1.- HOJA DE PRESENTACION:

MATERIA: ESTRUCTURA DE DATOS

INGIENERIA INFORMATICA

T1EDEQUIPO1 (UNIDAD 4)

EQUIPO #1: Castro Ramón David Alejandro 20010329

Martínez Ramos Rodrigo 20010347

Carrillo Ávila Juan Pablo 20010327

GRUPO: 4PM – 5PM HRS CLAVE: 3a3A

Fecha de entrega: 02-Junio-2023

INDICE DEL REPORTE

2) INTRODUCCIÓN (RESUMEN):

En este reporte de la unidad 5 se explicarán todos los códigos/programas que en este caso abarcan los temas de Datos Simples o arreglos y también vimos ejemplos relacionados a una parte de pilas y colas y listas enlazadas, así doble ligadas.

Este reporte tiene el propósito de que pongamos y documentemos todos lo que se hizo dentro de las diversas unidades en la metería de estructura de datos (en este caso es la Unidad 5) en el cual se mostrara paso a paso como fue que fimos evolucionando en temas además de nuestro avance con los códigos que se presentaron por parte de la maestra.

Además, nos beneficiara en la parte de que repasaremos todo lo visto en las unidades anteriores y recalcar nuestros conocimientos prácticos para resolver problemas por nuestra propia cuenta basándonos en apuntes y temas/documentos previos para saber si los resultados son los esperados al finalizar no solo cada unidad sino la materia de estructura de datos en el presente semestre

El presente proyecto académico está dirigido a estudiar y comprender la forma en cómo se trabaja con nodos en listas simplemente enlazadas.

Las listas permiten insertar y borrar elementos en cualquier lugar de la misma, al principio, en el medio o al final; pero hay algunas situaciones frecuentes en programación en las que es necesario restringir las inserciones y borrados de los elementos solo al principio o al final. Dos de las estructuras de datos que son útiles en tales situaciones son las pilas y las colas.

Una pila es una estructura lineal en la que los elementos pueden ser añadidos o eliminados solo por el final y una cola es una lista lineal en la que los elementos solo pueden ser añadidos por un extremo y eliminados por el otro.

3) COMPETENCIA ESPECIFICA:

Conoce y comprende las diferentes estructuras de datos, su clasificación y forma de manipularlas para buscar la manera más eficiente de resolver problemas.

Genéricas:

- Utilizar las clases predefinidas para el manejo de pilas, colas y listas enlazadas (dinámicas) y describir en un texto la diferencia de hacerlo con arreglos.

- Utilizar las estructuras lineales en la elaboración de códigos para la resolución de problemas elaborando un reporte.

Estructuras no lineales

Competencias Actividades de aprendizaje Específica(s):

Comprende y aplica estructuras no lineales para la solución de problemas.

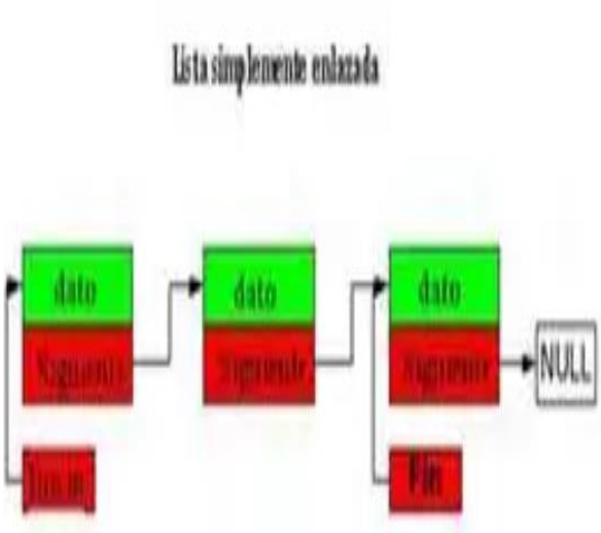
Genéricas:

- Habilidad para buscar y analizar información proveniente de fuentes diversas.
- La comprensión y manipulación de ideas y pensamientos.
- Metodologías para solución de problemas, organización del tiempo y para el aprendizaje.
- Habilidad en el manejo de equipo de cómputo
- Capacidad para trabajar en equipo.
- Capacidad de aplicar los conocimientos en la práctica.
- Elaborar un cuadro sinóptico o esquema con la clasificación de los árboles y sus aplicaciones.
- Implementar las operaciones básicas de inserción, eliminación y búsqueda en un árbol binario.

4) MARCO TEÓRICO:

3.1 Listas Simples Enlazadas

Las listas simplemente enlazadas son estructuras de datos semejantes a las estructuras array salvo que el acceso a un elemento no se hace mediante un índice, sino mediante un puntero.



La asignación de memoria se la realiza durante la ejecución.

En una lista simplemente enlazada, los elementos son contiguos en lo que concierne al enlazado.

En una lista simplemente enlazada los elementos están dispersos.

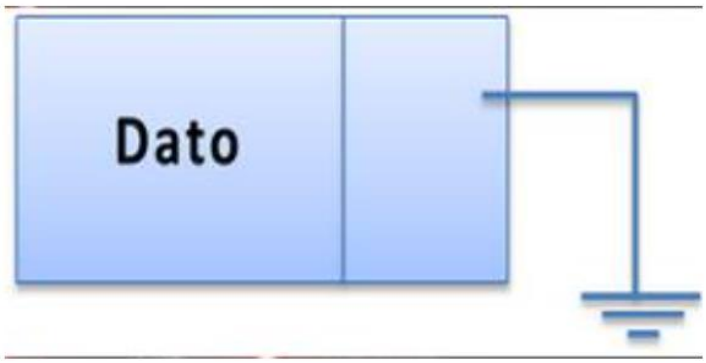
Es decir, que cada elemento se almacena en un lugar de memoria que le asigna el ordenador de forma aleatoria.

Esta asignación como ya se mencionó, se la realiza durante la ejecución.

Cada dirección de memoria designada auna lista, estará ocupada por nodos.

3.2 Nodos

Una lista se compone de nodos, que son estructuras de datos que nos permiten registrar datos de interés. Para que estos nodos se conviertan en una lista, debe existir un enlace entre ellos, que en términos más propios se los conoce como apuntadores o punteros.



3.5 Funciones

En listas simplemente enlazadas, existen algunas funciones principales o elementales:

- **Agregar. -** Una de la característica de las listas simplemente enlazadas, y de cualquier otro tipo de estructura similar, es que es dinámico, es por eso que una lista no es fija, más al contrario, podemos agregarle nodos al principio, al final, o en cualquier posición de la lista.
- **Eliminar. -** Así como podemos ir agregando nuevos nodos a nuestra lista, también podemos eliminar nodos de la misma; una de las razones principales, para liberar espacio en memoria ya que es posible el ya no estar utilizando ese nodo.
- **Buscar**. - Podemos obtener, mediante algoritmos de búsqueda, los datos o direcciones de uno o varios nodos de la lista.

5) MATERIAL Y EQUIPO:

✓ ***Computadora:*** Cualquier dispositivo electrónico (Computafora o Laptop) que tenga buen funcionamiento en sistema y que pueda ser compatible con un lenguaje de programación para que se pueda instalar, además de una buena memoria de almacenamiento.

✓ ***Software y versión usados:*** En la creación de los programas se utilizó el programa/aplicación de Eclipse que se instala por medio de su JDK Y el IDE que se puede encontrar en la plataforma del lenguaje de programación

✓***Materiales de apoyo para el desarrollo de la practica:*** Documentos de apoyo para la parte teórica presentados por la maestra el cual contenía los códigos y ejemplos de los programas que vemos y pasarlos con nuestros conocimientos al lenguaje de programación que elegimos en este caso Eclipse Versión 2023

6) DESARROLLO DE LA PRACTICA:

Listas simples enlazadas.

