Universidad Nacional Mayor de San Marcos
Universidad del Perú. Decana de América

Estructura de Datos

Semana 4

Universidad Nacional Mayor de San Marcos
Universidad del Perú. Decana de América

Logro de la sesión

Al finalizar la sesión, el estudiante:

- **Utiliza en forma crítica las listas enlazadas para la solución de problemas e implementa programas utilizando listas enlazadas.**

Estructuras de datos lineales

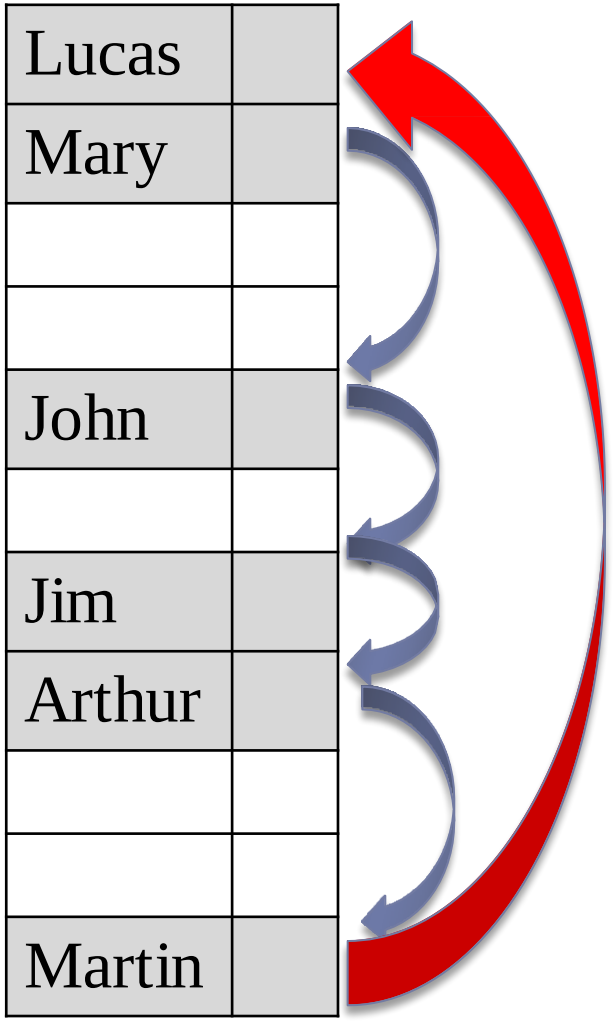
01

Implementación de un TAD lineal utilizando una estructura dinámica

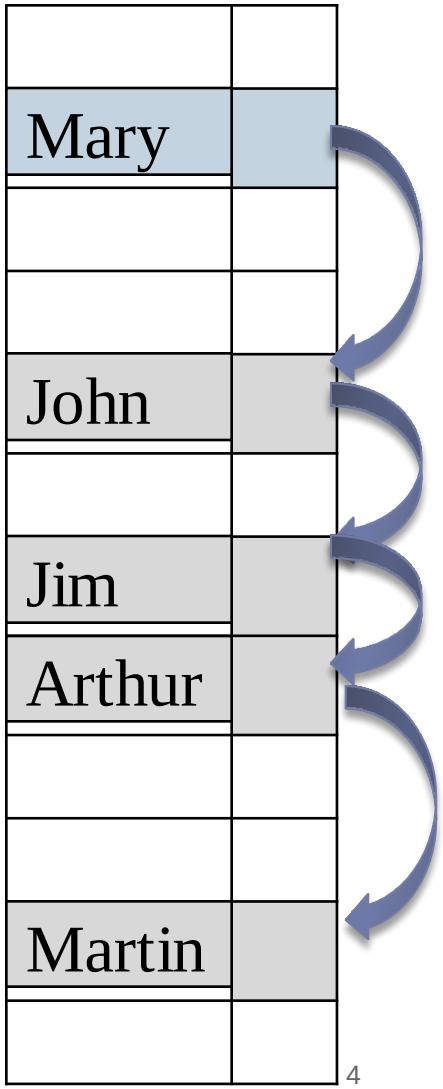
Estructuras de datos lineales

- **Estructura dinámica**

Los espacios en la memoria permiten que las órdenes físicas y lógicas puedan ser diferentes



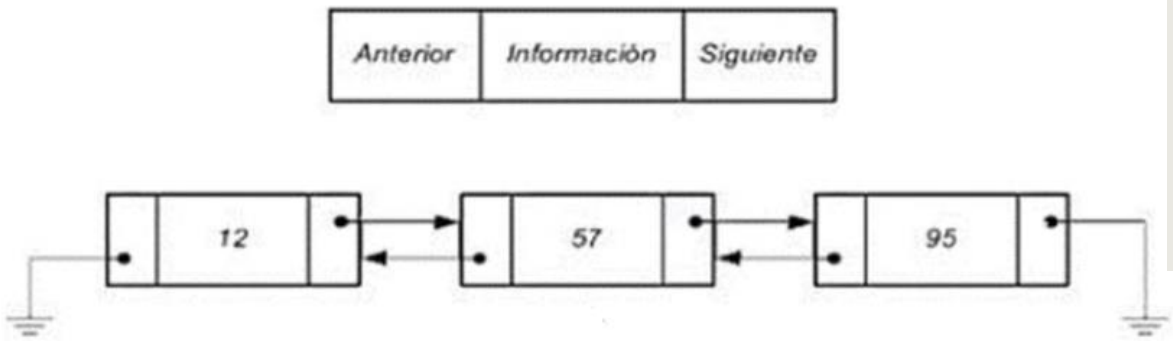
Cada ubicación no solo almacena (una referencia a) un objeto, sino también la referencia (dirección) a su sucesor en la Lista



Listas Doblemente Enlazadas

Contienen 2 punteros:

1. Uno apunta al Nodo Siguiente.
2. Otro apunta al Nodo Anterior.



La clase Nodo Doble

```
1  package Semana5;
2  public class NodoDoble {
3      public int dato;
4      NodoDoble siguiente,anterior;
5      public NodoDoble(int el){
6          this(el,null,null);
7      }
8      public NodoDoble(int el, NodoDoble s, NodoDoble a){
9          dato=el;
10         siguiente=s;
11         anterior=a;
12     }
13 }
```

Operaciones sobre las Listas Doblemente enlazadas

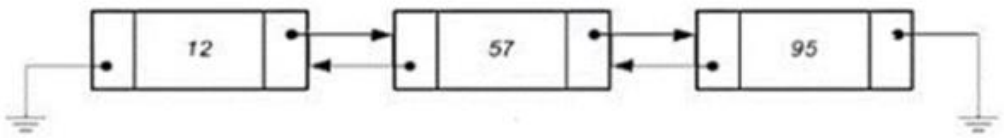
```java
package Semana5;
public class NodoDoble {
    public int dato;
    NodoDoble siguiente,anterior;
    public NodoDoble(int el){
        this(el,null,null);
    }
    public NodoDoble(int el, NodoDoble s, NodoDoble a
        dato=el;
        siguiente=s;
        anterior=a;
    }
}
```

Saber si la lista esta vacía

1. Retornar inicio==nulo

```java
public class ListaDoble {
    private NodoDoble inicio,fin;
    public ListaDoble(){
        inicio=fin=null;
    }
```

```java
//Método para saber cuando la lista está vacía
public boolean estaVacia(){
    return inicio==null;
}
```



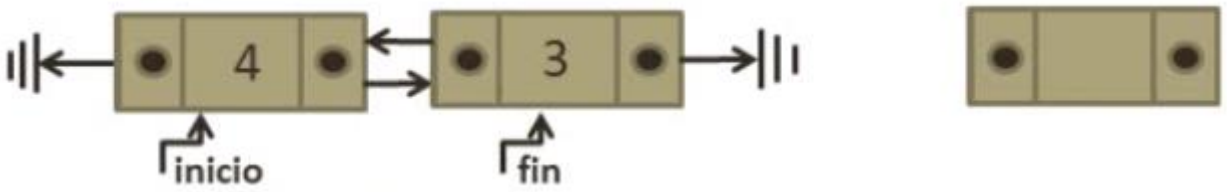
Agregar Un Nodo al Final

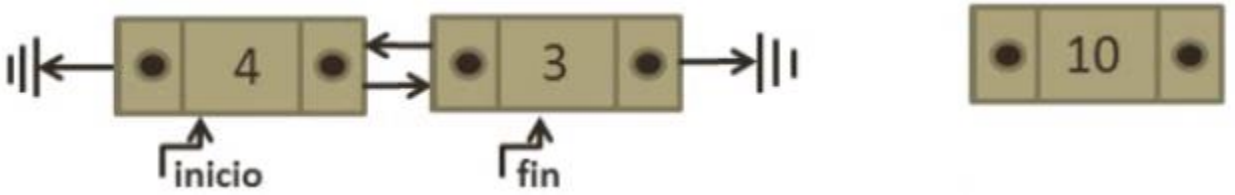
1. Si la Lista no está vacía entonces
   1. fin = nuevo NodoDoble(elemento, nulo, fin)
   2. Apuntar fin de anterior de siguiente a fin
2. Si No
   1. inicio = fin = nuevo NodoDoble(elemento)

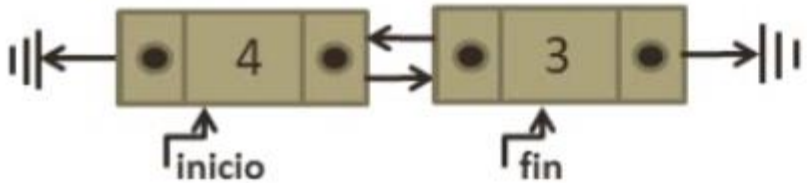
```java
//Método para agregar nodos al Final
public void agregarAlFinal(int el){
    if(!estaVacia()){
        fin=new NodoDoble(el, null, fin);
        fin.anterior.siguiente=fin;
    }else{
        inicio=fin=new NodoDoble(el);
    }
}
```

