

**Nombre:**Melanni del Rosario Tzul Baquiax.  
**Curso:**Laboratorio de Estructura de datos.  
**Proyecto:**DBMS (DataBase Management System)

**Id:8**

## **Manual de Técnico:**

Creamos una clase que se llama LecturaArchivo con un atributo de tipo ListaEstructura, en esta clase podremos manejar todas las funciones que nos permitirá leer las estructuras entrantes:



```
1 public class LecturaArchivo {  
2  
3     private ListaEstructura listaEstructura;
```

La función ingresarEstructuraArchivo se encarga de procesar un archivo XML que contiene información sobre estructuras de tablas. Lee el archivo, extrae los datos de las estructuras y las columnas, y crea instancias correspondientes. Luego, verifica y establece las relaciones entre las estructuras y las columnas de llave. Finalmente, guarda las estructuras en una lista y muestra la lista resultante en un objeto JTextArea.

```

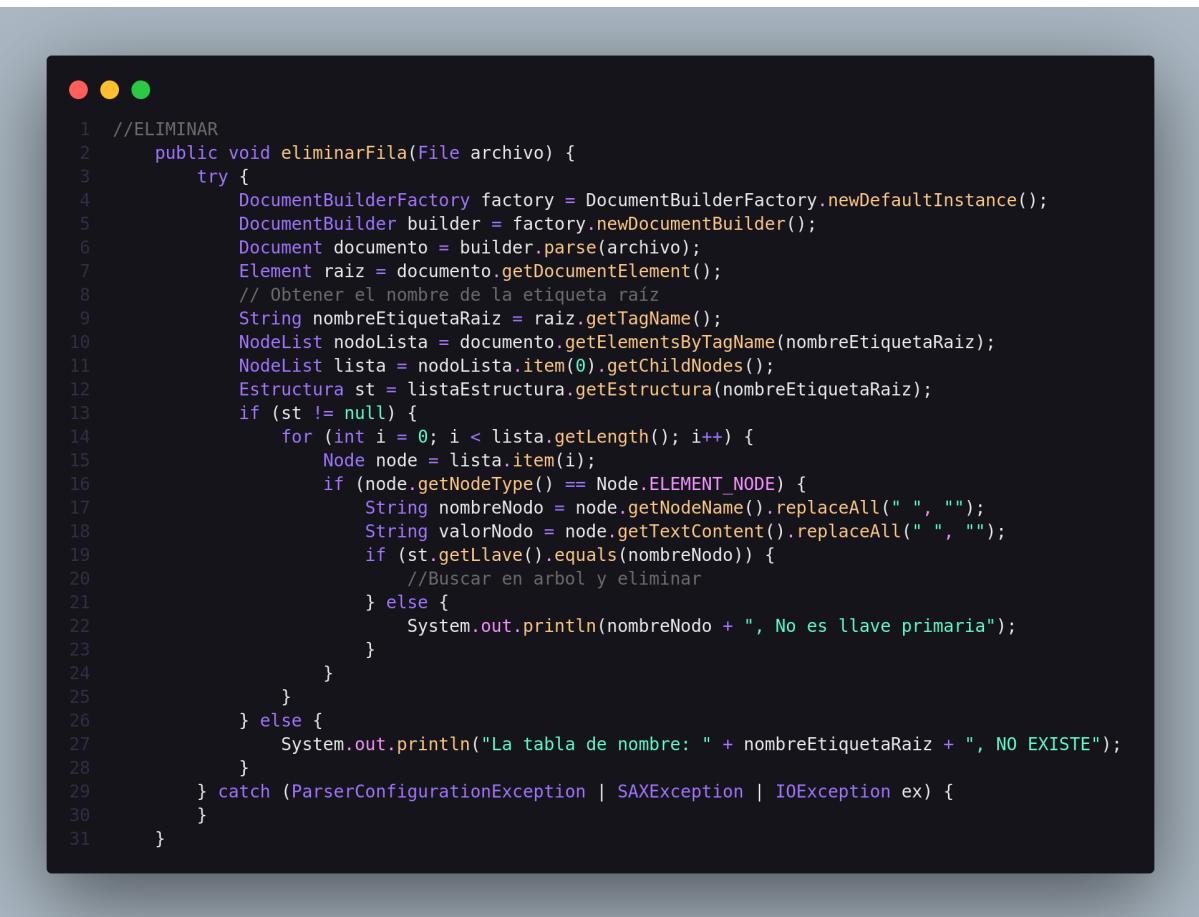
1 public void ingresarEstructuraArchivo(File archivo, JTextArea txtArea) {
2     try {
3         // Se crea una instancia del factory para obtener el builder
4         DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
5         DocumentBuilder builder = factory.newDocumentBuilder();
6         // Se parsea el archivo y se obtiene un objeto Document
7         Document documento = builder.parse(archivo);
8         // Se obtiene una lista de nodos con nombre "estructura"
9         NodeList nodoLista = documento.getElementsByTagName("estructura");
10        //Recorrido de las estructuras encontradas
11        for (int i = 0; i < nodoLista.getLength(); i++) {
12            Node nodo = nodoLista.item(i);
13            // Verificar si el tipo de nodo es un nodo de elemento
14            if (nodo.getNodeType() == Node.ELEMENT_NODE) {
15                Estructura estructura = new Estructura(); //Estructura
16                ListaColumna lista = new ListaColumna(); //Lista de campos de la estructura
17                NodeList nodosSecundarios = nodo.getChildNodes();
18                String llave = null; //clave de la tabla
19                //Recorrido de los hijos de una estructura
20                for (int j = 0; j < nodosSecundarios.getLength(); j++) {
21                    Node nodoHija = nodosSecundarios.item(j);
22                    // Verificar si el tipo de nodo es un nodo de elemento
23                    if (nodoHija.getNodeType() == Node.ELEMENT_NODE) {
24                        String nombreNodo = nodoHija.getNodeName().replaceAll(" ", ""); //Nombre del nodo
25                        String valorNodo = nodoHija.getTextContent().replaceAll(" ", ""); //Valor del nodo
26                        valorNodo = valorNodo.replaceAll("t", "");
27                        // Verificar si el nombre del nodo es "tabla"
28                        if (nombreNodo.equals("tabla")) {
29                            Estructura temporal = listaEstructura.getEstructura(valorNodo);
30                            if (temporal == null) {//si es igual a null
31                                estructura.setNombre(valorNodo);
32                            } else {
33                                System.out.println("La tabla con el nombre: " + valorNodo + " ya existe!");
34                                break;
35                            }
36                        } else if (nombreNodo.equals("clave")) {
37                            llave = valorNodo;
38                        } else if (nombreNodo.equals("relacion")) {
39                            boolean table = false;//Bandera para saber que si existe la tabla de referencia
40                            String nombreColumna = "";
41                            String valorColumna = "";
42                            NodeList listaRelacion = nodoHija.getChildNodes();
43                            for (int k = 0; k < listaRelacion.getLength(); k++) {
44                                Node hijaRelacion = listaRelacion.item(k);
45                                if (hijaRelacion.getNodeType() == Node.ELEMENT_NODE) {
46                                    String nombre = hijaRelacion.getNodeName().replaceAll(" ", "");
47                                    String value = hijaRelacion.getTextContent().replaceAll(" ", "");
48                                    Estructura st = listaEstructura.getEstructura(nombre);
49
50                                    if (st != null) {
51                                        if (st.getLlave().equals(value)) {
52                                            table = true;
53                                        }
54                                    } else {
55                                        nombreColumna = nombre;
56                                        valorColumna = value;
57                                    }
58                                }
59                            }
60                            if (table && !nombreColumna.equals("") && !valorColumna.equals("")) {
61                                NodoColumna nc = new NodoColumna(nombreColumna, valorColumna);
62                                lista.insertarFinal(nc);
63                            } else {
64                                System.out.println("La relacion no cumple los requisitos");
65                                break;
66                            }
67                        } else {
68                            NodoColumna nc = new NodoColumna(nombreNodo, valorNodo);
69                            lista.insertarFinal(nc);
70                        }
71                    }
72                }
73                if (llave != null) {
74                    NodoColumna c = lista.getNode(llave); // Se establece la llave de la estructura
75                    if (c != null) {
76                        estructura.setLlave(llave); // Se establece la llave de la estructura
77                        estructura.setColumnas(lista); // Se establece la lista de columnas de la estructura
78                        listaEstructura.insertarFinal(estructura); // Se inserta la estructura en la lista de estructuras
79                        System.out.println("Estructura creada"); // Se imprime un mensaje indicando que se ha creado la estructura
80                    } else {
81                        System.out.println("No tiene los datos completos"); // No se encontró la columna de llave en la lista
82                    }
83                }
84            }
85        }
86    } catch (ParserConfigurationException | SAXException | IOException ex) {
87        Logger.getLogger(Menu.class.getName()).log(Level.SEVERE, null, ex);
88    }
89    listaEstructura.imprimirLista( txtArea);
90 }
91 }

