UNIVERSIDAD MARIANO GÁLVEZ

SEDE: MAZATENANGO

CARRERA: INGENIERIA EN SISTEMAS

CURSO: PROGRAMACION III

CICLO: V

SEMESTRE: 1to.

MANUAL TECNICO

NOMBRE DEL ESTUDIANTE: ERALDO JOSUE ROJAS AGUILAR

CARNÉ: 3090-23-22967

FECHA DE ENTREGA: 15/6/2025

```
// Clase Vehiculo: define propiedades basicas de un vehiculo y metodos de acceso
public class Vehiculo {
    private String placa;    // Matricula o identificador unico del vehiculo
    private String marca;    // Marca comercial (p.ej. Toyota, Ford)
    private String modelo;   // Modelo específico del vehiculo
    private int anio;        // Año de fabricacion

    /**
     * Constructor Vehiculo
     * @param placa Matricula del vehiculo
     * @param marca Marca del vehiculo
     * @param modelo Modelo del vehiculo
     * @param anio Año de fabricacion
     */
    public Vehiculo(String placa, String marca, String modelo, int anio) {
        this.placa = placa;    // Asigna placa
        this.marca = marca;    // Asigna marca
        this.modelo = modelo;  // Asigna modelo
        this.anio = anio;      // Asigna año
    }

    // Getters y setters para cada propiedad
    public String getPlaca() { return placa; }
    public void setPlaca(String placa) { this.placa = placa; }
    public String getMarca() { return marca; }
    public void setMarca(String marca) { this.marca = marca; }
    public String getModelo() { return modelo; }
    public void setModelo(String modelo) { this.modelo = modelo; }
    public int getAnio() { return anio; }
```

```
    public void setAnio(int anio) { this.anio = anio; }

    /**
     * toString - representa el vehiculo como cadena legible
     */
    @Override
    public String toString() {
        return placa + " - " + marca + " " + modelo + " (" + anio + ")";
    }
}

// NodoVehiculo: elemento de un arbol binario que guarda un Vehiculo
public class NodoVehiculo {
    Vehiculo vehiculo;      // Dato principal: objeto Vehiculo
    NodoVehiculo izquierdo;  // Referencia a subarbol izquierdo
    NodoVehiculo derecho;    // Referencia a subarbol derecho

    /**
     * Constructor NodoVehiculo
     * @param vehiculo Objeto Vehiculo almacenado en el nodo
     */
    public NodoVehiculo(Vehiculo vehiculo) {
        this.vehiculo = vehiculo;  // Asigna dato
        this.izquierdo = null;     // Inicializa hijos en null
        this.derecho = null;
    }
}

// ArbolVehiculos: BST para ordenar y buscar Vehiculos por placa
```

```
public class ArbolVehiculos {
    private NodoVehiculo raiz;  // Nodo raiz del arbol

    public ArbolVehiculos() {
        this.raiz = null;       // Arbol inicialmente vacio
    }

    /**
     * insertar - agrega un nuevo vehiculo al arbol
     * @param v Vehiculo a insertar
     */
    public void insertar(Vehiculo v) {
        raiz = insertarRec(raiz, v);  // Uso de metodo recursivo
    }

    /**
     * insertarRec - recorre el arbol para colocar el vehiculo
     * compara cadenas de placas para decidir izquierda/derecha
     */
    private NodoVehiculo insertarRec(NodoVehiculo nodo, Vehiculo v) {
        if (nodo == null) {
            return new NodoVehiculo(v);  // Caso base: crear nodo nuevo
        }
        // Si v.placa < nodo.vehiculo.placa, va a subarbol izquierdo
        if (v.getPlaca().compareTo(nodo.vehiculo.getPlaca()) < 0) {
            nodo.izquierdo = insertarRec(nodo.izquierdo, v);
        }
        // Si v.placa > nodo.vehiculo.placa, va a subarbol derecho
        else if (v.getPlaca().compareTo(nodo.vehiculo.getPlaca()) > 0) {
```

```
        nodo.derecho = insertarRec(nodo.derecho, v);
    }
    // Si placas iguales, no inserta duplicado
    return nodo;
}

/**
 * buscar - encuentra un vehiculo por placa
 * @param placa Placa buscada
 * @return Vehiculo encontrado o null
 */
public Vehiculo buscar(String placa) {
    return buscarRec(raiz, placa);
}

private Vehiculo buscarRec(NodoVehiculo nodo, String placa) {
    if (nodo == null) return null;              // No encontrado
    if (placa.equals(nodo.vehiculo.getPlaca())) {
        return nodo.vehiculo;                   // Caso encontrado
    }
    // Decide subarbol segun comparacion
    if (placa.compareTo(nodo.vehiculo.getPlaca()) < 0) {
        return buscarRec(nodo.izquierdo, placa);
    } else {
        return buscarRec(nodo.derecho, placa);
    }
}

/**
```

```
     * recorridoInOrden - imprime todos los vehiculos ordenados por placa
     */
    public void recorridoInOrden() {
        recorridoInOrdenRec(raiz);
    }

    private void recorridoInOrdenRec(NodoVehiculo nodo) {
        if (nodo != null) {
            recorridoInOrdenRec(nodo.izquierdo);
            System.out.println(nodo.vehiculo);
            recorridoInOrdenRec(nodo.derecho);
        }
    }
}

// CargadorDatos: lee archivos .txt y carga Vehiculos, Multas y Traspasos
public class CargadorDatos {
    private ArbolVehiculos arbolVehiculos = new ArbolVehiculos();  // Estructura para vehiculos
    // Podrian agregarse arboles de Multas y Traspasos si se requieren

    /**
     * cargarDesdeCarpeta - inicia proceso de lectura recursiva
     * @param carpeta Directorio raiz con archivos .txt
     */
    public void cargarDesdeCarpeta(File carpeta) {
        if (carpeta.exists() && carpeta.isDirectory()) {
            cargarRecursivo(carpeta);
        } else {
            System.out.println("Ruta no valida: " + carpeta);
```

```
    }
}

/**
 * cargarRecursivo - recorre subdirectorios y archivos
 */
private void cargarRecursivo(File carpeta) {
    for (File archivo : carpeta.listFiles()) {
        if (archivo.isDirectory()) {
            cargarRecursivo(archivo);
        } else if (archivo.getName().toLowerCase().endsWith(".txt")) {
            String nombre = archivo.getName().toLowerCase();
            // Identifica tipo por nombre de archivo
            if (nombre.contains("vehiculos")) {
                cargarVehiculosDesdeArchivo(archivo);
            } else if (nombre.contains("multas")) {
                cargarMultasDesdeArchivo(archivo);
            } else if (nombre.contains("traspasos")) {
                cargarTraspasosDesdeArchivo(archivo);
            }
        }
    }
}

/**
 * cargarVehiculosDesdeArchivo - lee filas de vehiculos y los inserta
 */
private void cargarVehiculosDesdeArchivo(File archivo) {
    // Ejemplo de implementacion: BufferedReader, parseo de CSV y arbolVehiculos.insertar()
```

```
    }

    /**
     * cargarMultasDesdeArchivo - similar a vehiculos, crea objetos Multa
     */
    private void cargarMultasDesdeArchivo(File archivo) {
        // Aqui se leerian lineas, se parsea placa,monto,fecha y se insertaria en ArbolMultas
    }

    /**
     * cargarTraspasosDesdeArchivo - lee datos de traspasos y usa ArbolTraspasos
     */
    private void cargarTraspasosDesdeArchivo(File archivo) {
        // Implementar parseo de CSV: placa,origen,destino,fecha
    }
}

// Multa: contiene datos de una sancion sobre un vehiculo
public class Multa {
    private String placa;   // Vehiculo sancionado
    private double monto;   // Valor de la multa
    private String fecha;   // Fecha en formato YYYY-MM-DD

    public Multa(String placa, double monto, String fecha) {
        this.placa = placa;
        this.monto = monto;
        this.fecha = fecha;
    }
    // Getters y setters
```

```
    public String getPlaca() { return placa; }
    public void setPlaca(String placa) { this.placa = placa; }
    public double getMonto() { return monto; }
    public void setMonto(double monto) { this.monto = monto; }
    public String getFecha() { return fecha; }
    public void setFecha(String fecha) { this.fecha = fecha; }

    @Override
    public String toString() {
        return placa + "-> multa:" + monto + " fecha:" + fecha;
    }
}

// NodoMulta: nodo de arbol que almacena una Multa
public class NodoMulta {
    Multa multa;
    NodoMulta izquierdo;
    NodoMulta derecho;

    public NodoMulta(Multa multa) {
        this.multa = multa;
        this.izquierdo = null;
        this.derecho = null;
    }
}

// ArbolMultas: BST para Multas, ordena por placa del vehiculo
public class ArbolMultas {
    private NodoMulta raiz;  // Punto de inicio
```

```
public ArbolMultas() { raiz = null; }

/**
 * insertar - agrega una multa manteniendo orden
 */
public void insertar(Multa m) { raiz = insertarRec(raiz, m); }

private NodoMulta insertarRec(NodoMulta nodo, Multa m) {
   if (nodo == null) return new NodoMulta(m);
   if (m.getPlaca().compareTo(nodo.multa.getPlaca()) < 0) {
      nodo.izquierdo = insertarRec(nodo.izquierdo, m);
   } else if (m.getPlaca().compareTo(nodo.multa.getPlaca()) > 0) {