Empezamos con Nodo esta es una clase genérica la cual tendrá la estructura para nuestra lista enlazada y los métodos necesarios (Seters y Geters) para recorrer la lista, insertar datos y obtener la informacion:

```java
package MemoriaDinamica;

public class Nodo<T> {

    public T info;
    public Nodo sig;

    public Nodo(T info) {
        this.info = info;
        this.sig = null;
    }

    public T getInfo() {
        return info;
    }

    public void setInfo(T info) {
        this.info = info;
    }

    public Nodo getSig() {
        return sig;
    }

    public void setSig(Nodo sig) {
        this.sig = sig;
    }
}
```

Creamos nuestra clase Nodo la cual será genérica (<T>) la cual contendrá las variables que utilizaremos para gestionar la lista.

Creamos una variable de tipo genérico(<T>) la cual llamaremos info, esta se encargara de obtener he insertar información en la lista.

Creamos una variable de tipo Nodo la cual llamaremos sig esta se encargará de obtener las direcciones de el siguiente nodo en la lista.

Creamos nuestro constructor el cual se encargará de iniciar nuestra lista, iniciando por el dato que le mandemos y a siguiente le pondrá null.

Después tendremos los Set y Get necesarios para obtener y añadir nodos a nuestra lista.

Creamos una interfas generica(<T>) la cual contendra todos los metodos que ocuparemos para manipular la lista enlazada.

```
1  package EstructurasColas;
2
3  public interface ColaTDA <T>{
4
5      public boolean isEmptyCola();
6
7      public void pushCola(T Dato);
8
9      public T popCola();
10
11     public T peekCola();
12
13     public void freeCola();
14
15 }
```

Los métodos serán:

1. Boolean isEmptyCola(): este método se encargara de regresar un booleano dependiendo si la lista se encuentra vacía o no.
2. Void PushCola(T Dato): este método recibe como parámetro un dato, se encargara de insertar ese dato en la lista.
3. T PopCola(): este método se encargara de eliminar el ultimo nodo de la lista y devolverá el dato eliminado.
4. T PeekCola(): este método se encargara de devolver el ultimo nodo de la lista pero sin eliminarlo.
5. Void FreeCola(): Este método se encargara de eliminar toda la lista.

Creamos nuestra clase generica(<T>) la cual se encargara de manipular nuestra lista enlazada, a esta clase se le implementara la interfas ColaTDA para obtener sus metodos genericos.

```
1  package EstructurasColas;
2
3  import MemoriaDinamica.Nodo;
4
5  public class ColaD<T> implements ColaTDA<T>{
6
7      private Nodo cola;
8      private Nodo f;
9
10     public ColaD() {
11         cola = null;
12     }
```

Creamos nuestra lista la cual se llamará cola el cual será de tipo Nodo, también creamos un apuntador(f) el cual tendrá la última dirección de nuestra lista, igual será de tipo Nodo.

Crearemos nuestro arreglo el cual simplemente iniciara nuestra lista con null.

Añadimos el metodo isEmptyCola() de la interfas ColaTDA, este se encargara de devolver true en caso de que la lista este vacia y un false si la lista tiene datos

```
14     @Override
15     public boolean isEmptyCola() {
16         return (cola==null);
17     }
```

Añadimos el metodo pushCola(T Dato) este metodo resivira el dato que queremos ingresar a la lista, primero creara un nuevo nodo con el dato que resivio, despues la unira a la lista, en caso de que este vacia la pondra en el primer nodo, en caso contrario lo unira con el ultimo nodo creado y final mente el apuntador pasara al nuevo nodo.

```
19      @Override
20      public void pushCola(T Dato) {
21          Nodo u=new Nodo(Dato);
22          if(isEmptyCola()) {
23              cola=u;
24          }else {
25              f.sig = u;
26          }
27          f=u;
28      }
```

Añadimos el método popCola, este método se encargara de eliminar el primer dato en la lista pasando el apuntador del primer nodo(cola) al siguiente nodo.

```
30      @Override
31      public T popCola() {
32          Nodo u= cola;
33          T dato=(T) cola.getInfo();
34          cola=cola.getSig();
35          u=null;
36          return dato;
37      }
```

Añadimos el método peekCola este solo se encargara de devolver el primer dato en la lista.

```
39      @Override
40      public T peekCola() {
41          return (T)(cola.getInfo());
42      }
```

Añadimos el método freeCola este método solo se encargará de igualar cola y f a null así perdiendo todas las direcciones de los nodos.

```
44      @Override
45      public void freeCola() {
46          cola=null;
47          f=null;
48      }
```

Creamos los metodos toString los cuales se encargaran de imprimir toda la lista para poder visualizarla.

```
50      public String toString() {
51          Nodo u=cola;
52          return toString(u);
53      }
54
55      public String toString(Nodo i) {
56          return (i!=null)?"tope==>"+"["+i.getInfo()+"]\n"+toString(i.getSig()):"";
57      }
```

Crearemos otras listas simples enlazadas, las cuales serán lista ordenada y desordenada.

Empezamos creando una nueva interfaz genérica(<T>) la cual llamaremos OperaTDA

Esta clase tendra nuestros metodos genericos que nos serviran mas adelante para implementarlos en nuestra clase.

```
1  package Operaciones;
2
3  import MemoriaDinamica.Nodo;
4
5  public interface OperaTDA<T> {
6
7      public void insertarFrente(T dato);
8
9      public void insertarFinal(T dato);
10
11     public boolean isListaVacia();
12
13     public Nodo buscarLista(T dato);
14
15     public void Eliminar(Nodo dato);
16
17     public void Modificar(Nodo dato);
18
19     public String mostrarLista();
20
21 }
```

Empecemos por la lista desordenada

Creamos nuestra clase generica(<T>) he implementamos la interfaz OperaTDA que igualmente es generica.

Crearemos dos variables para llevar las direcciones de nuestros nodos, las dos variables seran de tipo Nodo.

```
1  package Operaciones;
2
3  import MemoriaDinamica.Nodo;
4  import Tools.ToolsPanel;
5
6  public class Desordenado<T> implements OperaTDA<T> {
7
8      private Nodo inicio;
9      private Nodo f;
```

Implementamos el metodo insertar frente el cual nor servira para insertar datos al frente de nuestra lista.

```
11     @Override
12     public void insertarFrente(T dato) {
13         Nodo u = new Nodo(dato);
14         if(isListaVacia()) {
15             inicio=u;
16             f=u;
17         }else {
18             u.sig=inicio;
19             inicio=u;
20         }
21     }
```

Implementamos el método insertar final, este método nos ayudara a insertar datos al final de la lista, con es una lista desordenada, no importa como se insertan los números.

```
23      @Override
24      public void insertarFinal(T dato) {
25          Nodo u = new Nodo(dato);
26          if(isListaVacia()) {
27              inicio = u;
28              f=u;
29          }else {
30              f.sig=u;
31          }
32          f=u;
33      }
```

Implementamos el metodo isListaVacia el cual nos dira si nuestra lista esta vacia o tiene algun dato insertado.

```
35      @Override
36      public boolean isListaVacia() {
37          return (inicio== null);
38      }
```

Implementamos el metodo buscarLista este metodo nos ayudara a buscar cualquier dato que querramos en nuestra lista, si se encuentra devolvera la direccion donde se encuentra, en caso contrario devolvera un null.

```
40      @Override
41      public Nodo buscarLista(T dato) {
42          Nodo i=inicio;
43          while(i!=null && dato!=i.getInfo()) {
44              i=i.getSig();
45          }
46          return (i!=null)? (Nodo)i:null;
47      }
```

Implementamos el metodo Eliminar el cual nos ayudara a eliminar un dato de nuestra lista.

```
49    @Override
50    public void Eliminar(Nodo existe) {
51        if(existe==inicio) {
52            inicio=inicio.sig;
53            existe=null;
54        }else {
55            Nodo antes=buscarAnterior(existe);
56            if(existe==f) {
57                f=antes;
58                f.sig=null;
59                existe=null;
60            }else {
61                antes.sig=existe.sig;
62                existe=null;
63            }
64        }
65    }
```

