GUÍA DE LABORATORIO N° 02

Semana 04: Listas doblemente Enlazadas y Circulares

INTRODUCCION

En las Listas Doblemente enlazadas 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").
En la lista Circular simplemente enlazada el último elemento (cola) se enlaza al primer elemento (cabeza), de tal modo que la lista puede ser recorrida de modo circular ("anillo")

I. OBJETIVOS

1. Desarrollo de ejercicios de Manejo de Cadenas usando Swing WT.
2. Codificar, compilar y ejecutar los ejercicios de aplicación.
3. Conocer la estructura de un programa visual en Java.
4. Se espera que el estudiante asocie los conocimientos nuevos con la "nueva plantilla" que se propone.

II. EQUIPOS Y MATERIALES

- Computadora personal.
- Programa NetBeans
- Notas de los ejercicios resueltos en la clase.

III. METODOLOGIA Y ACTIVIDADES

- Codificar los ejercicios desarrollados en el aula.
- Presentar avances y ejecución de cada uno de los ejercicios al docente o jefe de práctica encargado para la calificación correspondiente.
- Guardar la carpeta de sus archivos a sus memorias.
- Apagar el computador y dejarla en buen estado al retirarse del laboratorio dejar todo en orden.

IV. OBSERVACION

- El estudiante deberá crear una carpeta de trabajo, a donde deberá direccionar su proyecto a crear.
- Se recomienda que el estudiante haya planteado los ejercicios de clase para que pueda comprender las soluciones que se proponen.

V. EJERCICIOS.

Construya una aplicación 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.

SOLUCION: UTILIZANDO LISTAS DOBLEMENTE ENLAZADAS

Paso 1: Construya el siguiente diseño:

Ahora váyase al editor de código y genere el siguiente código faltante:

```
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")
```

Aquí se muestran los métodos que tiene que crear antes ejecutar los códigos

Instrucciones de los métodos utilizados en el aplicativo

```
468    void Resumen()
469    { Nodo aux=ini;
470      String nom="",acum="";
471      float suma=0,mayor=-9999;
472      float s;
473      //Recorriendo la lista
474      while(aux!=null)
475       {  s=aux.sueldo;
476          if(s>mayor)
477          { mayor=s;
478            nom=aux.nombre+" "+aux.apellidos;
479          }
480          suma=suma+s;
481          aux=aux.sig;
482         }
483        //Colocando la Información en los objetos
484      txtNomMay.setText(nom);
485   // Dandole formato al acumulado
486    DecimalFormat df2 = new DecimalFormat("####.00");
487    acum = df2.format(suma);
488      txtAcumulado.setText(acum);
489    }
490    void Habilitar()
491    {  btnActualizar.setEnabled(true);
492       btnEliminar.setEnabled(true);
493       btnGuardar.setEnabled(false);
494    }
495    void Deshabilitar()
496    {  btnActualizar.setEnabled(false);
497       btnEliminar.setEnabled(false);
498       btnGuardar.setEnabled(true);
499    }
500    void LimpiarEntradas()
501     {  txtCodigo.setText("") ;
502        txtNombre.setText("") ;
503        txtSueldo.setText("") ;
504        txtApellidos.setText("") ;
505        cbxSexo.setSelectedIndex(0);
506        txtCodigo.requestFocus() ;
507     }
461     void vaciar_tabla()
462     { //Obteniendo el numero de filas de la tabla
463         int filas=tblRegistros.getRowCount();
464         for(int i=0;i<filas;i++)
465             miModelo.removeRow(0);
466     }
```

```
416     void VerDatos(int ind)
417     {   //Variable para recorrer la lista
418         String cod,nom,ape,se,su;
419         float s;
420         switch(ind)
421         { case 1: // Recorrido hacia adelante
422                 {  vaciar_tabla();
423                    Nodo aux=ini;
424                   num=0;
425                    while(aux!=null)
426                        {   cod=aux.codigo;
427                            nom=aux.nombre;
428                            ape=aux.apellidos;
429                            se=aux.sexo;
430                            //dando Formato al sueldo
431                            DecimalFormat df2 = new DecimalFormat("####.00");
432                            su = df2.format(aux.sueldo);
433                            num++;
434                            Object[] fila={num,cod,nom,ape,se,su};
435                            miModelo.addRow(fila);
436                            aux=aux.sig;
437                        }
438                   } break;
439
440          case 2: // Recorrido hacia atras
441                 { vaciar_tabla();
442                   Nodo aux=fin;
443                   num=0;
444                    while(aux!=null)
445                        {   cod=aux.codigo;
446                            nom=aux.nombre;
447                            ape=aux.apellidos;
448                            se=aux.sexo;
449                            //dando Formato al sueldo
450                            DecimalFormat df2 = new DecimalFormat("####.00");
451                            su = df2.format(aux.sueldo);
452                            num++;
453                            Object[] fila={num,cod,nom,ape,se,su};
454                            miModelo.addRow(fila);
455                            aux=aux.ant;
456                        }
                    }break;
458         }
459        }
```