      nodo.derecho = insertarRec(nodo.derecho, m);
   }
   return nodo;
}

/**
 * buscar - devuelve la multa asociada a una placa
 */
public Multa buscar(String placa) { return buscarRec(raiz, placa); }

private Multa buscarRec(NodoMulta nodo, String placa) {
   if (nodo == null) return null;
   if (placa.equals(nodo.multa.getPlaca())) return nodo.multa;
   if (placa.compareTo(nodo.multa.getPlaca()) < 0) return buscarRec(nodo.izquierdo, placa);
   return buscarRec(nodo.derecho, placa);
}
```

```
}

// Traspaso: datos de transferencias de vehiculos entre propietarios
public class Traspaso {
    private String placa;
    private String propietarioOrigen;
    private String propietarioDestino;
    private String fecha;

    /** Constructor Traspaso */
    public Traspaso(String placa, String origen, String destino, String fecha) {
        this.placa = placa;
        this.propietarioOrigen = origen;
        this.propietarioDestino = destino;
        this.fecha = fecha;
    }
    // Getters y setters...
    public String getPlaca() { return placa; }
    public void setPlaca(String p) { this.placa = p; }
    public String getPropietarioOrigen() { return propietarioOrigen; }
    public void setPropietarioOrigen(String o) { this.propietarioOrigen = o; }
    public String getPropietarioDestino() { return propietarioDestino; }
    public void setPropietarioDestino(String d) { this.propietarioDestino = d; }
    public String getFecha() { return fecha; }
    public void setFecha(String f) { this.fecha = f; }

    @Override
    public String toString() {
        return placa + " " + propietarioOrigen + "->" + propietarioDestino + " (" + fecha + ")";
```

```
    }
}

// NodoTraspaso: nodo de arbol para almacenar un Traspaso
public class NodoTraspaso {
    Traspaso traspaso;
    NodoTraspaso izquierdo;
    NodoTraspaso derecho;

    public NodoTraspaso(Traspaso t) {
        this.traspaso = t;
        this.izquierdo = null;
        this.derecho = null;
    }
}

// ArbolTraspasos: BST para gestionar Traspasos por placa
public class ArbolTraspasos {
    private NodoTraspaso raiz;
    public ArbolTraspasos() { raiz = null; }
    public void insertar(Traspaso t) { raiz = insertarRec(raiz, t); }
    private NodoTraspaso insertarRec(NodoTraspaso nodo, Traspaso t) {
        if (nodo == null) return new NodoTraspaso(t);
        if (t.getPlaca().compareTo(nodo.traspaso.getPlaca()) < 0) nodo.izquierdo =
insertarRec(nodo.izquierdo, t);
        else if (t.getPlaca().compareTo(nodo.traspaso.getPlaca()) > 0) nodo.derecho =
insertarRec(nodo.derecho, t);
        return nodo;
    }
    public Traspaso buscar(String placa) { return buscarRec(raiz, placa); }
```

```
    private Traspaso buscarRec(NodoTraspaso nodo, String placa) {
        if (nodo == null) return null;
        if (placa.equals(nodo.traspaso.getPlaca())) return nodo.traspaso;
        if (placa.compareTo(nodo.traspaso.getPlaca()) < 0) return buscarRec(nodo.izquierdo, placa);
        return buscarRec(nodo.derecho, placa);
    }
}

// --- CLASE Interfaz.java: ventana principal Swing con botones y carga de datos ---
import java.awt.BorderLayout;  // Importa BorderLayout para agregar paneles
import java.awt.Color;         // Importa Color para manejo de colores
import java.io.File;           // Importa File para manejo de archivos
import javax.swing.JFrame;     // JFrame para la ventana principal
import javax.swing.JPanel;     // JPanel para paneles internos

/**
 * Interfaz: clase principal que extiende JFrame para la GUI de la aplicacion.
 * Contiene botones para diferentes vistas (vehiculos, multas, traspasos, etc.)
 */
public class Interfaz extends JFrame {
    private ArbolVehiculos arbolVehiculos;     // Arbol con datos de vehiculos
    private ListaDobleMultas listaMultas;      // Lista doble circular de multas
    private ArbolAVL arbolAVL;                 // Arbol AVL para estructuras balanceadas
    private TraspasoV traspasoVentana;         // Ventana interna para traspasos
    private final Color COLOR_ACCENT = new Color(231, 76, 60);  // Color secundario (rojo)
    private final Color COLOR = new Color(52, 152, 219);        // Color principal (azul)

    /**
     * Constructor Interfaz: inicializa componentes y carga datos una sola vez.
```

```
 */
public Interfaz() {
   initComponents();            // Metodo generado por NetBeans inicializa la GUI
   cargarDatosSoloUnaVez();      // Carga vehiculos y multas desde archivos
   // Obtiene referencias globales a estructuras si ya fueron creadas
   this.arbolVehiculos = GestorDatosGlobal.getVehiculos();
   this.arbolAVL = GestorDatosGlobal.getArbolAVL();
   // Al iniciar, muestra el panel de bienvenida
   Bienvenida bien = new Bienvenida();
   setPanel(bien.getJpabelbien());
   System.out.println("Vehiculos y multas cargados automaticamente al iniciar.");
}

/**
 * setPanel: agrega un JPanel al centro de jPanel5 y actualiza interfaz
 * @param panel JPanel a mostrar
 */
private void setPanel(JPanel panel) {
   panel.setSize(974, 547);              // Define tamaño del panel
   panel.setLocation(0, 0);              // Posiciona en coordenada (0,0)
   jPanel5.removeAll();                  // Limpia panel actual
   jPanel5.add(panel, BorderLayout.CENTER);  // Agrega nuevo panel al centro
   jPanel5.revalidate();                 // Refresca componente
   jPanel5.repaint();                    // Redibuja GUI
}

/**
 * cargarDatosSoloUnaVez: lee datos de vehiculos y multas desde carpetas fijas.
 * Solo se ejecuta al iniciar la aplicacion.
```

```
     */
    private void cargarDatosSoloUnaVez() {
        File carpetaVehiculos = new File("C:/Users/josue/Downloads/SIRVE_Datos_Vehiculos
DataSet");
        File carpetaMultas = new File("src/multas");
        CargadorDatos cargador = new CargadorDatos();       // Crea objeto cargador
        cargador.cargarDesdeCarpeta(carpetaVehiculos);      // Carga todos los archivos .txt
        this.arbolVehiculos = cargador.getArbolVehiculos(); // Guarda arbol de vehiculos
        this.listaMultas = cargador.getListaMultas();       // Guarda lista de multas
    }

    // Metodos actionPerformed para cada boton, muestran panels correspondientes

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        // Mostrar panel de registro de vehiculos
        RegistrarVehiculo reg = new RegistrarVehiculo();
        setPanel(reg.getRegistrarVehiculo());
    }

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
        // Mostrar vista de CargarVehiculos (busqueda)
        CargarVehiculos buscar = new CargarVehiculos();
        setPanel(buscar.getContenido());
    }

    private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
        // Mostrar panel de gestion de multas
        MultasV multa = new MultasV();
        setPanel(multa.getMultasV());
```

```
}

private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
   // Mostrar panel de traspasos
   TraspasoV traspaso = new TraspasoV();
   setPanel(traspaso.getTraspasoV());
}

private void jButton6ActionPerformed(java.awt.event.ActionEvent evt) {
   // Mostrar panel de exportacion/encriptacion
   EportarVentana venta = new EportarVentana();
   setPanel(venta.getventana());
}

private void jButton13ActionPerformed(java.awt.event.ActionEvent evt) {
   // Volver al panel de bienvenida
   Bienvenida bien = new Bienvenida();
   setPanel(bien.getJpabelbien());
}

// Eventos de mouse para cambiar color de fondo al entrar/salir botones
private void configurarHover() {
   // Ejemplo: jButton1 cambia a COLOR_ACCENT al pasar mouse
   jButton1.addMouseListener(new java.awt.event.MouseAdapter() {
      public void mouseEntered(java.awt.event.MouseEvent evt) {
         jButton1.setBackground(COLOR_ACCENT);
      }
      public void mouseExited(java.awt.event.MouseEvent evt) {
         jButton1.setBackground(COLOR);
```

```
        }
    });
    // Repetir para cada boton (jButton2, jButton3, ...)
  }

  // Metodos initComponents, main, y demas generados por NetBeans no se comentan manualmente
  // Fin de clase Interfaz

// Clase TablaVehiculos: panel Swing que muestra vehiculos en un JTable segun distintos recorridos de arboles
public class TablaVehiculos extends javax.swing.JPanel {
  private ArbolVehiculos arbolVehiculos;  // Arbol ABB con vehiculos cargados
  private ArbolAVL arbolAVL;              // Arbol AVL balanceado con los mismos vehiculos

  /**
   * Constructor TablaVehiculos
   * Inicializa componentes y obtiene referencias a estructuras globales
   */
  public TablaVehiculos() {
    initComponents();  // Metodo NetBeans que crea la GUI del JPanel