```

La función lee un archivo XML que contiene información de filas para una tabla existente. Verifica la existencia de la tabla y sus columnas, valida los valores de las columnas según su tipo de dato, y agrega las filas a la tabla. Luego, muestra los valores actualizados de la tabla en un objeto JTextArea.

```
1 /*FUNCION AGREGA FILAS A LA TABLA EXISTENTE*/
2 public void agregarFilaArchivo(File archivo, JTextArea txtArea) {
3     try {
4         DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
5         DocumentBuilder builder = factory.newDocumentBuilder();
6         Document documento = builder.parse(archivo);
7         Element raiz = documento.getDocumentElement();
8         // Obtener el nombre de la etiqueta raiz
9         String nombreEtiquetaRaiz = raiz.getTagName();
10        NodeList nodoLista = documento.getElementsByTagName(nombreEtiquetaRaiz);
11        NodeList lista = nodoLista.item(0).getChildNodes();
12        Estructura st=null;
13        for (int i = 0; i < lista.getLength(); i++) {
14            Node nodo = lista.item(i);
15            if (nodo.getNodeType() == Node.ELEMENT_NODE) {
16                ListaColumna listaFilas = new ListaColumna();
17                String nombre = nodo.getNodeName().replaceAll(" ", "");
18                st = listaEstructura.getEstructura(nombre);
19                int cantidadColumnas = 0;
20                if (st != null) {
21                    cantidadColumnas = st.getColumnas().obtenerCantidad();
22                    NodeList listaColumnas = nodo.getChildNodes();
23                    for (int j = 0; j < listaColumnas.getLength(); j++) {
24                        Node n = listaColumnas.item(j);
25                        if (n.getNodeType() == Node.ELEMENT_NODE) {
26                            String nombreNodo = n.getNodeName().replaceAll(" ", "");
27                            String valorNodo = n.getTextContent().replaceAll(" ", ""); //Valor del nodo
28                            NodoColumna nc = st.getColumnas().getNodo(nombreNodo);
29                            if (nc != null) {
30                                if (!valorNodo.equals("")) {
31                                    String tipo = nc.getTipo();
32                                    if (tipo.equals("int")) {
33                                        if (entero(valorNodo)) {
34                                            listaFilas.insertarFinal(new NodoColumna(nombreNodo, valorNodo));
35                                        } else {
36                                            txtArea.append("El valor no es un entero \n");
37                                            break;
38                                        }
39                                    } else {
40                                        listaFilas.insertarFinal(new NodoColumna(nombreNodo, valorNodo));
41                                    }
42                                } else {
43                                    txtArea.append("La columna " + nombreNodo + ", no contiene dato\n");
44                                }
45                            } else {
46                                txtArea.append("La columna " + nombreNodo + ", no existe\n");
47                                break;
48                            }
49                        }
50                    }
51                    if (listaFilas.obtenerCantidad() == cantidadColumnas) {
52                        st.ingresarFila(listaFilas);
53
54                    } else {
55                        txtArea.append("Los datos no son correctos para crear una fila\n");
56                    }
57                } else {
58                    txtArea.append("La tabla: " + nombre + ", NO EXISTE\n");
59                }
60            }
61        }
62    }
63    st.getArbol().imprimirValores(txtArea);
64    txtArea.append("!Creacion de filas exitosas; \n\n");
65 } catch (ParserConfigurationException | SAXException | IOException ex) {
66 }
67 }
68 }
```

