Tenemos 4 entidades principales para modelar como clases:
- Vehículo.
- Diésel y Gasolina, que heredan de Vehículo.
- Usuario.

La clase Vehículo será quien aporte la mayor parte de los atributos, incluyendo tres atributos que por como está redactado el enunciado, quedan un poco difusos, pero serán necesarios.
Estos tres atributos sería:
- *fechaInicio* (cuando se empieza a trabajar con el coche)
- *fechaTermino* (cuando se termina de trabajar con el coche)
- *entregado* (para indicar si el coche se ha entregado ya al dueño)

Los atributos para las fechas podrían ser Strings para simplificar el ejercicio, pero también se puede usar la clase LocalDate para hacerlo más interesante y garantizar calidad en estos datos.

El atributo *entregado* bastará con usar un tipo de dato boolean.

Este podría ser el código para la clase Vehículo, el cuál puede que necesitemos alterar posteriormente según desarrollamos el ejercicio:

```
package tallerVehiculos;

import java.time.LocalDate;

public abstract class Vehiculo {

	private String placa;
	private String VIN;
	private String marca;
	private String modelo;
	private String anio;
	private String tamanioMotor;
	private String traccion;
	private LocalDate fechaInicio;
	private LocalDate fechaTermino;
	private boolean entregado;

	public Vehiculo(String placa, String vIN, String marca, String modelo, String
anio, String tamanioMotor,
			String traccion) {
		this.placa = placa;
		VIN = vIN;
		this.marca = marca;
		this.modelo = modelo;
		this.anio = anio;
		this.tamanioMotor = tamanioMotor;
		this.traccion = traccion;
```

```
		entregado = false;
	}

	public String getPlaca() {
		return placa;
	}

	public void setPlaca(String placa) {
		this.placa = placa;
	}

	public String getVIN() {
		return VIN;
	}

	public void setVIN(String vIN) {
		VIN = vIN;
	}

	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 String getAnio() {
		return anio;
	}

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

	public String getTamanioMotor() {
		return tamanioMotor;
	}

	public void setTamanioMotor(String tamanioMotor) {
		this.tamanioMotor = tamanioMotor;
	}

	public String getTraccion() {
		return traccion;
	}

	public void setTraccion(String traccion) {
		this.traccion = traccion;
	}

	public void setFechaInicio(LocalDate fechaInicio) {
		this.fechaInicio = fechaInicio;
```

```
	}

	public String getFechaInicio() {
		if (fechaInicio == null)
			return "No Iniciado";
		else
			return String.format("%2d/%2d/%4d", fechaInicio.getDayOfMonth(),
					fechaInicio.getMonthValue(), fechaInicio.getYear());
	}

	public void setFechaTermino(LocalDate fechaTermino) {
		this.fechaTermino = fechaTermino;
	}

	public String getFechaTermino() {
		if (fechaTermino == null)
			return "No Terminado";
		else
			return String.format("%2d/%2d/%4d", fechaTermino.getDayOfMonth(),
					fechaTermino.getMonthValue(),
fechaTermino.getYear());
	}

	public boolean isEntregado() {
		return entregado;
	}

	public void setEntregado(boolean entregado) {
		this.entregado = entregado;
	}

	@Override
	/**
	 *Compara dos objetos para saber si son el mismo Vehículo.<br>
	 *Dos Vehículos se consideran el mismo si coincide su atributo VIN.
	 */
	public boolean equals(Object objeto) {
		if (objeto instanceof Vehiculo) {
			Vehiculo otroVehic = (Vehiculo) objeto;
			return this.VIN.equals(otroVehic.getVIN());
		}
		else
			return false;
	}

}
```

A continuación tenemos las clases Diesel y Gasolina, las cuáles heredan de Vehículo y aportan cada una dos atributos específicos de la entidad que modelan.

```java
package tallerVehiculos;

public class Diesel extends Vehiculo {

	private String tipoBomba;
	private String tipoFiltro;

	public Diesel(String placa, String vIN, String marca, String modelo, String
anio, String tamanioMotor,
			String traccion, String tipoBomba, String tipoFiltro) {
		super(placa, vIN, marca, modelo, anio, tamanioMotor, traccion);
		this.tipoBomba = tipoBomba;
		this.tipoFiltro = tipoFiltro;
	}

	public String getTipoBomba() {
		return tipoBomba;
	}

	public void setTipoBomba(String tipoBomba) {
		this.tipoBomba = tipoBomba;
	}

	public String getTipoFiltro() {
		return tipoFiltro;
	}

	public void setTipoFiltro(String tipoFiltro) {
		this.tipoFiltro = tipoFiltro;
	}

}
```

```java
package tallerVehiculos;

public class Gasolina extends Vehiculo {

	private String flujoGasolina;
	private String tipoBobina;

	public Gasolina(String placa, String vIN, String marca, String modelo, String
anio, String tamanioMotor,
			String traccion, String flujoGasolina, String tipoBobina) {
		super(placa, vIN, marca, modelo, anio, tamanioMotor, traccion);
		this.flujoGasolina = flujoGasolina;
		this.tipoBobina = tipoBobina;
	}

	public String getFlujoGasolina() {
		return flujoGasolina;
	}

	public void setFlujoGasolina(String flujoGasolina) {
		this.flujoGasolina = flujoGasolina;
	}

	public String getTipoBobina() {
		return tipoBobina;
	}

	public void setTipoBobina(String tipoBobina) {
		this.tipoBobina = tipoBobina;
	}

}
```

```
	public void setTipoBobina(String tipoBobina) {
		this.tipoBobina = tipoBobina;
	}

}
```

Por último, la clase Usuario, la cuál es una clase sencilla con un par de atributos.

```
package tallerVehiculos;

public class Usuario {

	private String nombre;
	private String password;

	public Usuario(String nombre, String password) {
		this.nombre = nombre;
		this.password = password;
	}

	public String getNombre() {
		return nombre;
	}

	public void setNombre(String nombre) {
		this.nombre = nombre;
	}

	public String getPassword() {
		return password;
	}

	public void setPassword(String password) {
		this.password = password;
	}

	@Override
	public boolean equals(Object objeto) {
		if (objeto instanceof Usuario) {
			Usuario otroUsuario = (Usuario) objeto;
			return this.nombre.equals(otroUsuario.getNombre());
		}
		else
			return false;
	}

}
```

Estas son las clases que, de momento (seguro surgirán más..), van a componer el Modelo, es decir, la parte que el usuario “no ve”.

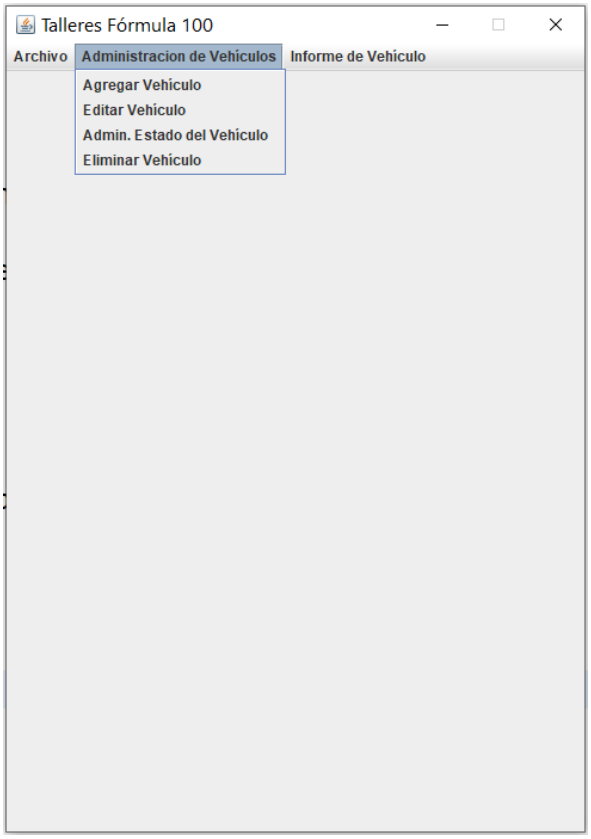
Ahora vamos a iniciar la parte que el usuario “sí ve”, es decir, la Vista. Creamos una clase Jframe que de momento no va a mostrar nada importante todavía.

Al marco le daremos unas dimensiones cualquiera, las que sean, no importan ahora mismo las dimensiones del marco porque por ahora solo queremos que se pueda ver “algo” en pantalla.
Lo que si vamos a incluir ya será la barra de menú (JmenuBar) que nos pide el enunciado.
Esta barra la construimos en un método private que la retornará para que el constructor del Jframe la añada al iniciar programa.
De momento las opciones de la barra de menú no van a tener ninguna funcionalidad, excepto la opción de “Salir” que mediante una clase ActionListener le asignamos la función de poner fin al programa.

```
package tallerVehiculos;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.SwingUtilities;

@SuppressWarnings("serial")
public class Taller extends JFrame{

	public Taller() {
		setTitle("Talleres Fórmula 100");
		setJMenuBar(crearBarraMenu());
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setSize(500, 700);
		setResizable(false);
		setLocationRelativeTo(null);
		setVisible(true);
	}

	private JMenuBar crearBarraMenu() {
		JMenuBar barra = new JMenuBar();

		JMenu archivo = new JMenu("Archivo");
		JMenuItem cerrar = new JMenuItem("Cerrar Sesión");
		JMenuItem salir = new JMenuItem("Salir");
		salir.addActionListener(new AccionSalir());
		archivo.add(cerrar);
		archivo.add(salir);

		JMenu administracion = new JMenu("Administracion de Vehículos");
		JMenuItem agregar = new JMenuItem("Agregar Vehículo");
		JMenuItem editar = new JMenuItem("Editar Vehículo");
		JMenuItem estado = new JMenuItem("Admin. Estado del Vehículo");
		JMenuItem eliminar = new JMenuItem("Eliminar Vehículo");
		administracion.add(agregar);
		administracion.add(editar);
		administracion.add(estado);
		administracion.add(eliminar);
```

```
                JMenu informe = new JMenu("Informe de Vehículo");
                JMenuItem mantenimiento = new JMenuItem("Lista Vehículos mantenimiento");
                JMenuItem entregados = new JMenuItem("Lista Vehículos entregados");
                informe.add(mantenimiento);
                informe.add(entregados);

                barra.add(archivo);
                barra.add(administracion);
                barra.add(informe);

                return barra;
        }

        class AccionSalir implements ActionListener {
                @Override
                public void actionPerformed(ActionEvent e) {
                        System.exit(0);
                }
        }

        public static void main(String[] args) {
                SwingUtilities.invokeLater(new Runnable() {
                        @Override
                        public void run() {
                                new Taller();
                        }
                });
        }

}
```

Esto nos muestra la siguiente interfaz:



Bien, ahora debemos pensar un poco qué aspecto queremos que tenga la interfaz.
Tenemos varias opciones distintas, lo cuál implica mostrar distintos formularios según escoja el usuario.
Algunas opciones son sencillas y podrían resolverse con ventanas emergentes JoptionPane o algo más elaborado como un Jdialog.
Pero otras si van a requerir unos formularios más completos mediante un Jpanel que ofrezca varios campos para introducir datos.
Hay distintas formas de elegir una maquetación para esto, ya según gustos del programador, podemos probar a simplemente poner/quitar paneles en el marco principal según las opciones escogidas.

Ahora pensemos que es lo primero que queremos mostrar al iniciar aplicación. El programa nos pide que un usuario haga login para poder trabajar, por tanto, tiene sentido que el primer panel a mostrar sea un login de usuario.
Esto nos recuerda que vamos a tener que gestionar dos tipos de entidades: Usuarios y Vehículos.
Vamos a centrarnos por ahora en la gestión de usuarios por ser la más sencilla.
El programa tiene que ofrecer la posibilidad de crear nuevos usuarios (sería lógico también poder eliminarlos, pero como no se pide, pues no lo hacemos) y que estos puedan iniciar y cerrar sesión.
Estos usuarios se han de guardar durante el tiempo de ejecución en alguna estructura, por ejemplo en un ArrayList.
Pero también se han de guardar fuera del tiempo de ejecución, es decir, que se guarden en disco para ser recuperados cada vez que se inicie el programa.
Para guardar los datos de un ArrayList en disco tenemos dos opciones:
- Uno. Estructurar los datos en un archivo de texto para ser escritos/leídos cuando sea necesario.
- Dos. “Serializar” el ArrayList y guardarlo en disco en un archivo binario.

Puesto que no tenemos interés en poder editar dichos datos mediante un archivo de texto, la segunda opción será la más rápida y cómoda. Volcar el ArrayList directamente al disco en forma de bits.