    this.arbolVehiculos = GestorDatosGlobal.getVehiculos();  // Carga ABB global
    this.arbolAVL = GestorDatosGlobal.getArbolAVL();        // Carga AVL global
  }

  /**
   * getTablaVehiculos - retorna el panel principal con la tabla
   */
  public JPanel getTablaVehiculos() {
    return PanelHistorialMultas;  // Panel definido en initComponents
```

```
}

/**
 * cargarInordenEnTabla - vacia la tabla y agrega filas segun recorrido in-orden en ABB
 */
public void cargarInordenEnTabla(JTable tabla) {
    DefaultTableModel modelo = (DefaultTableModel) tabla.getModel();
    modelo.setRowCount(0);                          // Limpia filas previas
    recorrerInorden(arbolVehiculos.getRaiz(), modelo);  // Agrega datos recursivamente
}

/**
 * cargarPreordenEnTabla - similar a inorden pero pre-orden
 */
public void cargarPreordenEnTabla(JTable tabla) {
    DefaultTableModel modelo = (DefaultTableModel) tabla.getModel();
    modelo.setRowCount(0);
    recorrerPreorden(arbolVehiculos.getRaiz(), modelo);
}

/**
 * cargarPostordenEnTabla - similar, usando recorrido post-orden
 */
public void cargarPostordenEnTabla(JTable tabla) {
    DefaultTableModel modelo = (DefaultTableModel) tabla.getModel();
    modelo.setRowCount(0);
    recorrerPostorden(arbolVehiculos.getRaiz(), modelo);
}
```

```
// Metodos recursivos para cada tipo de recorrido sobre el ABB
private void recorrerInorden(NodoVehiculo nodo, DefaultTableModel modelo) {
    if (nodo != null) {
        recorrerInorden(nodo.izquierdo, modelo);
        agregarAFila(nodo.vehiculo, modelo);
        recorrerInorden(nodo.derecho, modelo);
    }
}

private void recorrerPreorden(NodoVehiculo nodo, DefaultTableModel modelo) {
    if (nodo != null) {
        agregarAFila(nodo.vehiculo, modelo);
        recorrerPreorden(nodo.izquierdo, modelo);
        recorrerPreorden(nodo.derecho, modelo);
    }
}

private void recorrerPostorden(NodoVehiculo nodo, DefaultTableModel modelo) {
    if (nodo != null) {
        recorrerPostorden(nodo.izquierdo, modelo);
        recorrerPostorden(nodo.derecho, modelo);
        agregarAFila(nodo.vehiculo, modelo);
    }
}

/**
 * agregarAFila - construye y añade una fila al modelo con datos del vehiculo
 */
private void agregarAFila(Vehiculo v, DefaultTableModel modelo) {
```

```
    modelo.addRow(new Object[]{
        v.getPlaca(),                   // Columna de placa
        v.getDpi(),                     // Columna de DPI (identificacion)
        v.getPropietario(),             // Nombre del propietario
        v.getMarca(),                   // Marca del vehiculo
        v.getModelo(),                  // Modelo del vehiculo
        v.getAnio(),                    // Año de fabricacion
        v.getMultas(),                  // Numero de multas asociadas
        v.getTraspasos(),               // Numero de traspasos realizados
        v.getDepartamento()             // Departamento asociado
    });
}

// Evento para eliminar vehiculo seleccionado en JTable
private void EliminarVehiculoActionPerformed(java.awt.event.ActionEvent evt) {
    int fila = jTableVehiculos.getSelectedRow();
    if (fila == -1) {
        JOptionPane.showMessageDialog(this, "Selecciona un vehiculo para eliminar.");
        return;
    }
    String placa = jTableVehiculos.getValueAt(fila, 0).toString();  // Extrae placa

    // Elimina en ABB y AVL
    boolean borradoABB = arbolVehiculos.eliminar(placa);
    boolean borradoAVL = arbolAVL.eliminar(placa);

    if (borradoABB && borradoAVL) {
        ((DefaultTableModel) jTableVehiculos.getModel()).removeRow(fila);
        JOptionPane.showMessageDialog(this, "Vehiculo eliminado de ambos arboles.");
```

```
    } else {
        JOptionPane.showMessageDialog(this,
            "Error al eliminar en uno o ambos arboles para placa: " + placa,
            "Eliminar fallo",
            JOptionPane.WARNING_MESSAGE);
    }
}

// Eventos de botones de recorrido que invocan los metodos de carga
private void InordenActionPerformed(java.awt.event.ActionEvent evt) {
    cargarInordenEnTabla(jTableVehiculos);
}
private void PreOrdenActionPerformed(java.awt.event.ActionEvent evt) {
    cargarPreordenEnTabla(jTableVehiculos);
}
private void PostOrdenActionPerformed(java.awt.event.ActionEvent evt) {
    cargarPostordenEnTabla(jTableVehiculos);
}

// Evento para modificar datos de vehiculo seleccionado
private void ModificarVehiculoActionPerformed(java.awt.event.ActionEvent evt) {
    int fila = jTableVehiculos.getSelectedRow();
    if (fila == -1) {
        JOptionPane.showMessageDialog(this, "Selecciona un vehiculo para modificar.");
        return;
    }
    DefaultTableModel model = (DefaultTableModel) jTableVehiculos.getModel();
    // Obtiene valores actuales de la fila
    String placa       = model.getValueAt(fila, 0).toString();
```

```
    String dpi        = model.getValueAt(fila, 1).toString();
    String propietario = model.getValueAt(fila, 2).toString();
    String marca      = model.getValueAt(fila, 3).toString();
    String modelo     = model.getValueAt(fila, 4).toString();
    String anio       = model.getValueAt(fila, 5).toString();
    String multas     = model.getValueAt(fila, 6).toString();
    String traspasos  = model.getValueAt(fila, 7).toString();
    String depto      = model.getValueAt(fila, 8).toString();

    // Crea objeto Vehiculo con datos actualizados
    Vehiculo actualizado = new Vehiculo(
        placa, dpi, propietario, marca, modelo, anio, multas, traspasos, depto
    );
    // Modifica en ABB y AVL
    boolean modABB = arbolVehiculos.modificarVehiculo(placa, actualizado);
    boolean modAVL = arbolAVL.modificar(placa, actualizado);

    if (modABB && modAVL) {
        JOptionPane.showMessageDialog(this, "Vehiculo modificado correctamente.");
    } else {
        JOptionPane.showMessageDialog(this,
            "Error al modificar vehiculo con placa: " + placa,
            "Modificar fallo",
            JOptionPane.WARNING_MESSAGE);
    }
}

// Eventos para mostrar recorrido de arbol AVL en la tabla
private void PreOrdenAVLActionPerformed(java.awt.event.ActionEvent evt) {
```

```
    DefaultTableModel m = (DefaultTableModel) jTableVehiculos.getModel();
    m.setRowCount(0);
    for (Vehiculo v : arbolAVL.preOrderList()) {
        m.addRow(new Object[]{
            v.getPlaca(), v.getDpi(), v.getPropietario(),
            v.getMarca(), v.getModelo(), v.getAnio(),
            v.getMultas(), v.getTraspasos(), v.getDepartamento()
        });
    }
}

private void PostOrdenAVLActionPerformed(java.awt.event.ActionEvent evt) {
    DefaultTableModel m = (DefaultTableModel) jTableVehiculos.getModel();
    m.setRowCount(0);
    for (Vehiculo v : arbolAVL.postOrderList()) {
        m.addRow(new Object[]{
            v.getPlaca(), v.getDpi(), v.getPropietario(),
            v.getMarca(), v.getModelo(), v.getAnio(),
            v.getMultas(), v.getTraspasos(), v.getDepartamento()
        });
    }
}

private void InOrdenAVLActionPerformed(java.awt.event.ActionEvent evt) {
    DefaultTableModel m = (DefaultTableModel) jTableVehiculos.getModel();
    m.setRowCount(0);
    for (Vehiculo v : arbolAVL.inOrderList()) {
        m.addRow(new Object[]{
            v.getPlaca(), v.getDpi(), v.getPropietario(),
```

```
            v.getMarca(), v.getModelo(), v.getAnio(),
            v.getMultas(), v.getTraspasos(), v.getDepartamento()
        });
    }
  }

  // Fin de clase TablaVehiculos

/*
 * -- EXPLICACION DETALLADA CON CODIGO DE ArbolAVL --
 *
 * public class ArbolAVL {
 *     // Nodo interno con datos y altura
 *     class NodoAVL {
 *         Vehiculo vehiculo;        // Dato almacenado
 *         NodoAVL izquierdo, derecho; // Hijos izquierdo y derecho
 *         int altura;               // Altura del subarbol
 *
 *         NodoAVL(Vehiculo v) {
 *             vehiculo = v;         // Asigna dato
 *             altura = 1;           // Nodo hoja inicia con altura 1
 *         }
 *     }
 *
 *     private NodoAVL raiz;         // Raiz del arbol
 *
 *     // Metodo altura: devuelve 0 si el nodo es null, sino altura almacenada
 *     private int altura(NodoAVL n) { return (n == null) ? 0 : n.altura; }
 *
```

```
*     // Rotacion derecha: se usa en caso Izquierda-Izquierda
*     private NodoAVL rotarDerecha(NodoAVL y) {
*         NodoAVL x = y.izquierdo;    // x es hijo izquierdo de y
*         NodoAVL T2 = x.derecho;     // T2 sera subarbol intermedio
*         x.derecho = y;              // mueve y a subarbol derecho de x
*         y.izquierdo = T2;           // reubica T2 como hijo izquierdo de y
*         // Actualiza alturas despues de la rotacion
