

Universidad Mariano Gálvez de Guatemala
Facultad de Ingeniería en Sistemas de Información y Ciencias de la
Comunicación
Programación III
Ing. Melvin Cali



Proyecto III

Melannie Rosse Lorenzana Ajanel 7690-23-21245

Christian Emanuel García Figueroa 7690-23-23196

Carlos Guillermo Ardón Garrido 7690-23-21855

Guatemala 15 de mayo del 2025

<https://github.com/Melannie24/Proyecto3/tree/master>

Código Fuente

Clase Matriz Ortogonal

```
import java.util.ArrayList;
import java.util.List;

public class MatrizOrtogonal02 {
    private Nodo cabeceraFilas;
    private Nodo cabeceraColumnas;

    public MatrizOrtogonal02() {
        cabeceraFilas = new Nodo("FILAS", true);
        cabeceraColumnas = new Nodo("COLUMNAS", false);
    }

    public Nodo getCabeceraFilas() {
        return cabeceraFilas;
    }

    public Nodo getCabeceraColumnas() {
        return cabeceraColumnas;
    }

    public boolean insertar(String placa, String color, String linea, String modelo, String propietario)
    {
        if (!validarPlacaGuatemala(placa)) {
            System.err.println("error: Placa invalida segun formato.");
            return false;
        }
        if (existePlaca(placa)) {
            System.err.println("error: La placa ya existe.");
            return false;
        }

        Nodo nodoModelo = buscarCabeceraFila(modelo);
        if (nodoModelo == null) nodoModelo = insertarCabeceraFila(modelo);

        Nodo nodoPropietario = buscarCabeceraColumna(proprietario);
        if (nodoPropietario == null) nodoPropietario = insertarCabeceraColumna(proprietario);

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

        // Insertar en fila (modelo)
        Nodo actualFila = nodoModelo;
        while (actualFila.derecha != null) actualFila = actualFila.derecha;
        actualFila.derecha = nuevo;
        nuevo.izquierda = actualFila;
    }
}
```

```

        // Insertar en columna (propietario)
        Nodo actualCol = nodoPropietario;
        while (actualCol.abajo != null) actualCol = actualCol.abajo;
        actualCol.abajo = nuevo;

        nuevo.arriba = actualCol;
        return true;
    }

    private Nodo buscarCabeceraFila(String modelo) {
        Nodo actual = cabeceraFilas.abajo;
        while (actual != null) {
            if (modelo != null && actual.modelo != null && actual.modelo.equals(modelo)) return
actual;
            actual = actual.abajo;
        }
        return null;
    }

    private Nodo insertarCabeceraFila(String modelo) {
        Nodo nuevaCabecera = new Nodo(modelo, true);
        Nodo actual = cabeceraFilas;
        while (actual.abajo != null) actual = actual.abajo;
        actual.abajo = nuevaCabecera;
        nuevaCabecera.arriba = actual;
        return nuevaCabecera;
    }

    private Nodo buscarCabeceraColumna(String propietario) {
        Nodo actual = cabeceraColumnas.derecha;
        while (actual != null) {
            if (propietario != null && actual.propietario != null &&
actual.propietario.equals(propietario)) return actual;
            actual = actual.derecha;
        }
        return null;
    }

    private Nodo insertarCabeceraColumna(String propietario) {
        Nodo nuevaCabecera = new Nodo(propietario, false);
        Nodo actual = cabeceraColumnas;
        while (actual.derecha != null) actual = actual.derecha;
        actual.derecha = nuevaCabecera;
        nuevaCabecera.izquierda = actual;
        return nuevaCabecera;
    }

    public boolean existePlaca(String placa) {

```

```

    Nodo filaActual = cabeceraFilas.abajo;
    while (filaActual != null) {
        Nodo nodoActual = filaActual.derecha;
        while (nodoActual != null) {

            if (nodoActual.placa != null && nodoActual.placa.equalsIgnoreCase(placa)) {
                return true;
            }
            nodoActual = nodoActual.derecha;
        }
        filaActual = filaActual.abajo;
    }
    return false;
}

public List<Nodo> buscar(String criterio) {
    List<Nodo> resultados = new ArrayList<>();
    Nodo filaActual = cabeceraFilas.abajo;
    while (filaActual != null) {
        Nodo nodoActual = filaActual.derecha;
        while (nodoActual != null) {
            if (contieneCriterio(nodoActual, criterio)) {
                resultados.add(nodoActual);
            }
            nodoActual = nodoActual.derecha;
        }
        filaActual = filaActual.abajo;
    }
    return resultados;
}

private boolean contieneCriterio(Nodo nodo, String criterio) {
    String criterioLower = (criterio != null) ? criterio.toLowerCase() : "";
    return (nodo.placa != null && nodo.placa.toLowerCase().contains(criterioLower)) ||
        (nodo.color != null && nodo.color.toLowerCase().contains(criterioLower)) ||
        (nodo.linea != null && nodo.linea.toLowerCase().contains(criterioLower)) ||
        (nodo.modelo != null && nodo.modelo.toLowerCase().contains(criterioLower)) ||
        (nodo.propietario != null && nodo.propietario.toLowerCase().contains(criterioLower));
}

public boolean eliminarPorPlaca(String placa) {
    boolean eliminado = false;
    Nodo filaActual = cabeceraFilas.abajo;
    while (filaActual != null) {
        Nodo nodoActual = filaActual.derecha;
        while (nodoActual != null) {
            Nodo siguiente = nodoActual.derecha;
            if (nodoActual.placa != null && nodoActual.placa.equalsIgnoreCase(placa)) {
                String propietario = nodoActual.propietario;
                String modelo = nodoActual.modelo;
            }
        }
    }
}