Para poder hacer esto, necesitamos que los datos que vamos a volcar sean "serializables". Esto es fácil, basta con pedirle a nuestra clase Usuario que implemente la interfaz Serializable.

```
package tallerVehiculos;

import java.io.Serializable;

public class Usuario implements Serializable{

	private String nombre;
	private String password;

	public Usuario(String nombre, String password) {
		this.nombre = nombre;
		this.password = password;
	}
...
...
...
```

Bien, eso permite guardar objetos Usuario en disco, pero lo que queremos es guardar un ArrayList entero. De hecho, necesitamos crear toda la lógica para introducir Usuarios en el ArrayList, buscarlos, guardarlos en disco… Vamos a crea una nueva clase llamada GestionUsuario que se dedicará a estas tareas y que también implementará la interfaz Serializable, pues será este objeto (que contendrá ArrayList y la lógica para gestionarlo) quien volcaremos a disco para guardarlo:

```
package tallerVehiculos;

import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.io.Serializable;
import java.util.ArrayList;

import javax.swing.JOptionPane;

public class GestionUsuario implements Serializable{

	private ArrayList<Usuario> usuarios;

	public GestionUsuario() {
		usuarios = new ArrayList<Usuario>();
	}

	public boolean nuevoUsuario(Usuario user) {
		if (usuarios.contains(user)) {
			JOptionPane.showMessageDialog(null, "Ya existe un usuario con el
nombre: " + user.getNombre(),
					"Nuevo Usuario", JOptionPane.WARNING_MESSAGE);
			return false;
		}
		else {
```

```
                        usuarios.add(user);
                        return true;
                }
        }

        public boolean validarLogin(Usuario user) {
                int indice = usuarios.indexOf(user);
                if (indice == -1) {
                        JOptionPane.showMessageDialog(null, "No existe usuario con el
nombre: " + user.getNombre(),
                                        "Login Usuario", JOptionPane.WARNING_MESSAGE);
                        return false;
                }
                else {
                        if (usuarios.get(indice).getPassword().equals(user.getPassword()))
                                return true;
                        else {
                                JOptionPane.showMessageDialog(null, "Password es
incorrecto", "Login Usuario",
                                                JOptionPane.WARNING_MESSAGE);
                                return false;
                        }
                }
        }

        public boolean guardarUsuarios() {
                        try {
                                ObjectOutputStream obs = new ObjectOutputStream(new
FileOutputStream("usuarios.bin"));
                                obs.writeObject(this);
                                obs.close();
                                return true;
                        } catch (IOException e) {
                                JOptionPane.showMessageDialog(null, "Error al acceder al
archivo: usuarios.bin",
                                                "Guardar Usuarios", JOptionPane.ERROR_MESSAGE);
                                return false;
                        }
        }

}
```

De momento esa podría ser la clase que gestiona los usuarios. Tiene un método para añadir nuevos usuarios (comprobando que no se repitan), otro para validar el login comprobando que el password coincide con el usuario del mismo nombre y por último un método que se encarga de guardar toda esta estructura en disco, para luego ser recuperable desde otro punto del programa.

Esta clase como hemos dicho, forma parte del Modelo del programa y se comunicará con la Vista, es decir, con la interfaz visible para crear y loguear a usuarios.

Para facilitar esta interacción entre Modelo y Vista y tener mejor modulado el código, vamos a crear una clase llamada Controlador para que haga de nexo entre Modelo y Vista.

Además se encargará de otras gestiones como decidir cuándo guardar o recuperar los datos guardados en disco, que panel/formulario se ha de mostrar en pantalla, etc..

Esta clase irá creciendo según avancemos en el programa, de momento, va a "controlar" la gestión de usuarios, así que uno de sus atributos será precisamente nuestra clase GestionUsuario.

Otro atributo será una referencia a la clase Jframe principal que hemos llamado Taller. Mediante esta referencia el Controlador tendrá acceso al marco principal y así decidir qué ocurre en pantalla.
A su vez, la clase Taller tendrá también una referencia apuntando a Controlador obteniendo así comunicación en ambos sentidos.
Para agilizar la creación de este vínculo, Controlador recibe la referencia a Taller en su constructor.

Controlador tendrá también como atributos los distintos paneles que vayamos creando para cada una de las opciones que ofrece el programa, de este modo el Controlador podrá colocarlos en el marco principal cuando corresponda e interactuar con ellos.
Parte de esa interacción será asignarles "acciones" a los botones de los formularios de cada panel, así que al final de todo esta clase va a contener bastante código.

Vamos a ver una primera versión de Controlador, la cuál ya manda colocar un panel para que los usuarios puedan iniciar sesión.
Además cuenta con un método que intenta recuperar usuarios registrados guardados en disco.

Obviamente las primeras veces que iniciemos el programa, puesto que aún no tenemos datos guardados, nos lanzará el mensaje de que no ha encontrado dichos datos en disco.

```java
package tallerVehiculos;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;

import javax.swing.JOptionPane;

import paneles.*;

public class Controlador {

	//Datos Modelo
	private GestionUsuario gestorUsuarios;
	//Datos Vista
	private Taller taller; //JFrame principal
	//Paneles para la Vista
	private UsuarioLogin panelLogin;

	public Controlador (Taller t) {
		taller = t;
		iniciarGestorUsuarios();
		panelLogin = new UsuarioLogin();
		mostrarPanelLogin();
	}

	/**
	 * Establece en el marco principal que muestre
	 * el panel de inicio de sesion de usuarios.
	 */
	public void mostrarPanelLogin() {
		panelLogin.reset();
		taller.setContentPane(panelLogin);
		taller.pack();
		taller.setLocationRelativeTo(null);
	}

	/**
	 * Intentará recuperar los datos de gestion de
	 * usuarios guardados en disco.<br>Si no puede
	 * recuperarlos, ya sea porque no existe el archivo
	 * o porque no tiene acceso a él o está corrupto,
	 * iniciará un Gestor de Usuarios nuevo.
	 */
	private void iniciarGestorUsuarios() {
		File usuarios = new File("usuarios.bin");
		if (usuarios.exists()) {
			ObjectInputStream ois;
			try {
				ois = new ObjectInputStream(new FileInputStream(usuarios));
				Object datos = ois.readObject();
				if (datos instanceof GestionUsuario) {
					gestorUsuarios = (GestionUsuario)datos;
				}
				else {
					mensajeEmergente("Datos de usuarios guardados no son
válidos", "Gestion Usuarios");
					gestorUsuarios = new GestionUsuario();
				}
				ois.close();
```

```
                    } catch (IOException e) {
                          mensajeEmergente("No se pudo acceder a:\n" +
usuarios.getAbsolutePath(), "Gestion Usuarios");
                          gestorUsuarios = new GestionUsuario();
                    } catch (ClassNotFoundException e) {
                          mensajeEmergente("Datos de usuarios guardados no son
válidos", "Gestion Usuarios");
                          gestorUsuarios = new GestionUsuario();
                    }

             }
             else {
                    mensajeEmergente("No hay datos de Usuarios guardados", "Gestion
Usuarios");
                    gestorUsuarios = new GestionUsuario();
             }
      }

      /**
       * Puesto que en este código se va a recurrir varias veces a mostrar
       * JOptionPane informativos en pantalla, vale la pena simplificar el código
       * creando este método para no repetirlo decenas de veces
       * @param mensaje Texto informativo que mostrará el JOptionPane
       * @param titulo Título descriptivo del JOptionPane
       */
      private void mensajeEmergente(String mensaje, String titulo) {
            JOptionPane.showMessageDialog(null, mensaje, titulo,
JOptionPane.WARNING_MESSAGE);
      }

}
```

A la clase Taller le hacemos un par de cambios.

Añadimos una referencia a Controlador para conseguir comunicación bidireccional entre Vista y Controlador.

Añadimos un atributo boolean para controlar si hay un usuario logueado o no. La mayoría de opciones disponibles en el JmenuBar no deben poder ejecutarse si no ha iniciado sesión ningún usuario, mediante este atributo podremos saber cuando permitir que se ejecuten y cuando no.

Eliminamos las dimensiones que habíamos puesto inicialmente al marco. Ya no son necesarias pues estas dimensiones ahora dependerán del panel que se esté mostrando en cada momento.

```java
package tallerVehiculos;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.SwingUtilities;

@SuppressWarnings("serial")
public class Taller extends JFrame{

	public boolean usuarioLogueado;
	private Controlador controlador;

	public Taller() {
		usuarioLogueado = false;
		controlador = new Controlador(this);
		setTitle("Talleres Fórmula 100");
		setJMenuBar(crearBarraMenu());
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		pack();
		setResizable(false);
		setLocationRelativeTo(null);
		setVisible(true);

	}

	private JMenuBar crearBarraMenu() {
		JMenuBar barra = new JMenuBar();

		JMenu archivo = new JMenu("Archivo");
		JMenuItem cerrar = new JMenuItem("Cerrar Sesión");
		JMenuItem salir = new JMenuItem("Salir");
		salir.addActionListener(new AccionSalir());
		archivo.add(cerrar);
		archivo.add(salir);

		JMenu administracion = new JMenu("Administracion de Vehículos");
		JMenuItem agregar = new JMenuItem("Agregar Vehículo");
		JMenuItem editar = new JMenuItem("Editar Vehículo");
		JMenuItem estado = new JMenuItem("Admin. Estado del Vehículo");
		JMenuItem eliminar = new JMenuItem("Eliminar Vehículo");
		administracion.add(agregar);
		administracion.add(editar);
		administracion.add(estado);
		administracion.add(eliminar);

		JMenu informe = new JMenu("Informe de Vehículo");
		JMenuItem mantenimiento = new JMenuItem("Lista Vehículos mantenimiento");
		JMenuItem entregados = new JMenuItem("Lista Vehículos entregados");
		informe.add(mantenimiento);
		informe.add(entregados);

		barra.add(archivo);
		barra.add(administracion);
		barra.add(informe);

		return barra;
```

```
	}

	class AccionSalir implements ActionListener {
		@Override
		public void actionPerformed(ActionEvent e) {
			System.exit(0);
		}
	}

	public static void main(String[] args) {
		SwingUtilities.invokeLater(new Runnable() {
			@Override
			public void run() {
				new Taller();
			}
		});
	}
}
```

Bien, antes hemos visto que el Controlador ya ordenaba mostrar un panel para que el usuario pueda iniciar sesión.
Vamos a ver el código de este panel, que consisten en dos etiquetas, dos campos de texto (uno específico para insertar contraseñas) y dos botones. Un botón será para iniciar sesión y el otro será para crear un usuario nuevo

Ahora mismo dichos botones no tienen funcionalidad todavía, el código de este panel lo único que hace es mostrar los elementos visuales en pantalla.

Para conseguir una maquetación más o menos ordenada, se hace una combinación de varios subpaneles con distintos layouts según el caso y bordes combinados para crear espacio de relleno (padding) entre los elementos y el marco y además mostrar un borde con título.

Puede parecer complicado a simple vista, pero en realidad no lo es, y de todos modos estos son detalles estéticos y tampoco vale la pena dedicarle demasiado tiempo.

Un detalle a destacar es que los atributos de esta clase (los botones y los campos de texto) son PÚBLICOS en lugar de PRIVADOS que es lo habitualmente recomendado por el “principio de encapsulamiento” en las buenas prácticas de programación.

El motivo de declararlos “public” es para agilizar el código y no tener que crear métodos para acceder a dichos atributos. En este caso, sería

redundante porque es una clase destinada a ser utilizada únicamente por el Controlador, quien necesita acceso total a todos estos atributos y él mismo ya les da un tratamiento de “private”.

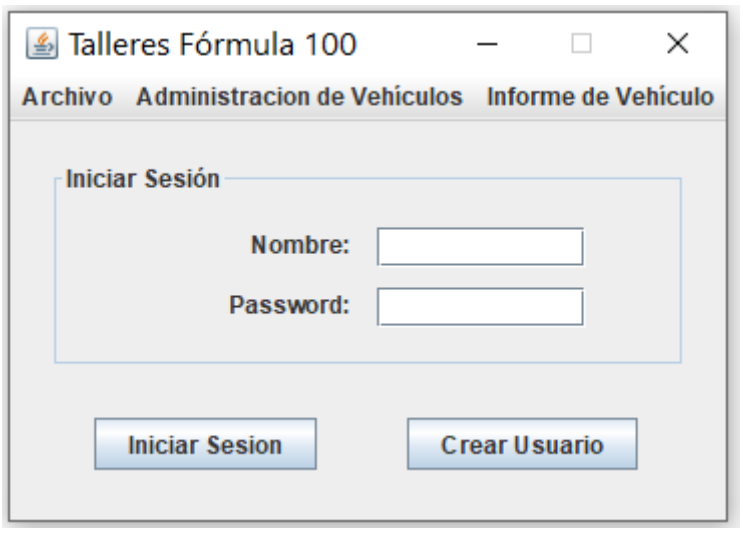
```
package paneles;

import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.GridLayout;

import javax.swing.BorderFactory;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JTextField;

public class UsuarioLogin extends JPanel{

	public JTextField campoNombre;
	public JPasswordField campoPassword;
	public JButton botonIniciar;
	public JButton botonCrear;

	public UsuarioLogin() {
		campoNombre = new JTextField(10);
		campoPassword = new JPasswordField(10);
		botonIniciar = new JButton("Iniciar Sesion");
		botonCrear = new JButton("Crear Usuario");

		setLayout(new BorderLayout());
		add(new PanelCentro(), BorderLayout.CENTER);
		add(new PanelSur(), BorderLayout.SOUTH);
	}

	public void reset() {
		campoNombre.setText(null);
		campoPassword.setText(null);
	}

	private class PanelCentro extends JPanel {
		public PanelCentro() {
			setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
			add(new PanelEtiquetaYCampo("Nombre: ", campoNombre));
			add(new PanelEtiquetaYCampo("Password: ", campoPassword));
			setBorder(BorderFactory.createCompoundBorder(
					BorderFactory.createEmptyBorder(20, 20, 10, 20),
					BorderFactory.createCompoundBorder(

	BorderFactory.createTitledBorder("Iniciar Sesión"),
						BorderFactory.createEmptyBorder(10, 10,
10, 10))));
		}
	}

	private class PanelEtiquetaYCampo extends JPanel {
		public PanelEtiquetaYCampo(String txtEtiqueta, JTextField campo) {
			setLayout(new GridLayout(1, 2));
			JPanel panelEtiq = new JPanel();
			panelEtiq.setLayout(new FlowLayout(FlowLayout.RIGHT));
```

```
                    panelEtiq.add(new JLabel(txtEtiqueta));
                    JPanel panelCampo = new JPanel();
                    panelCampo.setLayout(new FlowLayout(FlowLayout.LEFT));
                    panelCampo.add(campo);
                    add(panelEtiq);
                    add(panelCampo);
            }
      }

      private class PanelSur extends JPanel {
            public PanelSur() {
                    setLayout(new BoxLayout(this, BoxLayout.X_AXIS));
                    JPanel panelIniciar = new JPanel();
                    panelIniciar.add(botonIniciar);
                    JPanel panelCrear = new JPanel();
                    panelCrear.add(botonCrear);
                    add(panelIniciar);
                    add(panelCrear);
                    setBorder(BorderFactory.createEmptyBorder(10, 20, 20, 20));
            }
      }

}
```

Este, y el resto de paneles, se van a declarar en un package separado de las otras clases.
No es obligatorio y cada cuál puede organizar sus clases como prefiera.

