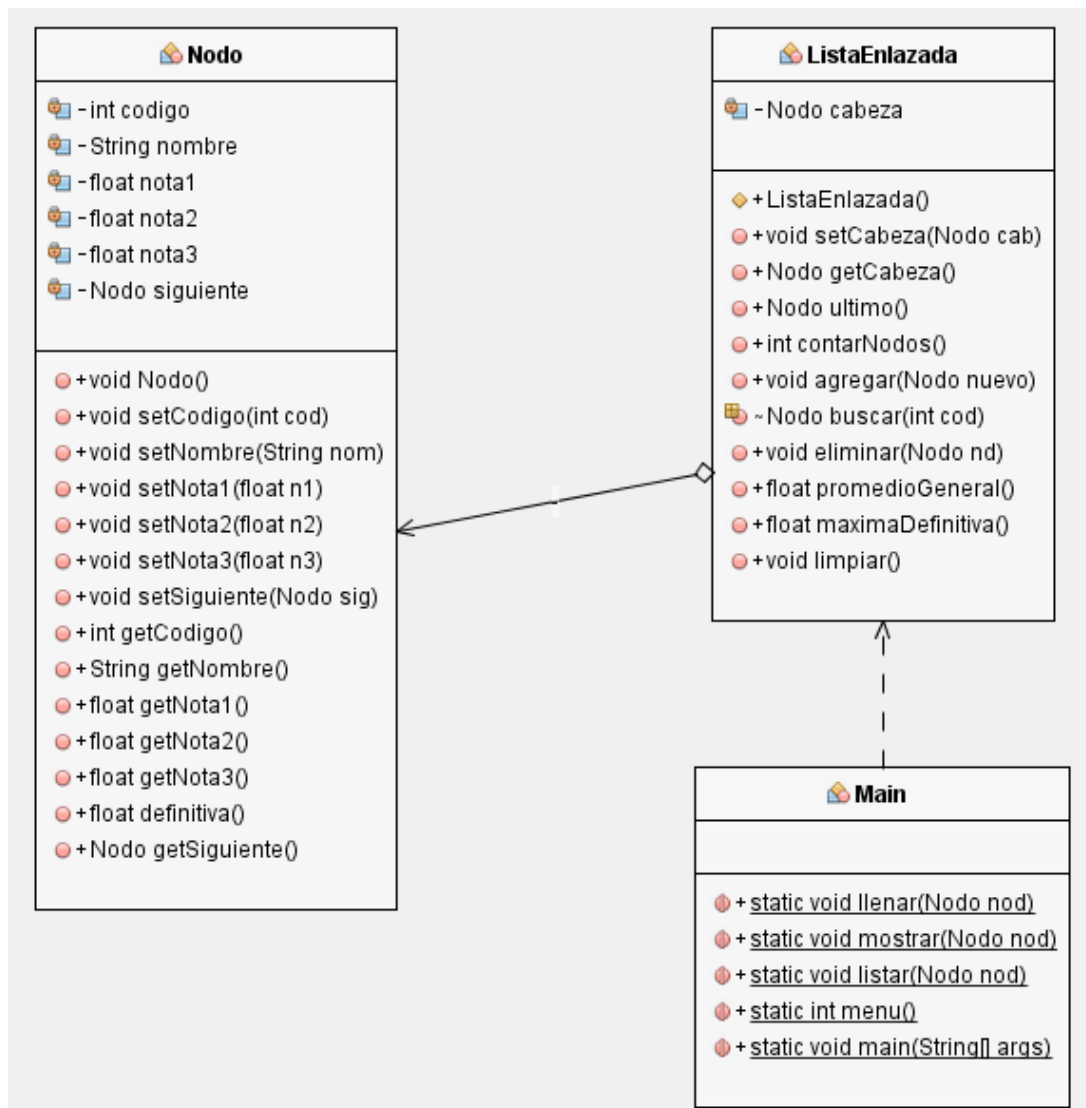
Las operaciones que se pueden hacer con una lista son:

- Inserción de un elemento.
- Borrado de un elemento.
- Recorrido de la lista.
- Búsqueda de un elemento.

Las listas enlazadas se dividen en:

- Listas enlazadas simples (con una sola dirección) y
- Listas enlazadas dobles (con dos direcciones).

