INSTITUTO SAN ISIDRO

Curso de Java a distancia

Clase 16: Estructuras dinámicas: Listas genéricas

Continuando con el tema de listas trabajaremos con las listas genéricas. Una lista se comporta como genérica cuando las inserciones y extracciones se realizan en cualquier parte de la lista. Codificaremos una serie de métodos para administrar listas genéricas.

Métodos a desarrollar:

Inserta un nodo en la posición (pos) y con la información que hay en el parámetro x.

```
void insertar(int pos, int x)
```

Extrae la información del nodo de la posición indicada (pos). Se debe eliminar el nodo.

```
int extraer(int pos)
```

Borra el nodo de la posición (pos).

```
void borrar(int pos)
```

Intercambia las informaciones de los nodos de las posiciones pos1 y pos2.

```
void intercambiar(int pos1,int pos2)
```

Retorna el valor del nodo con mayor información.

```
int mayor()
```

Retorna la posición del nodo con mayor información.

```
int posMayor()
```

Retorna la cantidad de nodos de la lista.

```
int cantidad()
```

Debe retornar true si la lista está ordenada de menor a mayor, false en caso contrario.

```
boolean ordenada()
```

Debe retornar true si existe la información que llega en el parámetro, false en caso contrario.

boolean existe(int info)

El método vacía debe retornar true si está vacía y false si no lo está.

boolean vacia()

Programa:

```
public class ListaGenerica {

   class Nodo {
      int info;
      Nodo sig;
   }

   private Nodo raiz;

   public ListaGenerica () {
      raiz=null;
   }

   void insertar (int pos, int x)
   {
      if (pos <= cantidad () + 1)    {
         Nodo nuevo = new Nodo ();
         nuevo.info = x;
         if (pos == 1){
            nuevo.sig = raiz;
            raiz = nuevo;
         } else
            if (pos == cantidad () + 1)    {
               Nodo reco = raiz;
               while (reco.sig != null) {
                  reco = reco.sig;
               }
               reco.sig = nuevo;
               nuevo.sig = null;
            } else {
               Nodo reco = raiz;
               for (int f = 1 ; f <= pos - 2 ; f++)
                  reco = reco.sig;
               Nodo siguiente = reco.sig;
               reco.sig = nuevo;
               nuevo.sig = siguiente;
            }
      }
   }
```

```
public int extraer (int pos) {
   if (pos <= cantidad ())    {
      int informacion;
      if (pos == 1) {
         informacion = raiz.info;
         raiz = raiz.sig;
      } else {
         Nodo reco;
         reco = raiz;
         for (int f = 1 ; f <= pos - 2 ; f++)
            reco = reco.sig;
         Nodo prox = reco.sig;
         reco.sig = prox.sig;
         informacion = prox.info;
      }
      return informacion;
   }
   else
      return Integer.MAX_VALUE;
}

public void borrar (int pos)
{
   if (pos <= cantidad ())    {
      if (pos == 1) {
         raiz = raiz.sig;
      } else {
         Nodo reco;
         reco = raiz;
         for (int f = 1 ; f <= pos - 2 ; f++)
            reco = reco.sig;
         Nodo prox = reco.sig;
         reco.sig = prox.sig;
      }
   }
}

public void intercambiar (int pos1, int pos2) {
   if (pos1 <= cantidad () && pos2 <= cantidad ())    {
      Nodo reco1 = raiz;
      for (int f = 1 ; f < pos1 ; f++)
         reco1 = reco1.sig;
      Nodo reco2 = raiz;
      for (int f = 1 ; f < pos2 ; f++)
         reco2 = reco2.sig;
      int aux = reco1.info;
      reco1.info = reco2.info;
      reco2.info = aux;
```

```
    }
}

public int mayor () {
    if (!vacia ()) {
        int may = raiz.info;
        Nodo reco = raiz.sig;
        while (reco != null) {
            if (reco.info > may)
                may = reco.info;
            reco = reco.sig;
        }
        return may;
    }
    else
        return Integer.MAX_VALUE;
}

public int posMayor() {
    if (!vacia ())    {
        int may = raiz.info;
        int x=1;
        int pos=x;
        Nodo reco = raiz.sig;
        while (reco != null){
            if (reco.info > may) {
                may = reco.info;
                pos=x;
            }
            reco = reco.sig;
            x++;
        }
        return pos;
    }
    else
        return Integer.MAX_VALUE;
}

public int cantidad ()
{
    int cant = 0;
    Nodo reco = raiz;
    while (reco != null) {
        reco = reco.sig;
        cant++;
    }
    return cant;
}
```

```
public boolean ordenada() {
   if (cantidad()>1) {
      Nodo reco1=raiz;
      Nodo reco2=raiz.sig;
      while (reco2!=null) {
         if (reco2.info<reco1.info) {
            return false;
         }
         reco2=reco2.sig;
         reco1=reco1.sig;
      }
   }
   return true;
}

public boolean existe(int x) {
   Nodo reco=raiz;
   while (reco!=null) {
      if (reco.info==x)
         return true;
      reco=reco.sig;
   }
   return false;
}

public boolean vacia ()
{
   if (raiz == null)
      return true;
   else
      return false;
}

public void imprimir ()
{
   Nodo reco = raiz;
   while (reco != null) {
      System.out.print (reco.info + "-");
      reco = reco.sig;
   }
   System.out.println();
}

public static void main(String[] ar) {
   ListaGenerica lg=new ListaGenerica();
   lg.insertar (1, 10);
   lg.insertar (2, 20);
```

```
        lg.insertar (3, 30);
        lg.insertar (2, 15);
        lg.insertar (1, 115);
        lg.imprimir ();
        System.out.println ("Luego de Borrar el primero");
        lg.borrar (1);
        lg.imprimir ();
        System.out.println ("Luego de Extraer el segundo");
        lg.extraer (2);
        lg.imprimir ();
        System.out.println ("Luego de Intercambiar el primero con el tercero");
        lg.intercambiar (1, 3);
        lg.imprimir ();
        if (lg.existe(20))
          System.out.println("Se encuentra el 20 en la lista");
        else
          System.out.println("No se encuentra el 20 en la lista");
        System.out.println("La posición del mayor es:"+lg.posMayor());
        if (lg.ordenada())
          System.out.println("La lista está ordenada de menor a mayor");
        else
          System.out.println("La lista no está ordenada de menor a mayor");
    }
}
```

