

**Universidad Internacional del Ecuador**

Carrera: Ingeniería en Software Modalidad En Línea

Materia: PROGRAMACION ESTRUCTURADA Y FUNCIONAL

Estudiante: SALAZAR CADENA JHONNATAN FRANCISCO  
Fecha: 27 de junio del 2025

# Evaluación en Contacto con el Docente

# Proyecto Sistema de Peaje

Docente: PALACIOS MOROCHO MILTON RICARDO

Tabla de contenido

[Evaluación en Contacto con el Docente 1](#_Toc202065485)

[Proyecto Sistema de Peaje 1](#_Toc202065486)

[Enunciado 3](#_Toc202065487)

[Considere el siguiente diagrama de clases: 3](#_Toc202065488)

[Introducción 3](#_Toc202065489)

[Descripción y funcionamiento de clases 4](#_Toc202065490)

[Clase: Peaje 4](#_Toc202065491)

[Clase Padre: Vehículo 7](#_Toc202065492)

[Sub-Clase: Carro 8](#_Toc202065493)

[Sub-Clase: Moto 9](#_Toc202065494)

[Sub-Clase: Camion 11](#_Toc202065495)

[Main Class 12](#_Toc202065496)

[Ejecución del programa 19](#_Toc202065497)

[Conclusiones 21](#_Toc202065498)

[Link del Proyecto 21](#_Toc202065499)

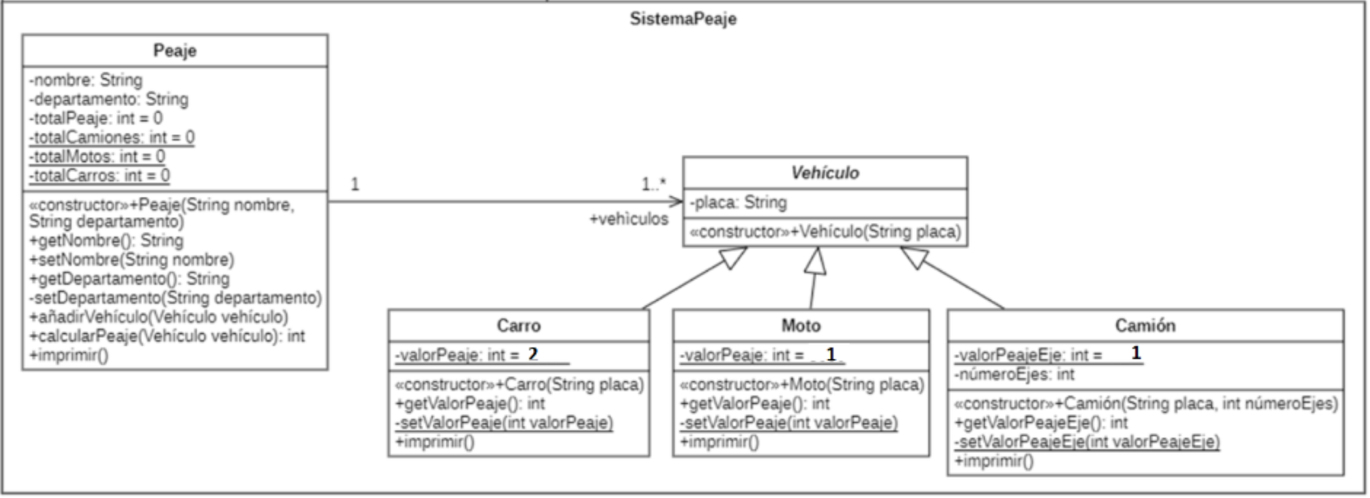
# **Enunciado**

Se requiere desarrollar un programa que modele un sistema de peaje. Las estaciones de peaje tienen un nombre, departamento o cantón en que están ubicadas y un valor numérico que representa el valor total de peaje recolectado. Los vehículos que llegan a un peaje tienen una placa (tipo *String*).

El peaje cobra diferentes valores de peaje según el tipo de vehículo. Si es un carro, el valor del peaje es de $2. Si es una moto, $1. Si es un camión, el valor del peaje depende del número de ejes, se cobra $1 por cada eje.

Se requiere que la estación de peaje calcule el valor del peaje de cada vehículo que llegue y el total de peajes recolectados. Así, al finalizar, el sistema debe *imprimir*en pantalla un listado con los vehículos que llegaron al  
peaje y el total acumulado.

## **Considere el siguiente diagrama de clases:**



# Introducción

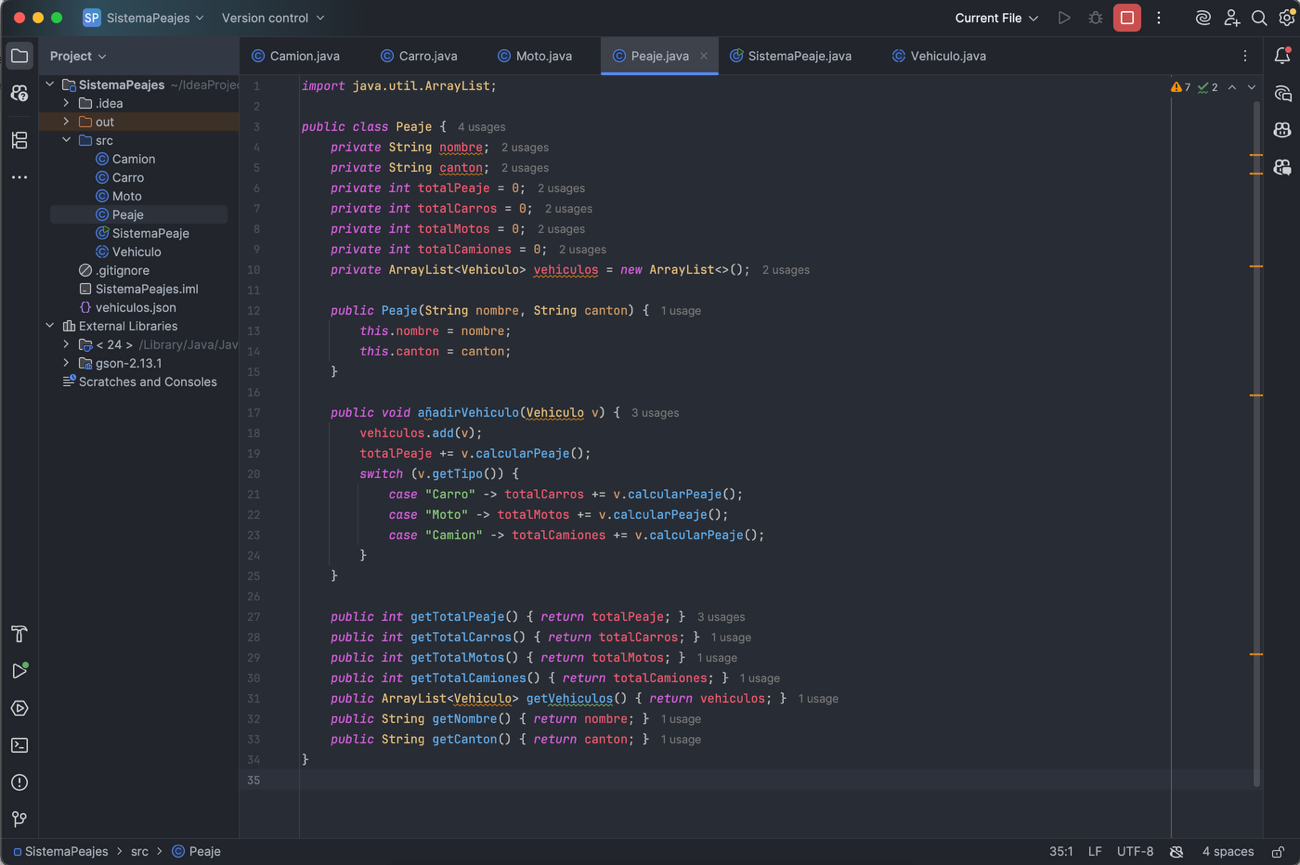
El presente documento describe el desarrollo de una aplicación de escritorio en Java que simula el funcionamiento de un sistema de control de peajes. El objetivo principal es registrar y calcular el valor de peaje generado por distintos tipos de vehículos (carros, motos y camiones) en diferentes estaciones distribuidas en cantones del Ecuador.