Instrucciones en los métodos para las operaciones de la lista enlazada

```
318   Nodo Buscar(Nodo inicio,String cod)
319   {     Nodo pos=inicio;
320               //Recorriendo la lista para encontrar la informacion
321               while(pos!=null && !cod.equalsIgnoreCase(pos.codigo))
322                   pos=pos.sig;
323         return pos;
324   }
325   Nodo InsertaFinal(Nodo inicio,String cod,String nom,String ape,String sex,float suel)
326    {     Nodo nuevo=new Nodo(cod,nom,ape,sex,suel);
327           //Realizando los enlaces correspondientes
328           nuevo.sig=inicio;
329           if(inicio==null)
330           {  fin=nuevo;
331              fin.sig=null;
332           }
333           if(inicio!=null)
334               inicio.ant=nuevo;
335
336           inicio=nuevo;
337           return inicio;
338    }
```

```
339     void Eliminar()
340   { Nodo actual;
341      boolean encontrado=false;
342      actual=ini;
343      while((actual!=null)&& (!encontrado)) //bucle de Busqueda
344      {   encontrado=actual.codigo.equalsIgnoreCase(txtCodigo.getText().trim());
345          if(!encontrado)
346                actual=actual.sig;
347      }
348      //Realizando los enlaces
349     if (actual != null)
350       {  if (actual == ini)
351          {   ini = actual.sig; // borrar el primero
352              if(actual.sig!=null)
353                   actual.sig.ant=null;
354          }
355          else if (actual.sig!=null) // No es el ultimo
356                  {   actual.ant.sig=actual.sig;
357                      actual.sig.ant=actual.ant;
358                  }
359                else
360                {   actual.ant.sig=null; // el ultimo
361                    fin=actual.ant;  // moviendo el final
362                }
363          actual=null;
364       }
365   }
```

Instrucciones del botón Guardar

```
285
286     private void btnGuardarActionPerformed(java.awt.event.ActionEvent evt) {
287         //Capturando la informacion de los objetos
288         String cod=txtCodigo.getText();
289         String nom=txtNombre.getText().toUpperCase();
290         String ape=txtApellidos.getText().toUpperCase();;
291         String sex=cbxSexo.getSelectedItem().toString();
292         String suel=txtSueldo.getText();
293        //Creando el nodo de la Lista en memoria y colocando la informacion
294         ini=InsertaFinal(ini,cod,nom,ape,sex,Float.parseFloat(suel));
295         LimpiarEntradas();
296         VerDatos(1);
297         Resumen();
298 }
```

Instrucciones del Botón Actualizar

```
367     private void btnActualizarActionPerformed(java.awt.event.ActionEvent evt) {
368         //Colocando la informacion en el puntero pFound
369         pFound.codigo=txtCodigo.getText();
370         pFound.nombre=txtNombre.getText().toUpperCase();
371         pFound.apellidos=txtApellidos.getText().toUpperCase();
372         pFound.sexo=cbxSexo.getSelectedItem().toString();
373         pFound.sueldo=Float.parseFloat(txtSueldo.getText());
374         LimpiarEntradas();
375         Deshabilitar();
376         VerDatos(1);
377         Resumen();
378 }
```