Para insertar en una determinada posición dentro de la lista:

```
   void insertar (int pos, int x)
```

Primero con un if verificamos que exista esa posición en la lista (por ejemplo si la lista tiene 4 nodos podemos insertar hasta la posición 5, es decir uno más allá del último):

```
      if (pos <= cantidad () + 1)    {
```

Si ingresa al if ya podemos crear el nodo:

```
         Nodo nuevo = new Nodo ();
         nuevo.info = x;
```

Ahora debemos analizar si la inserción es al principio de la lista, al final o en medio ya que los enlaces varían según donde se lo inserta.
Para saber si se inserta al principio de la lista preguntamos si en pos llega un 1:

```
         if (pos == 1){
```

Si llega un 1 luego enlazamos el puntero sig del nodo que creamos con la dirección del primer nodo de la lista (raiz apunta siempre al primer nodo de la lista) y luego desplazamos raiz al nodo que acabamos de crear:

```
            nuevo.sig = raiz;
            raiz = nuevo;
```

Si no se inserta al principio de la lista preguntamos si se inserta al final:

```
if (pos == cantidad () + 1)    {
```

En caso de insertarse al final recorremos la lista hasta el último nodo:

```
Nodo reco = raiz;
while (reco.sig != null) {
   reco = reco.sig;
}
```

y enlazamos el puntero sig del último nodo de la lista con la dirección del nodo que acabamos de crear (disponemos en sig del nodo creado el valor null ya que no hay otro nodo más adelante)

```
reco.sig = nuevo;
nuevo.sig = null;
```

Si no se inserta al principio o al final significa que tenemos que insertar en medio de la lista. Disponemos un for donde avanzamos un puntero auxiliar y nos detenemos una posición antes a donde tenemos que insertarlo:

```
for (int f = 1 ; f <= pos - 2 ; f++)
   reco = reco.sig;
```

Disponemos otro puntero auxiliar que apunte al nodo próximo a donde está apuntando reco. Ahora enlazamos el puntero sig del nodo apuntado por reco con la dirección del nodo creado y el puntero sig del nodo creado con la dirección del nodo siguiente:

```
Nodo siguiente = reco.sig;
reco.sig = nuevo;
nuevo.sig = siguiente;
```

El método extraer recibe como parámetro la posición del nodo a extraer:

```
public int extraer (int pos) {
```

Primero verificamos que la posición exista en la lista:

```
if (pos <= cantidad ())    {
```

En caso que exista verificamos si el nodo a extraer es el primero de la lista (este análisis debe hacerse ya que si es el primero de la lista se modifica el puntero raiz):

```
if (pos == 1) {
```

Si es el primero guardamos en una variable auxiliar la información del nodo y avanzamos el puntero raiz:

```
informacion = raiz.info;
raiz = raiz.sig;
```

Si el nodo a extraer no está al principio de la lista avanzamos con una estructura repetitiva hasta el nodo anterior a extraer:

```
for (int f = 1 ; f <= pos - 2 ; f++)
   reco = reco.sig;
```

Luego definimos otro puntero auxiliar y lo disponemos en el siguiente nodo a donde está apuntando reco:

```
Nodo prox = reco.sig;
```

Ahora enlazamos el puntero sig del nodo apuntado por reco al nodo siguiente del nodo apuntado por prox (es decir el nodo apuntado por prox queda fuera de la lista):

```
reco.sig = prox.sig;
```

El método borrar es muy similar al método extraer con la diferencia de que no retorna valor:

```
public void borrar (int pos)

{
   if (pos <= cantidad ())    {
      if (pos == 1) {
         raiz = raiz.sig;
      } else {
         Nodo reco;
         reco = raiz;
         for (int f = 1 ; f <= pos - 2 ; f++)
            reco = reco.sig;
         Nodo prox = reco.sig;
         reco.sig = prox.sig;
      }
   }
}
```

El método intercambiar recibe dos enteros que representan las posiciones de los nodos que queremos intercambiar sus informaciones:

```
public void intercambiar (int pos1, int pos2) {
```

Mediante un if verificamos que las dos posiciones existan en la lista:

```
   if (pos1 <= cantidad () && pos2 <= cantidad ())    {
```

Definimos un puntero auxiliar llamado reco1, lo inicializamos con la dirección del primer nodo y mediante un for avanzamos hasta la posición almacenada en pos1:

```
Nodo reco1 = raiz;
for (int f = 1 ; f < pos1 ; f++)
   reco1 = reco1.sig;
```

De forma similar con un segundo puntero auxiliar avanzamos hasta la posición indicada por pos2:

```
Nodo reco2 = raiz;
for (int f = 1 ; f < pos2 ; f++)
   reco2 = reco2.sig;
```

Por último intercambiamos las informaciones que almacenan cada nodo:

```
int aux = reco1.info;
reco1.info = reco2.info;
reco2.info = aux;
```

El método que retorna el mayor de la lista:

```
public int mayor () {
```

Verificamos que la lista no esté vacía:

```
if (!vacia ())    {
```

Suponemos que el mayor es el primero de la lista e inicializamos un puntero auxiliar con la dirección del segundo nodo de la lista:

```
int may = raiz.info;
Nodo reco = raiz.sig;
```