La aplicación cuenta con una interfaz gráfica amigable construida con Swing, que permite seleccionar una estación de peaje, ingresar datos de los vehículos y visualizar el total de dinero recolectado, incluyendo un desglose por tipo de vehículo. Además, se implementa persistencia de datos mediante la librería GSON, lo que permite guardar la información en formato JSON para su posterior recuperación.

Este sistema no solo busca automatizar el proceso de control en estaciones de peaje, sino también ofrecer una solución didáctica y funcional que demuestre la aplicación de conceptos de programación orientada a objetos, manejo de eventos gráficos y almacenamiento de datos estructurados.

# Descripción y funcionamiento de clases

## Clase: Peaje



La clase Peaje sirve para registrar los vehículos que pasan, calcula automáticamente el total recaudado en cada peaje al igual que separa el total recaudado por tipo de vehículo y consulta la información registrada.

import java.util.ArrayList;

Nos permite crear listas dinámicas, en este caso se usa para guardar todos los vehículos que pasan por el peaje.

public class Peaje {

private String nombre;

private String canton;

private int totalPeaje = 0;

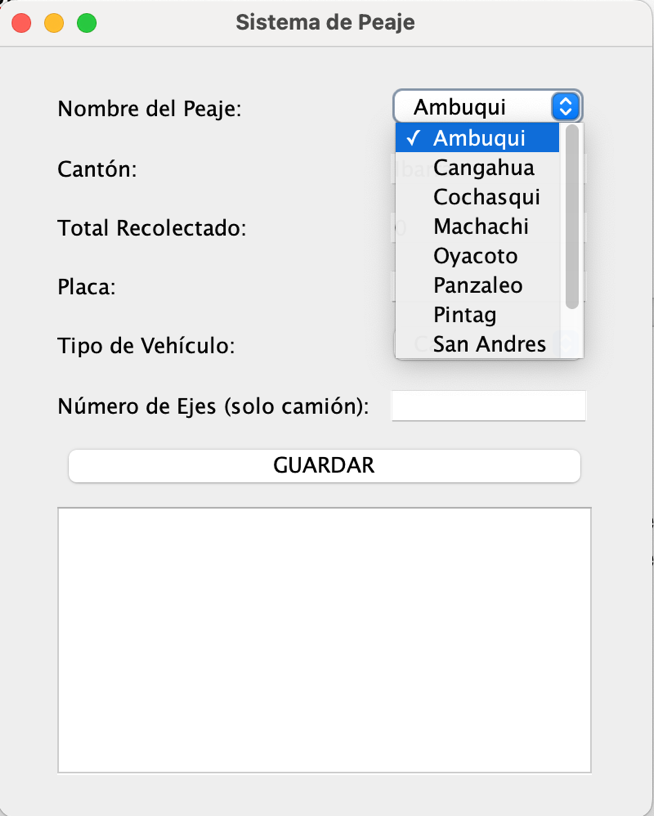
private int totalCarros = 0;

private int totalMotos = 0;

private int totalCamiones = 0;

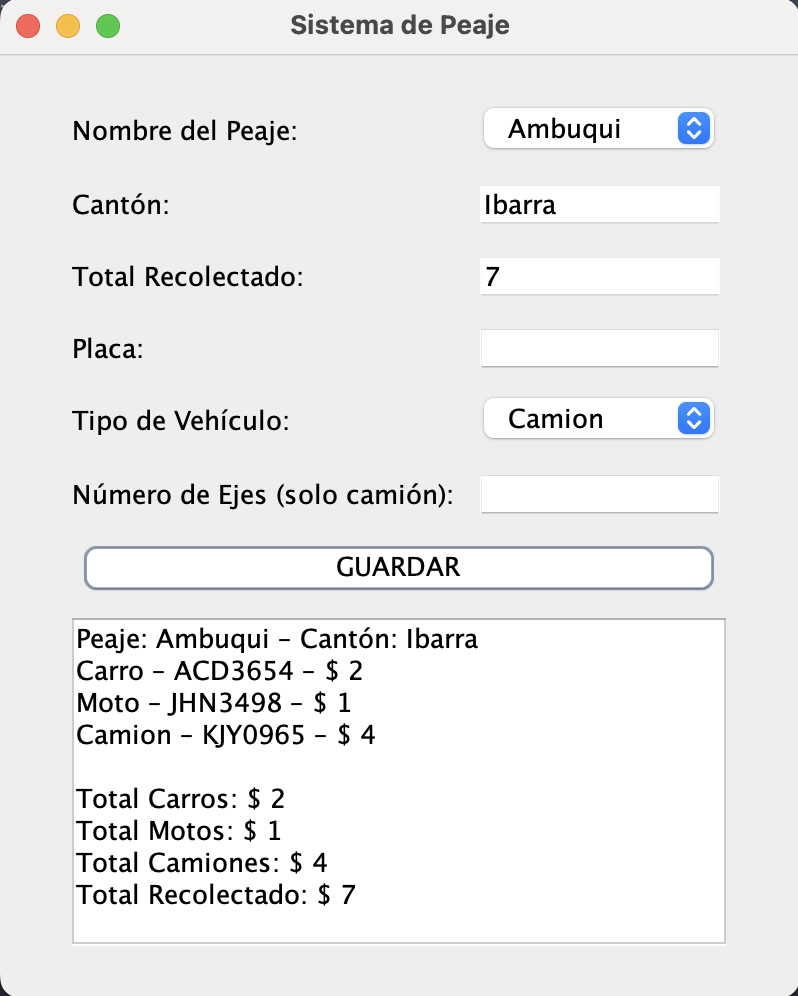
private ArrayList<Vehiculo> vehiculos = new ArrayList<>();

Representa una estación de cobro de peaje donde se registran vehículos y se calcula lo recaudado, son los atributos los cuales nos sirven para identificar cada peaje por ejemplo:



En este caso se utilizó un listado de peajes de Panavial el cual muestra el nombre de peaje y cada peaje está relacionado con un cantón el cual también cambia según el nombre de peaje seleccionado.

De la misma manera se nos muestra el total recaudado y el total por tipo de vehículo por ejemplo:



public Peaje(String nombre, String canton) {

this.nombre = nombre;

this.canton = canton;

}

Es el constructor de la clase, este método especial se ejecuta cuando se crea un nuevo objeto Peaje y recibe como parámetros el nombre y el cantón y los asigna a los atributos correspondientes usando this.

public void añadirVehiculo(Vehiculo v) {

vehiculos.add(v);

totalPeaje += v.calcularPeaje();

switch (v.getTipo()) {

case "Carro" -> totalCarros += v.calcularPeaje();

case "Moto" -> totalMotos += v.calcularPeaje();

case "Camion" -> totalCamiones += v.calcularPeaje();

}

}

Este método se utiliza para agregar el vehículo de la lista y totalPeaje += v.calcularPeaje(); suma el peaje del vehículo al total general, el resto del código se encarga de sumar el total de cada tipo de vehículo.

