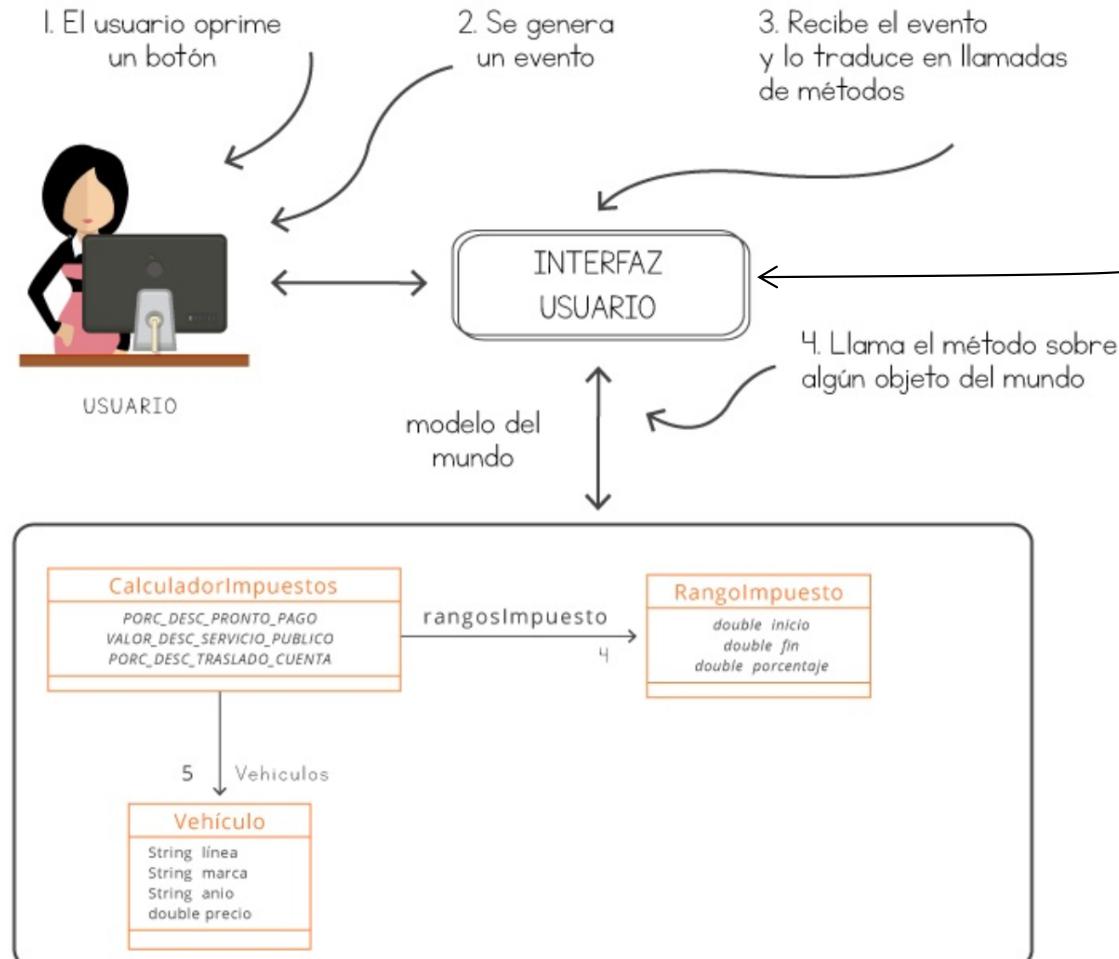
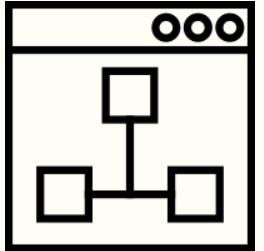


5 NÍVEL

Construcción de la interfaz gráfica
{Elementos gráficos estructurales}

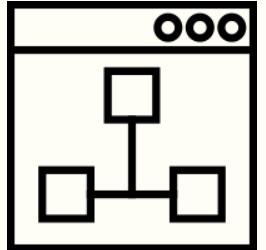


INTERFAZ DE USUARIO



Nuestro objetivo es estructurarla y comunicarla con las clases del mundo, sin mezclar sus responsabilidades

INTERFAZ DE USUARIO



USUARIO

Diseño funcional y gráfico

- ✓ Colores
- ✓ Distribución de los elementos
(menús, botones, ...)

Arquitectura

- ✓ Estructura clara
- ✓ Fácil de mantener

Los lenguajes de programación proveen un conjunto de clases (*framework* o librerías gráficas) para facilitar la construcción de interfaces. Trabajaremos swing y awt de java

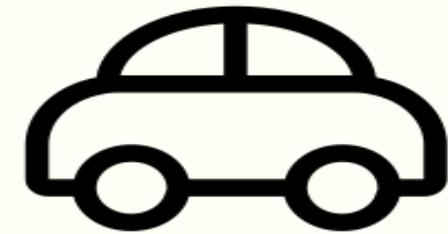
CASO – CÁLCULO DE IMPUESTOS

El programa debe permitir a una persona calcular el valor de los impuestos que debe pagar por su automóvil, dependiendo del valor del automóvil, y de los descuentos que contempla la ley.

Un vehículo se caracteriza por tener una marca (Peugeot), una línea (206), un modelo (2016) y un precio.

Para calcular el valor del impuesto el programa establece ciertos rangos de precio que tienen asociados los porcentajes que se aplican, así:

- 0 - 30 millones: 1.5%
- 30 - 70 millones: 2.0%
- 70 - 200 millones: 2.5%
- Mas 200 millones: 4%



Existen tres tipos de descuentos que pueden aplicar:

- Descuento por pronto pago: 10% si se paga antes del 31 de marzo.
- Descuento por ser de servicio público: 50.000 pesos anuales
- Descuento por traslado de cuenta: 5%

Los descuentos se aplican en el orden dado. Es decir, que si a un carro le aplican los tres y debe pagar 150.000 pesos de impuesto, realmente pagará 80.750.

CASO – CÁLCULO DE IMPUESTOS



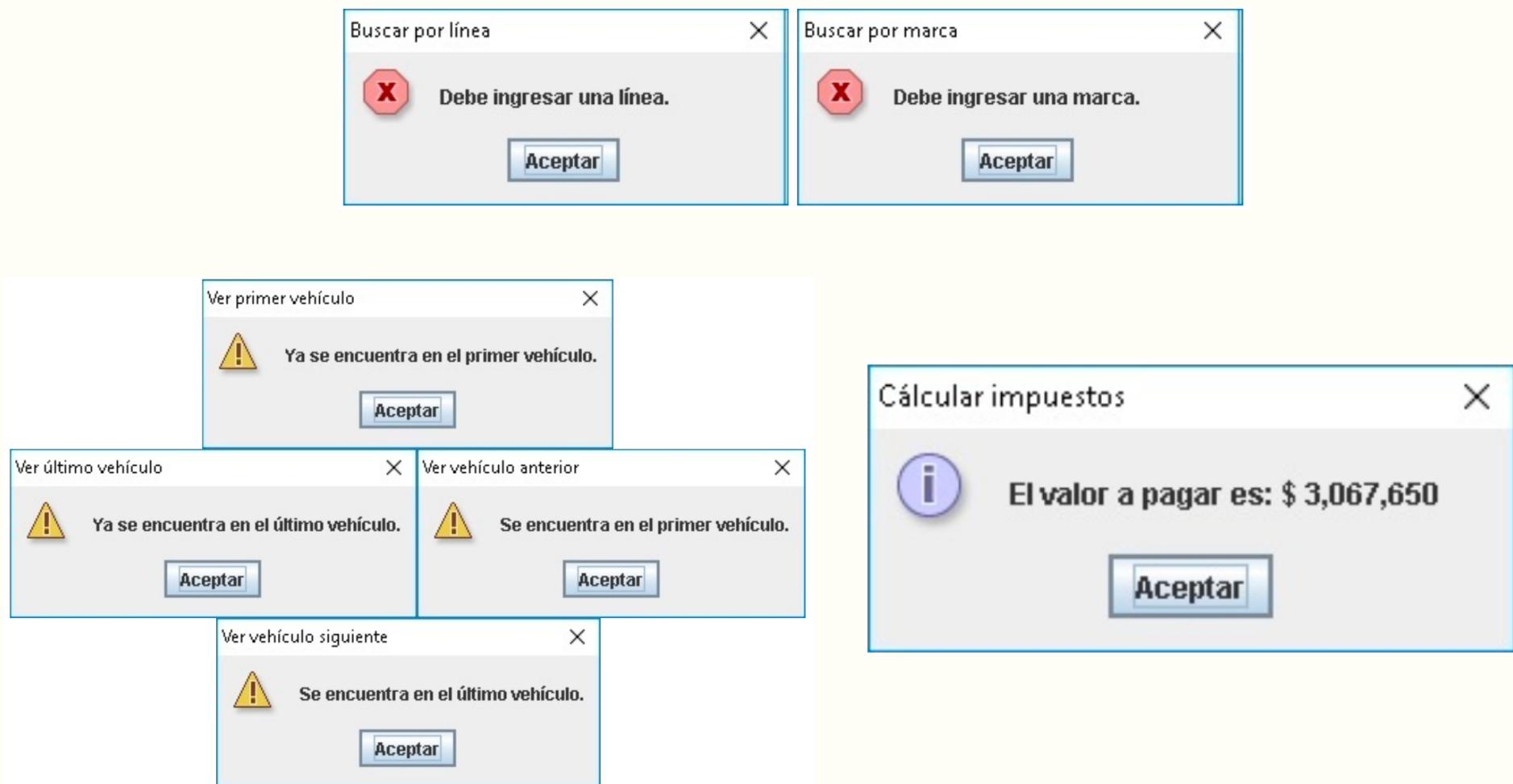
La ventana está dividida en 5 zonas:

1. Muestra la imagen con el nombre de la aplicación
2. Muestra información del vehículo y permite navegar por los vehículos existentes
3. Busca un vehículo por línea o marca y encuentra el vehículo más costoso
4. Permite seleccionar los descuentos que se desean aplicar
5. Permite calcular el impuesto del vehículo actual y 2 opciones más

Fíjese que cada zona tiene un borde y un título

CASO – CÁLCULO DE IMPUESTOS

Mensajes de la interfaz al usuario



REQ. FUNCIONALES

Nombre	R1 – Navegar entre vehículos
Resumen	Adelanta o retorcede entre los vehículos disponibles
Entradas	Dirección de navegación seleccionada
Resultados	Se muestra el vehículo siguiente o anterior dependiendo de la dirección seleccionada
	Si se quiere navegar a un vehículo anterior al primero o posterior al último, se muestra un mensaje indicándolo

REQ. FUNCIONALES

Nombre	R2 – Buscar vehículo por línea
Resumen	Busca un vehículo por una línea dada
Entradas	
Línea del vehículo	
Resultados	
Se muestra el primer vehículo que se encuentre que tenga la línea dada	
Si no se encuentra ningún vehículo con la línea buscada, se muestra un mensaje indicándolo	

REQ. FUNCIONALES

Nombre	R3 – Buscar vehículo por marca
Resumen	Busca un vehículo por una marca dada
Entradas	
Marca del vehículo	
Resultados	
Se muestra el primer vehículo que se encuentre que tenga la marca dada	
Si no se encuentra ningún vehículo con la marca buscada, se muestra un mensaje indicándolo	

REQ. FUNCIONALES

Nombre	R4 – Buscar vehículo más costoso
Resumen	Busca el vehículo con el mayor valor registrado
Entradas	
Ninguna	
Resultados	
Se muestra el vehículo más costoso	

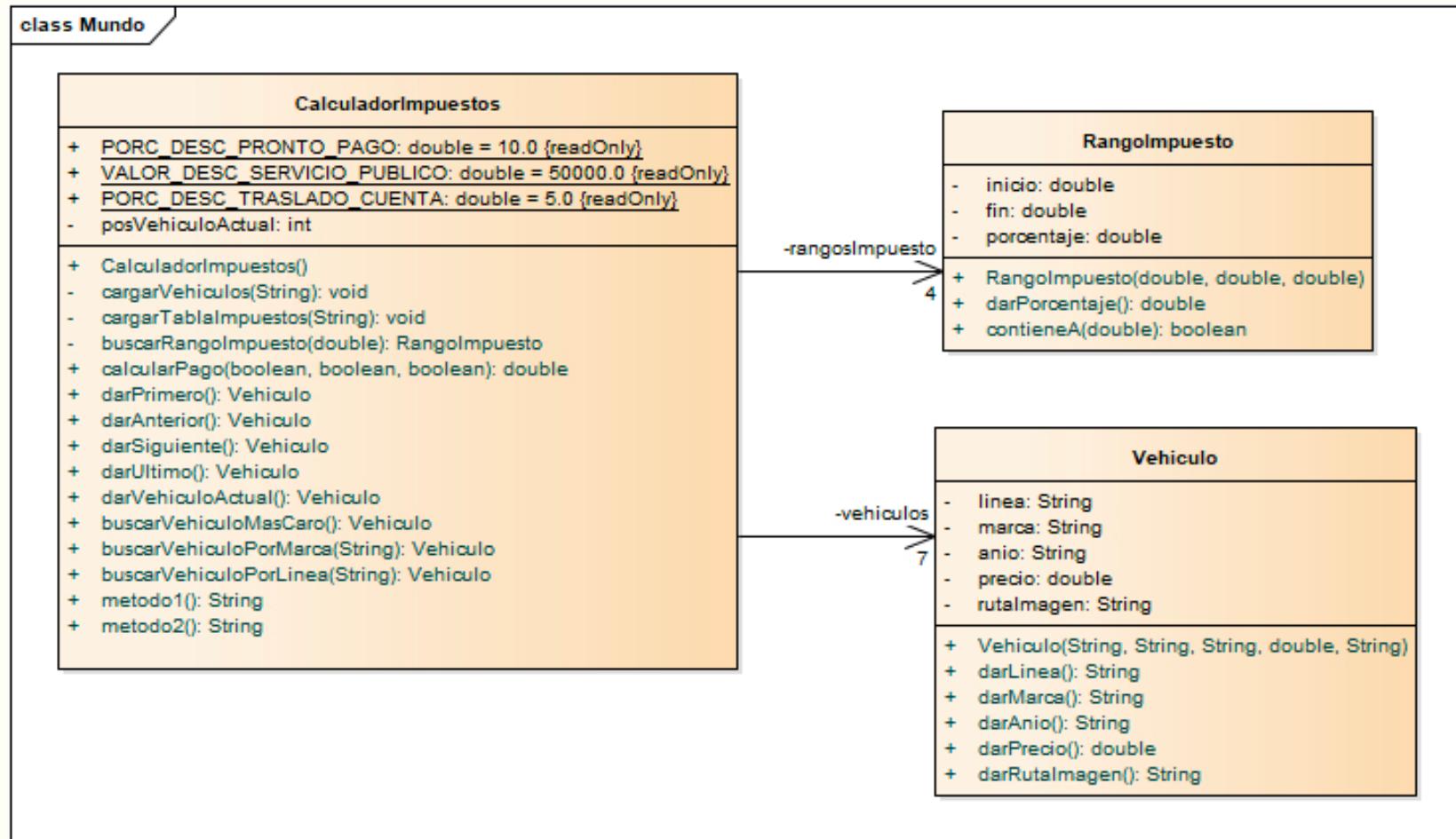
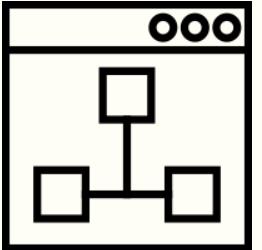
REQ. FUNCIONALES

Nombre	R5 – Calcular impuesto
Resumen	Calcula el impuesto de un vehículo teniendo en cuenta su valor y los descuentos que se pueden aplicar
Entradas	
	?
Resultados	
	?