Implementamos el método mostrarLista este método nos ayudara a guardar en una cadena cada dato de nuestra lista para así imprimirlo.

```
67    @Override
68    public String mostrarLista() {
69        String cad="";
70        Nodo i=inicio;
71        while(i!=null) {
72            cad+=i.getInfo()+"--->\n";
73            i=i.getSig();
74        }
75        return cad;
76    }
```

Implementamos el metodo buscarAnterior, este metodo nos ayudara a encontrar la direccion anterior del nodo donde nos encontremos, asi haciendo mas facil algunos metodos como el de eliminar.

```
78    public Nodo buscarAnterior(Nodo dato) {
79        Nodo i=inicio;
80        if(dato!=inicio) {
81            while(dato!=i.getSig() && i.getSig()!=null) {
82                i=i.getSig();
83            }
84        }else {
85            i=null;
86        }
87        return (Nodo) i;
88    }
```

Implementamos el metodo modificar, este metodo solo se encargara de modificar la informacion de cualquier nodo que este en la lista.

```
90      public void Modificar(Nodo dato) {
91          dato.setInfo(ToolsPanel.leerInt("A que numero lo quieres modificar"));
92      }
```

Ahora pasaremos con la lista ordenada

Creamos una nueva clase generica, a esta tambien le implementaremos la interfaz OperaTDA, esta clase contara igual con dos variables iniciales que manejaran las direcciones de la lista inicio y f.

```
1  package Operaciones;
2
3  import MemoriaDinamica.Nodo;
4  import Tools.ToolsPanel;
5
6  public class Ordenado<T> implements OperaTDA<T>{
7
8      private Nodo inicio;
9      private Nodo f;
```

Implementamos el metodo insertar frente, este metodo insertara datos al frente de nuestra lista, si la lista ya tiene datos el usuario solo podra insertar datos menores que el que se encuentra en la primera posicion.

```
11     @Override
12     public void insertarFrente(T dato) {
13         Nodo u = new Nodo(dato);
14         if(isListaVacia()) {
15             inicio=u;
16             f=u;
17         }else {
18             if((int)dato>=(int)inicio.getInfo()) {
19                 ToolsPanel.imprimeError("El dato no puede ser mayor o igual a"+inicio.getInfo());
20             }else {
21                 u.sig=inicio;
22                 inicio=u;
23             }
24         }
25     }
```

Implementamos el metodo insertar final, este metodo insertara datos al final de la lista solo si el dato dado por el usuario es mayor al ultimo dato de la lista

```
27    @Override
28    public void insertarFinal(T dato) {
29        Nodo u = new Nodo(dato);
30        if(isListaVacia()) {
31            inicio = u;
32            f=u;
33        }else {
34            if((int)dato<=(int)f.getInfo()) {
35                ToolsPanel.imprimePantalla("El dato no puede ser menor o igual que: "+f.getInfo());
36            }else
37                f.sig=u;
38        }
39        f=u;
40    }
```

Implementamos el metodo isListaVacia este metodo nos servira para verificar si inicio es igual con null, eso queiere decir que nuestra lista esta vacia en caso contrario la lista ya tiene datos.

```
42    @Override
43    public boolean isListaVacia() {
44        return (inicio==null);
45    }
```

Implementamos el metodo buscar lista, este metodo se encargara de encontrar el dato que busquemos, si se encuentra en la lista regresara la direccion del dato, sino regresara un null.

```
47    @Override
48    public Nodo buscarLista(T dato) {
49        Nodo i=inicio;
50        while(i!=null && dato!=i.getInfo()) {
51            i=i.getSig();
52        }
53        return (i!=null)? (Nodo)i:null;
54    }
55
56    public Nodo buscarAnterior(Nodo dato) {
57        Nodo i=inicio;
58        if(dato!=inicio) {
59            while(dato!=i.getSig() && i.getSig()!=null) {
60                i=i.getSig();
61            }
62        }else {
63            i=null;
64        }
65        return (Nodo) i;
66    }
```

Implementamos el metodo eliminar, este metodo se encargara de eliminar de la lista el dato que nosotros querramos.

```
68      @Override
69      public void Eliminar(Nodo existe) {
70          if(existe==inicio) {
71              inicio=inicio.sig;
72              existe=null;
73          }else {
74              Nodo antes=buscarAnterior(existe);
75              if(existe==f) {
76                  f=antes;
77                  f.sig=null;
78                  existe=null;
79              }else {
80                  antes.sig=existe.sig;
81                  existe=null;
82              }
83          }
84      }
```

Implementamos el metodo modificar, este metodo modificara el dato que querramos, siempre y cuando se encuentre entre el rango de numeros de el dato anterior y el dato posterior de el nodo donde nos encontramos

```
86      @Override
87      public void Modificar(Nodo dato) {
88          Nodo ant = buscarAnterior(dato);
89
90          boolean ban = false;
91          while(ban) {
92              int datos =ToolsPanel.leerInt("A que numero lo quieres cambiar");
93              if((int)ant.info>=0) {
94                  if(datos<=(int)ant.info) {
95                      ToolsPanel.imprimeError("El numero no puede ser menor o igual a: "+ant.info);
96                  }else {
97                      if(dato.sig!=null) {
98                          if(datos>=(int)dato.sig.info) {
99                              ToolsPanel.imprimeError("El numero no puede ser mayor o igual a: "+dato.sig.info);
100                         }
101                         else {
102                             dato.setInfo(datos);
103                             ban=true;
104                         }
105                     }else {
106                         dato.setInfo(datos);
107                         ban=true;
108                     }
109                 }
110             }else {
111                 if(dato.sig!=null) {
112                     if(datos>=(int)dato.sig.info) {
113                         ToolsPanel.imprimeError("El numero no puede ser mayor o igual a: "+dato.sig.info);
114                     }
115                     else {
116                         dato.setInfo(datos);
117                         ban=true;
```

```
                }
            }else {
                dato.setInfo(datos);
                ban=true;
            }
        }
    }

}
```

Implementamos el metodo mostrarlista, este metodo solo guardara los datos en una cadena y regresara esa cadena a donde fue llamado.

```
    @Override
    public String mostrarLista() {
        String cad="";
        Nodo i=inicio;
        while(i!=null) {
            cad+=i.getInfo()+"--->\n";
            i=i.getSig();
        }
        return cad;
    }
```

Lista doble enlazada

Para estos métodos necesitaremos una nueva clase Nodo, a esta la llamaremos Nodito.
Esta clase ahora tendrá recorridos de izquierda y derecha del nodo, así que crearemos los Set y Get necesarios.

```java
public class Nodito <T>{

    public T info;
    public Nodito izq;
    public Nodito der;
    public Nodito(T dato) {
        this.info=dato;
        this.izq=null;
        this.der=null;
    }
    public T getInfo() {
        return info;
    }
    public void setInfo(T info) {
        this.info = info;
    }
    public Nodito getIzq() {
        return izq;
    }
    public void setIzq(Nodito izq) {
        this.izq = izq;
    }
    public Nodito getDer() {
        return der;
    }
    public void setDer(Nodito der) {
        this.der = der;
    }
```

Creamos una nueva clase la cual tendra nuestra lista doble, esta tambien iniciaran con dos variables que tendran las direcciones de nuestra lista.