public int getTotalPeaje() { return totalPeaje; }

public int getTotalCarros() { return totalCarros; }

public int getTotalMotos() { return totalMotos; }

public int getTotalCamiones() { return totalCamiones; }

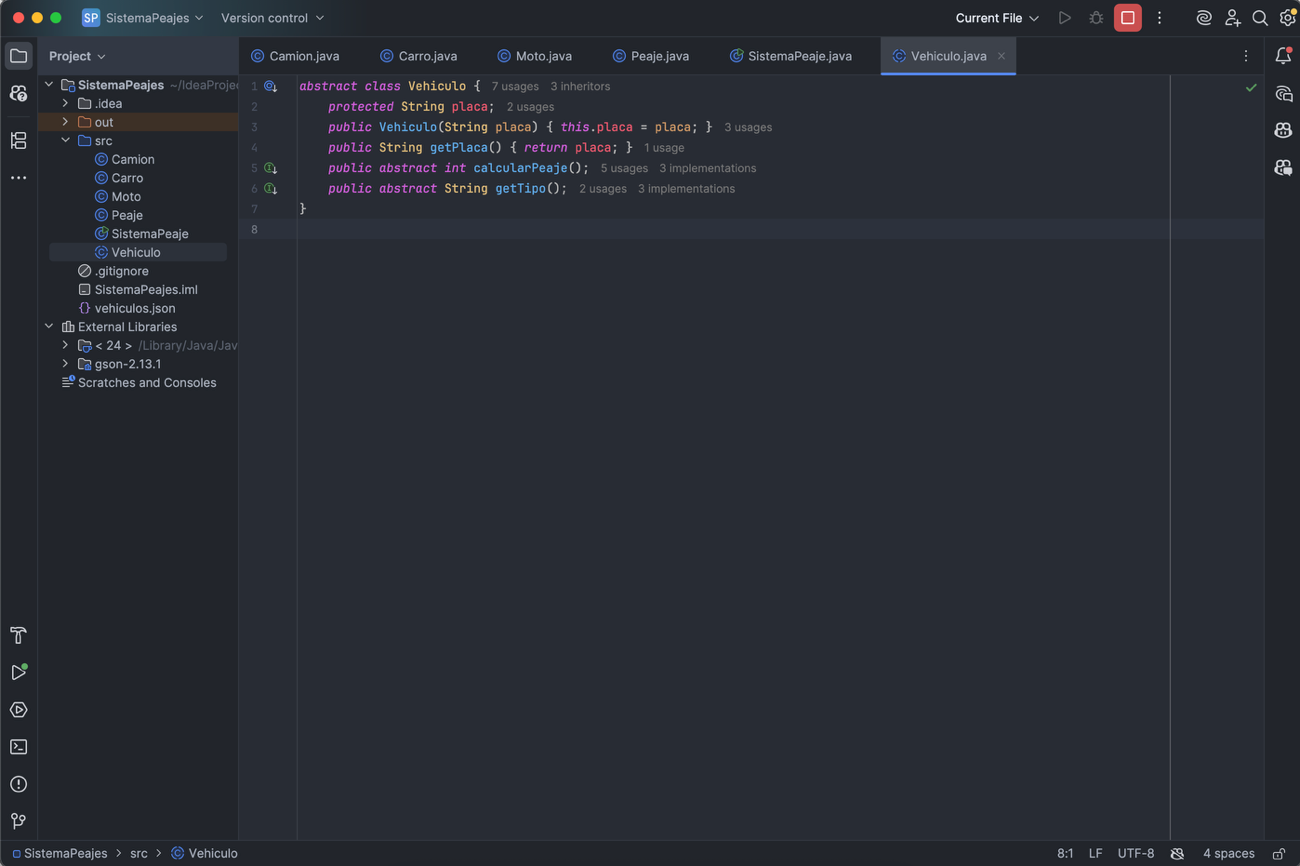
public ArrayList<Vehiculo> getVehiculos() { return vehiculos; }

public String getNombre() { return nombre; }

public String getCanton() { return canton; }

El método get permite obtener el valor de atributos privados desde fuera de la clase cada uno retorna el valor correspondiente de la variable de instancia.

## Clase Padre: Vehículo



La clase abstracta Vehículo establece la estructura común para todos los tipos de vehículos en el sistema de peaje. Define el atributo placa y métodos esenciales que cada subclase debe implementar. Esto permite que el sistema trabaje con todos los vehículos de manera genérica y polimórfica, facilitando la extensibilidad y el mantenimiento del código.

abstract class Vehiculo {

Al crear una clase abstracta significa que no se puede crear un objeto directamente de eta clase solo puede ser heredada por las clases hijas en este caso por las clases Carro, Moto, Camion.

protected String placa;

Este atributo llamado placa junto con la palabra clave protected permite que este atributo sea accedido directamente desde las clases hijas.

public Vehiculo(String placa) {

this.placa = placa;

}  
Define el constructor que inicializa el atributo placa con el valor recibido como argumento.  
El uso de this.placa asegura que se está asignando al atributo del objeto y no al parámetro local.

public String getPlaca() {

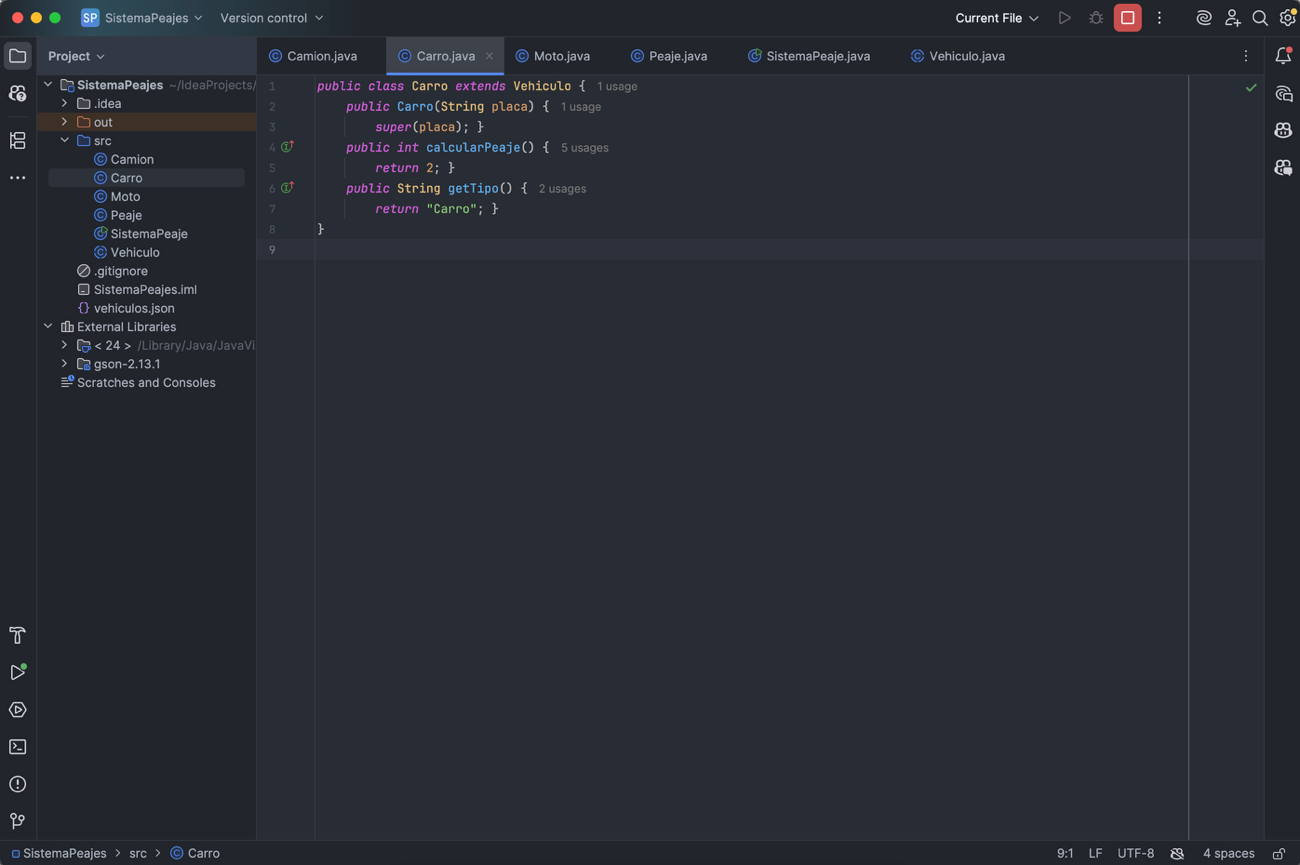
return placa;

}  
El método get devuelve la placa del vehículo desde cualquier parte del programa para consultar la placa.

public abstract int calcularPeaje();

public abstract String getTipo();  
Son métodos abstractos y obligan a todas las clases que heredan de Vehiculo a implementar su propia versión de estos métodos, el calcularPeaje debe retornar el valor del peaje para este tipo de vehículo, mientras que getTipo debe devolver un texto que identifique el tipo de vehículo por ejemplo Carro, Moto, Camion.

### Sub-Clase: Carro



La clase Carro representa un tipo específico de vehículo dentro del sistema de peajes. Implementa los métodos necesarios para calcular su peaje y devolver su tipo. Gracias a que extiende la clase Vehiculo, puede ser tratada de forma genérica por el sistema, lo que permite aprovechar los principios de **herencia y polimorfismo** en Java.

public class Carro extends Vehiculo {

Declara una clase pública llamada Carro que **hereda** de la clase abstracta Vehiculo. Esto significa que Carro es un subtipo de Vehiculo y está obligada a implementar los métodos abstractos definidos en la clase base.

public Carro(String placa) {

super(placa);

}

Define el constructor de la clase Carro y recibe como parámetro la placa del vehículo, utiliza la instrucción super(placa) para llamar al consytructor de la clase Vehiculo el cual se encarga de inicializar el atributo placa.

public int calcularPeaje() {

return 2;

}  
Implementa el método abstracto heredado de Vehiculo.  
En este caso, retorna el valor **2**, que representa el costo del peaje para un carro.  
Este valor puede ser usado por el sistema para calcular el total recaudado.