Mediante una estructura repetitiva recorremos toda la lista:

```
while (reco != null) {
```

Cada vez que encontramos un nodo con información mayor que la variable may la actualizamos con este nuevo valor y avanzamos el puntero reco para visitar el siguiente nodo:

```
if (reco.info > may)
   may = reco.info;
reco = reco.sig;
```

Fuera de la estructura repetitiva retornamos el mayor:

```
return may;
```

El método que retorna la posición del mayor es similar al anterior con la salvedad que debemos almacenar en otro auxiliar la posición donde se almacena el mayor:

```
public int posMayor() {
   if (!vacia ())    {
      int may = raiz.info;
      int x=1;
      int pos=x;
      Nodo reco = raiz.sig;
      while (reco != null){
         if (reco.info > may) {
            may = reco.info;
            pos=x;
         }
         reco = reco.sig;
         x++;
      }
```

```
            return pos;
        }
        else
            return Integer.MAX_VALUE;
    }
```

El método que debe retornar si está ordenada la lista de menor a mayor es:

```
    public boolean ordenada() {
```

Lo primero que verificamos si la lista tiene más de un nodo significa que debemos controlarla:

```
        if (cantidad()>1) {
```

Disponemos dos punteros auxiliares con las direcciones del primer y segundo nodo de la lista:

```
            Nodo reco1=raiz;
            Nodo reco2=raiz.sig;
```

Mediante un while mientras no se finaliza la lista:

```
            while (reco2!=null) {
```

controlamos si la información del segundo nodo es menor al nodo anterior significa que la lista no está ordenada y podemos parar el análisis retornando un false

```
                if (reco2.info<reco1.info) {
                    return false;
```

Dentro del while avanzamos los dos punteros a sus nodos siguientes respectivamente.

```
                reco2=reco2.sig;
                reco1=reco1.sig;
```

Fuera del while retornamos true indicando que la lista está ordenada de menor a mayor

```
        return true;
```

El método existe:

```
    public boolean existe(int x) {
```

Mediante un while recorremos la la lista:

```
        Nodo reco=raiz;
        while (reco!=null) {
```

y en cada nodo que visitamos controlamos si el parámetro x es igual a la información del nodo, en caso afirmativo salimos del método retornando true:

```
            if (reco.info==x)
                return true;
            reco=reco.sig;
```

Fuera del while retornamos false indicando que ningún nodo coincide con el parámetro x:

```
return false;
```

Problemas propuestos

1. Plantear una clase para administrar una lista genérica implementando los siguientes métodos:
   a) Insertar un nodo al principio de la lista.
   b) Insertar un nodo al final de la lista.
   c) Insertar un nodo en la segunda posición. Si la lista está vacía no se inserta el nodo.
   d) Insertar un nodo en la ante última posición.
   e) Borrar el primer nodo.
   f) Borrar el segundo nodo.
   g) Borrar el último nodo.
   h) Borrar el nodo con información mayor.

Solución

```
public class ListaGenerica {

   class Nodo {
      int info;
      Nodo sig;
   }

   private Nodo raiz;

   public ListaGenerica () {
      raiz=null;
   }

   void insertarPrimero(int x)
   {
        Nodo nuevo = new Nodo ();
      nuevo.info = x;
      nuevo.sig=raiz;
      raiz=nuevo;
   }

   public void insertarUtlimo(int x) {
        Nodo nuevo = new Nodo ();
      nuevo.info = x;
      if (raiz==null)
         raiz=nuevo;
      else {
         Nodo reco=raiz;
         while (reco.sig!=null) {
            reco=reco.sig;
         }
         reco.sig=nuevo;
      }
   }

   public void insertarSegundo(int x) {
      if (raiz!=null) {
           Nodo nuevo = new Nodo ();
         nuevo.info = x;
         if (raiz.sig==null) {
            //Hay un solo nodo.
            raiz.sig=nuevo;
         } else {
            nuevo.sig=raiz.sig;
            raiz.sig=nuevo;
         }
```

```
    }
}

public void insertarAnteUltimo(int x) {
    if (raiz!=null) {
        Nodo nuevo = new Nodo ();
        nuevo.info = x;
        if (raiz.sig==null) {
            //Hay un solo nodo.
            nuevo.sig=raiz;
            raiz=nuevo;
        } else {
            Nodo atras=raiz;
            Nodo reco=raiz.sig;
            while (reco.sig!=null) {
                atras=reco;
                reco=reco.sig;
            }
            nuevo.sig=atras.sig;
            atras.sig=nuevo;
        }
    }
}

public void borrarPrimero() {
    if (raiz!=null) {
        raiz=raiz.sig;
    }
}

public void borrarSegundo() {
    if (raiz!=null) {
        if (raiz.sig!=null) {
            Nodo tercero=raiz.sig;
            tercero=tercero.sig;
            raiz.sig=tercero;
        }
    }
}

public void borrarUltimo () {
    if (raiz!=null) {
        if (raiz.sig==null) {
            raiz=null;
        } else {
            Nodo reco=raiz.sig;
            Nodo atras=reco;
            while(reco.sig!=null) {
```

```
            atras=reco;
            reco=reco.sig;
         }
         atras.sig=null;
      }
   }

}
public void imprimir () {
   Nodo reco = raiz;
   while (reco != null) {
      System.out.print (reco.info + "-");
      reco = reco.sig;
   }
   System.out.println();
}

public void borrarMayor() {
   if (raiz!=null) {
      Nodo reco=raiz;
      int may=raiz.info;
      while (reco!=null) {
         if (reco.info>may) {
            may=reco.info;
         }
         reco=reco.sig;
      }
      reco=raiz;
      Nodo atras=raiz;
      while (reco!=null) {
         if (reco.info==may) {
            if (reco==raiz) {
               raiz=raiz.sig;
               atras=raiz;
               reco=raiz;
            } else {
               atras.sig=reco.sig;
               reco=reco.sig;
            }
         } else {
            atras=reco;
            reco=reco.sig;
         }
      }
   }
}
```