```

```

        if (nodoActual.izquierda != null) nodoActual.izquierda.derecha =
nodoActual.derecha;
        if (nodoActual.derecha != null) nodoActual.derecha.izquierda =
nodoActual.izquierda;
        if (nodoActual.arriba != null) nodoActual.arriba.abajo = nodoActual.abajo;
        if (nodoActual.abajo != null) nodoActual.abajo.arriba = nodoActual.arriba;

        nodoActual.desconectar(); // Desconectar referencias para facilitar la recoleccion

        eliminado = true;

        eliminarCabeceraColumnaSiVacia(propietario);
        eliminarCabeceraFilaSiVacia(modelo);
        break; // ya se elimino, no es necesario seguir en esta fila
    }
    nodoActual = siguiente;
}
if (eliminado) break; // Si se elimino, no es necesario seguir buscando en otras filas
filaActual = filaActual.abajo;
}
return eliminado;
}

```

```

private void eliminarCabeceraColumnaSiVacia(String propietario) {
    if (propietario == null) return;
    Nodo columnaActual = cabeceraColumnas.derecha;
    while (columnaActual != null) {
        if (propietario.equals(columnaActual.propietario)) {
            if (columnaActual.abajo == null) {
                if (columnaActual.izquierda != null) columnaActual.izquierda.derecha =
columnaActual.derecha;
                if (columnaActual.derecha != null) columnaActual.derecha.izquierda =
columnaActual.izquierda;
                columnaActual.desconectar(); // Desconectar la cabecera también
            }
            break;
        }
        columnaActual = columnaActual.derecha;
    }
}

```

```

private void eliminarCabeceraFilaSiVacia(String modelo) {
    if (modelo == null) return;
    Nodo filaActual = cabeceraFilas.abajo;
    while (filaActual != null) {
        if (modelo.equals(filaActual.modelo)) {
            if (filaActual.derecha == null) {
                if (filaActual.arriba != null) filaActual.arriba.abajo = filaActual.abajo;
                if (filaActual.abajo != null) filaActual.abajo.arriba = filaActual.arriba;
                filaActual.desconectar(); // Desconectar la cabecera también
            }
        }
    }
}

```

```

        break;
    }
    filaActual = filaActual.abajo;
}
}

private boolean validarPlacaGuatemala(String placa) {
    if (placa == null) return false;
    placa = placa.toUpperCase();

    String letrasIniciales = "ACMOPTU";
    String letrasPermitidas = "BCDFGHJKLMNPQRSTUVWXYZ";

    if (placa.length() != 7) return false;

    char primeraLetra = placa.charAt(0);
    if (letrasIniciales.indexOf(primeraLetra) == -1) return false;

    for (int i = 1; i <= 3; i++) {
        if (!Character.isDigit(placa.charAt(i))) return false;
    }

    for (int i = 4; i <= 6; i++) {
        if (letrasPermitidas.indexOf(placa.charAt(i)) == -1) return false;
    }

    return true;
}
}

```

Clase Matriz Panel

```

import javax.swing.*.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*.*;
import java.util.ArrayList;
import java.util.List;

public class MatrizPanel extends JPanel {
    private MatrizOrtogonal02 matriz;
    private JTable tabla;
    private DefaultTableModel modeloTabla;

    public MatrizPanel(MatrizOrtogonal02 matriz) {
        this.matriz = matriz;
        setLayout(new BorderLayout());

        modeloTabla = new DefaultTableModel();
        tabla = new JTable(modeloTabla);
        add(new JScrollPane(tabla), BorderLayout.CENTER);
    }
}