REQ. FUNCIONALES

Nombre	R6 – Visualizar información del vehículo
Resumen	Se muestra la información detallada de un vehículo. Esta incluye la marca, la línea, el modelo y el valor
Entradas	
Ninguna	
Resultados	
	Se muestra la información detallada del vehículo

MODELO DEL MUNDO





CONTRATOS

Los contratos de los métodos que invocaremos desde la interfaz para pedir los servicios que solicite el usuario, pasándoles como parámetro la información que este ingrese, son:

Método constructor

```
/**  
 * Crea un calculador de impuestos, cargando la información de dos archivos. <br>  
 * <b>post: </b> Se inicializaron los arreglos de vehículos y rangos.<br>  
 * Se cargaron los datos correctamente a partir de los archivos.  
 * @throws Exception Error al cargar los archivos.  
 */  
public CalculadorImpuestos() throws Exception  
{  
}
```

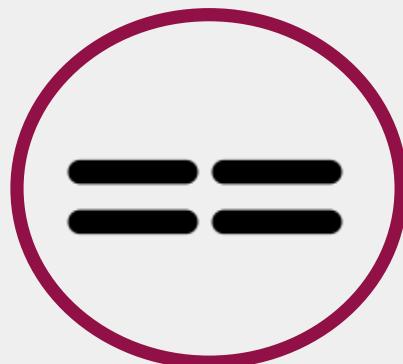
CONTRATOS

```
/**  
 * Calcula el pago de impuesto que debe hacer el vehículo actual. <br>  
 * <b>pre:</b> Las listas de rangos y vehículos están inicializadas.  
 * @param pDescProntoPago Indica si aplica el descuento por pronto pago.  
 * @param pDescServicioPublico Indica si aplica el descuento por servicio público.  
 * @param pDescTrasladoCuenta Indica si aplica el descuento por traslado de cuenta.  
 * @return Valor a pagar de acuerdo con las características del vehículo y los  
 * descuentos que se pueden aplicar.  
 */  
  
public double calcularPago(boolean pDescProntoPago, boolean pDescServicioPublico,  
                           boolean pDescTrasladoCuenta )  
{  
  
}
```

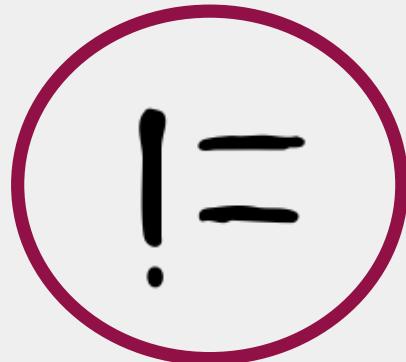


INTERFACES GRÁFICAS

Diseñaremos interfaces sencillas con campos de edición y botones para activar los requerimientos



- ✓ Clases
- ✓ Métodos
- ✓ Instrucciones iterativas



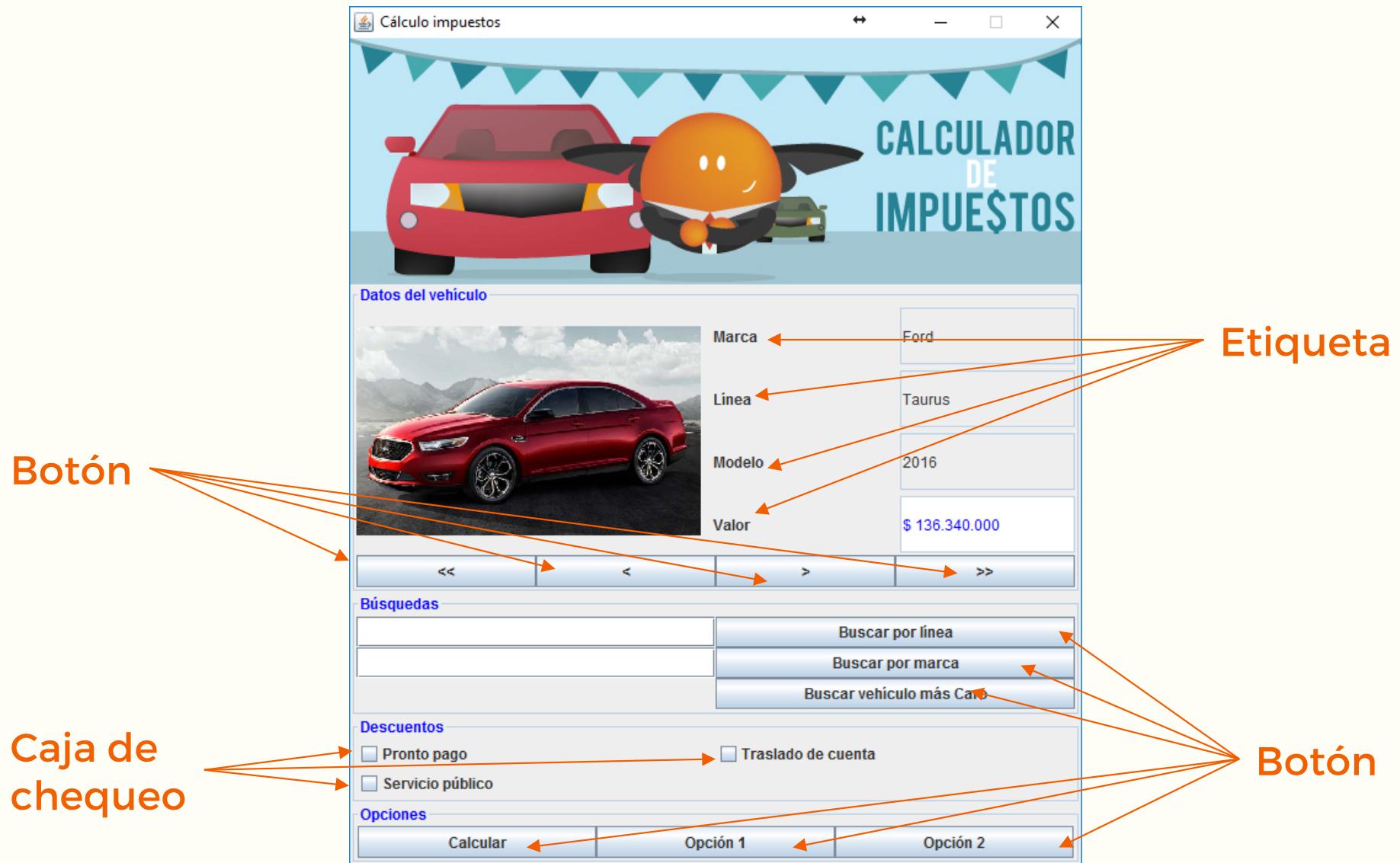
- ✓ Ventana
- ✓ Botón
- ✓ Distribuido gráfico
- ✓ Campo de texto...

Identificaremos que elementos de la interfaz tienen un propósito común y para cada uno diseñaremos una clase

