LISTAS ENLAZADAS DOBLES

Ing. Jaider Reyes Herazo

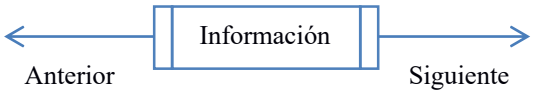
coordinador_empredimientos@uajs.edu.co

DESARROLLO

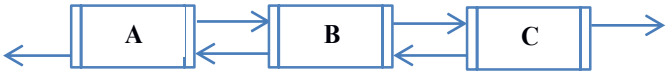
CONCEPTOS DE LISTAS ENLAZADAS DOBLES:

Muchas veces las listas enlazadas simples no soportan la función de recorrer una lista desde el último nodo de la lista hasta la cabeza o inicio de la misma. Esto teniendo en cuenta que la opción de retroceder en las listas no se puede realizar con un solo enlace o referencia (siguiente).

En este caso si el objetivo es recorrer la lista en ambos sentidos, se deberían tener dos referencias o enlaces en cada nodo; es decir, cada nodo además de la información tendrá un enlace al siguiente nodo de la lista y otro al nodo anterior.



Como la implementación de estas listas tienen dos enlaces o referencias en cada nodo, reciben el nombre de doblemente enlazadas. En este sentido se podrán realizar búsquedas y recorridos en ambos sentidos.