*         y.altura = Math.max(altura(y.izquierdo), altura(y.derecho)) + 1;
*         x.altura = Math.max(altura(x.izquierdo), altura(x.derecho)) + 1;
*         return x;                   // x es nueva raiz de subarbol
*     }
*
*     // Rotacion izquierda: simetrica para caso Derecha-Derecha
*     private NodoAVL rotarIzquierda(NodoAVL x) {
*         NodoAVL y = x.derecho;
*         NodoAVL T2 = y.izquierdo;
*         y.izquierdo = x;
*         x.derecho = T2;
*         x.altura = Math.max(altura(x.izquierdo), altura(x.derecho)) + 1;
*         y.altura = Math.max(altura(y.izquierdo), altura(y.derecho)) + 1;
*         return y;
*     }
*
*     // Factor de balance: diferencia entre alturas de subarbol izquierdo y derecho
*     private int getBalance(NodoAVL n) { return (n == null) ? 0 : altura(n.izquierdo) -
altura(n.derecho); }
*
*     // Insercion recursiva con balanceo
*     public void insertar(Vehiculo v) { raiz = insertarRec(raiz, v); }
```

```
*    private NodoAVL insertarRec(NodoAVL nodo, Vehiculo v) {
*        if (nodo == null) return new NodoAVL(v); // caso base
*        // Comparacion de placas para decidir posicion
*        if (v.getPlaca().compareTo(nodo.vehiculo.getPlaca()) < 0)
*            nodo.izquierdo = insertarRec(nodo.izquierdo, v);
*        else if (v.getPlaca().compareTo(nodo.vehiculo.getPlaca()) > 0)
*            nodo.derecho = insertarRec(nodo.derecho, v);
*        else return nodo; // ignorar duplicados
*
*        // Actualiza altura y calcula balance
*        nodo.altura = 1 + Math.max(altura(nodo.izquierdo), altura(nodo.derecho));
*        int balance = getBalance(nodo);
*        // Aplica caso de rotaciones segun balance y valor de placa
*        if (balance > 1 && v.getPlaca().compareTo(nodo.izquierdo.vehiculo.getPlaca()) < 0)
*            return rotarDerecha(nodo);          // Izquierda-Izquierda
*        if (balance < -1 && v.getPlaca().compareTo(nodo.derecho.vehiculo.getPlaca()) > 0)
*            return rotarIzquierda(nodo);        // Derecha-Derecha
*        if (balance > 1 && v.getPlaca().compareTo(nodo.izquierdo.vehiculo.getPlaca()) > 0) {
*            nodo.izquierdo = rotarIzquierda(nodo.izquierdo); // Izquierda-Derecha
*            return rotarDerecha(nodo);
*        }
*        if (balance < -1 && v.getPlaca().compareTo(nodo.derecho.vehiculo.getPlaca()) < 0) {
*            nodo.derecho = rotarDerecha(nodo.derecho);      // Derecha-Izquierda
*            return rotarIzquierda(nodo);
*        }
*        return nodo; // sin rotacion necesaria
*    }
*
*    // Busqueda simple como en ABB
```

```
*     public Vehiculo buscar(String placa) { return buscarRec(raiz, placa.toUpperCase()); }
*     private Vehiculo buscarRec(NodoAVL nodo, String placa) {
*         if (nodo == null) return null;
*         if (nodo.vehiculo.getPlaca().equalsIgnoreCase(placa)) return nodo.vehiculo;
*         if (placa.compareTo(nodo.vehiculo.getPlaca()) < 0)
*             return buscarRec(nodo.izquierdo, placa);
*         else return buscarRec(nodo.derecho, placa);
*     }
* }
*
* -- EXPLICACION DETALLADA CON CODIGO DE ArbolVehiculos --
*
* public class ArbolVehiculos {
*     private NodoVehiculo raiz; // raiz del ABB
*
*     public void insertar(Vehiculo v) { raiz = insertarRec(raiz, v); }
*     private NodoVehiculo insertarRec(NodoVehiculo nodo, Vehiculo v) {
*         if (nodo == null) return new NodoVehiculo(v); // crea nodo
*         if (v.getPlaca().compareTo(nodo.vehiculo.getPlaca()) < 0)
*             nodo.izquierdo = insertarRec(nodo.izquierdo, v);
*         else if (v.getPlaca().compareTo(nodo.vehiculo.getPlaca()) > 0)
*             nodo.derecho = insertarRec(nodo.derecho, v);
*         return nodo; // sin balanceo
*     }
*
*     public Vehiculo buscar(String placa) { return buscarRec(raiz, placa); }
*     private Vehiculo buscarRec(NodoVehiculo nodo, String placa) {
*         if (nodo == null) return null;
*         if (nodo.vehiculo.getPlaca().equals(placa)) return nodo.vehiculo;
```

```
*        if (placa.compareTo(nodo.vehiculo.getPlaca()) < 0)
*            return buscarRec(nodo.izquierdo, placa);
*        else return buscarRec(nodo.derecho, placa);
*    }
*
*    // recorrido in-orden imprime nodos ordenados
*    public void recorridoInOrden() { recorridoInOrdenRec(raiz); }
*    private void recorridoInOrdenRec(NodoVehiculo nodo) {
*        if (nodo != null) {
*            recorridoInOrdenRec(nodo.izquierdo);
*            System.out.println(nodo.vehiculo);
*            recorridoInOrdenRec(nodo.derecho);
*        }
*    }
* }
*/

// Clase BuscarMulta: panel para buscar multas por placa y mostrarlas en tabla
public class BuscarMulta extends javax.swing.JPanel {
    private ListaDobleMultas listaMultas;  // Lista doble circular de multas

    /**
     * Constructor BuscarMulta
     * Obtiene la referencia a la lista de multas global
     */
    public BuscarMulta() {
        initComponents();
        this.listaMultas = GestorDatosGlobal.getMultas();
    }
```

```
    /**
     * getPanelBuscarM - retorna el panel principal de esta vista
     */
    public JPanel getPanelBuscarM() {
        return jPanelm;
    }

    /**
     * Evento al presionar boton BuscarMulta
     * Lee la placa, busca en lista bidireccional y llena la tabla
     */
    private void BuscarMultaActionPerformed(java.awt.event.ActionEvent evt) {
        String placaBuscar = jTextField1.getText().trim();
        // Busca todas las multas para la placa usando busqueda eficiente
        ArrayList<Multa> multasEncontradas =
listaMultas.buscarMultasPorPlacaBidireccional(placaBuscar);

        DefaultTableModel modelo = (DefaultTableModel) jTable1.getModel();
        modelo.setRowCount(0);  // Limpia filas previas

        if (!multasEncontradas.isEmpty()) {
            // Agrega fila por cada multa encontrada
            for (Multa m : multasEncontradas) {
                modelo.addRow(new Object[]{
                    m.getPlaca(),   // Placa del vehiculo
                    m.getFecha(),   // Fecha de emision
                    m.getMotivo(),  // Motivo de la multa
                    m.getMonto(),   // Monto de la multa
```

```
                m.getDepartamento() // Departamento asociado
            });
        }
    } else {
        // Mensaje si no hay resultados
        JOptionPane.showMessageDialog(this, "No se encontraron multas para la placa: " +
placaBuscar);
    }
  }
}

// Clase BuscarVehiculo: panel para buscar vehiculos en ABB y AVL y comparar tiempos
public class BuscarVehiculo extends javax.swing.JPanel {
  private ArbolVehiculos arbolVehiculos;  // ABB de vehiculos
  private ArbolAVL arbolAVL;              // AVL de vehiculos balanceado

  /**
   * Constructor BuscarVehiculo
   * Configura placeholder y obtiene referencias a las estructuras globales
   */
  public BuscarVehiculo() {
    initComponents();
    this.arbolVehiculos = GestorDatosGlobal.getVehiculos();
    this.arbolAVL = GestorDatosGlobal.getArbolAVL();
    agregarPlaceholder(textfieldbuscar6, "Formato: P123ABC");
  }

  /**
   * getPanelBuscarVehiculo - retorna el panel principal de esta vista
```

```
 */
public JPanel getpanelBuscarVehiculo() {
   return jPanel6;
}

/**
 * agregarPlaceholder - muestra texto gris en campo hasta que tenga foco
 */
private void agregarPlaceholder(JTextField campo, String textoPlaceholder) {
   campo.setForeground(Color.GRAY);
   campo.setText(textoPlaceholder);

   campo.addFocusListener(new FocusAdapter() {
      @Override
      public void focusGained(FocusEvent e) {
         if (campo.getText().equals(textoPlaceholder)) {
            campo.setText("");
            campo.setForeground(Color.BLACK);
         }
      }
      @Override
      public void focusLost(FocusEvent e) {
         if (campo.getText().isEmpty()) {
            campo.setForeground(Color.GRAY);
            campo.setText(textoPlaceholder);
         }
      }
   });
}
```

```
/**
 * Evento al presionar boton BuscarVEhiculo (ABB)
 * Mide tiempo de busqueda ABB y muestra datos o mensaje
 */
private void BuscarVehiculo6BuscarVehiculoActionPerformed(java.awt.event.ActionEvent evt) {
    String placa = textfieldbuscar6.getText().trim().toUpperCase();
    if (placa.isEmpty()) {
        JOptionPane.showMessageDialog(this, "Ingrese una placa para buscar.");
        return;
    }
    long startAbb = System.nanoTime();
    Vehiculo v = arbolVehiculos.buscar(placa);
    long endAbb = System.nanoTime();
    jLabel3ABB.setText(String.format("%.3f ms", (endAbb - startAbb) / 1_000_000.0));

    if (v != null) {
        // Muestra datos en labels correspondientes
        placamostrar6.setText(v.getPlaca());
        dpimostrar6.setText(v.getDpi());