ENTIDADES GRÁFICAS



ENTIDADES GRÁFICAS





MUNDO VS INTERFAZ

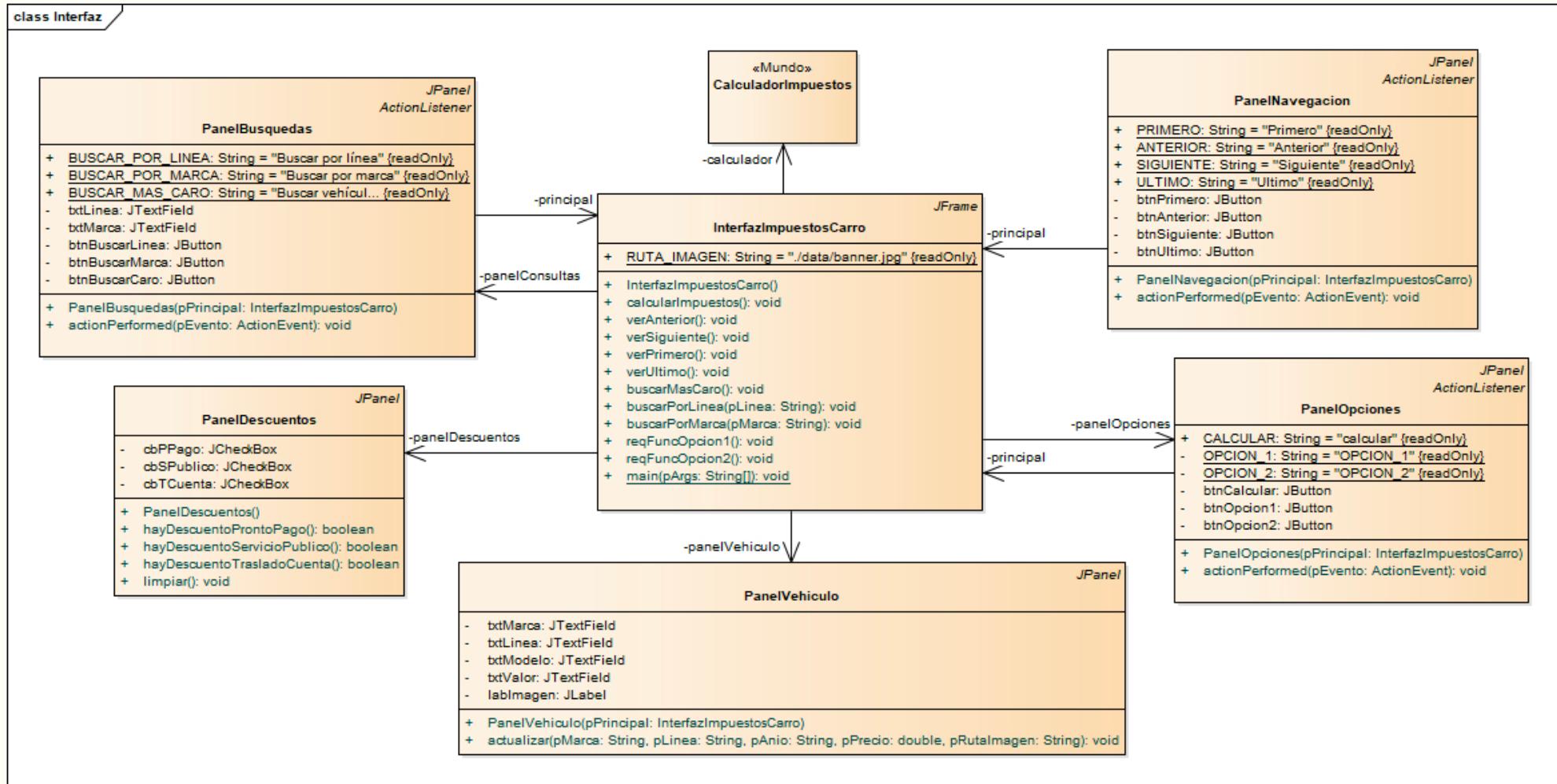
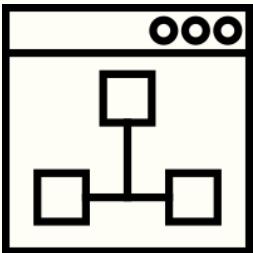


- ✓ Estudiante
- ✓ Tienda
- ✓ Avión
- ✓ Hotel....



- ✓ Ventana (JFrame)
- ✓ Panel (JPanel)
- ✓ Botón (JButton)
- ✓ Zona de texto (JTextField)
- ✓ Etiqueta (JLabel)
- ✓ Caja de chequeo (JCheckBox)

MODELO DE LA INTERFAZ



VENTANA

PRINCIPAL



- Contiene todos los elementos de visualización e interacción
- Típicamente tiene en la parte superior derecha los controles para cerrar el programa, minimizar y cambiar de tamaño.
- Es un **contenedor gráfico**
- Es un **objeto de una clase que se ha declarado de forma particular** (`InterfazImpuestosCarro`), y al igual que cualquier otra clase debe estar declarada en su propio archivo .java
- Para que tenga el comportamiento de una ventana estándar debemos decir que es de un tipo particular llamado `JFrame`.

DECLARACIÓN VENTANA PRINCIPAL

```
package uniandes.cupi2.impuestosCarro.interfaz;

import java.awt.*;
import javax.swing.*;

import uniandes.cupi2.impuestosCarro.mundo.*;

/**
 * Interfaz de cálculo de impuestos de carros
 */

public class InterfazImpuestosCarro extends JFrame
{
    ...
}
```

ALGUNOS MÉTODOS

La clase JFrame tiene métodos implementados que podemos utilizar para modificar el estado de la ventana. Algunos son:

- `setSize (ancho, alto)`: permite cambiar el ancho y el alto de la ventana. Los valores de los parámetros están dados en pixeles
- `setResizable (true/false)`: indica si el usuario puede o no cambiar el tamaño de la ventana
- `setTitle (titulo)`: cambia el título que se muestra en la parte superior de la ventana
- `setDefaultCloseOperation (EXIT_ON_CLOSE)`: indica que la aplicación debe terminarse al cerrar la ventana
- `setVisible (true/false)`: hace aparecer o desaparecer la ventana de la pantalla dependiendo del valor lógico
- `add (componente)`: permite agregar un componente gráfico a la ventana

CONFIGURACIÓN BÁSICA - VP

```
public class InterfazImpuestosCarro extends JFrame  
{  
    private CalculadorImpuestos calculador;
```

Atributo que es una asociación a la clase CalculadorImpuestos - Mundo

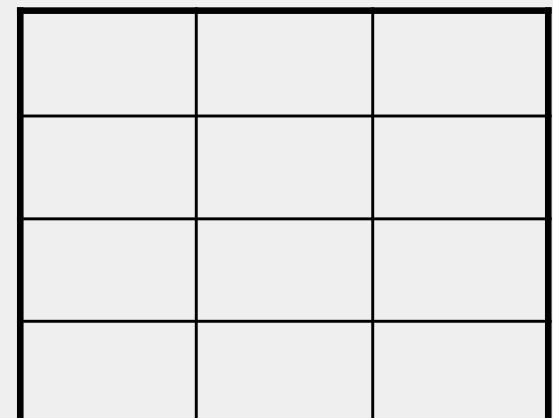
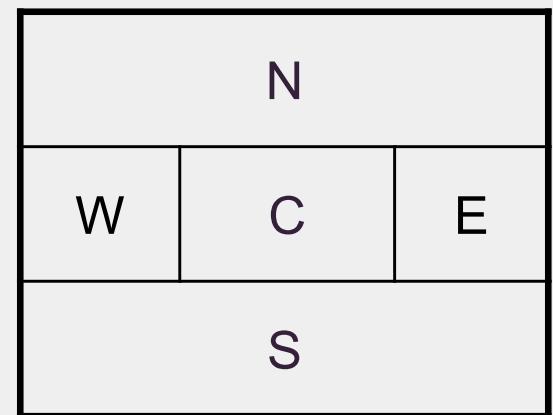
```
public InterfazImpuestosCarro () throws Exception  
{  
    setTitle( "Cálculo impuestos" );  
    setSize( 600, 700 );  
    setResizable( false );  
    setDefaultCloseOperation( EXIT_ON_CLOSE );  
    ...  
}
```

En el constructor de la clase se configura la ventana y se llaman los métodos como si pertenecieran a la clase ya que extiende a JFrame

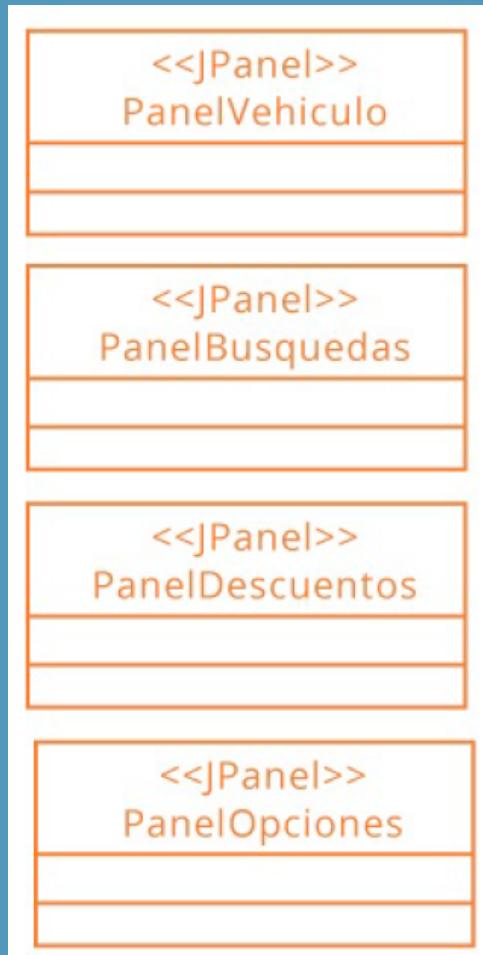
DISTRIBUCIÓN GRÁFICA

Para realizar la distribución gráfica de los elementos dentro de la ventana principal, Java incluye el concepto de Distribuidor Gráfico (**layout**).