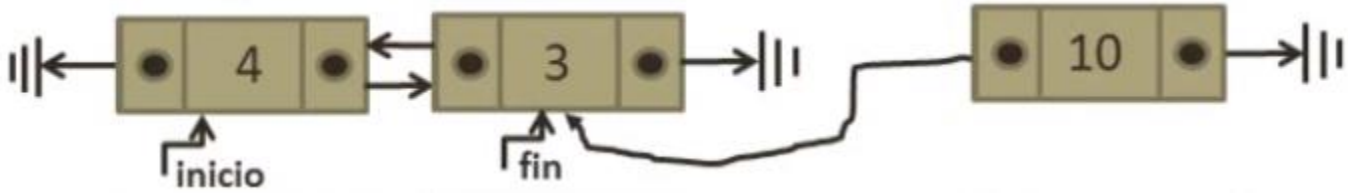
Agregar un nodo al final

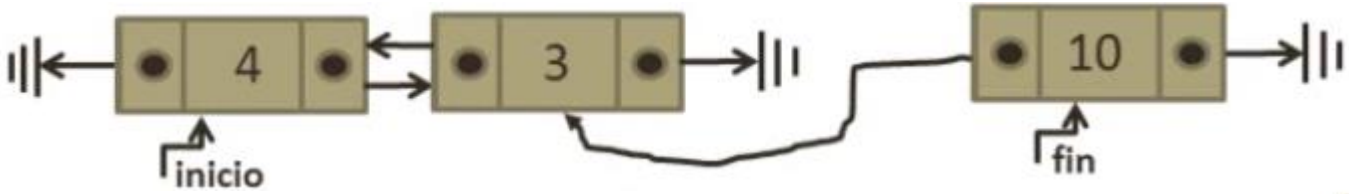
```
23      //Método para agregar nodos al Final
24      public void agregarAlFinal(int el){
25          if(!estaVacia()){
26              fin=new NodoDoble(el, null, fin);
27              fin.anterior.siguiente=fin;
28          }else{
29              inicio=fin=new NodoDoble(el);
30          }
31      }
```

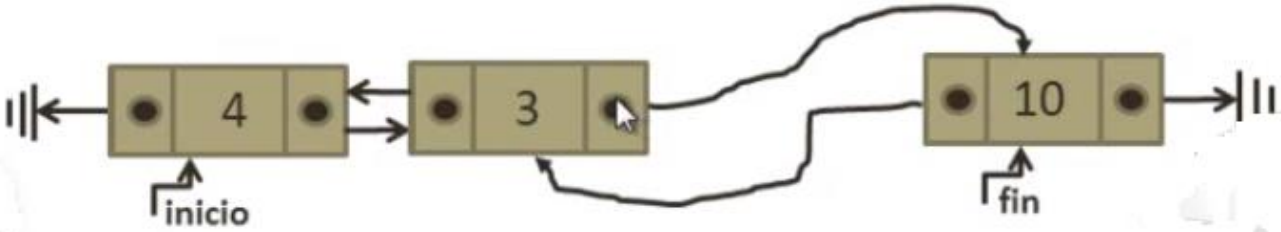










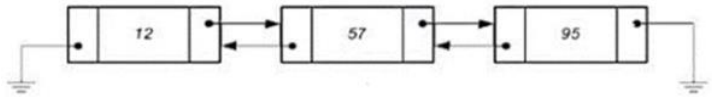


Operaciones sobre las Listas Doblemente enlazadas

Eliminar un Nodo del Inicio LD

1. elemento = inicio de dato
2. Si inicio es igual a fin entonces
   1. Apuntar inicio y fin a nulo
3. Si No
   1. Apuntar inicio a inicio de siguiente
   2. Apuntar inicio de anterior a nulo
4. Retornar el valor del elemento

```
69      //Metodo para eliminar del inicio
70      public int eliminarDelInicio(){
71          int elemento=inicio.dato;
72          if(inicio==fin){
73              inicio=fin=null;
74          }else{
75              inicio=inicio.siguiente;
76              inicio.anterior=null;
77          }
78          return elemento;
79      }
```



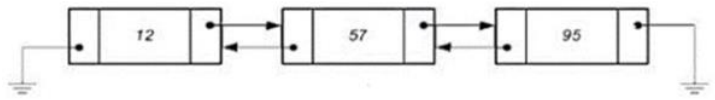
Eliminar un Nodo del Final LD

1. elemento = fin de dato
2. Si inicio es igual a fin entonces
   1. Apuntar inicio y fin a nulo
3. Si No
   1. Apuntar fin a fin de anterior
   2. Apuntar fin de siguiente a nulo
4. Retornar el valor del elemento

```
80      //metodo para eliminar del final
81      public int eliminarDelFinal(){
82          int elemento=fin.dato;
83          if(inicio==fin){
84              inicio=fin=null;
85          }else{
86              fin=fin.anterior;
87              fin.siguiente=null;
88          }
89          return elemento;
90
91      }
```

Clase ListaDoble

```java
package Semana5;
import javax.swing.JOptionPane;
public class ListaDoble {
    private NodoDoble inicio,fin;
    public ListaDoble(){
        inicio=fin=null;
    }
    //Método para saber cuando la lista está vacía
    public boolean estaVacia(){
        return inicio==null;
    }
    //Método para agregar nodos al Final
    public void agregarAlFinal(int el){
        if(!estaVacia()){
            fin=new NodoDoble(el, null, fin);
            fin.anterior.siguiente=fin;
        }else{
            inicio=fin=new NodoDoble(el);
        }
    }
```




```java
    //Método para agregar al inicio
    public void agregarAlInicio(int el){
        if(!estaVacia()){
            inicio=new NodoDoble(el, inicio, null);
            inicio.siguiente.anterior=inicio;
        }else{
            inicio=fin=new NodoDoble(el);
        }
    }
    //Método para mostrar la lista de inicio a fin
    public void mostrarListaInicioFin(){
        if(!estaVacia()){
            String datos="<=>";
            NodoDoble auxiliar=inicio;
            while(auxiliar!=null){
                datos= datos + "["+auxiliar.dato+"]<=>";
                auxiliar=auxiliar.siguiente;
            }
            JOptionPane.showMessageDialog(null, datos,
                    "Mostrando Lista de Inicio a Fin",
                    JOptionPane.INFORMATION_MESSAGE);
        }
    }
    //Método para mostrar la Lista de Fin a Inicio
    public void mostrarListaFinInicio(){
        if(!estaVacia()){
            String datos="<=>";
            NodoDoble auxiliar=fin;
            while(auxiliar!=null){
                datos= datos + "["+auxiliar.dato+"]<=>";
                auxiliar=auxiliar.anterior;
            }
            JOptionPane.showMessageDialog(null, datos,
                    "Mostrando Lista de Fin a Inicio",
                    JOptionPane.INFORMATION_MESSAGE);
        }
    }
```

Clase ListaDoble

```
58          //Metodo para eliminar del inicio
59          public int eliminarDelInicio(){
60              int elemento=inicio.dato;
61              if(inicio==fin){
62                  inicio=fin=null;
63              }else{
64                  inicio=inicio.siguiente;
65                  inicio.anterior=null;
66              }
67              return elemento;
68          }
69          //metodo para eliminar del final
70          public int eliminarDelFinal(){
71              int elemento=fin.dato;
72              if(inicio==fin){
73                  inicio=fin=null;
74              }else{
75                  fin=fin.anterior;
76                  fin.siguiente=null;
77              }
78              return elemento;
79          }
80      }
```

Clase UsoListaDoble

```
1   package Semana5;
2   import javax.swing.JOptionPane;
3   public class UsoListaDoble {
4       public static void main(String[] args) {
5           ListaDoble lista2=new ListaDoble();
6           int opcion=0,elemento;
7           do{
8               try{
9                   opcion=Integer.parseInt(JOptionPane.showInputDialog(null,
10                          "1. Agregar un Nodo al inicio\n"+
11                          "2. Agregar un Nodo al final\n"+
12                          "3. Mostrar la Lista de inicio a fin\n"+
13                          "4. Mostrar la Lista de fin a inicio\n"+
14                          "5. Eliminar un Nodo del inicio\n"+
15                          "6. Eliminar un Nodo del final\n"+
16                          "7. Salir\n"+
17                          "¿Que deseas hacer?", "Menu de opciones", JOptionPane.INFORMATION_MESSAGE));
18                  switch(opcion){
19                      case 1:
20                          elemento=Integer.parseInt(JOptionPane.showInputDialog(null,
21                                  "Ingresa el elemento del Nodo","Agregando el Nodo al inicio",
22                                  JOptionPane.INFORMATION_MESSAGE));
23                          lista2.agregarAlInicio(elemento);
24                          break;
```

Clase UsoListaDoble

```
25                  case 2:
26                      elemento=Integer.parseInt(JOptionPane.showInputDialog(null,
27                              "Ingresa el elemento del Nodo","Agregando el Nodo al final",
28                              JOptionPane.INFORMATION_MESSAGE));
29                      lista2.agregarAlFinal(elemento);
30
31                      break;
32                  case 3:
33                      if(!lista2.estaVacia()){
34                          lista2.mostrarListaInicioFin();
35                      }else{
36                      JOptionPane.showMessageDialog(null,"No hay nodos aun",
37                              "Lista vacia",JOptionPane.INFORMATION_MESSAGE);
38                      }
39                      break;
40                  case 4:
41                      if(!lista2.estaVacia()){
42                          lista2.mostrarListaFinInicio();
43                      }else{
44                      JOptionPane.showMessageDialog(null,"No hay nodos aun",
45                              "Lista vacia",JOptionPane.INFORMATION_MESSAGE);
46                      }
47                      break;
```

