|  |  |
| --- | --- |
| **Icono  Descripción generada automáticamente** | **Universidad Peruana Los Andes**  Facultad de Ingeniería  **Escuela Profesional de Sistemas y Computación** |

**Proyecto Final**

Asignatura: **Taller de Aplicaciones I**

Código: M00482F Apellidos y Nombres: Palma Baldeón Rogger Josué

Código: S01251J Apellidos y Nombres: Espinoza Fernández Oliver

Fecha: 22.10.24

1. **Enunciado del proyecto**

En el contexto de una clínica médica, la gestión eficiente de la información de los pacientes, tratamientos y citas es fundamental para garantizar una atención de calidad y un flujo de trabajo organizado. Sin un sistema adecuado para registrar y acceder a los datos de los pacientes, los profesionales de la salud enfrentan dificultades para mantener un seguimiento de los tratamientos, programar citas y comunicarse con los pacientes de manera efectiva.

Este código aborda la necesidad de un sistema de gestión integral para la clínica, que permita a los médicos y al personal administrativo:

* **Registrar y acceder a la información de los pacientes**: Almacenar datos relevantes como nombre, fecha de nacimiento, dirección, teléfono y correo electrónico, así como la descripción de tratamientos, fechas de inicio y fin, y observaciones.
* **Consultar la información médica**: Permitir a los profesionales de la salud obtener rápidamente información sobre los pacientes y sus tratamientos a través de un sistema de búsqueda que facilita la consulta de datos por identificador.
* **Gestionar las citas**: Facilitar la programación y seguimiento de citas médicas, asegurando que tanto los médicos como los pacientes tengan acceso a la información actualizada.
* **Mejorar la comunicación**: Proporcionar un medio para que el personal médico se comunique eficientemente con los pacientes sobre su estado de salud y las recomendaciones de tratamiento.

1. **Identificación de requerimientos**

|  |  |
| --- | --- |
| **Nombre:** | **Botones de Registros** |
| **Resumen:** | Este requisito define la funcionalidad de los botones de registro que permiten al usuario cambiar entre las diferentes visualizaciones de las tablas en la aplicación de la clínica. Los botones proporcionan una forma intuitiva de navegar entre las secciones que contienen información sobre pacientes, tratamientos, doctores y citas. Al hacer clic en cada botón, se mostrará la tabla correspondiente, facilitando el acceso y la gestión de datos relevantes. |
| **Entradas:** | **Botón "Pacientes":** Al hacer clic en este botón, se solicita al sistema que muestre la tabla de pacientes.  **Botón "Tratamientos":** Al hacer clic en este botón, se solicita al sistema que muestre la tabla de tratamientos.  **Botón "Doctores":** Al hacer clic en este botón, se solicita al sistema que muestre la tabla de doctores.  **Botón "Citas":** Al hacer clic en este botón, se solicita al sistema que muestre la tabla de citas. |
| **Resultados:** | **Visualización de la tabla seleccionada:** Al hacer clic en uno de los botones, la tabla correspondiente (pacientes, tratamientos, doctores, citas) se mostrará en la interfaz, reemplazando cualquier tabla previamente visible. |

|  |  |
| --- | --- |
| **Nombre:** | **Botón Guardar** |
| **Resumen:** | Este requisito define la funcionalidad del botón "Guardar", que permite al usuario guardar o actualizar la información ingresada en las tablas de pacientes, tratamientos, doctores y citas en la base de datos. Al hacer clic en este botón, se realizará la validación de los datos y se ejecutará la inserción o actualización correspondiente. |
| **Entradas:** | **Datos del paciente:** Nombre, fecha de nacimiento, dirección, teléfono, email.  **Datos del tratamiento:** Descripción del tratamiento, fecha de inicio, fecha de fin, observaciones.  **Datos del doctor:** Nombre, especialidad, teléfono, correo.  **Datos de la cita:** ID de paciente, ID de doctor, fecha y hora de la cita, estado. |
| **Resultados:** | **Almacenamiento exitoso: Los datos ingresados se guardarán en la base de datos, asegurando que la información esté disponible para futuras consultas.**  **Mensaje de confirmación: Se mostrará un mensaje al usuario indicando que los datos se han guardado exitosamente o que se ha producido un error.**  **Actualización de la interfaz: Tras guardar los datos, la tabla correspondiente se actualizará automáticamente para reflejar los cambios realizados.**  **Validación de datos: Se llevará a cabo una verificación de que todos los campos requeridos estén completos y en el formato correcto antes de proceder con el guardado.** |

|  |  |
| --- | --- |
| **Nombre:** | **Botón Modificar** |
| **Resumen:** | Este requisito define la funcionalidad del botón "Modificar", que permite al usuario editar la información existente de los pacientes, tratamientos, doctores y citas en la base de datos. Al hacer clic en este botón, se cargarán los datos actuales del elemento seleccionado en los campos de entrada para su edición. Al confirmar los cambios, se actualizarán los datos en la base de datos. |
| **Entradas:** | **Datos del paciente:** Nombre, fecha de nacimiento, dirección, teléfono, email.  **Datos del tratamiento:** Descripción del tratamiento, fecha de inicio, fecha de fin, observaciones.  **Datos del doctor:** Nombre, apellido, especialidad, ID de doctor (no modificable).  **Datos de la cita:** ID de paciente (no modificable), ID de doctor (no modificable), fecha y hora de la cita, estado. |
| **Resultados:** | **Actualización de datos:** Los datos modificados se actualizarán en la base de datos, asegurando que la información esté precisa y actualizada.  **Mensaje de confirmación:** Se mostrará un mensaje al usuario indicando que los datos se han modificado exitosamente o que ha ocurrido un error durante el proceso de modificación.  **Actualización de la interfaz:** La tabla correspondiente se actualizará automáticamente para reflejar los cambios realizados en la información.  **Validación de datos:** Se llevará a cabo una verificación para asegurar que todos los campos sean válidos y estén completos antes de proceder con la modificación. |

|  |  |
| --- | --- |
| **Nombre:** | **Botón Eliminar** |
| **Resumen:** | Este requisito define la funcionalidad del botón "Eliminar", que permite al usuario borrar la información de un paciente, tratamiento, doctor o cita de la base de datos. Al hacer clic en este botón, se solicitará confirmación al usuario para asegurar que desea proceder con la eliminación. Una vez confirmado, se eliminarán los datos seleccionados de la base de datos. |
| **Entradas:** | **Datos del paciente:** ID del paciente a eliminar.  **Datos del tratamiento:** ID del tratamiento a eliminar.  **Datos del doctor:** ID del doctor a eliminar.  **Datos de la cita:** ID de la cita a eliminar. |
| **Resultados:** | **Eliminación exitosa:** Los datos seleccionados se eliminarán de la base de datos, asegurando que la información no esté disponible para futuras consultas.  **Mensaje de confirmación:** Se mostrará un mensaje al usuario indicando que los datos se han eliminado exitosamente o que ha ocurrido un error durante el proceso de eliminación.  **Actualización de la interfaz:** La tabla correspondiente se actualizará automáticamente para reflejar la eliminación realizada, mostrando la información más actualizada.  **Validación de eliminación:** Se solicitará confirmación al usuario antes de proceder con la eliminación para evitar acciones accidentales. |

1. **Enlace del proyecto en GitHub**

**https://github.com/1234Oliver/Proyecto\_Taller**

1. **Diseño de Base de Datos (Script)**

USE [Clinica]

GO

/\*\*\*\*\*\* Object: Table [dbo].[Citas] Script Date: 25/10/2024 04:08:13 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[Citas](

[ID] [int] IDENTITY(1,1) NOT NULL,

[ID\_Paciente] [int] NULL,

[ID\_Doctor] [int] NULL,

[FechaHora] [datetime] NULL,

[Motivo] [varchar](255) NULL,

[Estado] [varchar](50) NULL,

PRIMARY KEY CLUSTERED

(

[ID] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY]

GO

/\*\*\*\*\*\* Object: Table [dbo].[Doctores] Script Date: 25/10/2024 04:08:13 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[Doctores](

[ID] [int] IDENTITY(1,1) NOT NULL,

[Nombre] [varchar](100) NULL,

[Especialidad] [varchar](100) NULL,

[Telefono] [varchar](20) NULL,

[Email] [varchar](100) NULL,

PRIMARY KEY CLUSTERED

(

[ID] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY]

GO

/\*\*\*\*\*\* Object: Table [dbo].[Pacientes] Script Date: 25/10/2024 04:08:13 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[Pacientes](

[ID] [int] IDENTITY(1,1) NOT NULL,

[Nombre] [varchar](100) NULL,

[FechaNacimiento] [date] NULL,

[Direccion] [varchar](255) NULL,

[Telefono] [varchar](20) NULL,

[Email] [varchar](100) NULL,

PRIMARY KEY CLUSTERED

(

[ID] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY]

GO

/\*\*\*\*\*\* Object: Table [dbo].[Tratamientos] Script Date: 25/10/2024 04:08:13 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[Tratamientos](

[ID] [int] IDENTITY(1,1) NOT NULL,

[ID\_Paciente] [int] NULL,

[DescripcionTratamiento] [varchar](255) NULL,

[FechaInicio] [date] NULL,

[FechaFin] [date] NULL,

[Observaciones] [text] NULL,

PRIMARY KEY CLUSTERED

(

[ID] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]

GO

SET IDENTITY\_INSERT [dbo].[Doctores] ON

GO

INSERT [dbo].[Doctores] ([ID], [Nombre], [Especialidad], [Telefono], [Email]) VALUES (1, N'juan', N'medico', N'987654321', N'sa@gmail.com')

GO

SET IDENTITY\_INSERT [dbo].[Doctores] OFF

GO

SET IDENTITY\_INSERT [dbo].[Pacientes] ON

GO

INSERT [dbo].[Pacientes] ([ID], [Nombre], [FechaNacimiento], [Direccion], [Telefono], [Email]) VALUES (2, N'Señor Bolainas', CAST(N'2004-12-07' AS Date), N'Calle Falsa 123', N'666666666', N'ThuXamakito@bb.pe')

GO

INSERT [dbo].[Pacientes] ([ID], [Nombre], [FechaNacimiento], [Direccion], [Telefono], [Email]) VALUES (5, N'Juan Pérez López', CAST(N'2000-12-12' AS Date), N'Av. Los Pinos 123, Lima, Perú', N'987654321', N' juan.perez85@gmail.com')

GO

INSERT [dbo].[Pacientes] ([ID], [Nombre], [FechaNacimiento], [Direccion], [Telefono], [Email]) VALUES (7, N'María Gómez Fernández', CAST(N'2005-10-10' AS Date), N'Calle Las Flores 45, Arequipa, Perú', N'976543210', N'maria.gomez92@hotmail.com')

GO

SET IDENTITY\_INSERT [dbo].[Pacientes] OFF

GO

SET IDENTITY\_INSERT [dbo].[Tratamientos] ON

GO

INSERT [dbo].[Tratamientos] ([ID], [ID\_Paciente], [DescripcionTratamiento], [FechaInicio], [FechaFin], [Observaciones]) VALUES (3, 2, N'efefewefsefe', CAST(N'2012-12-12' AS Date), CAST(N'2011-12-12' AS Date), N'efefefsef')

GO

SET IDENTITY\_INSERT [dbo].[Tratamientos] OFF

GO

ALTER TABLE [dbo].[Citas] WITH CHECK ADD CONSTRAINT [FK\_Citas\_Doctores] FOREIGN KEY([ID\_Doctor])

REFERENCES [dbo].[Doctores] ([ID])

GO

ALTER TABLE [dbo].[Citas] CHECK CONSTRAINT [FK\_Citas\_Doctores]

GO

ALTER TABLE [dbo].[Citas] WITH CHECK ADD CONSTRAINT [FK\_Citas\_Pacientes] FOREIGN KEY([ID\_Paciente])

REFERENCES [dbo].[Pacientes] ([ID])

GO

ALTER TABLE [dbo].[Citas] CHECK CONSTRAINT [FK\_Citas\_Pacientes]

GO

ALTER TABLE [dbo].[Tratamientos] WITH CHECK ADD CONSTRAINT [FK\_Tratamientos\_Pacientes] FOREIGN KEY([ID\_Paciente])

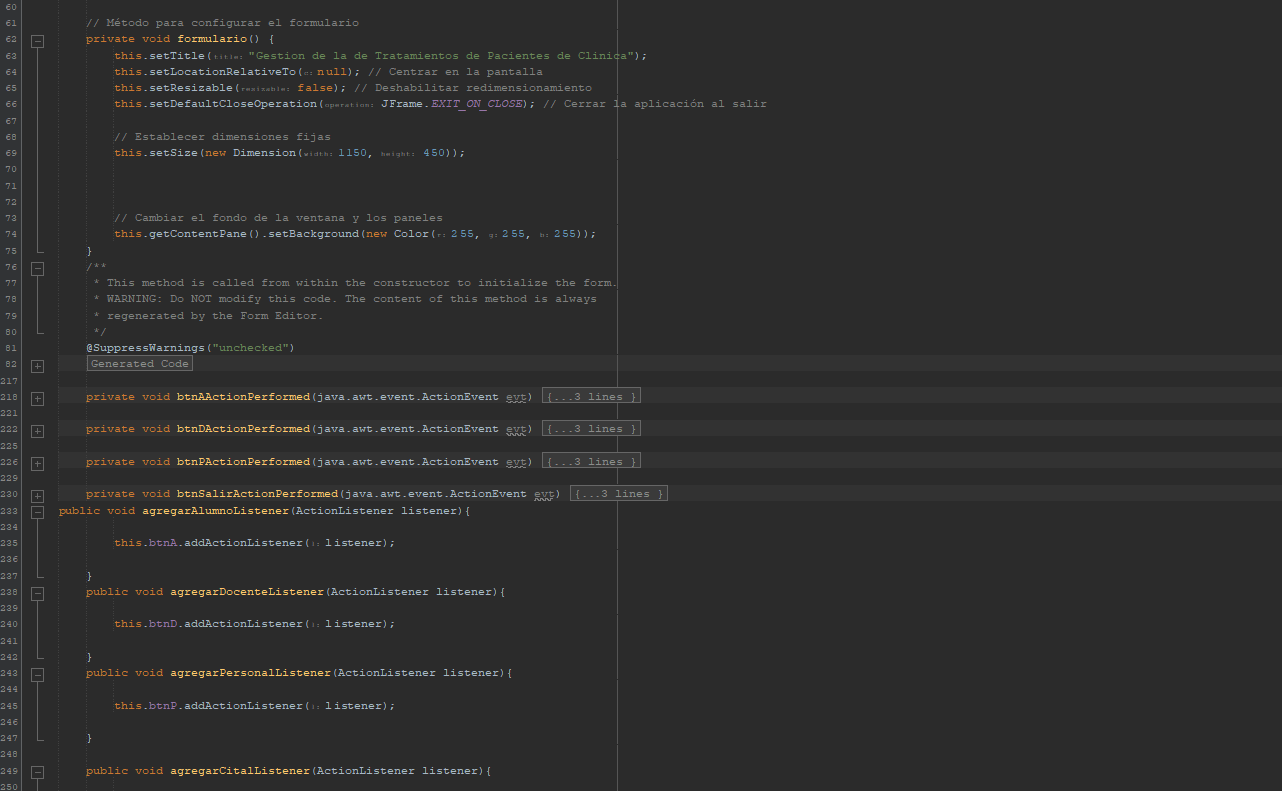
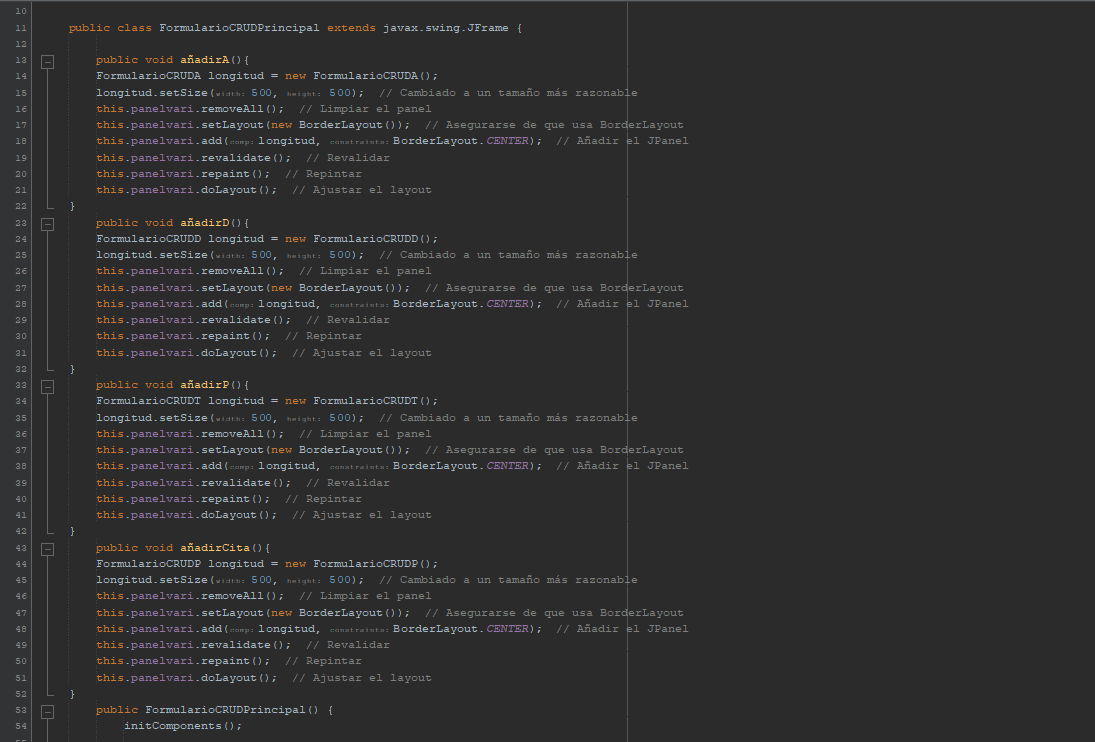
REFERENCES [dbo].[Pacientes] ([ID])

GO

ALTER TABLE [dbo].[Tratamientos] CHECK CONSTRAINT [FK\_Tratamientos\_Pacientes]

GO

1. **Código explicado del proyecto**

****

### Estructura General

1. **Paquete y Importaciones:**
   * El código está ubicado en el paquete Vista, y se importan varias clases necesarias para la creación de la interfaz gráfica y la conexión a la base de datos.
2. **Clase FormularioCRUDPrincipal:**
   * Esta clase principal crea la ventana principal de la aplicación y gestiona los formularios relacionados con pacientes, doctores, tratamientos y citas.