Vemos como la maquetación se ha creado con un BorderLayout, donde el panel central a su vez creamos dos subpaneles uno encima de otro.
Cada uno de estos subpaneles tiene sendos subpaneles, uno para las etiquetas y otro para los campos. Estos los maquetamos con GridLayout para conseguir que el subpanel con la etiqueta y el subpanel con el campo de texto, tengan exactamente el mismo tamaño.

Abajo, en el panel sur, componemos otros paneles para mostrar los botones centrados.
Esto nos da este resultado en pantalla:



Es un panel sencillo, pero no necesitamos más. El siguiente paso es dar funcionalidad a los botones y para ello nos vamos a la clase Controlador donde crearemos unos ActionListener para asignárselos a cada botón y actúen según corresponda.

Puesto que no tenemos usuarios, vamos a pensar en la "acción" para crear usuarios.
La mecánica sería que el usuario introduzca nombre y password y al pulsar el botón "Crear Usuario", estos datos se recuperen y se añada un objeto Usuario al ArrayList.
Antes de añadir, podemos pedir al usuario que confirme de nuevo el password que quiere registrar y una vez confirmado, entonces si creamos el usuario, lo almacenamos en el ArrayList y lo guardamos en disco.

Una vez se ha guardado el nuevo usuario, automáticamente iniciará sesión. Esto implica que el atributo boolean de la clase taller para saber si hay un usuario logueado lo cambiamos a true y además cambiaremos el panel de inserción de datos por otro panel sencillo de bienvenida.

La clase Controlador quedaría así con la inclusión de la clase ActionListener.
Además añadimos un método que se encargará de asignar estas acciones a los botones de los paneles. Según vayamos creando nuevos paneles y nuevas acciones, este método irá creciendo.

Marcamos en negrita los cambios importantes:

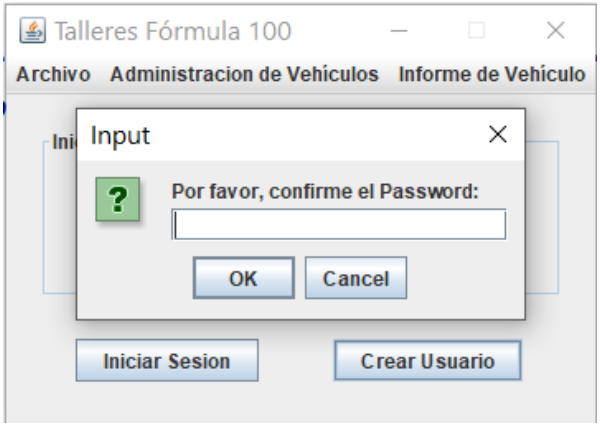
```
public class Controlador {

	//Datos Modelo
	private GestionUsuario gestorUsuarios;
	//Datos Vista
	private Taller taller; //JFrame principal
	//Paneles para la Vista
	private UsuarioLogin panelLogin;

	public Controlador (Taller t) {
		taller = t;
		iniciarGestorUsuarios();
		panelLogin = new UsuarioLogin();
		asignarAcciones();
		mostrarPanelLogin();
	}

	private void asignarAcciones() {
		panelLogin.botonCrear.addActionListener(new AccionCrearUsuario());
	}

	/**
	 * Establece en el marco principal que muestre
	 * el panel de inicio de sesion de usuarios.
	 */
```

```
	public void mostrarPanelLogin() {
		panelLogin.reset();
		taller.setContentPane(panelLogin);
		taller.pack();
		taller.setLocationRelativeTo(null);
	}

	/**
	 * Intentará recuperar los datos de gestion de
	 * usuarios guardados en disco.<br>Si no puede
	 * recuperarlos, ya sea porque no existe el archivo
	 * o porque no tiene acceso a él o está corrupto,
	 * iniciará un Gestor de Usuarios nuevo.
	 */
	private void iniciarGestorUsuarios() {
		File usuarios = new File("usuarios.bin");
		if (usuarios.exists()) {
			ObjectInputStream ois;
			try {
				ois = new ObjectInputStream(new FileInputStream(usuarios));
				Object datos = ois.readObject();
				if (datos instanceof GestionUsuario) {
					gestorUsuarios = (GestionUsuario)datos;
				}
				else {
					mensajeEmergente("Datos de usuarios guardados no son válidos", "Gestion
Usuarios");
					gestorUsuarios = new GestionUsuario();
				}
				ois.close();
			} catch (IOException e) {
				mensajeEmergente("No se pudo acceder a:\n" + usuarios.getAbsolutePath(), "Gestion
Usuarios");
				gestorUsuarios = new GestionUsuario();
			} catch (ClassNotFoundException e) {
				mensajeEmergente("Datos de usuarios guardados no son válidos", "Gestion Usuarios");
				gestorUsuarios = new GestionUsuario();
			}

		}
		else {
			mensajeEmergente("No hay datos de Usuarios guardados", "Gestion Usuarios");
			gestorUsuarios = new GestionUsuario();
		}
	}

	// +++++++++++++++++++++ Clases Action Listener  +++++++++++++++++++++
	class AccionCrearUsuario implements ActionListener {
		@Override
		public void actionPerformed(ActionEvent e) {
			if (panelLogin.campoNombre.getText().isEmpty() || panelLogin.campoPassword.getPassword().length
== 0)
				mensajeEmergente("Ha de rellenar los dos campos", "Crear Usuario");
			else {
				String nombre = panelLogin.campoNombre.getText();
				String password = String.copyValueOf(panelLogin.campoPassword.getPassword());
				String confirm = JOptionPane.showInputDialog(null, "Por favor, confirme el
Password:");
				if (password.equals(confirm)) {
					if (gestorUsuarios.nuevoUsuario(new Usuario(nombre, password))) {
						mensajeEmergente("Nuevo usuario creado.\nSesión iniciada",
								"Crear Usuario");
						gestorUsuarios.guardarUsuarios();
						taller.usuarioLogueado = true;
						taller.setContentPane(new PanelBienvenida(nombre));
						taller.pack();
						taller.setLocationRelativeTo(null);
					}
				}
				else
					mensajeEmergente("El password de registro no coincide", "Crear Usuario");
			}

		}
	}

	/**
	 * Puesto que en este código se va a recurrir varias veces a mostrar
	 * JOptionPane informativos en pantalla, vale la pena simplificar el código
	 * creando este método para no repetirlo decenas de veces
	 * @param mensaje Texto informativo que mostrará el JOptionPane
	 * @param titulo Título descriptivo del JOptionPane
	 */
	private void mensajeEmergente(String mensaje, String titulo) {
		JOptionPane.showMessageDialog(null, mensaje, titulo, JOptionPane.WARNING_MESSAGE);
	}

}
```

El panel de bienvenida simplemente muestra un saludo personalizado con el nombre del usuario que ha iniciado sesión y un par de líneas de texto. Para hacerlo ligeramente más vistoso, usamos algo de código HTML para mostrar el nombre del usuario subrayado y con un color distinto al del texto.

El resto del texto, aumentamos un poco la fuente con el método setFont()-

```
package paneles;

import java.awt.Font;

import javax.swing.BorderFactory;
import javax.swing.BoxLayout;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class PanelBienvenida extends JPanel{

	public PanelBienvenida(String nombreUsuario) {
		setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));

		JPanel panelTitulo = new JPanel();
		JLabel titulo = new JLabel("<html><p>Bienvenido <u style=\"color: blue\">"
		+ nombreUsuario + "</u></p></html>");
		titulo.setFont(new Font("Verdana", Font.BOLD, 32));
		panelTitulo.add(titulo);

		JPanel panelLinea1 = new JPanel();
		JLabel linea1 = new JLabel("Escoge una de las opciones disponibles en las barras de menú
superior.");
		linea1.setFont(new Font("Verdana", Font.PLAIN, 22));
		panelLinea1.add(linea1);

		JPanel panelLinea2 = new JPanel();
		JLabel linea2 = new JLabel("Puedes Cerrar Sesión desde el menú Archivo.");
		linea2.setFont(new Font("Verdana", Font.PLAIN, 22));
		panelLinea2.add(linea2);

		add(panelTitulo);
		add(panelLinea1);
		add(panelLinea2);

		setBorder(BorderFactory.createCompoundBorder(BorderFactory.createEmptyBorder(20, 20, 20,
20),
				BorderFactory.createCompoundBorder(BorderFactory.createRaisedBevelBorder(),
					BorderFactory.createEmptyBorder(10, 10, 10, 10))));
	}

}
```



Tras crear usuario y confirmar password, tenemos una calurosa bienvenida.

Ok. Podemos crear un nuevo Usuario e iniciar sesión con él. Respecto a la gestión de Usuarios, tenemos dos tareas pendientes:
- Iniciar Sesión con un Usuario ya existente.
- Cerrar Sesión.

Iniciar Sesión lo hacemos con el botón del formulario UsuarioLogin del mismo nombre. Tenemos que escribir una nueva clase ActionListener para tal cometido. Dicha acción tendrá que recoger los datos introducidos por el usuario y comprobar si existe dicho Usuario en el ArrayList.
Si existe, se iniciará sesión.
Gran parte del código se asemejará a la accion escrita para Crear Usuario.

Esta es la clase interna que tenemos que incluir dentro de la clase Controlador:

```
	class AccionIniciarSesion implements ActionListener {
		@Override
		public void actionPerformed(ActionEvent e) {
			if (panelLogin.campoNombre.getText().isEmpty() ||
panelLogin.campoPassword.getPassword().length == 0)
				mensajeEmergente("Ha de rellenar los dos campos", "Crear
Usuario");
			else {
				String nombre = panelLogin.campoNombre.getText();
				String password =
String.copyValueOf(panelLogin.campoPassword.getPassword());
				if (gestorUsuarios.validarLogin(new Usuario(nombre,
password))) {
					taller.usuarioLogueado = true;
					taller.setContentPane(new PanelBienvenida(nombre));
					taller.pack();
					taller.setLocationRelativeTo(null);
				}
			}
		}
	}
```

Fijémonos que en el caso de que el Usuario introducido **no** sea válido, no estamos realizando ninguna acción. Sería lógico que mostrásemos algún tipo de mensaje, pero en realidad no es necesario porque si lo recordamos bien, el método *validarLogin()* de nuestra clase GestionUsuario, ya se encarga de mostrar mensajes en el caso de que el nombre o el password, no sean correctos.

Bien, para que nuestro botón realice esta acción que acabamos de escribir, tenemos que asignársela. Recordemos que estas asignaciones de acciones a botones, las estamos agrupando en un método específico:

```
	private void asignarAcciones() {
		panelLogin.botonCrear.addActionListener(new AccionCrearUsuario());
		panelLogin.botonIniciar.addActionListener(new AccionIniciarSesion());
	}
```

Por si no está claro el motivo de crear este método, es simplemente por reestructurar el código de forma un poco más elegante y que sea más legible a simple vista.
Veremos que al final del programa este método va a tener varias líneas. Estas líneas se podrían poner directamente en el constructor de la clase Controlador, pero esto crearía un constructor muy extenso.
Es mejor agrupar estas instrucciones en un método, con un nombre descriptivo e invocar dicho método en el constructor.
De este modo, tenemos un constructor más legible, que con pocas líneas vemos claramente qué es lo que está haciendo.

Con estos cambios en el código ya tenemos la función de Iniciar Sesión implementada.
Ya solo nos falta la acción de Cerrar Sesión.

Esta acción la realiza uno de los item declarados en la barra de menú del Jframe en la clase Taller.
Esto significa que aunque escribamos la clase ActionListener dentro de la clase Controlador como hemos hecho hasta ahora, será dentro de la clase Taller donde la asignemos al JmenuItem correspondiente.

Podríamos escribir dicha acción dentro de la clase Taller, de hecho, la acción de Salir del sistema es ahí donde la escribimos.
Pero para evitar engrosar la clase Taller, podemos seguir escribiendo las clases de ActionListener dentro de Controlador.

Puesto que la clase Taller tiene una referencia a Controlador, podemos fácilmente asignar una acción escrita en Controlador, a un botón o JmenuItem creado en la clase Taller.

Este sería el código para Cerrar Sesión. Muy sencillo en realidad, primero comprobar si realmente hay una sesión iniciad consultando el boolean que nos lo indica. Si está iniciada, asignamos valor false a dicho boolean y volvemos a mostrar la pantalla de login.
Con eso la sesión quedaría cerrada.

```
class AccionCerrarSesion implements ActionListener {
	@Override
	public void actionPerformed(ActionEvent e) {
		if (taller.usuarioLogueado) {
			taller.usuarioLogueado = false;
			mostrarPanelLogin();
		}
		else
			mensajeEmergente("Ha de iniciar sesión para acceder a esta
función.",
					"Opción no disponible");
	}
}
```

Recordamos, esta clase la escribimos dentro de la clase Controlador. Pero es en la clase Taller donde la asignamos. Lo hacemos dentro del método que construye el JmenuBar.

```
	private JMenuBar crearBarraMenu() {
		JMenuBar barra = new JMenuBar();

		JMenu archivo = new JMenu("Archivo");
		JMenuItem cerrar = new JMenuItem("Cerrar Sesión");
		cerrar.addActionListener(controlador.new AccionCerrarSesion());
		JMenuItem salir = new JMenuItem("Salir");
		salir.addActionListener(new AccionSalir());
		archivo.add(cerrar);
		archivo.add(salir);

		JMenu administracion = new JMenu("Administracion de Vehículos");
		JMenuItem agregar = new JMenuItem("Agregar Vehículo");
		JMenuItem editar = new JMenuItem("Editar Vehículo");
		JMenuItem estado = new JMenuItem("Admin. Estado del Vehículo");
		JMenuItem eliminar = new JMenuItem("Eliminar Vehículo");
		administracion.add(agregar);
		administracion.add(editar);
		administracion.add(estado);
		administracion.add(eliminar);

		JMenu informe = new JMenu("Informe de Vehículo");
		JMenuItem mantenimiento = new JMenuItem("Lista Vehículos mantenimiento");
		JMenuItem entregados = new JMenuItem("Lista Vehículos entregados");
		informe.add(mantenimiento);
		informe.add(entregados);
```

```
            barra.add(archivo);
            barra.add(administracion);
            barra.add(informe);

            return barra;
        }
```

Y con esto ya tenemos completada la parte fácil de nuestro programa, la gestión de Usuarios.
Ahora hay que acometer la parte más, si no compleja, sí más extensa: la gestión de Vehículos.