Instrucciones del Botón Consultar

```
385     private void btnConsultarActionPerformed(java.awt.event.ActionEvent evt) {
386      String cod=txtCodigo.getText();
387         if(cod.equalsIgnoreCase("")) {
388             JOptionPane.showMessageDialog(this,"Ingrese un codigo por favor");
389         } else {
390             //Llamada a la funcion que retorna la posicion del dato buscado
391             pFound=Buscar(ini,cod);
392             //Verificando el puntero pFound para mostrar la inf. buscada
393             if(pFound!=null) {
394                 txtNombre.setText(pFound.nombre);
395                 txtApellidos.setText(pFound.apellidos);
396                 if(pFound.sexo.equalsIgnoreCase("MASCULINO"))
397                     cbxSexo.setSelectedIndex(2);
398                 else
399                     cbxSexo.setSelectedIndex(1);
400                 txtSueldo.setText(String.valueOf(pFound.sueldo));
401                 //Habilitamos los objetos para eliminar y actualizar
402                 Habilitar();
403             } else {
404                 JOptionPane.showMessageDialog(this,"El código: "+cod + ", no esta en la Lista..");
405             }
406         }
407     }
```

Instrucciones del Botón Eliminar

```
305     private void btnEliminarActionPerformed(java.awt.event.ActionEvent evt) {
306         Eliminar();
307         LimpiarEntradas();
308         VerDatos(1);
309         if(ini==null)
310              JOptionPane.showMessageDialog(this,"La lista esta vacía");
311         Deshabilitar();
312         Resumen();
313 }
```

Instrucciones del Botón Restaurar

```
300     private void btnReestaurarActionPerformed(java.awt.event.ActionEvent evt) {
301         LimpiarEntradas();
302         Deshabilitar();
303     }
```

Instrucciones del Adelante-Atras

```
409     private void btnAd_AtActionPerformed(java.awt.event.ActionEvent evt) {
410         VerDatos(1);
411     }
```

Instrucciones del Atrás - Adelante

```
314
315     private void btnAt_AdActionPerformed(java.awt.event.ActionEvent evt) {
316         VerDatos(2);
317     }
```

Instrucciones del Evento Opened del Window

```
380     private void formWindowOpened(java.awt.event.WindowEvent evt) {
381         Deshabilitar();
382     }
```

Luego de terminar Shift+F6 y ejecutarse el aplicativo deberá verse de la siguiente manera:

SOLUCION: UTILIZANDO LISTAS CIRCULARES SIMPLES

Ahora váyase al editor de código y genere el siguiente código faltante:

```
package listas_dobles;
import java.text.DecimalFormat;
import java.awt.Font;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;

public class listas_circulares extends javax.swing.JFrame {
    //Declaracion de la Lista Circular
    public class Nodo {
       String codigo;
       String nombre;
       String apellidos;
       String sexo;
       float sueldo;
       Nodo enlace;
     public Nodo(String cod, String nom,String ape,String sex,float suel)
      {  codigo=cod;
         nombre=nom;
         apellidos=ape;
         sexo=sex;
         sueldo=suel;
         enlace=this; //enlace a si mismo
      }
     }
      //Declaracion del formato de la tabla
   DefaultTableModel miModelo;
   String[] cabecera={"N°","Codigo","Nombres","Apellidos","Sexo","Sueldo"};
   String[][] data={};
 //Declaracion de variables locales
    public Nodo lc;
    public Nodo pFound;
    int num=0;
    //ant=null;
    public listas_circulares() {
        initComponents();
        lc=pFound=null;
        //Inicializando la Tabla
        miModelo=new DefaultTableModel(data,cabecera);
        tblRegistros.setModel(miModelo);
    }
    @SuppressWarnings("unchecked")
```