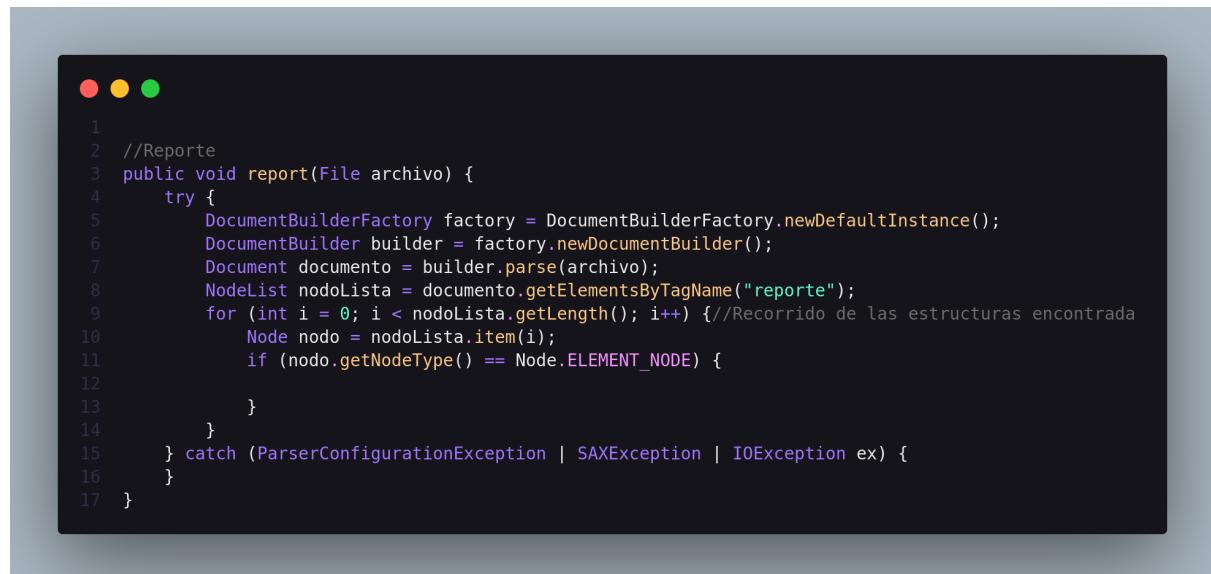
La función eliminarFila permite eliminar una fila de una tabla existente utilizando un archivo XML como referencia. El proceso implica la lectura y análisis del archivo, la obtención del nombre de la tabla, la búsqueda de la estructura correspondiente en la lista de estructuras y la validación de la existencia de la tabla. Si la tabla existe, se recorren los nodos del archivo XML para identificar la llave primaria de la fila a eliminar. A continuación, se realiza la eliminación de la fila correspondiente en el árbol asociado a la estructura de la tabla. Esta función es útil para el manejo de operaciones de eliminación en una base de datos.



```
1 //ELIMINAR
2 public void eliminarFila(File archivo) {
3     try {
4         DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
5         DocumentBuilder builder = factory.newDocumentBuilder();
6         Document documento = builder.parse(archivo);
7         Element raiz = documento.getDocumentElement();
8         // Obtener el nombre de la etiqueta raiz
9         String nombreEtiquetaRaiz = raiz.getTagName();
10        NodeList nodoLista = documento.getElementsByTagName(nombreEtiquetaRaiz);
11        NodeList lista = nodoLista.item(0).getChildNodes();
12        Estructura st = listaEstructura.getEstructura(nombreEtiquetaRaiz);
13        if (st != null) {
14            for (int i = 0; i < lista.getLength(); i++) {
15                Node node = lista.item(i);
16                if (node.getNodeType() == Node.ELEMENT_NODE) {
17                    String nombreNodo = node.getNodeName().replaceAll(" ", "");
18                    String valorNodo = node.getTextContent().replaceAll(" ", "");
19                    if (st.getLlave().equals(nombreNodo)) {
20                        //Buscar en arbol y eliminar
21                    } else {
22                        System.out.println(nombreNodo + ", No es llave primaria");
23                    }
24                }
25            }
26        } else {
27            System.out.println("La tabla de nombre: " + nombreEtiquetaRaiz + ", NO EXISTE");
28        }
29    } catch (ParserConfigurationException | SAXException | IOException ex) {
30    }
31 }
```