La primera función que hemos de implementar es la de Agregar Vehículos. Vamos a comenzar por la parte de la Vista, es decir, construiremos un panel con todos los componentes necesarios para que el usuario pueda introducir los datos de un Vehículo y escoger si es Diesel o Gasolina.

Podemos dividir el panel en dos partes principales. En una estarán los campos para introducir los atributos comunes a todos los Vehículos: placa, VIN, marca,…
En la otra parte estarán los atributos específicos según el tipo de vehículo. Podemos usar unos RadioButton para que el usuario escoja el tipo de Vehículo. Al elegir uno u otro, activaremos y desactivaremos los campos de cada tipo según la elección.

Para la introducción de datos se pueden usar simples campos JtextField donde el usuario escriba lo que quiera. Aunque algunos datos puede interesar restringir los valores: año de fabricación (desde 2019 hacia atrás), tipo de tracción (4x2 o 4x4) o tipo de flujo (Carburador o Inyección) será más interesante si configuramos unos ComboBox para que el usuario tenga que ceñirse a unos valores preestablecidos.

El formulario tendrá dos botones, Aceptar y Cancelar

Vamos a ver ahor la clase Jpanel con la que vamos a construir este formulario. Esta clase incorpora otras subclases internas para construir distintos subpaneles con los que conformar el panel principal y obtener un formulario con una maquetación mínimamente ordenada.
También incorpora dos ActionListener que son asignados a los RadioButton. Estas acciones se activan cuando el usuario elige un

RadioButton y lo que hacen es activar y desactivar los campos de datos específicos según la elección realizada.

Una peculiaridad de esta clase es que las opciones de Agregar Vehículo y Editar Vehículo van a poder usar el mismo formulario.
En lugar de escribir dos veces el código de un mismo formulario, podemos hacer que uno solo sea capaz de adaptarse a ambas opciones.
Para ello usaremos un boolean. Cuando tenga valor **true** el formulario sabrá que estamos Editando un Vehículo ya existente, el cuál le pasaremos mediante un método y de este modo adaptará su comportamiento a cada situación.

De momento vamos a poner el código necesario para Agregar Vehículos. Ojo porque es una clase bastante extensa debido a la cantidad de datos que ha de recoger. Y luego tendremos que añadir más código para cuando queramos Editar Vehículos.

```
package paneles;

import java.awt.BorderLayout;
import java.awt.Component;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.time.LocalDate;

import javax.swing.BorderFactory;
import javax.swing.BoxLayout;
import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.JTextField;

import tallerVehiculos.Vehiculo;

public class PanelDatosVehiculo extends JPanel{

	public boolean editando;
	//Atributos comunes de Vehiculos
	private JTextField campoPlaca;
	private JTextField campoVIN;
	private JTextField campoMarca;
	private JTextField campoModelo;
	private JComboBox<String> comboAnio;
	private JTextField campoTamanioMotor;
	private JComboBox<String> comboTraccion;
	//Atributos para Gasolina
	private JComboBox<String> comboTipoFlujo;
	private JTextField campoBobina;
	//Atributos para Diesel
	private JTextField campoTipoBomba;
	private JTextField campoTipoFiltro;
	//botones
```

```
	public JButton botonAceptar;
	public JButton botonCancelar;
	//RadioButton para escoger tipo de Vehiculo
	private JRadioButton radioGasolina;
	private JRadioButton radioDiesel;

	public PanelDatosVehiculo(Vehiculo v) {
		iniciarComponentes();
		setLayout(new BorderLayout());

		JPanel centro = new JPanel();
		centro.setLayout(new BoxLayout(centro, BoxLayout.Y_AXIS));
		centro.add(new PanelDatosComunes());
		centro.add(new PanelTipoVehiculo());
		centro.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

		JPanel sur = new JPanel();
		sur.setLayout(new BoxLayout(sur, BoxLayout.X_AXIS));
		JPanel surIzq = new JPanel();
		surIzq.add(botonAceptar);
		JPanel surDer = new JPanel();
		surDer.add(botonCancelar);
		sur.add(surIzq);
		sur.add(surDer);

		add(centro, BorderLayout.CENTER);
		add(sur, BorderLayout.SOUTH);
		sur.setBorder(BorderFactory.createEmptyBorder(0, 10, 10, 10));
	}

	private void iniciarComponentes() {
		campoPlaca = new JTextField(10);
		campoVIN = new JTextField(10);
		campoMarca = new JTextField(6);
		campoModelo = new JTextField(6);
		//Aceptamos coches desde el año 1920 hasta año actual
		comboAnio = new JComboBox<String>();
		for (int i = LocalDate.now().getYear(); i >= 1920; i--)
			comboAnio.addItem(Integer.toString(i));
		campoTamanioMotor = new JTextField(6);
		//Solo hay dos tipos de traccion para escoger
		comboTraccion = new JComboBox<String>(new String[] {"4x2", "4x4"});
		//Solo dos tipos de flujo de combustible para vehiculos Gasolina
		comboTipoFlujo = new JComboBox<String>(new String[] {"Carburador", "Inyección"});
		campoBobina = new JTextField(8);
		campoTipoBomba = new JTextField(8);
		campoTipoFiltro = new JTextField(8);

		botonAceptar = new JButton("Aceptar");
		botonCancelar = new JButton("Cancelar");

		radioGasolina = new JRadioButton("Gasolina", true); //Seleccionado por defecto
		seleccionarGasolina();
		radioGasolina.addActionListener(new AccionElegirGasolina());
		radioDiesel = new JRadioButton("Diesel");
		radioDiesel.addActionListener(new AccionElegirDiesel());
		ButtonGroup grupo = new ButtonGroup();
		grupo.add(radioGasolina);
		grupo.add(radioDiesel);
	}

	private void seleccionarGasolina() {
		//Activamos campos de Gasolina
		comboTipoFlujo.setEnabled(true);
		campoBobina.setEditable(true);
		//Desactivamos campos Diesel
		campoTipoBomba.setEditable(false);
		campoTipoFiltro.setEditable(false);
```

```
	}

	private void seleccionarDiesel() {
		//Desactivamos campos de Gasolina
		comboTipoFlujo.setEnabled(false);
		campoBobina.setEditable(false);
		//Activamos campos Diesel
		campoTipoBomba.setEditable(true);
		campoTipoFiltro.setEditable(true);
	}

	public void resetCampos() {
		editando = false;
		campoPlaca.setText(null);
		campoVIN.setText(null);
		campoMarca.setText(null);
		campoModelo.setText(null);
		comboAnio.setSelectedIndex(0);
		campoTamanioMotor.setText(null);
		comboTraccion.setSelectedIndex(0);
		comboTipoFlujo.setSelectedIndex(0);
		campoBobina.setText(null);
		campoTipoBomba.setText(null);
		campoTipoFiltro.setText(null);
	}

	class PanelDatosComunes extends JPanel {
		public PanelDatosComunes() {
			setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));

			JPanel panel1 = new JPanel();
			panel1.add(new PanelEtiqYCompon("Nº Placa: ", campoPlaca));
			panel1.add(new PanelEtiqYCompon("Nº VIN: ", campoVIN));
			JPanel panel2 = new JPanel();
			panel2.add(new PanelEtiqYCompon("Marca: ", campoMarca));
			panel2.add(new PanelEtiqYCompon("Modelo: ", campoModelo));
			panel2.add(new PanelEtiqYCompon("Año: ", comboAnio));
			JPanel panel3 = new JPanel();
			panel3.add(new PanelEtiqYCompon("Tamaño Motor: ", campoTamanioMotor));
			panel3.add(new PanelEtiqYCompon("Tipo de Tracción: ", comboTraccion));

			add(panel1); add(panel2); add(panel3);

			setBorder(BorderFactory.createCompoundBorder(
					BorderFactory.createTitledBorder("Datos Vehículo"),
					BorderFactory.createEmptyBorder(5, 5, 5, 5)));
		}
	}

	class PanelTipoVehiculo extends JPanel {
		public PanelTipoVehiculo() {
			setLayout(new BoxLayout(this, BoxLayout.X_AXIS));

			JPanel panelGasolina = new JPanel();
			panelGasolina.setLayout(new BoxLayout(panelGasolina, BoxLayout.Y_AXIS));
			panelGasolina.add(radioGasolina);
			panelGasolina.add(new PanelEtiqYCompon("Tipo de Flujo: ",
comboTipoFlujo));
			panelGasolina.add(new PanelEtiqYCompon("Tipo de Bobina: ", campoBobina));

			JPanel panelDiesel = new JPanel();
			panelDiesel.setLayout(new BoxLayout(panelDiesel, BoxLayout.Y_AXIS));
			panelDiesel.add(radioDiesel);
			panelDiesel.add(new PanelEtiqYCompon("Tipo de Bomba: ", campoTipoBomba));
			panelDiesel.add(new PanelEtiqYCompon("Tipo de Filtro: ",
campoTipoFiltro));

			add(panelGasolina); add(panelDiesel);

			setBorder(BorderFactory.createCompoundBorder(
```

```
                                BorderFactory.createTitledBorder("Tipo de Vehículo"),
                                BorderFactory.createEmptyBorder(5, 5, 5, 5)));
                }
        }

        class PanelEtiqYCompon extends JPanel {
                public PanelEtiqYCompon(String etiq, Component compo) {
                        add(new JLabel(etiq));
                        add(compo);
                        setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
                }
        }

        // +++++ Clase ActionListener +++++
        class AccionElegirGasolina implements ActionListener {
                @Override
                public void actionPerformed(ActionEvent e) {
                        seleccionarGasolina();
                }
        }

        class AccionElegirDiesel implements ActionListener {
                @Override
                public void actionPerformed(ActionEvent e) {
                        seleccionarDiesel();
                }
        }

}
```

Una vez tenemos escrita esta clase, tenemos que volver a la clase Controlador, añadir como atributo a esta nueva clase:

```
public class Controlador {

        //Datos Modelo
        private GestionUsuario gestorUsuarios;
        //Datos Vista
        private Taller taller; //JFrame principal
        //Paneles para la Vista
        private UsuarioLogin panelLogin;
        private PanelDatosVehiculo panelDatVehi;
```

En el constructor inicializamos este panel y así quedará ya disponible para cuando lo necesitemos:

```
        public Controlador (Taller t) {
                taller = t;
                iniciarGestorUsuarios();
                iniciarGestorVehiculos();
                panelLogin = new UsuarioLogin();
                panelDatVehi = new PanelDatosVehiculo();
                asignarAcciones();
                mostrarPanelLogin();
        }
```

Y además tenemos que crear el ActionListener que active este panel.
Lo que hará será resetear todos los campos para dejarlos en blanco y listos para crear un nuevo Vehiculo.

Luego le pedimos al Taller (que es el Jframe) que la muestre al usuario. Recordemos que antes de hacer esto, primero hay que comprobar si hay un usuario logueado.

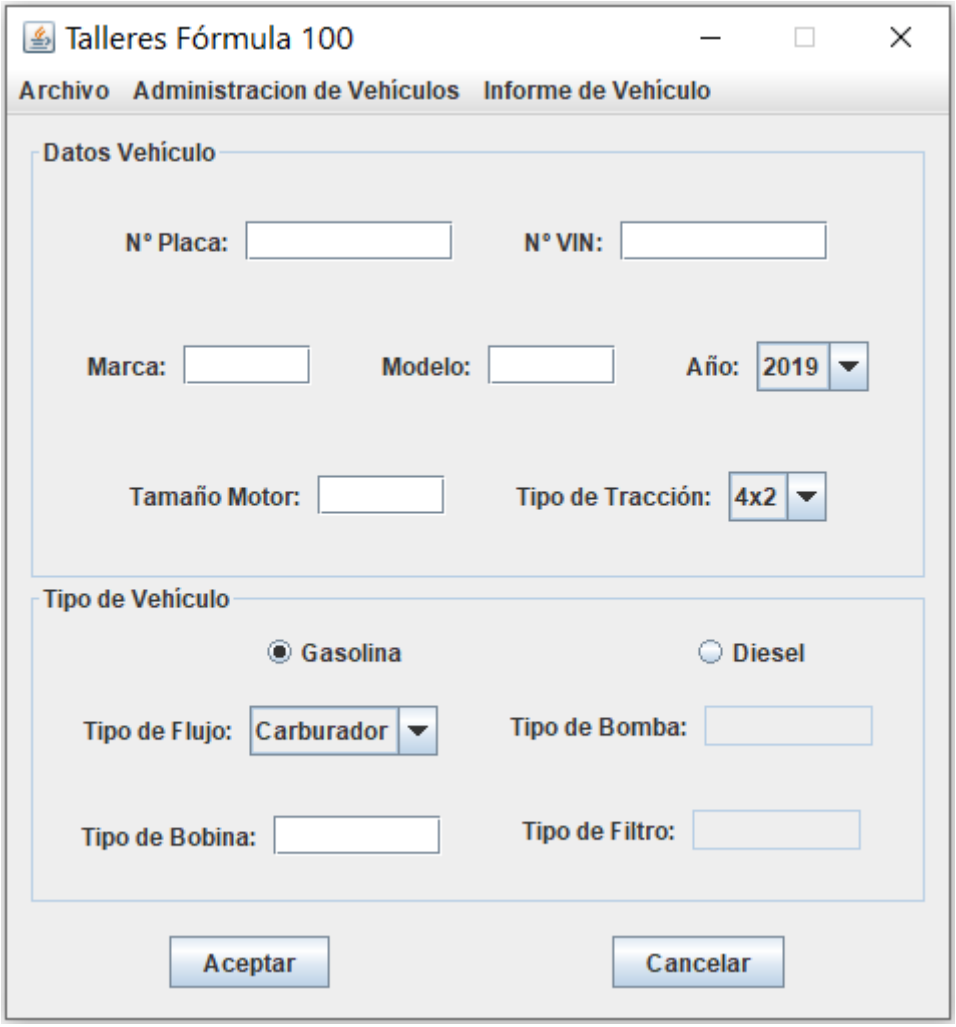
```
	class AccionAgregarVehiculo implements ActionListener {
		@Override
		public void actionPerformed(ActionEvent e) {
			if (taller.usuarioLogueado) {
				panelDatVehi.resetCampos();
				taller.setContentPane(panelDatVehi);
				taller.pack();
				taller.setLocationRelativeTo(null);
			}
			else
				mensajeEmergente("Ha de iniciar sesión para acceder a esta
función.",
					"Opción no disponible");
		}

	}
```