Clase UsoListaDoble

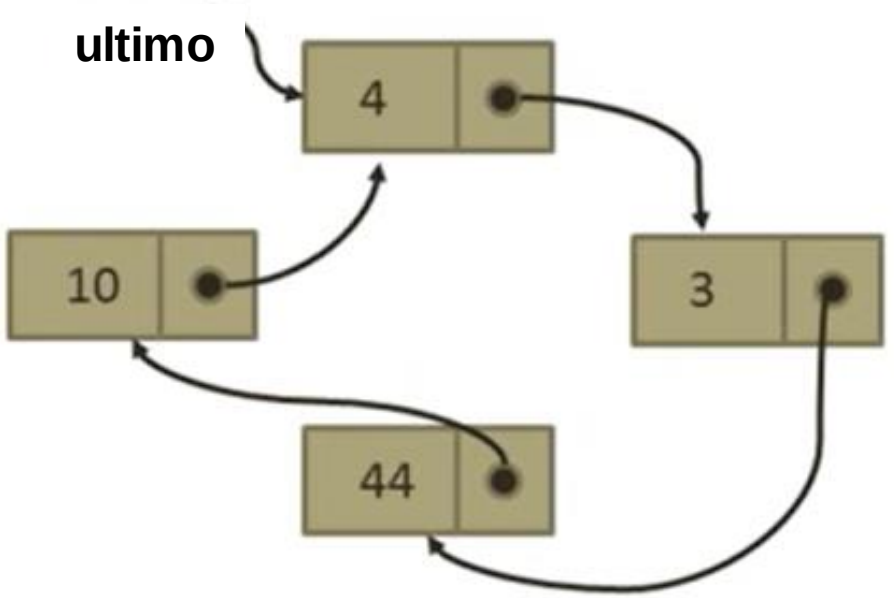
```
48                  case 5:
49                      if(!lista2.estaVacia()){
50                          elemento=lista2.eliminarDelInicio();
51                      JOptionPane.showMessageDialog(null,"El elemento eliminado es: "+elemento,
52                              "Eliminando del inicio",JOptionPane.INFORMATION_MESSAGE);
53                      }else{
54                              JOptionPane.showMessageDialog(null,"No hay nodos aun",
55                              "Lista vacia",JOptionPane.INFORMATION_MESSAGE);
56                      }
57                      break;
58                  case 6:
59                      if(!lista2.estaVacia()){
60                          elemento=lista2.eliminarDelFinal();
61                      JOptionPane.showMessageDialog(null,"El elemento eliminado es: "+elemento,
62                              "Eliminando del final",JOptionPane.INFORMATION_MESSAGE);
63                      }else{
64                              JOptionPane.showMessageDialog(null,"No hay nodos aun",
65                              "Lista vacia",JOptionPane.INFORMATION_MESSAGE);
66                      }
67                      break;
```

Clase UsoListaDoble

```
68                      case 7:
69                          JOptionPane.showMessageDialog(null,"Aplicacion Finalizada",
70                                  "Fin",JOptionPane.INFORMATION_MESSAGE);
71                          break;
72                      default:
73                          JOptionPane.showMessageDialog(null,"La Opcion no esta en el Menu",
74                                  "Incorrecto",JOptionPane.INFORMATION_MESSAGE);
75                  }
76              }catch(NumberFormatException n){
77                  JOptionPane.showMessageDialog(null, "Error "+n.getMessage());
78              }
79          }while(opcion!=7);
80      }
81  }
82
```

Listas circulares simplemente enlazadas

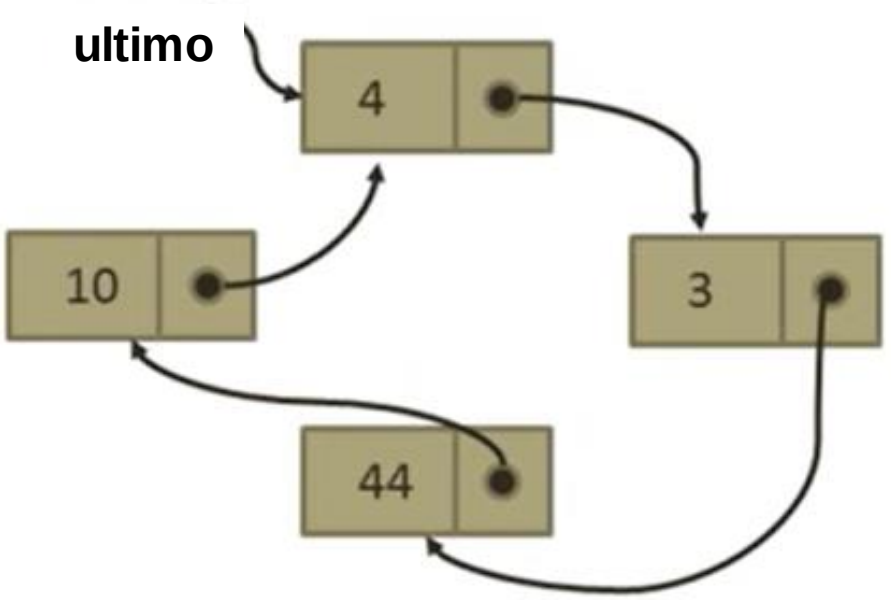
1. No tienen Inicio y Fin
2. Necesitamos establecer un Nodo a partir del cual podamos acceder a la Lista.
3. Es una Lista Simplemente Enlazada cuyo último Nodo apunta al Primero.



Clase NodoLC y Clase ListaLC

```java
package Semana5;
public class NodoLC {
    int dato;
    NodoLC siguiente;
    public NodoLC(int d){
        dato=d;
        siguiente=this;
    }
}
```

```java
public class ListaLC {
    NodoLC ultimo;
    public ListaLC(){
        ultimo=null;
    }
}
```

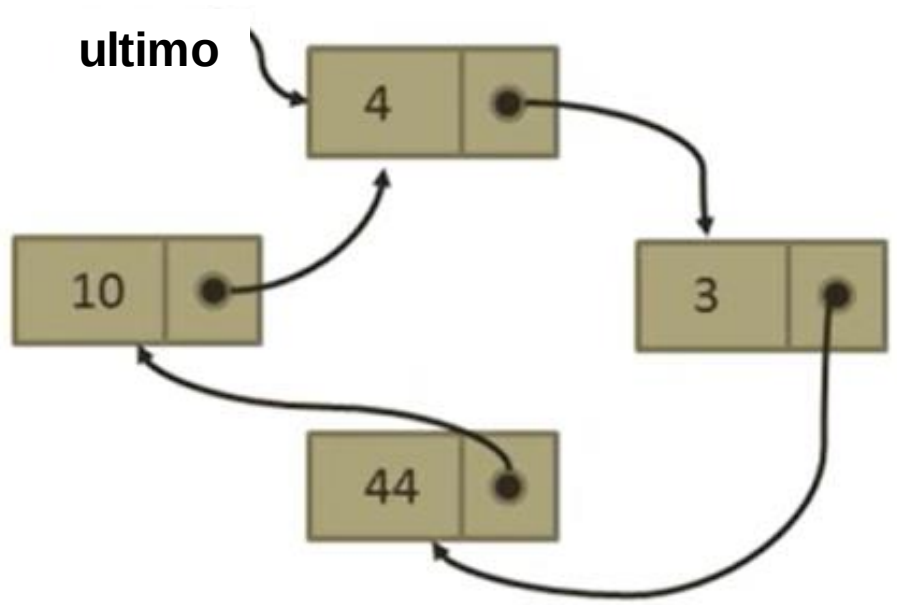


Operaciones sobre las listas circulares simplemente enlazadas

Saber si la ListaC esta vacía

1. Retornar ultimo==nulo

```
8       //Metodo para saber cuando la lista esta vacia
9       public boolean estaVacia(){
10          return ultimo==null;
11      }
```



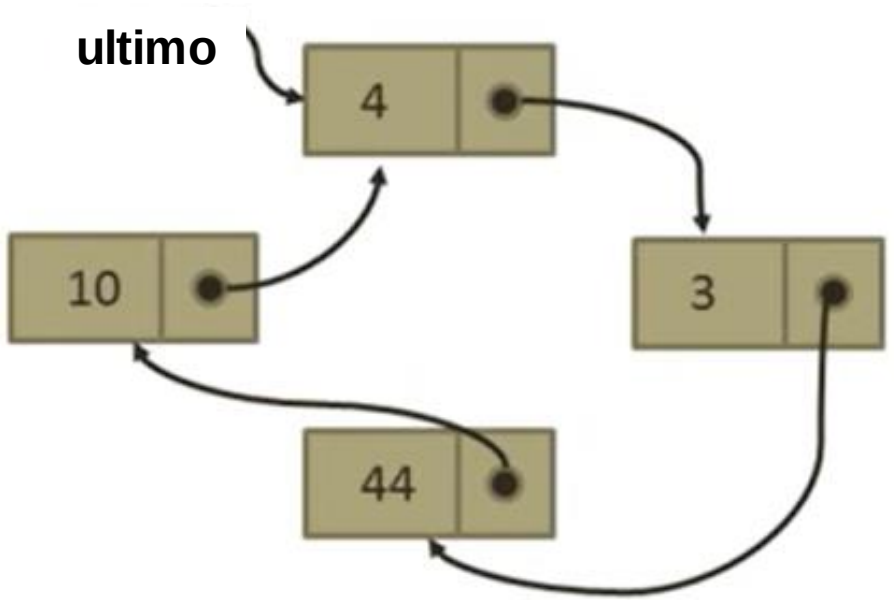
Agregar Un Nodo en la ListaC

1. Crear un nuevo nodo
2. Si ultimo es diferente de nulo entonces
   1. Apuntar nuevo de siguiente a ultimo de siguiente
   2. Apuntar ultimo de siguiente a nuevo
3. Apuntar ultimo a nuevo
4. Retornar "este"

```
12      //Metodo para insertar Nodos
13      public ListaLC insertar(int elemento){
14          NodoLC nuevo=new NodoLC(elemento);
15          if(ultimo!=null){
16              nuevo.siguiente=ultimo.siguiente;
17              ultimo.siguiente=nuevo;
18          }else{
19              ultimo=nuevo;
20          }
21          return this;
22      }
```

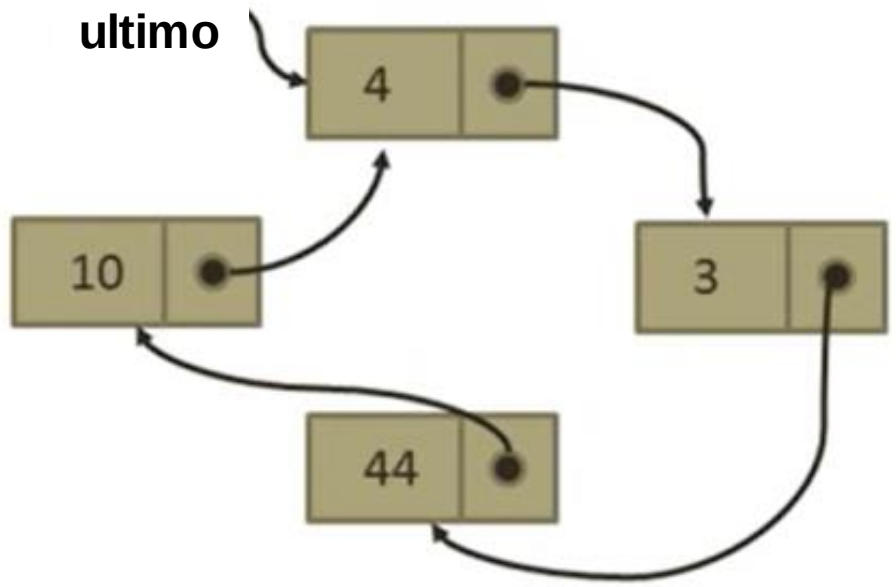
Operaciones sobre las listas circulares simplemente enlazadas – Mostrar Lista

```
23      //Metodo para mostrar la lista
24      public void mostrarLista(){
25          NodoLC auxiliar=ultimo.siguiente;
26          String cadena="";
27          do{
28              cadena=cadena + "["+auxiliar.dato+"]->
29              auxiliar=auxiliar.siguiente;
30          }while(auxiliar!=ultimo.siguiente);
31          JOptionPane.showMessageDialog(null, cadena,
32                  "Mostrando la Lista Circular",JOptionPane.INFORMATION_MESSAGE);
33      }
```