- **BorderLayout:** divide el espacio en 5 zonas (NORTH, CENTER, SOUTH, WEST, EAST)
`setLayout(new BorderLayout());
add(panelVehiculo, BorderLayout.NORTH);`
- **GridLayout:** divide el espacio en un número de filas y columnas establecido en su instancia:
`setLayout(new GridLayout(4,3));`



PANELES - DIVISIONES



- Dentro de la ventana principal aparecen los paneles encargados de agrupar los elementos por contenido y uso, de manera que para el usuario sea sencillo manipularlos
- Cada división se implementa como una clase aparte ([JPanel](#)) en el modelo y al igual que la ventana es un contenedor gráfico al que se le asigna su propio layout
- En el constructor de la ventana se debe crear una instancia de cada uno de los paneles y luego adicionarlas a ésta

AGREGANDO PANELES - VP

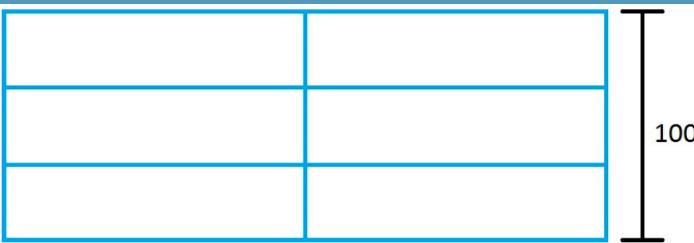
```
public class InterfazImpuestosCarro extends JFrame {  
  
    private CalculadorImpuestos calculador;  
  
    private PanelVehiculo panelVehiculo;  
    private PanelBusquedas panelBusquedas;  
    private PanelDescuentos panelDescuentos;  
    private PanelOpciones panelOpciones;  
  
    public InterfazImpuestosCarro () throws Exception {  
  
        setTitle ("Cálculo impuestos" );  
        setSize ( 290, 350 );  
        setResizable ( false );  
        setDefaultCloseOperation ( EXIT_ON_CLOSE );  
        setLayout ( new BorderLayout () );  
  
        JPanel centro = new JPanel ();  
        centro.setLayout ( new BorderLayout () );  
        add ( centro, BorderLayout.CENTER );  
    ...  
}
```

← **Atributos por cada uno de los paneles**

AGREGANDO PANELES - VP

```
public class InterfazImpuestosCarro extends JFrame {  
    ...  
    panelVehiculo = new PanelVehiculo ( this );  
    centro.add ( panelVehiculo, BorderLayout.CENTER );  
    panelBusquedas = new PanelBusesquedas ( this );  
    centro.add ( panelVehiculo, BorderLayout.SOUTH );  
    JPanel sur = new JPanel ();  
    sur.setLayout ( new BorderLayout () );  
    add ( sur, BorderLayout.CENTER );  
    panelDescuentos = new PanelDescuentos ();  
    sur.add ( panelDescuentos, BorderLayout.CENTER );  
    panelOpciones = new PanelOpciones ( this );  
    sur.add ( panelOpciones, BorderLayout.SOUTH );  
}  
}
```

PANELES - DIVISIONES



- Para la construcción de “cada uno” de los paneles de la ventana se hace como extensión de la clase JPanel
- Se utiliza el método setPreferredSize (dimension) para definir el tamaño de los paneles.
- Ejemplos:
 - ✓ PanelBusquedas
 - ✓ PanelDescuentos
 - ✓ PanelOpciones

CONFIGURANDO PANELES

```
public PanelBusquedas ( InterfazImpuestosCarro pPrincipal ) {  
    setLayout ( new GridLayout ( 3, 2 ) );  
    setPreferredSize ( new Dimension ( 0, 100 ) ); ←  
    TitledBorder border = new TitledBorder ( "Búsquedas" );  
    border.setTitleColor ( Color.BLUE );  
    setBorder ( border );  
}  
...
```

Se pasa el alto, 130 pixeles, para reserverlo y 0 de ancho para que le asigne el total de espacio disponible en la ventana

```
public PanelDescuentos ( ) {  
    setLayout ( new GridLayout ( 2, 2 ) );  
    TitledBorder border = new TitledBorder ( "Descuentos" );  
    border.setTitleColor ( Color.BLUE );  
    setBorder ( border );  
}  
...
```

Aquí no importante la dimensión porque se le asignará todo el espacio disponible

```
public PanelOpciones ( InterfazImpuestosCarro pPrincipal ) {  
    setLayout ( new GridLayout ( 1, 3 ) );  
    TitledBorder border = new TitledBorder ( "Opciones" );  
    border.setTitleColor ( Color.BLUE );  
    setBorder ( border );  
}  
...
```

CONFIGURANDO PANELES

```
public PanelVehiculo ( InterfazImpuestosCarro pPrincipal ) {  
    setLayout ( new BorderLayout () );  
    labImagen = new JLabel ();  
    labImagen.setBorder ( new EmptyBorder ( 0, 0, 0, 10 ) );  
    add ( labImagen, BorderLayout.WEST );  
  
    JPanel informacion = new JPanel ();  
    informacion.setLayout ( new GridLayout ( 4, 2, 10, 5 ) );  
    add ( informacion, BorderLayout.CENTER );  
  
    TitledBorder border = new TitledBorder ( "Datos del vehículo" );  
    border.setTitleColor ( Color.BLUE );  
    setBorder ( border );  
    ...  
}  
  
public PanelNavegacion ( InterfazImpuestosCarro pPrincipal ) {  
    setLayout ( new GridLayout ( 1, 4 ) );  
    ...  
}
```

Convención: Clase de los componentes de este tipo “Panel” seguido de una descripción

ETIQUETAS

Y ZONAS DE TEXTO

Una vez realizadas la divisiones, se agregan los elementos gráficos y de interacción.

Etiquetas (*labels*): permiten agregar un texto corto, la mayoría de las veces para explicar algún elemento. Son objetos de la clase `JLabel`, que se crean pasándole el texto al constructor. Algunos métodos son:

- `setText (etiqueta)`: permite cambiar el valor de la etiqueta
- `setForeground (color)`: permite cambiar el color de la etiqueta

Para agregar una etiqueta se siguen 4 pasos: (1) Declarar en el panel un atributo `JLabel`, (2) agregar la instrucción de creación de la etiqueta, (3) utilizar los métodos de la clase para configurarla, (4) utilizar la instrucción `add` del panel para agregarla en la zona que corresponda.