Creamos un constructor para iniciar nuestro puntero con null.

```java
package DobleLiga;

import MemoriaDinamica.Nodo;
import Operaciones.OperaTDA;
import Tools.ToolsPanel;

public class DatosDesordenadosDobleLiga<T> implements OperaTDA<T>{

    private Nodito puntero;
    private Nodito f;

    public DatosDesordenadosDobleLiga() {
        puntero = null;
    }
```

Implementamos el metodo insertar frente, este metodo solo insertara datos al inicio de la lista, como es una lista desordenada no importa como se inserten los numeros.

```
16    @Override
17    public void insertarFrente(T dato) {
18        Nodito p = new Nodito(dato);
19        if(isListaVacia()) {
20            puntero=p;
21            f=p;
22        }else {
23            p.der=puntero;
24            puntero.izq=p;
25            puntero=p;
26        }
27    }
```

Implementamos el metodo insertar final, este metodo solo insertara datos al final de la lista, como es una lista desordenada no importa como se inserten los numeros.

```
29    @Override
30    public void insertarFinal(T dato) {
31            Nodito p= new Nodito(dato);
32            if(isListaVacia()) {
33                puntero=p;
34            }else {
35                f.der=p;
36                p.izq=f;
37            }
38            f=p;
39    }
```

Implementamos el metodo is lista vacia que nos dira si nuestra lista se encuentra vacia o no.

```
41    @Override
42    public boolean isListaVacia() {
43        return (puntero==null);
44    }
```

Creamos el metodo buscar lista el cual nos buscara la direccio donde se encuentra nuestro dato a buscar.

```
46    public Nodito buscarListas(T dato) {
47        Nodito b=puntero;
48        while(b!=null&&!(dato.equals((Object)b.getInfo())))
49            b=b.getDer();
50        return (b);
51    }
```

Implementamos el método eliminar, el cual sacara de la lista la dirección del nodo que contenga el dato a eliminar.

```
53    public void Eliminar(Nodito dato) {
54        if(dato==puntero) {
55            puntero=puntero.der;
56            puntero.izq=null;
57            dato=null;
58        }else {
59            if(dato==f) {
60                f=f.izq;
61                f.der=null;
62                dato=null;
63            }else {
64                dato.der.izq=dato.izq;
65                dato.izq.der=dato.der;
66                dato=null;
67            }
68        }
69
70    }
71
```

Implementamos el metodo modificar el cual modificara la informacion del nodo que tengla la informacion del dato que queriamos modificar.

```
72    public void Modificar(Nodito dato) {
73        dato.info=ToolsPanel.leerString("A que dato lo deseas cambiar");
74    }
```

Implementamos el metodo mostrarLista el cual nos mostrara la lista de izquierda a derecha y de derecha a izquierda.

```
76      @Override
77      public String mostrarLista() {
78          String cad="";
79
80          for(Nodito j=puntero; j!=null; j=j.der) {
81              cad+=j.getInfo()+"==>";
82          }
83          cad+="\n";
84          for(Nodito j=f; j!=null; j=j.izq) {
85              cad+=j.getInfo()+"==>";
86          }
87
88          return cad;
89      }
```

Arboles

Para esta clase ocuparemos la clase Nodito.
Crearemos una clase generica(<T>) la cual se encargara de administrar nuestro arbol.

Empezamos por crear nuestro arbol, creamos una variable la cual llamaremos raiz de tipo Nodito, esta variable se encargara de tener la direccion donde comienza nuestro arbol.

Creamos nuestro constructor el cual iniciara nuestra raiz como null.

Creamos el Get y el Set los cuales se encargaran de obtener he insertar los datos de la raiz.

```java
package estrucnolineal;

import DobleLiga.Nodito;

public class ArbolBin<T>{

    private Nodito raiz;

    public ArbolBin() {
        raiz=null;
    }

    public Nodito getRaiz() {
        return raiz;
    }

    public void setRaiz(Nodito raiz) {
        this.raiz=raiz;
    }
```

Creamos un metodo que devolvera una bariable booleana, si la raiz es igual con null quiere decir que no tenemos ningun dato en el arbol asi que devolvera un true y en caso contrario devolvera un false.

```java
    public boolean arbolVacio() {
        return (raiz==null);
    }
```

Creamos un metodo que se encargara de vaciar el arreglo, igualando la raiz con null asi pérdiendo todos las direcciones del arbol.

```java
    public void vaciarArbol() {
        raiz=null;
    }
```

Creamos un metodo que se encargara de insertar los nuevos datos al arbol, resivimos el dato que queremos ingresar en los parametros, creamos un nuevo nodo y le insertamos el dato, si el arbol esta vacio insertamos el nodo en la raiz, en caso contrario buscaremos el que sera el padre del nuevo nodo y lo insertamos debajo de la direccion que regreso buscar padre dependiendo si es mayor o menor.

```java
    public void insertarArbol(T info) {

        Nodito p = new Nodito(info);

        if(arbolVacio()) {
            raiz=p;
        }else {
            Nodito padre = buscarPadre(raiz, p);

            if((int)p.info>=(int)padre.info)
                padre.der=p;
            else {
                padre.izq=p;
            }
        }

    }
```

Creamos un metodo que se encargara de buscar el padre del nuevo nodo que queremos ingresar al arbol, este metodo resivira como parametros la raiz y el dato que queremos ingresar, buscara la direccion dependiendo si el dato que queremos ingresar es mas grande igual o mas pequeño que el nodo donde se encuentra actual he ira recorriendo el arbol hasta encontrar un null, cuando lo encuentre regresara la ultiuma direccion donde estuvo.

```java
    public Nodito buscarPadre(Nodito actual, Nodito p) {
        Nodito padre = null;
        while(actual!=null) {
            padre = actual;
            if((int)p.info>=(int)padre.info) {
                actual = padre.der;
            }else {
                actual = padre.izq;
            }
        }
        return padre;
    }
```

Creamos un metodo que imprimira el arbol en preorden, este metodo resivira como parametro el arbol, recorrera este arbol de forma recursiva primero imprimiendo la informacion del nodo donde se encuentra y recorrera el arbol primero por la izquierda, una ves que termine con la

izquierda pasara con la derecha y asi sucesivamente, hasta llegar al ultimo nodo.

```
60    public String preorden(Nodito r) {
61        if(r!=null) {
62            return r.getInfo()+" - "+preorden(r.getIzq())+" - "+preorden(r.getDer());
63        }
64        else return "";
65    }
```

Crearemos un metodo que imprimira el arbol en la forma inorden, de forma recursiva recorrera el arbol primero por la izquierda, despues imprimira el nodo donde se encuentra y al final recorrera el arbol por su lado derecho, esto se repítira hasta que llegue al ultimo nodo.

```
67    public String inorden(Nodito r) {
68        if(r!=null) {
69            return inorden(r.getIzq())+" - "+r.getInfo()+" - "+inorden(r.getDer());
70        }
71        else return "";
72    }
```

Crearemos un metodo que imprimira el arbol en la forma inorden pero al reves, de forma recursiva recorrera el arbol primero por la derecha, despues imprimira el nodo donde se encuentra y al final recorrera el arbol por su lado izquierdo, esto se repítira hasta que llegue al ultimo nodo.

```
74    public String inorden2(Nodito r) {
75        if(r!=null) {
76            return inorden2(r.getDer())+" - "+r.getInfo()+" - "+inorden2(r.getIzq());
77        }
78        else return "";
79    }
```

Creamos un metodo que recorrera el arbol de forma recursiva, iniciando por recorrer el arbol en su lado izquierdo, cuando termine parasa al lado derecho y al final imprimira el nodo donde se encuentra.

```
81    public String posorden(Nodito r) {
82        if(r!=null) {
83            return posorden(r.getIzq())+" - "+posorden(r.getDer())+" - "+r.getInfo();
84        }
85        else return "";
86    }
```