Operaciones sobre las listas circulares simplemente enlazadas – Eliminar un elemento de la Lista

```
34      //Metodo para eliminar un nodo de la lista circular
35      public boolean eliminar(int elemento){
36          NodoLC actual;
37          boolean encontrado=false;
38          actual=ultimo;
39          while(actual.siguiente!=ultimo && !encontrado){
40              encontrado=(actual.siguiente.dato==elemento);
41              if(!encontrado){
42                  actual=actual.siguiente;
43              }
44          }
45          encontrado=(actual.siguiente.dato==elemento);
```




```
46              if(encontrado){
47                  NodoLC auxiliar=actual.siguiente;
48                  if(ultimo==ultimo.siguiente){
49                      ultimo=null;
50                  }else{
51                      if(auxiliar==ultimo){
52                          ultimo=actual;
53                      }
54                      actual.siguiente=auxiliar.siguiente;
55                  }
                    auxiliar=null;
57              }
58              return encontrado==true;
59          }
60      }
```

Clase ListaLC

```java
package Semana5;
import javax.swing.JOptionPane;
public class ListaLC {
    NodoLC ultimo;
    public ListaLC(){
        ultimo=null;
    }
    //Metodo para saber cuando la lista esta vacia
    public boolean estaVacia(){
        return ultimo==null;
    }
    //Metodo para insertar Nodos
    public ListaLC insertar(int elemento){
        NodoLC nuevo=new NodoLC(elemento);
        if(ultimo!=null){
            nuevo.siguiente=ultimo.siguiente;
            ultimo.siguiente=nuevo;
        }else{
            ultimo=nuevo;
        }
        return this;
    }
    //Metodo para mostrar la lista
    public void mostrarLista(){
        NodoLC auxiliar=ultimo.siguiente;
        String cadena="";
        do{
            cadena=cadena + "["+auxiliar.dato+"]->";
            auxiliar=auxiliar.siguiente;
        }while(auxiliar!=ultimo.siguiente);
        JOptionPane.showMessageDialog(null, cadena,
                "Mostrando la Lista Circular",JOptionPane.INFORMATION_MESSAGE);
    }
    //Metodo para eliminar un nodo de la lista circular
    public boolean eliminar(int elemento){
        NodoLC actual;
        boolean encontrado=false;
        actual=ultimo;
        while(actual.siguiente!=ultimo && !encontrado){
            encontrado=(actual.siguiente.dato==elemento);
            if(!encontrado){
                actual=actual.siguiente;
            }
        }
        encontrado=(actual.siguiente.dato==elemento);
        if(encontrado){
            NodoLC auxiliar=actual.siguiente;
            if(ultimo==ultimo.siguiente){
                ultimo=null;
            }else{
                if(auxiliar==ultimo){
                    ultimo=actual;
                }
                actual.siguiente=auxiliar.siguiente;
            }
            auxiliar=null;
        }
        return encontrado==true;
    }
}
```

Clase UsoListaLC

```
1   package Semana5;
2   import javax.swing.JOptionPane;
3   public class UsoListaLC {
4       public static void main(String[] args) {
5           ListaLC lista3=new ListaLC();
6           int opcion=0,elemento;
7           boolean eliminado=false;
8           do{
9               try{
10                  opcion=Integer.parseInt(JOptionPane.showInputDialog(null,
11                          "1. Agregar un Nodo a la lista circular\n"+
12                          "2. Eliminar un nodo de la lista circular\n"+
13                          "3. Mostrar datos de la Lista circular\n"+
14                          "4. Salir\n"+
15                          "¿Que deseas hacer?", "Menu de opciones", JOptionPane.INFORMATION_MESSAGE));
16                  switch(opcion){
17                      case 1:
18                          elemento=Integer.parseInt(JOptionPane.showInputDialog(null,
19                                  "Ingresa el elemento del Nodo","Agregando el Nodo a la lista circular",
20                                  JOptionPane.INFORMATION_MESSAGE));
21                          lista3.insertar(elemento);
22                          break;
```

Clase UsoListaLC

```
23                  case 2:
24                      if(!lista3.estaVacia()){
25                      elemento=Integer.parseInt(JOptionPane.showInputDialog(null,
26                              "Ingresa el elemento del Nodo a eliminar","Eliminando Nodo de la lista circular",
27                              JOptionPane.INFORMATION_MESSAGE));
28                      eliminado=lista3.eliminar(elemento);
29                      if(eliminado){
30                      JOptionPane.showMessageDialog(null,"El elemento eliminado es " + elemento,
31                              "Eliminando Nodos",JOptionPane.INFORMATION_MESSAGE);
32                      }else{
33                      JOptionPane.showMessageDialog(null,"El elemento "+elemento+" no esta en la lista",
34                              "Elemento no encontrado",JOptionPane.INFORMATION_MESSAGE);
35
36                      }
37                      }else{
38                      JOptionPane.showMessageDialog(null,"No hay nodos aun",
39                              "Lista vacia",JOptionPane.INFORMATION_MESSAGE);
40                      }
41                      break;
42                  case 3:
43                      if(!lista3.estaVacia()){
44                          lista3.mostrarLista();
45                      }else{
46                      JOptionPane.showMessageDialog(null,"No hay nodos aun",
47                              "Lista vacia",JOptionPane.INFORMATION_MESSAGE);
48                      }
49                      break;
```

Clase UsoListaLC

```
50                          case 4:
51                              JOptionPane.showMessageDialog(null,"Aplicacion Finalizada",
52                                      "Fin",JOptionPane.INFORMATION_MESSAGE);
53                              break;
54                          default:
55                              JOptionPane.showMessageDialog(null,"La Opcion no esta en el Menu",
56                                      "Incorrecto",JOptionPane.INFORMATION_MESSAGE);
57                      }
58                  }catch(NumberFormatException n){
59                      JOptionPane.showMessageDialog(null, "Error "+n.getMessage());
60                  }
61
62              }while(opcion!=4);
63          }
64      }
```

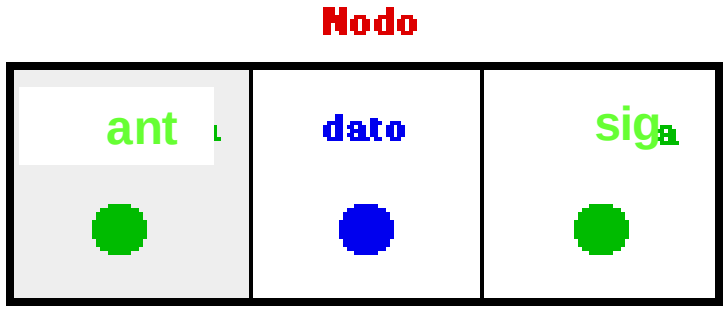
Lista doblemente enlazada

Cada nodo contiene dos enlaces, uno a su nodo predecesor y el otro a su nodo sucesor. La lista es eficiente tanto como en recorrido directo (“adelante”) como en recorrido inverso (“atras”).

Nodo

```
public class Nodo {
    int dato;
    Nodo sig;
    Nodo ant;
    public Nodo(int num)
    {  dato=num;
       sig=ant=null;
     }
  }
```

```
public class Nodo {
   private int data; // almacena el dato
   private Nodo sig; //"enlace" al próximo nodo
   private Nodo  ant; // "enlace" al anterior nodo
}
```



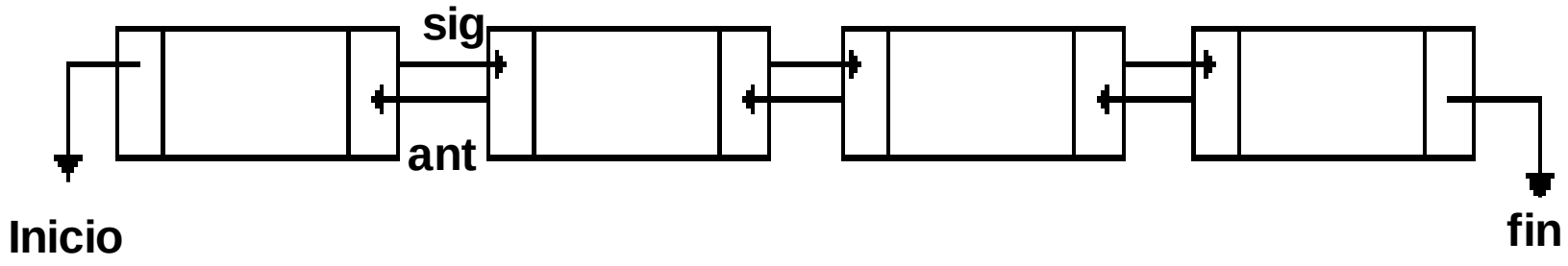
Operaciones de una lista doblemente enlazada

Añadir o insertar elementos.

Buscar elementos.

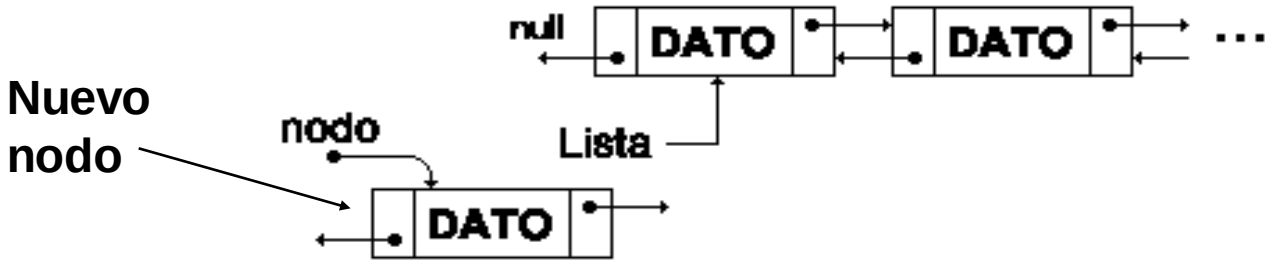
Borrar elementos.