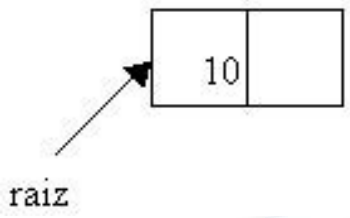
```
    public static void main(String[] ar) {
        ListaGenerica lg=new ListaGenerica();
        lg.insertarPrimero (10);
        lg.insertarPrimero(45);
        lg.insertarPrimero(23);
        lg.insertarPrimero(89);
        lg.imprimir();
        System.out.println("Insertamos un nodo al final:");
        lg.insertarUtlimo(160);
        lg.imprimir();
        System.out.println("Insertamos un nodo en la segunda posición:");
        lg.insertarSegundo(13);
        lg.imprimir();
        System.out.println("Insertamos un nodo en la anteultima posición:");
        lg.insertarAnteUltimo(600);
        lg.imprimir();
        System.out.println("Borramos el primer nodo de la lista:");
        lg.borrarPrimero();
        lg.imprimir();
        System.out.println("Borramos el segundo nodo de la lista:");
        lg.borrarSegundo();
        lg.imprimir();
        System.out.println("Borramos el ultimo nodo de la lista:");
        lg.borrarUltimo();
        lg.imprimir();
        System.out.println("Borramos el mayor de la lista:");
        lg.borrarMayor();
        lg.imprimir();
    }
}
```

Estructuras dinámicas: Listas genéricas ordenadas

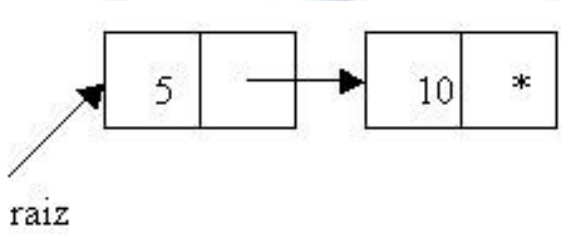
Una lista genérica es ordenada si cuando insertamos información en la lista queda ordenada respecto al campo info (sea de menor a mayor o a la inversa)

Ejemplo:

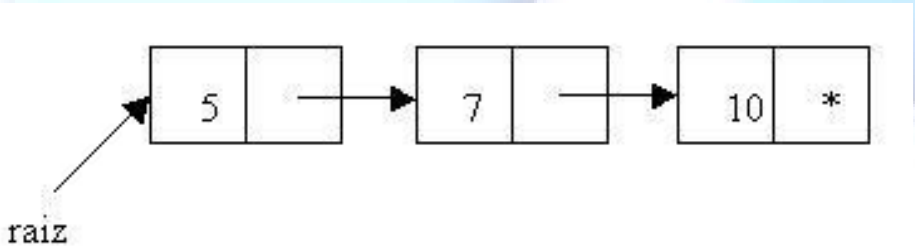
listaOrdenada.insertar(10)



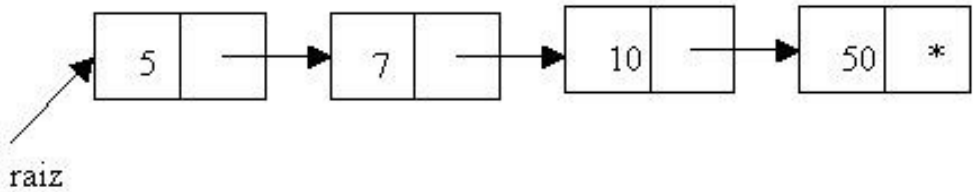
listaOrdenada.insertar(5)



listaOrdenada.insertar(7)



listaOrdenada.insertar(50)



Podemos observar que si recorremos la lista podemos acceder a la información de menor a mayor.
No se requiere un método para ordenar la lista, sino que siempre permanece ordenada, ya que se inserta ordenada.

Programa:

```
public class ListaOrdenada {

   class Nodo {
      int info;
      Nodo sig;
   }

   private Nodo raiz;

   public ListaOrdenada() {
      raiz=null;
   }

   void insertar(int x)
   {
      Nodo nuevo = new Nodo ();
      nuevo.info = x;
      if (raiz==null) {
         raiz=nuevo;
      } else {
         if (x<raiz.info) {
            nuevo.sig=raiz;
            raiz=nuevo;
         } else {
            Nodo reco=raiz;
            Nodo atras=raiz;
            while (x>=reco.info && reco.sig!=null) {
               atras=reco;
               reco=reco.sig;
            }
            if (x>=reco.info) {
               reco.sig=nuevo;
            } else {
               nuevo.sig=reco;
               atras.sig=nuevo;
            }
         }
      }
   }

   public void imprimir () {
      Nodo reco = raiz;
      while (reco != null) {
```

```
            System.out.print (reco.info + "-");
            reco = reco.sig;
        }
        System.out.println();
    }


    public static void main(String[] ar) {
        ListaOrdenada lo=new ListaOrdenada();
        lo.insertar(10);
        lo.insertar(5);
        lo.insertar(7);
        lo.insertar(50);
        lo.imprimir();
    }
}
```

El método insertar lo resolvemos de la siguiente forma:

Creamos primeramente el nodo, ya que siempre se insertará la información en la lista:

```
Nodo nuevo = new Nodo ();
nuevo.info = x;
```

Se puede presentar las siguientes situaciones, si está vacía, lo insertamos inmediatamente:

```
if (raiz==null) {
   raiz=nuevo;
} else {
```