OPERACIONES A REALIZAR EN UNA LISTA ENLAZADA DOBLE:

Al igual que en las listas enlazadas simples, en las listas dobles, se poden realizar las siguientes operaciones teniendo en cuenta los dos apuntadores:

- 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**

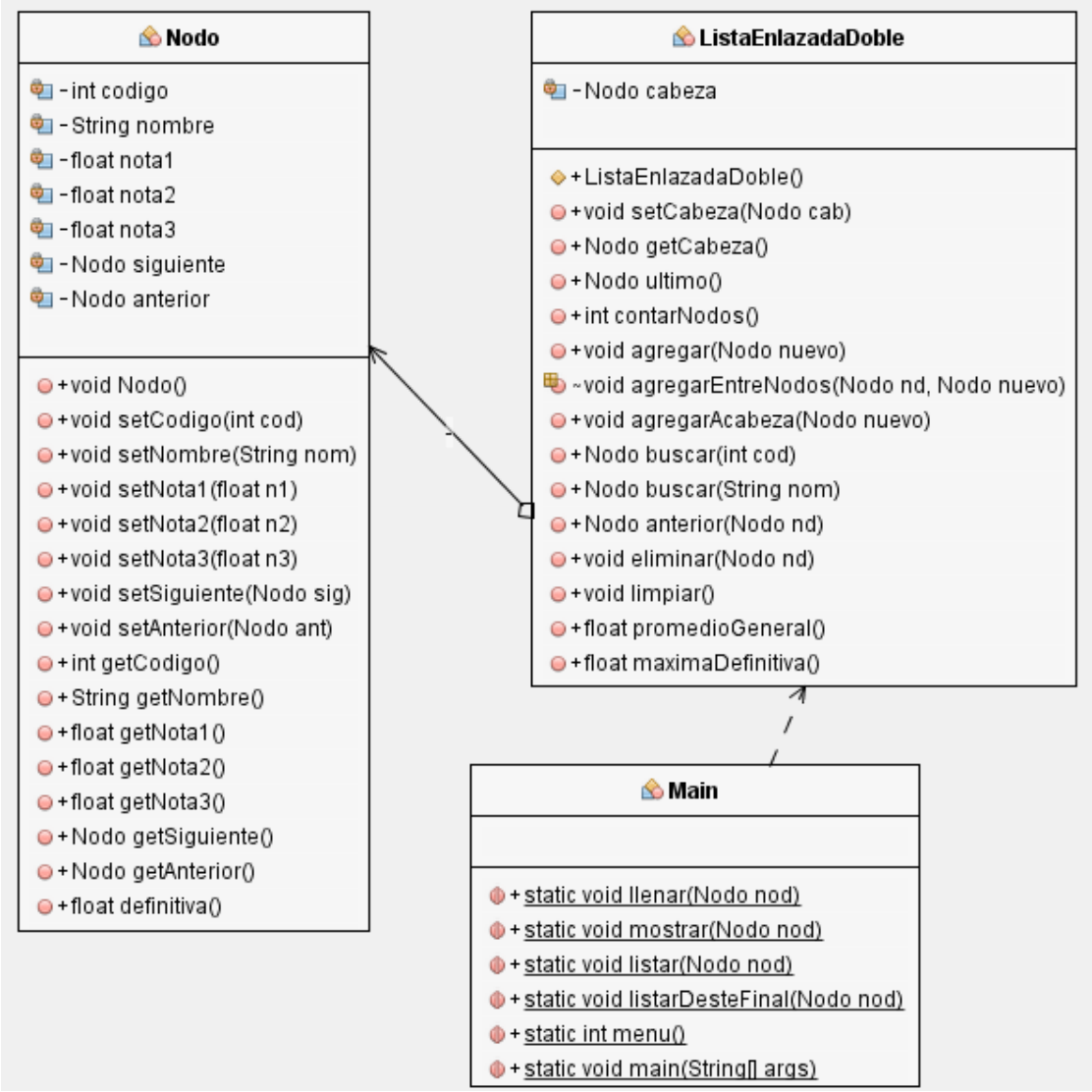
Teniendo en cuenta el ejercicio de la unidad anterior, a continuación, se implementa un ejercicio que permite almacenar en una lista doblemente enlazada la información de los estudiantes de un curso; correspondiente a su código o identificación, el nombre y las tres notas que se le toman a un estudiante 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 doblemente enlazada.
- Mostar la información de los nodos almacenados.
- Determinar el número de nodos almacenados en la lista.
- Buscar la información de un estudiante en ambos sentidos.
- Eliminar un determinado nodo de la lista doblemente enlazada.
- Realizar operaciones sobre la información de la lista en ambos sentidos.