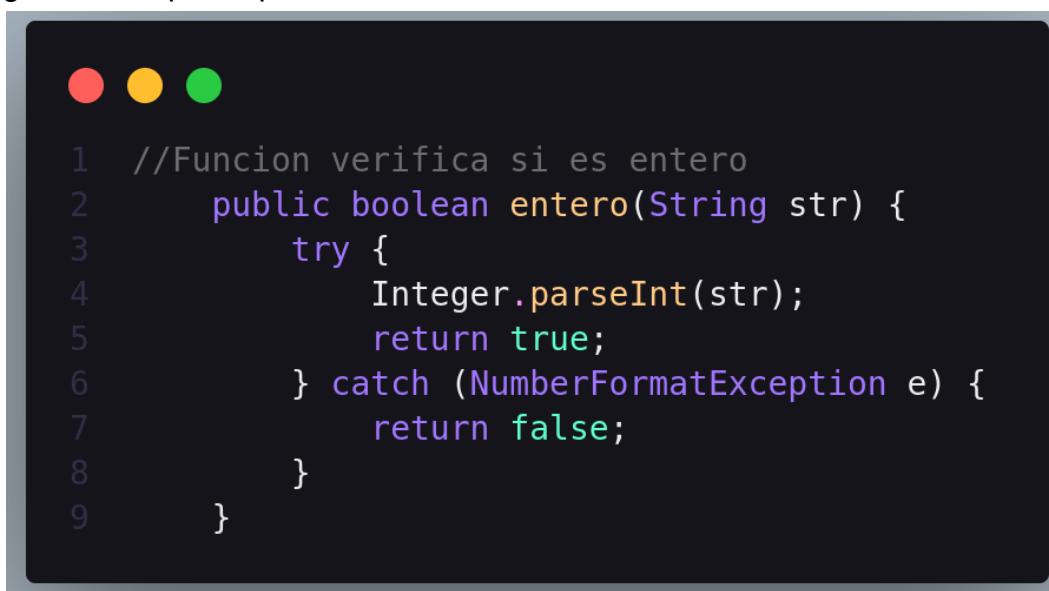
La función report permite generar un informe o reporte utilizando un archivo XML como referencia. En la función, se lee y analiza el archivo XML utilizando un DocumentBuilder. Luego, se obtiene una lista de nodos con el nombre "reporte" del documento. A continuación, se realiza un recorrido de los nodos encontrados en la lista. Dentro del bucle, se puede implementar la lógica necesaria para generar el informe, como la obtención de datos, cálculos, formateo y presentación. En el código proporcionado, actualmente no se realiza ninguna acción dentro del bucle, por lo que se requeriría agregar la lógica específica para generar el informe deseado. Esta función es útil para automatizar la generación de informes basados en datos almacenados en un archivo XML.

Esta función está diseñada para identificar los nodos de tipo "reporte" en un archivo XML, pero no realiza ninguna operación concreta dentro del bloque condicional. Probablemente, se espera que se añada código adicional para realizar el procesamiento deseado y generar el informe correspondiente.



```
1 //Reporte
2 public void report(File archivo) {
3     try {
4         DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
5         DocumentBuilder builder = factory.newDocumentBuilder();
6         Document documento = builder.parse(archivo);
7         NodeList nodoLista = documento.getElementsByTagName("reporte");
8         for (int i = 0; i < nodoLista.getLength(); i++) { //Recorrido de las estructuras encontradas
9             Node nodo = nodoLista.item(i);
10            if (nodo.getNodeType() == Node.ELEMENT_NODE) {
11                ...
12            }
13        }
14    } catch (ParserConfigurationException | SAXException | IOException ex) {
15    }
16 }
17 }
```

La función entero verifica si una cadena de texto puede ser convertida a un número entero. En la función, se intenta convertir la cadena a un entero utilizando el método parseInt de la clase Integer. Si la conversión tiene éxito, es decir, la cadena representa un número entero válido, la función devuelve true. De lo contrario, si ocurre una excepción NumberFormatException, se captura y la función devuelve false, indicando que la cadena no es un número entero válido. Esta función es útil para validar si una cadena contiene un número entero antes de realizar operaciones o asignaciones que requieren un valor numérico entero.



```
1 //Funcion verifica si es entero
2 public boolean entero(String str) {
3     try {
4         Integer.parseInt(str);
5         return true;
6     } catch (NumberFormatException e) {
7         return false;
8     }
9 }
```

La clase ListaEstructura representa una lista enlazada y proporciona funcionalidades para verificar si está vacía. Esta clase también incluye otros métodos para agregar, eliminar o buscar estructuras en la lista.

```
1 public class ListaEstructura {  
2  
3     //Atributo de tipo estructura  
4     private Estructura raiz;  
5  
6     //Apunta a null  
7     public ListaEstructura(){  
8         this.raiz=null;  
9     }  
10  
11    public boolean isEmpty(){  
12        return raiz == null;  
13    }  
14}
```

El método insertarFinal de la clase ListaEstructura permite agregar una nueva estructura al final de la lista enlazada.

```
● ● ●

1  /*Funcion que inserta al final*/
2  public void insertarFIal(Estructura estructura){
3      Estructura nuevaEstructura = estructura;
4
5      if(isEmpty()){
6          raiz =nuevaEstructura;
7          return;
8      }
9      Estructura actual = raiz;
10     while (actual.getSiguiente() != null) {
11         actual=actual.getSiguiente();
12     }
13     actual.setSiguiente(nuevaEstructura);
14 }
15 }
```

En resumen, la función getEstructura recorre la lista enlazada buscando una estructura con un nombre específico y devuelve dicha estructura si se encuentra, o null si no se encuentra.

```
● ● ●

1  /*Funcion de tipo Estructura obtener una estructura,
2   asi obtenerla*/
3  public Estructura getEstructura(String nombre){
4      Estructura actual = raiz;
5      while(actual != null){
6          if (actual.getNombre().equals(nombre)) {
7              return actual;
8          }
9          actual = actual.getSiguiente();
10     }
11     return null;
12 }
```