Moverse a través de la lista, siguiente y anterior.



Insertar nuevo nodo

Insertar nodo en la primera posición

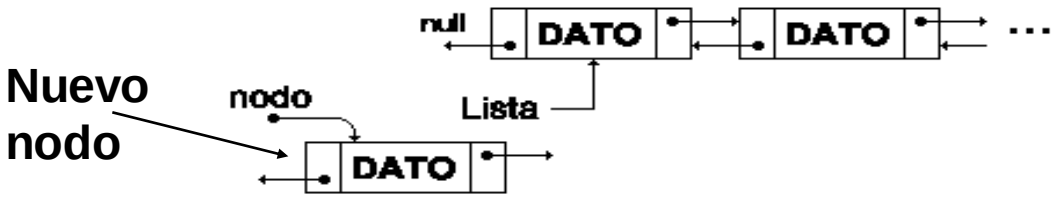


InsertaNodo(nodo)

1-. Nodo→siguiente // debe apuntar a Lista.

2-. Nodo→anterior y Lista→anterior // debe apuntar a null

3-. Lista→anterior //debe apuntar a nodo.




```
public class Nodo {
      int dato;
      Nodo sig;
      Nodo ant;
      public Nodo(int num)
      {  dato=num;
         sig=ant=null;
       }
    }
```

```
Nodo InsertaInicio(Nodo inicio, int num)
{     Nodo nuevo=new Nodo(num); // se llama al constructor
       //Realizando los enlaces correspondientes
      nuevo.sig=inicio;
      if(inicio==null) // solo para la primera vez
      {  fin=nuevo;
        fin.sig=null;
      }
      if(inicio!=null)
         inicio.ant=nuevo;
      inicio=nuevo;
      return inicio;
}
```

Recorrido en una lista doble

HACIA ADELANTE

```
int Suma(Nodo inicio)
 {   Nodo aux=inicio;
     int suma=0;
     //Recorriendo la lista
     while(aux!=null)
      {    //Sumando los datos
           suma=suma+aux.dato;
          //Avanzando al siguiente nodo
          aux=aux.sig;
       }
    return suma;
 }
```

HACIA ATRAS

```
int Suma(Nodo fin)
 {   Nodo aux=fin;
     int suma=0;
     //Recorriendo la lista
     while(aux!=null)
      {    //Sumando los datos
           suma=suma+aux.dato;
          //Retrocediendo al nodo anterior
          aux=aux.ant;
       }
    return suma;
 }
```

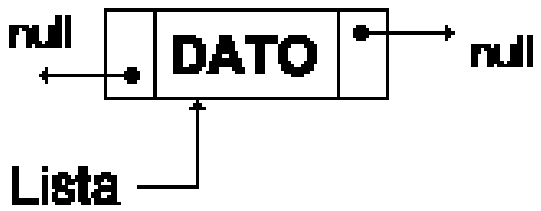
Eliminar

Caso 1 Eliminar el único nodo

En este caso, ese nodo será el apuntado por Lista.

1.- Eliminamos el nodo.

2.- Hacemos que Lista apunte a NULL.

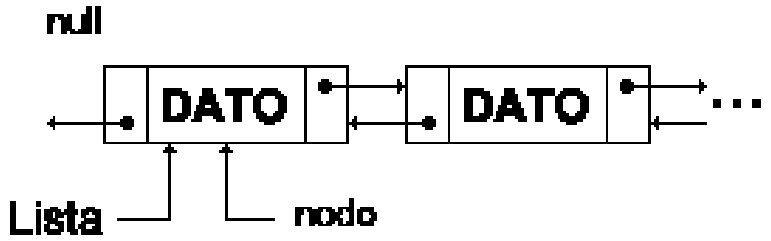


eliminaPrimer(nodo)

1.- Si nodo apunta a Lista // hacemos que Lista apunte Lista=siguiente.

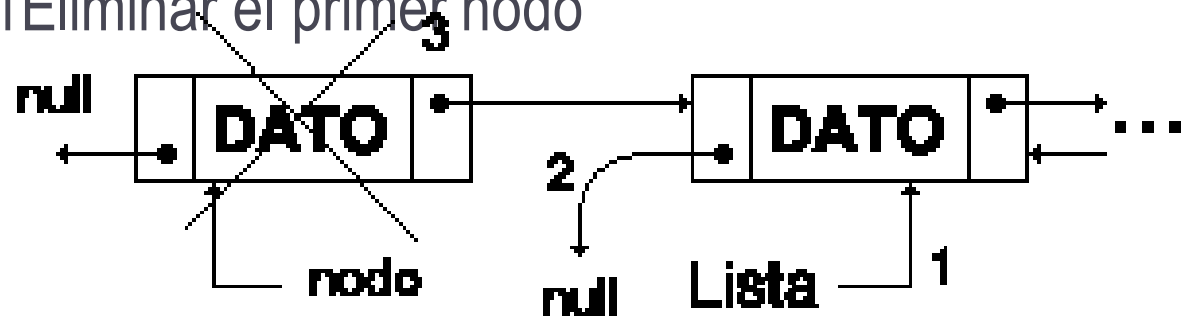
2.- Hacemos que nodo=siguiente=anterior// apunte a NULL

3.- Borramos el nodo apuntado por nodo.

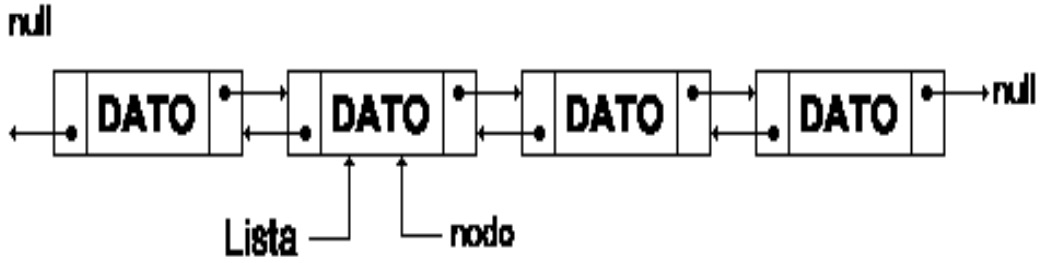


Caso 2

Caso 2.1Eliminar el primer nodo



Caso 2.2 Eliminar un nodo intermedio



eliminaMedio(nodo)

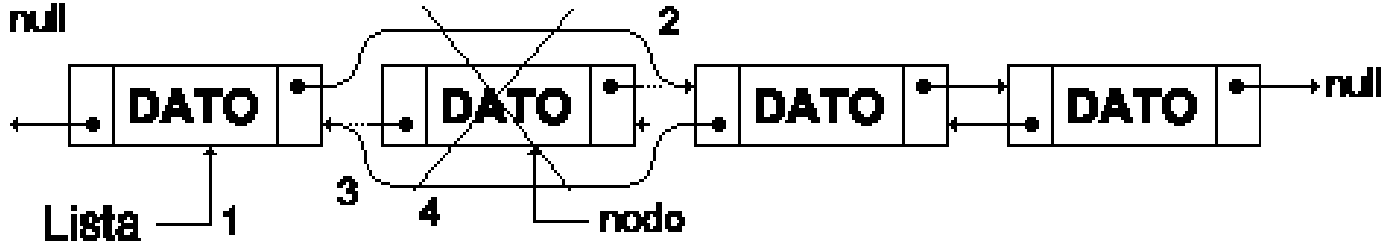
1-. Si nodo apunta a Lista

Lista=siguiente

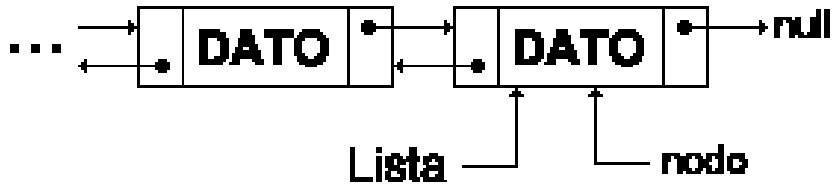
2-. nodo= siguiente

3-. Nodo = anterior

4-. Borramos el nodo apuntado por nodo



Caso 3 Eliminar el último nodo



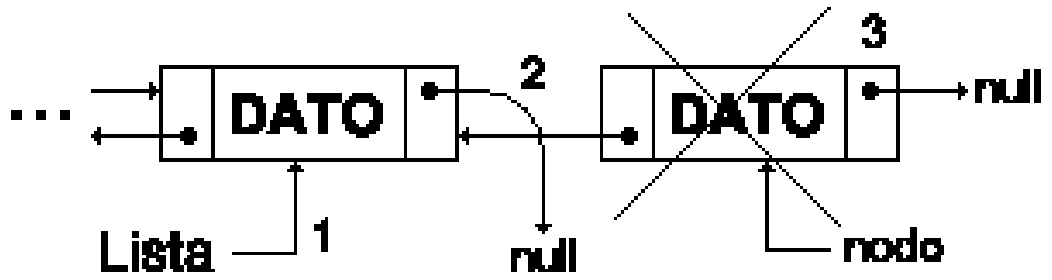
eliminaUltimo(nodo)

1-. Si nodo apunta a Lista

Lista=anterior.

2- nodo=anterior=siguiente apunte a NULL

3-. Borramos el nodo apuntado por nodo.



```
void Eliminar(Nodo inicio, int elem)
{   Nodo actual;
    boolean encontrado=false;
    actual=inicio;
    while((actual!=null)&& (!encontrado)) //bucle de Búsqueda
      {   encontrado=(actual.dato==elem); // se actualiza elvalor de encontrado
           if(!encontrado)  // severifica para pasar al siguiente nodo
                 actual=actual.sig;
       }
  //Realizando los enlaces
 if (actual != null)  // se verifica si se encontro el elemento buscado
  {  if (actual == ini)
     {   ini = actual.sig; // borrar el primero
        if(actual.sig!=null)
             actual.sig.ant=null;
     }
     else if (actual.sig!=null) // No es el ultimo
           {   actual.ant.sig=actual.sig;
              actual.sig.ant=actual.ant;
           }
         else
         {   actual.ant.sig=null; // el ultimo
            fin=actual.ant;  // moviendo el final
         }
     actual=null;
   }
}
```

BUSCAR UN NODO CON ALGUNA CARACTERÍSTICA

```
public class Nodo{
   // atributos
     public String codigo;
   // puntero al siguiente nodo
     public Nodo sig ;
    public Nodo ant;
   // el constructor de nodos
     Nodo (String cod)
      {        codigo=cod;
               ant = sig = null;
      }
}
```

```
Nodo Buscar(Nodo inicio,String cod)
{    //Generamos un puntero y lo colocamos al inicio de la lista
     Nodo pos=inicio;
          //Recorriendo la lista para encontrar la información
          while(pos!=null && !cod.equalsIgnoreCase(pos.codigo))
               pos=pos.sig;
          //Retorno de la posición del dato
          return pos;
}
```

EJEMPLO

Construya una aplicación que utilice listas doblemente enlazadas y que permita realizar el registro de empleados donde se podrá guardar, mostrar, consultar, actualizar y eliminar el registro de empleado. Para todas estas operaciones considere el ingreso del código del empleado.