### Constructor

* **FormularioCRUDPrincipal()**:
  + Se llama al método initComponents() para inicializar los componentes de la interfaz gráfica.
  + Se crea una instancia de CConexion, que se encarga de establecer una conexión con la base de datos.
  + Se llama al método formulario(), que configura la ventana, incluyendo el título, la ubicación y el color de fondo.

### Métodos de Añadir Formularios

Los métodos añadirA(), añadirD(), añadirP() y añadirCita():

* Cada uno de estos métodos crea una nueva instancia de un formulario correspondiente (pacientes, doctores, tratamientos o citas).
* Configuran el tamaño del nuevo panel, limpian el panel existente (panelvari), y añaden el nuevo formulario al panel.
* Luego, revalidan, repintan y ajustan el diseño del panel para reflejar el cambio.

### Métodos de Configuración

* **formulario()**: Configura la ventana principal, estableciendo el título, la ubicación, las dimensiones y el color de fondo.

### Inicialización de Componentes

* **initComponents()**:
  + Este método genera y organiza todos los componentes de la interfaz gráfica, incluidos botones y paneles. Los botones permiten al usuario seleccionar qué formulario abrir (pacientes, doctores, tratamientos, citas, o salir de la aplicación).
  + Cada botón tiene un ActionListener asociado, que permite manejar eventos cuando se hace clic en ellos. Sin embargo, los métodos para manejar los clics (btnAActionPerformed, btnDActionPerformed, etc.) están definidos pero no implementados.

### Métodos de Agregación de Listeners

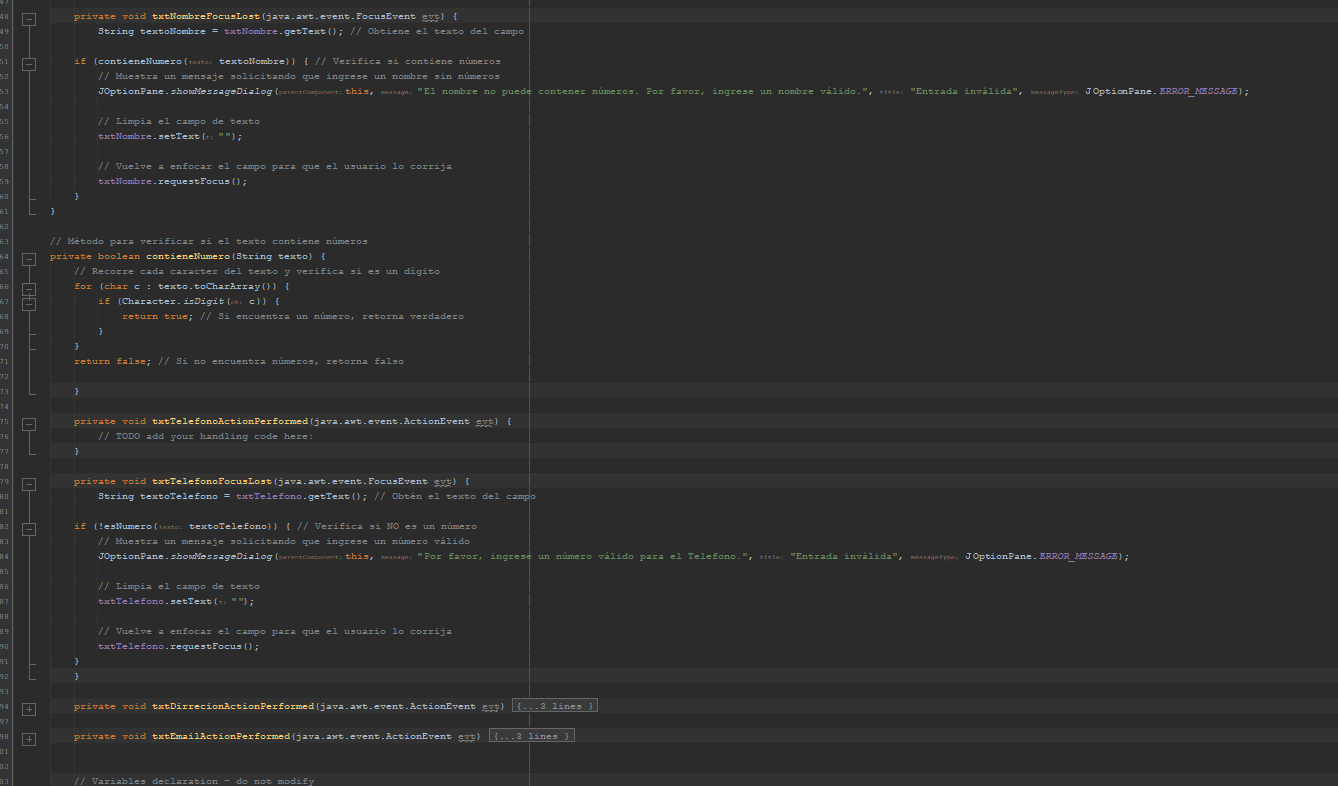
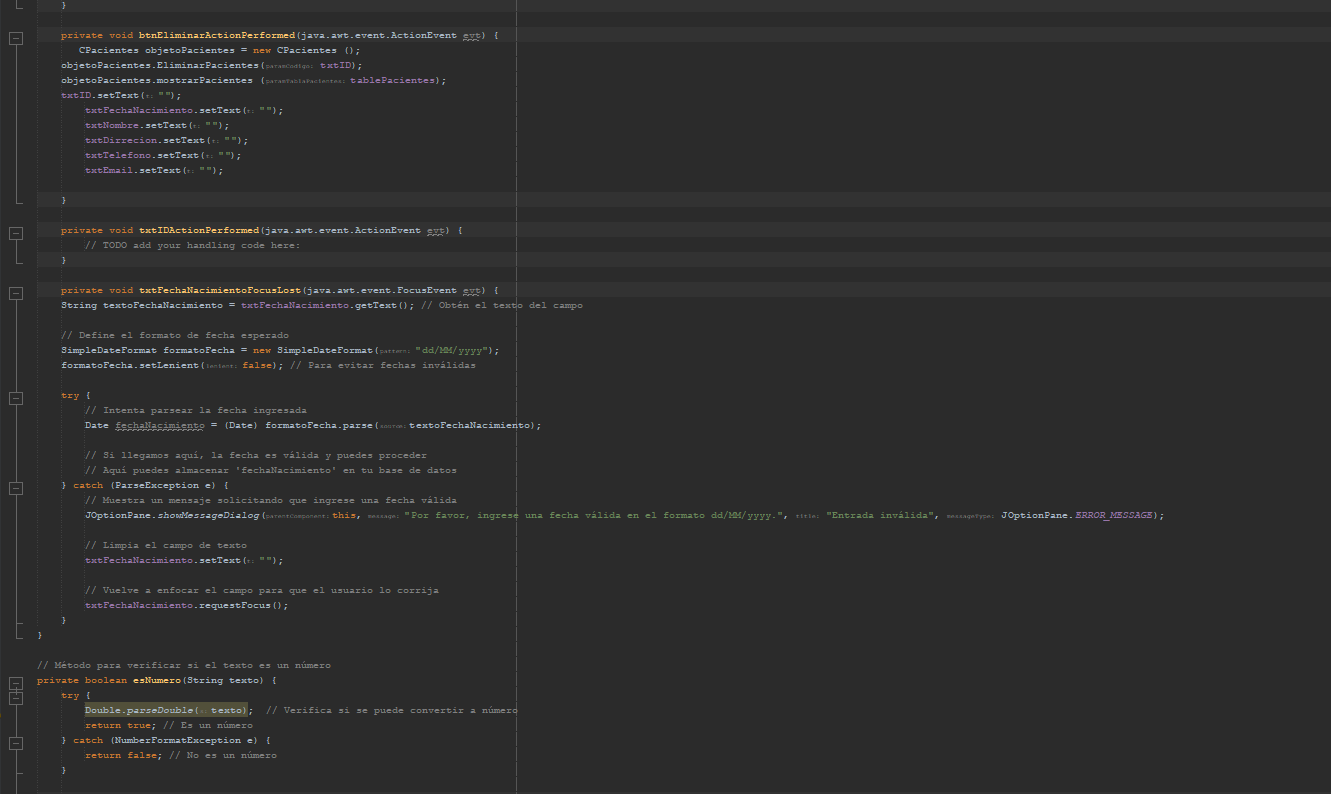
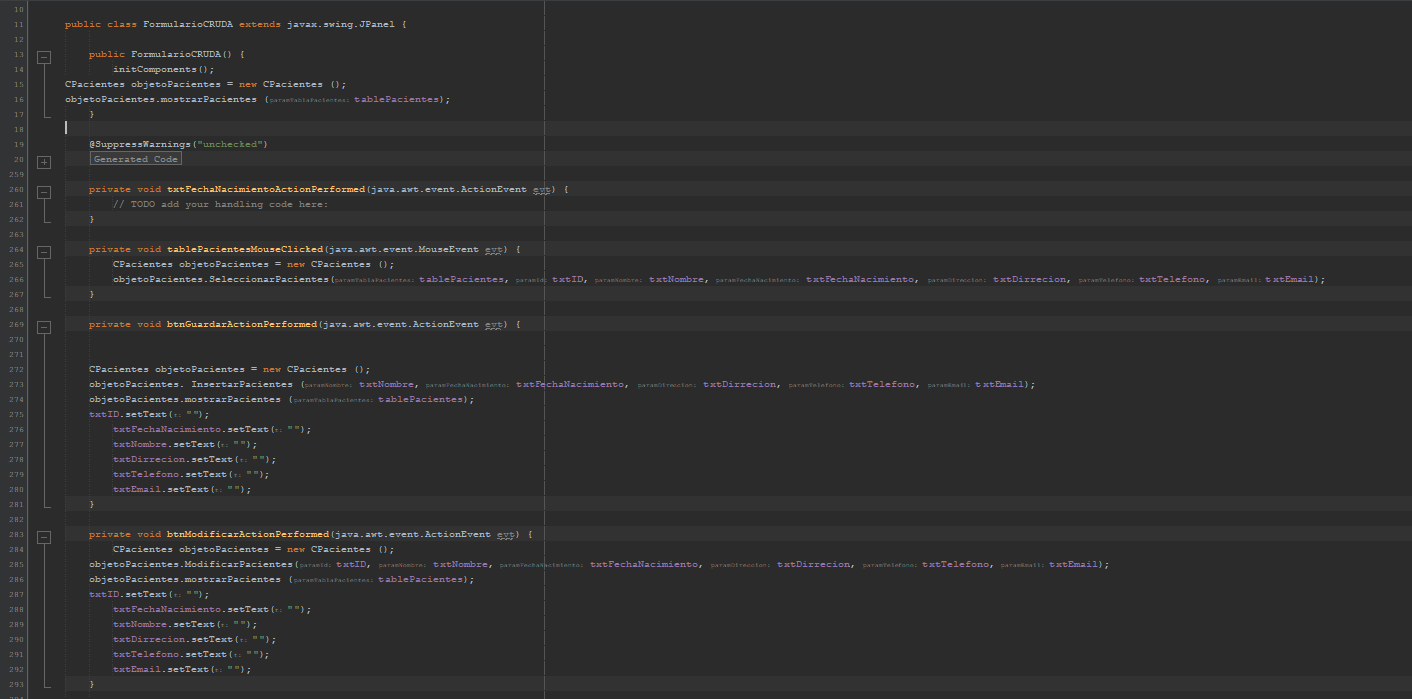
* Los métodos agregarAlumnoListener, agregarDocenteListener, agregarPersonalListener, agregarCitalListener, y agregarSalirListener permiten añadir ActionListener a los botones para manejar eventos de clics. Esto proporciona flexibilidad para definir la lógica específica en otros lugares del código.

### Método main

* **main(String[] args)**:
  + Este es el punto de entrada de la aplicación. Configura el look and feel de la aplicación y crea una nueva instancia de FormularioCRUDPrincipal, haciéndola visible.

### Componentes de la Interfaz

* La interfaz incluye:
  + Un panel (jPanel1) que contiene botones para seleccionar diferentes formularios.
  + Un panel (panelvari) donde se cargan los formularios seleccionados.
  + Etiquetas y botones con configuraciones de estilo y colores para mejorar la experiencia del usuario.



### Componentes Principales

1. **Atributos de la Clase**:
   * Contiene campos de texto para ingresar los datos del paciente: txtID, txtNombre, txtFechaNacimiento, txtDirrecion, txtTelefono, y txtEmail.
   * Botones para realizar acciones: btnGuardar, btnModificar, y btnEliminar.
   * Una tabla (tablePacientes) para mostrar la lista de pacientes.
2. **Constructor**:
   * Al inicializar el formulario, crea una instancia de CPacientes y llama al método mostrarPacientes para cargar los datos de los pacientes en la tabla.
3. **Métodos de Inicialización**:
   * initComponents(): Configura la interfaz gráfica, incluyendo la disposición de los paneles y la configuración de los componentes visuales.