ETIQUETAS

Y ZONAS DE TEXTO

Zonas de Texto: Son objetos de la clase `JTextField`. Permiten al usuario ingresar información correspondiente a las entradas de los requerimientos funcionales y obtener las respuestas calculadas por el programa. Algunos métodos son:



- `getText ()`: retorna la cadena de caracteres tecleada por el usuario (todo lo que el usuario teclea se maneja como una cadena)
- `setText (texto)`: despliega en la zona de texto la cadena que pasa como parámetro
- `setEditable (editable)`: indica si el contenido de la zona de texto puede ser modificado por el usuario, usualmente las respuestas del programa no se permiten editar
- `setForeground (color)`: define el color de los caracteres que aparecen en la zona de texto
- `setBackground (color)`: define el color del fondo de la zona de texto

AGREGANDO COMPONENTES

```
public class PanelVehiculo extends JPanel
```

```
{
```

```
// Atributos
```

```
private JTextField txtMarca;  
private JTextField txtLinea;  
private JTextField txtModelo;  
private JTextField txtValor;  
private JLabel labImagen;
```

```
}
```

(1) Se declara un atributo en la clase por cada componente gráfico cuyo valor cambiará después de ser creado: 1 etiqueta y 4 zonas de texto asociadas

Convención:
Etiquetas: "lab" + descripción
Zonas Texto: "txt" + descripción

AGREGANDO COMPONENTES

```
public class PanelVehiculo extends JPanel  
{  
    ...  
    public PanelVehiculo ( InterfazImpuestosCarro pPrincipal )  
    {  
        ...  
        labImagen = new JLabel ( );  
        JLabel labMarca = new JLabel ( "Marca" );  
        JLabel labLinea = new JLabel ( "Línea" );  
        JLabel labModelo = new JLabel ( "Modelo" );  
        JLabel labValor = new JLabel ( "Valor" );  
  
        txtMarca = new JTextField ( );  
        txtLinea = new JTextField ( );  
        txtModelo = new JTextField ( );  
        txtValor = new JTextField ( );  
        ...  
    }  
}
```

(2) En el constructor del panel se crea el objeto que representa cada uno de los componentes

AGREGANDO COMPONENTES

```
public class PanelVehiculo extends JPanel {  
    public PanelVehiculo ( InterfazImpuestosCarro pPrincipal ) {  
        ...  
        txtMarca.setEditable ( false );  
        txtLinea.setEditable ( false );  
        txtModelo.setEditable ( false );  
        txtValor.setEditable ( false );  
        txtValor.setForeground ( Color.BLUE );  
        txtValor.setBackground ( Color.WHITE );  
  
        add( labImagen, BorderLayout.WEST );  
        informacion.add ( labMarca );  
        informacion.add ( txtMarca );  
        informacion.add ( labLinea );  
        informacion.add ( txtLinea );  
        informacion.add ( labModelo );  
        informacion.add ( txtModelo );  
        informacion.add ( labValor );  
        informacion.add ( txtValor );  
        add ( new PanelNavegacion ( pPrincipal ), BorderLayout.SOUTH );  
    }  
}
```

(3) Utilizando los métodos de la clase se configura el componente

(4) Se adicionan en el panel los componentes creados, teniendo presente el orden utilizado por el distribuidor (de izquierda a derecha y de arriba a abajo)

VALIDACIÓN DE DATOS

Tomamos la cadena de caracteres que teclee el usuario

```
try
{
    String strModelo = txtModelo.getText();
    int nModelo = Integer.parseInt(strModelo);
}
catch (Exception e)
{
    txtModelo.setText("");
}
```

Método parseInt de la clase Integer

Si se produce una excepción borramos la información tecleada por el usuario y se podría dar un mensaje de error

La interfaz muchas veces tiene la responsabilidad de convertir al formato y tipo adecuado los datos que digita el usuario para poder manipularlos, así como advertirle si comete un error al digitarlos (i.e. si se espera un número y teclea una letra)

- Las zonas de texto siempre se leen como **Strings**
- Convertir una cadena (que contiene sólo dígitos) en un número

FORMATO DE DATOS

La clase `String` ofrece métodos para transformar cadenas como:

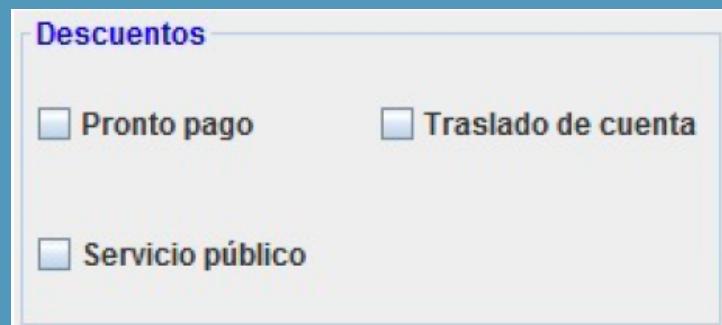
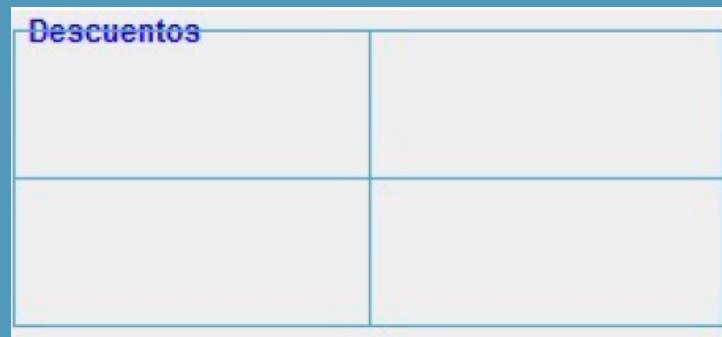
- `toLowerCase ()`: convierte la cadena a minúsculas
- `toUpperCase ()`: convierte la cadena a mayúsculas
- `trim ()`: elimina los caracteres en blanco del inicio y fin de la cadena

Para formatear adecuadamente los valores numéricos, por ejemplo el # real 1624350,7898 que se debe pagar como impuesto del vehículo en el caso de estudio, se muestra como \$ 1'624.350,79, utilizando el siguiente código:

```
DecimalFormat df = ( DecimalFormat )NumberFormat.getInstance();
f.applyPattern ( "$ ###,###.##" );
String strPago = df.format ( pago );
txtTotal.setText ( strPago );
```

La clase `DecimalFormat` realiza este tipo de formateo → paquete `java.text`

OPCIONES - CAJAS DE CHEQUEO



Swing provee un componente gráfico (`JCheckBox`) que permite seleccionar o no una opción. Por ejemplo, los descuentos a los que tiene derecho el cliente. Algunos métodos son:

- `isSelected ()`: retorna un valor lógico indicando si el usuario seleccionó o no la opción
- `setSelected (seleccionado)`: marca como seleccionado o no, de acuerdo al parámetro

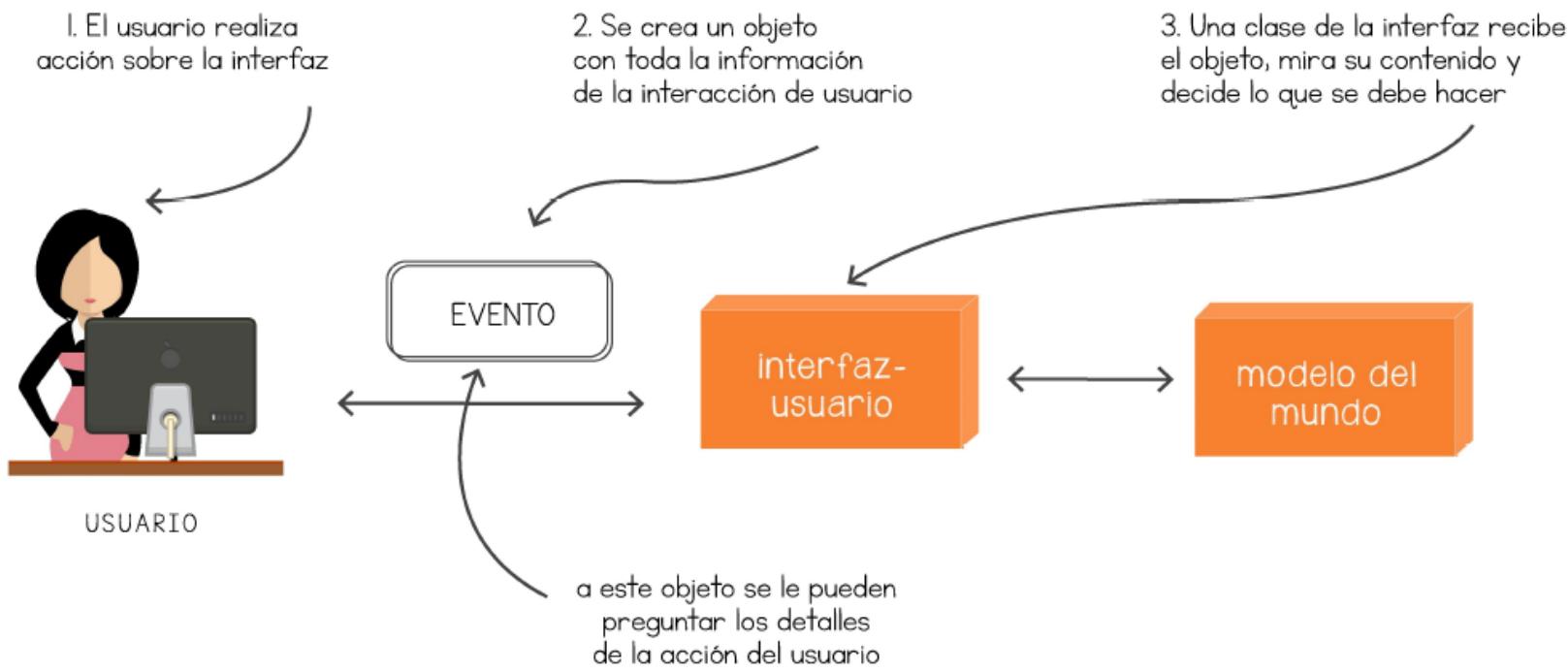
Convención:
Opciones: "cb" + descripción