Una vez tenemos esta clase ActionListener escrita dentro de la clase Controlador, tenemos que irnos a la clase Taller para pedirle al JmenuItem correspondiente que implemente esta acción, tal y como hicimos anteriormente con la acción de Cerrar Sesión.

```
public class Taller extends JFrame{

	public boolean usuarioLogueado;
	private Controlador controlador;

	public Taller() {
		usuarioLogueado = false;
		controlador = new Controlador(this);
		setTitle("Talleres Fórmula 100");
		setJMenuBar(crearBarraMenu());
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		pack();
		setResizable(false);
		setLocationRelativeTo(null);
		setVisible(true);
	}

	private JMenuBar crearBarraMenu() {
		JMenuBar barra = new JMenuBar();

		JMenu archivo = new JMenu("Archivo");
		JMenuItem cerrar = new JMenuItem("Cerrar Sesión");
		cerrar.addActionListener(controlador.new AccionCerrarSesion());
		JMenuItem salir = new JMenuItem("Salir");
		salir.addActionListener(new AccionSalir());
		archivo.add(cerrar);
		archivo.add(salir);

		JMenu administracion = new JMenu("Administracion de Vehículos");
		JMenuItem agregar = new JMenuItem("Agregar Vehículo");
		agregar.addActionListener(controlador.new AccionAgregarVehiculo());
```

```
            JMenuItem editar = new JMenuItem("Editar Vehículo");
            JMenuItem estado = new JMenuItem("Admin. Estado del Vehículo");
            JMenuItem eliminar = new JMenuItem("Eliminar Vehículo");
```

Y con esto ya tenemos todo listo para iniciar sesión, y elegir la opción Agregar Vehículo para acceder a nuestro formulario en pantalla.



Perfecto, con esto ya tenemos la parte de la Vista para esta función. Pero nos falta la parte del Modelo, tenemos que crear un clase que gestione Vehículos similar a la que escribimos para gestionar usuarios, con su ArrayList, su método para añadir vehículos, buscarlos, guardarlos en disco…

```java
package tallerVehiculos;

import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.io.Serializable;
import java.util.ArrayList;

import javax.swing.JOptionPane;

@SuppressWarnings("serial")
public class GestionVehiculos implements Serializable{

	private ArrayList<Vehiculo> vehiculos;

	public GestionVehiculos() {
		vehiculos = new ArrayList<Vehiculo>();
	}

	public boolean nuevoVehiculo(Vehiculo v) {
		if (vehiculos.contains(v)) {
			JOptionPane.showMessageDialog(null, "Ya existe un Vehículo con
VIN: " + v.getVIN(),
					"Nuevo Vehículo", JOptionPane.WARNING_MESSAGE);
			return false;
		}
		else {
			vehiculos.add(v);
			return true;
		}
	}

	/**
	 * Busca un Vehiculo a partir del atributo VIN
	 * @param VIN String con el VIN del Vehículo a buscar
	 * @return Vehículo coincidente con el VIN.<br>
	 * Valor <b>null</b> si no existe.
	 */
	public Vehiculo buscarVehiculo(String VIN) {
		for (Vehiculo v: vehiculos) {
			if (v.getVIN().equals(VIN))
				return v;
		}

		return null;
	}

	public boolean guardarVehiculos() {
		try {
			ObjectOutputStream obs = new ObjectOutputStream(new
FileOutputStream("vehiculos.bin"));
			obs.writeObject(this);
			obs.close();
			return true;
		} catch (IOException e) {
			JOptionPane.showMessageDialog(null, "Error al acceder al archivo:
vehiculos.bin",
					"Guardar Vehículos", JOptionPane.ERROR_MESSAGE);
			e.printStackTrace();
			return false;
		}
```

```
	}

}
```

Recordemos que ha de implementar la interfaz Serializable para poder ser guardado en disco y, por lo tanto, la clase Vehículo también ha de implementarla.

```
public abstract class Vehiculo implements Serializable{

	private String placa;
	private String VIN;
	private String marca;
	...
```

Una vez escrita nuestra clase gestora de Vehículos, tenemos que "presentársela" a la clase Controlador para que se ocupe de ella así que añadimos un nuevo atributo.

```
public class Controlador {

	//Datos Modelo
	private GestionUsuario gestorUsuarios;
	private GestionVehiculos gestorVehiculos;
	//Datos Vista
	private Taller taller; //JFrame principal
	...
```

Una vez "presentados", ahora tenemos que indicarle al Controlador que ha de hacer con ella. El proceso será el mismo que le indicamos para la gestión de usuarios.
Al arrancar el programa ha de buscar en disco a ver si hay vehículos guardados. Si los hay, pues los recuperamos. Si no los hay, o no puede acceder al archivo por algún motivo, pues ha de crear una nueva instancia con un registro de Vehículos vacío.
Así que añadimos este método a la clase Controlador:

```
	private void iniciarGestorVehiculos() {
		File vehiculos = new File("vehiculos.bin");
		if (vehiculos.exists()) {
			ObjectInputStream ois;
			try {
				ois = new ObjectInputStream(new FileInputStream(vehiculos));
				Object datos = ois.readObject();
				if (datos instanceof GestionVehiculos) {
					gestorVehiculos = (GestionVehiculos)datos;
				}
				else {
					mensajeEmergente("Datos de Vehículos guardados no son
válidos", "Gestión Vehículos");
					gestorVehiculos = new GestionVehiculos();
				}
				ois.close();
```

```
                    } catch (IOException e) {
                          mensajeEmergente("No se pudo acceder a:\n" +
vehiculos.getAbsolutePath(), "Gestion Vehículos");
                          gestorVehiculos = new GestionVehiculos();
                    } catch (ClassNotFoundException e) {
                          mensajeEmergente("Datos de vehículos guardados no son
válidos", "Gestión Vehículos");
                          gestorVehiculos = new GestionVehiculos();
                    }
             }
             else {
                    mensajeEmergente("No hay datos de Vehículos guardados", "Gestión
Vehículos");
                    gestorVehiculos = new GestionVehiculos();
             }
      }
```

Y una vez escrito, lo invocamos en el constructor de la clase Controlador:

```
      public Controlador (Taller t) {
             taller = t;
             iniciarGestorUsuarios();
             iniciarGestorVehiculos();
             panelLogin = new UsuarioLogin();
             panelDatVehi = new PanelDatosVehiculo();
             asignarAcciones();
             mostrarPanelLogin();
      }
```

Ahora sí, al arrancar programa se recuperarán datos de Usuarios y de Vehículos, si los hubiera.

Ahora necesitamos crear las clases ActionListener para los botones Aceptar y Cancelar del formulario de datos de Vehículos.
Empecemos por lo fácil, por la acción de Cancelar.
Esta acción simplemente tiene que hacer que el programa vuelva a mostrar el panel de bienvenida.
Pero aquí vamos a encontrarnos con un pequeño fallo que no habíamos tenido en cuenta.
En las acciones AccionIniciarSesion y AccionCrearUsuario invocábamos este panel de bienvenida y le pasábamos el nombre del usuario para tener el saludo personalizado. Este nombre lo teníamos a mano porque estábamos trabajando con Usuarios.

Pero ahora no estamos trabajando con Usuarios, sino con Vehículos, y no tenemos el nombre del Usuario que se ha logueado guardado en ningún sitio. No podemos invocar el panel de bienvenida (nuestra clase PanelBienvenida) sin proporcionarle un nombre y no tenemos ninguno.

Para corregir este fallo, tenemos que guardar este nombre en algún sitio, por ejemplo, a la clase Controlador podemos añadir un nuevo atributo String donde guardar el nombre del Usuario logueado.

```
public class Controlador {

	//Datos Modelo
	private String nombreLogueado;
	private GestionUsuario gestorUsuarios;
	private GestionVehiculos gestorVehiculos;
```

Este nombre tendrá que ir cambiando según los usuarios cambian de sesión, así que tenemos que actualizar las clases ActionListener que se encargan de estos asuntos para que interactúen con este nuevo atributo.

```
class AccionCrearUsuario implements ActionListener {
	@Override
	public void actionPerformed(ActionEvent e) {
		if (panelLogin.campoNombre.getText().isEmpty() ||
panelLogin.campoPassword.getPassword().length == 0)
			mensajeEmergente("Ha de rellenar los dos campos", "Crear Usuario");
		else {
			String nombre = panelLogin.campoNombre.getText();
			String password = String.copyValueOf(panelLogin.campoPassword.getPassword());
			String confirm = JOptionPane.showInputDialog(null, "Por favor, confirme el
Password:");
			if (password.equals(confirm)) {
				if (gestorUsuarios.nuevoUsuario(new Usuario(nombre, password))) {
					mensajeEmergente("Nuevo usuario creado.\nSesión iniciada",
							"Crear Usuario");
					gestorUsuarios.guardarUsuarios();
					taller.usuarioLogueado = true;
					nombreLogueado = nombre; //Guardamos nombre del usuario logueado
					taller.setContentPane(new PanelBienvenida(nombreLogueado));
					taller.pack();
					taller.setLocationRelativeTo(null);
				}
			}
			else
				mensajeEmergente("El password de registro no coincide", "Crear Usuario");
		}

	}
}

class AccionIniciarSesion implements ActionListener {
	@Override
	public void actionPerformed(ActionEvent e) {
		if (panelLogin.campoNombre.getText().isEmpty() ||
panelLogin.campoPassword.getPassword().length == 0)
			mensajeEmergente("Ha de rellenar los dos campos", "Crear Usuario");
		else {
			String nombre = panelLogin.campoNombre.getText();
			String password = String.copyValueOf(panelLogin.campoPassword.getPassword());
			if (gestorUsuarios.validarLogin(new Usuario(nombre, password))) {
				taller.usuarioLogueado = true;
				nombreLogueado = nombre;
				taller.setContentPane(new PanelBienvenida(nombreLogueado));
				taller.pack();
				taller.setLocationRelativeTo(null);
			}
		}
	}
}
```

Corregido este detalle, ahora podemos volver a las acciones del formulario de Vehículos.
Dijimos de empezar por la fácil, cancelar la creación/edición de un Vehículo.
Añadimos a la clase Controlador esta clase ActionListener que como vemos, consiste en mostrar el panel de bienvenida, y ahora sí tenemos un nombre guardado que proporcionarle.

```
	class AccionCancelarVehiculo implements ActionListener {
		@Override
		public void actionPerformed(ActionEvent e) {
			taller.setContentPane(new PanelBienvenida(nombreLogueado));
			taller.pack();
			taller.setLocationRelativeTo(null);
		}
	}
```

Para que funcione, se lo asignamos al botón Cancelar de nuestro PanelDatosVehiculo, que es un atributo de Controlador.
Esto lo hacemos añadiendo una nueva línea al método asignarAcciones(), quien acordamos que se iba a encargar de estos asuntos.

```
private void asignarAcciones() {
	panelLogin.botonCrear.addActionListener(new AccionCrearUsuario());
	panelLogin.botonIniciar.addActionListener(new AccionIniciarSesion());
	panelDatVehi.botonCancelar.addActionListener(new AccionCancelarVehiculo());
}
```

Y ahora si podemos comprobar que tras acceder al formulario de Agregar Vehículo, si pulsamos Cancelar podremos volver a la pantalla de saludo inicial.

Pasemos a lo “difícil”, crear un nuevo Vehículo con los datos introducidos cuando se pulse el botón Aceptar.
Para crear un Vehículo debemos comprobar que se han introducido todos los datos necesarios, según el tipo de Vehículo escogido.
Una vez se ha “validado” el formulario, debemos recopilar esos datos y construir un Diésel o un Gasolina según se haya seleccionado.
Recordemos que el botón Aceptar funcionará de forma distinta según si estamos Creando un Vehículo o lo estamos Editando, asunto que controlábamos mediante un booleano, así que debemos tener esto en cuenta.

