

# Sistema de Gestión de coches - TurboDesk

**TurboDesk** es una aplicación de escritorio en Java con Swing para gestionar un taller de coches. Permite operaciones CRUD sobre clientes, coches, citas y mensajes de soporte, conectándose a una base de datos MySQL mediante JDBC. Incluye procedimientos almacenados, trigger y cursor. La interfaz es intuitiva, con validaciones y manejo de eventos, y cuenta con documentación, pruebas unitarias y control de versiones en GitHub.



[Citas](#)[Soporte](#)

## Funcionalidades Principales de TurboDesk

### Gestión de Coches

Registrar, listar, editar y eliminar coches.



### Gestión de Citas

Crear, listar y eliminar citas, asociando clientes y coches, con validación de fechas para evitar errores.



## Gestión de Soporte

Enviar mensajes de soporte que generan correos automáticamente para atención rápida y eficiente.

Soporte - TurboDesk

Nombre:

Apellido:

Correo Generado:

Mensaje:

Generar Correo

Guardar en BD

Volver al Menú





# Tecnologías Utilizadas en el Proyecto

## Lenguaje y GUI

Java SE 17 con Swing para la interfaz gráfica, ofreciendo una experiencia intuitiva y robusta.

## Base de Datos y Persistencia

MySQL 10.4 con JDBC para conexión y manipulación de datos, incluyendo procedimientos y triggers.

## Control y Pruebas

Git y GitHub para control de versiones; JUnit 5 para pruebas unitarias que garantizan la calidad del código.

## IDE y Documentación

Eclipse como entorno de desarrollo y JavaDoc para documentación completa y accesible.

## Requisitos Previos para la Instalación



### Java SE 17

Descargar desde Oracle para ejecutar la aplicación correctamente.



### MySQL 10.4 o superior

Instalar para gestionar la base de datos del taller.



### Driver JDBC

Descargar mysql-connector-java.jar compatible con MySQL 10.4 para la conexión.



### Eclipse IDE y JUnit 5

Configurar Eclipse para importar el proyecto y ejecutar pruebas unitarias.

## Requisitos Previos para la Instalación

### Java SE 17

Descargar desde Oracle para ejecutar la aplicación correctamente.

### MySQL 10.4 o superior

Instalar para gestionar la base de datos del taller.

### Driver JDBC

Descargar mysql-connector-java.jar compatible con MySQL 10.4 para la conexión.

### Eclipse IDE y JUnit 5

Configurar Eclipse para importar el proyecto y ejecutar pruebas unitarias.



# Configuración del Proyecto en Eclipse

1

## Clonar Repositorio

Usar git clone para obtener el código fuente desde GitHub.

2

## Configurar Base de Datos

Crear la base taller y ejecutar el script SQL para tablas y datos.

3

## Importar Proyecto

Importar en Eclipse como proyecto existente para comenzar el desarrollo.

4

## Añadir Librerías

Agregar mysql-connector-java.jar y JUnit 5 al build path para funcionalidades y pruebas.



# Uso y Ejecución de la Aplicación

## Inicio y Navegación

Ejecutar MenuInicio.java para acceder al menú principal con opciones de coches, citas y soporte.



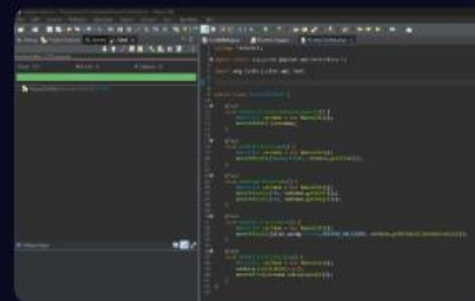
## Gestión Funciones

Registrar y administrar coches, crear y eliminar citas, y enviar mensajes de soporte.



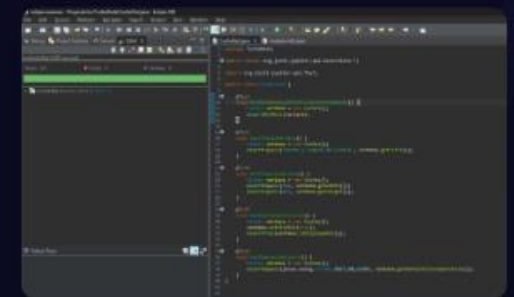
## Pruebas Unitarias

Ejecutar pruebas desde test/TurboDesk para validar la robustez del sistema.