N°	Codigo	Nombres	Apellidos	Sexo	Sueldo
1	1200	DARIO	BEJARANO	MASCULINO	4345,00
2	6542	MARIO	SALAMANCA	MASCULINO	3356,00
3	8541	GABRIELA	BRAVO	FEMENINO	3444,00
4	8544	LILY	SALAZAR	FEMENINO	5825,00
5	0609	NAZLY	ESTRADA	FEMENINO	5621,00
6	1254	MILUSKA	PRIETO	FEMENINO	2345,00
7	3334	JORGE	LAZO	MASCULINO	9841,00

DECLARACIÓN DEL CLASE NODO

```
1  package listas_dobles;
2  import java.text.DecimalFormat;
3  import java.awt.Font;
4  import javax.swing.JOptionPane;
5  import javax.swing.table.DefaultTableModel;
6  public class listas_dobles extends javax.swing.JFrame {
7      //Declaracion de la Lista doblemente enlazada
8  public class Nodo {
9        String codigo;
10       String nombre;
11       String apellidos;
12       String sexo;
13       float sueldo;
14       Nodo sig;
15       Nodo ant;
16     public Nodo(String cod, String nom,String ape,String sex,float suel)
17      {  codigo=cod;
18         nombre=nom;
19         apellidos=ape;
20         sexo=sex;
21         sueldo=suel;
22         sig=ant=null;
23      }
24     }
```

```
25      //Declaracion del formato de la tabla
26      DefaultTableModel miModelo;
27      String[] cabecera={"N°","Codigo","Nombres","Apellidos","Sexo","Sueldo"};
28      String[][] data={};
29    //Declaracion de variables locales
30       public Nodo ini,fin;
31       public Nodo pFound;
32       int num=0;
33       public listas_dobles(){
34            initComponents();
35            //Inicializando los punteros
36            ini=fin=pFound=null;
37            //Habilitando los encabezados de la tabla
38            miModelo=new DefaultTableModel(data,cabecera);
39            tblRegistros.setModel(miModelo);
40       }
41       @SuppressWarnings("unchecked")
```

BOTÓN GUARDAR

```
    private void btnGuardarActionPerformed(java.awt.event.ActionEvent evt) {
        //Capturando la informacion de los objetos
        String cod=txtCodigo.getText();
        String nom=txtNombre.getText().toUpperCase();
        String ape=txtApellidos.getText().toUpperCase();;
        String sex=cbxSexo.getSelectedItem().toString();
        String suel=txtSueldo.getText();
       //Creando el nodo de la Lista en memoria y colocando la informacion
        ini=InsertaFinal(ini,cod,nom,ape,sex,Float.parseFloat(suel));
        LimpiarEntradas();
        VerDatos(1);
        Resumen();
}
```

```
Nodo InsertaFinal(Nodo inicio,String cod,String nom,String ape,String sex,float suel)
 {      Nodo nuevo=new Nodo(cod,nom,ape,sex,suel);
         //Realizando los enlaces correspondientes
        nuevo.sig=inicio;
        if(inicio==null)
        {   fin=nuevo;
            fin.sig=null;
        }
        if(inicio!=null)
            inicio.ant=nuevo;

        inicio=nuevo;
        return inicio;
 }
```

```
private void btnConsultarActionPerformed(java.awt.event.ActionEvent evt) {
 String cod=txtCodigo.getText();
    if(cod.equalsIgnoreCase("")) {
        JOptionPane.showMessageDialog(this,"Ingrese un codigo por favor");
    } else {
        //Llamada a la funcion que retorna la posicion del dato buscado
        pFound=Buscar(ini,cod);
        //Verificando el puntero pFound para mostrar la inf. buscada
        if(pFound!=null) {
            txtNombre.setText(pFound.nombre);
            txtApellidos.setText(pFound.apellidos);
            if(pFound.sexo.equalsIgnoreCase("MASCULINO"))
                cbxSexo.setSelectedIndex(2);
            else
                cbxSexo.setSelectedIndex(1);
            txtSueldo.setText(String.valueOf(pFound.sueldo));
            //Habilitamos los objetos para eliminar y actualizar
            Habilitar();
        } else {
            JOptionPane.showMessageDialog(this,"El código: "+cod + ", no esta en la Lista..");
        }
    }
}
```

BOTÓN CONSULTAR

```
Nodo Buscar(Nodo inicio,String cod)
{     Nodo pos=inicio;
            //Recorriendo la lista para encontrar la informacion
            while(pos!=null && !cod.equalsIgnoreCase(pos.codigo))
                pos=pos.sig;
     return pos;
}
```

BOTÓN ACTUALIZAR

```
    private void btnActualizarActionPerformed(java.awt.event.ActionEvent evt) {
        //Colocando la informacion en el puntero pFound
        pFound.codigo=txtCodigo.getText();
        pFound.nombre=txtNombre.getText().toUpperCase();
        pFound.apellidos=txtApellidos.getText().toUpperCase();
        pFound.sexo=cbxSexo.getSelectedItem().toString();
        pFound.sueldo=Float.parseFloat(txtSueldo.getText());
        LimpiarEntradas();
        Deshabilitar();
        VerDatos(1);
        Resumen();
}
```

```
    private void btnEliminarActionPerformed(java.awt.event.ActionEvent evt) {
        Eliminar();
        LimpiarEntradas();
        VerDatos(1);
        if(ini==null)
             JOptionPane.showMessageDialog(this,"La lista esta vacía");
        Deshabilitar();
        Resumen();
}
```

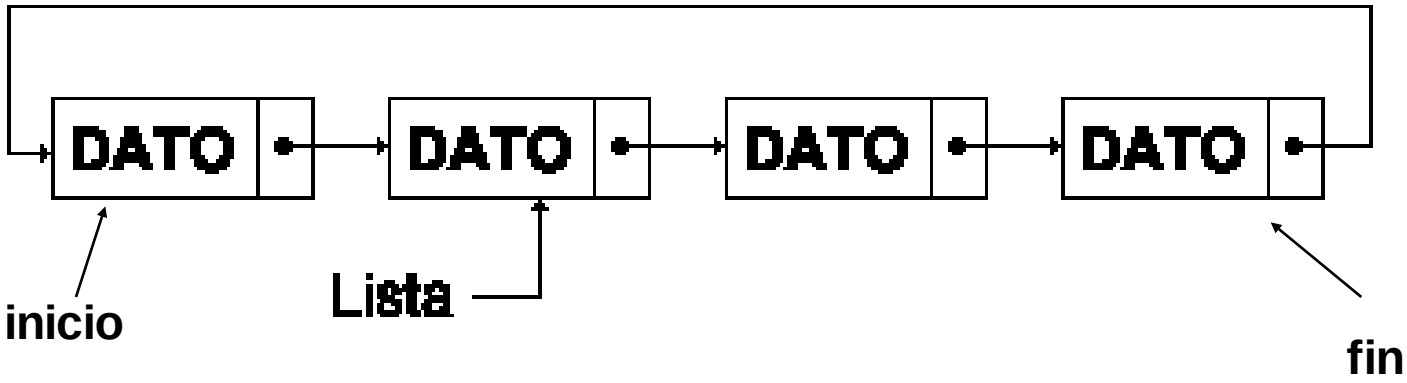
BOTÓN ELIMINAR

```
void Eliminar()
{ Nodo actual;
   boolean encontrado=false;
   actual=ini;
   while((actual!=null)&& (!encontrado)) //bucle de Busqueda
   {   encontrado=actual.codigo.equalsIgnoreCase(txtCodigo.getText().trim());
       if(!encontrado)
             actual=actual.sig;
   }
   //Realizando los enlaces
  if (actual != null)
    {  if (actual == ini)
       {   ini = actual.sig; // borrar el primero
           if(actual.sig!=null)
                 actual.sig.ant=null;
       }
       else if (actual.sig!=null) // No es el ultimo
              {   actual.ant.sig=actual.sig;
                  actual.sig.ant=actual.ant;
              }
            else
            {   actual.ant.sig=null; // el ultimo
                fin=actual.ant;  // moviendo el final
            }
       actual=null;
    }
}
```

Lista Circular

El ultimo elemento (cola) se enlaza al primer elemento (cabeza), de tal modo que la lista puede se recorrida de modo circular (“anillo”)

Una lista circular es una lista lineal en la que el último nodo a punta al primero.



Operaciones de una lista circular

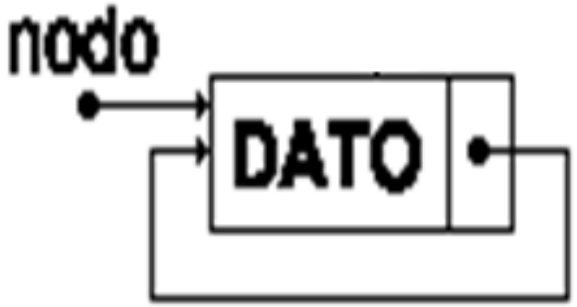
Las operaciones que se pueden realizar sobre las listas circulares :

- Añadir o insertar elementos.
- Buscar o localizar elementos.
- Borrar elementos.
- Moverse a través de la lista

Declaración de una lista circular

Los elementos de lista pueden ser de cualquier tipo, solo varia a las demas listas en el constructor, la referencia enlace en vez de inicializarla a null se inicializa para que apunte al mismo nodo, asi forma una lista circular de un solo nodo

```
public class Nodo {
      int dato;
      Nodo enlace;
      public Nodo(int num)
      {  dato=num;
          enlace=this;
       }
    }
```

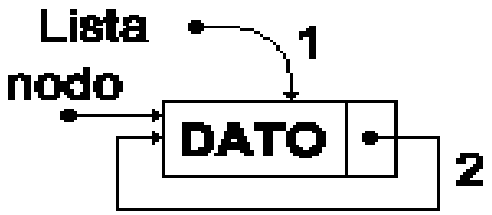


Insertar un elemento

Insertar elemento en la lista vacía

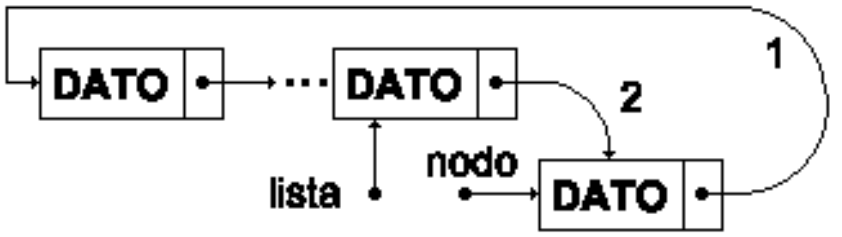
lista apunta a nodo.