- **Diseño de clases UML de la solución para una lista enlazada simple**



- **Implementación de la clase Nodo en el fichero Nodo.java**

En el fichero **Nodo.java** se declara la clase **Nodo**, que tendrá toda la información requerida del estudiante y su respectivo enlace o apuntador.

```

public class Nodo {
    private int codigo;
    private String nombre;
    private float nota1;
    private float nota2;
    private float nota3;
    private Nodo siguiente;
}
    
```

```
public void Nodo(){
    codigo = 0; nombre = "";
    nota1 = 0;
    nota2 = 0;
    nota3 = 0; siguiente =
    null;
}
public void setCodigo(int cod){
    codigo = cod;
}
public void setNombre(String nom){
    nombre = nom;
}
public void setNota1(float n1){
    nota1 = n1;
}
public void setNota2(float n2){
    nota2 = n2;
}
public void setNota3(float n3){
    nota3 = n3;
}
public void setSiguiente(Nodo sig){
    siguiente = sig;
}
public int getCodigo(){ return
    codigo;
}
public String getNombre(){
    return nombre;
}
public float getNota1(){
    return nota1;
}
public float getNota2(){
    return nota2;
}
public float getNota3(){ return
    nota3;
}
public float definitiva(){
    return (getNota1()+getNota2()+getNota3())/3;
}
public Nodo getSiguiente(){
    return siguiente;
}
}
```

- **Implementación de la clase ListaEnlazada en el fichero ListaEnlazada.java:**

El fichero **ListaEnlazada.java** es para declarar la clase ListaEnlazada, en esta clase se declaran todos los métodos que permiten crear la lista enlazada y realizar las operaciones básicas sobre la misma, como: agregar nodos a la lista, buscar información, eliminar nodos y realizar operaciones.

```
public class ListaEnlazada {
    private Nodo cabeza; //Se declara el atributo de la clase, cabeza primer nodo de la lista.
    //Método constructor de la clase ListaEnlazada.
    public ListaEnlazada(){
        cabeza = null; //Inicialmente la lista está vacía, la cabeza apunta a nulo.
    }

    //Implementación del método que asigna el primer nodo de la lista (nodo cabeza).
    public void setCabeza(Nodo cab){
        cabeza = cab;
    }
    //Implementación del método para obtener el primer nodo de la lista (nodo cabeza).
    public Nodo getCabeza(){
        return cabeza;
    }

    //Método que recorre la lista y devuelve el último nodo de la lista.
    public Nodo ultimo(){
        Nodo temp = cabeza;
        while(temp != null){
            //Si temp apunta a nulo el nodo temporal (temp) es el último de la lista.
            if(temp.getSiguiente() == null){
                break; //Se rompe el ciclo.
            }else{
                //De lo contrario se pasa al siguiente nodo de la lista.
                temp = temp.getSiguiente();
            }
        }
        return temp;
    }

    //Método que recorre la lista para contar cuantos nodos hay almacenados.
    public int contarNodos(){
        int contador = 0;
        Nodo temp = cabeza;
        while(temp != null){
            contador++;
            temp = temp.getSiguiente();
        }
        return contador;
    }

    //Método para agregar un nuevo nodo en la lista, por el final.
    public void agregar(Nodo nuevo){
        if(cabeza == null){ //La lista está vacía, no hay nodos.
            setCabeza(nuevo); //Se asigna el primer nodo de la lista.
        }else{
            ultimo().setSiguiente(nuevo); //El ultimo nodo apuntara al nuevo nodo que se agregó.
        }
    }
}
```

```
}

//Implementación del método que busca un nodo dentro de la lista, pasando como parámetro de
//búsqueda el código del estudiante.
Nodo buscar(int cod){
    Nodo temp = cabeza;
    while(temp != null){
        if(temp.getCodigo() == cod){ break;
        }else{
            temp = temp.getSiguiente();
        }
    }
    return temp;
}

//Implementación del método que elimina el nodo de la lista pasado como parámetro (nd) el nodo
//que se quiere eliminar.
public void eliminar(Nodo nd){
    Nodo anterior;
    if(nd == cabeza){ //Si el nodo a eliminar en la lista es el primero entonces.
        cabeza = cabeza.getSiguiente();
    }else{ //De lo contrario, se busca el nodo anterior al que se quiere eliminar (nd).
        anterior = cabeza;
        while(anterior.getSiguiente() != nd){
            anterior = anterior.getSiguiente();
        }
        //El siguiente de anterior será el nodo que le sigue al que se va a eliminar (nd).
        anterior.setSiguiente(nd.getSiguiente());
    }
    nd.setSiguiente(null); //El enlace del nodo a borrar, apuntara su siguiente a nulo.
}

//Implementación del método que calcula el promedio de todas las notas definitivas almacenadas
//en la lista.
public float promedioGeneral(){
    int cantidad = 0;
    float suma = 0;
    Nodo temp = cabeza;
    while(temp != null){
        cantidad++;
        suma = suma + temp.definitiva();
        temp = temp.getSiguiente();
    }
    if(cantidad > 0)
        return suma/cantidad; else
        return 0;
}

//Implementación del método que devuelve la nota definitiva más alta de las almacenadas. public
float maximaDefinitiva(){
    float def, max; max = 0;
    Nodo temp = cabeza;
    while(temp != null){
        def = temp.definitiva();
```

```
        if(def > max){
            max = def;
        }
        temp = temp.getSiguiente();
    }
    return max;
}
public void limpiar(){
    while(cabeza != null){
        eliminar(cabeza);
    }
}
}
```