Para empezar, vamos a dotar a la clase PanelDatosVehiculo un método que nos diga si el formulario es validable o no. Este método comprobará que los campos de texto necesarios NO están vacíos.

```
	public boolean formularioEsValido() {
		if (!campoPlaca.getText().isEmpty() &&
				!campoVIN.getText().isEmpty() &&
				!campoMarca.getText().isEmpty() &&
				!campoModelo.getText().isEmpty() &&
				!campoTamanioMotor.getText().isEmpty()) {
			//Comprobamos campos específicos
			if (radioGasolina.isSelected())
				return !campoBobina.getText().isEmpty();
			else //Es diesel
				return (!campoTipoBomba.getText().isEmpty() &&
						!campoTipoFiltro.getText().isEmpty());
		}
		else
			return false;
	}
```

Y ahora le añadimos otro método que nos retorne un Vehiculo, ya sea Diesel o Gasolina, con los datos introducidos:

```
	public Vehiculo getNuevoVehiculo() {
		String placa = campoPlaca.getText();
		String VIN = campoVIN.getText();
		String marca = campoMarca.getText();
		String modelo = campoModelo.getText();
		String anio = (String)comboAnio.getSelectedItem();
		String tamanioMotor = campoTamanioMotor.getText();
		String traccion = (String)comboTraccion.getSelectedItem();
		//Tenemos datos comunes. Ahora vemos si es Gasolina o Diesel
		if (radioGasolina.isSelected()) {
			String flujo = (String)comboTipoFlujo.getSelectedItem();
			String bobina = campoBobina.getText();
			return new Gasolina(placa, VIN, marca, modelo, anio, tamanioMotor,
					traccion, flujo, bobina);
		}
		else {//Diesel
			String bomba = campoTipoBomba.getText();
			String filtro = campoTipoFiltro.getText();
			return new Diesel(placa, VIN, marca, modelo, anio, tamanioMotor,
					traccion, bomba, filtro);
		}
	}
```

Bien, ya podemos validar el formulario y podemos construir un Vehículo con sus datos.

Ahora necesitamos la acción para que el botón Aceptar use estos métodos y haga llegar el Vehículo construido al Gestor de Vehículos para que lo guarde.

De nuevo recuerdo que esta acción ha de tener en cuenta si estamos Editando o no.
Vámonos a la clase Controlador y creamos el siguiente ActionListener:

```
class AccionAceptarVehiculo implements ActionListener {
	@Override
	public void actionPerformed(ActionEvent e) {
		if (panelDatVehi.editando) {
			//TODO Aquí va el código para EDITAR un Vehículo
		}
		else { //Estamos creando nuevo Vehículo
			if (panelDatVehi.formularioEsValido()) {
				if
(gestorVehiculos.nuevoVehiculo(panelDatVehi.getNuevoVehiculo())) {
					mensajeEmergente("Nuevo Vehículo registrado", "Nuevo
Vehículo");
					gestorVehiculos.guardarVehiculos();
				}
			}
			else
				mensajeEmergente("Faltan campos por rellenar.", "Nuevo
Vehículo");
		}
	}
}
```

Nótese que ya hemos dejado reservado el espacio donde posteriormente irá el código para Editar un Vehículo.
Ya solo queda asignar esta acción al botón Aceptar del panel correspondiente.

```
private void asignarAcciones() {
	panelLogin.botonCrear.addActionListener(new AccionCrearUsuario());
	panelLogin.botonIniciar.addActionListener(new AccionIniciarSesion());
	panelDatVehi.botonCancelar.addActionListener(new AccionCancelarVehiculo());
	panelDatVehi.botonAceptar.addActionListener(new AccionAceptarVehiculo());
}
```

Y listo, ya podemos crear Nuevos Vehículos y quedarán guardados.

Ahora toca ponerse con la opción de Editar Vehículo. Para poder Editarlo, primero el usuario ha de indicarnos cuál Vehículo quiere visualizar para ser editado.
El enunciado dice que nos puede proporcionar placa o VIN, así que podemos pedir este dato mediante un JoptionPane. Como no sabemos si nos ha dado un VIN o una placa, buscaremos en el ArrayList por ambos atributos, a ver cuál nos devuelve una coincidencia.
Una vez encontrado este Vehículo, ahora sí mostraremos nuestro formulario, pero esta vez mostrando los datos del Vehículo registrado para que el usuario pueda editarlos.

Lo primero que vamos a hacer es un método para la clase GestionVehiculos para que recibiendo un String, nos busque un Vehículo cuya placa o VIN coincida

A esta clase ya le hicimos un método llamado buscarVehiculo() para buscar por VIN, pero como también hay que buscar por placa, lo vamos a modificar:

```
	public Vehiculo buscarVehiculo(String dato) {
		for (Vehiculo v: vehiculos) {
			if (v.getVIN().equals(dato) || v.getPlaca().equals(dato))
				return v;
		}

		return null;
	}
```

El siguiente paso es crear un ActionListener para el JmenuItem llamado “Editar Vehículo”.
Esta acción lo primero que hará será pedir el dato al usuario y luego buscar un Vehículo a través de ese dato.
Si ha encontrado algo, le pasamos ese Vehículo a la clase PanelDatosVehiculo y le pedimos a Taller que muestre este panel.

El Vehículo lo pasamos mediante un método que aún no hemos escrito, será lo siguiente que haremos.
Ahora, en Controlador, creamos esta clase ActionListener:

```
class AccionEditarVehiculo implements ActionListener {
	@Override
	public void actionPerformed(ActionEvent e) {
		String dato = JOptionPane.showInputDialog(null,
				"Introduzca placa o VIN del Vehículo que quiere EDITAR",
				"Editar Vehículo", JOptionPane.QUESTION_MESSAGE);
		if (dato != null) {
			Vehiculo v = gestorVehiculos.buscarVehiculo(dato);
			if (v == null)
				mensajeEmergente("No hay Vehículos con esa placa o VIN",
						"Editar Vehículo");
			else {
				panelDatVehi.editarVehiculo(v);
				taller.setContentPane(panelDatVehi);
				taller.pack();
				taller.setLocationRelativeTo(null);
			}
		}
	}
}
```

Y en la clase Taller, asignamos esta acción al JmenuItem correspondiente:

```
	private JMenuBar crearBarraMenu() {
		JMenuBar barra = new JMenuBar();

		JMenu archivo = new JMenu("Archivo");
		JMenuItem cerrar = new JMenuItem("Cerrar Sesión");
		cerrar.addActionListener(controlador.new AccionCerrarSesion());
		JMenuItem salir = new JMenuItem("Salir");
		salir.addActionListener(new AccionSalir());
		archivo.add(cerrar);
		archivo.add(salir);

		JMenu administracion = new JMenu("Administracion de Vehículos");
		JMenuItem agregar = new JMenuItem("Agregar Vehículo");
		agregar.addActionListener(controlador.new AccionAgregarVehiculo());
		JMenuItem editar = new JMenuItem("Editar Vehículo");
		editar.addActionListener(controlador.new AccionEditarVehiculo());
		JMenuItem estado = new JMenuItem("Admin. Estado del Vehículo");
		JMenuItem eliminar = new JMenuItem("Eliminar Vehículo");
		administracion.add(agregar);
		administracion.add(editar);
		administracion.add(estado);
		administracion.add(eliminar);

		JMenu informe = new JMenu("Informe de Vehículo");
		JMenuItem mantenimiento = new JMenuItem("Lista Vehículos mantenimiento");
		JMenuItem entregados = new JMenuItem("Lista Vehículos entregados");
		informe.add(mantenimiento);
		informe.add(entregados);

		barra.add(archivo);
		barra.add(administracion);
		barra.add(informe);

		return barra;
	}
```

Ya tenemos la Accion lista, pero no podemos ejecutarla todavía. Tenemos dos cosas pendientes aún:
- Primero, escribir el método por el que le pasamos un Vehículo editable al formulario PanelDatosVehiculo.
- Segundo, completar la clase AccionAceptarVehiculo para el botón Aceptar sepa que hacer cuando estamos Editando un Vehículo.

Comencemos por lo primero. Vamos a la clase PanelDatosVehiculo y escribiremos la clase para que reciba un Vehículo y muestre los datos en pantalla.
Además de este cambio, vamos a añadir un nuevo atributo para conservar una referencia al Vehículo que estamos editando. De este modo, luego cuando aceptemos los cambios nos será mucho más fácil actualizar este Vehículo con los nuevos datos introducidos

Añadimos atributo:

```
public class PanelDatosVehiculo extends JPanel{

	public boolean editando;
	private Vehiculo vehiculoEditable;
	//Atributos comunes de Vehiculos
	private JTextField campoPlaca;
	...
	...
```

Y ahora sí, escribimos el método editarVehiculo().
Este método activa el modo edicion, guardar referencia del Vehículo editable y muestra datos en pantalla.
Además ha de comprobar si el Vehículo recibido es Diesel o Gasolina, para decidir que datos mostrar, y que campos ha de bloquear.
Recordemos que el campo VIN no es editable. Así mismo, no se podrán cambiar los RadioButton.
La activación/desactivación de estos campos según si editamos o no, será más cómodo si lo hacemos con otro método al que pasaremos un booleano, con true activamos modo edición, con false lo desactivamos.
Así que vamos a crear estos dos métodos:

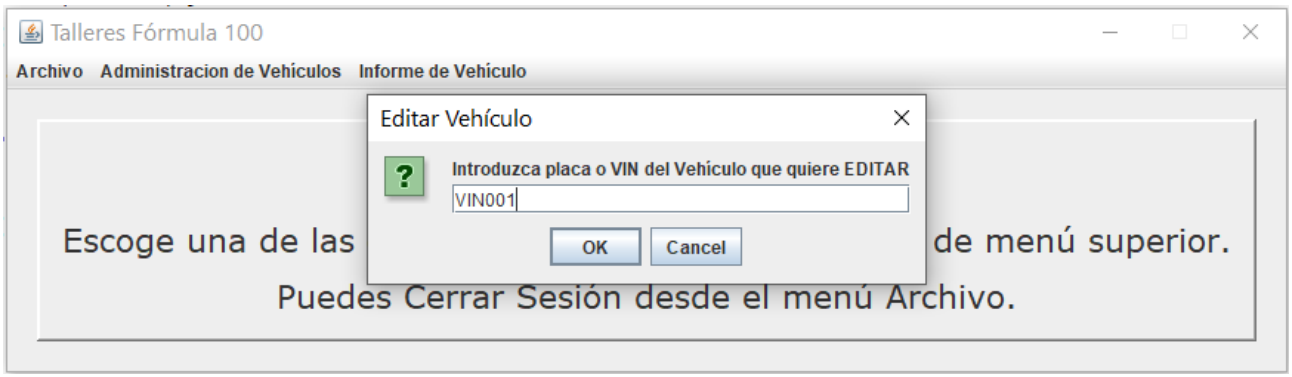
```
	public void modoEdicion(boolean modo) {
		if (modo) {
			editando = true;
			campoVIN.setEditable(false);//Dato no modificable
			//RadioButton tampoco son editables
			radioGasolina.setEnabled(false);
			radioDiesel.setEnabled(false);
		}
		else {
			editando = false;
			campoVIN.setEditable(true);
			radioGasolina.setEnabled(true);
			radioDiesel.setEnabled(true);
		}
	}

	public void editarVehiculo(Vehiculo v) {
		modoEdicion(true);
		vehiculoEditable = v; //Guardamos referencia
		//Mostramos datos comunes
		campoPlaca.setText(v.getPlaca());
		campoVIN.setText(v.getVIN());
		campoMarca.setText(v.getMarca());
		campoModelo.setText(v.getModelo());
		comboAnio.setSelectedItem(v.getAnio());
		campoTamanioMotor.setText(v.getTamanioMotor());
		comboTraccion.setSelectedItem(v.getTraccion());
		//Ahora depende de si es Diesel o Gasolina
		if (v instanceof Gasolina) {
			radioGasolina.setSelected(true);
			seleccionarGasolina();
			comboTipoFlujo.setSelectedItem((((Gasolina)
```

```
v).getFlujoGasolina()));
                    campoBobina.setText(((Gasolina)v).getTipoBobina());
            }
            else {
                    radioDiesel.setSelected(true);
                    seleccionarDiesel();
                    campoTipoBomba.setText(((Diesel)v).getTipoBomba());
                    campoTipoFiltro.setText(((Diesel)v).getTipoFiltro());
            }
      }
```

Así, cuando vamos a editar un Vehiculo, activamos el modo edición. Este modo edición, debemos asegurarnos de que se desactiva cuando vamos a Crear un Vehículo.
Como al Crear Vehículo una de las cosas que hacemos es invocar al método resetCampos(), dentro de este método podemos añadir una línea para desactivar el modo edición.

```
      public void resetCampos() {
            editando = false;
            campoPlaca.setText(null);
            campoVIN.setText(null);
            campoMarca.setText(null);
            campoModelo.setText(null);
            comboAnio.setSelectedIndex(0);
            campoTamanioMotor.setText(null);
            comboTraccion.setSelectedIndex(0);
            comboTipoFlujo.setSelectedIndex(0);
            campoBobina.setText(null);
            campoTipoBomba.setText(null);
            campoTipoFiltro.setText(null);
            modoEdicion(false);
      }
```

Pues nada, vamos a probar como funciona esto. Si ya tenemos vehículos creados, podemos probar la opción Editar Vehículo:



Voilá, aquí tenemos los datos recuperados de un Vehículo registrado anteriormente.

Pero ojo, todavía no podemos editar los datos.
El botón Aceptar todavía no sabe como actuar cuando estamos editando, si ahora lo pulsamos, no hará absolutamente nada porque no tiene instrucciones para este modo.