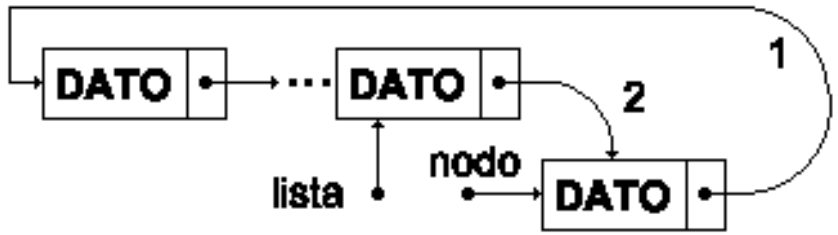
Lista→siguiente apunte a nodo.



Insertar elemento en una lista no vacía

1. Hacemos que nodo = siguiente apunte a lista = siguiente.
2. Después que lista = siguiente apunte a nodo.






```
public class Nodo {
      int dato;
      Nodo enlace;
      public Nodo(int num)
      {  dato=num;
         enlace=this;
       }
   }
```

```
Nodo InsertaFinal(Nodo lc, int num)
{     //Llamada al constructor
      Nodo nuevo=new Nodo(num);
       //Realizando los enlaces correspondientes
      if(lc!=null)
      {  nuevo.enlace=lc.enlace;
         lc.enlace=nuevo;
      }
      lc=nuevo;
      return lc;
}
```

RECORRIDO DE UNA LISTA CIRCULAR

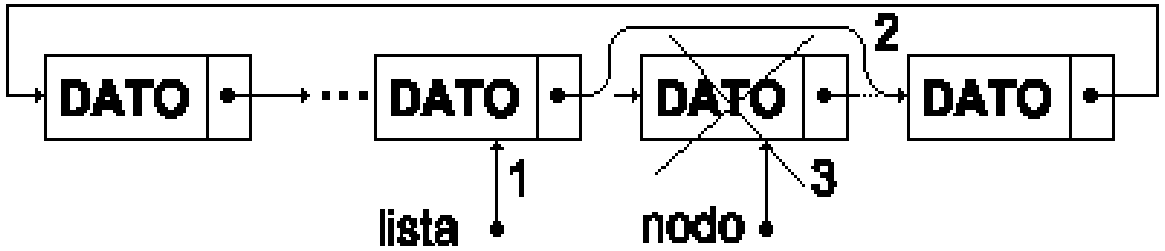
```
public class Nodo {
     int dato;
     Nodo enlace;
     public Nodo(int num)
     {  dato=num;
        enlace=this;
      }
   }
```

```
int Resumen()
{   int suma=0;
    Nodo p;
    if(lc!=null) // verificando si la lista es vacia
      {  p=lc.enlace;
          do
            {
              suma=suma+p.dato;  // sumando los datos
               p=p.enlace; // pasando a la siguiente posicion

           }while(p!=lc.enlace);

    return suma;
 }
```

Eliminar un elemento de la lista

Eliminar el único nodo de la lista.

1. lista = siguiente mientras lista = siguiente sea distinto de nodo.
2. Hacemos que lista = siguiente apunte a nodo = siguiente.
3. Eliminamos el nodo.



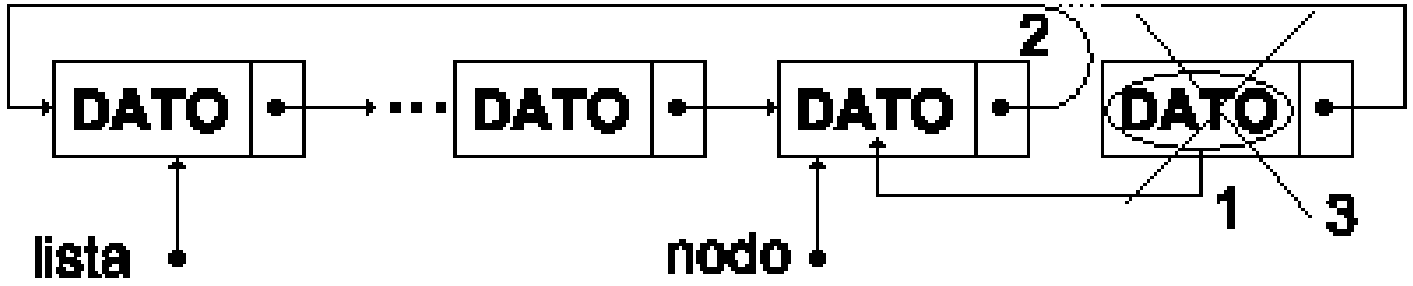
Eliminar un nodo en una lista circular con más de un elemento

1. Borramos el nodo apuntado por lista.
2. Hacemos que lista valga NULL.

Eliminar un elemento de la lista

Caso general

1. Copiamos el contenido del nodo = siguiente sobre el contenido de nodo.
2. Hacemos que nodo = siguiente apunte a nodo = siguiente = siguiente.
3. Eliminamos nodo = siguiente.
4. Si lista es el nodo = siguiente, hacemos lista = nodo.



```
public class Nodo {
    int dato;
    Nodo enlace;
    public Nodo(int num)
    {  dato=num;
       enlace=this;
     }
   }
```

```
void Eliminar(Nodo lc, int elem)
 { Nodo actual;
   boolean encontrado=false;
   actual=lc;
   while((actual.enlace!=lc)&& (!encontrado)) //bucle de Busqueda
       {   encontrado=actual.enlace.dato==elem;
          if(!encontrado)  actual=actual.enlace;
       }
   //Verificando el dato nuevamente
   encontrado=actual.enlace.dato==elem;
   //Enlace de nodo anterior con el siguiente
  if (encontrado)
      {  Nodo p;
        p=actual.enlace;  // Nodo a Eliminar
        if(lc==lc.enlace) lc=null; //Lista con un solo nodo
        else
        {  if(p==lc)  lc=actual; // Se borra el elemento referenciado por lc  el nuevo acceso a la lista es el anterior
          actual.enlace=p.enlace;
        }
    }
}
```

BUSCAR UN NODO CON ALGUNA CARACTERÍSTICA

```
public class Nodo {
      int dato;
      Nodo enlace;
      public Nodo(int num)
      {  dato=num;
         enlace=this;
       }
    }
```

```
Nodo Buscar(Nodo lc,int elem)
{       Nodo actual;
        boolean encontrado=false;
        actual=lc;
        //bucle de Busqueda
        while((actual.enlace!=lc)&& (!encontrado))
        {   encontrado=actual.enlace.dato==elem;
              if(!encontrado)
                    actual=actual.enlace;
        }
        // se retorna la referencia
        return actual.enlace;
}
```

EJEMPLO

Construya una aplicación que utilice listas circular y que permita realizar el registro de empleados donde se podrá guardar, mostrar, consultar, actualizar y eliminar el registro de empleado. Para todas estas operaciones considere el ingreso del código del empleado.

DECLARACIÓN DEL CLASE NODO

```
1   package listas_dobles;
2   import java.text.DecimalFormat;
3   import java.awt.Font;
4   import javax.swing.JOptionPane;
5   import javax.swing.table.DefaultTableModel;
6
7   public class listas_circulares extends javax.swing.JFrame {
8       //Declaracion de la Lista Circular
9      public class Nodo {
10        String codigo;
11        String nombre;
12        String apellidos;
13        String sexo;
14        float sueldo;
15        Nodo enlace;
16       public Nodo(String cod, String nom,String ape,String sex,float suel)
17        {   codigo=cod;
18           nombre=nom;
19           apellidos=ape;
20           sexo=sex;
21           sueldo=suel;
22           enlace=this; //enlace a si mismo
23        }
24       }
```

```
25          //Declaracion del formato de la tabla
26        DefaultTableModel miModelo;
27        String[] cabecera={"N°","Codigo","Nombres","Apellidos","Sexo","Sueldo"};
28        String[][] data={};
29      //Declaracion de variables locales
30         public Nodo lc;
31         public Nodo pFound;
32         int num=0;
33         //ant=null;
34         public listas_circulares() {
35             initComponents();
36             lc=pFound=null;
37             //Inicializando la Tabla
38             miModelo=new DefaultTableModel(data,cabecera);
39             tblRegistros.setModel(miModelo);
40         }
41         @SuppressWarnings("unchecked")
```

BOTÓN GUARDAR

```
271     private void btnGuardarActionPerformed(java.awt.event.ActionEvent evt) {
272         //Capturando la informacion de los objetos
273         String cod=txtCodigo.getText();
274         String nom=txtNombre.getText().toUpperCase();
            String ape=txtApellidos.getText().toUpperCase();;
276         String sex=cbxSexo.getSelectedItem().toString();
277         String suel=txtSueldo.getText();
278         //Creando el nodo de la Lista en memoria y colocando la informacion
279         lc=InsertaFinal(lc,cod,nom,ape,sex,Float.parseFloat(suel));
280         LimpiarEntradas();
281         //insertar(num,cod,nom,suel);
282         VerDatos();
283         Resumen();
284 }
```

```
316 Nodo InsertaFinal(Nodo lc,String cod,String nom,String ape,String sex,float suel)
317   {     Nodo nuevo=new Nodo(cod,nom,ape,sex,suel);
318          //Realizando los enlaces correspondientes
319         //nuevo.sig=inicio;
320         if(lc!=null)
321         {  nuevo.enlace=lc.enlace;
322            lc.enlace=nuevo;
323         }
324         lc=nuevo;
325         return lc;
326   }
```

```
private void btnConsultarActionPerformed(java.awt.event.ActionEvent evt) {
String cod=txtCodigo.getText();
    if(cod.equalsIgnoreCase("")) {
        JOptionPane.showMessageDialog(this,"Ingrese un codigo por favor");
    }
    else {
        //Llamada a la funcion que retorna la posicion del dato buscado
        pFound=Buscar(lc,cod);
        //Verificando el puntero pFound para mostrar la inf. buscada
        if(pFound!=null) {
            txtNombre.setText(pFound.nombre);
            txtApellidos.setText(pFound.apellidos);
            if(pFound.sexo.equalsIgnoreCase("MASCULINO"))
                cbxSexo.setSelectedIndex(2);
            else
                cbxSexo.setSelectedIndex(1);
            txtSueldo.setText(String.valueOf(pFound.sueldo));
            //Habilitamos los objetos para eliminar y actualizar
            Habilitar();

          } else {
            JOptionPane.showMessageDialog(this,"El código: "+cod + ", no esta en la Lista..");
        }
    }
}
```