- **Implementación de la clase Main en el fichero Main.java:**

Finalmente se implementa la clase Main en el fichero **Main.java**, que permite capturar la información que se almacenara en la lista y mostrar los resultados de las operaciones realizadas.

```
public class Main {
//Se declaran los siguientes métodos, que serán llamados dentro del método static void main:

//Método para asignar los valores a los atributos de la clase Nodo, aquí se crea una instancia por
//parámetro de la clase Nodo llamada nod.
public static void llenar(Nodo nod){
    int cod = Integer.parseInt(JOptionPane.showInputDialog("Digite CODIGO del Estudiante: "));
    nod.setCodigo(cod);
    String nom = JOptionPane.showInputDialog("Digite NOMBRE del Estudiante: ");
    nod.setNombre(nom);
    float n1 = Float.parseFloat(JOptionPane.showInputDialog("Digite La NOTA 1:"));
    nod.setNota1(n1);
    float n2 = Float.parseFloat(JOptionPane.showInputDialog("Digite La NOTA 2:"));
    nod.setNota2(n2);
    float n3 = Float.parseFloat(JOptionPane.showInputDialog("Digite La NOTA 3:"));
    nod.setNota3(n3);
}

//Método para obtener y mostrar los valores asignados a los atributos de la clase Nodo. public static
void mostrar(Nodo nod){
    String datosNodo = "";
    datosNodo = datosNodo+String.valueOf(""+CODIGO: "+nod.getCodigo()+"\n"+NOMBRE:
"+nod.getNombre()+"\n"+
    "NOTA 1: "+nod.getNota1()+"\n"+NOTA 2: "+nod.getNota2()+"\n"+NOTA 3:
"+nod.getNota3()+"\n"+
    "Definitiva: "+nod.definitiva()+"\n\n");
    JOptionPane.showMessageDialog(null, "===== INFORMACIÓN DE LOS NODOS DE LA LISTA
===== \n"+ datosNodo);
}

//Método para listar cada uno de los nodos de la lista y visualizarlos en pantalla. public static
void listar(Nodo nod){
    Nodo temp = nod;
```

```
while(temp != null){
    mostrar(temp);
    temp = temp.getSiguiente();
}
}

//Método para visualizar el menú de opciones y asignar la opción seleccionada. public static int menu(){
int opcion = 0; do{
opcion = Integer.parseInt(JOptionPane.showInputDialog("===== SELECCIONE UNA OPCIÓN DEL
MENÚ ===== \n"+
"1. Agregar un Nodo a la Lista \n"+"2. Mostrar Nodos de la Lista \n"+
"3. Cantidad de Nodos de la Lista \n"+"4. Buscar la Información de un estudiante \n"+ "5. Eliminar Nodo de la
Lista \n"+"6. Informe: Promedio General y Máxima Nota \n"+
"7. Borrar toda la Lista \n"+"8. Salir"+" \n \n Seleccione una opción del 1 al 8: ");
}while(opcion <= 0 || opcion > 8); return opcion;
}

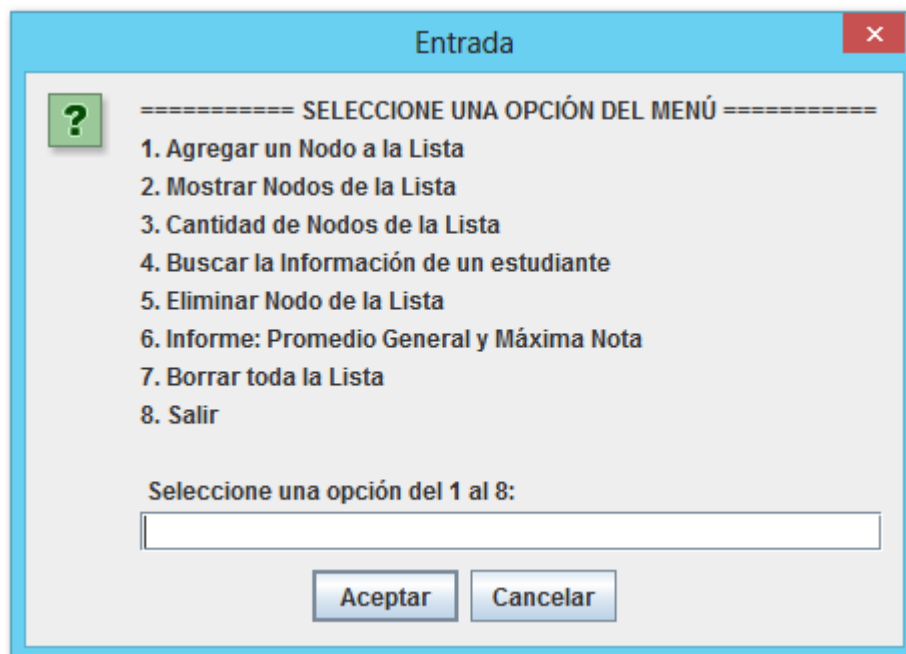
public static void main(String[] args) {
ListaEnlazada lis = new ListaEnlazada(); //Se crea el objeto lis de la clase ListaEnlazada. int opcion, cod;
Nodo aux; do{
opcion = menu(); switch(opcion) { case 1:
aux = new Nodo(); //Cuando se agrega un nodo se crea un nuevo objeto de la clase nodo. llenar(aux);
lis.agregar(aux); break;
case 2:
if(lis.getCabeza() != null){ listar(lis.getCabeza());
}else{
JOptionPane.showMessageDialog(null, "La Lista Está Vacía....");
}

break; case 3:
JOptionPane.showMessageDialog(null, "===== NÚMERO DE NODOS DE LA LISTA
===== \n"+lis.contarNodos()+
" Nodos"); break;
case 4:
cod = Integer.parseInt(JOptionPane.showInputDialog("Digite CODIGO del Estudiante a Buscar:
"));

aux = lis.buscar(cod); if(aux != null){
mostrar(aux);
}else{
JOptionPane.showMessageDialog(null, "La información del estudiante No se encuentra en la
lista");
}
break;
```

```
case 5:
    cod = Integer.parseInt(JOptionPane.showInputDialog("Digite CODIGO del Estudiante a Eliminar:
"));
    aux = lis.buscar(cod);
    if(aux != null){
        lis.eliminar(aux);
        JOptionPane.showMessageDialog(null, "La información fue eliminada correctamente. .. ");
    }else{
        JOptionPane.showMessageDialog(null, "El código del estudiante NO EXISTE en la Lista");
    }
    break;
case 6:
    JOptionPane.showMessageDialog(null, "===== INFORME PROMEDIO GENERAL Y
MÁXIMA NOTA ===== \n\n"+
        "Promedio General: "+lis.promedioGeneral()+"\n y Máxima Nota: "+lis.maximaDefinitiva());
    break;
case 7:
    lis.limpiar();
    JOptionPane.showMessageDialog(null, "La Lista Está Vacía. .. ");
    break;
case 8:
    break;
}
}while(opcion != 8);
}
```