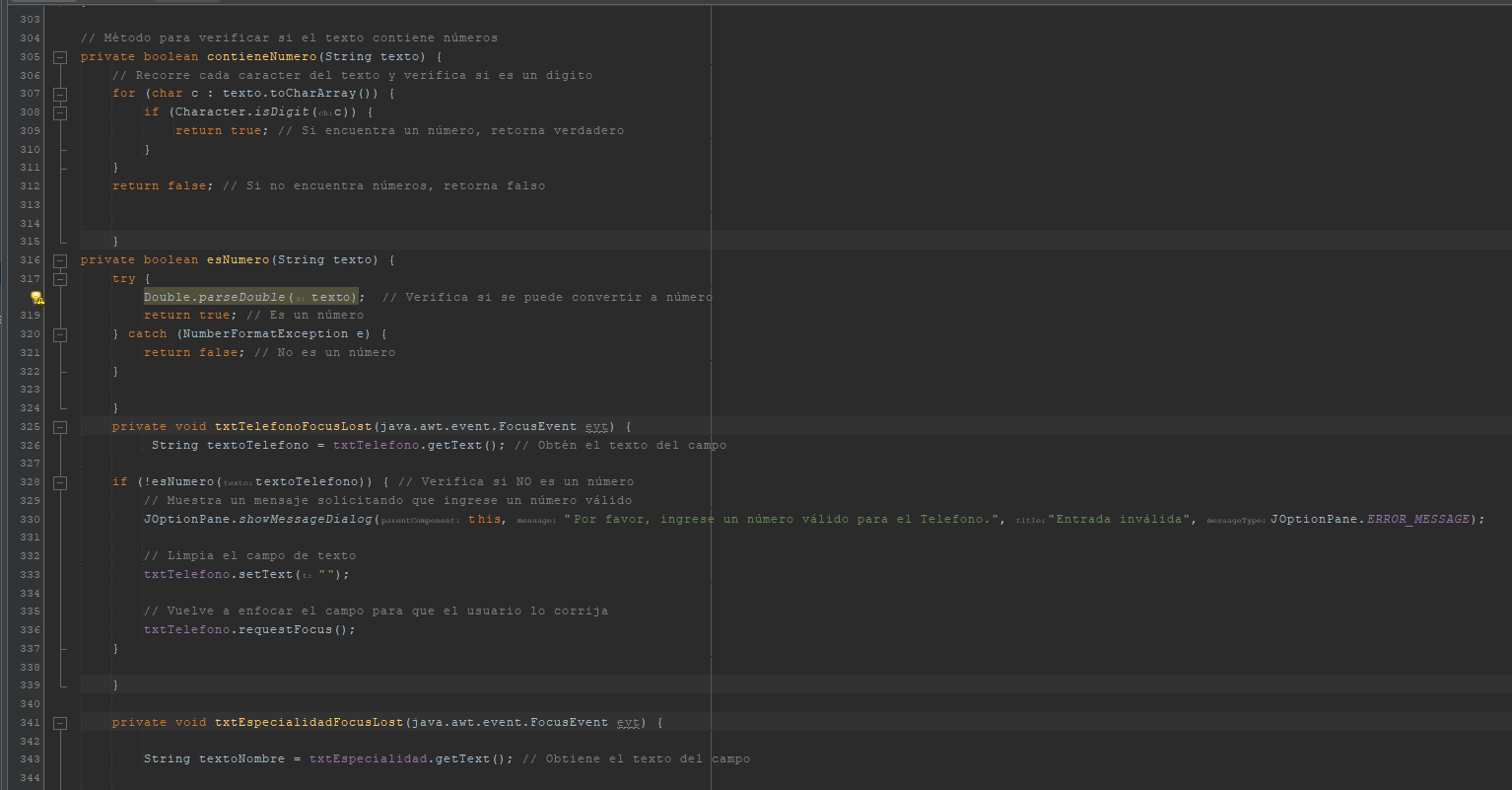
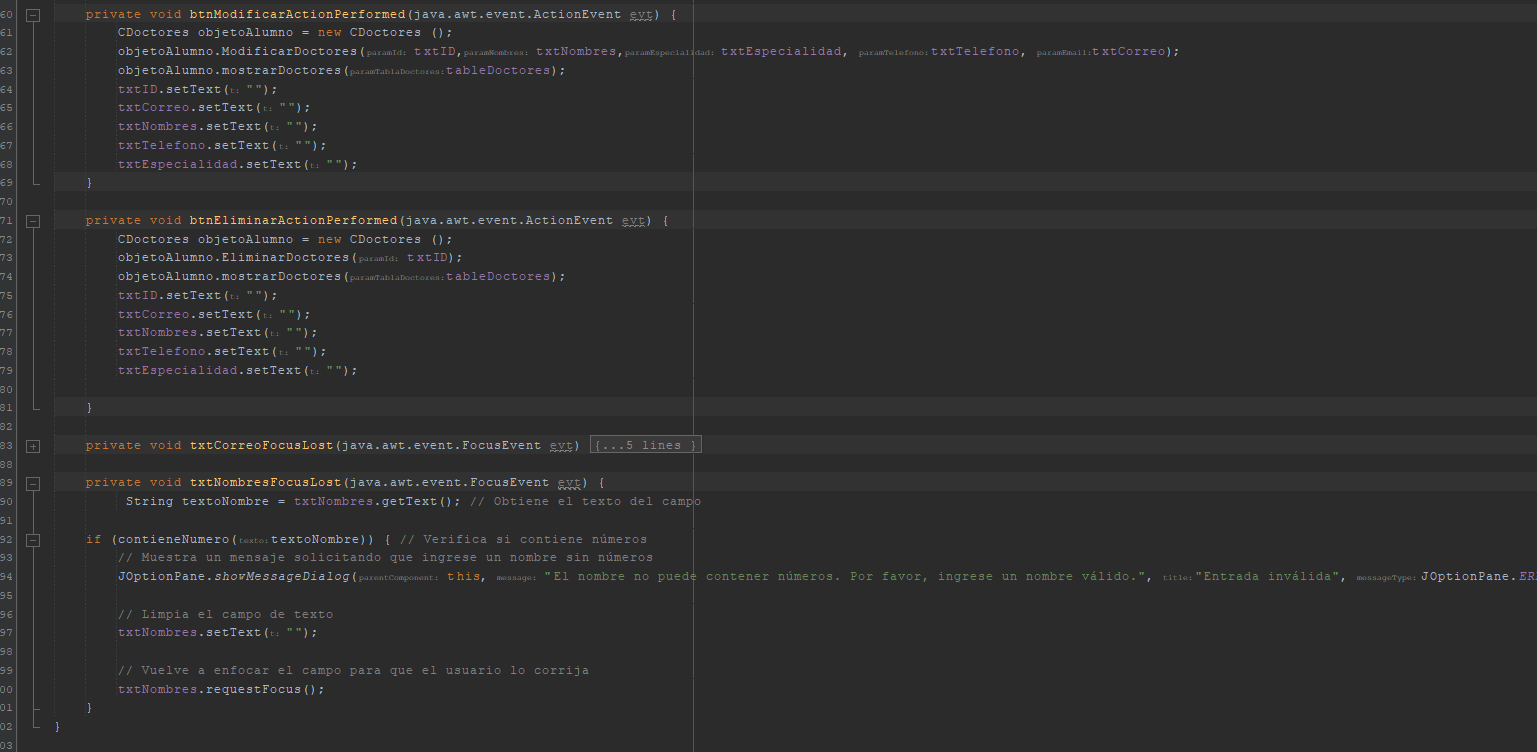
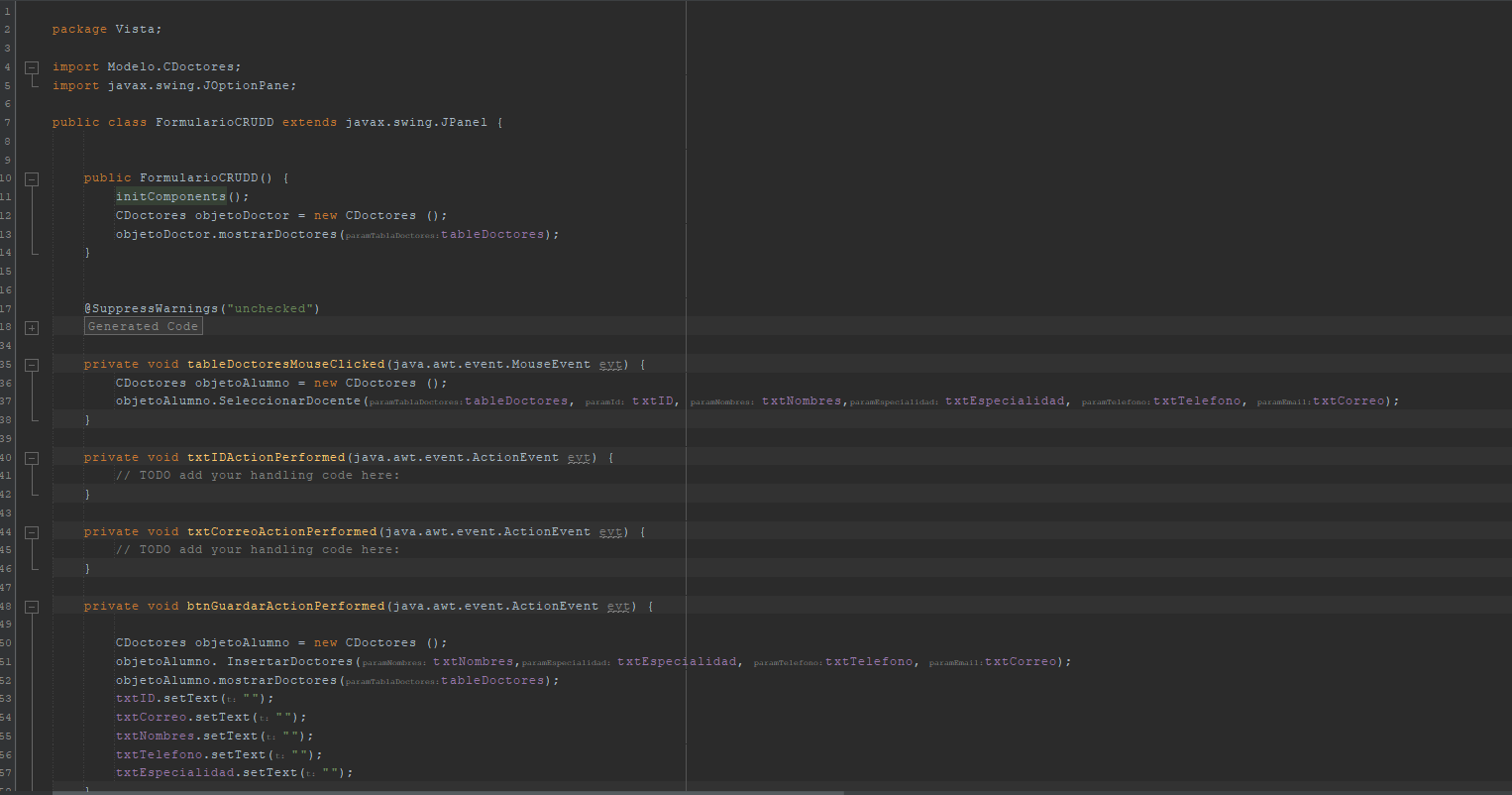
### Funcionalidades

1. **Guardar Paciente**:
   * Al hacer clic en el botón "GUARDAR", se llama a InsertarPacientes de la clase CPacientes, pasando los datos ingresados por el usuario. Luego, actualiza la tabla con mostrarPacientes.
2. **Modificar Paciente**:
   * Al hacer clic en "Modificar", se llama a ModificarPacientes, pasando el ID y los datos del paciente a actualizar. Después, actualiza la tabla.
3. **Eliminar Paciente**:
   * Al hacer clic en "Eliminar", se llama a EliminarPacientes, pasando el ID del paciente a eliminar y actualizando la tabla.

### Validaciones

1. **Validación de la Fecha**:
   * Al perder el foco el campo de fecha de nacimiento, se valida que la fecha ingresada sea correcta y esté en el formato dd/MM/yyyy. Si no es válida, se muestra un mensaje de error y se limpia el campo.
2. **Validación de Nombre**:
   * Al perder el foco el campo de nombre, se verifica que no contenga números. Si los contiene, se muestra un mensaje de error y se limpia el campo.
3. **Validación de Teléfono**:
   * Al perder el foco el campo de teléfono, se valida que el texto ingresado sea un número. Si no lo es, se muestra un mensaje de error y se limpia el campo.

---------------------------------------------------------------------------------------------------------------------------



### Descripción General

* **Clase FormularioCRUDD:** Esta clase extiende javax.swing.JPanel, lo que significa que representa un panel que puede contener otros componentes gráficos (como botones, etiquetas, y tablas). El nombre sugiere que está diseñado para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre registros de doctores.

### Componentes Clave

* **Constructor (FormularioCRUDD):**
  + Inicializa los componentes de la interfaz gráfica mediante el método initComponents().
  + Crea una instancia de CDoctores, que es una clase del modelo (probablemente parte de la lógica de negocio o acceso a datos) y llama al método mostrarDoctores(tableDoctores) para cargar los datos de los doctores en una tabla.
* **Método initComponents():**
  + Aquí se configuran y organizan los diferentes componentes de la interfaz, incluidos:
    - **Tabla (tableDoctores):** Muestra la lista de doctores. Permite seleccionar un registro haciendo clic en él.
    - **Etiquetas (jLabel1 a jLabel5):** Proporcionan descripciones para cada campo de entrada.
    - **Campos de texto (txtID, txtCorreo, txtNombres, txtTelefono, txtEspecialidad):** Permiten la entrada de datos del usuario.
    - **Botones (btnGuardar, btnModificar, btnEliminar):** Ejecutan acciones CRUD.