```

```
    actualizarTablaCompleta();  
}
```

```
public void actualizarTablaCompleta() {  
    modeloTabla.setRowCount(0);  
    modeloTabla.setColumnCount(0);  
  
    String[] columnas = {"Placa", "Color", "Línea", "Modelo", "Propietario"};  
    for (String col : columnas) {  
        modeloTabla.addColumn(col);  
    }  
  
    List<Nodo> listaVehiculos = obtenerTodosLosVehiculos();  
    for (Nodo nodo : listaVehiculos) {  
        Object[] fila = {  
            nodo.placa,  
            nodo.color,  
            nodo.linea,  
            nodo.modelo,  
            nodo.propietario  
        };  
        modeloTabla.addRow(fila);  
    }  
}
```

```
// CORREGIDO: Nombre del método para búsqueda  
public void actualizarTablaBusqueda(List<Nodo> resultados) {  
    modeloTabla.setRowCount(0);  
    modeloTabla.setColumnCount(0);  
  
    String[] columnas = {"Placa", "Color", "Línea", "Modelo", "Propietario"};  
    for (String col : columnas) {  
        modeloTabla.addColumn(col);  
    }  
  
    for (Nodo nodo : resultados) {  
        Object[] fila = {  
            nodo.placa,  
            nodo.color,  
            nodo.linea,  
            nodo.modelo,  
            nodo.propietario  
        };  
        modeloTabla.addRow(fila);  
    }  
}
```

```

private List<Nodo> obtenerTodosLosVehiculos() {
    List<Nodo> lista = new ArrayList<>();
    Nodo filaActual = matriz.getCabeceraFilas().abajo;
    while (filaActual != null) {
        Nodo nodoActual = filaActual.derecha;
        while (nodoActual != null) {
            lista.add(nodoActual);
            nodoActual = nodoActual.derecha;
        }

        filaActual = filaActual.abajo;
    }
    return lista;
}
}

```

Clase Nodo

```

public class Nodo {
    public String placa;
    public String color;
    public String linea;
    public String modelo;
    public String propietario;

    public Nodo derecha, izquierda, arriba, abajo;

    // Constructor para nodo vehículo
    public Nodo(String placa, String color, String linea, String modelo, String propietario) {
        this.placa = placa;
        this.color = color;
        this.linea = linea;
        this.modelo = modelo;
        this.propietario = propietario;
    }

    // Constructor para cabeceras (modelo o propietario)
    public Nodo(String valor, boolean esModelo) {
        if (esModelo) this.modelo = valor;
        else this.propietario = valor;
    }

    // Método para desconectar las referencias del nodo
    public void desconectar() {
        this.derecha = null;
        this.izquierda = null;
        this.arriba = null;
        this.abajo = null;
    }
}

```



```
}  
}
```

Clase Ventana Principal

```
import javax.swing.*;  
import java.awt.*;  
import java.util.List;  
  
public class VentanaPrincipal extends JFrame {  
    private MatrizOrtogonal02 matriz;  
    private MatrizPanel panelMatriz;  
  
    private JTextField txtPlaca, txtColor, txtLinea, txtModelo, txtPropietario;  
    private JTextField txtBuscar;  
  
    public VentanaPrincipal() {  
        matriz = new MatrizOrtogonal02();  
        panelMatriz = new MatrizPanel(matriz);  
  
        setTitle("Gestión de Vehículos - Matriz Ortogonal");  
        setSize(1000, 600);  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setLayout(new BorderLayout());  
  
        add(panelMatriz, BorderLayout.CENTER);  
        add(crearPanelControles(), BorderLayout.NORTH);  
        add(crearPanelBusqueda(), BorderLayout.SOUTH);  
    }  
  
    private JPanel crearPanelControles() {  
        JPanel panel = new JPanel();  
  
        txtPlaca = new JTextField(7);  
        txtColor = new JTextField(7);  
        txtLinea = new JTextField(7);  
        txtModelo = new JTextField(7);  
        txtPropietario = new JTextField(7);  
        JButton btnInsertar = new JButton("Insertar");  
        JButton btnEliminar = new JButton("Eliminar por Placa");  
  
        panel.add(new JLabel("Placa:"));  
        panel.add(txtPlaca);  
        panel.add(new JLabel("Color:"));  
        panel.add(txtColor);  
        panel.add(new JLabel("Línea:"));  
        panel.add(txtLinea);  
        panel.add(new JLabel("Modelo:"));  
        panel.add(txtModelo);  
        panel.add(new JLabel("Propietario:"));
```

```

        panel.add(txtPropietario);
        panel.add(btnInsertar);
        panel.add(btnEliminar);

        btnInsertar.addActionListener(e -> insertarVehiculo());
        btnEliminar.addActionListener(e -> eliminarVehiculo());

        return panel;
    }

    private JPanel crearPanelBusqueda() {
        JPanel panel = new JPanel();

        txtBuscar = new JTextField(20);
        JButton btnBuscar = new JButton("Buscar");
        JButton btnMostrarTodo = new JButton("Mostrar Todo");

        panel.add(new JLabel("Buscar:"));
        panel.add(txtBuscar);
        panel.add(btnBuscar);
        panel.add(btnMostrarTodo);

        btnBuscar.addActionListener(e -> buscarVehiculos());
        btnMostrarTodo.addActionListener(e -> {
            panelMatriz.actualizarTablaCompleta();
            txtBuscar.setText("");
        });

        return panel;
    }

    private void insertarVehiculo() {
        String placa = txtPlaca.getText().trim().toUpperCase();
        String color = txtColor.getText().trim();
        String linea = txtLinea.getText().trim();
        String modelo = txtModelo.getText().trim();
        String propietario = txtPropietario.getText().trim();

        if (placa.isEmpty() || color.isEmpty() || linea.isEmpty() || modelo.isEmpty() ||
            propietario.isEmpty()) {
            JOptionPane.showMessageDialog(this, "Completa todos los campos.");
            return;
        }

        // Validar que color solo tenga letras
        if (!validarColor(color)) {
            JOptionPane.showMessageDialog(this, "El color solo debe contener letras.");
            return;
        }
    }