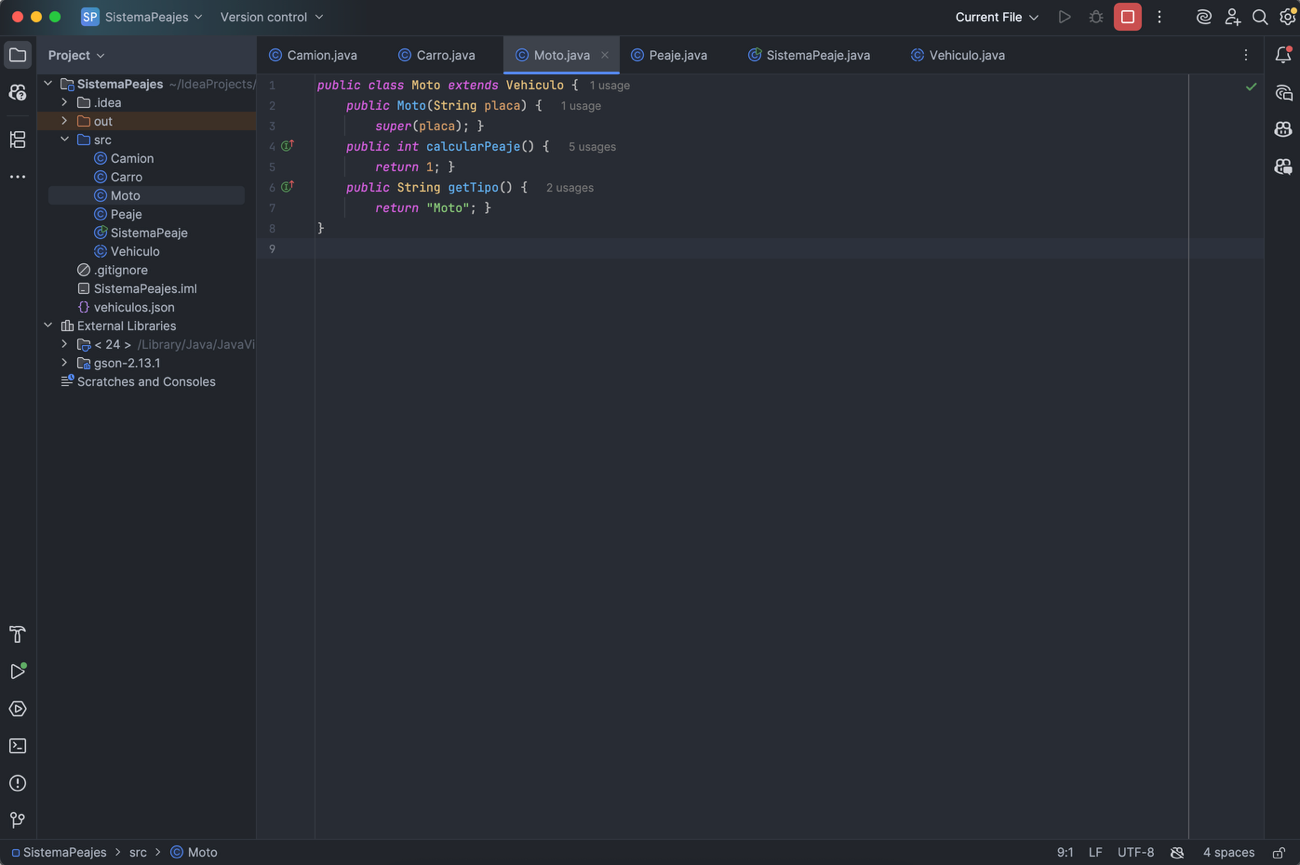
public String getTipo() {

return "Carro";

}

Devuelve el tipo del vehículo como una cadena de texto, en este caso "Carro".  
Este valor es importante para clasificar los vehículos y calcular los totales por tipo en la clase Peaje.

### Sub-Clase: Moto



La clase Moto representa un tipo de vehículo concreto dentro del sistema de peajes. Hereda el atributo placa y cumple con los requisitos establecidos por la clase abstracta Vehiculo, implementando su propio costo de peaje y nombre de tipo. Esta implementación facilita la reutilización del código y permite que el sistema trate todos los vehículos de forma uniforme utilizando **polimorfismo.**

public class Moto extends Vehiculo {

Declara una clase pública llamada Moto que **hereda de la clase abstracta Vehiculo**. Esto significa que es un tipo de vehículo específico y debe implementar los métodos abstractos definidos en la clase padre.

public Moto(String placa) {

super(placa);

}

Este constructor recibe la **placa del vehículo** como parámetro.  
Mediante super(placa) se llama al constructor de la clase Vehiculo, que asigna ese valor al atributo placa.

public int calcularPeaje() {

return 1;

}

Devuelve el valor del peaje que debe pagar una moto, en este caso $**1**.  
Este valor se usará por la clase Peaje para sumar correctamente los montos recaudados.

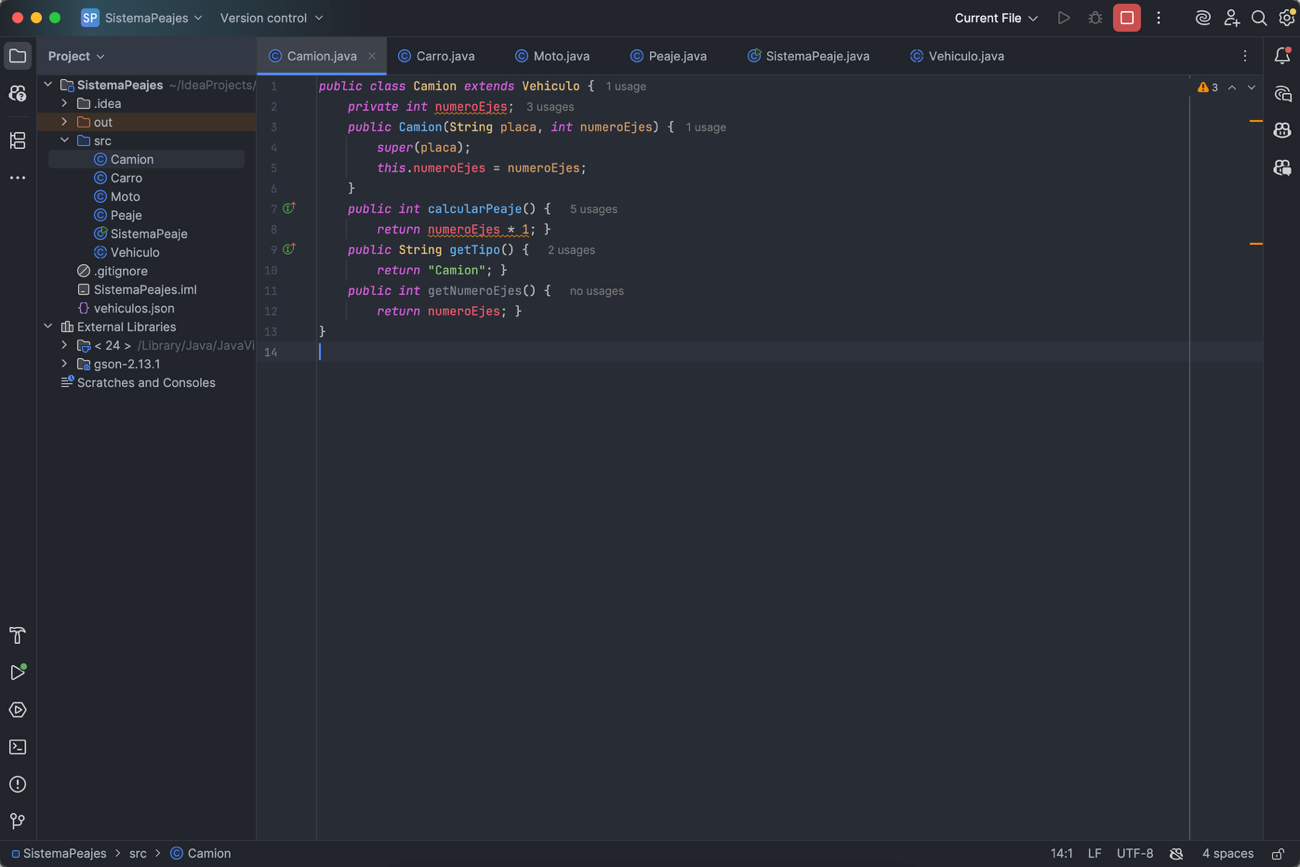
public String getTipo() {

return "Moto";

}

Devuelve el tipo de vehículo como una cadena de texto: "Moto".  
Esto permite identificarlo en el sistema, especialmente al separar el total por tipo de vehículo.