En resumen, la función imprimirLista recorre la lista enlazada e imprime la información de todas las estructuras presentes en el componente JTextArea proporcionado.



```
1 public void imprimirLista(JTextArea txtArea){
2     Estructura actual = raiz;
3     while (actual!=null) {
4         txtArea.append("Tabla:" + actual.getNombre() + ", Clave: " + actual.getLlave()+"\n", Columnas:\n"+actual.getColumnas().imprimirLista()+"\n");
5         actual=actual.getSiguiente();
6     }
7 }
```

La clase ListaColumna permite crear una lista enlazada de nodos NodoColumna. El atributo raiz apunta al primer nodo de la lista, y el método isEmpty() se utiliza para verificar si la lista está vacía.



```
1 public class ListaColumna {
2
3     private NodoColumna raiz;
4
5     public ListaColumna(){
6         this.raiz = null;
7     }
8
9     public boolean isEmpty(){
10        return raiz == null;
11    }
}
```

El método `getNodo()` busca y devuelve el primer nodo de la lista cuyo nombre coincide con el nombre proporcionado como argumento.

```
● ● ●  
1 //Obtiene un nodo por su nombre, busca el nombre con ese nombre  
2 public NodoColumna getNodo(String nombre){  
3     NodoColumna actual = raiz;  
4     while(actual != null){  
5         if(actual.getNombre().equals(nombre)){  
6             return actual;  
7         }  
8         actual = actual.getSiguiente();  
9     }  
10    return null;  
11}  
12
```

El método `insertarFinal()` agrega un nuevo nodo al final de la lista enlazada. Si la lista está vacía, el nuevo nodo se convierte en la raíz de la lista. Si la lista no está vacía, se recorre la lista hasta llegar al último nodo y se establece el nuevo nodo como el siguiente nodo del último nodo.

```
● ● ●  
1 public void insertarFinal(NodoColumna nodoColumna){  
2     NodoColumna nuevoNodo= nodoColumna;  
3     if (isEmpty()) {  
4         raiz = nuevoNodo;  
5         return;  
6     }  
7  
8     NodoColumna actual =raiz;  
9     while (actual.getSiguiente()!= null) {  
10        actual=actual.getSiguiente();  
11  
12    }  
13    actual.setSiguiente(nuevoNodo);  
14}
```

El método imprimirLista() recorre la lista de nodos y genera una representación en forma de texto de cada nodo, mostrando su nombre y tipo. Luego, devuelve esta representación en forma de cadena.

```
1 //Imprime la lista
2     public String imprimirLista(){
3         String texto="";
4         NodoColumna actual = raiz;
5         while (actual!=null) {
6             texto += "Nombre " + actual.getNombre()+ ", Tipo " + actual.getTipo()+"\n";
7             actual = actual.getSiguiente();
8         }
9
10        return texto;
11    }
```

La anotación @Override indica que el método toString() está sobrescribiendo el método de la clase padre, y dentro de este método se genera una representación en forma de texto de los nodos de la lista.

```
1 /*@Override Indicar que un método en una clase hija está sobrescribiendo un método de la clase padre*/
2     @Override
3     public String toString(){
4         String texto="";
5         NodoColumna actual = raiz;
6         while (actual != null) {
7             texto += actual.getNombre()+ ": " + actual.getTipo()+"\n";
8             actual = actual.getSiguiente();
9         }
10        return texto;
11    }
12
```

La función obtenerCantidad() cuenta y devuelve el número de elementos presentes en la lista enlazada, utilizando un bucle para recorrer la lista y aumentar el contador en cada iteración.

```
● ● ●

1  /*Función que devuelve el número de elementos de la lista enlazada */
2  public int obtenerCantidad() {
3      int cantidad = 0;
4      NodoColumna actual = raiz;
5      while (actual != null) {
6          cantidad++;
7          actual = actual.getSiguiente();
8      }
9      return cantidad;
10 }
```

La clase Estructura representa una tabla en un sistema, con su nombre, columnas, llave primaria y un árbol B + para organizar los datos. Proporciona métodos para ingresar filas en el árbol, obtener y establecer información de la tabla, y gestionar el árbol B +.

```
1  public class Estructura {  
2  
3      /* Atributos */  
4      private String nombre;  
5      private String llave;  
6      private Estructura siguiente;  
7      private ListaColumna columnas;  
8      private ArbolBMas arbol;  
9  
10     /* Constructor vacio */  
11     public Estructura() {  
12         this.siguiente = null;  
13         this.arbol = new ArbolBMas();  
14     }  
15  
16     public void ingresarFila(ListaColumna dato) {  
17         this.arbol.insertar(0, dato);  
18     }  
19  
20     public String getNombre() {  
21         return nombre;  
22     }  
23  
24     public void setNombre(String nombre) {  
25         this.nombre = nombre;  
26     }  
27  
28     public String getLlave() {  
29         return llave;  
30     }  
31  
32     public void setLlave(String llave) {  
33         this.llave = llave;  
34     }  
35  
36     public Estructura getSiguiente() {  
37         return siguiente;  
38     }  
39  
40     public void setSiguiente(Estructura siguiente) {  
41         this.siguiente = siguiente;  
42     }  
43  
44     public ListaColumna getColumnas() {  
45         return columnas;  
46     }  
47  
48     public void setColumnas(ListaColumna columnas) {  
49         this.columnas = columnas;  
50     }  
51  
52     public ArbolBMas getArbol() {  
53         return arbol;  
54     }  
55  
56     public void setArbol(ArbolBMas arbol) {  
57         this.arbol = arbol;  
58     }  
59 }  
60 }
```

