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("%d2/%d2/%d4", fechaInicio.getDayOfMonth(),
					fechaInicio.getDayOfMonth(), fechaInicio.getYear());
	}

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

	public String getFechaTermino() {
		if (fechaTermino == null)
			return "No Terminado";
		else
			return String.format("%d2/%d2/%d4", fechaTermino.getDayOfMonth(),
					fechaTermino.getDayOfMonth(),
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”.

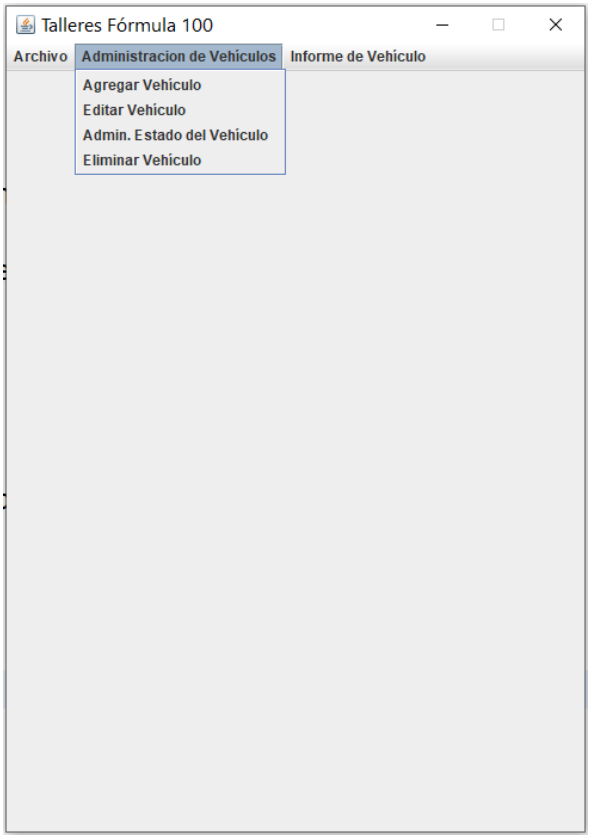
Ahora vamos a iniciar la parte que el usuario “sí ve”, es decir, la Vista. Creamos una clase Jframe que de momento no va a mostrar nada importante todavía.

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

	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”.

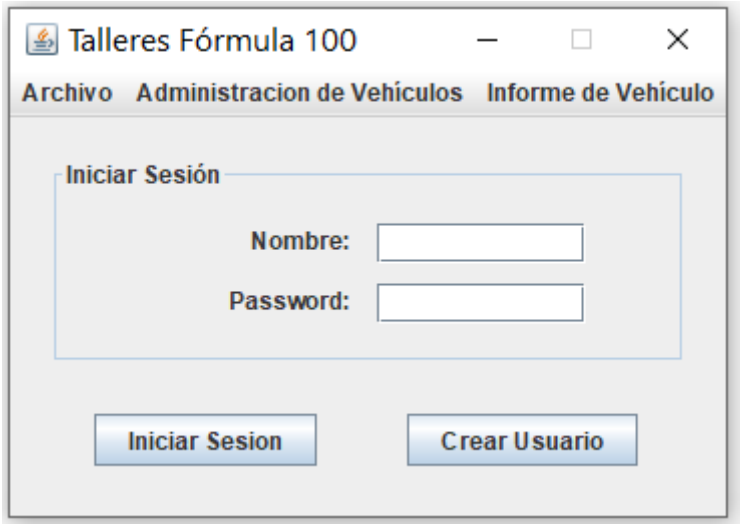
```java
package paneles;

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:

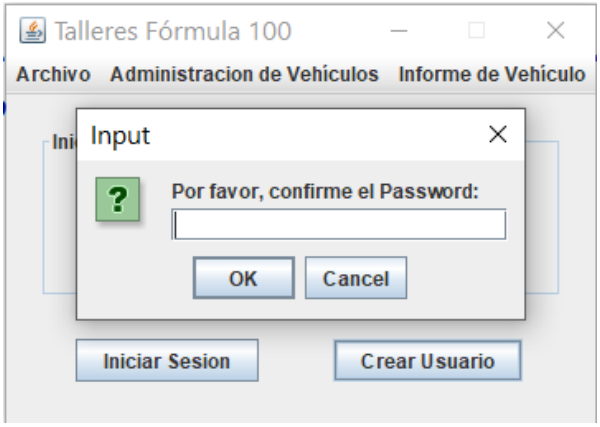
```
public class Controlador {

	//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.