Aquí se muestran los métodos para el manejo de los nodos en una lista circular

```
302    Nodo Buscar(Nodo lc,String cod)
303    {   Nodo actual;
304        boolean encontrado=false;
305        actual=lc;
306        //bucle de Busqueda
307        while((actual.enlace!=lc)&& (!encontrado))
308        {   encontrado=actual.enlace.codigo.equalsIgnoreCase(txtCodigo.getText().trim());
309            if(!encontrado)
310                  actual=actual.enlace;
311        }
312
313
314           return actual.enlace;
315    }
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     }
```

```
327   void Eliminar()
328   { Nodo actual;
329      boolean encontrado=false;
330      actual=lc;
331      //bucle de Busqueda
332      while((actual.enlace!=lc)&& (!encontrado))
333      {   encontrado=actual.enlace.codigo.equalsIgnoreCase(txtCodigo.getText().trim());
334          if(!encontrado)
335                actual=actual.enlace;
336      }
337      //Verificando el dato nuevamente
338      encontrado=actual.enlace.codigo.equalsIgnoreCase(txtCodigo.getText().trim());
339      //Enlace de nodo anterior con el siguiente
340     if (encontrado)
341       {  Nodo p;
342          p=actual.enlace;  // Nodo a Eliminar
343          if(lc==lc.enlace) lc=null; //Lista con un solo nodo
344          else
345          {  if(p==lc)
346                lc=actual; // Se borra el elemento referenciado por lc
347                           // el nuevo acceso a la lista es el anterior
348             actual.enlace=p.enlace;
349          }
350
351       }
352      VerDatos();
353   }
```

Instrucciones de los métodos utilizados en el aplicativo

```
400  void VerDatos(){
401   //Variable para recorrer la lista
402   String cod,nom,ape,se,su;
403   float s;
404   vaciar_tabla(); // limpiando la tabla
405   Nodo p; // puntero adicional para el recorrido
406   if(lc!=null)
407     {  num=0;
408        p=lc.enlace;
409        do
410        { cod=p.codigo;
411          nom=p.nombre;
412          ape=p.apellidos;
413          se=p.sexo;
414          //dando Formato al sueldo
415          DecimalFormat df2 = new DecimalFormat("####.00");
416          su = df2.format(p.sueldo);
417          num++;
418          Object[] fila={num,cod,nom,ape,se,su};
419          miModelo.addRow(fila);
420
421                          p=p.enlace;
422       }while(p!=lc.enlace);
423    }
424 }
```

```
426  void Resumen()
427  { String nom="",acum="";
428    float suma=0,mayor=-9999;
429    float s;
430    Nodo p;
431    if(lc!=null)
432    {  p=lc.enlace;
433       do
434       { s=p.sueldo;
435         if(s>mayor)
436         { mayor=s;
437           nom=p.nombre+" "+p.apellidos;
438         }
439         suma=suma+s;
440       p=p.enlace;
441       }while(p!=lc.enlace);
442    //Colocando la Información en los objetos
443     txtNomMay.setText(nom);
444    // Dandole formato al acumulado
445     DecimalFormat df2 = new DecimalFormat("####.00");
446     acum = df2.format(suma);
447     txtAcumulado.setText(acum);
448    }
449  }
```

```
451  void Habilitar()
452  {  btnActualizar.setEnabled(true);
453     btnEliminar.setEnabled(true);
454     btnGuardar.setEnabled(false);
455  }
456  void Deshabilitar()
457  {  btnActualizar.setEnabled(false);
458     btnEliminar.setEnabled(false);
459     btnGuardar.setEnabled(true);
460  }
```

```
461   void LimpiarEntradas()
462   {  txtCodigo.setText("") ;
463      txtNombre.setText("") ;
464      txtSueldo.setText("") ;
465      txtApellidos.setText("") ;
466      cbxSexo.setSelectedIndex(0);
467      txtCodigo.requestFocus() ;
468   }
469   void vaciar_tabla()
470   {     //Obteniendo el numero de filas de la tabla
471       int filas=tblRegistros.getRowCount();
472       for(int i=0;i<filas;i++)
473           miModelo.removeRow(0);
474   }
```

Instrucciones del 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();
276         String ape=txtApellidos.getText().toUpperCase();;
277         String sex=cbxSexo.getSelectedItem().toString();
278         String suel=txtSueldo.getText();
279         //Creando el nodo de la Lista en memoria y colocando la informacion
280         lc=InsertaFinal(lc,cod,nom,ape,sex,Float.parseFloat(suel));
281         LimpiarEntradas();
282         //insertar(num,cod,nom,suel);
283         VerDatos();
284         Resumen();
    }
```

Instrucciones del Botón Actualizar

