CLASE NODO

```
public class Nodo {
    String placa;
    String color;
    String linea;
    int modelo;
    String propietario;
    Nodo arriba;
    Nodo abajo;
    Nodo izquierda;
    Nodo derecha;

    public Nodo(String placa, String color, String linea, int modelo, String
propietario) {
        this.placa = placa;
        this.color = color;
        this.linea = linea;
        this.modelo = modelo;
        this.propietario = propietario;
        this.arriba = null;
        this.abajo = null;
        this.izquierda = null;
        this.derecha = null;
    }
}
```

CLASE MATRIZ

```
public class Matriz {
    Nodo raiz;



    public Matriz() {
        this.raiz = null;


    }
```

```
    public void insertar(String placa, String color, String linea, int modelo,
String propietario) {


       Nodo nuevo = new Nodo(placa, color, linea, modelo, propietario);

        System.out.println("Nodo agregado: " + nuevo.placa + ", " + nuevo.color +
", " + nuevo.linea + ", " + nuevo.modelo + ", " + nuevo.propietario);

        if (raiz == null) {
            // Si la matriz está vacía el nuevo nodo será la raíz
            raiz = nuevo;
```

```
        } else {
            // Buscar la posición adecuada para el nuevo nodo
            Nodo actual = raiz;
            Nodo anterior = null;

            System.out.println("Buscando posición para el nuevo nodo...");

            while (actual != null && actual.modelo < modelo) {
                anterior = actual;
                actual = actual.abajo;
                System.out.println("Actual: " + actual);
                System.out.println("Anterior: " + anterior);
            }

            System.out.println("Se encontró la posición adecuada para el
modelo...");


            while (actual != null && actual.linea.compareTo(nuevo.linea) < 0) {
                anterior = actual;
                actual = actual.derecha;
                System.out.println("Actual: " + actual);
                System.out.println("Anterior: " + anterior);
            }

            System.out.println("Se encontró la posición adecuada para la
línea...");


            //No permite valores duplicados
            if (actual != null && actual.modelo == modelo &&
actual.linea.equals(nuevo.linea) && actual.color.equals(nuevo.color) &&
actual.placa.equals(nuevo.placa)) {
                System.out.println("El nodo ya existe en la matriz");
                return;
            }
            /**/

            System.out.println("Posición encontrada: " + anterior + ", " +
actual);


            //
            if (anterior == null) {
                raiz = nuevo;
            } else if (anterior.abajo == null) {
                anterior.abajo = nuevo;
                nuevo.arriba = anterior;
            } else {
                nuevo.abajo = actual;
                actual.arriba = nuevo;
                nuevo.arriba = anterior;
                anterior.abajo = nuevo;
            }
```

```
            System.out.println("Nodo agregado en la dirección vertical...");


            //
            if (anterior == null) {
                raiz = nuevo;
            } else if (anterior.derecha == null) {
                anterior.derecha = nuevo;
                nuevo.izquierda = anterior;
            } else {
                nuevo.derecha = actual;
                actual.izquierda = nuevo;
                nuevo.izquierda = anterior;
                anterior.derecha = nuevo;
            }


            System.out.println("Nodo agregado en la dirección horizontal...");

            System.out.println("--------------------------");


        }
    }


    public List<Nodo> buscar(String placa, String color, String linea, int
modelo, String propietario) {
        List<Nodo> resultados = new ArrayList<>();
        Set<Nodo> nodosAgregados = new HashSet<>();

        Nodo actual = raiz;
        while (actual != null) {
            Nodo temp = actual;
            while (temp != null) {
                if ((placa == null || temp.placa.equals(placa))
                        && (color == null || temp.color.equals(color))
                        && (linea == null || temp.linea.equals(linea))
                        && (modelo == -1 || temp.modelo == modelo)
                        && (propietario == null ||
temp.propietario.equals(propietario))) {
                    if (!nodosAgregados.contains(temp)) {
                        resultados.add(temp);
                        nodosAgregados.add(temp);


                    }
                }
                temp = temp.derecha;
            }
            actual = actual.abajo;
        }

        return resultados;
    }
```

```
    public void eliminar(String placa, String color, String linea, int modelo,
String propietario) {
        List<Nodo> nodosAEliminar = buscar(placa, color, linea, modelo,
propietario);
        if (nodosAEliminar.isEmpty()) {
            System.out.println("No se encontraron nodos para eliminar");
            return;
        }
        for (Nodo nodoActual : nodosAEliminar) {
            String placaEliminada = nodoActual.placa;

            Nodo nodoArriba = nodoActual.arriba;
            Nodo nodoAbajo = nodoActual.abajo;
            Nodo nodoIzquierda = nodoActual.izquierda;
            Nodo nodoDerecha = nodoActual.derecha;

            if (nodoArriba != null) {
                nodoArriba.abajo = nodoAbajo;
            }
            if (nodoAbajo != null) {
                nodoAbajo.arriba = nodoArriba;
            }
            if (nodoIzquierda != null) {
                nodoIzquierda.derecha = nodoDerecha;
            }
            if (nodoDerecha != null) {
                nodoDerecha.izquierda = nodoIzquierda;
            }

            if (nodoActual == raiz) {
                raiz = nodoActual.derecha != null ? nodoActual.derecha :
nodoActual.abajo;
            }

            nodoActual = null;

            System.out.println("Se eliminó correctamente la placa " +
placaEliminada);

        }

        System.out.println("Se han eliminado " + nodosAEliminar.size() + "
nodos");

    }




}
```