Si no está vacía la lista, verificamos si lo debemos insertar en la primera posición de la lista (analizamos si la información a insertar es menor a lo apuntado por raiz en el campo info):

```
if (x<raiz.info) {
   nuevo.sig=raiz;
   raiz=nuevo;
} else {
```

Sino analizamos si lo debemos insertar en medio o al final de la lista.
Mientras la información a insertar sea mayor o igual a la información del nodo que visitamos (x>=reco.info) y no lleguemos al final de la lista (reco.sig!=null) avanzamos reco al siguiente nodo y fijamos un puntero en el nodo anterior (atras)

```
Nodo reco=raiz;
Nodo atras=raiz;
while (x>=reco.info && reco.sig!=null) {
   atras=reco;
   reco=reco.sig;
}
```

Cuando salimos del while si la condición (x>=reco.info) continua siendo verdadera significa que se inserta al final de la lista, en caso contrario se inserta en medio de la lista:

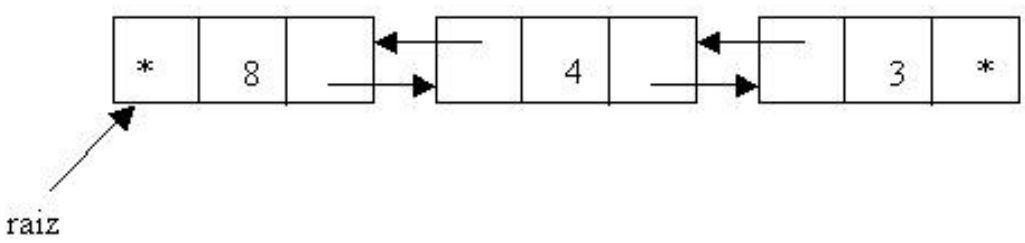
```
if (x>=reco.info) {
   reco.sig=nuevo;
} else {
   nuevo.sig=reco;
   atras.sig=nuevo;
}
```

Estructuras dinámicas: Listas genéricas doblemente encadenadas

A las listas vistas hasta el momento podemos recorrerlas solamente en una dirección (Listas simplemente encadenadas). Hay problemas donde se requiere recorrer la lista en ambas direcciones, en estos casos el empleo de listas doblemente encadenadas es recomendable.

Como ejemplo pensemos que debemos almacenar un menú de opciones en una lista, la opción a seleccionar puede ser la siguiente o la anterior, podemos desplazarnos en ambas direcciones.

Representación gráfica de una lista doblemente encadenada:



Observemos que una lista doblemente encadenada tiene dos punteros por cada nodo, uno apunta al nodo siguiente y otro al nodo anterior.
Seguimos teniendo un puntero (raiz) que tiene la dirección del primer nodo.
El puntero sig del último nodo igual que las listas simplemente encadenadas apunta a null, y el puntero ant del primer nodo apunta a null.

Se pueden plantear Listas tipo pila, cola y genéricas con enlace doble.
Hay que tener en cuenta que el requerimiento de memoria es mayor en las listas doblemente encadenadas ya que tenemos dos punteros por nodo.

La estructura del nodo es:

```
class Nodo {
        int info;
        Nodo sig, ant;
      }
```

Resolveremos algunos métodos para administrar listas genéricas empleando listas doblemente encadenadas para analizar la mecánica de enlace de nodos.
Muchos de los métodos, para listas simple y doblemente encadenadas no varía, como por ejemplo: el constructor, vacia, cantidad, etc.