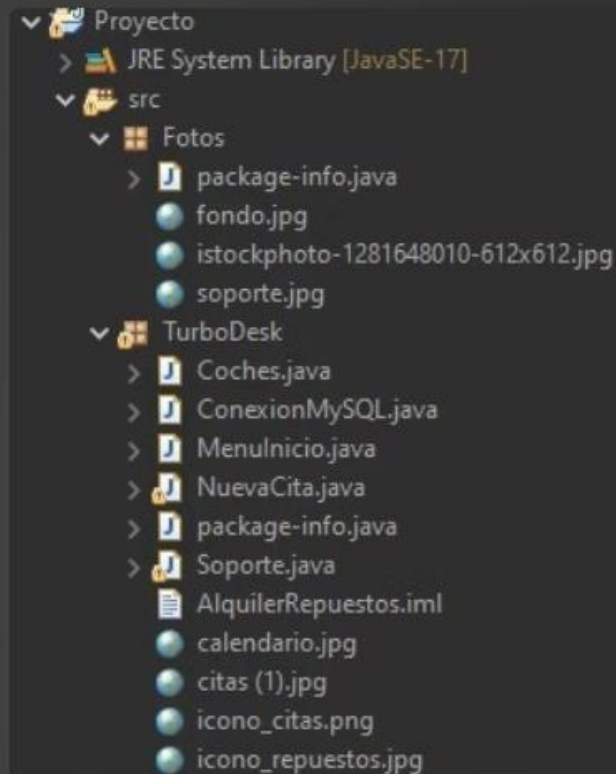


## Pruebas Unitarias

Ejecutar pruebas desde test/TurboDesk para validar la robustez del sistema.



# Estructura y Documentación del Proyecto



## Carpetas Principales

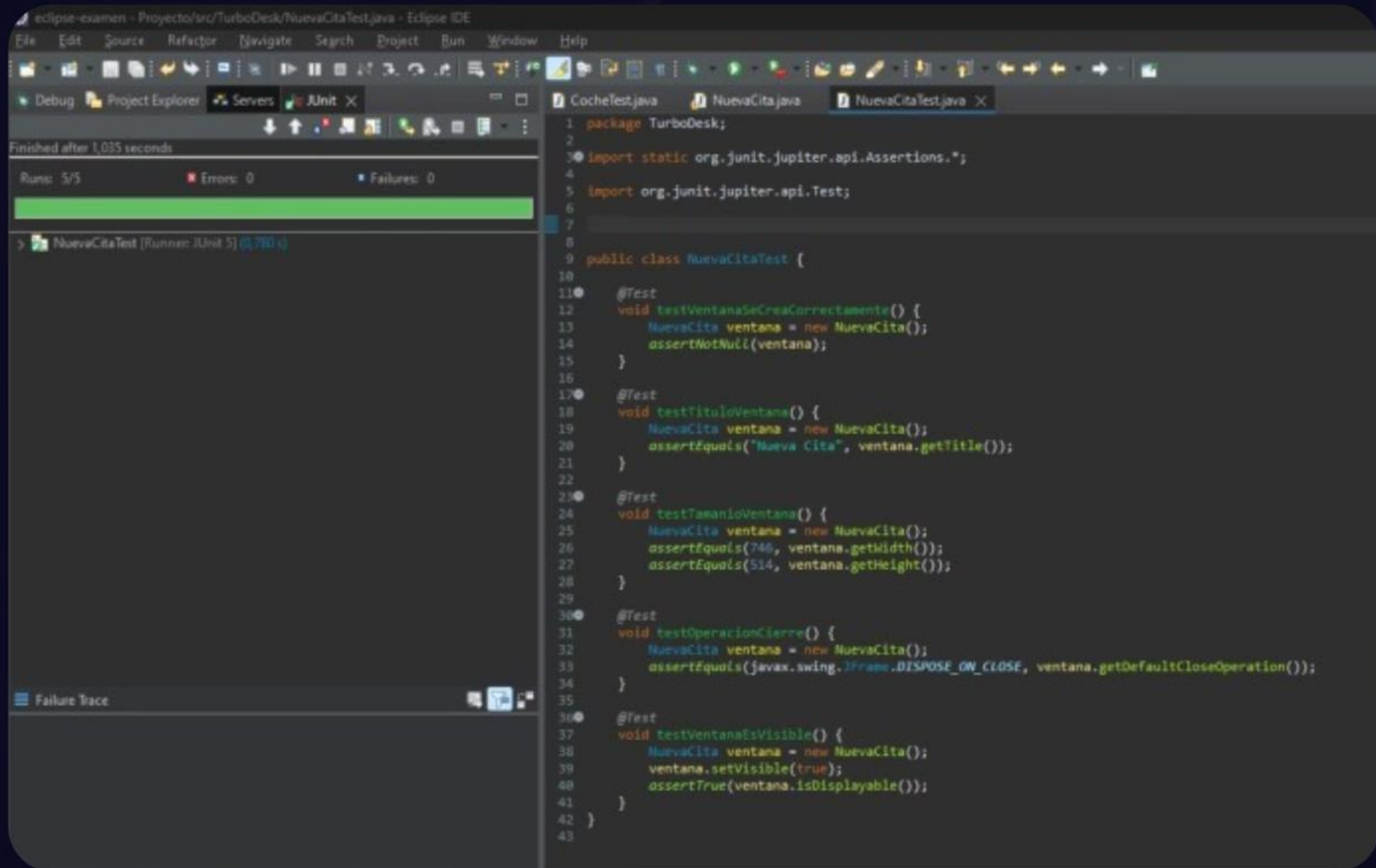
src/TurboDesk con código fuente, test/TurboDesk con pruebas unitarias, sql con scripts y docs con diagramas y JavaDoc.