La clase NodoColumna representa un nodo en la lista enlazada que almacena información sobre una columna de una estructura. Cada nodo contiene el nombre y tipo de la columna, así como una referencia al siguiente nodo en la lista.

```
● ● ●

1  public class NodoColumna {
2      private String tipo;
3      private String nombre;
4      private NodoColumna siguiente;
5
6      public NodoColumna(String nombre, String tipo){
7          this.nombre=nombre;
8          this.tipo=tipo;
9          this.siguiente=null;
10     }
11
12     public String getTipo() {
13         return tipo;
14     }
15
16     public void setTipo(String tipo) {
17         this.tipo = tipo;
18     }
19
20     public String getNombre() {
21         return nombre;
22     }
23
24     public void setNombre(String nombre) {
25         this.nombre = nombre;
26     }
27
28     public NodoColumna getSiguiente() {
29         return siguiente;
30     }
31
32     public void setSiguiente(NodoColumna siguiente) {
33         this.siguiente = siguiente;
34     }
35 }
```

Creación de la clase árbol b + en el cual tiene dos atributos uno de tipo NodoArbolBMas y un de tipo entero y un constructor el cual inicializa los atributos creados.

```
1 public class ArbolBMas {  
2  
3     private NodoArbolBMas raiz;  
4     private int orden;  
5  
6     // Constructor:  
7     public ArbolBMas() {  
8         this.orden = 200; // - Crea un nuevo árbol B+ con un orden predeterminado .  
9         this.raiz = new NodoArbolBMas(orden, true); // - Inicializa la raíz como un nuevo nodo hoja.  
10    }  
11}
```

Función que permite buscar cuando el nodo ya sobrepasó su orden, entonces llama a la función dividirNodos.

```
1 public void insertar(int llave, ListaColumna valor) {  
2     NodoArbolBMas nodo = raiz;  
3     if (nodo.getNumLlaves() == 2 * orden - 1) { // Verificar si el nodo actual está lleno  
4         // Crear una nueva raíz y asignar la raíz actual como su hijo  
5         NodoArbolBMas nuevaRaiz = new NodoArbolBMas(orden, false);  
6         nuevaRaiz.getHijos()[0] = raiz;  
7         raiz = nuevaRaiz;  
8         nodo.dividirNodos(orden, nuevaRaiz.getHijos()[0]); // Dividir el nodo actual y actualizar la raíz  
9         insertarNoCompleto(nuevaRaiz, llave, valor); // Insertar el par de valores en el nuevo nodo raíz o en sus hijos  
10    } else {  
11        insertarNoCompleto(nodo, llave, valor); // Si el nodo no está lleno, insertar el par de valores en el nodo actual o en sus hijos  
12    }  
13}
```

La función se encarga de insertar de manera adecuada una nueva llave y valor en un árbol B + no completo, realizando ajustes y divisiones de nodos según sea necesario para mantener el equilibrio de la estructura del árbol.

```
● ● ●
1 private void insertarNoCompleto(NodoArbolBMas nodo, int llave, ListaColumna valor) {
2     int i = nodo.getNumLlaves() - 1;
3     if (nodo.isEsHoja()) { // Si el nodo es una hoja
4         // Desplazarse hacia atrás hasta encontrar la posición adecuada para la nueva llave
5         while (i >= 0 && llave < nodo.getLlaves()[i]) {
6             nodo.getLlaves()[i + 1] = nodo.getLlaves()[i]; // Desplazar la llave hacia la derecha
7             nodo.getValores()[i + 1] = nodo.getValores()[i]; // Desplazar el valor hacia la derecha
8             i--;
9         }
10        // Insertar la nueva llave y valor en la posición correcta
11        nodo.getLlaves()[i + 1] = llave;
12        nodo.getValores()[i + 1] = valor;
13        nodo.setNumLlaves(nodo.getNumLlaves() + 1); // Incrementar el número de llaves en el nodo
14    } else {
15        // Si el nodo no es una hoja
16        // Buscar el hijo adecuado para la inserción
17        while (i >= 0 && llave < nodo.getLlaves()[i]) {
18            i--;
19        }
20        i++; // Incrementar i para obtener el hijo correcto
21        // Verificar si el hijo tiene el número máximo de llaves permitidas
22        if (nodo.getHijos()[i].getNumLlaves() == 2 * orden - 1) {
23            // Realizar una división de nodos para mantener el equilibrio del árbol
24            nodo.dividirNodos(i, nodo.getHijos()[i]);
25            // Ajustar i según la posición de la nueva llave en el nodo actualizado
26            if (llave > nodo.getLlaves()[i]) {
27                i++;
28            }
29        }
30        // Llamar recursivamente a insertNonFull en el hijo correspondiente
31        insertarNoCompleto(nodo.getHijos()[i], llave, valor);
32    }
33}
```

La función buscar realiza una búsqueda descendente desde la raíz hasta una hoja en el árbol B +, siguiendo el camino adecuado según el valor de la llave buscada. Luego, llama al método de búsqueda en el nodo hoja correspondiente y devuelve el resultado de la búsqueda.

```
● ○ ● ●  
1 public Object buscar(int llave) {  
2     NodoArbolBMas nodo = raiz; // Comenzar la búsqueda desde la raíz del árbol  
3     // Mientras no se llegue a una hoja  
4     while (!nodo.isEsHoja()) {  
5         int i = 0;  
6         // Buscar la posición adecuada para la llave en el nodo actual  
7         while (i < nodo.getNumLlaves() && llave >= nodo.getLlaves()[i]) {  
8             i++;  
9         }  
10        // Moverse al hijo correspondiente  
11        nodo = nodo.getHijos()[i];  
12    }  
13    // Una vez se llega a una hoja, llamar al método buscar en el nodo hoja  
14    return nodo.buscar(llave);  
15}  
16}
```