```

```

// Validar que modelo tenga exactamente 4 dígitos numéricos
if (!validarModelo(modelo)) {
    JOptionPane.showMessageDialog(this, "El modelo debe ser un número de 4 dígitos.");

    return;
}

if (!matriz.insertar(placa, color, linea, modelo, propietario)) {
    JOptionPane.showMessageDialog(this, "No se pudo insertar el vehículo. Verifica la
placa.");
} else {
    panelMatriz.actualizarTablaCompleta();
    limpiarCampos();
}
}

private void eliminarVehiculo() {
    String placa = txtPlaca.getText().trim().toUpperCase();
    if (placa.isEmpty()) {
        JOptionPane.showMessageDialog(this, "Ingresa la placa del vehículo a eliminar.");
        return;
    }

    if (matriz.eliminarPorPlaca(placa)) {
        JOptionPane.showMessageDialog(this, "Vehículo con placa " + placa + " eliminado.");
        panelMatriz.actualizarTablaCompleta();
        limpiarCampos();
    } else {
        JOptionPane.showMessageDialog(this, "No se encontró ningún vehículo con la placa "
+ placa + ".");
    }
}

private void buscarVehiculos() {
    String criterio = txtBuscar.getText().trim();
    if (!criterio.isEmpty()) {
        List<Nodo> resultados = matriz.buscar(criterio);
        panelMatriz.actualizarTablaBusqueda(resultados);
    } else {
        JOptionPane.showMessageDialog(this, "Ingresa un criterio de búsqueda.");
    }
}

private void limpiarCampos() {
    txtPlaca.setText("");
    txtColor.setText("");
    txtLinea.setText("");
    txtModelo.setText("");
    txtPropietario.setText("");
}

```

```

private boolean validarColor(String color) {
    // Solo letras (mayúsculas, minúsculas, espacios y caracteres comunes en español)
    return color.matches("[a-zA-ZáéíóúÁÉÍÓÚüÜñÑ ]+");
}

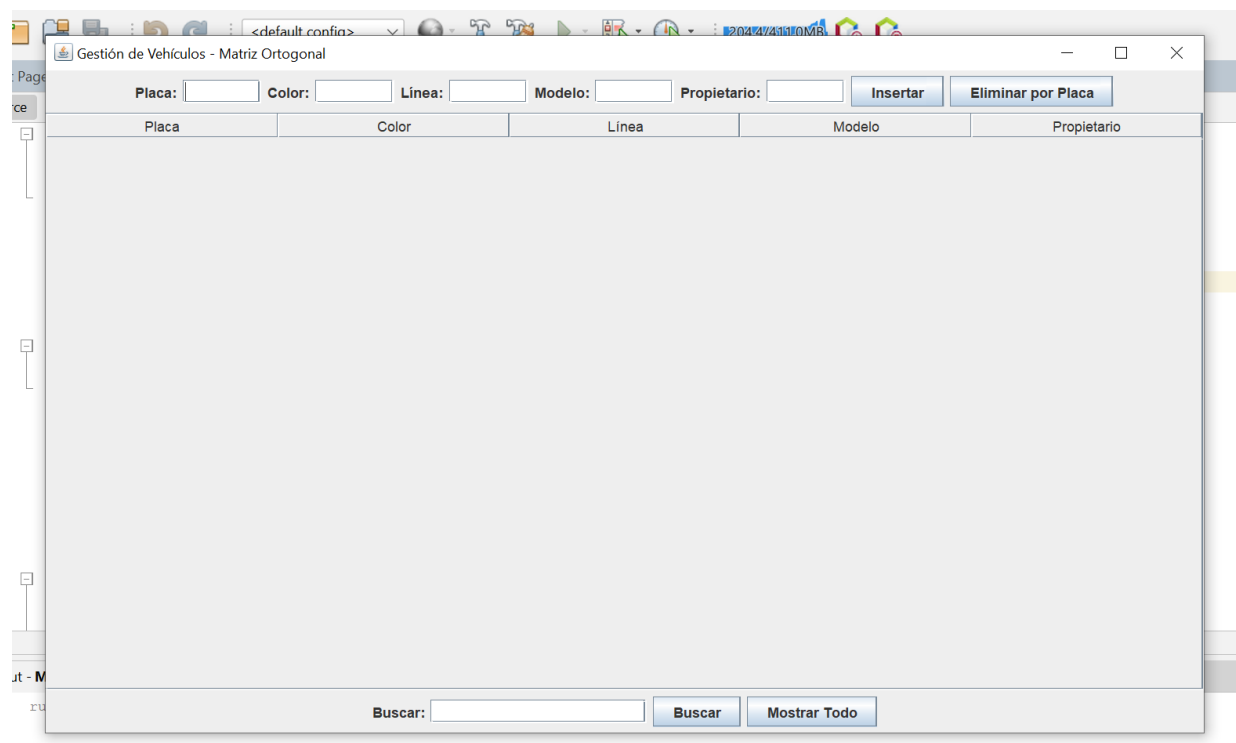
private boolean validarModelo(String modelo) {
    // Solo números, exactamente 4 dígitos
    return modelo.matches("\\d{4}");
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> {
        VentanaPrincipal ventana = new VentanaPrincipal();
        ventana.setVisible(true);
    });
}
}