Crearemos un metodo que busque cualquier dato que querramos en el arbol, para esto resiviremos dos parametros los cuales seran, el arbol y el

dato a buscar, para esto usaremos un while para recorrer el arbol dependiendo si en el nodo donde nos encontramos es mas grande que el dato que querremos nos iremos a la derecha del arbol en caso contrario nos iremos a la izquierda, repitiendo este porseso encontraremos el dato que buscamos.

```
88      public Nodito buscarDato(Nodito r, T dato) {
89          while(r!=null) {
90              if(r.getInfo()==dato) {
91                  return r;
92              }else {
93                  if((int)dato<(int)r.getInfo()) {
94                      r=r.getIzq();
95                  }else {
96                      r=r.getDer();
97                  }
98              }
99          }
100         return r;
101     }
```

Para este metodo lamentablemente no pudimos imprimir el arbol vertical mente pero si orizontalmente, primero tenemos que verificar que el arbol no este vacio, una ves verificado empesaremos a recorrer el arbol, primero por la derecha paraque salgan arriba los nodos de la derecha una ves llegando al dinal se empesaran a imprimir para esto daremos los espacios necesarios para que tome la forma del arbol para hacer esto mientras vamos recorriendo el arbol imcrementaremos el nivel, una ves llegada hasta la raiz se empezaran a imprimir los de la izquierda cada ves mas abajo.

```
103     public void graficarArbol(Nodito nodo, int nivel) {
104         if (nodo != null) {
105             graficarArbol(nodo.der, nivel + 1);
106             System.out.println("    ".repeat(nivel) + nodo.info+"<");
107             graficarArbol(nodo.izq, nivel + 1);
108         }
109 
110     }
```

Creamos un metodo que nos dara la altura del arbol, el cual recorrera todo el arbol he ira guardando cuantas veces recorrio el arbol en las variables altura izqueida y altura derecha, al final ocuparemos la funcion Math.max para sacar la altura del arbol y le aumentaremos uno para contar la raiz como 1.

```
112    public int Altura(Nodito r) {
113        if (r == null) {
114            return 0;
115        }
116
117        int alturaIzquierdo = Altura(r.izq);
118        int alturaDerecho = Altura(r.der);
119
120
121        return Math.max(alturaIzquierdo, alturaDerecho)+1;
122    }
```

Creamos un metodo que nos dara las hojas del arbol osea los nodos de los extremos, para esto recorreremos el arbol hasta que derecha he isquierda sean nulos asi sabremos que son las hojas del arbol.

```
124    public String Hojas(Nodito nodo) {
125        String cad= "";
126        if (nodo != null) {
127            if(nodo.izq==null && nodo.der==null) {
128                cad+=nodo.info+", ";
129            }
130            cad+=Hojas(nodo.der);
131            cad+=Hojas(nodo.izq);
132        }
133        return cad;
134    }
```

Para este metodo compararemos cada nodo si tiene dos nodos hijos eso significa que es un nodo interior asi que lo agregamos a la cadena.

```
136     public String Interiores(Nodito nodo) {
137         String cad="";
138         if (nodo == null || (nodo.izq == null && nodo.der == null)) {
139             return cad;
140         }
141         // Es un nodo interior, imprime el valor
142         if(nodo!=raiz)
143             cad+=nodo.info+", ";
144 
145         // Recorrer recursivamente los nodos hijo (izquierdo y derecho)
146         cad+=Interiores(nodo.izq);
147         cad+=Interiores(nodo.der);
148 
149         return cad;
150 
151     }
152 
153 }
```

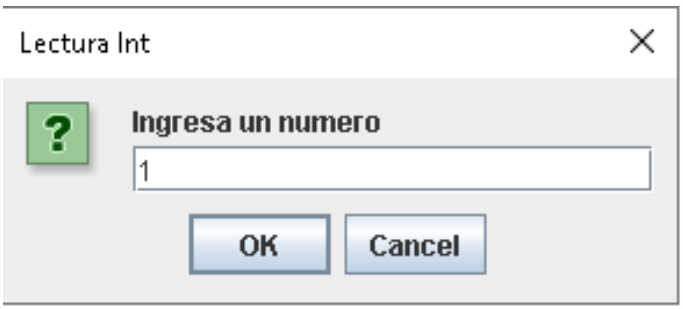
Finalmente crearemos un menú para manipular nuestro árbol.

7) RESULTADOS:

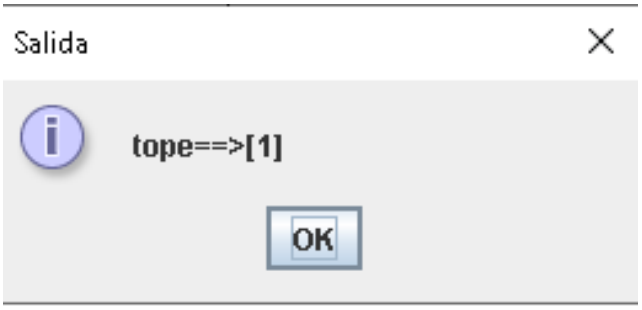
Lista simple enlazada

Menu

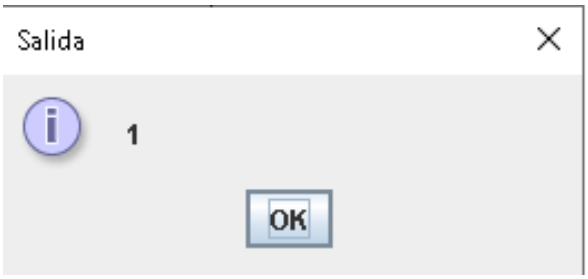
Push

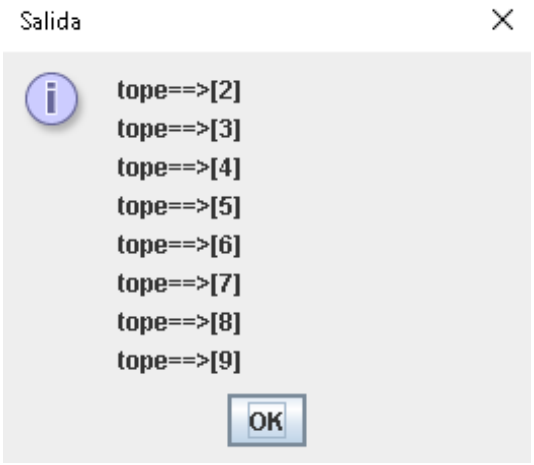


Imprimir



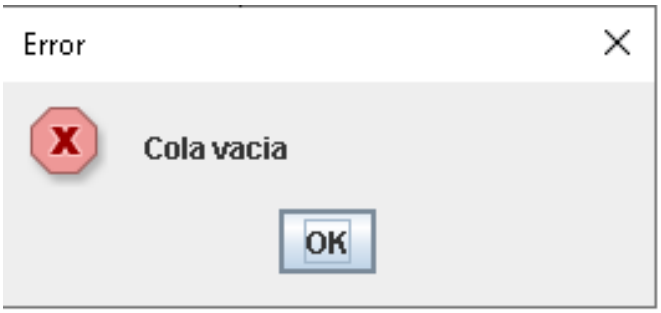
Pop





Peek

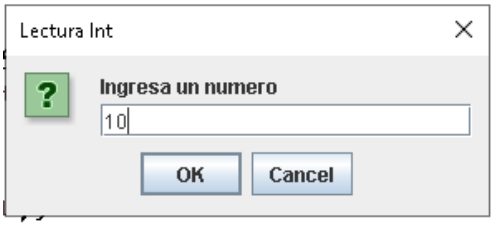
Free

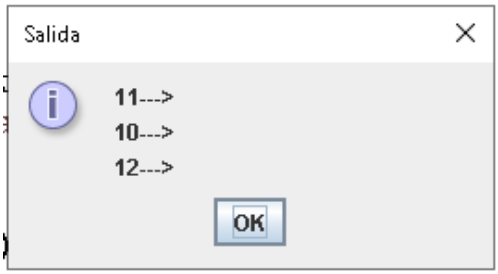


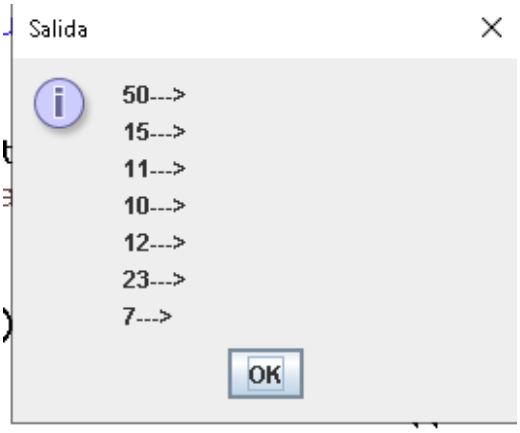
Lista simple enlazada desordenada

Menu

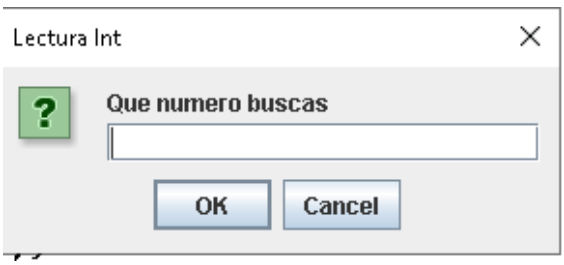
Insertar Frente, Final

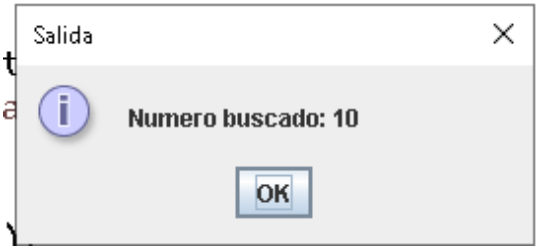


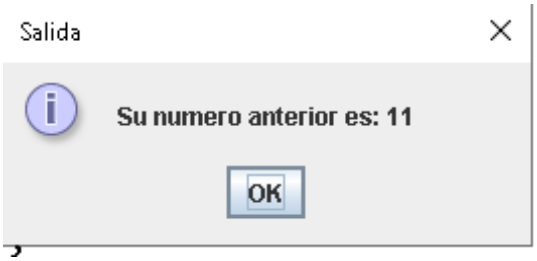




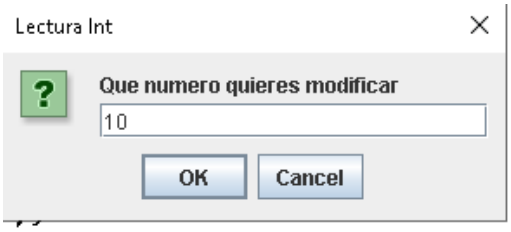
Buscar

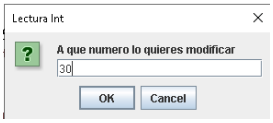


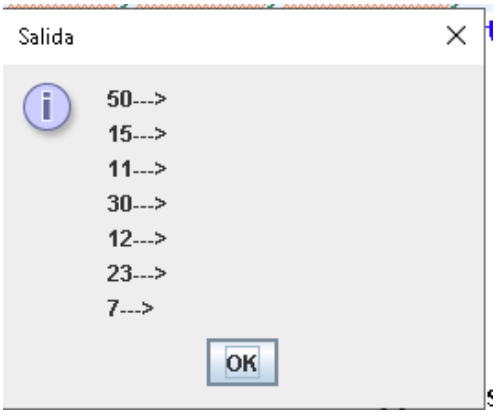




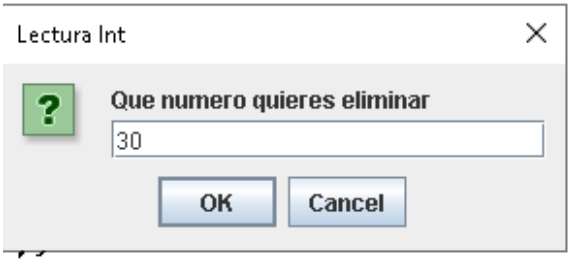
Modificar

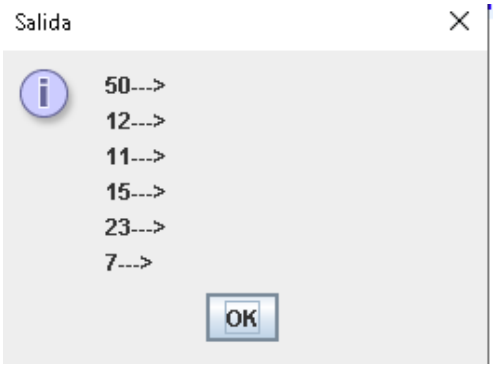






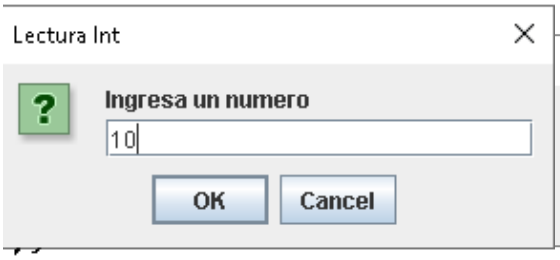
Eliminar

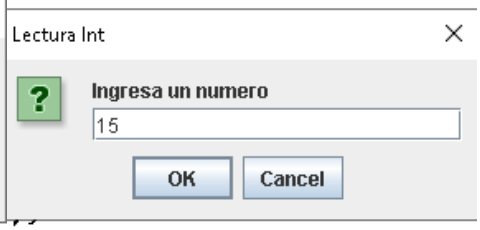


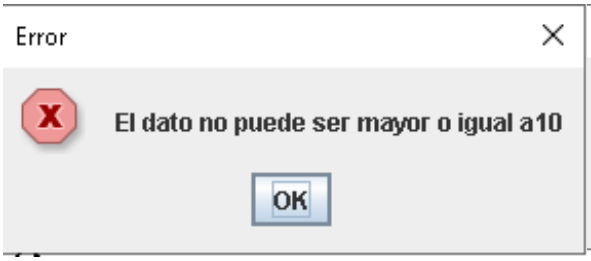


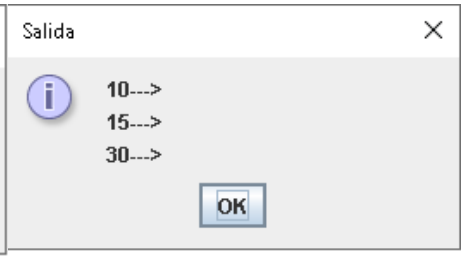
Lista simple ordenada

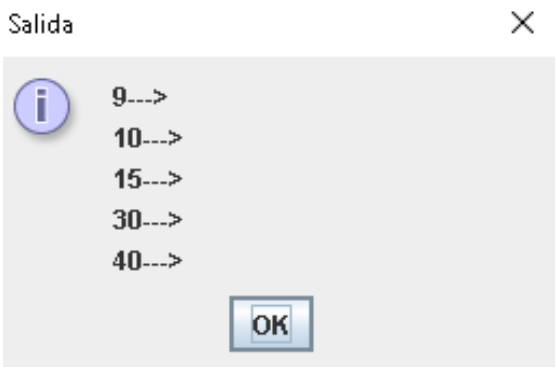
Insertar frente y fianl.