### Sub-Clase: Camion



La clase Camion es una implementación concreta de la clase abstracta Vehiculo y añade un atributo único (numeroEjes) que influye directamente en el cálculo del peaje. Esta clase demuestra cómo se puede extender una estructura genérica para adaptarla a necesidades específicas, aplicando principios de **herencia, polimorfismo** y **encapsulamiento.**

public class Camion extends Vehiculo {

Declara la clase pública Camion, que **hereda de la clase abstracta** Vehiculo.  
Representa un tipo específico de vehículo que paga su peaje en función del número de ejes que posee.

private int numeroEjes;

Define un atributo exclusivo de los camiones llamado numeroEjes.  
Este valor indica cuántos ejes tiene el camión y se utilizará para calcular el valor del peaje.

public Camion(String placa, int numeroEjes) {

super(placa);

this.numeroEjes = numeroEjes;

}

Este constructor inicializa el objeto Camion con la placa y el número de ejes, con super(placa) llama al constructor de la clase Vehiculo para registrar la placa, y con numeroEjes asigna el valor del número de ejes al atributo propio de la clase.

public int calcularPeaje() {

return numeroEjes \* 1;

}  
Calcula el valor del peaje que debe pagar el camión multiplicando la cantidad de ejes por 1. Por ejemplo, un camión con 3 ejes pagará $3.

public String getTipo() {

return "Camion";

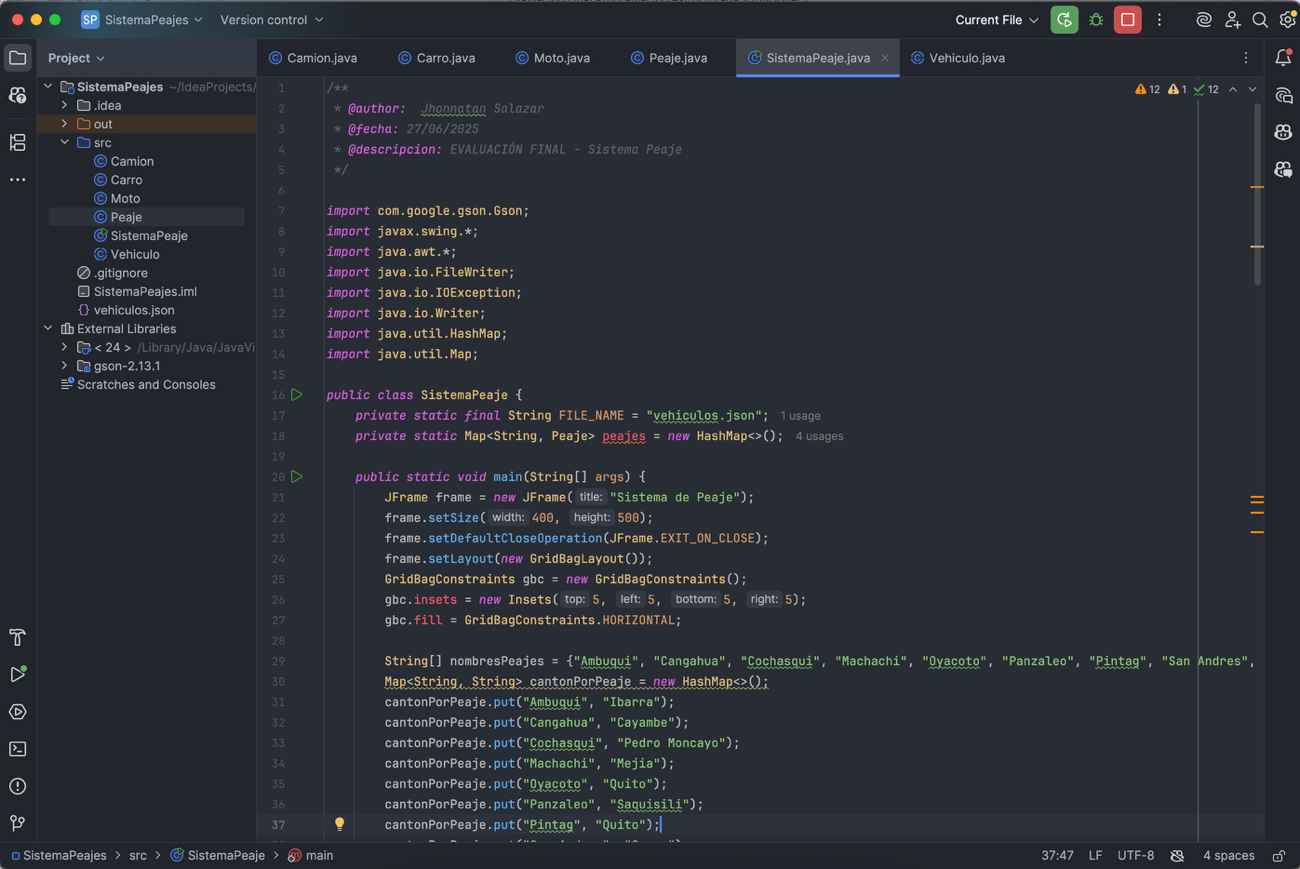
}

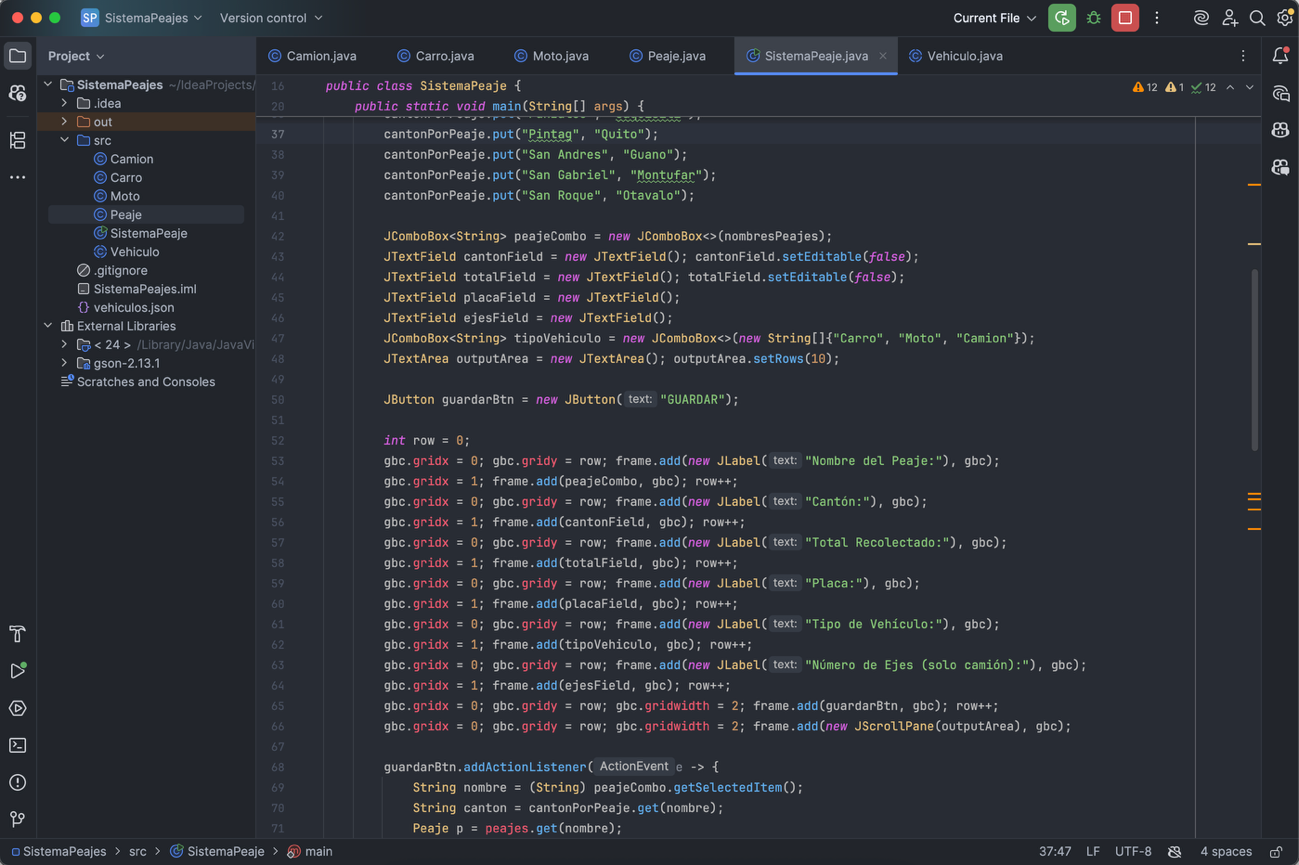
Devuelve el tipo de vehículo como "Camion", lo cual permite su identificación y clasificación en el sistema de peaje.