Programa:

```
public class ListaGenerica {

   class Nodo {
      int info;
      Nodo ant,sig;
   }

   private Nodo raiz;

   public ListaGenerica () {
      raiz=null;
   }

   void insertar (int pos, int x)
   {
      if (pos <= cantidad () + 1)    {
         Nodo nuevo = new Nodo ();
         nuevo.info = x;
         if (pos == 1){
            nuevo.sig = raiz;
            if (raiz!=null)
               raiz.ant=nuevo;
            raiz = nuevo;
         } else
            if (pos == cantidad () + 1)    {
               Nodo reco = raiz;
               while (reco.sig != null) {
                  reco = reco.sig;
               }
               reco.sig = nuevo;
               nuevo.ant=reco;
               nuevo.sig = null;
            } else {
               Nodo reco = raiz;
               for (int f = 1 ; f <= pos - 2 ; f++)
                  reco = reco.sig;
               Nodo siguiente = reco.sig;
               reco.sig = nuevo;
               nuevo.ant=reco;
               nuevo.sig = siguiente;
               siguiente.ant=nuevo;
            }
      }
   }

   public int extraer (int pos) {
      if (pos <= cantidad ())    {
```

```
      int informacion;
      if (pos == 1) {
         informacion = raiz.info;
         raiz = raiz.sig;
         if (raiz!=null)
            raiz.ant=null;
      } else {
         Nodo reco;
         reco = raiz;
         for (int f = 1 ; f <= pos - 2 ; f++)
            reco = reco.sig;
         Nodo prox = reco.sig;
         reco.sig = prox.sig;
         Nodo siguiente=prox.sig;
         if (siguiente!=null)
            siguiente.ant=reco;
         informacion = prox.info;
      }
      return informacion;
   }
   else
      return Integer.MAX_VALUE;
}

public void borrar (int pos)
{
   if (pos <= cantidad ())    {
      if (pos == 1) {
         raiz = raiz.sig;
         if (raiz!=null)
            raiz.ant=null;
      } else {
         Nodo reco;
         reco = raiz;
         for (int f = 1 ; f <= pos - 2 ; f++)
            reco = reco.sig;
         Nodo prox = reco.sig;
         prox=prox.sig;
         reco.sig = prox;
         if (prox!=null)
            prox.ant=reco;
      }
   }
}

public void intercambiar (int pos1, int pos2) {
   if (pos1 <= cantidad () && pos2 <= cantidad ())    {
      Nodo reco1 = raiz;
```

```
      for (int f = 1 ; f < pos1 ; f++)
         reco1 = reco1.sig;
      Nodo reco2 = raiz;
      for (int f = 1 ; f < pos2 ; f++)
         reco2 = reco2.sig;
      int aux = reco1.info;
      reco1.info = reco2.info;
      reco2.info = aux;
   }
}

public int mayor () {
   if (!vacia ()) {
      int may = raiz.info;
      Nodo reco = raiz.sig;
      while (reco != null) {
         if (reco.info > may)
            may = reco.info;
         reco = reco.sig;
      }
      return may;
   }
   else
      return Integer.MAX_VALUE;
}

public int posMayor() {
   if (!vacia ())    {
      int may = raiz.info;
      int x=1;
      int pos=x;
      Nodo reco = raiz.sig;
      while (reco != null){
         if (reco.info > may) {
            may = reco.info;
            pos=x;
         }
         reco = reco.sig;
         x++;
      }
      return pos;
   }
   else
      return Integer.MAX_VALUE;
}

public int cantidad ()
{
```

```
    int cant = 0;
    Nodo reco = raiz;
    while (reco != null) {
        reco = reco.sig;
        cant++;
    }
    return cant;
}

public boolean ordenada() {
    if (cantidad()>1) {
        Nodo reco1=raiz;
        Nodo reco2=raiz.sig;
        while (reco2!=null) {
            if (reco2.info<reco1.info) {
                return false;
            }
            reco2=reco2.sig;
            reco1=reco1.sig;
        }
    }
    return true;
}

public boolean existe(int x) {
    Nodo reco=raiz;
    while (reco!=null) {
        if (reco.info==x)
            return true;
        reco=reco.sig;
    }
    return false;
}

public boolean vacia ()
{
    if (raiz == null)
        return true;
    else
        return false;
}

public void imprimir ()
{
    Nodo reco = raiz;
    while (reco != null) {
        System.out.print (reco.info + "-");
        reco = reco.sig;
```

```
        }
        System.out.println();
    }

    public static void main(String[] ar) {
        ListaGenerica lg=new ListaGenerica();
        lg.insertar (1, 10);
        lg.insertar (2, 20);
        lg.insertar (3, 30);
        lg.insertar (2, 15);
        lg.insertar (1, 115);
        lg.imprimir ();
        System.out.println ("Luego de Borrar el primero");
        lg.borrar (1);
        lg.imprimir ();
        System.out.println ("Luego de Extraer el segundo");
        lg.extraer (2);
        lg.imprimir ();
        System.out.println ("Luego de Intercambiar el primero con el tercero");
        lg.intercambiar (1, 3);
        lg.imprimir ();
        if (lg.existe(10))
            System.out.println("Se encuentra el 20 en la lista");
        else
            System.out.println("No se encuentra el 20 en la lista");
        System.out.println("La posición del mayor es:"+lg.posMayor());
        if (lg.ordenada())
            System.out.println("La lista está ordenada de menor a mayor");
        else
            System.out.println("La lista no está ordenada de menor a mayor");
    }
}
```

Para insertar en una determinada posición dentro de la lista:

```
    void insertar (int pos, int x)
```

Primero con un if verificamos que exista esa posición en la lista (por ejemplo si la lista tiene 4 nodos podemos insertar hasta la posición 5, es decir uno más allá del último):

```
        if (pos <= cantidad () + 1)    {
```

Si ingresa al if ya podemos crear el nodo:

```
            Nodo nuevo = new Nodo ();
            nuevo.info = x;
```

Ahora debemos analizar si la inserción es al principio de la lista, al final o en medio ya que los enlaces varían según donde se lo inserta.
Para saber si se inserta al principio de la lista preguntamos si en pos llega un 1:

```
if (pos == 1){
```

Si llega un 1 luego enlazamos el puntero sig del nodo que creamos con la dirección del primer nodo de la lista (raiz apunta siempre al primer nodo de la lista)
Verificamos si raiz está apuntando actualmente a un nodo, en caso afirmativo enlazamos el puntero ant con el nodo que acabamos de crear y luego desplazamos raiz al nodo creado:

```
nuevo.sig = raiz;
if (raiz!=null)
   raiz.ant=nuevo;
raiz = nuevo;
```

Si no se inserta al principio de la lista preguntamos si se inserta al final:

```
if (pos == cantidad () + 1)    {
```

En caso de insertarse al final recorremos la lista hasta el último nodo:

```
Nodo reco = raiz;
while (reco.sig != null) {
   reco = reco.sig;
}
```

y enlazamos el puntero sig del último nodo de la lista con la dirección del nodo que acabamos de crear (disponemos en sig del nodo creado el valor null ya que no hay otro nodo más adelante) El puntero ant del nodo que creamos lo enlazamos con el nodo que era último hasta este momento y está siendo apuntado por reco:

```
reco.sig = nuevo;
nuevo.ant=reco;
nuevo.sig = null;
```

Si no se inserta al principio o al final significa que tenemos que insertar en medio de la lista. Disponemos un for donde avanzamos un puntero auxiliar y nos detenemos una posición antes a donde tenemos que insertarlo:

```
for (int f = 1 ; f <= pos - 2 ; f++)
   reco = reco.sig;
```

Disponemos otro puntero auxiliar que apunte al nodo próximo a donde está apuntando reco. Ahora enlazamos el puntero sig del nodo apuntado por reco con la dirección del nodo creado y el puntero sig del nodo creado con la dirección del nodo siguiente. El puntero ant del nodo apuntado por nuevo lo enlazamos con el nodo apuntado por raiz y el puntero ant del nodo apuntado por siguiente lo apuntamos a nuevo (con esto tenemos actualizados los cuatro punteros internos a la lista):

```
Nodo siguiente = reco.sig;
reco.sig = nuevo;
nuevo.ant=reco;
nuevo.sig = siguiente;
siguiente.ant=nuevo;
```