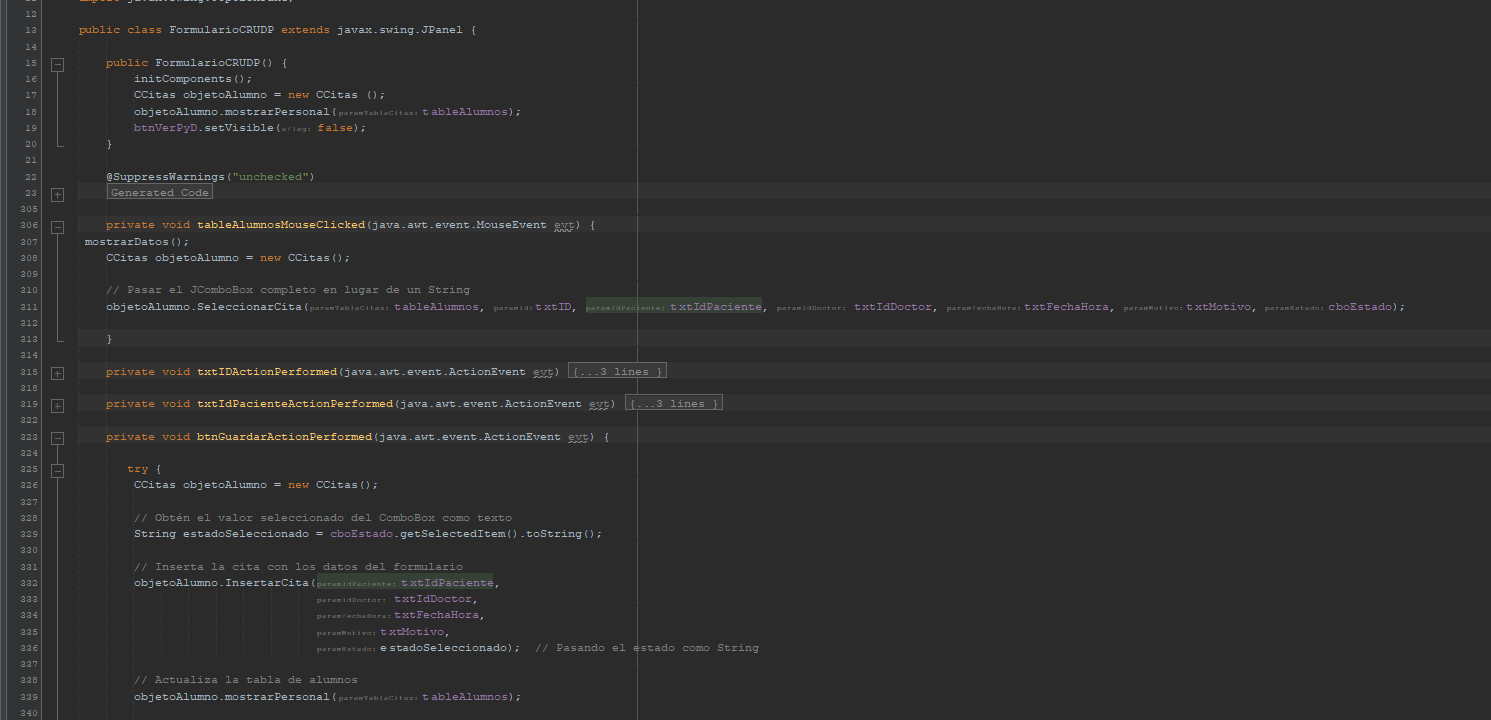
### Funcionalidad

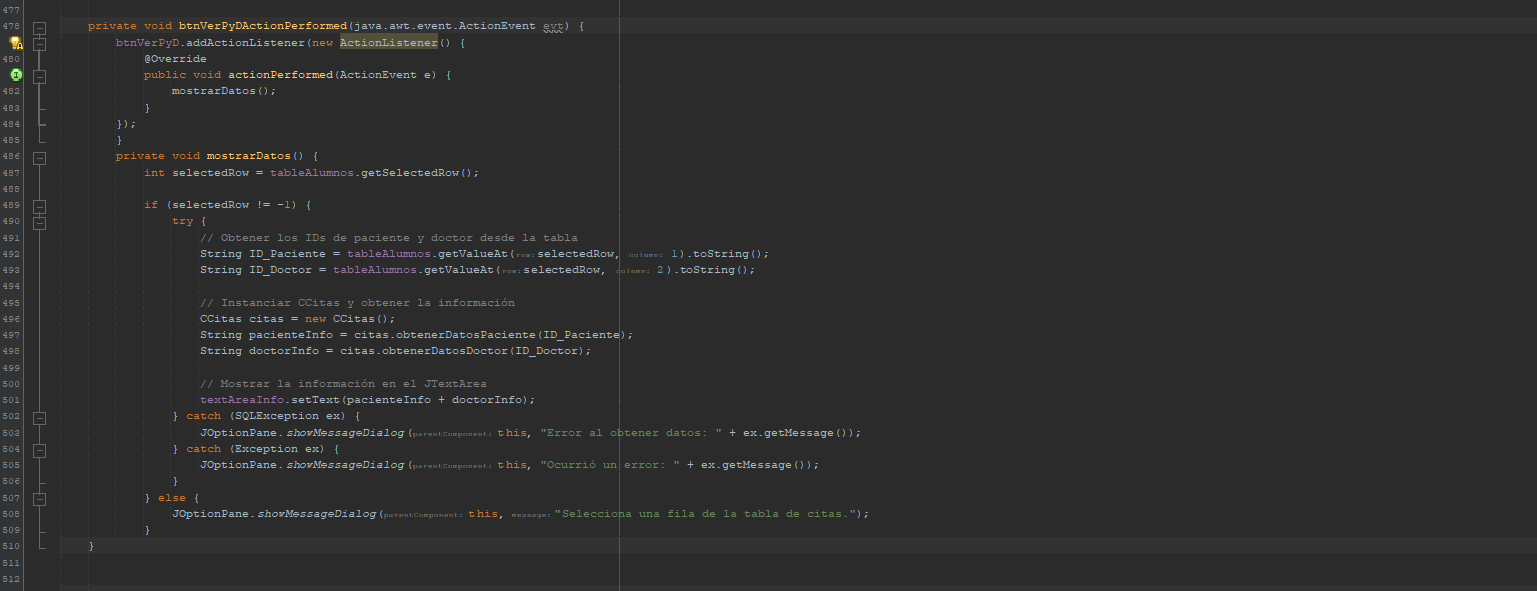
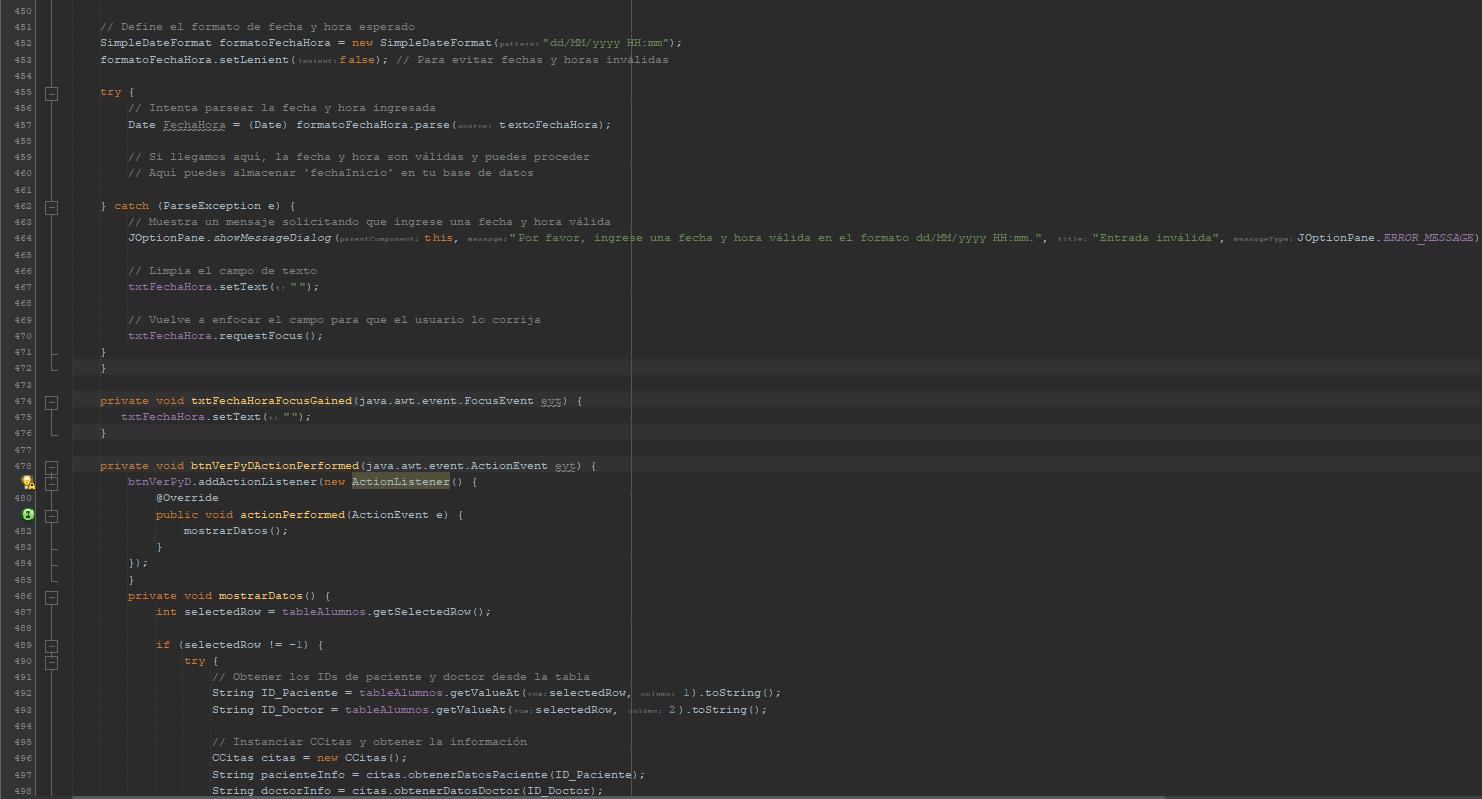
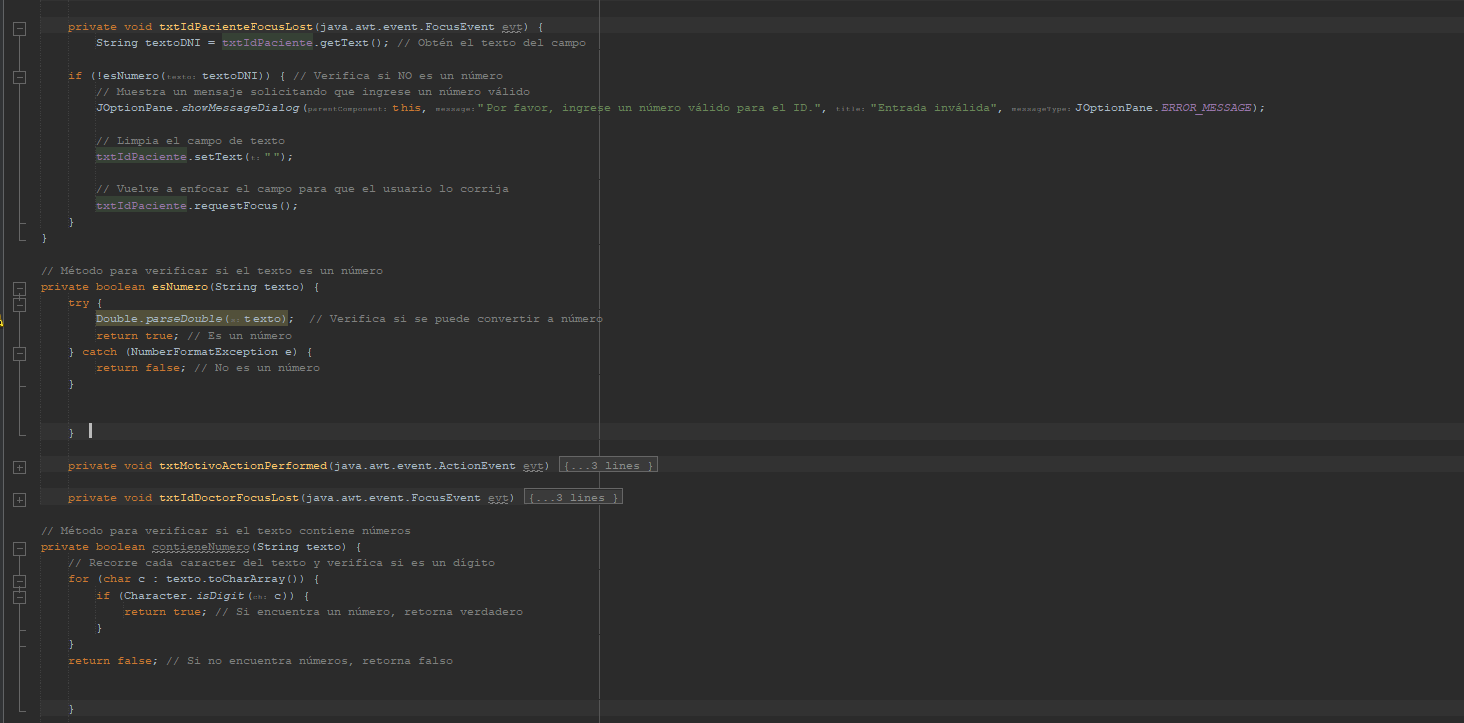
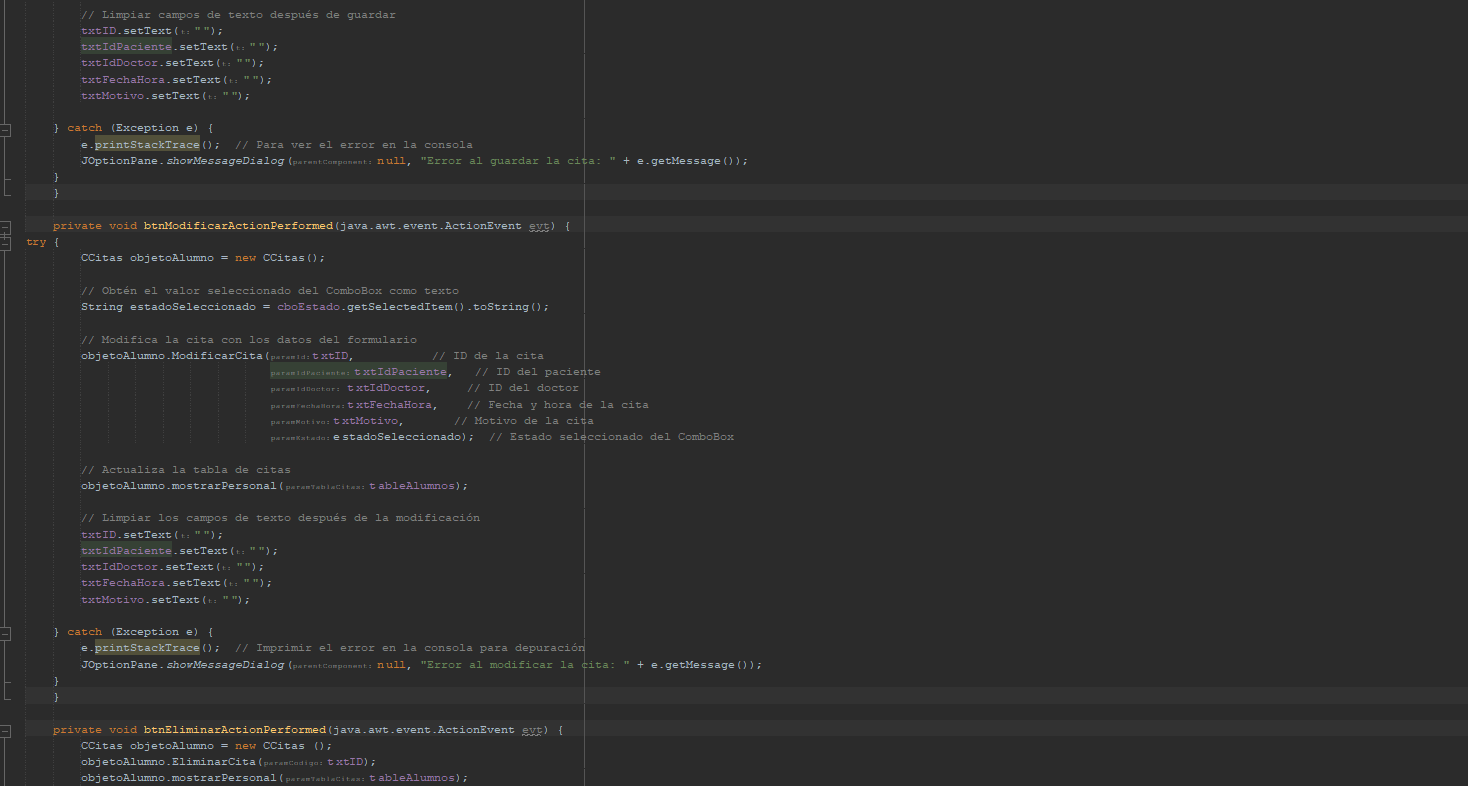
* **Mostrar Doctores:** Al iniciar el formulario, se muestran los doctores existentes en la tabla.
* **Selección de Doctores:** Cuando el usuario hace clic en una fila de la tabla, el método tableDoctoresMouseClicked se activa y llena los campos de texto con la información del doctor seleccionado.
* **Guardar Nuevo Doctor:** El botón "Guardar" llama al método InsertarDoctores de la clase CDoctores, que probablemente inserta un nuevo registro en la base de datos.
* **Modificar Doctor Existente:** Al hacer clic en "Modificar", se actualiza el registro existente basado en el ID proporcionado.
* **Eliminar Doctor:** El botón "Eliminar" elimina el doctor cuyo ID se encuentra en el campo de texto correspondiente.
* **Validaciones:** Hay validaciones para asegurarse de que los nombres y especialidades no contengan números, y que los teléfonos solo contengan dígitos.

### Manejo de Errores

* **JOptionPane:** Se utiliza para mostrar mensajes de error al usuario si los datos ingresados no son válidos (por ejemplo, si se ingresan números en un campo que no debe contenerlos).

**-**-------------------------------------------------------------------------------------------------------------------------------------



**Importaciones**:

* Se importan clases necesarias para la gestión de eventos, interacción con la base de datos, manejo de fechas y visualización de mensajes.

**Clase FormularioCRUDP**:

* Esta clase extiende JPanel, lo que significa que es un componente de interfaz gráfica de usuario (GUI) en Java.

**Constructor**:

* Al instanciar FormularioCRUDP, se inicializan los componentes de la interfaz y se carga la lista de citas a través de un objeto de la clase CCitas.

**Método initComponents**:

* Aquí se definen los componentes de la interfaz, incluyendo tablas, campos de texto, botones, y su disposición en el panel.

**Interacción con la Tabla**:

* tableAlumnos: Muestra las citas. Al hacer clic en una fila, se cargan los datos de la cita en los campos de texto correspondientes.
* btnVerPyD: Muestra información del paciente y doctor asociados a la cita seleccionada.

**Operaciones CRUD**:

* btnGuardar: Inserta una nueva cita en la base de datos utilizando los valores de los campos de texto.
* btnModificar: Actualiza la cita seleccionada en la base de datos.
* btnEliminar: Elimina la cita seleccionada.

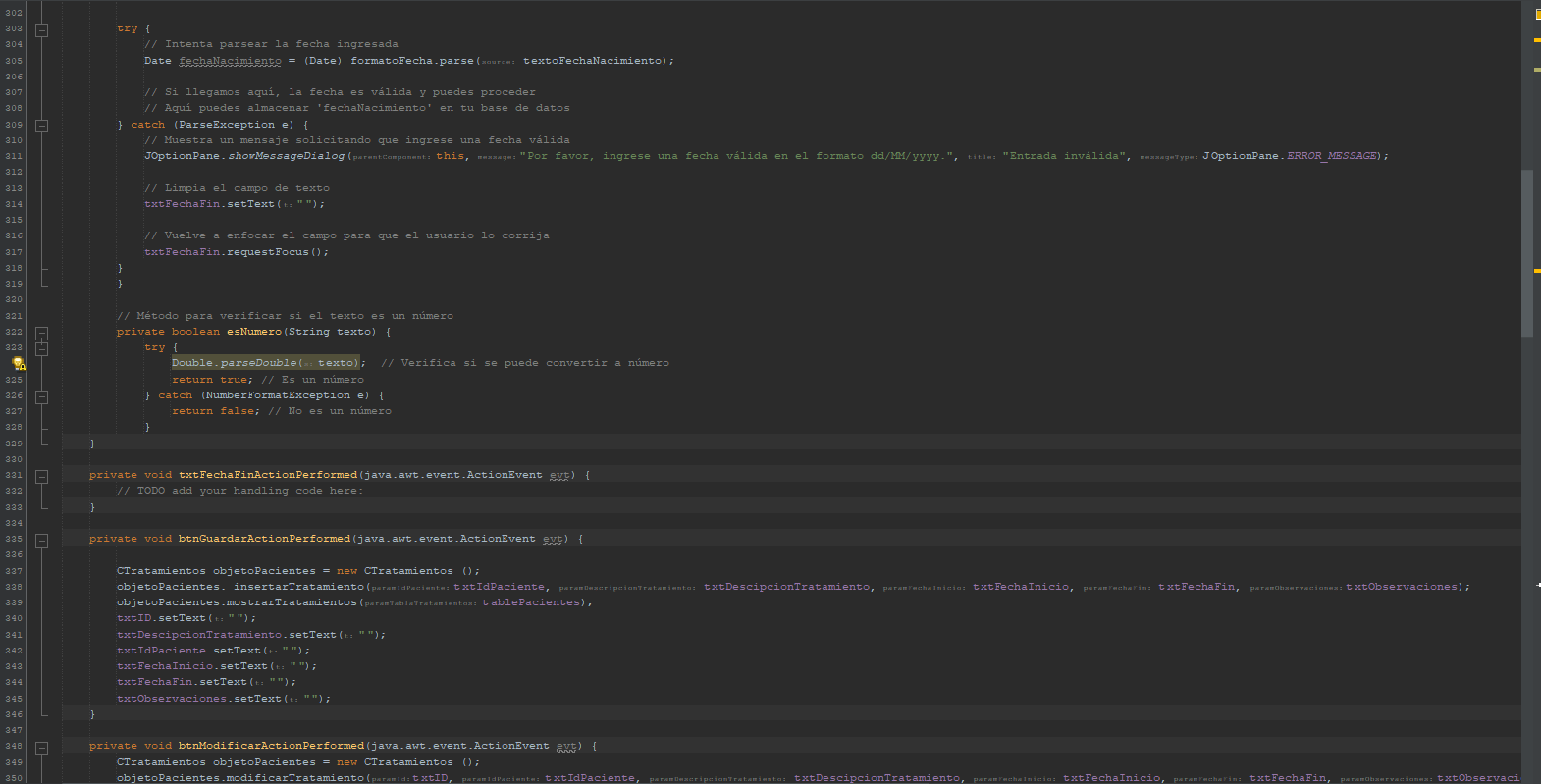
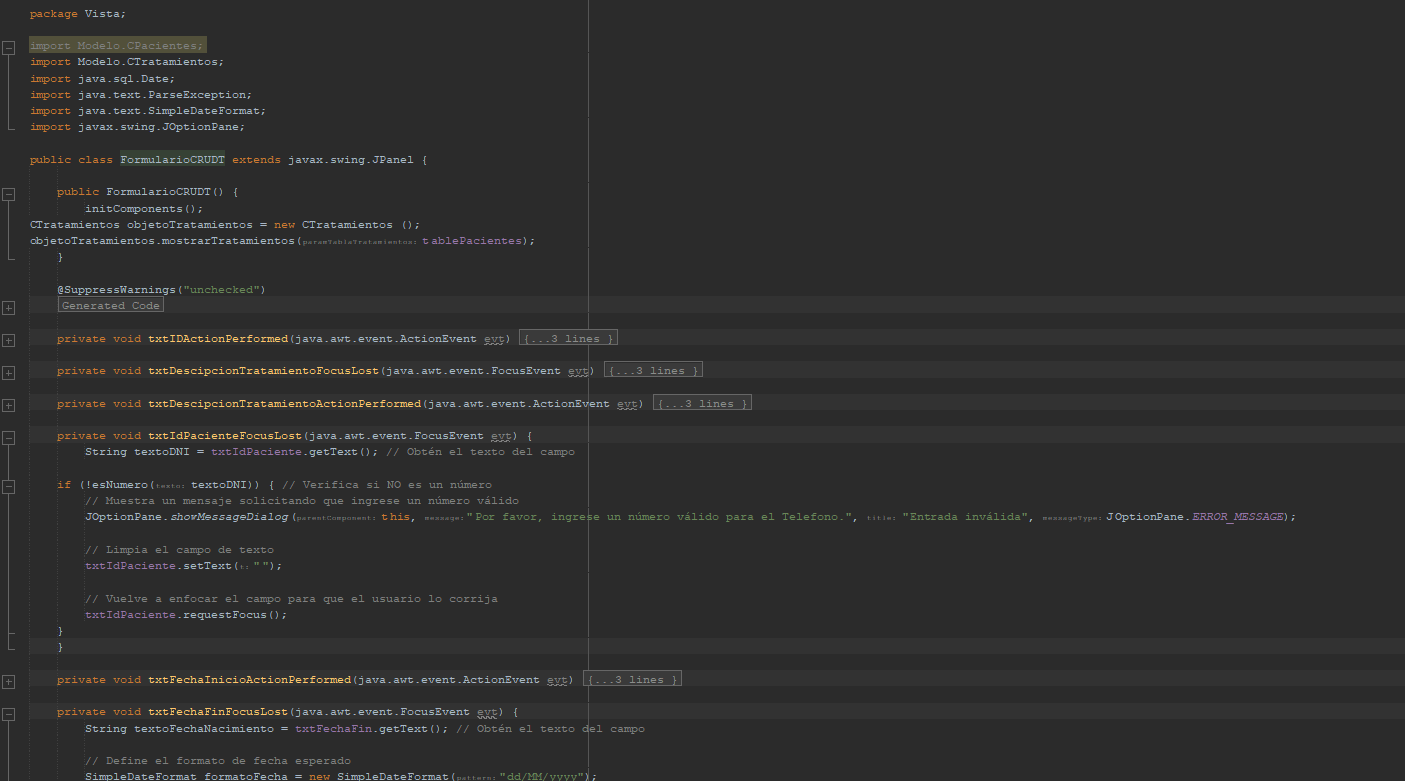
**Validaciones**:

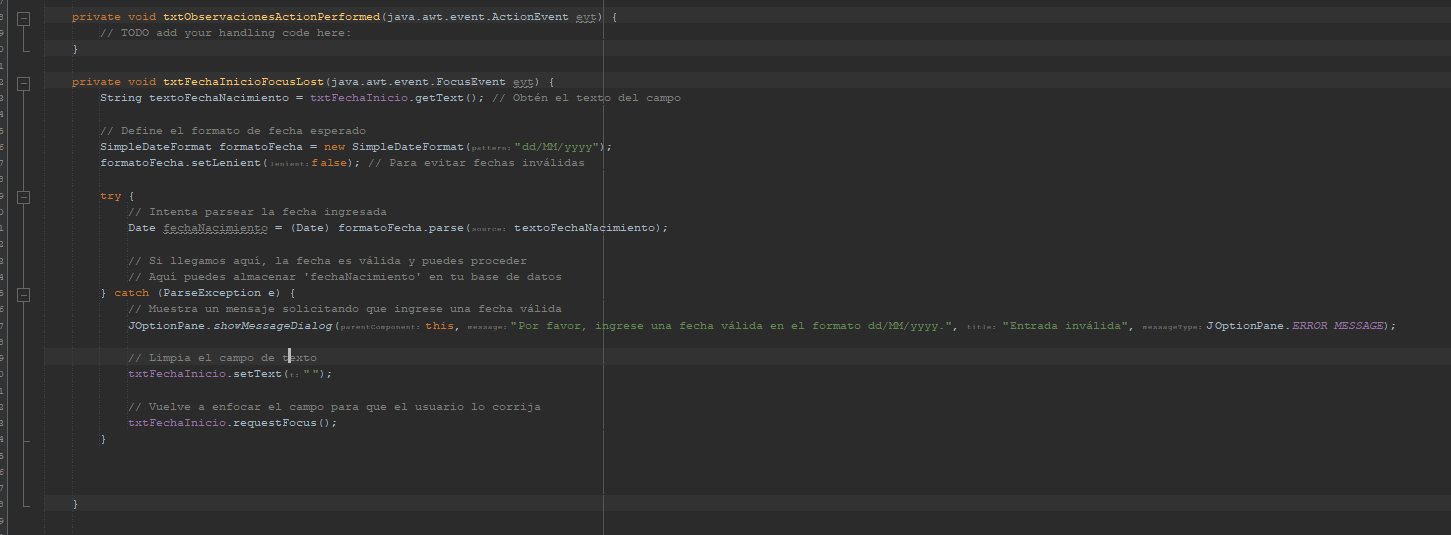
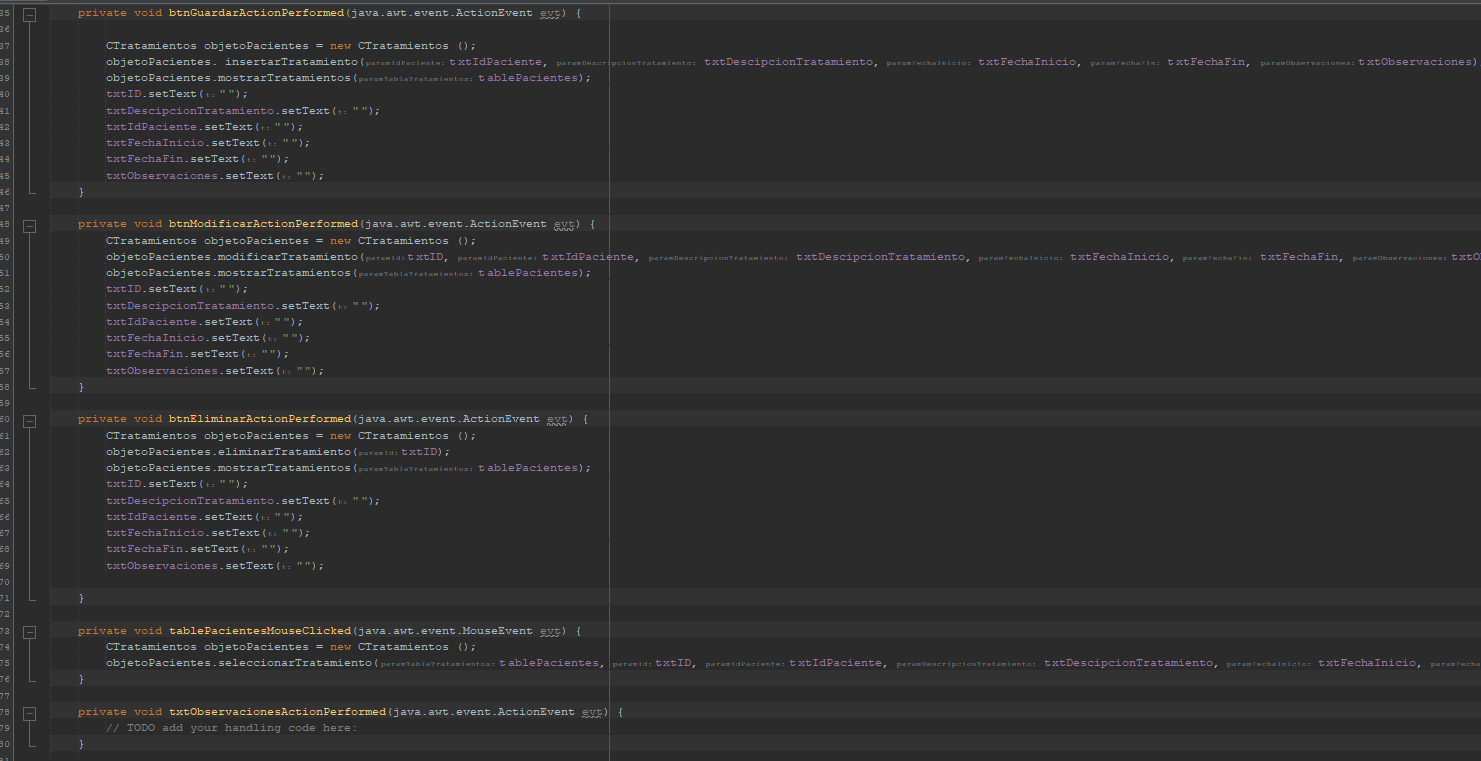
* Al perder el foco en los campos de ID de paciente y fecha/hora, se validan las entradas para asegurar que son números y que el formato de fecha/hora es correcto.

**Métodos Adicionales**:

* mostrarDatos: Se encarga de mostrar la información del paciente y doctor cuando se selecciona una cita en la tabla.

--------------------------------------------------------------------------------------------------------------------------------------





### Descripción del FormularioCRUDT

El FormularioCRUDT es un componente de interfaz gráfica de usuario (GUI) en Java, diseñado para gestionar tratamientos médicos. Utiliza bibliotecas de Swing para construir la interfaz y maneja operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para los tratamientos de los pacientes.

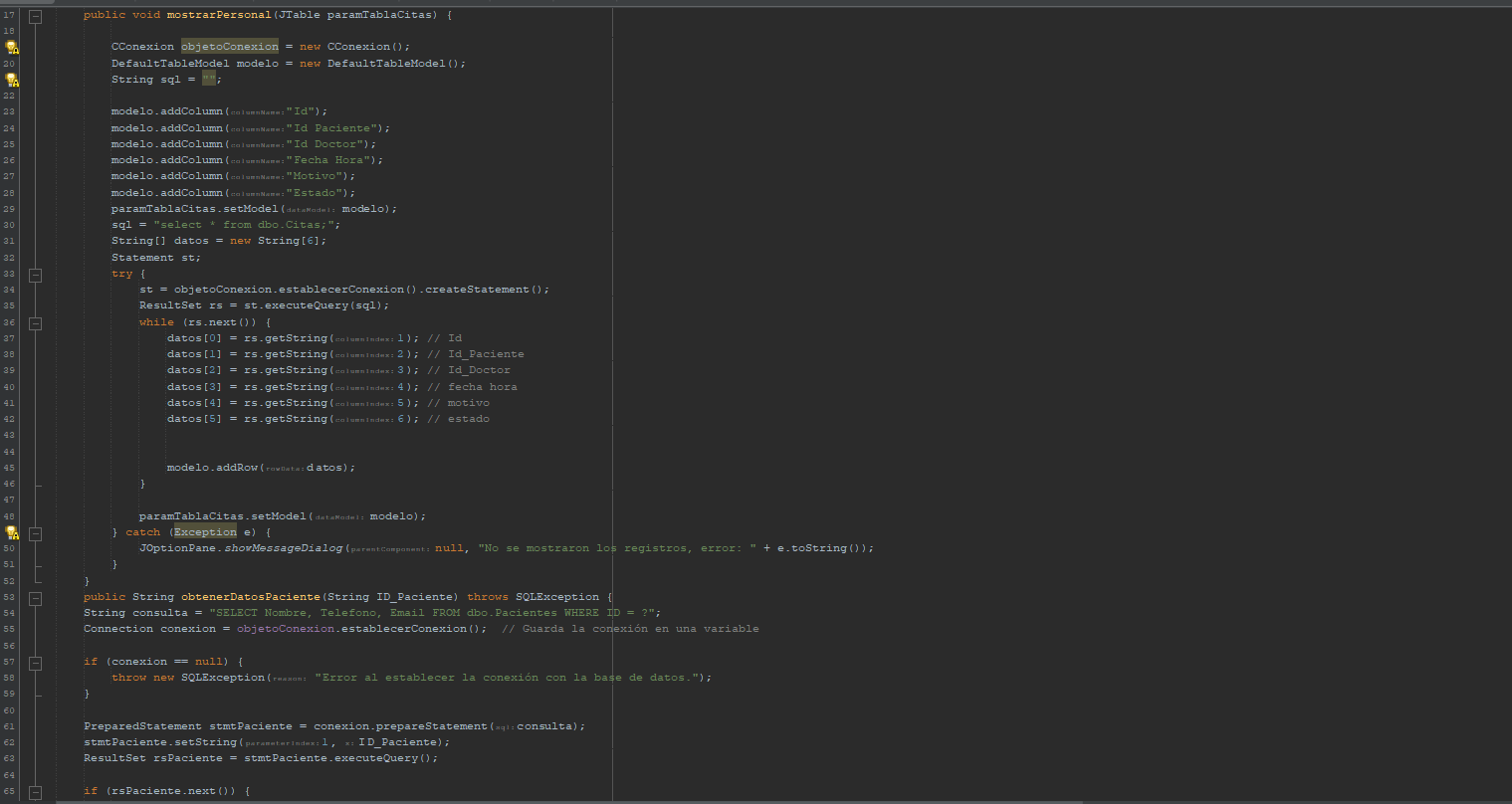
#### Componentes Principales

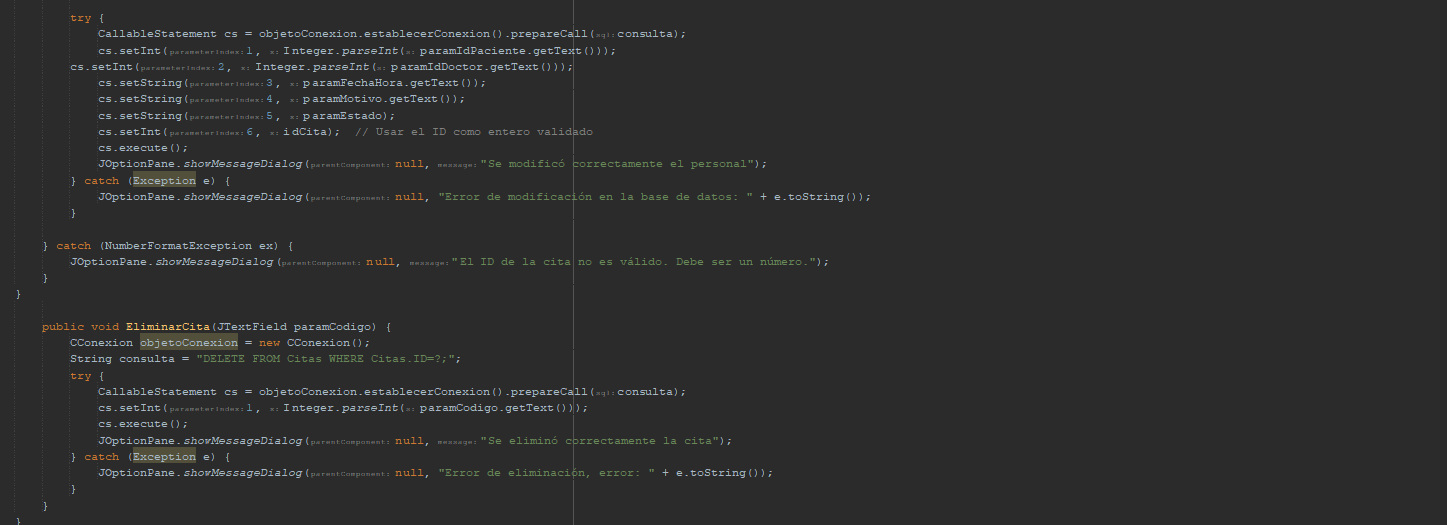
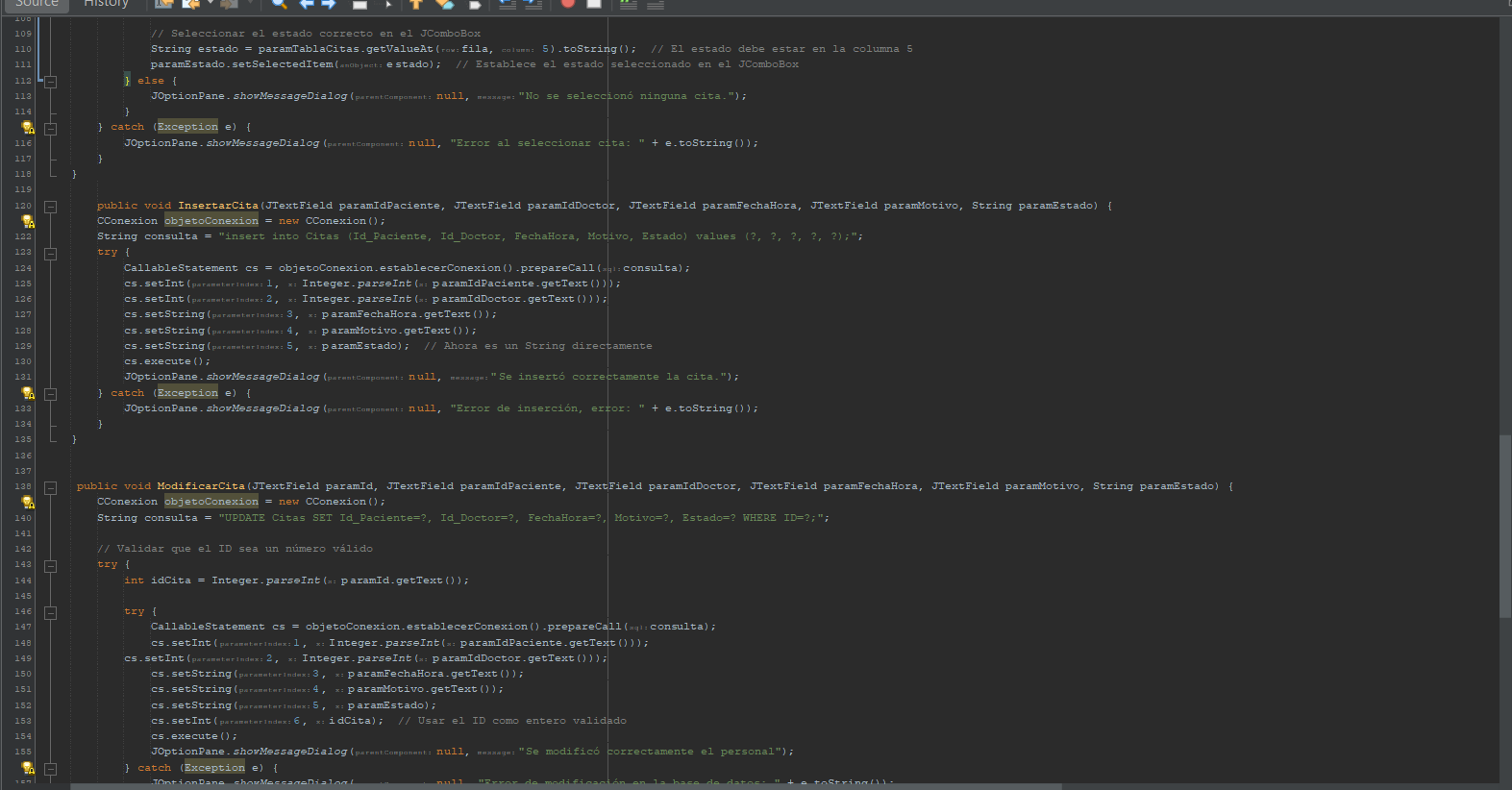
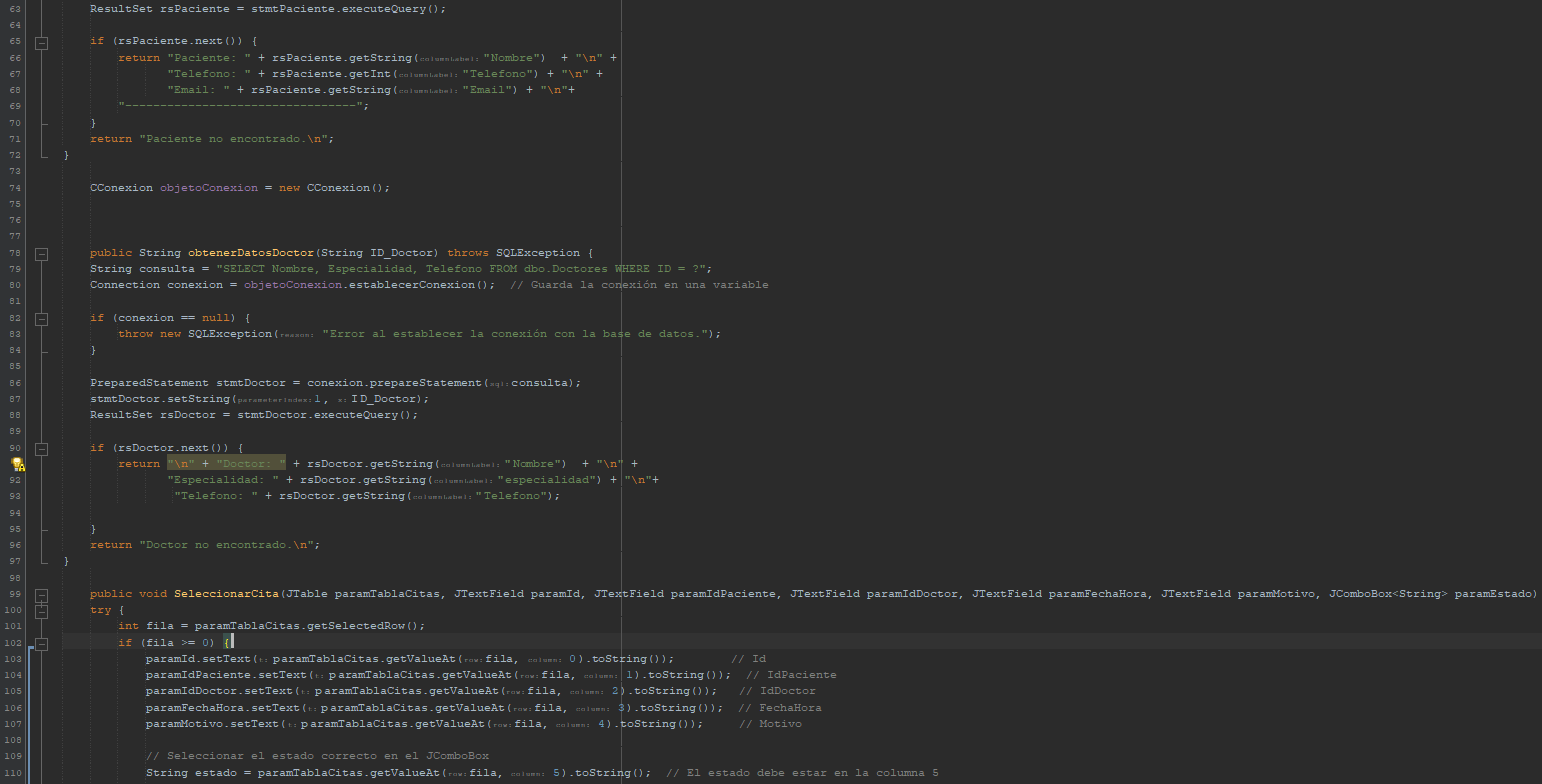
1. **Paneles y Componentes**:
   * Se divide en dos paneles: uno para los campos de entrada (jPanel1) y otro para mostrar la tabla de tratamientos (jPanel2).
   * Incluye etiquetas (jLabel) para cada campo de entrada, que son:
     + ID del tratamiento
     + Descripción del tratamiento
     + ID del paciente
     + Fecha de inicio
     + Fecha de fin
     + Observaciones
2. **Campos de Texto**:
   * Campos de texto (JTextField) permiten la entrada de información como el ID del tratamiento, descripción, ID del paciente y fechas.
3. **Botones**:
   * Tres botones (JButton) permiten realizar las operaciones CRUD: "Guardar", "Modificar" y "Eliminar".
   * Cada botón tiene un evento asociado que define la acción a realizar cuando se hace clic en él.
4. **Tabla**:
   * Una tabla (JTable) para mostrar los tratamientos existentes, permitiendo al usuario seleccionar un tratamiento y cargar sus detalles en los campos de entrada.