## Documentación

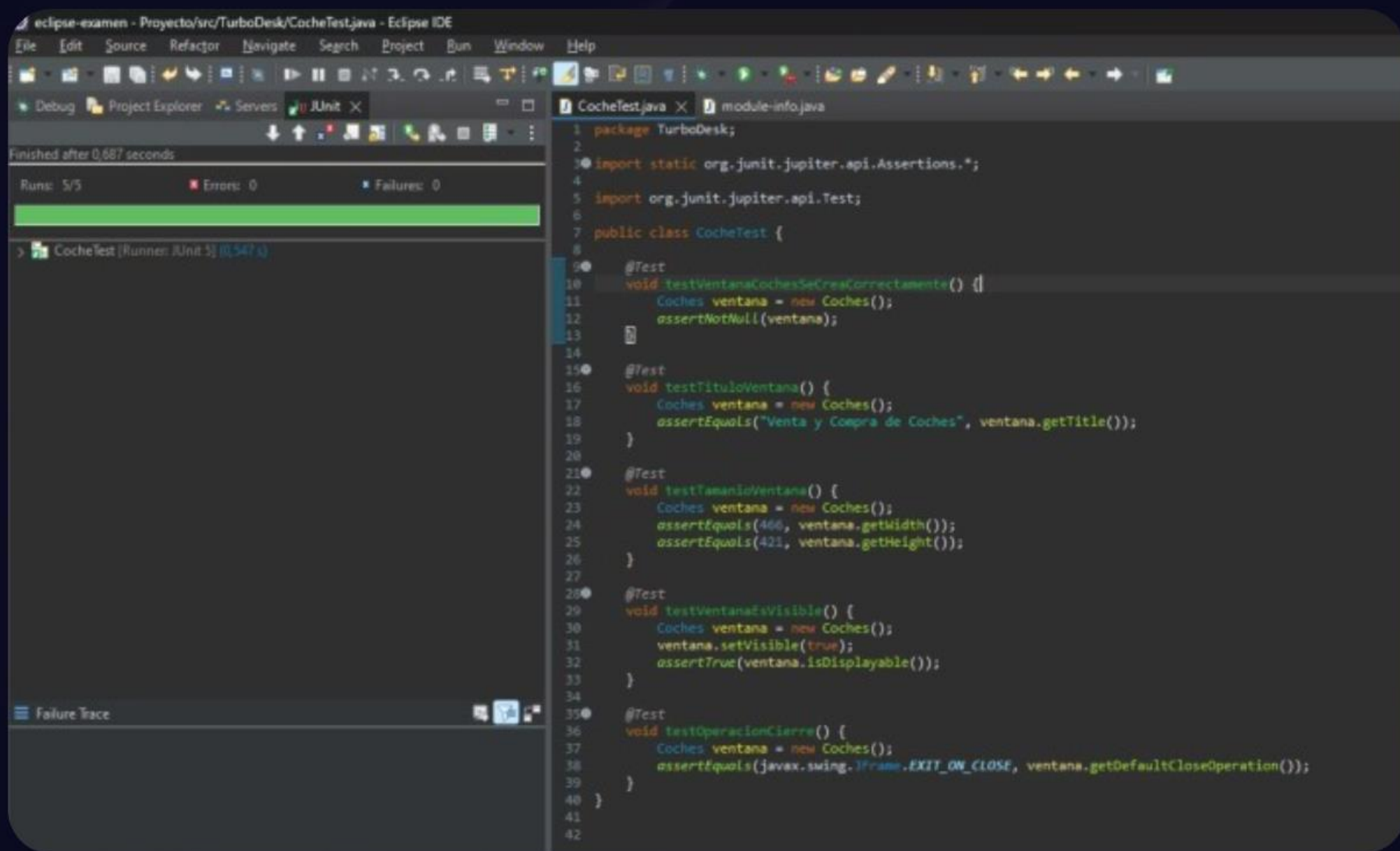
JavaDoc para clases y métodos, diagrama E/R generado con MySQL Workbench y script SQL comentado.