```
353     private void btnActualizarActionPerformed(java.awt.event.ActionEvent evt) {
354         //Colocando la informacion en el puntero pFound
355         pFound.codigo=txtCodigo.getText();
356         pFound.nombre=txtNombre.getText().toUpperCase();
357         pFound.apellidos=txtApellidos.getText().toUpperCase();
358         pFound.sexo=cbxSexo.getSelectedItem().toString();
359         pFound.sueldo=Float.parseFloat(txtSueldo.getText());
360         LimpiarEntradas();
361         Deshabilitar();
362         VerDatos();
363         Resumen();
364   }
```

Instrucciones del Botón Eliminar

```
291     private void btnEliminarActionPerformed(java.awt.event.ActionEvent evt) {
292         Eliminar();
293         LimpiarEntradas();
294         VerDatos();
295         if(lc==null)
296             JOptionPane.showMessageDialog(this,"La lista esta vacia");
297         Deshabilitar();
298         Resumen();
299   }
```

Instrucciones del Botón Restaurar

```
286     private void btnReestaurarActionPerformed(java.awt.event.ActionEvent evt) {
287         LimpiarEntradas();
288         Deshabilitar();
289     }
```

Instrucciones del Botón Consultar

```
371     private void btnConsultarActionPerformed(java.awt.event.ActionEvent evt) {
372     String cod=txtCodigo.getText();
373         if(cod.equalsIgnoreCase("")) {
374             JOptionPane.showMessageDialog(this,"Ingrese un codigo por favor");
375         }
376         else {
377             //Llamada a la funcion que retorna la posicion del dato buscado
378             pFound=Buscar(lc,cod);
379             //Verificando el puntero pFound para mostrar la inf. buscada
380             if(pFound!=null) {
381                 txtNombre.setText(pFound.nombre);
382                 txtApellidos.setText(pFound.apellidos);
383                 if(pFound.sexo.equalsIgnoreCase("MASCULINO"))
384                     cbxSexo.setSelectedIndex(2);
385                 else
386                     cbxSexo.setSelectedIndex(1);
387                 txtSueldo.setText(String.valueOf(pFound.sueldo));
388                 //Habilitamos los objetos para eliminar y actualizar
389                 Habilitar();
390 
391               } else {
392                 JOptionPane.showMessageDialog(this,"El código: "+cod + ", no esta en la Lista..");
393             }
394         }
395     }
```

Instrucciones del Evento Opened del Window

Luego de terminar Shift+F6 y ejecutarse el aplicativo deberá verse de la siguiente manera:

EJERCICIO PROPUESTO

Construya la solución del siguiente aplicativo usando listas doblemente enlazadas y listas circulares. Su programa también tendrá la opción de consultar, actualizar y eliminar registros.

Construya un aplicativo que permita guardar en una lista la información: Nombres, Apellido Paterno, Apellido Materno, Sueldo base, Ventas realizadas, Estado civil, Número de Hijos y Sueldo Neto.

El programa debe calcular para cada empleado el sueldo neto mediante la relación siguiente:

Sueldo Neto = Sueldo Base + Comisión por Ventas – Descuento por Impuesto – Descuento por Seguro.

Donde: la comisión por ventas es el 5% de las ventas realizadas
El descuento por seguro se obtiene como sigue:

- Si el empleado es soltero el descuento es de S/. 100.
- Si el empleado es casado sin hijos el descuento es de S/. 120.
- Si el empleado es casado con hijos el descuento es de S/. 50 + S/. 70 por cada hijo.

Suponga que llamamos TA a la suma del Sueldo base + comisión por ventas, entonces el Descuento por el Impuesto (DI) se obtiene como sigue:

Rango del TA	Monto de DI
0 a 1500	0
1500 a 2300	3% del TA
2301 a 3000	4% del TA
3001 a mas	6% del TA

El programa debe mostrar el monto total que la empresa debe pagar por concepto de sueldos, el monto total por comisiones de ventas, el monto total de los descuentos por impuesto y por seguro.