        propietariomostar6.setText(v.getPropietario());
        marcamostrar6.setText(v.getMarca());
        modelomostrar6.setText(v.getModelo());
        anomostrar6.setText(v.getAnio());
        Multasmostrar6.setText(v.getMultas());
        Traspasomostrar6.setText(v.getTraspasos());
        jLabel2.setText(v.getDepartamento());
    } else {
        // Limpia campos y muestra alerta
```

```
        JOptionPane.showMessageDialog(this,
            "No se encontro vehiculo con placa: " + placa,
            "Busqueda fallida",
            JOptionPane.WARNING_MESSAGE);
        placamostrar6.setText(""); dpimostrar6.setText("");
        propietariomostar6.setText(""); marcamostrar6.setText("");
        modelomostrar6.setText(""); anomostrar6.setText("");
        Multasmostrar6.setText(""); Traspasomostrar6.setText("");
    }
}

/**
 * Evento al presionar boton BuscarAVL
 * Mide tiempo de busqueda AVL y actualiza tiempo en label
 */
private void BuscarAVLActionPerformed(java.awt.event.ActionEvent evt) {
    String placa = textfieldbuscar6.getText().trim().toUpperCase();
    long startAvl = System.nanoTime();
    Vehiculo v = arbolAVL.buscar(placa);
    jLabel4AVL.setText(String.format("%.3f ms", (System.nanoTime() - startAvl) / 1_000_000.0));

    if (v != null) {
        // Muestra datos (mismos labels que ABB)
        placamostrar6.setText(v.getPlaca());
        dpimostrar6.setText(v.getDpi());
        propietariomostar6.setText(v.getPropietario());
        marcamostrar6.setText(v.getMarca());
        modelomostrar6.setText(v.getModelo());
        anomostrar6.setText(v.getAnio());
```

```
            Multasmostrar6.setText(v.getMultas());
            Traspasomostrar6.setText(v.getTraspasos());
            jLabel2.setText(v.getDepartamento());
        } else {
            JOptionPane.showMessageDialog(this,
                "No se encontro vehiculo con placa: " + placa,
                "Busqueda AVL fallida",
                JOptionPane.WARNING_MESSAGE);
        }
    }
}

// Clase CargarVehiculos: JFrame que muestra opciones para cargar y visualizar vehiculos
public class CargarVehiculos extends javax.swing.JFrame {
    private ArbolVehiculos arbolVehiculos;  // ABB global de vehiculos

    /**
     * Constructor CargarVehiculos
     * Inicializa componentes, obtiene arbol global y carga panel de busqueda
     */
    public CargarVehiculos() {
        initComponents();                   // Inicia GUI generada por NetBeans
        this.arbolVehiculos = GestorDatosGlobal.getVehiculos();  // Obtiene ABB global
        cargarUna();                        // Llama a mostrar panel de busqueda
    }

    /**
     * cargarUna - instancia y muestra el panel de BuscarVehiculo en jPanel2
     */
```

```
private void cargarUna() {
   BuscarVehiculo busca = new BuscarVehiculo();            // Crea panel de busqueda
   Paneles(busca.getpanelBuscarVehiculo());                // Lo añade a la vista
}

/**
 * getContenido - retorna el panel principal jPanel1 de este JFrame
 */
public JPanel getContenido() {
   return jPanel1;
}

/**
 * Paneles - metodo reutilizable para cambiar el contenido de jPanel2
 * @param a JPanel a mostrar
 */
public void Paneles(JPanel a) {
   a.setSize(974, 547);              // Define tamano del panel
   a.setLocation(0, 0);              // Posicion en (0,0)
   jPanel2.removeAll();              // Limpia contenido previo
   jPanel2.add(a, BorderLayout.CENTER); // Agrega nuevo panel
   jPanel2.revalidate();             // Refresca layout
   jPanel2.repaint();                // Redibuja componente
}

/**
 * Panelprincipal - similar a Paneles pero sin limpiar primero
 */
public void Panelprincipal(JPanel a) {
```

```
        jPanel2.add(a, BorderLayout.CENTER);
        jPanel2.revalidate();
        jPanel2.repaint();
    }

    /**
     * Evento jButton1: muestra panel de BuscarVehiculo de nuevo
     */
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        BuscarVehiculo busca = new BuscarVehiculo();
        Paneles(busca.getpanelBuscarVehiculo());
    }

    /**
     * Evento TablaMultas: muestra tabla de vehiculos con TablaVehiculos
     */
    private void TablaMultasActionPerformed(java.awt.event.ActionEvent evt) {
        TablaVehiculos tabla = new TablaVehiculos();
        Paneles(tabla.getTablaVehiculos());
    }

    /**
     * Evento historialtraspaso: (pendiente de implementacion)
     */
    private void historialtraspasoActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO: implementar accion para mostrar historial de traspasos
    }
}}
```

```
// Clase ListaDobleMultas: implementa una lista doblemente enlazada de multas
public class ListaDobleMultas {
    private NodoMulta cabeza; // Primer nodo de la lista
    private NodoMulta cola;   // Ultimo nodo de la lista

    /**
     * insertar - agrega una multa al final de la lista
     * @param multa objeto Multa a insertar
     */
    public void insertar(Multa multa) {
        NodoMulta nuevo = new NodoMulta(multa); // Crea nodo con la multa
        if (cabeza == null) {
            // Lista vacia: nuevo es cabeza y cola
            cabeza = cola = nuevo;
        } else {
            // Conecta el nodo actual cola con el nuevo
            cola.setSiguiente(nuevo);
            nuevo.setAnterior(cola);
            cola = nuevo; // Actualiza cola
        }
    }

    /**
     * getCabeza - retorna el primer nodo de la lista
     * @return NodoMulta cabeza
     */
    public NodoMulta getCabeza() {
        return cabeza;
    }
```

```
/**
 * buscarMultasPorPlacaBidireccional - busca coincidencias partiendo de ambos extremos
 * @param placa texto de placa a buscar (sin distinguir mayusculas)
 * @return ArrayList<Multa> con todas las multas encontradas
 */
public ArrayList<Multa> buscarMultasPorPlacaBidireccional(String placa) {
    ArrayList<Multa> resultado = new ArrayList<>();
    NodoMulta desdeInicio = cabeza;
    NodoMulta desdeFin = cola;

    // Recorre simultaneamente desde ambos extremos hasta encontrarse o cruzarse
    while (desdeInicio != null && desdeFin != null
           && desdeInicio != desdeFin
           && desdeInicio.getAnterior() != desdeFin) {
        if (desdeInicio.getMulta().getPlaca().equalsIgnoreCase(placa)) {
            resultado.add(desdeInicio.getMulta());
        }
        if (desdeFin.getMulta().getPlaca().equalsIgnoreCase(placa)) {
            resultado.add(desdeFin.getMulta());
        }
        desdeInicio = desdeInicio.getSiguiente();
        desdeFin = desdeFin.getAnterior();
    }

    // Si punteros coinciden en el mismo nodo, verificar una vez
    if (desdeInicio != null && desdeInicio == desdeFin) {
        if (desdeInicio.getMulta().getPlaca().equalsIgnoreCase(placa)) {
            resultado.add(desdeInicio.getMulta());
```

```
        }
    }
    return resultado;
}

/**
 * existePlaca - chequea si hay alguna multa con la placa dada
 * @param placa cadena de placa
 * @return true si existe al menos una coincidencia
 */
public boolean existePlaca(String placa) {
    NodoMulta actual = cabeza;
    while (actual != null) {
        if (actual.getMulta().getPlaca().equalsIgnoreCase(placa)) {
            return true;
        }
        actual = actual.getSiguiente();
    }
    return false;
}

/**
 * eliminarMulta - remueve la primera multa que coincida en todos los campos
 * @param placa texto de placa
 * @param fecha fecha de multa
 * @param motivo motivo de la multa
 * @param monto monto de la multa (string para comparar)
 * @param departamento departamento
 * @return true si se elimino correctamente
```

```
 */
public boolean eliminarMulta(String placa,
                             String fecha,
                             String motivo,
                             String monto,
                             String departamento) {
   NodoMulta actual = cabeza;
   while (actual != null) {
      Multa m = actual.getMulta();
      if (m.getPlaca().equalsIgnoreCase(placa)
       && m.getFecha().equals(fecha)
       && m.getMotivo().equals(motivo)
       && String.valueOf(m.getMonto()).equals(monto)
       && m.getDepartamento().equals(departamento)) {

         // Reconecta nodos anterior y siguiente
         if (actual.getAnterior() != null) {
            actual.getAnterior().setSiguiente(actual.getSiguiente());
         } else {
            // Se elimina cabeza
            cabeza = actual.getSiguiente();
         }
         if (actual.getSiguiente() != null) {
            actual.getSiguiente().setAnterior(actual.getAnterior());
         } else {
            // Se elimina cola
            cola = actual.getAnterior();