#### Funcionalidad

* **Constructor**:
  + Al instanciar el formulario, se llama a initComponents(), que inicializa los componentes de la interfaz.
  + Crea un objeto de la clase CTratamientos, que probablemente gestiona la lógica de negocio y la interacción con la base de datos, y muestra los tratamientos en la tabla.
* **Eventos de los Campos**:
  + Los campos de texto tienen validaciones específicas. Por ejemplo, al perder el foco, se valida si el ID del paciente es un número y si las fechas están en el formato correcto.
  + Se utilizan cuadros de diálogo (JOptionPane) para mostrar mensajes de error al usuario si la entrada es inválida.
* **Acciones de los Botones**:
  + btnGuardar: Inserta un nuevo tratamiento utilizando los datos ingresados y actualiza la tabla.
  + btnModificar: Modifica un tratamiento existente con los nuevos datos.
  + btnEliminar: Elimina un tratamiento seleccionado.
* **Seleccionar Tratamiento**:
  + Al hacer clic en una fila de la tabla, los campos de entrada se llenan automáticamente con la información del tratamiento seleccionado.

--------------------------------------------------------------------------------------------------------------------------------------





### Componentes Principales

1. **Conexión a la Base de Datos**:
   * La clase utiliza CConexion para establecer conexiones a la base de datos, asegurando que todas las operaciones se realicen con una conexión válida.
2. **Métodos de la Clase**:
   * **mostrarPersonal(JTable paramTablaCitas)**:
     + Este método recupera todas las citas de la base de datos y las muestra en un JTable.
     + Se utiliza un Statement para ejecutar una consulta SQL que selecciona todos los registros de la tabla Citas.
     + Los resultados se recorren y se añaden a un modelo de tabla (DefaultTableModel).
   * **obtenerDatosPaciente(String ID\_Paciente)**:
     + Este método busca la información del paciente en la base de datos a partir del ID proporcionado.
     + Se utiliza un PreparedStatement para evitar inyecciones SQL y mejorar la seguridad.
     + Devuelve una cadena con el nombre, teléfono y correo electrónico del paciente, o un mensaje indicando que no se encontró.
   * **obtenerDatosDoctor(String ID\_Doctor)**:
     + Similar al método anterior, pero para obtener la información del doctor.
     + Retorna el nombre, especialidad y teléfono del doctor correspondiente al ID proporcionado.
   * **SeleccionarCita(JTable paramTablaCitas, JTextField... )**:
     + Este método permite seleccionar una cita en la tabla y carga sus detalles en los campos de texto correspondientes.
     + Si no se selecciona ninguna fila, muestra un mensaje de error.
   * **InsertarCita(JTextField... )**:
     + Inserta una nueva cita en la base de datos usando los datos de los campos de texto.
     + Utiliza un CallableStatement para ejecutar la inserción.
     + Muestra un mensaje de éxito o error según el resultado de la operación.
   * **ModificarCita(JTextField... )**:
     + Modifica una cita existente en la base de datos.
     + Verifica que el ID de la cita sea un número válido antes de proceder con la actualización.
     + Al igual que la inserción, utiliza un CallableStatement.
   * **EliminarCita(JTextField paramCodigo)**:
     + Elimina una cita de la base de datos utilizando el ID proporcionado en el campo de texto.
     + Maneja excepciones y muestra mensajes de éxito o error.

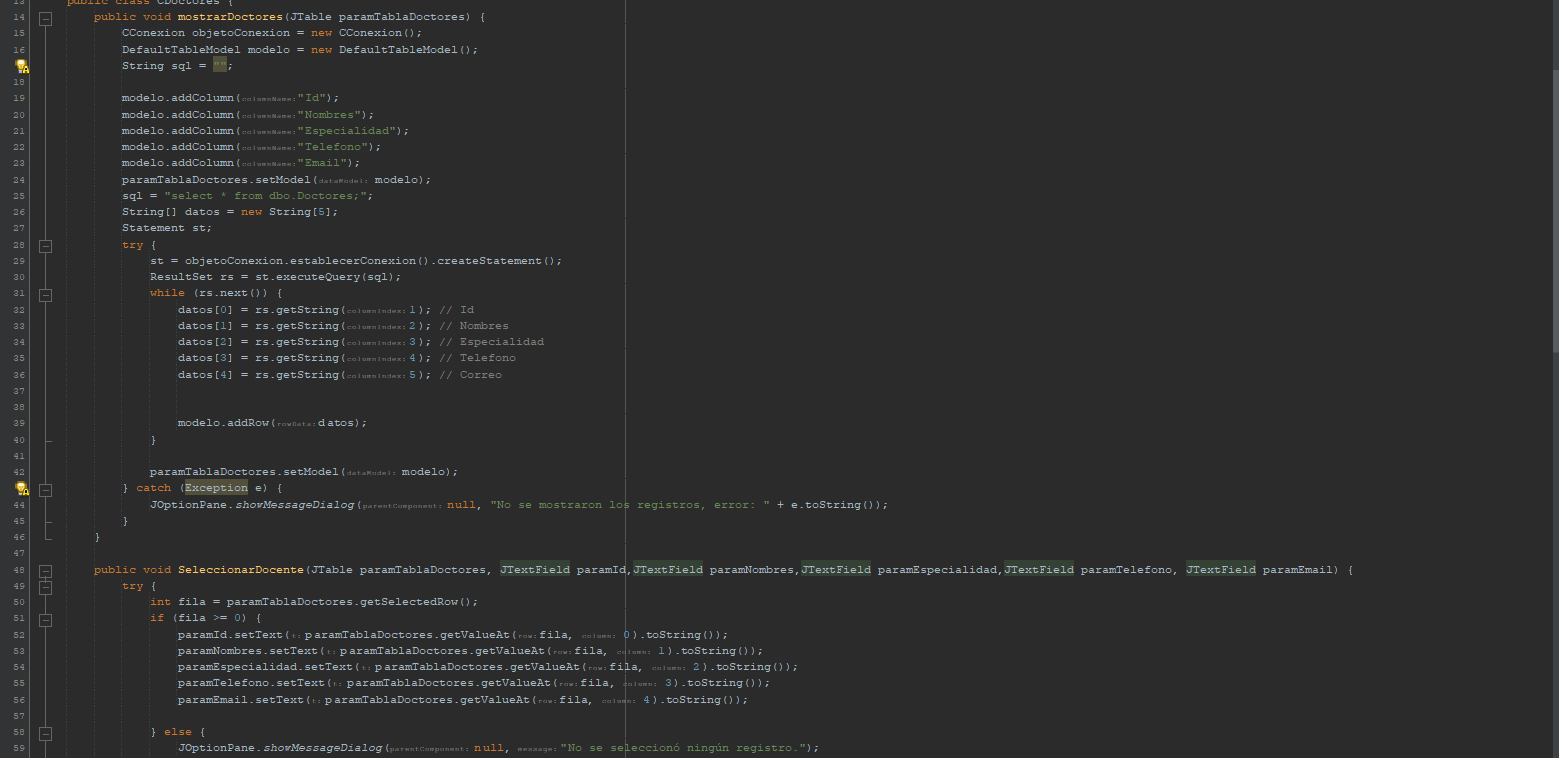
### Manejo de Errores

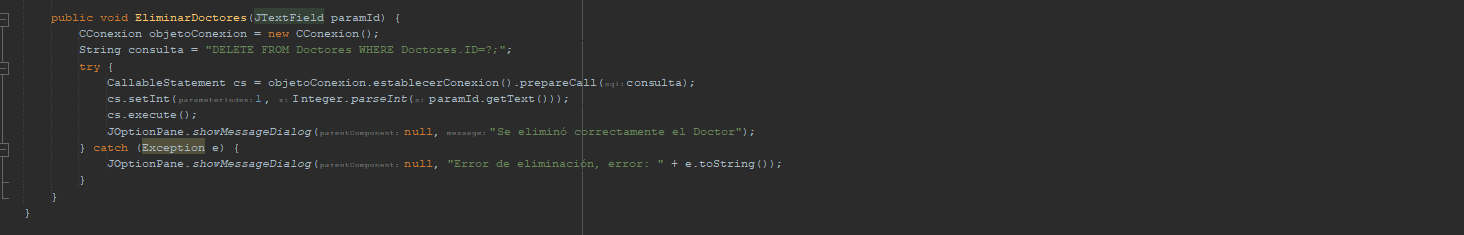
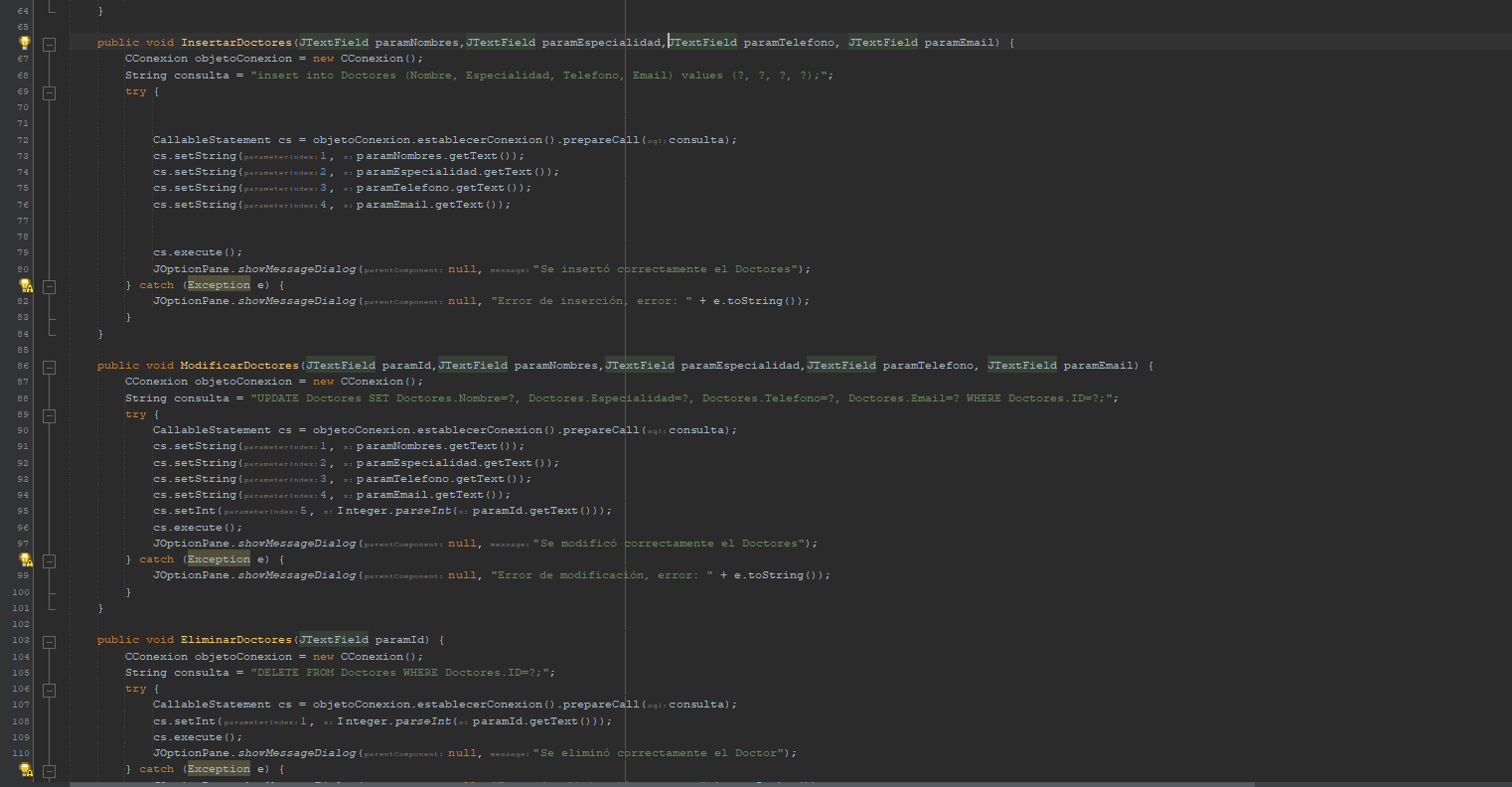
La clase implementa manejo de excepciones utilizando try-catch para capturar y mostrar errores durante las operaciones de base de datos, como conexiones fallidas o problemas al ejecutar consultas.

### Consideraciones de Seguridad

* **Uso de PreparedStatement**:
  + Se utilizan PreparedStatement en los métodos que manejan consultas con parámetros, lo que ayuda a prevenir inyecciones SQL.

---------------------------------------------------------------------------------------------------------------------------





### Descripción General

La clase CDoctores proporciona métodos para mostrar, seleccionar, insertar, modificar y eliminar registros de doctores en una base de datos. Utiliza JDBC para las operaciones de acceso a datos y maneja las interacciones con componentes de la interfaz gráfica, como JTable y JTextField.

### Componentes Principales

1. **Conexión a la Base de Datos**:
   * Al igual que en la clase CCitas, se utiliza CConexion para establecer conexiones a la base de datos. Esto garantiza que todas las operaciones se realicen en un contexto de conexión adecuado.
2. **Métodos de la Clase**:
   * **mostrarDoctores(JTable paramTablaDoctores)**:
     + Este método recupera la información de todos los doctores de la base de datos y la muestra en un JTable.
     + Se crea un DefaultTableModel para definir las columnas (ID, Nombres, Especialidad, Teléfono, Email).
     + Se ejecuta una consulta SQL que selecciona todos los registros de la tabla Doctores, y los resultados se añaden al modelo de tabla.
   * **SeleccionarDocente(JTable paramTablaDoctores, JTextField... )**:
     + Este método permite seleccionar un registro de doctor en la tabla y cargar sus datos en los campos de texto correspondientes.
     + Se verifica si se ha seleccionado alguna fila y, de ser así, se extraen los datos y se establecen en los campos de texto. Si no se selecciona nada, se muestra un mensaje de error.
   * **InsertarDoctores(JTextField... )**:
     + Este método inserta un nuevo registro de doctor en la base de datos utilizando los datos de los campos de texto.
     + Se utiliza un CallableStatement para preparar la consulta de inserción y se asignan los valores de los campos de texto a los parámetros de la consulta.
     + Muestra un mensaje de éxito o error según el resultado de la operación.
   * **ModificarDoctores(JTextField... )**:
     + Este método actualiza un registro existente de doctor en la base de datos.
     + Similar al método de inserción, pero aquí se utiliza la consulta UPDATE. Se asegura que el ID del doctor a modificar sea un número válido antes de ejecutar la actualización.
     + También se muestra un mensaje de éxito o error.
   * **EliminarDoctores(JTextField paramId)**:
     + Elimina un registro de doctor de la base de datos utilizando el ID proporcionado en el campo de texto.
     + Utiliza un CallableStatement para preparar la consulta de eliminación y maneja posibles excepciones, mostrando mensajes adecuados al usuario.