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.

- **Diseño de clases UML de la solución**



- **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 sus dos enlaces o apuntadores (siguiente y anterior).

```
public class Nodo {
    private int codigo;
    private String nombre;
    private float nota1;
    private float nota2;
    private float nota3;
    private Nodo siguiente; //Se declara un apuntador hacia delante del nodo.
    private Nodo anterior; //y otro apuntador hacia atrás del nodo.
    public void Nodo(){
        codigo = 0;
        nombre = "";
```

```
        nota1 = 0;
        nota2 = 0;
        nota3 = 0;
        siguiente = null;
        anterior = 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 void setAnterior(Nodo ant){
        anterior = ant;
    }
    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 Nodo getSiguiente(){
        return siguiente;
    }
    public Nodo getAnterior(){
        return anterior;
    }
      public float definitiva(){
        return (getNota1()+getNota2()+getNota3())/3;
    }
}
```

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

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

```
public class ListaEnlazadaDoble {
   private Nodo cabeza; //Se declara el atributo de la clase, cabeza primer nodo de la lista.
//Método constructor de la clase ListaEnlazadaDoble.
   public ListaEnlazadaDoble(){
      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;
      if(temp != null){
         while(temp.getSiguiente() != null){
            temp = temp.getSiguiente();
         }
      }
      return temp;
   }

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

//Método para agregar un nuevo nodo en la lista, por el final.
   public void agregar(Nodo nuevo){
      Nodo temp = ultimo();
      if(temp != null){
         temp.setSiguiente(nuevo); //El enlace del ultimo nodo apunta al nuevo nodo.
         nuevo.setAnterior(temp); //El enlace anterior del nuevo nodo apunta al último nodo.
         nuevo.setSiguiente(null); //El enlace siguiente del nuevo nodo apunta a nulo.
      }
```

```
        else{ //Esta condición se da, en caso de que la lista este vacía y se agrega el primer nodo.
            nuevo.setAnterior(null);
            cabeza = nuevo;
        }
    }

//Método para agregar nodos entre dos nodos ya existentes en la lista enlazada doble.
    void agregarEntreNodos(Nodo nd, Nodo nuevo){
        nuevo.setSiguiente(nd.getSiguiente());
        if(nd.getSiguiente() != null){
            nd.getSiguiente().setAnterior(nuevo);
        }
        nd.setSiguiente(nuevo);
        nuevo.setAnterior(nd);
    }

//Método para agregar nodos por la cabeza de la lista enlazada doble.
    public void agregarAcabeza(Nodo nuevo){
        if(cabeza == null){
            cabeza=nuevo;
        }else{
            nuevo.setAnterior(null);
            nuevo.setSiguiente(cabeza);
            cabeza.setAnterior(nuevo);
            cabeza=nuevo;
        }
    }

//Busca un nodo en la lista enlazada doble desde la cabeza (inicio) hasta el último nodo de la lista,
//se pasa como parámetro de búsqueda el código del estudiante.
    public Nodo buscar(int cod){
        Nodo temp = cabeza;
        while(temp != null){
            if(temp.getCodigo() == cod){
            break;
            }else{
                temp = temp.getSiguiente();
            }
        }
        return temp;
    }

//Busca un nodo en la lista enlazada doble desde el final (último nodo) hasta el inicio de la lista
//(cabeza), se pasa como parámetro de búsqueda el nombre del estudiante.
    public Nodo buscar(String nom){
        Nodo temp = ultimo();
        while(temp != null){
            if(temp.getNombre().equals(nom)){ //Se compara si las cadenas son iguales.
                break;
            }else{
                temp = temp.getAnterior(); //Se recorre la lista desde el último nodo hasta el primero.
            }
        }
        return temp;
    }
```

```
//Este método devuelve el nodo que se encuentra antes (anterior) de un nodo pasado como
//parámetro. En este caso devuelve el nodo que esta antes del nodo a eliminar.
   public Nodo anterior(Nodo nd){
      Nodo temp = cabeza;
      while(temp != null){
         if(temp.getSiguiente() == nd){
            break;
         }else{
            temp = temp.getSiguiente();
         }
      }
      return temp;
   }

//Método que eliminar un nodo de la lista enlazada doble.
   public void eliminar(Nodo nd){
      Nodo ante;
      if(nd == cabeza){ //En caso de que el nodo a eliminar sea el primer nodo de la lista doble.
         cabeza = cabeza.getSiguiente();
         if(cabeza != null)
            cabeza.setAnterior(null);
         }else{ //De lo contrario, se busca el nodo anterior al que se quiere eliminar (nd).
            ante = anterior(nd);
            ante.setSiguiente(nd.getSiguiente());
            if(nd.getSiguiente()!= null){
               nd.getSiguiente().setAnterior(ante);
            }
         }
         nd.setAnterior(null);
         nd.setSiguiente(null);
   }

   public void limpiar(){
      while(cabeza != null){
         eliminar(cabeza);
      }
      cabeza = null;
   }

//Implementación del método que calcula el promedio de todas las notas definitivas almacenadas
//en la lista doblemente enlazada.
   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,
//recorriendo la lista desde el último nodo hasta el primero.
   public float maximaDefinitiva(){
      float max = 0;
      float def;
      Nodo temp = ultimo();
      while(temp != null){
         def = temp.definitiva();
         if(def > max){
            max = def;
         }
         temp = temp.getAnterior();
      }
      return max;
   }
}
```

- **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 doble y mostrar los resultados de las operaciones realizadas.