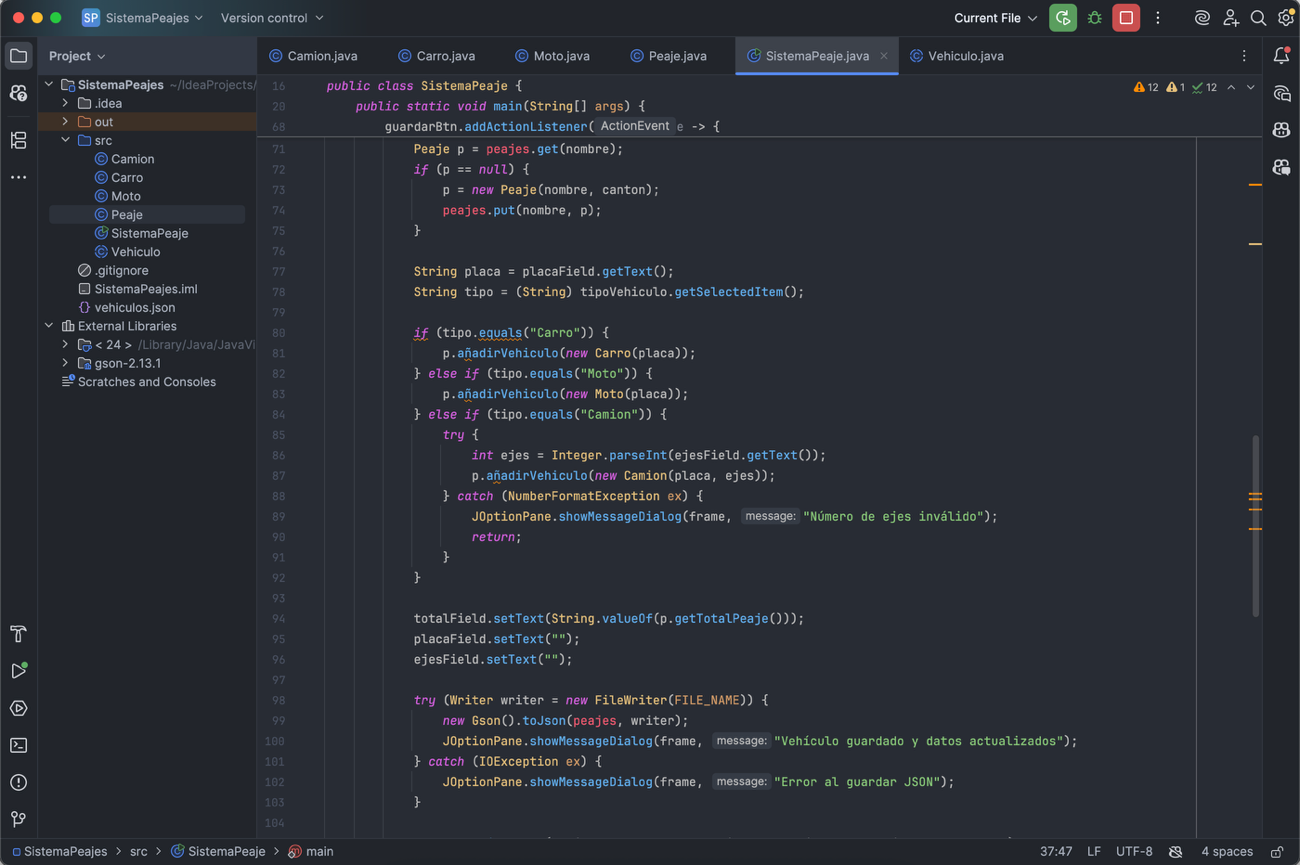
public int getNumeroEjes() {

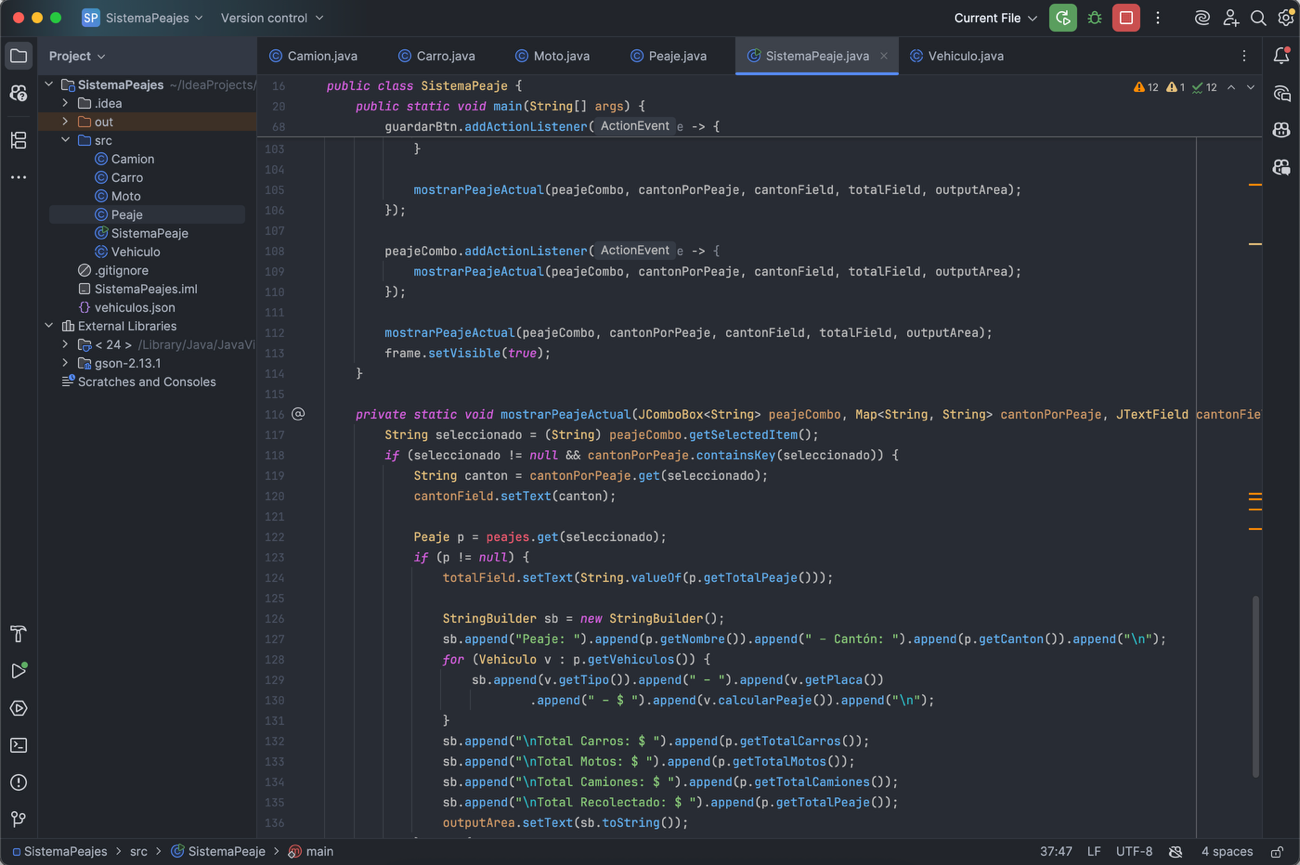
return numeroEjes;

}  
Permite acceder al número de ejes del camión desde fuera de la clase. Es útil si se requiere mostrar o validar este dato en la interfaz del sistema.