CAJAS DE CHEQUEO

```
public class PanelDescuentos extends JPanel
{
    private JCheckBox cbPPago;
    private JCheckBox cbSPublico;
    private JCheckBox cbTCuenta;

    public PanelDescuentos ()
    {
        ...
        cbPPago = new JCheckBox ( "Pronto pago" );
        cbSPublico = new JCheckBox ( "Servicio público" );
        cbTCuenta = new JCheckBox ( "Traslado de cuenta" );
        add( cbPPago );
        add( cbTCuenta );
        add( cbSPublico );
    }
}
```

ELEMENTOS DE INTERACCIÓN



Existen muchas formas de interacción a través de las cuales el usuario puede expresar sus órdenes. Todas estas acciones se convierten en eventos y son manipulados mediante objetos

BOTONES

Manejaremos interacciones a través de botones que son componentes gráficos que pertenecen a la clase JButton y se agregan y declaran como cualquier otro.

```
public class PanelNavegacion extends JPanel implements ActionListener {
```

```
    private JButton btnPrimero;  
    private JButton btnAnterior;  
    private JButton btnSiguiente;  
    private JButton btnUltimo;
```

Declaración de los componentes como atributos

```
public PanelNavegacion ( InterfazImpuestosCarro pPrincipal ){
```

```
    ...  
    btnPrimero = new JButton( "<<" );  
    add( btnPrimero );  
    btnAnterior = new JButton( "<" );  
    add( btnAnterior );  
    btnSiguiente = new JButton( ">" );  
    add( btnSiguiente );  
    btnUltimo = new JButton( ">>" );  
    add( btnUltimo );  
    ...  
}
```

Creación de los objetos agregándolos al panel

MANEJO DE EVENTOS

3 pasos para manejar un evento con un botón de la interfaz:

1. Decidir el nombre del evento: a los eventos de los botones se les asocia un nombre, usualmente una constante, por ejemplo:

```
public class PanelNavegacion extends JPanel implements ActionListener {  
    // Constantes  
    public final static String PRIMERO = "Primero";  
    public final static String ANTERIOR = "Anterior";  
    public final static String SIGUIENTE = "Siguiente";  
    public final static String ULTIMO = "Ultimo";  
  
    public PanelNavegacion ( InterfazImpuestosCarro pPrincipal ){  
        ...  
        btnPrimero.setActionCommand ( PRIMERO );  
        btnPrimero.setActionCommand ( ANTERIOR );  
        btnPrimero.setActionCommand ( SIGUIENTE );  
        btnPrimero.setActionCommand ( ULTIMO );  
    ...}  
}
```

MANEJO DE EVENTOS

2. Implementar el método que va atender el evento: el panel que tiene el botón debe agregar una declaración en el encabezado de la clase (`implements ActionListener`) e implementar un método especial llamado `actionPerformed` que recibe como parámetro el evento ocurrido en el panel, por ejemplo:

```
public class PanelNavegacion extends JPanel implements ActionListener {  
...  
    public void actionPerformed ( ActionEvent pEvento ) {  
  
        String comando = pEvento.getActionCommand();  
        if( comando.equals ( PRIMERO ) )  
            //Por definir  
        else if( comando.equals ( ANTERIOR ) )  
            //Por definir  
        else if( comando.equals ( SIGUIENTE ) )  
            //Por definir  
        else  
            //Por definir  
    }  
}
```

Cada vez que se oprime un botón se ejecuta el método anterior, que utilizará el nombre del mismo para decidir la acción a realizar. Si no se implementa este método y la clase dice `implements ActionListener` se producirá un error de compilación

MANEJO DE EVENTOS

3. Declarar al panel como el encargado de atender los eventos de sus botones: para esto se utiliza el método `addActionListener`, pasando como referencia el panel. Se hace en el constructor del panel utilizando la variable `this`, con la que Java permite hacer referencia al objeto que está ejecutando un método, por ejemplo:

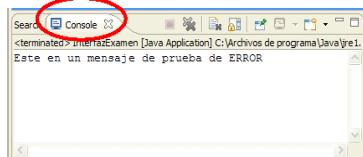
```
public class PanelNavegacion extends JPanel implements ActionListener  
{  
...  
public PanelNavegacion ( InterfazImpuestosCarro pPrincipal ){  
...  
    btnPrimero.addActionListener( this );  
    btnAnterior.addActionListener( this );  
    btnSiguiente.addActionListener( this );  
    btnUltimo.addActionListener( this );  
}  
}
```

Tarea: Revisar el código completo de la clase `PanelNavegacion`

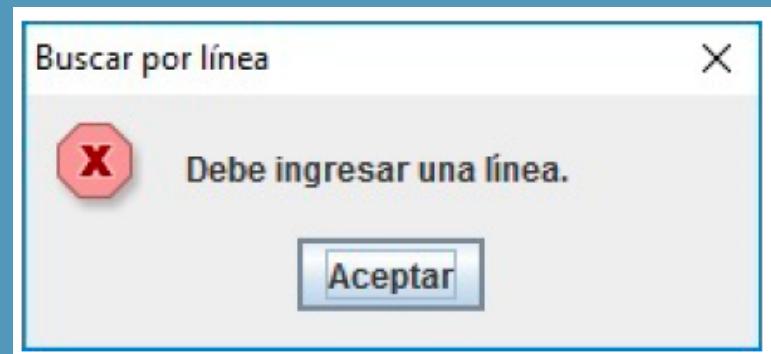
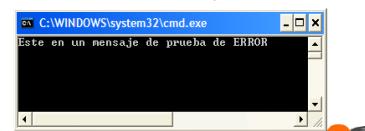
MENSAJES AL USUARIO

System.out.println("Este es un mensaje de prueba de ERROR");

Si se ejecuta desde eclipse, el mensaje aparece en la ventana Console



Si se ejecuta por fuera de eclipse (con run.bat)



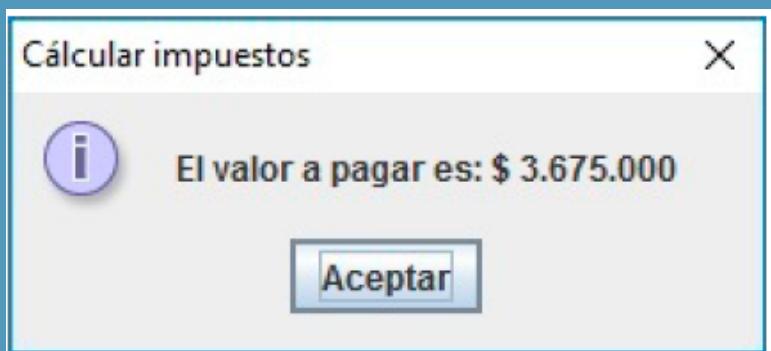
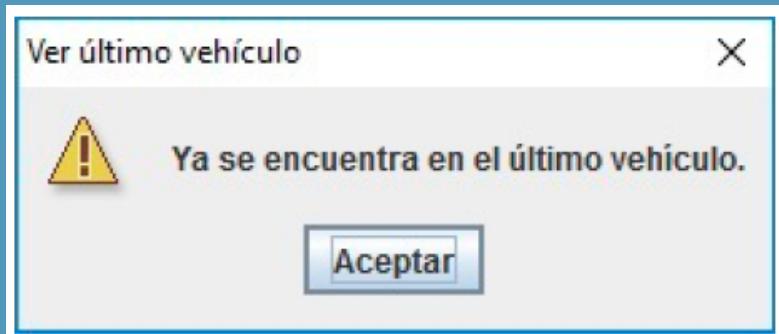
USUARIO