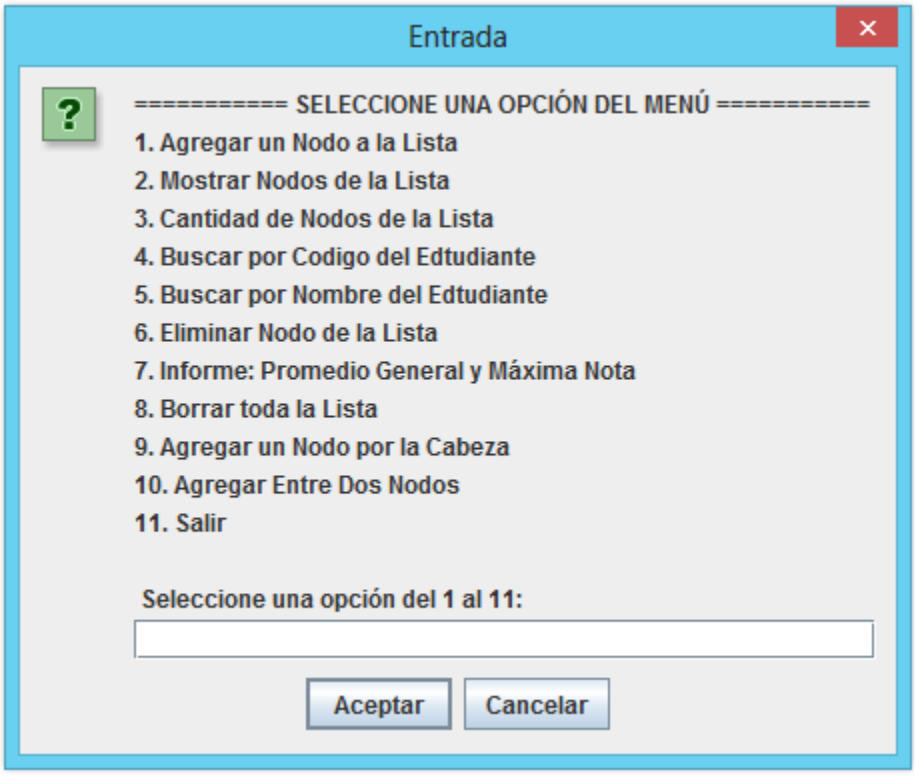
```
public class Main {
//Se declaran los siguientes métodos, que serán llamados dentro del método static void main:
//Método para asignar los valores a los atributos de la clase Nodo.
   public static void llenar(Nodo nod){
      int cod = Integer.parseInt(JOptionPane.showInputDialog("Digite CODIGO 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, "========== INFORMACÓN 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();
      }
   }

//Este es un método alternativo que se puede utilizar para listar la información de los nodos desde
//el final de la lista enlazada doble.
    public static void listarDesteFinal(Nodo nod){
      Nodo temp = nod;
      while(temp != null){
         mostrar(temp);
         temp = temp.getAnterior();
      }
   }
//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 por Código del Estudiante \n"+
               "5. Buscar por Nombre del Estudiante \n"+"6. Eliminar Nodo de la Lista \n"+
                  "7. Informe: Promedio General y Máxima Nota \n"+"8. Borrar toda la Lista \n"+"9. Agregar
un Nodo por la Cabeza \n"+
                  "10. Agregar Entre Dos Nodos \n"+"11. Salir"+"\n \n Seleccione una opción del 1 al 11:"));
         }while(opcion <= 0 || opcion > 11);
         return opcion;
   }

   public static void main(String[] args) {
//Se crea el objeto lis de la clase ListaEnlazadaDoble.
      ListaEnlazadaDoble lis = new ListaEnlazadaDoble();
      int opcion, cod;
      String nom;
      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){
               listarDesteFinal(lis.ultimo()); //Pueden utilizar cualquiera de los dos métodos para listar.
            }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:
                nom = JOptionPane.showInputDialog("Digite NOMBRE del Estudiante a Buscar: ");
                aux = lis.buscar(nom);
                if(aux != null){
                    mostrar(aux);
                }else{
                    JOptionPane.showMessageDialog(null, "La información del estudiante No se encuentra en la
lista");
                }
                break;
            case 6:
                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 7:
                JOptionPane.showMessageDialog(null, "========= INFORME PROMEDIO GENERAL Y
MÁXIMA NOTA ========= \n\n"+
                        "Promedio General: "+lis.promedioGeneral()+"\n y Máxima Nota: "+lis.maximaDefinitiva());
                break;
            case 8:
                lis.limpiar();
                JOptionPane.showMessageDialog(null, "La Lista Está Vacía. ..");
                break;
            case 9:
                aux = new Nodo();
                llenar(aux);
                lis.agregarAcabeza(aux);
                break;
            case 10:
                cod = Integer.parseInt(JOptionPane.showInputDialog("CODIGO del Estudiante Después del
que Quiere Agregar el Nuevo Nodo: "));
                Nodo nd = lis.buscar(cod);
                if(nd != null){
```

```
                aux = new Nodo();
                llenar(aux);
                lis.agregarEntreNodos(nd, aux);
            }else{
                JOptionPane.showMessageDialog(null, "El código del estudiante NO EXISTE en la Lista");
            }
            break;
        case 11:
            break;
        }
    }while(opcion != 11);
  }

}
```

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



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.