Main Class  
  










El código SistemaPeaje.java es una aplicación Java con interfaz gráfica construida con Swing que permite gestionar el registro de vehículos en estaciones de peaje. Entre sus funcionalidades principales se encuentran: la selección de un peaje y la visualización automática de su cantón, el registro de vehículos mediante el ingreso de su tipo, placa y, en el caso de camiones, el número de ejes; el cálculo automático del peaje según el tipo de vehículo; la visualización del historial de vehículos registrados por peaje; y el desglose de los montos recaudados por tipo de vehículo y del total general. Además, toda la información es guardada en un archivo JSON (vehiculos.json) utilizando la librería Gson, permitiendo la persistencia de datos. El sistema está diseñado aplicando los principios de programación orientada a objetos, integrando interfaces gráficas y mecanismos de almacenamiento de forma eficiente y estructurada.

import com.google.gson.Gson;

import javax.swing.\*;

import java.awt.\*;

import java.io.FileWriter;

import java.io.IOException;

import java.io.Writer;

import java.util.HashMap;

import java.util.Map;

Con la importación de algunas librerías podemos construir la interfaz gráfica utilizando swing ya que nos permite implementar diseños de ventanas y botones en los cuales podemos introducir texto y almacenarlos en formato Json, en la caso de Map utilizamos para acceder a los cantones definidos para cada nombre de peaje asociado con el mismo.

public class SistemaPeaje {

private static final String FILE\_NAME = "vehiculos.json";

private static Map<String, Peaje> peajes = new HashMap<>();

Esta clase principal administra la lógica del sistema utilizando FILE\_NAME nombre del archivo JSON donde guarda los datos de los peajes, en el caso de Map<String, Peaje> peajes, asocia el nombre del peaje permitiendo consultar y actualizar los datos por peaje.

public static void main(String[] args) {

Método principal que ejecuta la aplicación. Aquí se configura y muestra la interfaz gráfica completa.

JFrame frame = new JFrame("Sistema de Peaje");

frame.setSize(400, 500);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setLayout(new GridBagLayout());

Es la configuración de la ventana principal del sistema definiendo su tamaño (400x500). Usa GridBagLayout para alinear elementos en filas y columnas y uyiliza EXIT\_ON\_CLOSE para cerrar el programa cuendo cierra la ventana.

GridBagConstraints gbc = new GridBagConstraints();

gbc.insets = new Insets(5, 5, 5, 5);

gbc.fill = GridBagConstraints.HORIZONTAL;

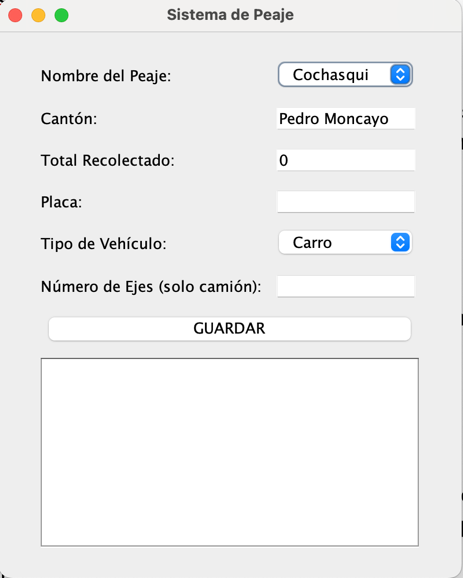
Controla con colocación y el espacio entre componentes dando un margen de 5 píxeles entra cada componente.

String[] nombresPeajes = {...};

Map<String, String> cantonPorPeaje = new HashMap<>();

// cantonPorPeaje.put(...) para cada uno

Se define los nombres de los peajes disponibles en este caso se tomó como ejemplo a todos los nombres de los peajes de Panavial para crear el programa y utilizando Map<String, String> cantonPorPeaje mapea cada peaje a su cantón correspondiente como Ambuqui a Ibarra ejemplo:



El peaje de Cochasqui se encuentra ubicado en el cantón de Pedro Moncayo, al seleccionar cualquier nombre de la lista de peajes el cantón se actualiza automáticamente para cada peaje de la lista.

JComboBox<String> peajeCombo = new JComboBox<>(nombresPeajes);

JTextField cantonField = new JTextField(); cantonField.setEditable(false);

JTextField totalField = new JTextField(); totalField.setEditable(false);

JTextField placaField = new JTextField();

JTextField ejesField = new JTextField();

JComboBox<String> tipoVehiculo = new JComboBox<>(new String[]{"Carro", "Moto", "Camion"});

JTextArea outputArea = new JTextArea(); outputArea.setRows(10);

JButton guardarBtn = new JButton("GUARDAR");

* **peajeCombo**: selector desplegable de peajes.
* **cantonField**: muestra el cantón correspondiente al peaje (no editable).
* **totalField**: muestra el total recaudado en el peaje (no editable).
* **placaField**: entrada para escribir la placa del vehículo.
* **ejesField**: entrada para escribir el número de ejes (solo para camiones).
* **tipoVehiculo**: selector del tipo de vehículo.
* **outputArea**: área para mostrar detalles de los vehículos y totales.
* **guardarBtn**: botón que guarda un nuevo vehículo.

gbc.gridx = 0; gbc.gridy = row;

frame.add(new JLabel("Nombre del Peaje:"), gbc);

// luego gbc.gridx = 1; frame.add(componente correspondiente);

Cada componente es añadido fila por fila esto se repite para cada etiqueta y su campo respectivo nos da un orden para poder identificar.

guardarBtn.addActionListener(e -> {

String nombre = (String) peajeCombo.getSelectedItem();

String canton = cantonPorPeaje.get(nombre);

Peaje p = peajes.get(nombre);

if (p == null) {

p = new Peaje(nombre, canton);

peajes.put(nombre, p);

}

String placa = placaField.getText();

String tipo = (String) tipoVehiculo.getSelectedItem();

if (tipo.equals("Carro")) {

p.añadirVehiculo(new Carro(placa));

} else if (tipo.equals("Moto")) {

p.añadirVehiculo(new Moto(placa));

} else if (tipo.equals("Camion")) {

try {

int ejes = Integer.parseInt(ejesField.getText());

p.añadirVehiculo(new Camion(placa, ejes));

} catch (NumberFormatException ex) {

JOptionPane.showMessageDialog(frame, "Número de ejes inválido");

return;

}

}

totalField.setText(String.valueOf(p.getTotalPeaje()));

placaField.setText("");

ejesField.setText("");

try (Writer writer = new FileWriter(FILE\_NAME)) {

new Gson().toJson(peajes, writer);

JOptionPane.showMessageDialog(frame, "Vehículo guardado y datos actualizados");

} catch (IOException ex) {

JOptionPane.showMessageDialog(frame, "Error al guardar JSON");

}

mostrarPeajeActual(peajeCombo, cantonPorPeaje, cantonField, totalField, outputArea);

});

Cuando se presiona el botón, el programa:

1. Obtiene el nombre del peaje seleccionado y su cantón.
2. Busca el objeto Peaje en el mapa peajes. Si no existe, lo crea.
3. Obtiene la placa ingresada y el tipo de vehículo seleccionado.
4. Según el tipo:
   * Si es **Carro**, se crea un objeto Carro.
   * Si es **Moto**, se crea un objeto Moto.
   * Si es **Camion**, intenta leer el número de ejes y crear un Camion.
     + Si falla (no es número), muestra error con JOptionPane.
5. Llama a p.añadirVehiculo(...) para registrar el vehículo.
6. Actualiza el campo totalField con el nuevo total del peaje.
7. Limpia los campos placaField y ejesField.
8. Guarda todo el Map<String, Peaje> como JSON en vehiculos.json con la librería Gson.
   * Si ocurre un error, lo muestra al usuario.
9. Llama a mostrarPeajeActual(...) para refrescar la interfaz.

peajeCombo.addActionListener(e -> {

mostrarPeajeActual(...);

});

Cuando el usuario selecciona un nuevo peaje en el ComboBox, la interfaz se actualiza para mostrar los datos correspondientes a ese peaje (cantón, vehículos, totales, etc.)

private static void mostrarPeajeActual(JComboBox<String> peajeCombo, Map<String, String> cantonPorPeaje, JTextField cantonField, JTextField totalField, JTextArea outputArea) {

String seleccionado = (String) peajeCombo.getSelectedItem();

if (seleccionado != null && cantonPorPeaje.containsKey(seleccionado)) {

String canton = cantonPorPeaje.get(seleccionado);

cantonField.setText(canton);

Peaje p = peajes.get(seleccionado);

if (p != null) {

totalField.setText(String.valueOf(p.getTotalPeaje()));

StringBuilder sb = new StringBuilder();

sb.append("Peaje: ").append(p.getNombre()).append(" – Cantón: ").append(p.getCanton()).append("\n");

for (Vehiculo v : p.getVehiculos()) {

sb.append(v.getTipo()).append(" – ").append(v.getPlaca())