Pues vamonos a la clase Controlador, a completar la clase AccionAceptarVehiculo.
Rellenamos la parte que dejamos pendiente de completar. De nuevo tenemos que comprobar si el formulario es válido.
En caso afirmativo, tenemos que decirle a la clase PanelDatosVehiculo que acepte esos datos, actualice el Vehículo que estamos editando, y guarde los datos en disco.
Para hacer que actualice el Vehiculo editable, vamos a invocar un método que todavía no hemos escrito:

```
class AccionAceptarVehiculo implements ActionListener {
	@Override
	public void actionPerformed(ActionEvent e) {
		if (panelDatVehi.editando) {
			if (panelDatVehi.formularioEsValido()) {
				panelDatVehi.aceptarEdicion();
				mensajeEmergente("Los datos se han modificado", "Editar
Vehículo");
				gestorVehiculos.guardarVehiculos();
			}
			else
				mensajeEmergente("Faltan campos por rellenar.", "Nuevo
Vehículo");
		}
		else { //Estamos creando nuevo Vehículo
			if (panelDatVehi.formularioEsValido()) {
				if
(gestorVehiculos.nuevoVehiculo(panelDatVehi.getNuevoVehiculo())) {
					mensajeEmergente("Nuevo Vehículo registrado", "Nuevo
Vehículo");
					gestorVehiculos.guardarVehiculos();
				}
			}
			else
				mensajeEmergente("Faltan campos por rellenar.", "Nuevo
Vehículo");
		}
	}
}
```

Y ahora volvemos a la clase PanelDatosVehiculo y creamos el método que hemos bautizado como aceptarEdicion().

```
	public void aceptarEdicion() {
		//Datos comunes
		vehiculoEditable.setPlaca(campoPlaca.getText());
		vehiculoEditable.setMarca(campoMarca.getText());
		vehiculoEditable.setModelo(campoModelo.getText());
		vehiculoEditable.setAnio((String)comboAnio.getSelectedItem());
		vehiculoEditable.setTamanioMotor(campoTamanioMotor.getText());
		vehiculoEditable.setTraccion((String)comboTraccion.getSelectedItem());
		//Ahora según si Diesel o Gasolina
		if (vehiculoEditable instanceof Gasolina) {


((Gasolina)vehiculoEditable).setFlujoGasolina((String)comboTipoFlujo.getSelectedItem()
);
			((Gasolina)vehiculoEditable).setTipoBobina(campoBobina.getText());
		}
		else {
			((Diesel)vehiculoEditable).setTipoBomba(campoTipoBomba.getText());

	((Diesel)vehiculoEditable).setTipoFiltro(campoTipoFiltro.getText());
		}
	}
```

Y listo, con esto ya se guardan los cambios al Editar Vehículos.

Podemos Crear y Editar Vehículos. Nos faltan dos gestiones más por implementar:
- Administrar el estado de Vehículo: cuándo se inician las reparaciones, cuándo finalizan y si se ha entregado o no al cliente.
- Eliminar un Vehículo de los registros.

Vamos a empezar por lo fácil, Eliminar un Vehículo. Básicamente tendremos que pedir placa o VIN, buscar el Vehículo y eliminarlo.
Nos vamos a la clase Controlador y creamos una nueva clase ActionListener llamada AccionEliminarVehiculo.

Esta clase será muy similar al código escrito para la clase AccionEditarVehiculo.
La diferencia será que no mostraremos datos en formulario. Si encontramos el Vehículo indicado, pedimos confirmación al usuario para eliminarlo y si da el OK, eliminamos Vehículo.

```
class AccionEliminarVehiculo implements ActionListener {
	@Override
	public void actionPerformed(ActionEvent e) {
		String dato = JOptionPane.showInputDialog(null,
				"Introduzca placa o VIN del Vehículo que quiere ELIMINAR",
				"Eliminar Vehículo", JOptionPane.QUESTION_MESSAGE);
		if (dato != null) {
			Vehiculo v = gestorVehiculos.buscarVehiculo(dato);
			if (v == null)
				mensajeEmergente("No hay Vehículos con esa placa o VIN",
						"Eliminar Vehículo");
			else {
				int confirm = JOptionPane.showConfirmDialog(null,
						"¿Seguro que quiere ELIMINAR vehículo con VIN:
" + v.getVIN(),
						"Eliminar Vehículo", JOptionPane.YES_NO_OPTION,
						JOptionPane.QUESTION_MESSAGE);
				if (confirm == JOptionPane.YES_OPTION) {
					gestorVehiculos.eliminarVehiculo(v);
					gestorVehiculos.guardarVehiculos();
					mensajeEmergente("¡¡Vehículo Eliminado!!", "Eliminar
Vehículo");
				}
			}
		}
	}
}
```

Para pedir al gestor de vehiculos que elimine el seleccionado, estamos llamando a un método llamado eliminarVehiculo().

Este método aún no lo hemos escrito, así que vamos a la clase GestionVehículo y añadimos el siguiente método, súper sencillo:

```
	public void eliminarVehiculo(Vehiculo v) {
		vehiculos.remove(v);
	}
```

Este pequeño cambio en esta clase, nos va a suponer un pequeño problema.
Nuestro programa, para guardar los datos, lo que hace es una copia del objeto gestorVehiculos, que es una instancia de la clase GestionVehiculos. Tal cuál está en la RAM del sistema lo copia al disco duro.
Luego al iniciar programa, recupera esta copia y sigue trabajando con dicho objeto.

El problema es que ahora hemos cambiado la clase GestionVehiculos, hemos añadido un método que antes no existía y por lo tanto no existe dentro del objeto que se ha guardado en disco. Esto significa que el programa cuando quiera recuperar el objeto guardado en disco, Java se va a dar cuenta que pertenece a una clase distinta a la que existe actualmente, y no va a poder recuperarlo.

Es decir, si tenemos datos de Vehiculos guardados en disco,.. ya los hemos perdido. Son irrecuperables. Al arrancar el programa nos mostrará el mensaje de que ya no puede acceder al archivo donde habíamos guardado los datos.

Esto no es problema real durante la creación del programa, pues se supone que estaremos trabajando con datos de prueba.

Pero si puede llegar a ser un auténtico problema si estuvieramos modificando un programa ya finalizado y que el usuario/cliente ya había estado trabajando con datos reales.
Esto es algo a tener en cuenta, aunque de todos modos, en el “mundo real” seguramente nuestro programa estaría usando otros sistemas para guardar datos como bases de datos y/o archivos XML y no tendríamos este problema.

Tras este inciso, para probar nuestra accion eliminar Vehículo, solo nos falta irnos a la clase Taller y agregar esta acción al JmenuItem correspondiente:

```
	private JMenuBar crearBarraMenu() {
		JMenuBar barra = new JMenuBar();

		JMenu archivo = new JMenu("Archivo");
		JMenuItem cerrar = new JMenuItem("Cerrar Sesión");
		cerrar.addActionListener(controlador.new AccionCerrarSesion());
		JMenuItem salir = new JMenuItem("Salir");
		salir.addActionListener(new AccionSalir());
		archivo.add(cerrar);
		archivo.add(salir);

		JMenu administracion = new JMenu("Administracion de Vehículos");
		JMenuItem agregar = new JMenuItem("Agregar Vehículo");
		agregar.addActionListener(controlador.new AccionAgregarVehiculo());
		JMenuItem editar = new JMenuItem("Editar Vehículo");
		editar.addActionListener(controlador.new AccionEditarVehiculo());
		JMenuItem estado = new JMenuItem("Admin. Estado del Vehículo");
		JMenuItem eliminar = new JMenuItem("Eliminar Vehículo");
		eliminar.addActionListener(controlador.new AccionEliminarVehiculo());
		administracion.add(agregar);
		administracion.add(editar);
		administracion.add(estado);
		administracion.add(eliminar);

		JMenu informe = new JMenu("Informe de Vehículo");
		JMenuItem mantenimiento = new JMenuItem("Lista Vehículos mantenimiento");
		JMenuItem entregados = new JMenuItem("Lista Vehículos entregados");
		informe.add(mantenimiento);
		informe.add(entregados);

		barra.add(archivo);
		barra.add(administracion);
		barra.add(informe);

		return barra;
	}
```

Y ya podemos crear algunos Vehículos y luego eliminarlos.

Ahora hay que ponerse con lo de Administrar el Estado del Vehículo.

Necesitaríamos pedirle al usuario una placa o VIN y buscar el Vehículo, igual que hemos hecho para estas dos últimas acciones.

Una vez encontrado, el usuario debe tener disponibles tres opciones: Iniciar Reparación, Finalizar Reparación y marcar el Vehículo como “entregado” al cliente.

Para iniciar y finalizar la reparación el usuario ha de indicar la fecha. Pero podríamos facilitarle la tarea presentándole dos botones.
Cuando pulse el botón Iniciar Reparación, se mostrará y guardará (las fechas son atributos de Vehículo) la fecha actual.
Y lo mismo para el botón Finalizar Reparación. Así cuando el usuario inicie y termine la reparación, solo tendrá que pulsar los botones para grabar la fecha

Para indicar si el vehículo está entregado, podemos usar un simple CheckBox.

Así que manos a la obra, crearemos un nuevo panel para mostrar estos elementos.

Podemos usar un BorderLayout y dividir así en tres partes:
- Norte, un titulo y explicación descriptiva de que hacer en esta pantalla.
- Centro, dos botones con dos JtextField y un Jcheckbox
- Sur, simplemente un botón para salir de este panel.

EL centro se dividirá horizontalmente en tres subpaneles. Los dos primeros incorporan un JtextField y un botón debajo. El tercero tan solo el checkbox.

Uno de los atributos de esta clase será una referencia al Vehículo sobre el que estamos trabajando.
Este vehículo tiene entre sus atributos la fecha Inicio reparacion, fecha final y el boolean entregado.
Dependiendo del valor de estos atributos, decidiremos que botones estarán activados.
Si por ejemplo consta que el Vehículo tiene una reparación ya iniciada, no tiene sentido que pueda pulsar el botón Iniciar Reparación, así que en este caso lo desactivaríamos.
Otro ejemplo, si consta que el Vehículo ya ha sido entregado al dueño, no debería poder activar ningún botón ya que no podemos interactuar con un Vehículo que ya no está en nuestro taller.

Esta clase, que podemos llamar PanelGestionVehiculo, de momento vamos a escribir lo necesario para mostrar en pantalla los elementos y un método

para que reciba el Vehículo a gestionar. Este método será quien analice los atributos del Vehículo y decida que elementos estarán activos.

```
package paneles;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Font;

import javax.swing.BorderFactory;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.SwingConstants;

import tallerVehiculos.Vehiculo;

public class PanelEstadoVehiculo extends JPanel{

	private Vehiculo vehiculo;
	public JButton botonIniciar;
	public JButton botonFinalizar;
	public JButton botonSalir;
	public JCheckBox checkEntregado;
	private JTextField campoInicio;
	private JTextField campoTermino;

	public PanelEstadoVehiculo() {
		iniciarComponentes();
		setLayout(new BorderLayout());
		add(new PanelNorte(), BorderLayout.NORTH);
		add(new PanelCentro(), BorderLayout.CENTER);
		add(new PanelSur(), BorderLayout.SOUTH);
	}

	private void iniciarComponentes() {
		vehiculo = null;
		botonIniciar = new JButton("<html><p style=\"text-
align:center\">Iniciar<br>Reparación</p></html>");
		botonIniciar.setBackground(new Color(72, 193, 51));
		botonIniciar.setForeground(Color.WHITE);
		botonFinalizar = new JButton("<html><p style=\"text-
align:center\">Finalizar<br>Reparación</p></html>");
		botonFinalizar.setBackground(Color.RED);
		botonFinalizar.setForeground(Color.WHITE);
		botonSalir = new JButton("SALIR");
		botonSalir.setFont(new Font("Verdana", Font.BOLD, 22));
		checkEntregado = new JCheckBox("Entregado");
		checkEntregado.setFont(new Font("Verdana", Font.BOLD, 18));
		//Los campos de texto no son editables para el usuario
		campoInicio = new JTextField(5);
		campoInicio.setEditable(false);
		campoTermino = new JTextField(5);
		campoTermino.setEditable(false);
	}

	public void setVehiculo(Vehiculo v) {
```

```
            vehiculo = v;

            campoInicio.setText(vehiculo.getFechaInicio());
            campoTermino.setText(vehiculo.getFechaTermino());

            checkEntregado.setSelected(vehiculo.isEntregado());

            //Decidimos que botones estarán activos
            if (vehiculo.isEntregado()) {
                  //Vehiculo entregado, no se puede alterar su estado de reparación
                  botonIniciar.setEnabled(false);
                  botonFinalizar.setEnabled(false);
                  //Pero si se puede revertir la entrega al cliente
                  checkEntregado.setEnabled(true);
            }
            else {
                  if (vehiculo.getFechaInicio().equals("No Iniciado")) {
                        //Aun no se ha iniciado la reparación de este Vehiculo
                        botonIniciar.setEnabled(true);
                        botonFinalizar.setEnabled(false);
                        checkEntregado.setEnabled(true);
                  }
                  else if (vehiculo.getFechaTermino().equals("No Terminado")){
                        //Reparacion iniciada, pero no terminada
                        botonIniciar.setEnabled(false);
                        botonFinalizar.setEnabled(true);
                        //No se puede entregar hasta que se termine la reparacion
                        checkEntregado.setEnabled(false);
                  }
                  else {
                        //Reparacion iniciada y terminada.
                        //Aunque cabe la posibilidad de iniciar una nueva reparación
                        botonIniciar.setEnabled(true);
                        botonFinalizar.setEnabled(false);
                        checkEntregado.setEnabled(true);
                  }
            }
      }

      class PanelNorte extends JPanel {
            public PanelNorte() {
                  setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
                  JPanel panelTitulo = new JPanel();
                  JLabel titulo = new JLabel("<html><p><u>ESTADO del
Vehiculo</u></p></html>");
                  titulo.setFont(new Font("Verdana", Font.BOLD, 32));
                  panelTitulo.add(titulo);

                  JPanel panelLinea1 = new JPanel();
                  JLabel linea1 = new JLabel("Utilice los botones para Iniciar y
Finalizar el estado de Reparación");
                  linea1.setFont(new Font("Verdana", Font.PLAIN, 22));
                  panelLinea1.add(linea1);

                  JPanel panelLinea2 = new JPanel();
                  JLabel linea2 = new JLabel("Marque y desmarque la casilla para
indicar si el Vehículo ha sido entregado.");
                  linea2.setFont(new Font("Verdana", Font.PLAIN, 22));
                  panelLinea2.add(linea2);

                  add(panelTitulo);
```

```
                    add(panelLinea1);
                    add(panelLinea2);
                    setBorder(BorderFactory.createCompoundBorder(
                                  BorderFactory.createEmptyBorder(20, 20, 10, 20),

      BorderFactory.createCompoundBorder(BorderFactory.createRaisedBevelBorder(),
                                                BorderFactory.createEmptyBorder(10, 10,
10, 10))));
             }
      }

      class PanelCentro extends JPanel {
             public PanelCentro() {
                    setLayout(new BoxLayout(this, BoxLayout.X_AXIS));
                    JPanel inicio = new JPanel();
                    inicio.add(new PanelCampoYboton(campoInicio, botonIniciar));
                    JPanel terminar = new JPanel();
                    terminar.add(new PanelCampoYboton(campoTermino, botonFinalizar));
                    JPanel entregado = new JPanel();
                    entregado.add(checkEntregado);
                    entregado.setBorder(BorderFactory.createEmptyBorder(10, 10, 10,
10));

                    add(inicio); add(terminar); add(entregado);
                    setBorder(BorderFactory.createEmptyBorder(10, 20, 10, 20));
             }
      }

      class PanelCampoYboton extends JPanel {
             public PanelCampoYboton(JTextField campo, JButton boton) {
                    setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
                    campo.setHorizontalAlignment(SwingConstants.CENTER);
                    campo.setFont(new Font("Verdana", Font.BOLD, 18));
                    boton.setFont(new Font("Verdana", Font.BOLD, 22));
                    add(campo);
                    add(boton);
                    setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
             }
      }

      class PanelSur extends JPanel {
             public PanelSur() {
                    add(botonSalir);
                    setBorder(BorderFactory.createEmptyBorder(10, 20, 20, 20));
             }
      }

}
```