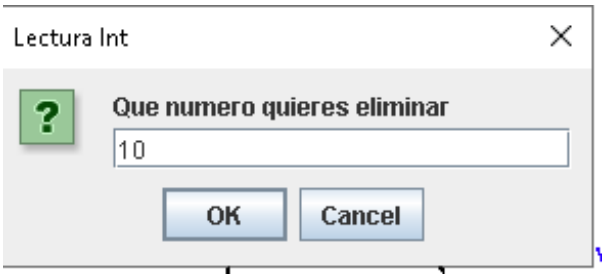


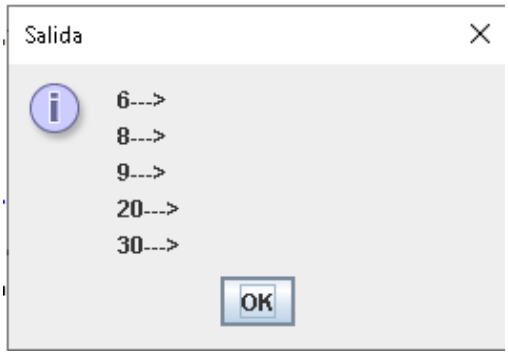




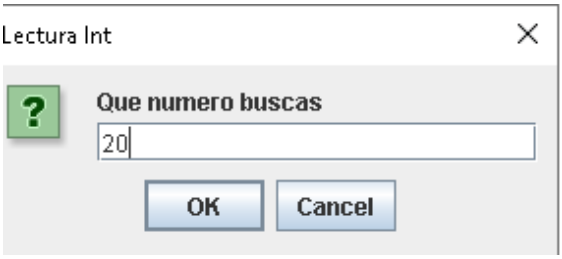


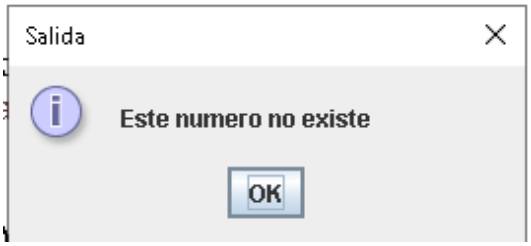
Eliminar



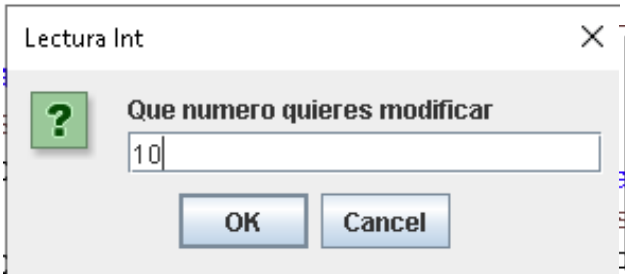


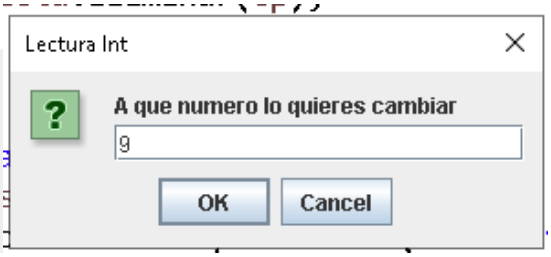
Busqueda

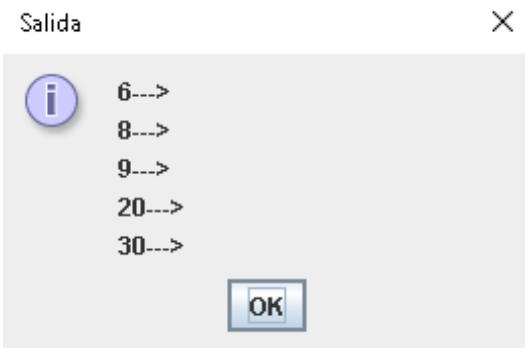




Modificar



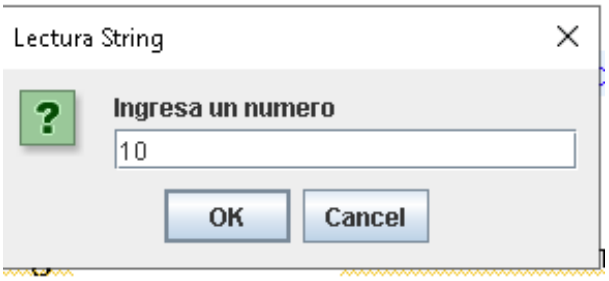


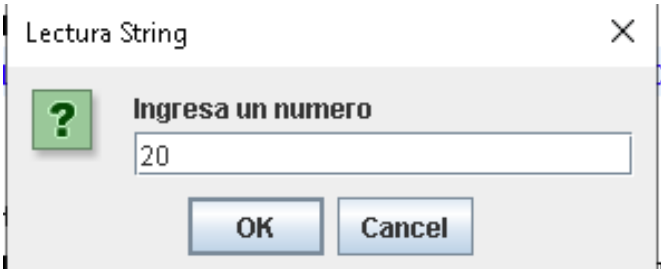


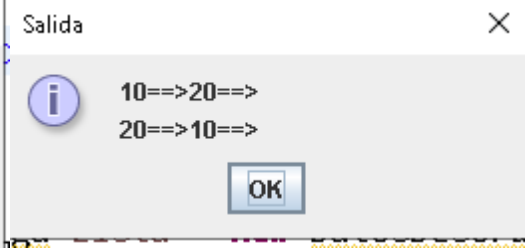
Lista enlazada doble liga

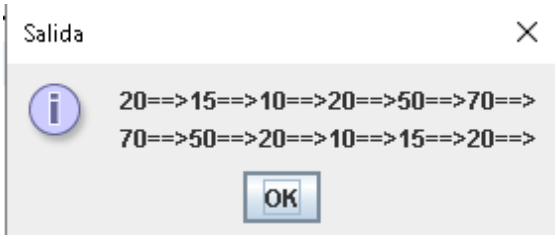
Menu

Insertar frente y final.