         }
         return true;
```

```
        }
        actual = actual.getSiguiente();
    }
    return false; // No se encontro coincidencia
}

/**
 * toList - convierte la lista en un ArrayList en orden desde la cabeza
 * @return List<Multa> con todas las multas en secuencia
 */
public List<Multa> toList() {
    List<Multa> resultado = new ArrayList<>();
    NodoMulta actual = cabeza;
    while (actual != null) {
        resultado.add(actual.getMulta());
        actual = actual.getSiguiente();
    }
    return resultado;
}

// Clase interna NodoMulta: representa un nodo de la lista doblemente enlazada
public class NodoMulta {
    private Multa multa;         // Objeto de datos
    private NodoMulta anterior;   // Nodo previo
    private NodoMulta siguiente;  // Nodo siguiente

    public NodoMulta(Multa multa) {
        this.multa = multa;
        this.anterior = null;
```

```
        this.siguiente = null;
    }

    // Getters y setters para enlaces y dato
    public Multa getMulta() { return multa; }
    public NodoMulta getAnterior() { return anterior; }
    public void setAnterior(NodoMulta anterior) { this.anterior = anterior; }
    public NodoMulta getSiguiente() { return siguiente; }
    public void setSiguiente(NodoMulta siguiente) { this.siguiente = siguiente; }
  }
}



// Clase ListaDobleMultas: implementa una lista doblemente enlazada de multas
public class ListaDobleMultas {
  private NodoMulta cabeza; // Primer nodo de la lista
  private NodoMulta cola;   // Ultimo nodo de la lista

  /**
   * insertar - agrega una multa al final de la lista
   * @param multa objeto Multa a insertar
   */
  public void insertar(Multa multa) {
    NodoMulta nuevo = new NodoMulta(multa); // Crea nodo con la multa
    if (cabeza == null) {
      // Lista vacia: nuevo es cabeza y cola
      cabeza = cola = nuevo;
    } else {
```

```
        // Conecta el nodo actual cola con el nuevo
        cola.setSiguiente(nuevo);
        nuevo.setAnterior(cola);
        cola = nuevo; // Actualiza cola
    }
}

/**
 * getCabeza - retorna el primer nodo de la lista
 * @return NodoMulta cabeza
 */
public NodoMulta getCabeza() {
    return cabeza;
}

/**
 * buscarMultasPorPlacaBidireccional - busca coincidencias partiendo de ambos extremos
 * @param placa texto de placa a buscar (sin distinguir mayusculas)
 * @return ArrayList<Multa> con todas las multas encontradas
 */
public ArrayList<Multa> buscarMultasPorPlacaBidireccional(String placa) {
    ArrayList<Multa> resultado = new ArrayList<>();
    NodoMulta desdeInicio = cabeza;
    NodoMulta desdeFin = cola;

    // Recorre simultaneamente desde ambos extremos hasta encontrarse o cruzarse
    while (desdeInicio != null && desdeFin != null
            && desdeInicio != desdeFin
            && desdeInicio.getAnterior() != desdeFin) {
```

```
        if (desdeInicio.getMulta().getPlaca().equalsIgnoreCase(placa)) {
            resultado.add(desdeInicio.getMulta());
        }
        if (desdeFin.getMulta().getPlaca().equalsIgnoreCase(placa)) {
            resultado.add(desdeFin.getMulta());
        }
        desdeInicio = desdeInicio.getSiguiente();
        desdeFin = desdeFin.getAnterior();
    }

    // Si punteros coinciden en el mismo nodo, verificar una vez
    if (desdeInicio != null && desdeInicio == desdeFin) {
        if (desdeInicio.getMulta().getPlaca().equalsIgnoreCase(placa)) {
            resultado.add(desdeInicio.getMulta());
        }
    }
    return resultado;
}

/**
 * existePlaca - chequea si hay alguna multa con la placa dada
 * @param placa cadena de placa
 * @return true si existe al menos una coincidencia
 */
public boolean existePlaca(String placa) {
    NodoMulta actual = cabeza;
    while (actual != null) {
        if (actual.getMulta().getPlaca().equalsIgnoreCase(placa)) {
            return true;
```

```
        }
        actual = actual.getSiguiente();
    }
    return false;
}

/**
 * eliminarMulta - remueve la primera multa que coincida en todos los campos
 * @param placa texto de placa
 * @param fecha fecha de multa
 * @param motivo motivo de la multa
 * @param monto monto de la multa (string para comparar)
 * @param departamento departamento
 * @return true si se elimino correctamente
 */
public boolean eliminarMulta(String placa,
                             String fecha,
                             String motivo,
                             String monto,
                             String departamento) {
    NodoMulta actual = cabeza;
    while (actual != null) {
        Multa m = actual.getMulta();
        if (m.getPlaca().equalsIgnoreCase(placa)
         && m.getFecha().equals(fecha)
         && m.getMotivo().equals(motivo)
         && String.valueOf(m.getMonto()).equals(monto)
         && m.getDepartamento().equals(departamento)) {
```

```
            // Reconecta nodos anterior y siguiente
            if (actual.getAnterior() != null) {
                actual.getAnterior().setSiguiente(actual.getSiguiente());
            } else {
                // Se elimina cabeza
                cabeza = actual.getSiguiente();
            }
            if (actual.getSiguiente() != null) {
                actual.getSiguiente().setAnterior(actual.getAnterior());
            } else {
                // Se elimina cola
                cola = actual.getAnterior();
            }
            return true;
        }
        actual = actual.getSiguiente();
    }
    return false; // No se encontro coincidencia
}

/**
 * toList - convierte la lista en un ArrayList en orden desde la cabeza
 * @return List<Multa> con todas las multas en secuencia
 */
public List<Multa> toList() {
    List<Multa> resultado = new ArrayList<>();
    NodoMulta actual = cabeza;
    while (actual != null) {
        resultado.add(actual.getMulta());
```

```
            actual = actual.getSiguiente();
        }
        return resultado;
    }

    // Clase interna NodoMulta: representa un nodo de la lista doblemente enlazada
    public class NodoMulta {
        private Multa multa;         // Objeto de datos
        private NodoMulta anterior;   // Nodo previo
        private NodoMulta siguiente;  // Nodo siguiente

        public NodoMulta(Multa multa) {
            this.multa = multa;
            this.anterior = null;
            this.siguiente = null;
        }

        // Getters y setters para enlaces y dato
        public Multa getMulta() { return multa; }
        public NodoMulta getAnterior() { return anterior; }
        public void setAnterior(NodoMulta anterior) { this.anterior = anterior; }
        public NodoMulta getSiguiente() { return siguiente; }
        public void setSiguiente(NodoMulta siguiente) { this.siguiente = siguiente; }
    }
}
```

```
// Clase GestorDatosGlobal: punto unico de acceso a los datos cargados
public class GestorDatosGlobal {
    // Instancia estatica de CargadorDatos que lee y almacena estructuras
    private static final CargadorDatos cargador = new CargadorDatos();

    // Ruta donde se encuentran los archivos de datos (vehiculos, multas, traspasos)
    private static final File carpetaDatos = new File(
        "C:\\Users\\josue\\Downloads\\SIRVE_Datos_Vehiculos DataSet"
    );

    // Bloque estatica: se ejecuta al cargar la clase por primera vez
    static {
        // Inicia carga de datos desde la carpeta especificada
        cargador.cargarDesdeCarpeta(carpetaDatos);
    }

    /**
     * getVehiculos - retorna el arbol ABB de vehiculos cargado
     * @return ArbolVehiculos con todos los vehiculos leidos
     */
    public static ArbolVehiculos getVehiculos() {
        return cargador.getArbolVehiculos();
    }

    /**
     * getMultas - retorna la lista doble circular de multas cargadas
     * @return ListaDobleMultas con todas las multas leidas
     */
    public static ListaDobleMultas getMultas() {
```

```
        return cargador.getListaMultas();
    }

    /**
     * getTraspasos - retorna la lista circular de traspasos cargados
     * @return ListaCircularTraspasos con todos los traspasos leidos
     */
    public static ListaCircularTraspasos getTraspasos() {
        return cargador.getListaTraspasos();
    }

    /**
     * getArbolAVL - retorna el arbol AVL de vehiculos balanceado
     * @return ArbolAVL con todos los vehiculos leidos y balanceados
     */
    public static ArbolAVL getArbolAVL() {
        return cargador.getArbolAVL();
    }

    /**
     * getCarpetaDatos - expone la ruta de los archivos de datos
     * @return File que representa la carpeta de datos
     */
    public static File getCarpetaDatos() {
        return carpetaDatos;
    }
}
```

