# Proyecto de Investigación

## Guía 1

**Juan Manuel Sierra Quintero**

**Santiago Echeverri Alvira**

**Edward Alejandro Suarez Ruiz**

**Facultad de Ingeniería, Universidad EAN**

**Desarrollo de Software - Grupo 1 - Tercer Ciclo - Virtual – 2025**

**Dilsa Enith Triana Martinez**

**25 de agosto de 2025**

# Tabla de Contenido

**Objetivo**.....3

**Informe técnico**.....4

**Diagrama de Clases**.....6

# Objetivo

Desarrollar una aplicación de escritorio basada en el patrón de arquitectura MVC que permita calcular de manera automática y precisa el impuesto vehicular en Colombia a partir del avalúo comercial ingresado por el usuario, garantizando la validación de datos, la correcta aplicación de tarifas y una interfaz gráfica intuitiva y amigable

# Informe técnico

**Proyecto:** Calculadora de Impuestos Vehiculares
**Lenguaje:** Java (JDK 21)
**Entorno:** Apache NetBeans IDE 24

### 1. Objetivo del Sistema

El propósito del proyecto es desarrollar una aplicación de escritorio que permita calcular el impuesto vehicular en Colombia de acuerdo con el avalúo comercial ingresado por el usuario. El sistema sigue el **patrón de arquitectura MVC (Modelo–Vista–Controlador)** con el fin de separar responsabilidades, facilitar el mantenimiento y garantizar la escalabilidad.

### 2. Decisiones de Diseño Realizadas

#### 2.1. Uso del Patrón MVC

- **Modelo (Paquete modelo):** Encapsula la lógica de negocio. Contiene las clases Vehiculo (con su atributo avaluoComercial) y CalculadoraImpuestos (donde se definen las reglas tributarias).
- **Vista (Paquete vista):** Representada por la clase Grafica, que gestiona los componentes gráficos de la interfaz.
- **Controlador (Paquete Controlador):** Encargado de coordinar la interacción entre la vista y el modelo (ControladorVehiculo).

**Decisión clave:** La separación clara permite modificar reglas tributarias o la interfaz gráfica sin afectar las demás capas.

#### 2.2. Validación de Datos de Entrada

En el método ejecutarCalculo() del **Controlador**:

- Se valida que el campo *Avalúo comercial* no esté vacío.
- Se valida que el valor ingresado sea numérico y positivo.
- En caso de error, se generan mensajes mediante JOptionPane, evitando así fallas en tiempo de ejecución.

**Decisión clave:** Priorizar la experiencia del usuario evitando cálculos con datos inválidos.

#### 2.3. Definición de Constantes

En la clase CalculadoraImpuestos:

- Se definen constantes para las tarifas de impuestos (TARIFA_BAJA, TARIFA_MEDIA, TARIFA_ALTA) y límites de avalúo (LIMITE_TARIFA_BAJA, LIMITE_TARIFA_MEDIA).
- Se establece un valor fijo de **derechos de semaforización** (DERECHOS_SEMAFORIZACION).

**Decisión clave:** Centralizar los valores tributarios en constantes facilita futuras actualizaciones sin necesidad de modificar la lógica del código.

### 2.4. Formateo del Resultado

- Se utiliza NumberFormat con la configuración regional **es-CO** para mostrar el resultado en pesos colombianos.
- Se eliminan decimales (setMaximumFractionDigits(0)) para reflejar la forma en que se presentan los impuestos.
- Se cambia el color del texto a verde oscuro (new java.awt.Color(0,100,0)), resaltando el valor final.

**Decisión clave:** Mejorar la legibilidad y adaptarse a los estándares locales de presentación monetaria.

### 2.5. Extensibilidad del Modelo

La clase Vehiculo fue diseñada con un único atributo (avaluoComercial), pero incluye comentarios para posibles futuras extensiones (por ejemplo, **marca** o **año del vehículo**).

**Decisión clave:** Preparar la aplicación para crecer en funcionalidad sin afectar su estabilidad actual.

### 2.6. Experiencia del Usuario

- Se implementa un botón **Calcular** para procesar el impuesto.
- Se implementa un botón **Limpiar** para restablecer los campos de la interfaz.
- La ventana se centra automáticamente en pantalla (vista.setLocationRelativeTo(null)).
- Se aplica el Look and Feel **Nimbus** para una apariencia moderna.

**Decisión clave:** Priorizar la usabilidad con una interfaz clara y amigable.

## 3. Conclusiones

El desarrollo de la aplicación se realizó aplicando buenas prácticas de programación orientada a objetos y siguiendo el patrón MVC. Las decisiones tomadas permiten:

- Separación de responsabilidades (modelo, vista, controlador).
- Facilidad de mantenimiento y escalabilidad.
- Validaciones robustas que previenen errores.
- Presentación clara y adaptada al contexto colombiano.
- Base sólida para extenderse con nuevas funcionalidades (más atributos del vehículo, reportes, persistencia en base de datos, etc.).

En resumen, las elecciones de diseño permiten garantizar que la **Calculadora de Impuestos Vehiculares** sea confiable, extensible y fácil de usar

# Diagrama de Clases

https://drive.google.com/file/d/1PbNJ6SOAvYTLzsh0Gf-uO1sUkZuA6Rhw/view?usp=sharing