- **Mensajes en consola:** para mostrar un mensaje por la ventana del sistema operacional, o en eclipse en una ventana especial llamada consola, se utiliza la instrucción `System.out.println` (cadena)
- **Mensajes en una ventana:** se puede utilizar la clase `JOptionPane` para enviarle pequeños mensajes al usuario en una ventana adicional. Es muy útil en caso de error o para mostarle resultados.

```
JOptionPane.showMessageDialog( this, "Debe  
ingresar una línea.", "Buscar por línea",  
JOptionPane.ERROR_MESSAGE );
```

MENSAJES AL USUARIO

USUARIO



- Mensajes en una ventana:

```
JOptionPane.showMessageDialog( this , "Ya se encuentra en el último vehículo.", "Ver último vehículo" , JOptionPane.WARNING_MESSAGE );
```

Debe ser null si está en el main

```
JOptionPane.showMessageDialog( this , "El valor a pagar es: $3.675.000" , "Cálculo de Impuestos" , JOptionPane.INFORMATION_MESSAGE );
```

LECTURA

SIMPLE DE DATOS

- Pedir información al usuario: Para entradas sencillas a requerimientos (un nombre, un valor numérico) es posible utilizar la clase `JOptionPane`, que abre una ventana de diálogo y retorna la cadena tecleada por el usuario

```
String strModelo = JOptionPane.showInputDialog( this ,  
"Introduzca el modelo buscado:", "Buscar modelo",  
JOptionPane.QUESTION_MESSAGE );
```

```
if( strModelo != null )
```

```
{
```

```
// el usuario tecleó algo
```

```
}
```

Debe ser null si está en el main

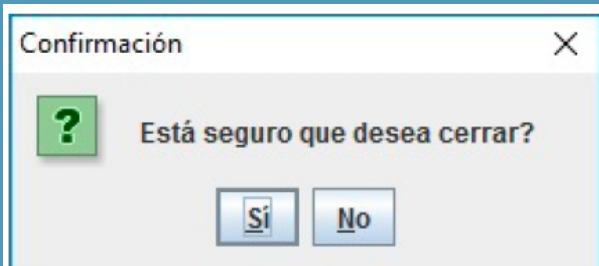
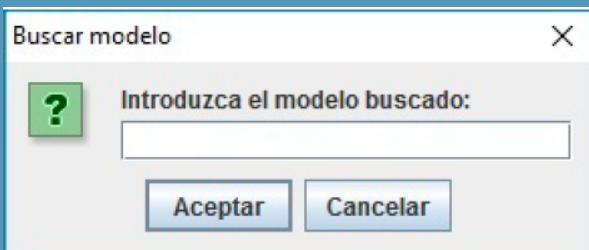
```
int resp = JOptionPane.showConfirmDialog( this , "Está seguro  
que desea cerrar?", "Confirmación",  
JOptionPane.YES_NO_OPTION );
```

```
if( resp == JOptionPane.YES_OPTION )
```

```
{
```

```
// el usuario seleccionó Sí
```

```
}
```





RESPONSABILIDADES

- La ejecución del programa inicia a través de un método llamado `main()` éste se implementa en la clase de la ventana principal del programa y su principal tarea es crear una instancia de la ventana y hacerla visible en la pantalla.

```
public static void main( String[] args ) {  
    InterfazImpuestosCarro interfaz = new InterfazImpuestosCarro();  
    interfaz.setVisible( true );  
}
```

- La creación del modelo del mundo es responsabilidad de la interfaz, específicamente del constructor de la ventana principal

```
public class InterfazImpuestosCarro extends JFrame {  
    ...  
    private CalculadorImpuestos calculador;  
    public InterfazImpuestosCarro() throws Exception {  
        calculador = new CalculadorImpuestos();  
        ...  
    }
```

La signatura debe ser idéntica a esta

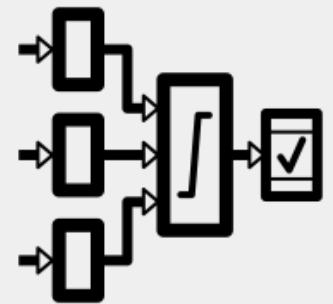
Definimos un atributo de tipo
CalculadorImpuestos, que
permite “hablar” con el mundo



RESPONSABILIDADES

Un panel debe tener los siguientes métodos:

- Constructor
- Método para atender eventos
- Métodos indispensables para permitir el acceso a la información tecleada por el usuario: el programador debe decidir si estos se encargan de hacer las conversiones o si esta labor se la deja a quienes manipulan dicha información
- Métodos para refrescar la información desplegada en el panel, cuyo objetivo es actualizar los datos mostrados al usuario.





RESPONSABILIDADES

Protocolo:

- **Paso 1:** usuario genera un evento, que se convierte en un objeto. El panel que contiene el botón debe reaccionar
- **Paso 2:** el panel reacciona con su método `actionPerformed` solicitando a la ventana principal que ejecute el requerimiento funcional pedido. Si se debe convertir información es responsabilidad del panel hacerlo
- **Paso 3:** la ventana principal completa la información para cumplir con el requerimiento, pidiéndola a los demás paneles. Debe realizar todas las verificaciones y en caso de un error notificar al usuario
- **Paso 4:** el mundo debe hacer una modificación o calcular algún valor. Se utiliza la asociación que tiene la interfaz con el modelo para llamar a los métodos que ayudan a implementar los requerimientos funcionales, cualquier excepción lanzada por los métodos del mundo deben ser atrapadas aquí. Si se pide calcular un valor al terminar este paso deben estar los datos completos para refrescar en la interfaz, en caso de una modificación se ejecuta el paso 5
- **Paso 5:** Retornar los nuevos valores a desplegar
- **Paso 6:** Se pide a todos los paneles que tienen información que pudo haber cambiado que actualicen sus valores, se utilizan para esto los métodos de refresco



RESPONSABILIDADES

Dado el protocolo, es claro, que los paneles deben conocer la ventana principal para ejecutar los métodos que implementan los requerimientos funcionales. Para esto, los constructores de los paneles la deben recibir y la deben guardar en un atributo.

```
public class PanelBusquedas extends JPanel implements ActionListener {  
    public PanelBusquedas( InterfazImpuestosCarro pPrincipal ) {  
        principal = pPrincipal;  
    }  
    public class PanelNavegacion extends JPanel implements ActionListener {  
        public PanelNavegacion( InterfazImpuestosCarro pPrincipal ) {  
            principal = pPrincipal;  
        }  
        public class PanelOpciones extends JPanel implements ActionListener {  
            public PanelOpciones( InterfazImpuestosCarro pPrincipal ) {  
                principal = pPrincipal;  
            }  
            public class PanelVehiculo extends JPanel implements ActionListener {  
                public PanelVehiculo( InterfazImpuestosCarro pPrincipal ) {  
                    add( new PanelNavegacion( pPrincipal ), BorderLayout.SOUTH );  
                }  
            }  
        }
```



RESPONSABILIDADES

- Los métodos que implementan los requerimientos funcionales deben hacer parte de la clase de la ventana principal y tienen como objetivo coordinar los paneles y el modelo del mundo para satisfacer los pedidos del cliente.

```
public void buscarPorLinea( String pLinea ) {  
  
    Vehiculo respuesta = calculador.buscarVehiculoPorLinea( pLinea );  
    if( respuesta == null )  
        JOptionPane.showMessageDialog( this, "No se encontró ningún vehículo de  
        esta línea", "Buscar por línea", JOptionPane.ERROR_MESSAGE );  
    else  
        panelVehiculo.actualizar( respuesta.darMarca(), respuesta.darLinea(),  
        respuesta.darAnio(), respuesta.darPrecio(), respuesta.darRutImagen() );  
}
```