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class UtilidadesCifradoCesar {

   /**
    * Método para aplicar cifrado César a un texto dado.
    * @param texto El texto original a cifrar.
    * @param desplazamiento La cantidad de posiciones que se desplaza cada letra (clave).
    * @return Texto cifrado.
    */
   public static String cifradoCesar(String texto, int desplazamiento) {
      StringBuilder resultado = new StringBuilder();
      // Ajustamos desplazamiento para que siempre esté en rango 0-25 (alfabeto inglés)
      desplazamiento %= 26;
      // Recorremos caracter a caracter el texto
      for (char c : texto.toCharArray()) {
         if (Character.isLetter(c)) { // Solo letras serán cifradas
            // Base es 'A' o 'a' según sea mayúscula o minúscula para mantener caso
            char base = Character.isUpperCase(c) ? 'A' : 'a';
            // Calculamos nueva posición con desplazamiento (sumamos 26 para evitar negativos)
            int indice = (c - base + desplazamiento + 26) % 26;
            // Convertimos índice a caracter cifrado
            resultado.append((char) (base + indice));
         } else {
```

```
            // Caracteres no letras quedan igual
            resultado.append(c);
        }
    }
    return resultado.toString();
}

/**
 * Cifra el contenido de un archivo y lo guarda en otro archivo.
 * @param origen Archivo original a cifrar.
 * @param destino Archivo donde se guardará el cifrado.
 * @param desplazamiento Clave de cifrado (desplazamiento César).
 * @throws IOException Si hay errores de lectura o escritura.
 */
public static void cifrarArchivo(File origen, File destino, int desplazamiento) throws IOException {
    // Usamos try-with-resources para cerrar automáticamente lectores/escritores
    try (BufferedReader lector = new BufferedReader(new FileReader(origen));
         BufferedWriter escritor = new BufferedWriter(new FileWriter(destino))) {
        String linea;
        // Leemos línea por línea, ciframos y escribimos
        while ((linea = lector.readLine()) != null) {
            escritor.write(cifradoCesar(linea, desplazamiento));
            escritor.newLine(); // saltos de línea para preservar estructura original
        }
    }
}

/**
 * Descifra un archivo cifrado con César aplicando el desplazamiento inverso.
```

```
     * @param origen Archivo cifrado.
     * @param destino Archivo donde se guardará el texto descifrado.
     * @param desplazamiento Clave original usada para cifrar.
     * @throws IOException Si hay errores de lectura o escritura.
     */
    public static void descifrarArchivo(File origen, File destino, int desplazamiento) throws IOException {
        // Descifrar es cifrar con desplazamiento negativo
        cifrarArchivo(origen, destino, -desplazamiento);
    }
}



import javax.swing.*;
import java.io.*;
import java.util.List;

/**
 * JPanel para exportar datos de estructuras y permitir cifrar/descifrar archivos o carpetas completas.
 */
public class EportarVentana extends javax.swing.JPanel {

    // Variables que guardan las estructuras de datos (árboles, listas, etc)
    private ArbolVehiculos arbolVehiculos;
    private ArbolAVL arbolAVL;
    private ListaDobleMultas listaMultas;
    private ListaCircularTraspasos listasTraspasos;

    // Componentes gráficos (checkbox para seleccionar qué exportar, botones para acciones)
```

```
    private JCheckBox jCheckBox1ABB;
    private JCheckBox jCheckBox2AVL;
    private JCheckBox jCheckBox3ListaDoble;
    private JCheckBox jCheckBox4ListaCircular;
    private JCheckBox jCheckBox5Carpeta; // Checkbox para cifrar carpeta completa
    private JTextField jTextFieldClave;  // Campo para ingresar clave de cifrado
    private JButton jButtonGuardar;      // Botón para guardar texto plano sin cifrar
    private JButton jButtonCifrar;       // Botón para cifrar datos o carpeta
    private JButton jButtonDescifrar;    // Botón para descifrar archivo

    /**
     * Constructor del panel.
     * Aquí se inicializan las estructuras y componentes gráficos.
     */
    public EportarVentana() {
        initComponents(); // Método generado (no mostrado aquí) que inicializa los componentes Swing

        // Obtenemos las estructuras globales para exportar
        this.arbolVehiculos = GestorDatosGlobal.getVehiculos();
        this.arbolAVL = GestorDatosGlobal.getArbolAVL();
        this.listaMultas = GestorDatosGlobal.getMultas();
        this.listasTraspasos = GestorDatosGlobal.getTraspasos();

        // Agregamos los botones al panel (si no están en initComponents)
        add(jButtonGuardar);
        add(jButtonCifrar);
        add(jButtonDescifrar);
    }
```

```
/**
 * Método para obtener el panel principal (si se necesita usarlo desde otra clase)
 */
public JPanel getventana(){
   return this;  // Se devuelve el panel mismo o un subpanel si existe
}