El método extraer recibe como parámetro la posición del nodo a extraer:

```
public int extraer (int pos) {
```

Primero verificamos que la posición exista en la lista:

```
    if (pos <= cantidad ())    {
```

En caso que exista verificamos si el nodo a extraer es el primero de la lista (este análisis debe hacerse ya que si es el primero de la lista se modifica el puntero raiz):

```
        if (pos == 1) {
```

Si es el primero guardamos en una variable auxiliar la información del nodo y avanzamos el puntero raiz, luego si raiz apunta a un nodo disponemos el puntero ant de dicho nodo a null:

```
            informacion = raiz.info;
            raiz = raiz.sig;
            if (raiz!=null)
               raiz.ant=null;
```

Si el nodo a extraer no está al principio de la lista avanzamos con una estructura repetitiva hasta el nodo anterior a extraer:

```
            for (int f = 1 ; f <= pos - 2 ; f++)
               reco = reco.sig;
```

Luego definimos otro puntero auxiliar y lo disponemos en el siguiente nodo a donde está apuntando reco:

```
            Nodo prox = reco.sig;
```

Ahora enlazamos el puntero sig del nodo apuntado por reco al nodo siguiente del nodo apuntado por prox (es decir el nodo apuntado por prox queda fuera de la lista) disponemos finalmente otro puntero llamado siguiente que apunte al nodo que se encuentra una posición más adelante del nodo apuntado por prox, si dicho puntero apunta a un nodo actualizamos el puntero ant de dicho nodo con la dirección del nodo apuntado por reco:

```
            reco.sig = prox.sig;
            Nodo siguiente=prox.sig;
            if (siguiente!=null)
               siguiente.ant=reco;
            informacion = prox.info;
```

El método borrar es muy similar al método extraer con la diferencia de que no retorna valor:

```
public void borrar (int pos)
{
   if (pos <= cantidad ())    {
      if (pos == 1) {
         raiz = raiz.sig;
         if (raiz!=null)
            raiz.ant=null;
      } else {
```

```
            Nodo reco;
            reco = raiz;
            for (int f = 1 ; f <= pos - 2 ; f++)
               reco = reco.sig;
            Nodo prox = reco.sig;
            prox=prox.sig;
            reco.sig = prox;
            if (prox!=null)
               prox.ant=reco;
         }
      }
   }
```

El método intercambiar recibe dos enteros que representan las posiciones de los nodos que queremos intercambiar sus informaciones:

```
   public void intercambiar (int pos1, int pos2) {
```

Mediante un if verificamos que las dos posiciones existan en la lista:

```
      if (pos1 <= cantidad () && pos2 <= cantidad ())    {
```

Definimos un puntero auxiliar llamado reco1, lo inicializamos con la dirección del primer nodo y mediante un for avanzamos hasta la posición almacenada en pos1:

```
         Nodo reco1 = raiz;
         for (int f = 1 ; f < pos1 ; f++)
            reco1 = reco1.sig;
```

De forma similar con un segundo puntero auxiliar avanzamos hasta la posición indicada por pos2:

```
         Nodo reco2 = raiz;
         for (int f = 1 ; f < pos2 ; f++)
            reco2 = reco2.sig;
```

Por último intercambiamos las informaciones que almacenan cada nodo:

```
         int aux = reco1.info;
         reco1.info = reco2.info;
         reco2.info = aux;
```

El método que retorna el mayor de la lista:

```
   public int mayor () {
```

Verificamos que la lista no esté vacía:

```
      if (!vacia ())    {
```

Suponemos que el mayor es el primero de la lista e inicializamos un puntero auxiliar con la dirección del segundo nodo de la lista:

```
int may = raiz.info;
Nodo reco = raiz.sig;
```

Mediante una estructura repetitiva recorremos toda la lista:

```
while (reco != null) {
```

Cada vez que encontramos un nodo con información mayor que la variable may la actualizamos con este nuevo valor y avanzamos el puntero reco para visitar el siguiente nodo:

```
if (reco.info > may)
   may = reco.info;
reco = reco.sig;
```

Fuera de la estructura repetitiva retornamos el mayor:

```
return may;
```

El método que retorna la posición del mayor es similar al anterior con la salvedad que debemos almacenar en otro auxiliar la posición donde se almacena el mayor:

```
public int posMayor() {
   if (!vacia ())    {
      int may = raiz.info;
      int x=1;
      int pos=x;
      Nodo reco = raiz.sig;
      while (reco != null){
         if (reco.info > may) {
            may = reco.info;
            pos=x;
         }
         reco = reco.sig;
         x++;
      }
      return pos;
   }
   else
      return Integer.MAX_VALUE;
}
```

El método que debe retornar si está ordenada la lista de menor a mayor es:

```
public boolean ordenada() {
```

Lo primero que verificamos si la lista tiene más de un nodo significa que debemos controlarla:

```
if (cantidad()>1) {
```

Disponemos dos punteros auxiliares con las direcciones del primer y segundo nodo de la lista:

```
Nodo reco1=raiz;
Nodo reco2=raiz.sig;
```

Mediante un while mientras no se finaliza la lista:

```
while (reco2!=null) {
```

controlamos si la información del segundo nodo es menor al nodo anterior significa que la lista no está ordenada y podemos parar el análisis retornando un false

```
if (reco2.info<reco1.info) {
  return false;
```

Dentro del while avanzamos los dos punteros a sus nodos siguientes respectivamente.

```
reco2=reco2.sig;
reco1=reco1.sig;
```

Fuera del while retornamos true indicando que la lista está ordenada de menor a mayor

```
return true;
```

El método existe:

```
public boolean existe(int x) {
```

Mediante un while recorremos la la lista:

```
Nodo reco=raiz;
while (reco!=null) {
```

y en cada nodo que visitamos controlamos si el parámetro x es igual a la información del nodo, en caso afirmativo salimos del método retornando true:

```
if (reco.info==x)
  return true;
reco=reco.sig;
```

Fuera del while retornamos false indicando que ningún nodo coincide con el parámetro x:

```
return false;
```

Problemas propuestos

1. Plantear una clase para administrar una lista genérica doblemente encadenada implementando los siguientes métodos:
   a) Insertar un nodo al principio de la lista.
   b) Insertar un nodo al final de la lista.
   c) Insertar un nodo en la segunda posición. Si la lista está vacía no se inserta el nodo.
   d) Insertar un nodo en la ante última posición.
   e) Borrar el primer nodo.
   f) Borrar el segundo nodo.
   g) Borrar el último nodo.
   h) Borrar el nodo con información mayor.