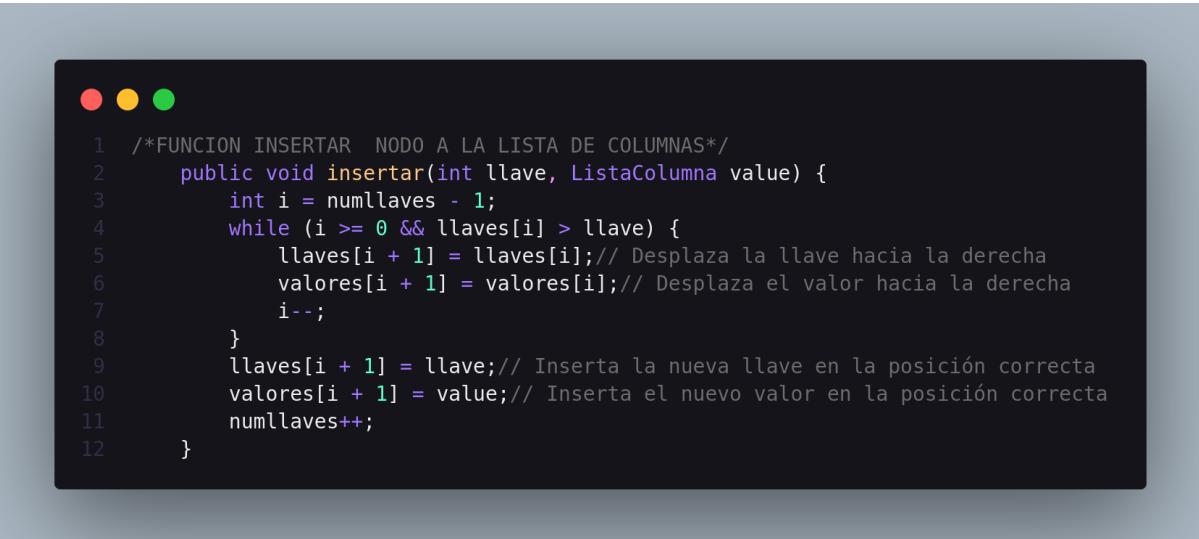
La función imprimirValores recorre las hojas del árbol B + y muestra en el JTextArea todos los valores asociados a las filas almacenadas en el árbol.

```
● ○ ● ●  
1 //Funcion que me imprime todas las filas creadas  
2 public void imprimirValores(JTextArea txtArea) {  
3     NodoArbolBMas nodo = raiz;  
4     while (!nodo.isEsHoja()) {  
5         nodo = nodo.getHijos()[0];  
6     }  
7     while (nodo != null) {  
8         for (int i = 0; i < nodo.getNumLlaves(); i++) {  
9             txtArea.append(nodo.getValores()[i].toString());  
10            txtArea.append("\n\n");  
11        }  
12        nodo = nodo.getSiguienteHoja();  
13    }  
14}
```

La clase NodoArbolBMas representa un componente fundamental en un árbol B+, que se utiliza para organizar y almacenar datos eficientemente. Cada nodo contiene llaves, valores y referencias a otros nodos, y su estructura jerárquica permite realizar búsquedas y consultas de manera rápida en el árbol.

```
● ● ●
1  public class NodoArbolBMas {
2
3      private int[] llaves;
4      private ListaColumna[] valores;
5      private NodoArbolBMas[] hijos;
6      private boolean esHoja;
7      private int numllaves; //Número de hijos que puede tener
8      private NodoArbolBMas siguienteHoja;
9
10
11
12     //CONSTRUCTOR
13     public NodoArbolBMas(int orden, boolean esHoja) {
14         this.llaves = new int[orden];
15         this.valores = new ListaColumna[orden];
16         this.hijos = new NodoArbolBMas[orden + 1];
17         this.esHoja = esHoja;
18         this.numllaves = 0;
19         this.siguienteHoja = null;
20     }
21
22     //GET Y SET
23
24     public int[] getLlaves() {
25         return llaves;
26     }
27
28     public void setLlaves(int[] llaves) {
29         this.llaves = llaves;
30     }
31
32     public ListaColumna[] getValores() {
33         return valores;
34     }
35
36     public void setValores(ListaColumna[] valores) {
37         this.valores = valores;
38     }
39
40     public NodoArbolBMas[] getHijos() {
41         return hijos;
42     }
43
44     public void setHijos(NodoArbolBMas[] hijos) {
45         this.hijos = hijos;
46     }
47
48     public boolean isEsHoja() {
49         return esHoja;
50     }
51
52     public void setEsHoja(boolean esHoja) {
53         this.esHoja = esHoja;
54     }
55
56     public int getNumllaves() {
57         return numllaves;
58     }
59
60     public void setNumllaves(int numllaves) {
61         this.numllaves = numllaves;
62     }
63
64     public NodoArbolBMas getSiguienteHoja() {
65         return siguienteHoja;
66     }
67
68     public void setSiguienteHoja(NodoArbolBMas siguienteHoja) {
69         this.siguienteHoja = siguienteHoja;
70     }
}
```

La función insertar se encarga de insertar una nueva llave y su valor asociado en el nodo, manteniendo las llaves ordenadas de forma ascendente. Esto implica desplazar las llaves y valores existentes si es necesario para hacer espacio para la nueva llave y valor.



```
1 /*FUNCION INSERTAR NODO A LA LISTA DE COLUMNAS*/
2     public void insertar(int llave, ListaColumna value) {
3         int i = numllaves - 1;
4         while (i >= 0 && llaves[i] > llave) {
5             llaves[i + 1] = llaves[i];// Desplaza la llave hacia la derecha
6             valores[i + 1] = valores[i];// Desplaza el valor hacia la derecha
7             i--;
8         }
9         llaves[i + 1] = llave;// Inserta la nueva llave en la posición correcta
10        valores[i + 1] = value;// Inserta el nuevo valor en la posición correcta
11        numllaves++;
12    }
```

