Listas Simples V1

Programación II (Universidad de Córdoba Colombia)

UNIVERSIDAD DE CÓRDOBA
comprometida con el desarrollo regional

CENTRO DE INNOVACIÓN EN TIC PARA APOYO A LA ACADEMIA
CINTIA

LISTAS ENLAZADAS

DESARROLLO

IMPLEMENTACIÓN DE LISTAS ENLAZADAS EN JAVA:

- **Ejercicio propuesto implementado en Java**

La siguiente guía, complementa el ejercicio propuesto de la Unidad III; implementado los métodos que agregarán nodos por la cabeza de la lista y después de un nodo existente en la lista. Igualmente se implementa un método que ordenar la lista enlazada, según la nota definitiva del estudiante. Es de recordar que el ejercicio 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 permitirá al usuario las siguientes opciones:

- Agregar nodos a la lista por la cabeza, por el final y después de un nodo existente.
- 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.
- Ordenar la información de la lista, según la nota definitiva.
- 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 enlazada y facilitaran realizar operaciones sobre la información de los nodos.

- **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. A la clase Nodo, la única modificación que se le hace, es agregar el método copiar(Nodo nd); al cual se le pasa como parámetro un objeto de la clase Nodo y tiene como función asignar valores a los atributos de la clase Nodo; en la parte que corresponde a la información del nodo.

```
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;
    }
    void copiar(Nodo nd){
```

```
        codigo = nd.getCodigo();
        nombre = nd.getNombre();
        nota1 = nd.getNota1();
        nota2 = nd.getNota2();
        nota3 = nd.getNota3();
    }
}
```

- **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 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. En esta clase la modificación que se hace es agregar estos cuatro nuevos métodos:

- agregarEntreNodos(Nodo nd, Nodo nuevo): El método que agrega un nodo, después de un nodo existente en la lista.
- agregarAcabeza(Nodo nuevo): El método que agrega un nuevo nodo, por la cabeza de la lista.
- cambiar(Nodo nod1, Nodo nod2): Es el método que cambia un nodo de una posición a otra dentro de la lista enlazada.
- OrdenarIntercambio(): Este método ordena la información de la lista enlazada, según la nota definitiva de cada estudiante.

```
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);
        }
    }

    void agregarEntreNodos(Nodo nd, Nodo nuevo){
        nuevo.setSiguiente(nd.getSiguiente());
        nd.setSiguiente(nuevo);
    }

    void agregarAcabeza(Nodo nuevo){
        nuevo.setSiguiente(cabeza);
        setCabeza(nuevo);
    }

    void cambiar(Nodo nod1, Nodo nod2){
        Nodo temp = new Nodo();
        temp.copiar(nod1);
        nod1.copiar(nod2);
        nod2.copiar(temp);
    }

    void OrdenarIntercambio(){
        Nodo ni;
        Nodo nj;
        ni = cabeza;
```

```
        while(ni != null){
            nj = ni.getSiguiente();
            while(nj != null){
                if(ni.definitiva() < nj.definitiva()){
                    cambiar(ni,nj);
                }
             nj = nj.getSiguiente();
            }
            ni = ni.getSiguiente();
        }
    }
}
```

- **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. En esta implementación, a la clase Main; solo se actualiza el método menu(), incluyendo las nuevas opciones, asociadas a las funcionalidades que se agregaron al ejercicio. Igualmente se agregan al switch; los case correspondientes a las nuevas opciones (case 8, 9 y 10).

```
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, "========== 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();
        }
    }

//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. Agregar Entre Dos Nodos \n"+"9. Agregar un Nodo por
la Cabeza \n"+"10. Ordenar Por Nota Definitiva \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) {
        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:
                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 9:
                aux = new Nodo();
                llenar(aux);
                lis.agregarAcabeza(aux);
                break;
            case 10:
                lis.OrdenarIntercambio();
                break;
            case 11:
                break;
            }
        }while(opcion != 11);
    }
}
```

EJERCICIOS/ACTIVIDADES

- El programa departamental de niños bajos en peso, quiere recolectar la información de los niños entre uno y seis años de los municipios de Sahagún, Montería y Lorica. Entre los datos relevantes del representante del niño (madre o padre) están la identificación y el nombre. En cuanto al niño se requiere tener el número de registro civil, el nombre, la talla y peso. Tengan en cuenta que un representante puede tener registrados hasta 2 niños.

  Para la solución de la aplicación se deben implementar en Java las clases necesarias que permitan llenar una lista enlazada, que guarde la información recolectada de los niños en cada municipio. El programa debe realizar las siguientes operaciones sobre los nodos almacenados en la lista enlazada:

  - Agregar la información del niño y su representante al final de la lista.
  - Insertar la información de un nuevo niño, entre dos nodos existentes.
  - Agregar la información de un niño y su representante al principio de la lista (agregar nodos por la cabeza de la lista).
  - Buscar la información de un niño pasando como dato de búsqueda la identificación el número de registro civil.
  - Ordenar la información de la lista teniendo en cuenta el peso de los niños registrados.
  - Eliminar la información de un niño y su representante.

  El director del programa departamental de niños bajos en peso requiere que la aplicación cuente con un menú de opciones para realizar las operaciones descritas anteriormente y para generar los siguientes informes sobre los datos almacenados en la lista:

  - Generar un listado por municipios, con la información de todos los niños registrados; se debe tener un consolidado de cuántos niños tiene el programa en cada municipio.

  - Se considera que los niños entre 2 y 3 años con un peso menor de 15 kilos están bajos de peso, lo mismo para los niños entre 4 y 6 años que pesan menos de 25 kilos. La dirección quiere un listado con la información de los niños que estén en estas condiciones y saber las cantidades en cada municipio para enviar unos mercados con una dieta especial.

  - Los niños entre 4 y 6 años que midan menos de un metro de estatura se consideran que están bajos de estatura, por esta razón se requiere saber la cantidad de niños que están bajos de talla en cada municipio.

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.