```

Capturas de Código

Se ingresa al Proyecto Control de Parqueo y visualizamos Nuestro panel o pantalla principal



Realizamos el ingreso de un vehiculo

Colocamos los datos y precionamos el boton insertar

Gestión de Vehículos - Matriz Ortogonal

Placa: P456GHF Color: BLANCO Linea: 56 Modelo: 2024 Propietario: CARLOS ARDON

Insertar Eliminar por Placa

Placa	Color	Linea	Modelo	Propietario
-------	-------	-------	--------	-------------

Buscar: [] Buscar Mostrar Todo

Gestión de Vehículos - Matriz Ortogonal

Placa: [] Color: [] Linea: [] Modelo: [] Propietario: []

Insertar Eliminar por Placa

Placa	Color	Linea	Modelo	Propietario
P456GHF	BLANCO	56	2024	CARLOS ARDON

Buscar: [] Buscar Mostrar Todo

Temenos el primer vehiculo ya en nustro panel, realizamos el mismo procedimiento con 3 mas

Gestión de Vehículos - Matriz Ortogonal

Placa:

Color:

Línea:

Modelo:

Propietario:

Insertar

Eliminar por Placa

Placa	Color	Línea	Modelo	Propietario
P456GHF	BLANCO	56	2024	CARLOS ARDON
P786JGF	AZUL	45	2024	Christian Emanuel García
M567JJJ	NEGRO	F	2025	Melannie Rosse Lorenzana
O456JKL	gris	46	2023	Carlos Guillermo Ardón

Buscar:

Buscar

Mostrar Todo

Procedemos a eliminar un vehiculo

Gestión de Vehículos - Matriz Ortogonal

Placa:

Color:

Línea:

Modelo:

Propietario:

Insertar

Eliminar por Placa

Placa	Color	Línea	Modelo	Propietario
P456GHF	BLANCO	56	2024	CARLOS ARDON
P786JGF	AZUL	45	2024	Christian Emanuel García
M567JJJ	NEGRO	F	2025	Melannie Rosse Lorenzana
O456JKL	gris	46	2023	Carlos Guillermo Ardón

Message

Vehículo con placa P786JGF eliminado.

OK

Buscar:

Buscar

Mostrar Todo

Gestión de Vehículos - Matriz Ortogonal

Placa:

Color:

Línea:

Modelo:

Propietario:

Insertar

Eliminar por Placa

Placa	Color	Línea	Modelo	Propietario
P456GHF	BLANCO	56	2024	CARLOS ARDON
V567JJJ	NEGRO	F	2025	Melannie Rosse Lorenzana
O456JKL	gris	46	2023	Carlos Guillermo Ardón

Buscar:

Buscar

Mostrar Todo

Realizamos la búsqueda de un vehiculo

Gestión de Vehículos - Matriz Ortogonal

Placa:

Color:

Línea:

Modelo:

Propietario:

Insertar


Eliminar por Placa

Placa	Color	Línea	Modelo	Propietario
P456GHF	BLANCO	56	2024	CARLOS ARDON
M567JJJ	NEGRO	F	2025	Melannie Rosse Lorenzana
O456JKL	gris	46	2023	Carlos Guillermo Ardón

Buscar: negro

Buscar

Mostrar Todo

 Gestión de Vehículos - Matriz Ortogonal

Placa:

Color:

Línea:

Modelo:

Propietario:

Insertar


Eliminar por Placa

Placa	Color	Línea	Modelo	Propietario
M567JJJ	NEGRO	F	2025	Melannie Rosse Lorenzana

Buscar:

Buscar

Mostrar Todo

 Gestión de Vehículos - Matriz Ortogonal

Placa:

Color:

Línea:

Modelo:

Propietario:

Insertar

Eliminar por Placa

Placa	Color	Línea	Modelo	Propietario
P456GHF	BLANCO	56	2024	CARLOS ARDON
M567JJJ	NEGRO	F	2025	Melannie Rosse Lorenzana
O456JKL	gris	46	2023	Carlos Guillermo Ardón

Buscar:

Buscar

Mostrar Todo

Manual Técnico - Documentación Interna

Este manual describe la estructura interna y el funcionamiento del sistema de gestión de vehículos implementado mediante una matriz ortogonal. Está destinado a desarrolladores y personal técnico encargado del mantenimiento y la expansión del sistema.