/**
 * Acción al presionar botón DESCIFRAR
 * Se solicita la clave, se abre diálogo para seleccionar archivo cifrado
 * y luego se guarda el archivo descifrado.
 */
private void jButtonDescifrarActionPerformed(java.awt.event.ActionEvent evt) {
   try {
      int clave = obtenerClave(); // Obtenemos clave numérica desde campo de texto
      descifrarArchivoSeleccionado(clave); // Método que maneja selector y descifrado
   } catch (NumberFormatException ex) {
      // Si la clave no es un número válido mostramos error
      JOptionPane.showMessageDialog(this,
         "Clave inválida: " + ex.getMessage(),
         "Error", JOptionPane.ERROR_MESSAGE);
   } catch (IOException io) {
      // Si hay problemas con archivos también mostramos error
      JOptionPane.showMessageDialog(this,
         "Error al descifrar: " + io.getMessage(),
         "Error", JOptionPane.ERROR_MESSAGE);
   }
}
```

```
  /**
   * Método que abre un JFileChooser para seleccionar archivo cifrado y otro para guardar descifrado.
   * @param clave Clave para aplicar descifrado César.
   * @throws IOException Si hay error en lectura o escritura.
   */
  private void descifrarArchivoSeleccionado(int clave) throws IOException {
    JFileChooser selector = new JFileChooser();
    selector.setDialogTitle("Seleccionar archivo cifrado");
    if (selector.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {
      File origen = selector.getSelectedFile();

      JFileChooser saveSel = new JFileChooser();
      saveSel.setDialogTitle("Guardar archivo descifrado");
      if (saveSel.showSaveDialog(this) == JFileChooser.APPROVE_OPTION) {
        File destino = saveSel.getSelectedFile();
        UtilidadesCifradoCesar.descifrarArchivo(origen, destino, clave);

        JOptionPane.showMessageDialog(this,
          "Datos descifrados en: " + destino.getAbsolutePath());
      }
    }
  }

  /**
   * Acción al presionar botón CIFRAR.
   * Si la opción "Cifrar carpeta completa" está activada, cifra todos los archivos
   * en la carpeta global de datos, sino cifra solo las estructuras seleccionadas.
```

```
 */
private void jButtonCifrarActionPerformed(java.awt.event.ActionEvent evt) {
   try {
      int clave = obtenerClave(); // Obtenemos clave de texto ingresada


      if (jCheckBox5Carpeta.isSelected()) {
         cifrarCarpetaCompleta(clave);
      } else {
         cifrarEstructurasSeleccionadas(clave);
      }
   } catch (NumberFormatException ex) {
      JOptionPane.showMessageDialog(this,
         "Clave inválida: " + ex.getMessage(),
         "Error", JOptionPane.ERROR_MESSAGE);
   } catch (IOException io) {
      JOptionPane.showMessageDialog(this,
         "Error al cifrar: " + io.getMessage(),
         "Error", JOptionPane.ERROR_MESSAGE);
   }
}

/**
 * Cifra todos los archivos de la carpeta global de datos y los guarda
 * en otra carpeta seleccionada por el usuario.
 * @param clave Clave para cifrado César.
 * @throws IOException Si hay errores de archivos.
 */
private void cifrarCarpetaCompleta(int clave) throws IOException {
   JFileChooser selector = new JFileChooser(GestorDatosGlobal.getCarpetaDatos());
```

```
    selector.setDialogTitle("Seleccionar carpeta destino");
    selector.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY); // Solo directorios

    if (selector.showSaveDialog(this) == JFileChooser.APPROVE_OPTION) {
        File destino = selector.getSelectedFile();

        // Recorremos todos los archivos de la carpeta original
        for (File f : GestorDatosGlobal.getCarpetaDatos().listFiles()) {
            if (f.isFile()) {
                // Creamos archivo destino con mismo nombre en carpeta destino
                File out = new File(destino, f.getName());
                // Ciframos archivo origen y lo guardamos en destino
                UtilidadesCifradoCesar.cifrarArchivo(f, out, clave);
            }
        }

        JOptionPane.showMessageDialog(this,
            "Carpeta cifrada en: " + destino.getAbsolutePath());
    }
}

/**
 * Lee la clave de texto desde el campo de texto y la convierte a int.
 * @return Clave numérica para cifrado.
 * @throws NumberFormatException Si la clave está vacía o no es un número válido.
 */
private int obtenerClave() throws NumberFormatException {
    String txt = jTextFieldClave.getText().trim();
    System.out.println("Clave escrita: " + txt);
```

```
        if (txt.isEmpty()) {
            throw new NumberFormatException("Debe ingresar la clave");
        }
        return Integer.parseInt(txt);
    }

    /**
     * Cifra y guarda en un archivo el contenido de las estructuras seleccionadas por el usuario.
     * @param clave Clave para cifrado César.
     * @throws IOException Si hay errores en archivos.
     */
    private void cifrarEstructurasSeleccionadas(int clave) throws IOException {
        StringBuilder sb = new StringBuilder();

        // Recorremos cada estructura según el checkbox marcado y agregamos su contenido a un StringBuilder
        if (jCheckBox1ABB.isSelected()) {
            for (Vehiculo v : arbolVehiculos.inOrderList()) sb.append(v).append("\n");
        }
        if (jCheckBox2AVL.isSelected()) {
            for (Vehiculo v : arbolAVL.inOrderList()) sb.append(v).append("\n");
        }
        if (jCheckBox3ListaDoble.isSelected()) {
            for (Multa m : listaMultas.toList()) sb.append(m).append("\n");
        }
        if (jCheckBox4ListaCircular.isSelected()) {
            for (Traspaso t : listasTraspasos.toList()) sb.append(t).append("\n");
        }
```

```
    // Abrimos diálogo para guardar el archivo cifrado
    JFileChooser selector = new JFileChooser();
    selector.setDialogTitle("Guardar archivo cifrado");
    if (selector.showSaveDialog(this) == JFileChooser.APPROVE_OPTION) {
        File archivo = selector.getSelectedFile();

        // Creamos archivo temporal con texto plano sin cifrar
        File temp = File.createTempFile("data_plain", ".txt");
        try (BufferedWriter bw = new BufferedWriter(new FileWriter(temp))) {
            bw.write(sb.toString());
        }

        // Aplicamos cifrado al archivo temporal y guardamos resultado en archivo seleccionado
        UtilidadesCifradoCesar.cifrarArchivo(temp, archivo, clave);

        // Eliminamos archivo temporal
        temp.delete();

        JOptionPane.showMessageDialog(this,
            "Datos guardados en: " + archivo.getAbsolutePath());
    }
}

// Métodos vacíos para checkbox, normalmente aquí se pueden poner acciones si se desea
private void jCheckBox1ABBActionPerformed(java.awt.event.ActionEvent evt) { }
private void jCheckBox2AVLActionPerformed(java.awt.event.ActionEvent evt) { }
private void jCheckBox3ListaDobleActionPerformed(java.awt.event.ActionEvent evt) { }

/**
```

```
 * Acción para botón GUARDAR (sin cifrar).
 * Guarda en un archivo el texto plano de las estructuras seleccionadas.
 */
private void jButtonGuardarActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        StringBuilder sb = new StringBuilder();

        // Recorremos las estructuras según los checkbox marcados
        if (jCheckBox1ABB.isSelected()) {
            for (Vehiculo v : arbolVehiculos.inOrderList()) {
                sb.append(v).append("\n");
            }
        }
        if (jCheckBox2AVL.isSelected()) {
            for (Vehiculo v : arbolAVL.inOrderList()) {
                sb.append(v).append("\n");
            }
        }
        if (jCheckBox3ListaDoble.isSelected()) {
            for (Multa m : listaMultas.toList()) {
                sb.append(m).append("\n");
            }
        }
        if (jCheckBox4ListaCircular.isSelected()) {
            for (Traspaso t : listasTraspasos.toList()) {
                sb.append(t).append("\n");
            }
        }
```

```
            // Abrimos diálogo para guardar archivo de texto plano
            JFileChooser chooser = new JFileChooser();
            chooser.setDialogTitle("Guardar texto plano");
            if (chooser.showSaveDialog(this) == JFileChooser.APPROVE_OPTION) {
                File destino = chooser.getSelectedFile();

                try (BufferedWriter bw = new BufferedWriter(new FileWriter(destino))) {
                    bw.write(sb.toString());
                }

                JOptionPane.showMessageDialog(this,
                    "Texto plano guardado en: " + destino.getAbsolutePath());
            }
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(this,
                "Error al guardar: " + ex.getMessage());
        }
    }

    private void jTextFieldClaveActionPerformed(java.awt.event.ActionEvent evt) {
        // Acción para el campo de texto clave (opcional)
    }
}
```