# Pruebas Unitarias NuevaCitas



# Pruebas Unitarias CocheTest



The screenshot shows the Eclipse IDE interface. The top menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar contains various icons for file operations and development. The left sidebar shows the Project Explorer with 'CocheTest.java' selected. Below it, the Run console shows 'Finished after 0.687 seconds' and 'Runs: 5/5', 'Errors: 0', 'Failures: 0'. The main editor displays the code for 'CocheTest.java'.

```
1 package TurboDesk;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 import org.junit.jupiter.api.Test;
6
7 public class CocheTest {
8
9     @Test
10     void testVentanaCochesSeCreaCorrectamente() {
11         Coches ventana = new Coches();
12         assertNotNull(ventana);
13     }
14
15     @Test
16     void testTituloVentana() {
17         Coches ventana = new Coches();
18         assertEquals("Venta y Compra de Coches", ventana.getTitle());
19     }
20
21     @Test
22     void testTamanoVentana() {
23         Coches ventana = new Coches();
24         assertEquals(460, ventana.getWidth());
25         assertEquals(421, ventana.getHeight());
26     }
27
28     @Test
29     void testVentanaEsVisible() {
30         Coches ventana = new Coches();
31         ventana.setVisible(true);
32         assertTrue(ventana.isDisplayable());
33     }
34
35     @Test
36     void testOperacionCierre() {
37         Coches ventana = new Coches();
38         assertEquals(javax.swing.JFrame.EXIT_ON_CLOSE, ventana.getDefaultCloseOperation());
39     }
40 }
41
42
```

# Refactorizacion en Coches

## 1. Separación de métodos por funcionalidad

El código separa las operaciones en métodos claros y específicos:

`mostrarFormularioVenta()`

`insertarCoche(...)`

`mostrarListaCoches()`

`eliminarCoche(...)`

`mostrarFormularioEdicion(...)`

`actualizarCoche(...)`

## 2. Uso de JDialog para formularios modales

Para registrar o editar coches, el código usa JDialog en lugar de abrir una nueva ventana JFrame.

`JDialog dialog = new JDialog(this, "Registrar Coche", true);`



# Refactorizacion en Coches

## 3. Control de errores con try-catch

Las operaciones con base de datos están envueltas en bloques try-catch, capturando tanto `SQLException` como `NumberFormatException`.

```
} catch (NumberFormatException | SQLException ex) {  
    ex.printStackTrace();  
    JOptionPane.showMessageDialog(this, "Error al registrar el coche: " + ex.getMessage());  
}
```

## 4. Uso de `SwingUtilities.invokeLater` en `main()`

Esto garantiza que la interfaz gráfica se cree en el hilo correcto del EDT (Event Dispatch Thread):

```
public static void main(String[] args) {  
    SwingUtilities.invokeLater(() -> (new Coches()).setVisible(true));  
}
```

## 5. Uso de `JOptionPane` para interacciones rápidas

El uso de `JOptionPane.showMessageDialog()` y `showOptionDialog()` para mostrar listas e interactuar es simple pero funcional.

```
int opcion = JOptionPane.showOptionDialog(this, scrollPane, "Selecciona una opción", -1, -1, null, options, options[2]);
```

# Refactorizacion en Citas

## 1. Uso correcto de Swing en el hilo EDT

Creación de la ventana principal dentro de `SwingUtilities.invokeLater()`:

```
public static void main(String[] args) {  
    SwingUtilities.invokeLater(() -> new NuevaCita().setVisible(true));  
}
```

## 2. Validaciones previas antes de operaciones críticas

Se valida que nombre, apellidos y fecha estén presentes antes de insertar la cita:

```
if (nombre.isEmpty() || apellidos.isEmpty()) {  
    JOptionPane.showMessageDialog(this, "Por favor, completa todos los campos.");  
    return;  
}
```