1. Estructura de la Matriz Ortogonal

La implementación utiliza una estructura de datos de matriz ortogonal para almacenar la información de los vehículos. Esta estructura se basa en nodos interconectados que representan tanto los datos de los vehículos como las cabeceras de filas (modelos) y columnas (propietarios).

1.1. Clase MatrizOrtogonal02

Esta clase contiene la lógica principal para la gestión de la matriz ortogonal.

* *Atributos:*

- * **cabeceraFilas**: Un nodo especial que sirve como punto de entrada para las cabeceras de las filas (modelos de vehículos). Es un nodo de cabecera (**esCabecera = true**).

- * **cabeceraColumnas**: Un nodo especial que sirve como punto de entrada para las cabeceras de las columnas (propietarios de vehículos). Es un nodo de cabecera (**esCabecera = false**).

* *Constructor MatrizOrtogonal02():*

- * Inicializa **cabeceraFilas** y **cabeceraColumnas** como nodos de cabecera con nombres descriptivos.

* *Métodos:*

- * **getCabeceraFilas()**: Retorna la cabecera de las filas.

- * **getCabeceraColumnas()**: Retorna la cabecera de las columnas.

- * **insertar(String placa, String color, String linea, String modelo, String propietario)**:

- * Valida el formato de la placa utilizando **validarPlacaGuatemala()**.

- * Verifica si la placa ya existe utilizando **existePlaca()**.

- * Busca o inserta la cabecera de fila correspondiente al modelo utilizando **buscarCabeceraFila()** e **insertarCabeceraFila()**.

- * Busca o inserta la cabecera de columna correspondiente al propietario utilizando **buscarCabeceraColumna()** e **insertarCabeceraColumna()**.

- * Crea un nuevo Nodo con la información del vehículo.

- * Inserta el nuevo nodo en la fila (enlazándolo a la derecha del último nodo del modelo) y en la columna (enlazándolo abajo del último nodo del propietario).

- * Retorna true si la inserción fue exitosa, false en caso contrario.

- * **buscarCabeceraFila(String modelo)**: Recorre las cabeceras de fila y retorna el nodo de cabecera correspondiente al modelo, o null si no existe.

- * **insertarCabeceraFila(String modelo)**: Crea un nuevo nodo de cabecera para el modelo y lo inserta al final de la lista de cabeceras de fila.

- * **buscarCabeceraColumna(String propietario)**: Recorre las cabeceras de columna y retorna el nodo de cabecera correspondiente al propietario, o null si no existe.

- * **insertarCabeceraColumna(String propietario)**: Crea un nuevo nodo de cabecera para el propietario y lo inserta al final de la lista de cabeceras de columna.

- * **existePlaca(String placa)**: Recorre toda la matriz buscando un nodo con la placa

especificada (ignorando mayúsculas/minúsculas). Retorna true si la encuentra, false en caso contrario.

- * buscar(String criterio): Recorre toda la matriz y retorna una List<Nodo> con todos los nodos cuyo placa, color, linea, modelo o propietario contengan el criterio de búsqueda (ignorando mayúsculas/minúsculas). Utiliza el método auxiliar contieneCriterio().

- * contieneCriterio(Nodo nodo, String criterio): Método auxiliar que verifica si algún atributo del nodo contiene el criterio de búsqueda (ignorando mayúsculas/minúsculas).

- * eliminarPorPlaca(String placa): Recorre la matriz buscando el nodo con la placa especificada (ignorando mayúsculas/minúsculas). Si lo encuentra, lo desvincula de sus nodos vecinos en la fila y la columna, y luego llama a eliminarCabeceraColumnaSiVacia() y eliminarCabeceraFilaSiVacia() para eliminar las cabeceras si ya no tienen nodos asociados. Retorna true si se eliminó el nodo, false en caso contrario.

- * eliminarCabeceraColumnaSiVacia(String propietario): Busca la cabecera de columna del propietario y verifica si no tiene nodos de vehículo asociados (columnaActual.abajo == null). Si está vacía, la desvincula de las otras cabeceras de columna.

- * eliminarCabeceraFilaSiVacia(String modelo): Busca la cabecera de fila del modelo y verifica si no tiene nodos de vehículo asociados (filaActual.derecha == null). Si está vacía, la desvincula de las otras cabeceras de fila.

- * validarPlacaGuatemala(String placa): Valida si la placa cumple con el formato específico de Guatemala (una letra inicial en "ACMOPTU", seguida de tres dígitos y luego tres letras permitidas).

1.2. Clase Nodo

Esta clase representa cada nodo dentro de la matriz ortogonal.

* *Atributos:*

- * placa: Placa del vehículo (String).

- * color: Color del vehículo (String).

- * linea: Línea del vehículo (String).

- * modelo: Modelo del vehículo (String). También se utiliza para las cabeceras de fila.