BOTÓN CONSULTAR

```
Nodo Buscar(Nodo lc,String cod)
{   Nodo actual;
    boolean encontrado=false;
    actual=lc;
    //bucle de Busqueda
    while((actual.enlace!=lc)&& (!encontrado))
    {   encontrado=actual.enlace.codigo.equalsIgnoreCase(txtCodigo.getText().trim());
        if(!encontrado)
              actual=actual.enlace;
    }


       return actual.enlace;
}
```

BOTÓN ACTUALIZAR

```
    private void btnActualizarActionPerformed(java.awt.event.ActionEvent evt) {
        //Colocando la informacion en el puntero pFound
        pFound.codigo=txtCodigo.getText();
        pFound.nombre=txtNombre.getText().toUpperCase();
        pFound.apellidos=txtApellidos.getText().toUpperCase();
        pFound.sexo=cbxSexo.getSelectedItem().toString();
        pFound.sueldo=Float.parseFloat(txtSueldo.getText());
        LimpiarEntradas();
        Deshabilitar();
        VerDatos();
        Resumen();
}
```

```
    private void btnEliminarActionPerformed(java.awt.event.ActionEvent evt) {
        Eliminar();
        LimpiarEntradas();
        VerDatos();
        if(lc==null)
             JOptionPane.showMessageDialog(this,"La lista esta vacía");
        Deshabilitar();
        Resumen();
}
```

BOTÓN ELIMINAR

```
void Eliminar()
{ Nodo actual;
   boolean encontrado=false;
   actual=lc;
   //bucle de Busqueda
   while((actual.enlace!=lc)&& (!encontrado))
   {   encontrado=actual.enlace.codigo.equalsIgnoreCase(txtCodigo.getText().trim());
       if(!encontrado)
             actual=actual.enlace;
   }
   //Verificando el dato nuevamente
   encontrado=actual.enlace.codigo.equalsIgnoreCase(txtCodigo.getText().trim());
   //Enlace de nodo anterior con el siguiente
  if (encontrado)
    {  Nodo p;
       p=actual.enlace;  // Nodo a Eliminar
       if(lc==lc.enlace) lc=null; //Lista con un solo nodo
       else
       {  if(p==lc)
             lc=actual; // Se borra el elemento referenciado por lc
                        // el nuevo acceso a la lista es el anterior
          actual.enlace=p.enlace;
       }

    }
   VerDatos();
}
```

MÉTODO RESUMEN

```
void Resumen()
{ String nom="",acum="";
  float suma=0,mayor=-9999;
  float s;
  Nodo p;
  if(lc!=null)
  {   p=lc.enlace;
      do
      { s=p.sueldo;
        if(s>mayor)
        { mayor=s;
          nom=p.nombre+" "+p.apellidos;
        }
        suma=suma+s;
      p=p.enlace;
      }while(p!=lc.enlace);
  //Colocando la Información en los objetos
   txtNomMay.setText(nom);
  // Dandole formato al acumulado
   DecimalFormat df2 = new DecimalFormat("####.00");
   acum = df2.format(suma);
   txtAcumulado.setText(acum);
  }
 }
```

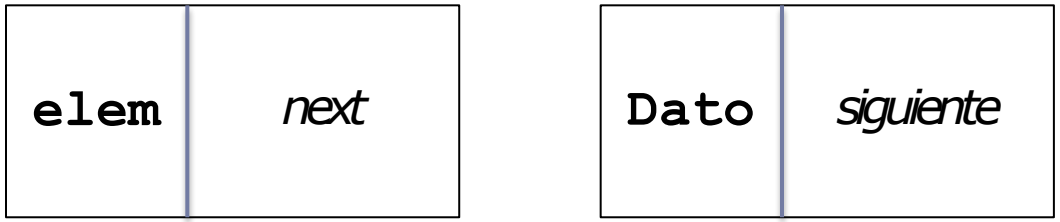
Estructuras de datos lineales

- Necesitamos definir una clase, **Nodo**, para representar cada ubicación. La clase Nodo tiene dos atributos:

  - **Dato o elem:** es la referencia a un elemento almacenado en la Lista. Es decir, almacena la dirección de memoria donde se almacena el elemento. El tipo de datos de elem debe ser del mismo tipo que los elementos de la Lista

  - **Next o siguiente:** es la referencia al nodo que almacena el siguiente elemento en la Lista. Su tipo de datos debe ser **Nodo**

Nodo

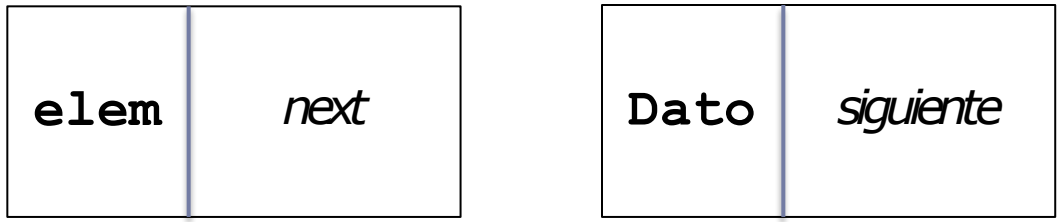


Estructuras de datos lineales

- **Especificación formal:**
  - Secuencia de elementos **{a1,a2,…,an}** donde cada elemento tiene un único predecesor (excepto el primero que no tiene predecesor) y un único sucesor (excepto el último que no tiene sucesor).

  - Las operaciones básicas de un TAD Lista son:
    - **Agregar** un elemento a la Lista
    - **Eliminar** un elemento de la Lista
    - **Consultar** un elemento de la Lista

Nodo



Estructuras de datos lineales

- **Especificación formal (operaciones de agregar):**

```
public interface IList {

    public void addFirst(String newElem);

    public void addLast(String newElem);

    public void insertAt(int index, String newElem);
```

Estructuras de datos lineales

- **Especificación formal (operaciones de consulta):**

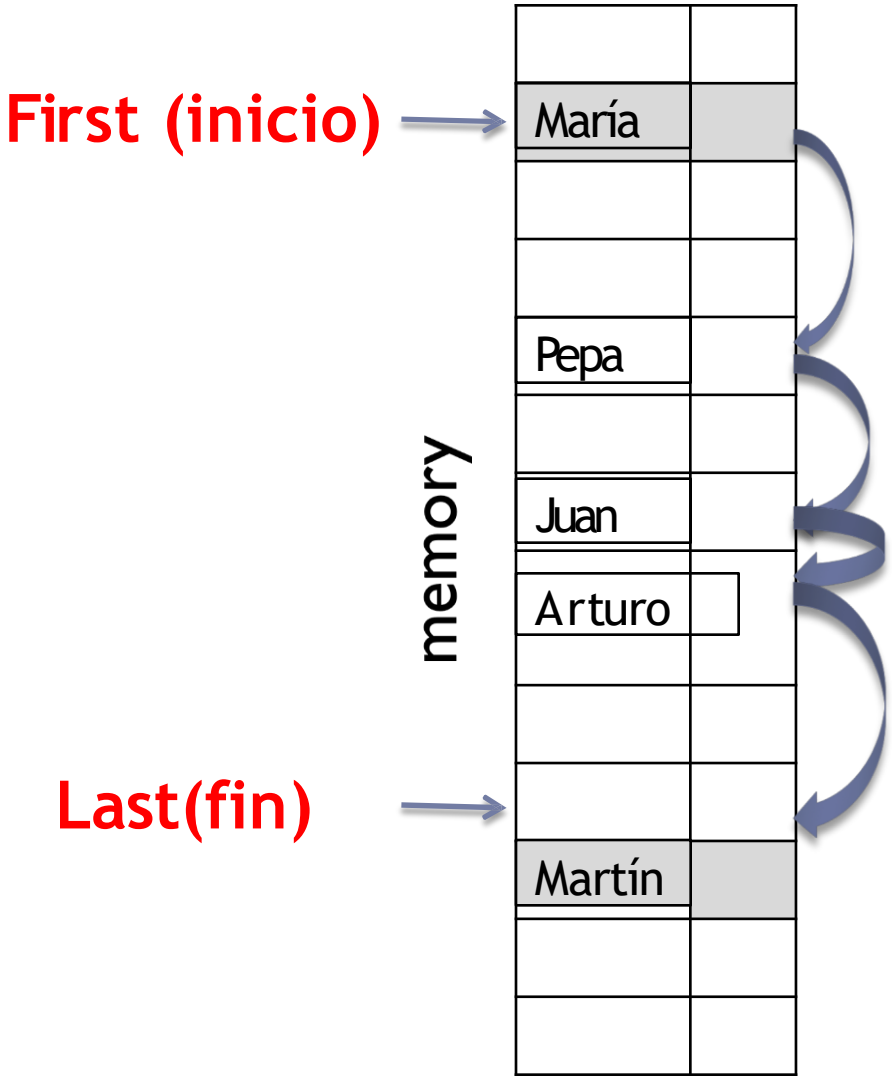
```
public boolean isEmpty();

public boolean contains(String elem);

public int getSize();

public int getIndexOf(String elem);

public String getFirst();

public String getLast();

public String getAt(int index);

public String toString();
```

Estructuras de datos lineales

- **Especificación formal (operaciones de borrado):**

```
public void removeFirst();

public void removeLast();

public void removeAll(String elem);

public void removeAt(int index);
```

Estructuras de datos lineales



Universidad Nacional Mayor de San Marcos
Universidad del Perú. Decana de América

Guía de Laboratorio

Universidad Nacional Mayor de San Marcos

Universidad del Perú. Decana de América

Universidad Nacional Mayor de San Marcos

Universidad del Perú. Decana de América