CLASE INICIO

```
public class Inicio {

	public static void main(String[] args) {

		Matriz matriz = new Matriz();
		insertarNodo(matriz);
	    buscarNodo(matriz);
	    eliminarNodo(matriz);



	}

	public static void insertarNodo(Matriz matriz) {



		System.out.println("Iniciando metodo insertar...");

		matriz.insertar("FBC123", "Rojo", "Sedán", 2018, "Gerson Escobar");
		matriz.insertar("DEF456", "Azul", "Camioneta", 2020, "Marco
Tohom");
		matriz.insertar("GHI789", "Verde", "Hatchback", 2021, "Kevin
Cruz");
		matriz.insertar("JKL012", "Negro", "SUV", 2019, "VAni Alcantara");
		matriz.insertar("MNO345", "Rojo", "Sedán", 2022, "Jose Gonzalez");
		matriz.insertar("PQR678", "Verde", "Hatchback", 2021, "Sofia
Hernandez");

	}


	public static void eliminarNodo(Matriz matriz) {

		System.out.println("");

		System.out.println("Iniciando metodo eliminar...");

		matriz.eliminar(null, "Rojo", null, -1, null);



	}


	public static void buscarNodo(Matriz matriz) {

		System.out.println("");
		System.out.println("Iniciando el metodo buscar...");
		System.out.println("Buscando resultados...");

		List<Nodo> resultados = new ArrayList<>();
```

```
            resultados.addAll(matriz.buscar(null, "Rojo", null, -1,null));



            System.out.println("Se encontraron " + resultados.size() + "
resultados:");

            for (Nodo resultado : resultados) {
                  System.out.println("Placa: " + resultado.placa + ", Color: "
+ resultado.color + ", Línea: " + resultado.linea + ", Modelo: " +
resultado.modelo + ", Propietario: " + resultado.propietario);
            }
      }

}
```

LO QUE SE MUESTRA EN CONSOLA

```
Iniciando metodo insertar...
Nodo agregado: FBC123, Rojo, Sedán, 2018, Juan Perez
Nodo agregado: DEF456, Azul, Camioneta, 2020, Maria Rodriguez
Buscando posición para el nuevo nodo...
Actual: null
Anterior: matrices.Nodo@5e265ba4
Se encontró la posición adecuada para el modelo...
Se encontró la posición adecuada para la línea...
Posición encontrada: matrices.Nodo@5e265ba4, null
Nodo agregado en la dirección vertical...
Nodo agregado en la dirección horizontal...
--------------------------
Nodo agregado: GHI789, Verde, Hatchback, 2021, Pedro Gomez
Buscando posición para el nuevo nodo...
Actual: matrices.Nodo@156643d4
Anterior: matrices.Nodo@5e265ba4
Actual: null
Anterior: matrices.Nodo@156643d4
Se encontró la posición adecuada para el modelo...
Se encontró la posición adecuada para la línea...
Posición encontrada: matrices.Nodo@156643d4, null
Nodo agregado en la dirección vertical...
Nodo agregado en la dirección horizontal...
--------------------------
Nodo agregado: JKL012, Negro, SUV, 2019, Ana Martinez
Buscando posición para el nuevo nodo...
Actual: matrices.Nodo@156643d4
Anterior: matrices.Nodo@5e265ba4
Se encontró la posición adecuada para el modelo...
Actual: matrices.Nodo@123a439b
Anterior: matrices.Nodo@156643d4
Actual: null
Anterior: matrices.Nodo@123a439b
Se encontró la posición adecuada para la línea...
Posición encontrada: matrices.Nodo@123a439b, null
```

```
Nodo agregado en la dirección vertical...
Nodo agregado en la dirección horizontal...
--------------------------
Nodo agregado: MNO345, Rojo, Sedán, 2022, Jose Gonzalez
Buscando posición para el nuevo nodo...
Actual: matrices.Nodo@156643d4
Anterior: matrices.Nodo@5e265ba4
Actual: matrices.Nodo@123a439b
Anterior: matrices.Nodo@156643d4
Actual: matrices.Nodo@7de26db8
Anterior: matrices.Nodo@123a439b
Actual: null
Anterior: matrices.Nodo@7de26db8
Se encontró la posición adecuada para el modelo...
Se encontró la posición adecuada para la línea...
Posición encontrada: matrices.Nodo@7de26db8, null
Nodo agregado en la dirección vertical...
Nodo agregado en la dirección horizontal...
--------------------------
Nodo agregado: PQR678, Verde, Hatchback, 2021, Sofia Hernandez
Buscando posición para el nuevo nodo...
Actual: matrices.Nodo@156643d4
Anterior: matrices.Nodo@5e265ba4
Actual: matrices.Nodo@123a439b
Anterior: matrices.Nodo@156643d4
Se encontró la posición adecuada para el modelo...
Se encontró la posición adecuada para la línea...
Posición encontrada: matrices.Nodo@156643d4, matrices.Nodo@123a439b
Nodo agregado en la dirección vertical...
Nodo agregado en la dirección horizontal...
--------------------------

Iniciando el metodo buscar...
Buscando resultados...
Se encontraron 2 resultados:
Placa: FBC123, Color: Rojo, Línea: Sedán, Modelo: 2018, Propietario: Juan Perez
Placa: MNO345, Color: Rojo, Línea: Sedán, Modelo: 2022, Propietario: Jose
Gonzalez

Iniciando metodo eliminar...
Se eliminó correctamente la placa FBC123
Se eliminó correctamente la placa MNO345
Se han eliminado 2 nodos
```