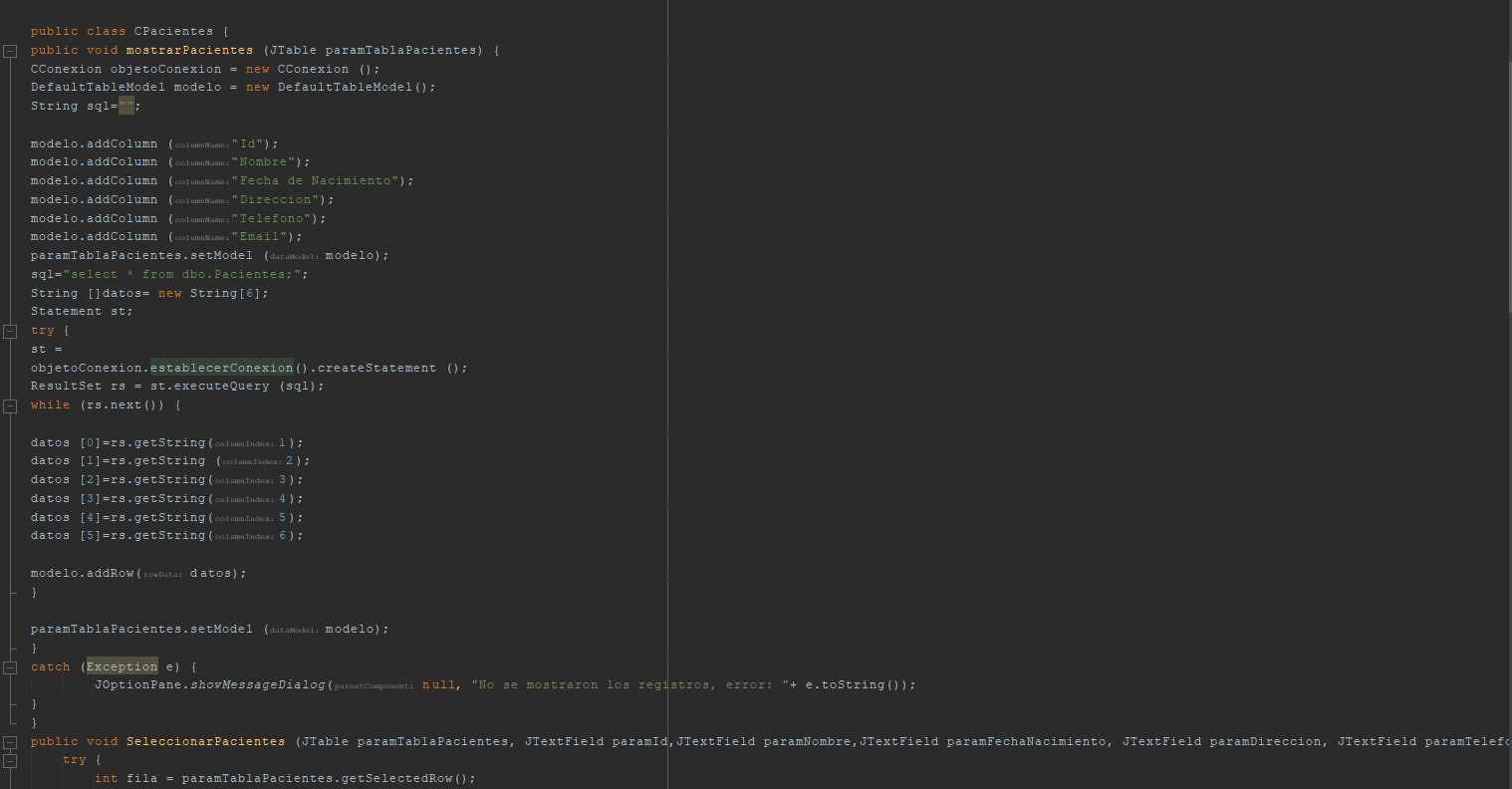
### Manejo de Errores

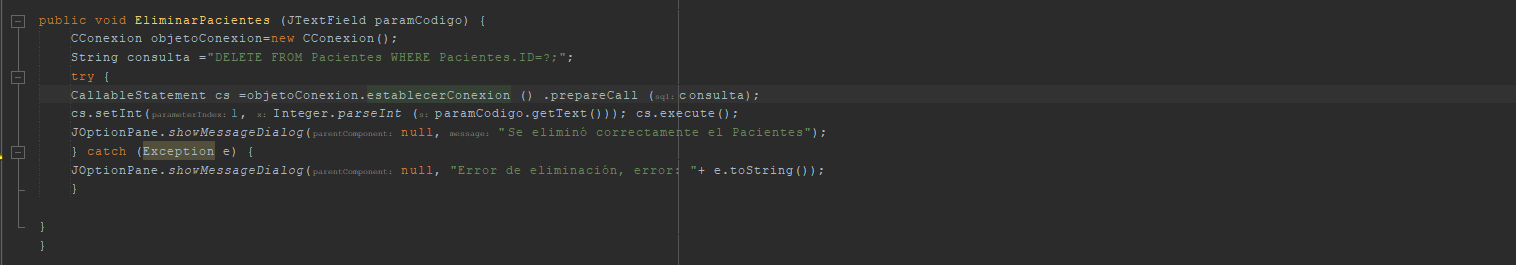
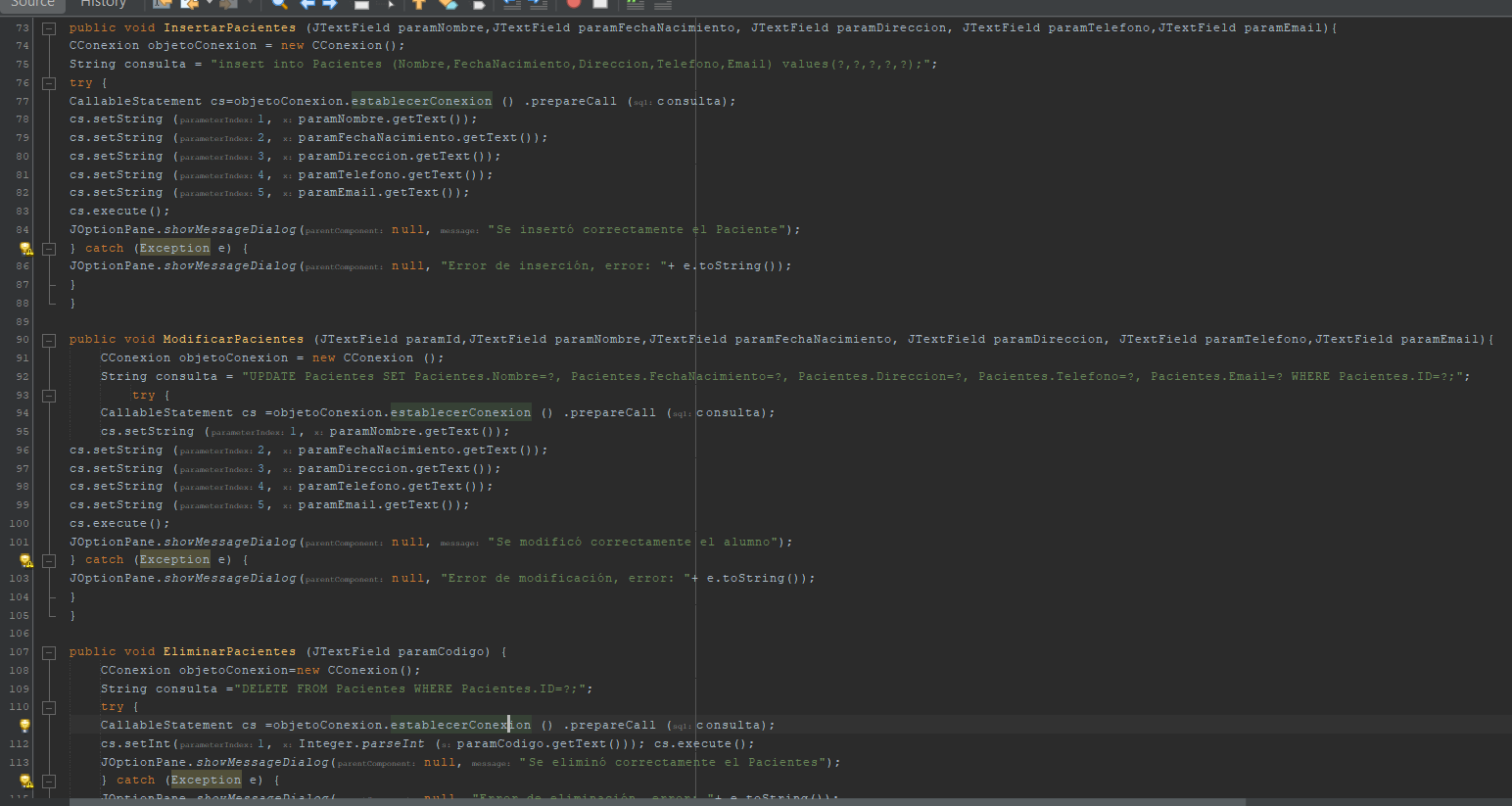
La clase implementa manejo de excepciones con bloques try-catch, que permiten capturar errores durante las operaciones de acceso a la base de datos, y mostrar mensajes de error informativos en caso de problemas.

### Consideraciones de Seguridad

* **Uso de PreparedStatement**:
  + Se emplean PreparedStatement para las consultas que contienen parámetros, lo que ayuda a prevenir inyecciones SQL y mejora la seguridad de la aplicación.

--------------------------------------------------------------------------------------------------------------------------------------





### Descripción General

La clase CPacientes proporciona métodos para mostrar, seleccionar, insertar, modificar y eliminar registros de pacientes en una base de datos. Utiliza JDBC para las operaciones de acceso a datos y maneja interacciones con componentes de la interfaz gráfica, como JTable y JTextField.

### Componentes Principales

1. **Conexión a la Base de Datos**:
   * Se utiliza CConexion para establecer conexiones a la base de datos, asegurando que las operaciones se realicen en un contexto de conexión adecuado.
2. **Métodos de la Clase**:
   * **mostrarPacientes(JTable paramTablaPacientes)**:
     + Este método recupera la información de todos los pacientes de la base de datos y la muestra en un JTable.
     + Se crea un DefaultTableModel para definir las columnas (Id, Nombre, Fecha de Nacimiento, Dirección, Teléfono, Email).
     + Se ejecuta una consulta SQL que selecciona todos los registros de la tabla Pacientes, y los resultados se añaden al modelo de tabla.
   * **SeleccionarPacientes(JTable paramTablaPacientes, JTextField... )**:
     + Este método permite seleccionar un registro de paciente en la tabla y cargar sus datos en los campos de texto correspondientes.
     + Se verifica si se ha seleccionado alguna fila. Si es así, se extraen los datos y se establecen en los campos de texto. Si no se selecciona nada, se omite el mensaje de error.
   * **InsertarPacientes(JTextField... )**:
     + Este método inserta un nuevo registro de paciente en la base de datos utilizando los datos de los campos de texto.
     + Se utiliza un CallableStatement para preparar la consulta de inserción y se asignan los valores de los campos de texto a los parámetros de la consulta.
     + Muestra un mensaje de éxito o error según el resultado de la operación.
   * **ModificarPacientes(JTextField... )**:
     + Este método actualiza un registro existente de paciente en la base de datos.
     + Se utiliza la consulta UPDATE y se aseguran que los datos del paciente a modificar sean correctos.
     + También muestra un mensaje de éxito o error.
   * **EliminarPacientes(JTextField paramCodigo)**:
     + Este método elimina un registro de paciente de la base de datos utilizando el ID proporcionado en el campo de texto.
     + Utiliza un CallableStatement para preparar la consulta de eliminación y maneja posibles excepciones, mostrando mensajes adecuados al usuario.

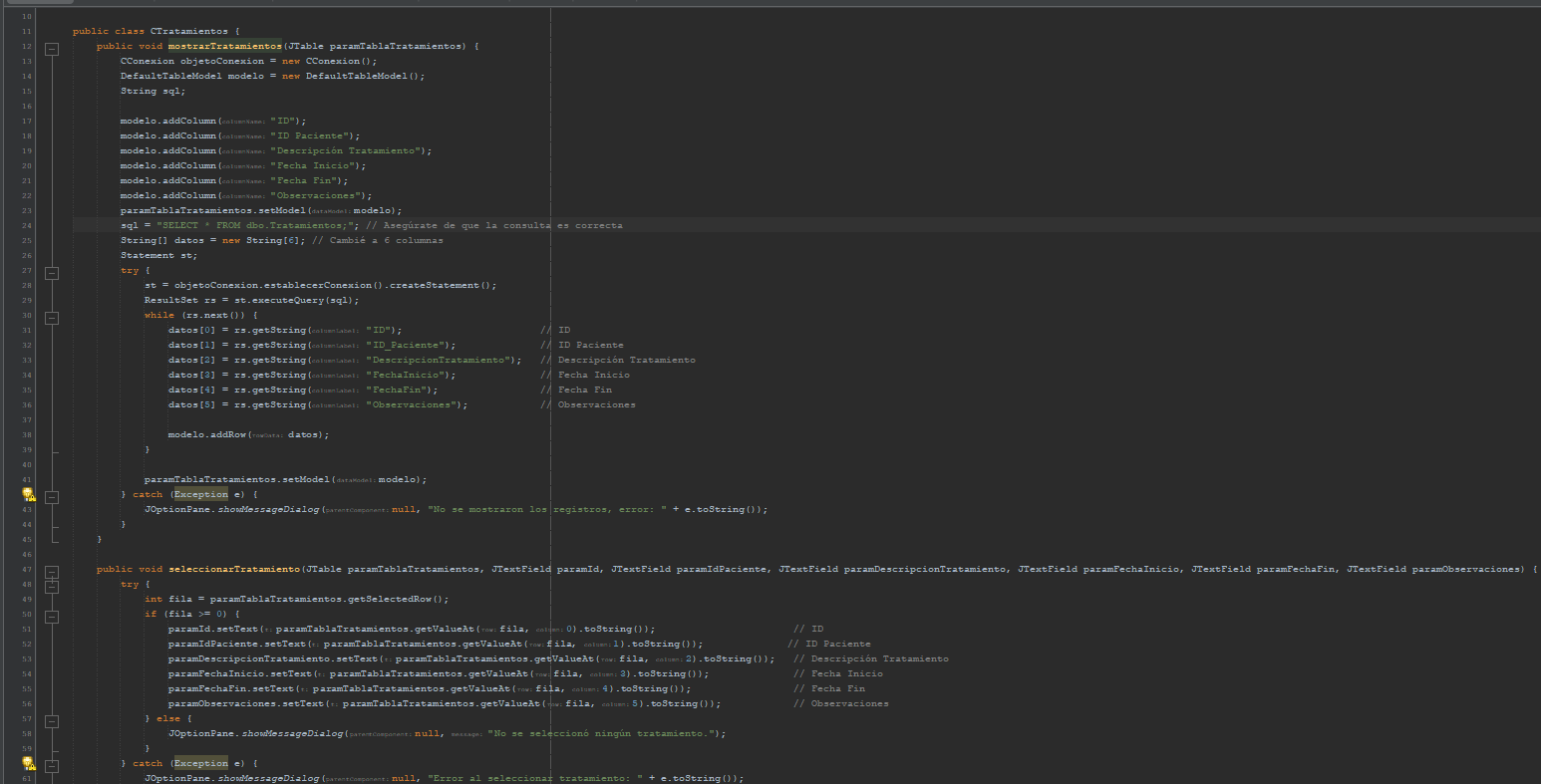
### Manejo de Errores

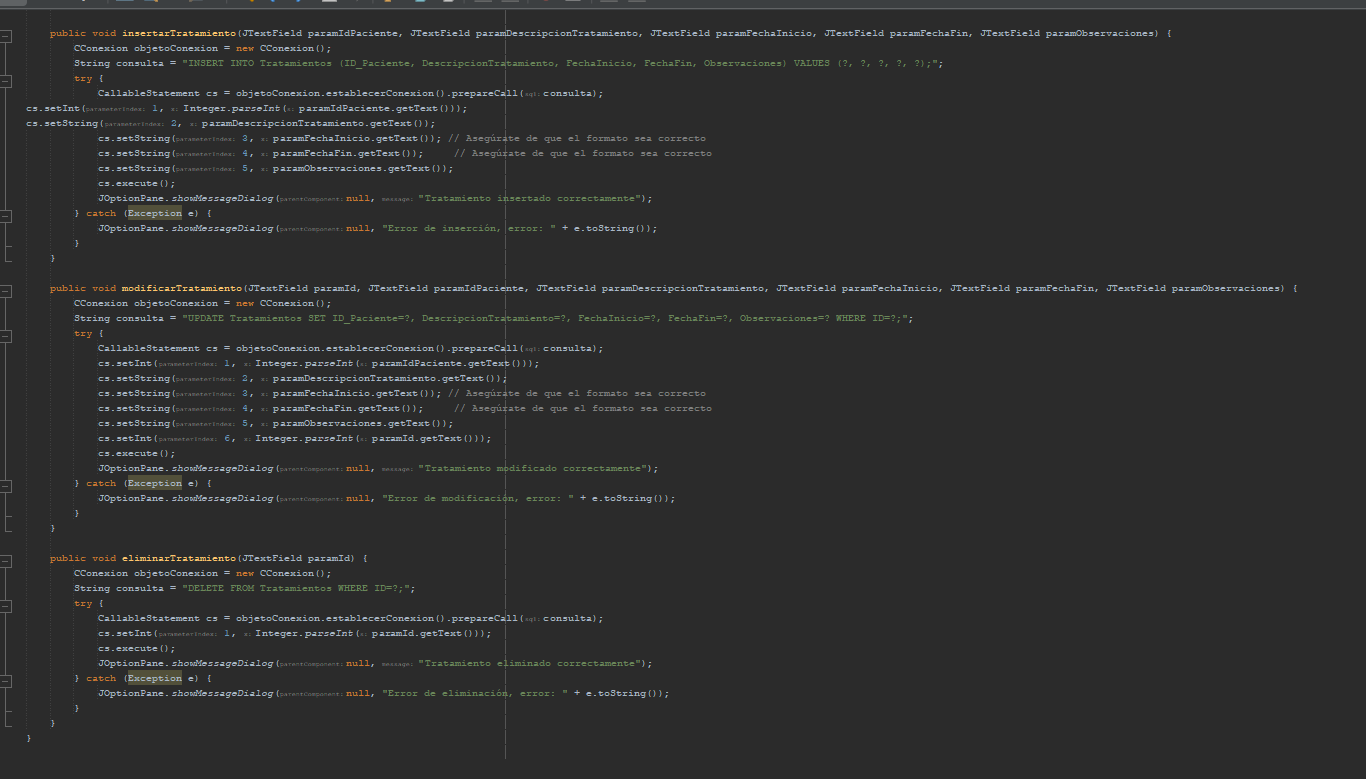
La clase implementa manejo de excepciones con bloques try-catch, que permiten capturar errores durante las operaciones de acceso a la base de datos y mostrar mensajes de error informativos.

### Consideraciones de Seguridad

* **Uso de PreparedStatement**:
  + Al igual que en la clase anterior, se emplean PreparedStatement para las consultas que contienen parámetros, lo que ayuda a prevenir inyecciones SQL y mejora la seguridad de la aplicación.