- * propietario: Propietario del vehículo (String). También se utiliza para las cabeceras de columna.

- * derecha: Referencia al siguiente nodo en la misma fila (modelo).

- * izquierda: Referencia al nodo anterior en la misma fila (modelo).

- * arriba: Referencia al nodo superior en la misma columna (propietario).

- * abajo: Referencia al nodo inferior en la misma columna (propietario).

* *Constructores:*

- * Nodo(String placa, String color, String linea, String modelo, String propietario): Constructor para los nodos que almacenan la información de los vehículos.

- * Nodo(String valor, boolean esModelo): Constructor para los nodos de cabecera. Si esModelo es true, el valor se asigna al atributo modelo; de lo contrario, se asigna a propietario.

* *Método desconectar():*

- * Establece todas las referencias (derecha, izquierda, arriba, abajo) a null para facilitar la recolección de basura.

2. Interfaz Gráfica de Usuario (GUI)

La interfaz gráfica se implementa utilizando Swing y consta de las siguientes clases:

2.1. Clase MatrizPanel

Este panel muestra la información de los vehículos en una tabla.

* *Atributos:*

- * matriz: Una instancia de la clase MatrizOrtogonal02 para acceder a los datos.
- * tabla: Un JTable para mostrar los datos en formato tabular.
- * modeloTabla: Un DefaultTableModel que gestiona los datos de la tabla.

* *Constructor MatrizPanel(MatrizOrtogonal02 matriz):*

- * Recibe una instancia de MatrizOrtogonal02.
- * Configura el layout del panel como BorderLayout.
- * Crea un DefaultTableModel y un JTable asociado.
- * Agrega la tabla a un JScrollPane y lo añade al centro del panel.
- * Llama a actualizarTablaCompleta() para mostrar todos los vehículos al inicio.

* *Métodos:*

- * actualizarTablaCompleta(): Limpia la tabla, define las columnas ("Placa", "Color", "Línea", "Modelo", "Propietario") y agrega todas las filas de vehículos obtenidas mediante obtenerTodosLosVehiculos().
- * actualizarTablaBusqueda(List<Nodo> resultados): Limpia la tabla, define las columnas y agrega solo las filas de los nodos proporcionados en la lista resultados.
- * obtenerTodosLosVehiculos(): Recorre la matriz ortogonal y retorna una List<Nodo> con todos los nodos que representan vehículos.

2.2. Clase VentanaPrincipal

Esta clase crea la ventana principal de la aplicación y contiene los controles para interactuar con la matriz.

* *Atributos:*

- * matriz: Una instancia de MatrizOrtogonal02.
- * panelMatriz: Una instancia de MatrizPanel para mostrar la tabla de vehículos.
- * txtPlaca, txtColor, txtLinea, txtModelo, txtPropietario: Campos de texto para ingresar la información de un nuevo vehículo.
- * txtBuscar: Campo de texto para ingresar el criterio de búsqueda.

* *Constructor VentanaPrincipal():*

- * Crea instancias de MatrizOrtogonal02 y MatrizPanel.
- * Configura el título, tamaño y operación de cierre de la ventana.
- * Establece el layout como BorderLayout.
- * Agrega el panelMatriz al centro.
- * Llama a crearPanelControles() y crearPanelBusqueda() para obtener los paneles de control y búsqueda, y los agrega a la parte superior e inferior de la ventana, respectivamente.

* *Métodos:*

- * crearPanelControles(): Crea un JPanel con etiquetas y campos de texto para la placa, color, línea, modelo y propietario, así como botones para "Insertar" y "Eliminar por Placa". Asigna los ActionListener a los botones para llamar a los métodos insertarVehiculo() y eliminarVehiculo().

* crearPanelBusqueda(): Crea un JPanel con una etiqueta, un campo de texto (txtBuscar) y botones para "Buscar" y "Mostrar Todo". Asigna los ActionListener a los botones para llamar a los métodos buscarVehiculos() y actualizarTablaCompleta() del panelMatriz.

* insertarVehiculo(): Obtiene los valores de los campos de texto, realiza validaciones básicas (campos no vacíos, formato de color y modelo), y llama al método insertar() de la matriz. Muestra mensajes de éxito o error mediante JOptionPane y actualiza la tabla llamando a actualizarTablaCompleta() del panelMatriz.

* eliminarVehiculo(): Obtiene la placa del campo de texto, verifica que no esté vacío y llama al método eliminarPorPlaca() de la matriz. Muestra mensajes de éxito o error y actualiza la tabla.

* buscarVehiculos(): Obtiene el criterio de búsqueda del campo de texto y llama al método buscar() de la matriz. Actualiza la tabla de resultados llamando a actualizarTablaBusqueda() del panelMatriz.

* limpiarCampos(): Limpia los campos de texto de la información del vehículo.

* validarColor(String color): Valida si la cadena color contiene solo letras, espacios y caracteres comunes en español.

* validarModelo(String modelo): Valida si la cadena modelo contiene exactamente 4 dígitos numéricos.