La función buscar recorre las llaves del nodo en orden ascendente y compara cada llave con la llave buscada. Si encuentra una coincidencia, devuelve el valor correspondiente; de lo contrario, devuelve null. Esto permite buscar un valor asociado a una llave dentro del nodo.



```
1 /*FUNCION BUSCAR*/
2     public Object buscar(int llave) {
3         int i = 0;
4         while (i < numllaves && llave > llaves[i]) {
5             i++;
6         }
7         if (i < numllaves && llave == llaves[i]) {
8             return valores[i];// Si se encuentra la llave, se devuelve el valor correspondiente
9         } else {
10             return null;// Si no se encuentra la llave, se devuelve null
11         }
12     }
```

La función dividirNodos se encarga de dividir un nodo en dos partes, creando un nodo hermano derecho (rightSibling). Se mueven las llaves, los valores y los hijos correspondientes según sea necesario, y se actualizan las referencias de la siguiente hoja. Esta operación se utiliza durante la inserción de una nueva llave en un árbol B+ para mantener el equilibrio y la estructura adecuada del árbol.

```
1 public void dividirNodos(int splitIndex, NodoArbolBMas rightSibling) {
2     // Establecer el número de llaves del hermano derecho
3     rightSibling.numllaves = numllaves - splitIndex;
4     // Mover las llaves y los valores al hermano derecho y limpiar los elementos en el nodo actual
5     for (int i = 0; i < rightSibling.numllaves; i++) {
6         rightSibling.llaves[i] = llaves[splitIndex + i];
7         rightSibling.valores[i] = valores[splitIndex + i];
8         llaves[splitIndex + i] = 0;
9         valores[splitIndex + i] = null;
10    }
11    // Si el nodo actual no es una hoja, mover los hijos al hermano derecho y limpiar los elementos en el nodo actual
12    if (!esHoja) {
13        for (int i = 0; i < rightSibling.numllaves + 1; i++) {
14            rightSibling.hijos[i] = hijos[splitIndex + i];
15            hijos[splitIndex + i] = null;
16        }
17    }
18    // Actualizar el número de llaves en el nodo actual
19    numllaves = splitIndex;
20    // Establecer las referencias de la siguiente hoja para mantener la estructura de la lista
21    rightSibling.siguienteHoja = siguienteHoja;
22    siguienteHoja = rightSibling;
23}
```

El método `toString` en esta clase devuelve una representación en cadena de los elementos contenidos en el arreglo de valores. Esto permite obtener una visualización más legible y comprensible de los valores almacenados en el nodo al llamar al método `toString` en una instancia de la clase `NodoArbolBMas`.

```
1 /*toString es un metodo de java, obtiene la informacion del objeto por defecto
2  * me trae el espacio en memoria*/
3 @Override
4 public String toString() {
5     String texto = "";
6     for (int i = 0; i < valores.length; i++) {
7         texto += valores[i].toString(); // Concatena la representación en cadena de cada elemento en "valores"
8     }
9     return texto; // Devuelve la concatenación de las representaciones en cadena de los elementos en "valores"
10 }
```

Se creó la funcionalidad para poder leer archivo para carga de datos, como lo es carga masiva, cargar los datos, cargar la fila a eliminar y los reportes. JFileChooser nos proporciona una interfaz gráfica que permite al usuario seleccionar archivos.

Este código muestra un cuadro de diálogo que permite al usuario seleccionar un archivo con extensión ".dat" ,".rpt", ".xml". Una vez que el usuario realiza la selección, se obtiene el archivo seleccionado para su posterior procesamiento almacenándolo en la variable archivo.

```
/*
 * Generated by IntelliJ IDEA
 */
@SuppressWarnings("unchecked")
Generated Code

private void btnReportesActionPerformed(java.awt.event.ActionEvent evt) {
    JFileChooser fileChooser = new JFileChooser();
    FileNameExtensionFilter filter = new FileNameExtensionFilter(description: "Archivos DAT", extensions: "rpt");
    fileChooser.setFileFilter(filter);
    fileChooser.setFileSelectionMode(mode.JFileChooser.FILES_ONLY);
    int result = fileChooser.showOpenDialog(parent: this);
    if (result != JFileChooser.CANCEL_OPTION) {
        File archivo = fileChooser.getSelectedFile();
    }
}

//BOTON PARA CARGAR ARCHIVO DE VARIAS ESTRUCTURAS
private void btnCargaMasivaActionPerformed(java.awt.event.ActionEvent evt) {
    JFileChooser fileChooser = new JFileChooser();
    FileNameExtensionFilter filter = new FileNameExtensionFilter(description: "Archivos XML", extensions: "xml");
    fileChooser.setFileFilter(filter);
    fileChooser.setFileSelectionMode(mode.JFileChooser.FILES_ONLY);
    int result = fileChooser.showOpenDialog(parent: this);
    if (result != JFileChooser.CANCEL_OPTION) {
        File archivo = fileChooser.getSelectedFile();
        la.ingresarEstructuraArchivo(archivo, txtArea: TextAreaCargaMasiva);
    }
}

/*BOTON PARA CARGAR LOS ATRIBUTOS DE LA TABLA, AGREGA FILAS */
private void btnCargarArchivoDatosActionPerformed(java.awt.event.ActionEvent evt) {
    JFileChooser fileChooser = new JFileChooser();
    FileNameExtensionFilter filter = new FileNameExtensionFilter(description: "Archivos DAT", extensions: "dat");
    fileChooser.setFileFilter(filter);
    fileChooser.setFileSelectionMode(mode.JFileChooser.FILES_ONLY);
    int result = fileChooser.showOpenDialog(parent: this);
    if (result != JFileChooser.CANCEL_OPTION) {
        File archivo = fileChooser.getSelectedFile();
        la.agregarFilaArchivo(archivo, txtArea: txtAreaCargaDatos);
    }
}

Run - Run (DBMS) ×
```