--------------------------------------------------------------------------------------------------------------------------------------





### Descripción General

La clase CTratamientos está diseñada para interactuar con una base de datos que almacena información sobre tratamientos médicos. Utiliza JDBC para realizar operaciones sobre la base de datos y muestra la información en un JTable.

### Componentes Principales

1. **Conexión a la Base de Datos**:
   * Se utiliza la clase CConexion para establecer conexiones a la base de datos, permitiendo que los métodos realicen operaciones de acceso a datos.
2. **Métodos de la Clase**:
   * **mostrarTratamientos(JTable paramTablaTratamientos)**:
     + Este método se encarga de recuperar todos los tratamientos de la base de datos y mostrarlos en un JTable.
     + Se crea un DefaultTableModel que define las columnas (ID, ID Paciente, Descripción Tratamiento, Fecha Inicio, Fecha Fin, Observaciones).
     + Se ejecuta una consulta SQL que selecciona todos los registros de la tabla Tratamientos, y los resultados se añaden al modelo de tabla.
   * **seleccionarTratamiento(JTable paramTablaTratamientos, JTextField... )**:
     + Este método permite seleccionar un tratamiento de la tabla y cargar su información en los campos de texto correspondientes.
     + Verifica si se ha seleccionado una fila en la tabla. Si es así, extrae los datos de esa fila y los establece en los campos de texto. Si no se selecciona nada, muestra un mensaje informando al usuario.
   * **insertarTratamiento(JTextField... )**:
     + Este método inserta un nuevo tratamiento en la base de datos utilizando los datos proporcionados en los campos de texto.
     + Utiliza un CallableStatement para preparar la consulta de inserción, asignando los valores de los campos de texto a los parámetros de la consulta.
     + Se muestra un mensaje de éxito o error según el resultado de la operación.
   * **modificarTratamiento(JTextField... )**:
     + Este método actualiza un tratamiento existente en la base de datos.
     + Utiliza la consulta UPDATE y asegura que se utilizan los datos correctos del tratamiento a modificar.
     + También muestra un mensaje informando si la operación fue exitosa o si ocurrió un error.
   * **eliminarTratamiento(JTextField paramId)**:
     + Este método elimina un tratamiento de la base de datos utilizando el ID proporcionado en el campo de texto.
     + Utiliza un CallableStatement para preparar la consulta de eliminación y maneja excepciones, mostrando mensajes adecuados al usuario.

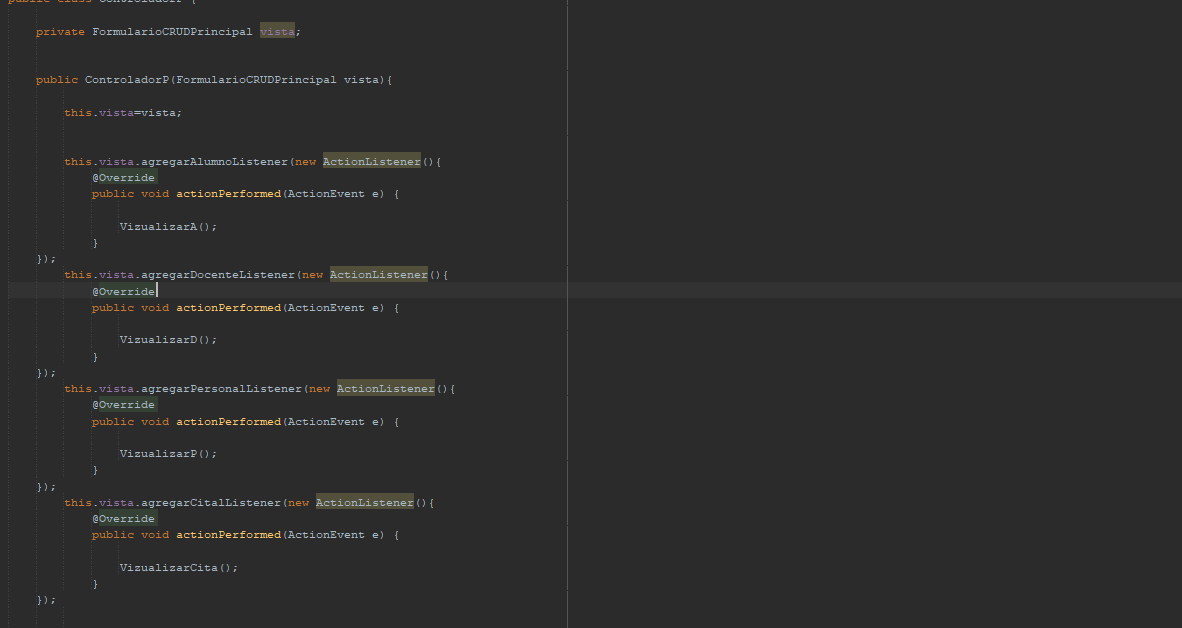
### Manejo de Errores

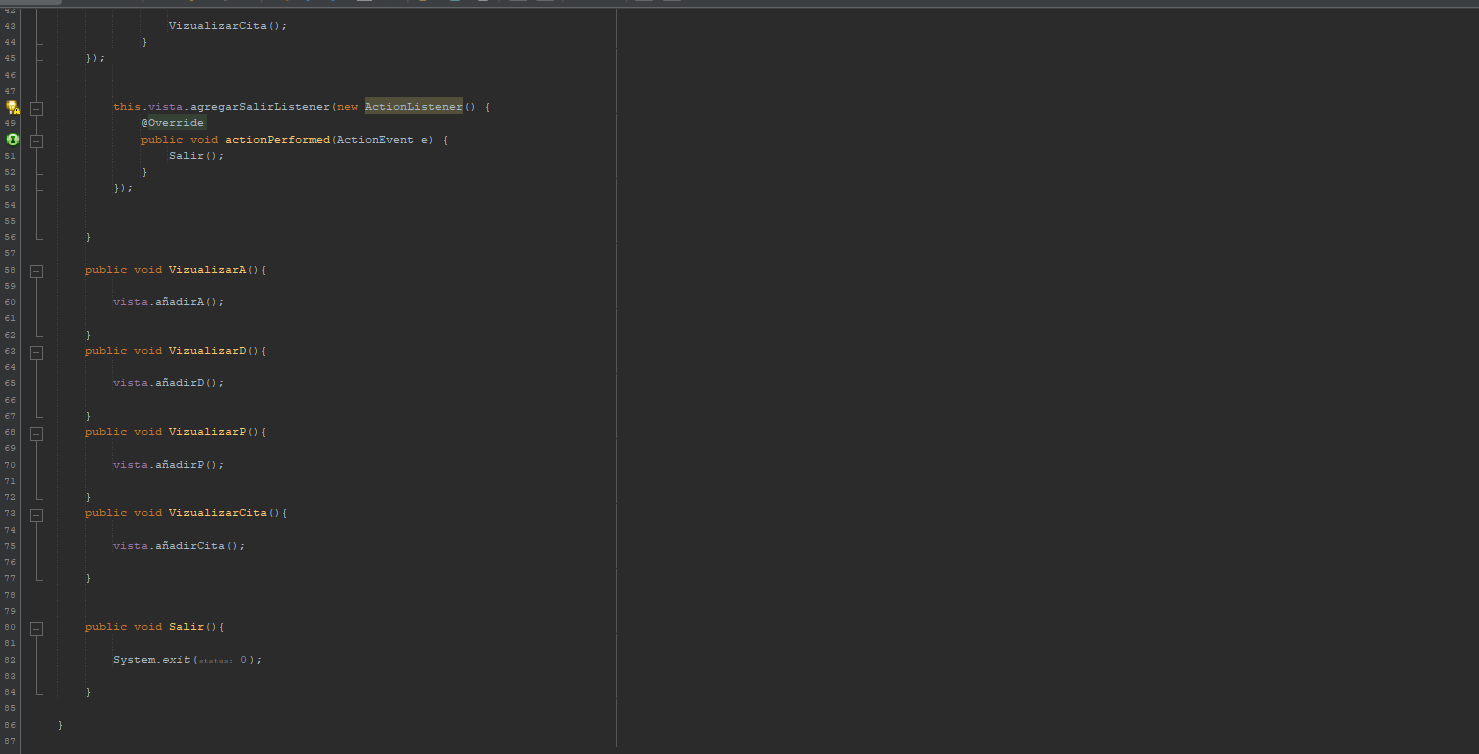
La clase implementa manejo de excepciones con bloques try-catch, lo que permite capturar errores que pueden ocurrir durante las operaciones de acceso a la base de datos y proporciona mensajes de error informativos.

### Consideraciones de Seguridad

* **Uso de PreparedStatement**:
  + Al igual que las otras clases, utiliza PreparedStatement para las consultas que contienen parámetros, lo que ayuda a prevenir inyecciones SQL y mejora la seguridad general de la aplicación.

--------------------------------------------------------------------------------------------------------------------------------------





### Descripción General

La clase ControladorP es responsable de gestionar las interacciones del usuario en la vista FormularioCRUDPrincipal. Escucha eventos de la interfaz y realiza las acciones correspondientes, como mostrar diferentes formularios o salir de la aplicación.

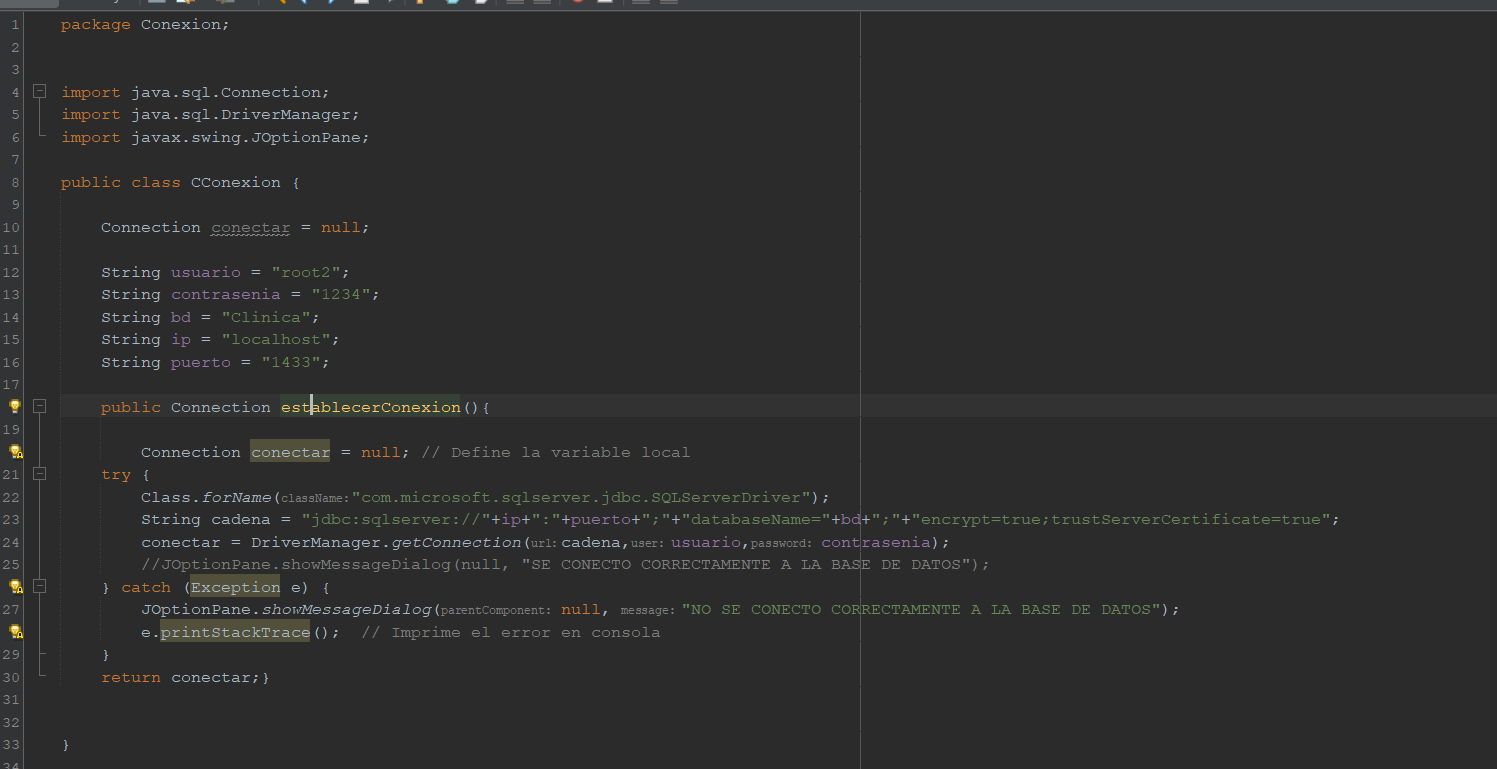
### Componentes Principales

1. **Atributos**:
   * private FormularioCRUDPrincipal vista;
     + Este atributo guarda la instancia de la vista FormularioCRUDPrincipal, permitiendo que el controlador interactúe con ella.
2. **Constructor**:
   * **public ControladorP(FormularioCRUDPrincipal vista)**:
     + El constructor recibe un objeto de la clase FormularioCRUDPrincipal y lo asigna al atributo vista.
     + Se registran varios ActionListener para manejar eventos de acción en la vista. Esto permite que el controlador responda a los eventos generados por los botones o componentes de la interfaz.
3. **Manejo de Eventos**:
   * Para cada acción del usuario (como hacer clic en botones), se registran ActionListener que llaman a métodos específicos del controlador:
     + **this.vista.agregarAlumnoListener(...)**:
       - Este listener se activa cuando el usuario quiere agregar un alumno. Llama al método VizualizarA().
     + **this.vista.agregarDocenteListener(...)**:
       - Este listener se activa para agregar un docente, llamando al método VizualizarD().
     + **this.vista.agregarPersonalListener(...)**:
       - Este listener se utiliza para agregar personal, llamando al método VizualizarP().
     + **this.vista.agregarCitalListener(...)**:
       - Este listener se activa para agregar citas, llamando al método VizualizarCita().
     + **this.vista.agregarSalirListener(...)**:
       - Este listener se encarga de cerrar la aplicación cuando se hace clic en el botón de salir, llamando al método Salir().
4. **Métodos para Mostrar Formularios**:
   * **public void VizualizarA()**:
     + Llama al método añadirA() de la vista para mostrar la interfaz de agregar un alumno.
   * **public void VizualizarD()**:
     + Llama al método añadirD() de la vista para mostrar la interfaz de agregar un docente.
   * **public void VizualizarP()**:
     + Llama al método añadirP() de la vista para mostrar la interfaz de agregar personal.
   * **public void VizualizarCita()**:
     + Llama al método añadirCita() de la vista para mostrar la interfaz de agregar una cita.
5. **Método para Salir**:
   * **public void Salir()**:
     + Cierra la aplicación llamando a System.exit(0), lo que finaliza el programa.

### Consideraciones

* **Desacoplamiento**:
  + Este diseño permite un desacoplamiento claro entre la lógica de negocio (que puede estar en otras clases, como los modelos) y la interfaz de usuario. El controlador maneja los eventos y llama a los métodos adecuados de la vista para realizar acciones.
* **Escalabilidad**:
  + Agregar nuevas funcionalidades o cambiar las existentes es más sencillo, ya que la lógica está separada de la vista. Solo se necesita modificar el controlador y, si es necesario, la vista correspondiente.

--------------------------------------------------------------------------------------------------------------------------------------



### Descripción General

La clase CConexion establece una conexión a una base de datos utilizando JDBC (Java Database Connectivity). Contiene la configuración necesaria para conectarse y un método para realizar la conexión.

### Componentes Principales

1. **Atributos**:
   * **Connection conectar**:
     + Este atributo es una referencia a la conexión de la base de datos.
   * **String usuario**:
     + Nombre de usuario para la conexión a la base de datos (en este caso, "root2").
   * **String contrasenia**:
     + Contraseña para la conexión a la base de datos (en este caso, "1234").
   * **String bd**:
     + Nombre de la base de datos a la que se quiere conectar ("Clinica").
   * **String ip**:
     + Dirección IP del servidor donde está alojada la base de datos (en este caso, "localhost").
   * **String puerto**:
     + Puerto utilizado para conectarse a la base de datos (en este caso, "1433").
2. **Método establecerConexion()**:
   * **public Connection establecerConexion()**:
     + Este método es responsable de crear y devolver una conexión a la base de datos.
     + **Paso a paso**:
       - Se define una variable local Connection conectar para almacenar la conexión.
       - **Carga del Driver**:
         * Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");:

Carga el driver de SQL Server. Esto es necesario para que JDBC sepa cómo comunicarse con el servidor SQL.

* + - * **Cadena de conexión**:
        + String cadena = "jdbc:sqlserver://"+ip+":"+puerto+";"+"databaseName="+bd+";"+"encrypt=true;trustServerCertificate=true";:

Construye la cadena de conexión utilizando los atributos definidos. Incluye el protocolo JDBC, la dirección del servidor, el puerto y el nombre de la base de datos. También se especifica que se desea usar cifrado y confiar en el certificado del servidor.

* + - * **Conexión**:
        + conectar = DriverManager.getConnection(cadena,usuario,contrasenia);:

Intenta establecer la conexión a la base de datos utilizando el DriverManager con la cadena de conexión y las credenciales proporcionadas.

* + - * **Manejo de Excepciones**:
        + Si la conexión falla, se captura la excepción y se muestra un mensaje de error utilizando JOptionPane:

JOptionPane.showMessageDialog(null, "NO SE CONECTO CORRECTAMENTE A LA BASE DE DATOS");

También se imprime la traza del error en la consola para ayudar en la depuración: e.printStackTrace();

* + Finalmente, el método devuelve el objeto conectar, que puede ser null si la conexión falló.

### Consideraciones

* **Manejo de Credenciales**:
  + Las credenciales (usuario y contraseña) están hardcodeadas en la clase, lo que no es recomendable para aplicaciones en producción. Es mejor utilizar archivos de configuración o variables de entorno para manejar la información sensible.
* **Cifrado y Certificado**:
  + La cadena de conexión incluye opciones para cifrar la conexión y confiar en el certificado del servidor, lo que es una buena práctica de seguridad.
* **Reutilización**:
  + Este diseño permite reutilizar la clase CConexion en diferentes partes de la aplicación, facilitando la gestión de conexiones a la base de datos.