.append(" – $ ").append(v.calcularPeaje()).append("\n");

}

sb.append("\nTotal Carros: $ ").append(p.getTotalCarros());

sb.append("\nTotal Motos: $ ").append(p.getTotalMotos());

sb.append("\nTotal Camiones: $ ").append(p.getTotalCamiones());

sb.append("\nTotal Recolectado: $ ").append(p.getTotalPeaje());

outputArea.setText(sb.toString());

} else {

totalField.setText("0");

outputArea.setText("");

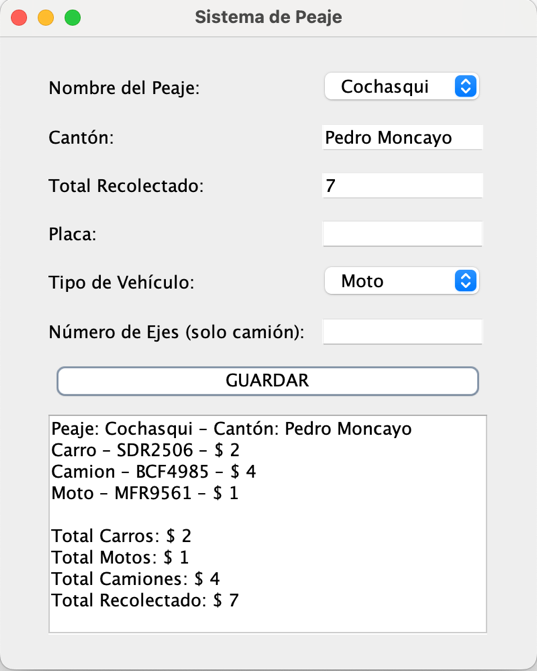
}

**¿Qué hace este método?**

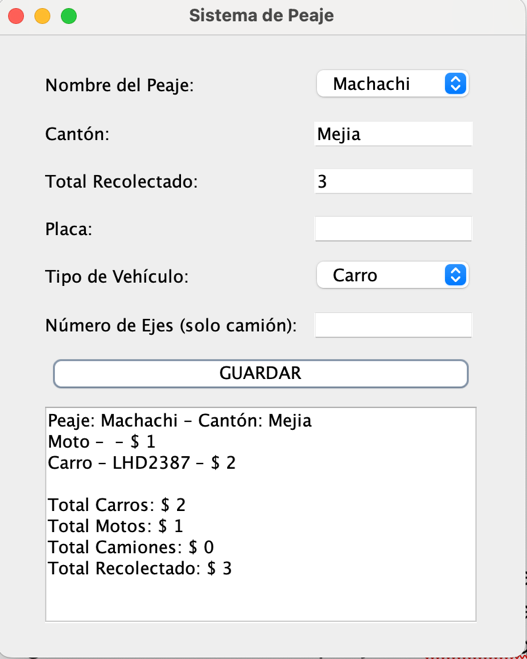
1. Obtiene el peaje seleccionado y su cantón.
2. Si existe un objeto Peaje para ese nombre:
   * Muestra el total recaudado.
   * Recorre todos los vehículos del peaje y muestra:
     + Tipo, placa y monto pagado.
   * Muestra los totales recaudados por:
     + Carros
     + Motos
     + Camiones
     + Total general
3. Si el peaje aún no tiene vehículos, muestra campos vacíos o en 0.

El contenido se muestra en el JTextArea outputArea.

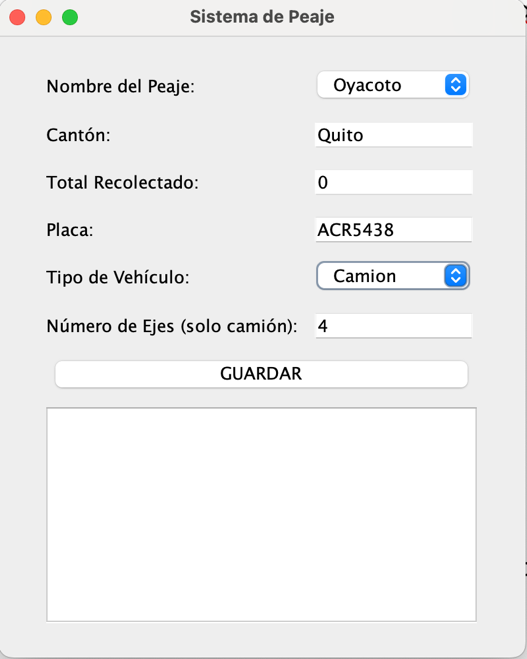
# Ejecución del programa



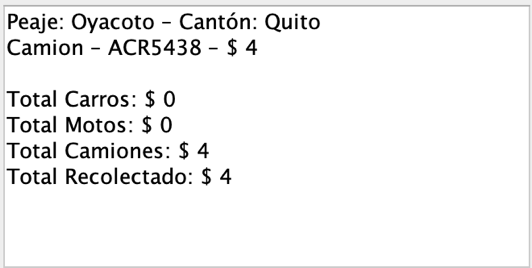
Al ingresar varios datos de diferentes vehículos podemos observar que nos da un resumen detallado de las placas y de los valores recaudados por cada tipo de vehículo al igual que un total general recaudado en el peaje de Cochasqui.

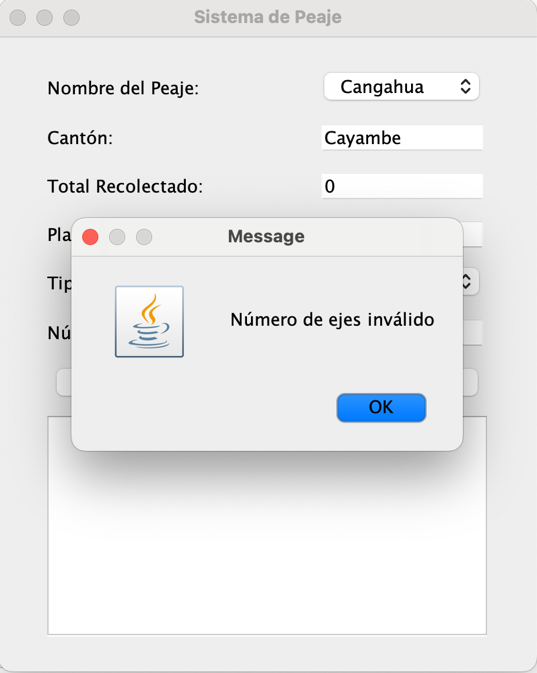


Al seleccionar otro peaje de la lista en este caso Machachi podemos observar que cambian los valores recaudados y lo realiza de una forma automática.



En el caso de seleccionar Camion nos pide el ingreso del número de ejes ya que el valor finar se multiplicara por los ejes que tenga el vehículo y nos mostrara en pantalla el valor total.





Si al momento de seleccionar Camion no escribimos el número de ejes el sistema nos da un mensaje de error como inválido permitiéndonos editar el numero para guardar.

# Conclusiones

Después de desarrollar este proyecto, comprendí de forma práctica cómo aplicar los principios fundamentales de la programación orientada a objetos, como la herencia, el polimorfismo y la abstracción. La creación de clases específicas para cada tipo de vehículo me permitió estructurar el código de forma limpia y reutilizable, facilitando el mantenimiento y la extensión futura del sistema. Esta experiencia reforzó mi habilidad para diseñar soluciones que representen entidades del mundo real en entornos computacionales.

Asimismo, aprendí a construir una interfaz gráfica de usuario utilizando la biblioteca Swing, lo que me permitió crear un entorno amigable para ingresar datos y visualizar resultados. Implementar funcionalidades como la actualización automática del cantón según el peaje seleccionado y el manejo de eventos al presionar botones me ayudó a conectar la lógica de negocio con la experiencia del usuario de manera efectiva. También entendí la importancia del control de errores, como validar el número de ejes en los camiones, para garantizar el correcto funcionamiento del sistema.

Finalmente, experimenté con la persistencia de datos usando la librería Gson para guardar la información en formato JSON. Esta parte del proyecto me mostró lo esencial que es mantener los datos disponibles incluso después de cerrar la aplicación, una práctica muy útil en sistemas reales. En general, este proyecto me permitió integrar conocimientos teóricos con habilidades prácticas en programación estructurada y funcional, fortaleciendo mi formación como futuro ingeniero en software.

# Link del Proyecto

<https://github.com/Jhonna555/SistemaPeaje.git>