Una vez escrita la clase, nos vamos a Controlador para incluirla como atributo y la inicializamos en su constructor:

```
public class Controlador {

      //Datos Modelo
      private String nombreLogueado;
      private GestionUsuario gestorUsuarios;
      private GestionVehiculos gestorVehiculos;
      //Datos Vista
```

```
	private Taller taller; //JFrame principal
	//Paneles para la Vista
	private UsuarioLogin panelLogin;
	private PanelDatosVehiculo panelDatVehi;
	private PanelEstadoVehiculo panelEstadoVehi;

	public Controlador (Taller t) {
		taller = t;
		iniciarGestorUsuarios();
		iniciarGestorVehiculos();
		panelLogin = new UsuarioLogin();
		panelDatVehi = new PanelDatosVehiculo();
		panelEstadoVehi = new PanelEstadoVehiculo();
		asignarAcciones();
		mostrarPanelLogin();
	}
```

Y para poder visualizarla, a Controlador le añadimos una clase ActionListener que se encargará de pedir al usuario un dato para buscar Vehículo, pasárselo a la clase PanelGestionVehiculo y mostrar el panel en pantalla.

```
class AccionGestionarVehiculo implements ActionListener {
	@Override
	public void actionPerformed(ActionEvent e) {
		String dato = JOptionPane.showInputDialog(null,
				"Introduzca placa o VIN del Vehículo que quiere GESTIONAR",
				"Gestionar Vehículo", JOptionPane.QUESTION_MESSAGE);
		if (dato != null) {
			Vehiculo v = gestorVehiculos.buscarVehiculo(dato);
			if (v == null)
				mensajeEmergente("No hay Vehículos con esa placa o VIN",
							"Gestionar Vehículo");
			else {
				panelEstadoVehi.setVehiculo(v);
				taller.setContentPane(panelEstadoVehi);
				taller.pack();
				taller.setLocationRelativeTo(null);
			}
		}
	}
}
```

Para que esta acción sea efectiva, tenemos que irnos a Taller y asignarsela al JmenuItem correspondiente.

```
	private JMenuBar crearBarraMenu() {
		JMenuBar barra = new JMenuBar();

		JMenu archivo = new JMenu("Archivo");
		JMenuItem cerrar = new JMenuItem("Cerrar Sesión");
		cerrar.addActionListener(controlador.new AccionCerrarSesion());
		JMenuItem salir = new JMenuItem("Salir");
		salir.addActionListener(new AccionSalir());
		archivo.add(cerrar);
		archivo.add(salir);

		JMenu administracion = new JMenu("Administracion de Vehículos");
```

```
            JMenuItem agregar = new JMenuItem("Agregar Vehículo");
            agregar.addActionListener(controlador.new AccionAgregarVehiculo());
            JMenuItem editar = new JMenuItem("Editar Vehículo");
            editar.addActionListener(controlador.new AccionEditarVehiculo());
            JMenuItem estado = new JMenuItem("Admin. Estado del Vehículo");
            estado.addActionListener(controlador.new AccionGestionarVehiculo());
            JMenuItem eliminar = new JMenuItem("Eliminar Vehículo");
            eliminar.addActionListener(controlador.new AccionEliminarVehiculo());
            administracion.add(agregar);
            administracion.add(editar);
            administracion.add(estado);
            administracion.add(eliminar);

            JMenu informe = new JMenu("Informe de Vehículo");
            JMenuItem mantenimiento = new JMenuItem("Lista Vehículos mantenimiento");
            JMenuItem entregados = new JMenuItem("Lista Vehículos entregados");
            informe.add(mantenimiento);
            informe.add(entregados);

            barra.add(archivo);
            barra.add(administracion);
            barra.add(informe);

            return barra;
        }
```

Con esto, al pulsar esta opción en la barra de menú, e indicarle un Vehículo, ya podemos visualizar el panel en pantalla. Aunque los botones todavía no tienen acciones asignadas, no hacen nada todavía:

Siguiente paso será escribir clases ActionListener para los botones, y también para el JcheckBox.

Veamos como sería la acción Iniciar Reparación.

En la clase Controlador, escribimos la siguiente clase:

```
	class AccionIniciarReparacion implements ActionListener {
		@Override
		public void actionPerformed(ActionEvent e) {
			panelEstadoVehi.iniciar();
			gestorVehiculos.guardarVehiculos();
		}
	}
```

Muy sencilla, al panel que gestiona el estado del Vehículo le damos orden de iniciar invocando a un método del mismo nombre. Este método lo escribiremos a continuación.
Después de iniciar, pedimos al Gestor de Vehículos que guarde los datos, pues al iniciar reparación, los atributos de fechas de dicho Vehículo han cambiado y hay que guardarlo.

Veamos en que consiste el método iniciar(), vámonos ahora a la clase PanelEstadoVehiculo.
Cuando pulsamos iniciar, el atributo Fecha Inicio del Vehículo que estamos gestionando ha de coger la fecha actual (LocalDate.now()).
La fecha de Termino, hemos de asegurarnos que tiene valor null, ya que si iniciamos ahora reparación, no tiene fecha de termino todavía.
Tras hacer esto, hay que actualizar los campos de texto en pantalla con los datos de fechas actuales, para esto, le pedimos al objeto Vehículo que nos propociones estas fechas, pues ya le escribimos unos métodos para que transforme las fechas LocalDate en cadenas String formateadas.

Además, ha de desactivar el botón Iniciar, activar el botón Finalizar y desactivar el checkBox, pues no se puede entregar si está en proceso de reparación:

Este sería el método iniciar():

```
	public void iniciar() {
		vehiculo.setFechaInicio(LocalDate.now());
		vehiculo.setFechaTermino(null);
		campoInicio.setText(vehiculo.getFechaInicio());
		campoTermino.setText(vehiculo.getFechaTermino());
		botonIniciar.setEnabled(false);
		botonFinalizar.setEnabled(true);
		checkEntregado.setEnabled(false);
	}
```

Y con esto ya tendríamos nuestra acción lista para probarla, solo nos falta asignársela al botón Inicio de este panel.

Para ello, volvemos a la clase Controlador y actualizamos el método asignarAcciones() que hemos ido escribiendo poco a poco a lo largo de este ejercicio.

```
private void asignarAcciones() {
	panelLogin.botonCrear.addActionListener(new AccionCrearUsuario());
	panelLogin.botonIniciar.addActionListener(new AccionIniciarSesion());

	panelDatVehi.botonCancelar.addActionListener(new AccionCancelarVehiculo());
	panelDatVehi.botonAceptar.addActionListener(new AccionAceptarVehiculo());

	panelEstadoVehi.botonIniciar.addActionListener(new AccionIniciarReparacion());
}
```

Listo, si gestionamos un vehículo y pulsamos el botón iniciar, se coge la fecha actual y se alteran los elementos que pueden estar activos:

Si hemos entendido bien todo este proceso, escribir ahora el código para Finalizar la reparación, es pan comido.

El proceso es muy similar al que acabamos de realizar, podemos ir a la clase PanelEstadoVehiculo y escribimos el método para Finalizar la reparación.
Este método reactiva el botón Iniciar (pues cabe la posibilidad de que el taller tenga que volver a hacer nuevas reparaciones), desactiva el botón Finalizar, captura la fecha actual y la muestra en el campo correspondiente.

Además, activa el checkbox de Entregado, ya que ahora sí es posible entregar el coche pues se han finalizado las reparaciones actuales.

```
	public void finalizar() {
		vehiculo.setFechaTermino(LocalDate.now());
		campoInicio.setText(vehiculo.getFechaInicio());
		campoTermino.setText(vehiculo.getFechaTermino());
		botonIniciar.setEnabled(true);
		botonFinalizar.setEnabled(false);
		checkEntregado.setEnabled(true);
	}
```

Nos vamos a Controlador y escribimos la acción que invocará este método:

```
	class AccionFinalizarReparacion implements ActionListener {
		@Override
		public void actionPerformed(ActionEvent e) {
			panelEstadoVehi.finalizar();
			gestorVehiculos.guardarVehiculos();
		}
	}
```

Y como es habitual, asignamos esta acción al botón correspondiente mediante el método asignarAcciones() de la clase Controlador:

```
private void asignarAcciones() {
	panelLogin.botonCrear.addActionListener(new AccionCrearUsuario());
	panelLogin.botonIniciar.addActionListener(new AccionIniciarSesion());

	panelDatVehi.botonCancelar.addActionListener(new AccionCancelarVehiculo());
	panelDatVehi.botonAceptar.addActionListener(new AccionAceptarVehiculo());

	panelEstadoVehi.botonIniciar.addActionListener(new AccionIniciarReparacion());
	panelEstadoVehi.botonFinalizar.addActionListener(new
AccionFinalizarReparacion());
}
```

Ahora ya podemos Finalizar la reparación:

¿Siguiente paso? Pues escribir el código para Entregar el Vehículo al cliente.
Mismo procedimiento, vamos a PanelEstadoVehiculo a elaborar el método que se ocupe de esta tarea:

```
public void entregar() {
	/*
	 * Hay dos posibilidades, que el Vehículo NO está
	 * entregado y por lo tanto, se va a entregar ahora.
	 * La otra es que SI esté entregado, pero que por algún
	 * motivo el vehículo regresa de nuevo al taller y por
	 * lo tanto, dejará de estar Entregado.
	 *
	 * Consultamos el atributo "entregado"
	 * del objeto Vehículo que estamos gestionando.
	 */

	if (vehiculo.isEntregado()) { //YA entregado, pero regresa al taller
		botonIniciar.setEnabled(true); //Se puede iniciar una nueva reparación
		botonFinalizar.setEnabled(false);
		vehiculo.setEntregado(false); //Deja de estar entregado
	}
	else {//No entregado, se entrega ahora al cliente
		botonIniciar.setEnabled(false); //Ya no se pueden iniciar reparaciones
		botonFinalizar.setEnabled(false);
		vehiculo.setEntregado(true); //Ahora consta como entregado.
	}
}
```

En los comentarios del código se narra las posibilidades que existen y los cambios que se han de realizar en los elementos que hay en pantalla y también en el objeto Vehículo.

Volvemos a Controlador y escribimos la acción para ejecutar el método entregar():

```
	class AccionEntregarVehiculo implements ActionListener {
		@Override
		public void actionPerformed(ActionEvent e) {
			panelEstadoVehi.entregar();
			gestorVehiculos.guardarVehiculos();
		}
	}
```

Y lógicamente, asignamos esta acción para que funcione. Esta vez no se asigna a un botón, si no al JcheckBox para elegir si está entregado o no el Vehículo:

```
private void asignarAcciones() {
	panelLogin.botonCrear.addActionListener(new AccionCrearUsuario());
	panelLogin.botonIniciar.addActionListener(new AccionIniciarSesion());

	panelDatVehi.botonCancelar.addActionListener(new AccionCancelarVehiculo());
	panelDatVehi.botonAceptar.addActionListener(new AccionAceptarVehiculo());

	panelEstadoVehi.botonIniciar.addActionListener(new AccionIniciarReparacion());
	panelEstadoVehi.botonFinalizar.addActionListener(new
AccionFinalizarReparacion());
	panelEstadoVehi.checkEntregado.addActionListener(new AccionEntregarVehiculo());
	}
```

Prácticamente ya hemos terminado con este panel. Solo nos falta hacer que el botón SALIR funcione.
Este botón lo que tiene que hacer es simplemente volver a la pantalla de bienvenida, sin hacer ningún cambio.
Esta acción la escribimos directamente en la clase Controlador:

```
	class AccionSalirGestion implements ActionListener {
		@Override
		public void actionPerformed(ActionEvent e) {
			taller.setContentPane(new PanelBienvenida(nombreLogueado));
			taller.pack();
			taller.setLocationRelativeTo(null);
		}
	}
```

Y asignamos al botón correspondiente:

```
private void asignarAcciones() {
	panelLogin.botonCrear.addActionListener(new AccionCrearUsuario());
	panelLogin.botonIniciar.addActionListener(new AccionIniciarSesion());

	panelDatVehi.botonCancelar.addActionListener(new AccionCancelarVehiculo());
	panelDatVehi.botonAceptar.addActionListener(new AccionAceptarVehiculo());

	panelEstadoVehi.botonIniciar.addActionListener(new AccionIniciarReparacion());
	panelEstadoVehi.botonFinalizar.addActionListener(new
AccionFinalizarReparacion());
	panelEstadoVehi.checkEntregado.addActionListener(new AccionEntregarVehiculo());
	panelEstadoVehi.botonSalir.addActionListener(new AccionSalirGestion());
}
```

Y listo. Gestión de Vehículos finalizada.

Lo siguiente será mostrar los Informes de Vehículos.