Solución

```
public class ListaGenericaDoble {

   class Nodo {
      int info;
      Nodo ant,sig;
   }

   private Nodo raiz;

   public ListaGenericaDoble () {
      raiz=null;
   }

   void insertarPrimero(int x)
   {
       Nodo nuevo = new Nodo ();
      nuevo.info = x;
      nuevo.sig=raiz;
      if (raiz!=null)
         raiz.ant=nuevo;
      raiz=nuevo;
   }

   public void insertarUtlimo(int x) {
       Nodo nuevo = new Nodo ();
      nuevo.info = x;
      if (raiz==null)
         raiz=nuevo;
      else {
         Nodo reco=raiz;
         while (reco.sig!=null) {
            reco=reco.sig;
         }
         reco.sig=nuevo;
         nuevo.ant=reco;
      }
   }

   public void insertarSegundo(int x) {
      if (raiz!=null) {
          Nodo nuevo = new Nodo ();
         nuevo.info = x;
         if (raiz.sig==null) {
            //Hay un solo nodo.
            raiz.sig=nuevo;
```

```
            nuevo.ant=raiz;
        } else {
            Nodo tercero=raiz.sig;
            nuevo.sig=tercero;
            tercero.ant=nuevo;
            raiz.sig=nuevo;
            nuevo.ant=raiz;
        }
    }
}

public void insertarAnteUltimo(int x) {
    if (raiz!=null) {
         Nodo nuevo = new Nodo ();
        nuevo.info = x;
        if (raiz.sig==null) {
            //Hay un solo nodo.
            nuevo.sig=raiz;
            raiz=nuevo;
        } else {
            Nodo reco=raiz;
            while (reco.sig!=null) {
                reco=reco.sig;
            }
            Nodo anterior=reco.ant;
            nuevo.sig=reco;
            nuevo.ant=anterior;
            anterior.sig=nuevo;
            reco.ant=nuevo;
        }
    }
}

public void borrarPrimero() {
    if (raiz!=null) {
        raiz=raiz.sig;
    }
}

public void borrarSegundo() {
    if (raiz!=null) {
        if (raiz.sig!=null) {
            Nodo tercero=raiz.sig;
            tercero=tercero.sig;
            raiz.sig=tercero;
            if (tercero!=null)
                tercero.ant=raiz;
        }
```

```
    }
}

public void borrarUltimo () {
    if (raiz!=null) {
        if (raiz.sig==null) {
            raiz=null;
        } else {
            Nodo reco=raiz;
            while(reco.sig!=null) {
                reco=reco.sig;
            }
            reco=reco.ant;
            reco.sig=null;
        }
    }

}
public void imprimir () {
    Nodo reco = raiz;
    while (reco != null) {
        System.out.print (reco.info + "-");
        reco = reco.sig;
    }
    System.out.println();
}

public void borrarMayor() {
    if (raiz!=null) {
        Nodo reco=raiz;
        int may=raiz.info;
        while (reco!=null) {
            if (reco.info>may) {
                may=reco.info;
            }
            reco=reco.sig;
        }
        reco=raiz;
        while (reco!=null) {
            if (reco.info==may) {
                if (reco==raiz) {
                    raiz=raiz.sig;
                    if (raiz!=null)
                        raiz.ant=null;
                    reco=raiz;
                } else {
                    Nodo atras=reco.ant;
                    atras.sig=reco.sig;
```

```
                reco=reco.sig;
                if (reco!=null)
                   reco.ant=atras;
             }
          } else {
             reco=reco.sig;
          }
       }
    }
  }


  public static void main(String[] ar) {
     ListaGenericaDoble lg=new ListaGenericaDoble();
     lg.insertarPrimero (10);
     lg.insertarPrimero(45);
     lg.insertarPrimero(23);
     lg.insertarPrimero(89);
     lg.imprimir();
     System.out.println("Insertamos un nodo al final:");
     lg.insertarUtlimo(160);
     lg.imprimir();
     System.out.println("Insertamos un nodo en la segunda posición:");
     lg.insertarSegundo(13);
     lg.imprimir();
     System.out.println("Insertamos un nodo en la anteultima posición:");
     lg.insertarAnteUltimo(600);
     lg.imprimir();
     System.out.println("Borramos el primer nodo de la lista:");
     lg.borrarPrimero();
     lg.imprimir();
     System.out.println("Borramos el segundo nodo de la lista:");
     lg.borrarSegundo();
     lg.imprimir();
     System.out.println("Borramos el ultimo nodo de la lista:");
     lg.borrarUltimo();
     lg.imprimir();
     System.out.println("Borramos el mayor de la lista:");
     lg.borrarMayor();
     lg.imprimir();
  }
}
```

Muchas gracias hasta la próxima clase.

Alsina 16 [B1642FNB] San Isidro | Pcia. De Buenos Aires |Argentina |

TEL.: [011] 4742-1532 o [011] 4742-1665 |

www.institutosanisidro.com.ar info@institutosanisidro.com.ar