DOCUMENTACION TECNICA

```
# Documentación Técnica

### Clase Nodo

 - **Variables:**
```

- **placa:** Variable de tipo String que representa la placa del vehículo.

- **color:** Variable de tipo String que representa el color del vehículo.

- **linea:** Variable de tipo String que representa la línea del vehículo.

- **modelo:** Variable de tipo Int que representa el modelo del vehículo.

- **propietario:** Variable de tipo String que representa el propietario del vehículo.

- **arriba:** Variable de tipo Nodo que representa el nodo ubicado arriba.

- **abajo:** Variable de tipo Nodo que representa el nodo ubicado abajo.

- **izquierda:** Variable de tipo Nodo que representa el nodo ubicado a la izquierda.

- **derecha:** Variable de tipo Nodo que representa el nodo ubicado a la derecha.

Clase Matriz:

- **Variables:**

  - **raiz:** Variable de tipo Nodo que representa el nodo raíz de la estructura de datos. En una matriz, la raíz se refiere al primer nodo de la estructura, que no tiene nodos verticalmente ni a la horizontalmente, y a partir del cual se construye la matriz mediante la inserción de nuevos nodos.

- **Métodos:**

  - **Insertar:** Método con parámetro que permite insertar un nuevo nodo en una matriz.

    Se crea un nuevo objeto Nodo con los datos del vehículo a insertar (Estos datos son la placa, el color, la linea, el modelo y el propietario que son recibidos a través de un parametro). Si la raíz de la matriz es

nula, el nuevo nodo se convierte en la raíz. De lo contrario, se busca la posición adecuada para el nuevo nodo en la matriz. Se hace esto recorriendo la matriz hacia abajo hasta encontrar la fila adecuada según el modelo y luego recorriendo la fila hacia la derecha hasta encontrar la posición adecuada según la línea.

    En cada paso del recorrido se guarda el nodo actual y el nodo anterior. Se verifica si el nodo ya existe en la matriz. Si es así, se muestra un mensaje y se sale del método. Sino, se inserta el nuevo nodo en la matriz. (Si el nodo anterior es nulo, entonces el nuevo nodo se convierte en la raíz. De lo contrario, se agrega el nuevo nodo a la posición adecuada en la dirección vertical y horizontal). Se muestra un mensaje de éxito en la inserción del nuevo nodo.

  - **Buscar:** Método con parámetro que realiza una búsqueda en la matriz.

    La busqueda puede ser por placa, color, linea, modelo o propietario que son recibidos a traves de un parámetro, recorriendo sus nodos en orden de izquierda a derecha y de arriba a abajo, verificando si cada nodo cumple con los criterios de búsqueda especificados.

    Si un nodo cumple con los criterios y no ha sido agregado previamente a la lista de resultados, se agrega a dicha lista y se marca como agregado en un conjunto para evitar duplicados.

    Al finalizar el recorrido, se retorna la lista de resultados.

  - **Eliminar:** Método con parámetro que se encarga de remover los nodos de la matriz que cumplan con los criterios especificados por el usuario.