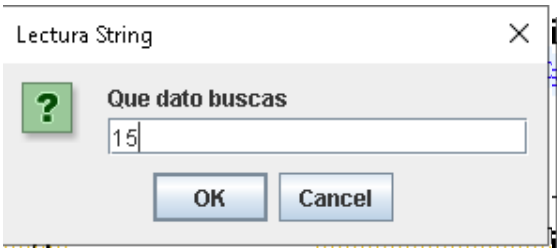


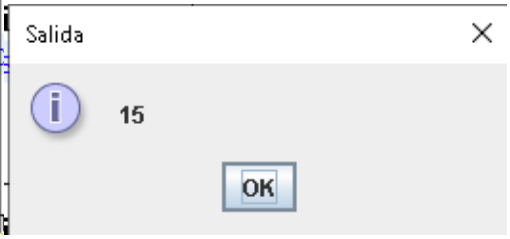




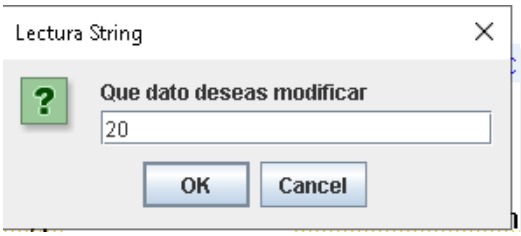


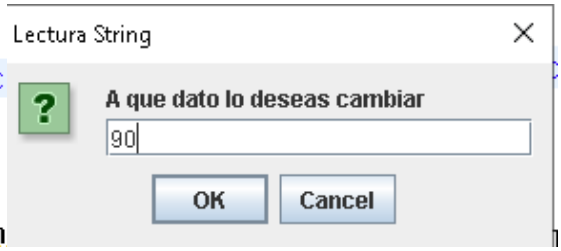
Busquedas

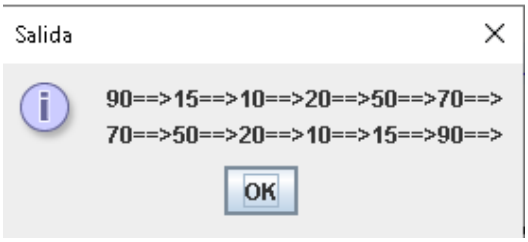




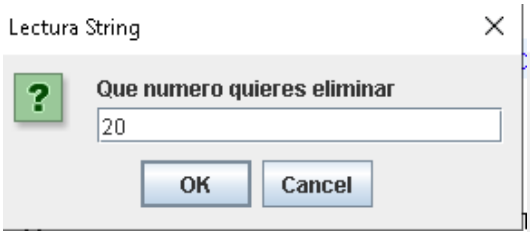
Modificar

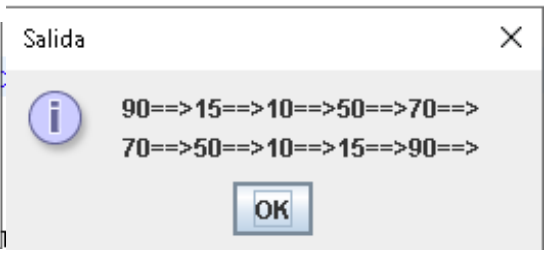






Eliminar

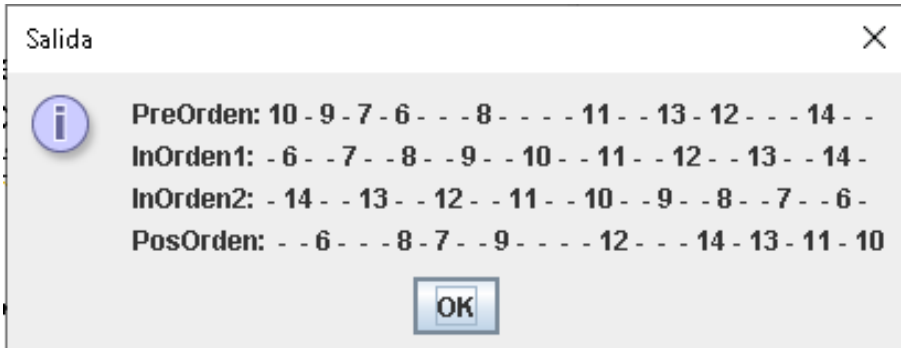




Arboles

Menu

Recorridos

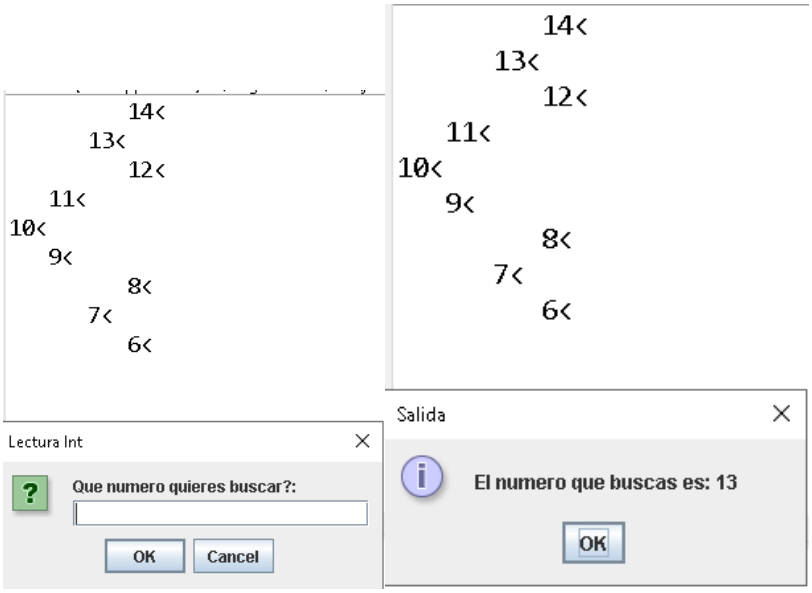


Imprecion del arbol

```
nPane;

in(String[] args) {
ecorridos,Buscar,Hojas,In

nu0(String menu){
 arb;
n();

(menu);
1){
ertar":
nsertarArbol(ToolsPanel.leerInt("Inserte un dato"));
```

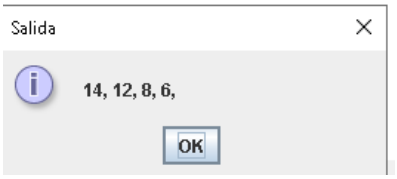
```
les\Arbol [Java Application] C:\Program Files\Java\jdk-15\bin\javaw.exe  (2 jun. 2
        14<
    13<
        12<
  11<
10<
  9<
        8<
    7<
        6<
```

Búsqueda



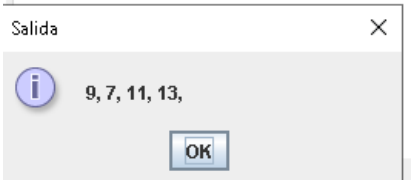
Hojas

```
          14<
       13<
          12<
   11<
10<
   9<
          8<
       7<
          6<
```

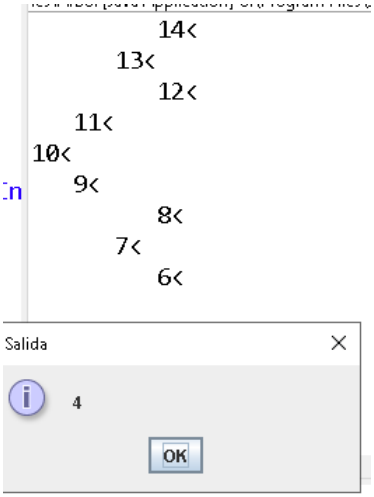


Interiores

```
          14<
       13<
          12<
   11<
10<
   9<
          8<
       7<
          6<
```



Altura



8) CONCLUSIONES:

En conclusión, La implementación de pilas y colas mediante listas enlazadas posibilita la representación eficiente de los datos en situaciones donde es necesario indicar el orden de procesamiento de los mismos y no es posible prever la cantidad de elementos a procesar por cuanto este tipo de representación permite crear y destruir variables dinámicamente.

¿Qué es la cola en programación?

Una cola es una estructura de datos que almacena elementos en una lista y permite acceder a los datos por uno de los dos extremos de la lista. Un elemento se inserta en la cola (parte final) de la lista y se suprime o elimina por la frente (parte inicial, cabeza) de la lista.

Pues en las colas como en toda estructura de datos las operaciones principales son insertar y eliminar, aunque en varias implementaciones de colas puedan recibir nombres diferentes.

Las listas simplemente enlazadas, son posiblemente las estructuras de datos más fáciles, rápidas y sencillas de estudiar, aprender y entender.

La resolución de una función relacionada con listas simplemente enlazadas, es fácil y rápida, en especial porque no se requieren demasiadas líneas de código, por ende la solución es inmediata.

La implementación de una aplicación basada en listas simplemente enlazadas, también supone un fácil desarrollo, es especial si se trabaja con lenguajes de programación como *Java*, dada las facilidades que brinda al momento de trabajar con este tipo de estructuras,

un ejemplo es la facilidad, eficacia y rapidez para eliminar un nodo de la lista, ya que con otras herramientas, lo más probable sea tener que soltar el enlace nodo por nodo, mientras que en *Java*, solo soltamos el enlace del nodo que queremos eliminar.

Las listas simplemente enlazadas, así como también otro tipo de estructuras similares, son útiles a la hora de trabajar problemas como PILAS y COLAS, ya que se maneja la misma lógica de agregar, borrar o buscar elementos.

9) Bibliografía:

Katrib mora, miguel . Lenguajes de programación y Técnicas de compilación/

LIPSCHUTZ, SEYMOUR. Estructura de datos/ Syemour Lipschutz.—Ciudad de la Habana: Edición Revolucionaria, 1989.—390 p.

Estructuras De Datos Básicas*":*

http://users.dcc.uchile.cl,

*"*Listas Enlazadas Simples, Y Árboles Binarios", 2002:

http://tutorialms-dos.bligoo.com

"La Listas Simplemente Enlazada", 2007.

http://es.kioskea.net

Estructuras De Datos: Listas Enlazadas, Pilas Y Colas".

http://www.calcifer.org