- **Menú de opciones de la aplicación en ejecución:**



EJERCICIOS/ACTIVIDADES

- Para complementar el ejercicio propuesto anterior, implementar los métodos que permitan agregar nodos por la cabeza y entre dos nodos de la lista. Incluir las dos nuevas opciones en el menú de la clase Main para realizar las pruebas respectivas.
- Implementar los métodos necesarios que permitan ordenar los nodos de la lista del ejercicio propuesto, teniendo en cuenta la nota definitiva de los estudiantes, de la mayor nota a la menor.

BIBLIOGRAFÍA

- Zahonero, I., y Joyanes Aguilar, L. (1999). Estructura de Datos - Algoritmos, Abstracción y Objetos. España: McGraw-Hill.
- Allen Weiss, M. (2004). Estructuras de datos en Java. España: Addison Wesley - Pearson. 776 pp.
- Cairó, O. & Guardati, S. (2002). Estructuras de datos (2nd ed.). México: McGraw-Hill.
- Joyanes, L. (2000). Programación en C++: Algoritmos, estructuras de datos y objetos (1st ed.). Madrid: McGraw-Hill.
- Kernighan, B., & Ritchie, D. (1991). El Lenguaje de Programación C (2nd ed.). México: Prentice Hall.
- Stroustrup, B. (2002). El Lenguaje de Programación C++ (1th ed.). Madrid: Pearson Educación.