    Recibe como parámetros la placa, color, línea, modelo y propietario del vehículo a eliminar.

    El método primero llama al método buscar, que retorna una lista de nodos que coinciden con los criterios de búsqueda especificados. Si la lista está vacía, significa que no se encontraron nodos para eliminar y se imprime un mensaje por consola. En caso contrario, el método itera sobre cada uno de los nodos encontrados y procede a removerlos de la matriz.

    Para cada nodo a eliminar, se guardará la placa en una variable auxiliar para posteriormente imprimir un mensaje que indique cuál placa se eliminó.

    Se actualizan las referencias a los nodos adyacentes del nodo actual que se está eliminando y, en caso de que sea la raíz, se actualiza la referencia a la raíz. Finalmente, se elimina el nodo actual asignándole el valor nulo.

Después de eliminar los nodos, se imprime un mensaje indicando la cantidad de nodos que se eliminaron.

Clase Inicio:

La clase Inicio es una clase que contiene el método main y que sirve para iniciar la aplicación. Se utiliza para probar los métodos de la clase Matriz y para verificar su correcto funcionamiento.

- **Métodos:**

  - **Main:** Este método crea una instancia de la clase Matriz y luego llama a los métodos insertarNodo, buscarNodo y eliminarNodo.

  - **insertarNodo:** Este método recibe como parámetro una instancia de la clase Matriz. Este método se encarga de llamar al método insertar de la clase Matriz varias veces para insertar nodos en la estructura.

  - **buscarNodo:** Este método recibe como parámetro una instancia de la clase Matriz. Este método se encarga de llamar al método buscar de la clase Matriz para buscar nodos en la estructura. Luego, muestra por consola los resultados obtenidos.

  - **eliminarNodo:** Este método recibe como parámetro una instancia de la clase Matriz. Este método se encarga de llamar al método eliminar de la clase Matriz para eliminar nodos de la estructura. En este caso, el método eliminar recibe como argumentos null, "Rojo", null, -1, null para eliminar todos los nodos que sean de color rojo.

DOCUMENTACIÓN DE USUARIO

Documentación de Usuario

El código proporcionado es una implementación de una matriz de nodos, la cual es una estructura de datos que organiza y almacena datos en forma de una tabla. En este caso, los nodos representan vehículos y se almacenan en la matriz según su modelo, línea y propietario.

La clase Nodo tiene las siguientes propiedades:

- placa: la placa del vehículo (cadena de caracteres)

- color: el color del vehículo (cadena de caracteres)

- linea: la línea del vehículo (cadena de caracteres)

- modelo: el modelo del vehículo (entero)

- propietario: el nombre del propietario del vehículo (cadena de caracteres)

- arriba: el nodo que está arriba del nodo actual en la matriz (puntero a otro Nodo)

- abajo: el nodo que está abajo del nodo actual en la matriz (puntero a otro Nodo)

- izquierda: el nodo que está a la izquierda del nodo actual en la matriz (puntero a otro Nodo)

- derecha: el nodo que está a la derecha del nodo actual en la matriz (puntero a otro Nodo)

La clase Matriz tiene las siguientes propiedades:

- raiz: el nodo raíz de la matriz (puntero a un Nodo)

La clase Matriz tiene los siguientes métodos:

- Matriz(): el constructor de la clase Matriz que inicializa la raíz en null.

- insertar(String placa, String color, String linea, int modelo, String propietario): este método permite insertar un nuevo nodo en la matriz, representando un nuevo vehículo. Los parámetros son la placa, color, línea, modelo y propietario del vehículo. Si la matriz está vacía, el nuevo nodo se convierte en la raíz. De lo contrario, el método busca la posición adecuada para el nuevo nodo y lo agrega. No se permiten valores duplicados.

- buscar(String placa, String color, String linea, int modelo, String propietario): este método busca nodos en la matriz que coincidan con los parámetros especificados. Los parámetros son opcionales, por lo que se pueden omitir. Si se omite un parámetro, se ignorará en la búsqueda. El método devuelve una lista de nodos que coinciden con los parámetros especificados.

- eliminar(String placa, String color, String linea, int modelo, String propietario): este método elimina los nodos de la matriz que coinciden con los parámetros especificados. Los parámetros son opcionales, por lo que se pueden omitir. Si se omite un parámetro, se ignorará en la búsqueda. El

método devuelve un mensaje si no se encontraron nodos para eliminar. Si se eliminan nodos, se imprimirá la placa del vehículo eliminado.

Es importante tener en cuenta que el código es sensible a mayúsculas y minúsculas en los valores de cadena de caracteres. Además, es importante asegurarse de que los parámetros se pasen en el orden correcto.