* main(String[] args): Método principal que crea e inicia la VentanaPrincipal en el hilo de eventos de Swing.

3. Consideraciones de Diseño

* *Matriz Ortogonal:* Se eligió una matriz ortogonal para optimizar las búsquedas por modelo y propietario, ya que los nodos están indexados por ambos criterios.

* *Validación de Placas:* Se implementó una validación específica para el formato de placas de Guatemala.

* *Interfaz Swing:* Se utilizó Swing para crear una interfaz gráfica sencilla y funcional para la interacción del usuario.

* *Manejo de Errores:* Se incluyen mensajes de error básicos para informar al usuario sobre problemas durante la inserción o eliminación.

4. Posibles Mejoras

* Implementar funcionalidades de edición de vehículos existentes.

* Añadir persistencia de datos (guardar y cargar la matriz desde un archivo).

* Mejorar la interfaz de búsqueda con opciones de filtrado por campos específicos.

* Implementar pruebas unitarias para asegurar la robustez del código.

* Considerar el uso de un patrón de diseño como MVC (Modelo-Vista-Controlador) para una mejor separación de responsabilidades.

Manual de Usuario - Gestión de Vehículos

Este manual describe cómo utilizar la aplicación de gestión de vehículos. Está destinado a los usuarios finales que interactuarán con el sistema para registrar, buscar y eliminar información de vehículos.

1. Introducción

La aplicación de Gestión de Vehículos permite mantener un registro organizado de la información de los vehículos, incluyendo su placa, color, línea, modelo y propietario. La información se almacena de manera eficiente, permitiendo búsquedas rápidas y sencillas.

2. Interfaz Principal

Al iniciar la aplicación, se mostrará una ventana con las siguientes secciones:

- * **Panel de Controles (Parte Superior de la ventana):** Contiene campos para ingresar la información de un nuevo vehículo (Placa, Color, Línea, Modelo, Propietario) y botones para realizar las acciones de "Insertar" y "Eliminar por Placa".
- * **Tabla de Vehículos (Centro):** Muestra la lista de todos los vehículos registrados en el sistema en formato de tabla con las columnas: Placa, Color, Línea, Modelo y Propietario.
- * **Panel de Búsqueda (Parte Inferior):** Contiene un campo de texto para ingresar un criterio de búsqueda y botones para "Buscar" y "Mostrar Todo".

3. Funcionalidades Principales

3.1. Insertar un Nuevo Vehículo

Para registrar un nuevo vehículo en el sistema, siga estos pasos:

1. En el ***Panel de Controles***, complete los siguientes campos con la información del vehículo:
 - * **Placa:** Ingrese la placa del vehículo. Asegúrese de seguir el formato de placa de Guatemala (ejemplo: P123BCD).
 - * **Color:** Ingrese el color del vehículo (solo letras).
 - * **Línea:** Ingrese la línea o marca del vehículo (ejemplo: Toyota).
 - * **Modelo:** Ingrese el año del modelo del vehículo (debe ser un número de 4 dígitos, ejemplo: 2023).
 - * **Propietario:** Ingrese el nombre del propietario del vehículo.
2. Haga clic en el botón **"Insertar"**.
3. Si la información es válida y la placa no existe, el vehículo se agregará al registro y se mostrará en la ***Tabla de Vehículos***. Si hay algún error (placa inválida o ya existente, formato incorrecto), se mostrará un mensaje informativo.

3.2. Eliminar un Vehículo por Placa

Para eliminar un vehículo del registro, siga estos pasos:

1. En el ***Panel de Controles***, ingrese la placa del vehículo que desea eliminar en el campo **"Placa"**.
2. Haga clic en el botón **"Eliminar por Placa"**.

3. Si se encuentra un vehículo con la placa especificada, se mostrará un mensaje de confirmación y el vehículo se eliminará de la *Tabla de Vehículos*. Si no se encuentra ningún vehículo con esa placa, se mostrará un mensaje indicándolo.

3.3. Buscar Vehículos

Para buscar vehículos en el registro, puede utilizar un criterio de búsqueda:

1. En el *Panel de Búsqueda*, ingrese el texto que desea buscar en el campo *"Buscar"*. Puede ingresar parte de la placa, el color, la línea, el modelo o el nombre del propietario.
2. Haga clic en el botón *"Buscar"*.
3. La *Tabla de Vehículos* se actualizará para mostrar solo los vehículos que coincidan con su criterio de búsqueda.
4. Para volver a ver todos los vehículos registrados, haga clic en el botón *"Mostrar Todo"*.

4. Consideraciones Importantes

* *Formato de Placa:* Al ingresar la placa, asegúrese de seguir el formato estándar de placas de Guatemala. La aplicación validará este formato.

* *Formato de Color:* El campo de color solo debe contener letras.

* *Formato de Modelo:* El campo de modelo debe ser un número de exactamente 4 dígitos.

* *Búsqueda:* La búsqueda es flexible y buscará el texto ingresado en todos los campos de información del vehículo.