

### Iniciar sesión

Usuario:

Contraseña:

☒ Recordar Usuario[¿No tienes cuenta? ¡Regístrate ahora!](#)[¿No tienes cuenta? ¡Regístrate ahora!](#)☒ Recordar Usuario

Contraseña:

Usuario:

# THE BARBER APP

PROYECTO INTEGRADO - ANDRÉS RAMÍREZ GÓMEZ - CURSO 24/25

## ÍNDICE

INTRODUCCIÓN .....	2
BASE DE DATOS .....	4
DIAGRAMA DIA .....	4
ER .....	4
DIAGRAMA MYSQL WORKBENCH .....	5
VISTAS .....	6
VISTALOGIN .....	6
MÉTODOS VISTALOGIN .....	6
VISTAREGISTRO .....	8
MÉTODOS VISTAREGISTRO .....	8
VISTAINICIO .....	10
MÉTODOS VISTAINICIO .....	10
VISTACITAS .....	13
MÉTODOS VISTACITAS .....	13
VISTAINVENTARIO .....	16
MÉTODOS VISTAINVENTARIO .....	16
DIALOGOS CITAS .....	19
DIALOGOCREARCITA .....	19
MÉTODOS DIALOGOCREARCITA .....	19
DIALOGOEDITARCITA .....	21
MÉTODOS DIALOGOEDITARCITA .....	21
DIALOGOELIMINARCITA .....	23
MÉTODOS DIALOGOELIMINARCITA .....	23
DIALOGOS INVENTARIO .....	25
DIALOGOAGREGARPRODUCTO .....	25
MÉTODOS DIALOGOAGREGARPRODUCTO .....	25
DIALOGOEDITARPRODUCTO .....	27
MÉTODOS DIALOGOEDITARPRODUCTO .....	27
DIALOGOELIMINARPRODUCTO .....	29
MÉTODOS DIALOGOELIMINARPRODUCTO .....	29
CALENDARIO .....	31
MÉTODOS CALENDARIO .....	31
CONEXIÓNBD .....	34
MÉTODOS CONEXIÓNBD .....	34
USUARIOS .....	34
SERVICIOS .....	34
INVENTARIO .....	35
CITAS .....	35
ROUNDEDCELLRENDERER .....	37
MÉTODOS ROUNDEDCELLRENDERER .....	37
SESIÓNUSUARIO .....	39
MÉTODOS SESIONUSUARIO .....	39
CONTROLADOR .....	41
MÉTODOS CONTROLADOR .....	41
CLASES POJO .....	43
PRODUCTO .....	43
MÉTODOS PRODUCTO .....	43
CITA .....	44
MÉTODOS CITA .....	44
BARBERAPP (MAIN) .....	45
MÉTODOS BARBERAPP .....	45
FUNCIONAMIENTO DE LA APP .....	46
CONTROL DE ERRORES .....	52
VISTALOGIN .....	52
VISTAREGISTRO .....	53
DIALOGOCREARCITA .....	55
DIALOGOEDITARCITA .....	55
DIALOGOAGREGARPRODUCTO .....	56
DIALOGOEDITARPRODUCTO .....	58
CONCLUSIÓN .....	60

# INTRODUCCIÓN

En muchas barberías pequeñas y medianas, la gestión diaria de citas y productos se realiza de forma manual o con herramientas no especializadas, como agendas en papel, hojas de cálculo o aplicaciones genéricas. Esta metodología provoca varios problemas recurrentes:

- \* Confusión con las citas, provocando solapamientos o pérdida de clientes.
- \* Falta de visibilidad sobre el stock, lo que genera desabastecimientos inesperados de productos esenciales.
- \* Poca personalización, ya que no se lleva un control individual de los clientes ni del historial de servicios.
- \* Pérdida de tiempo, al no contar con una interfaz unificada y ágil para registrar, consultar o modificar la información.

Esta falta de digitalización afecta directamente a la eficiencia, la organización del trabajo diario y la satisfacción del cliente.

TheBarberApp resuelve este problema proporcionando una plataforma centralizada y visualmente atractiva, especialmente diseñada para barberías.

TheBarberApp es una aplicación de escritorio desarrollada en Java, diseñada para gestionar de manera eficiente una barbería.

Su objetivo principal es facilitar la administración de citas, inventario de productos y usuarios, optimizando las tareas cotidianas tanto para el personal como para los clientes.

La interfaz gráfica está construida con Swing y hace uso de un diseño visual moderno y funcional, con paneles dinámicos y tablas estilizadas que permiten una experiencia intuitiva.

Entre sus funcionalidades principales se incluyen:

- Gestión de citas personalizadas por usuario, con un calendario interactivo.
- Control de inventario con visualización de productos con bajo stock.
- Autenticación de usuarios y vista de información personalizada.
- Actualización automática de datos en pantalla tras cualquier cambio.

TheBarberApp está pensada para pequeñas y medianas barberías que buscan una herramienta práctica, profesional y fácil de usar para centralizar la gestión de su negocio.

# ESTRUCTURA DEL PROYECTO

BarberApp/

└─ src/

└─ main/

└─ java/

└─ es/

└─ studium/

└─ BarberApp/

└─ BarberApp.java

└─ Calendario.java

└─ Cita.java

└─ ConexionBD.java

└─ Controlador.java

└─ DialogoAgregarProducto.java

└─ DialogoCrearCita.java

└─ DialogoEditarCita.java

└─ DialogoEditarProducto.java

└─ DialogoEliminarCita.java

└─ DialogoEliminarProducto.java

└─ Producto.java

└─ RoundedCellRenderer.java

└─ SesionUsuario.java

└─ VistaCitas.java

└─ VistaInicio.java

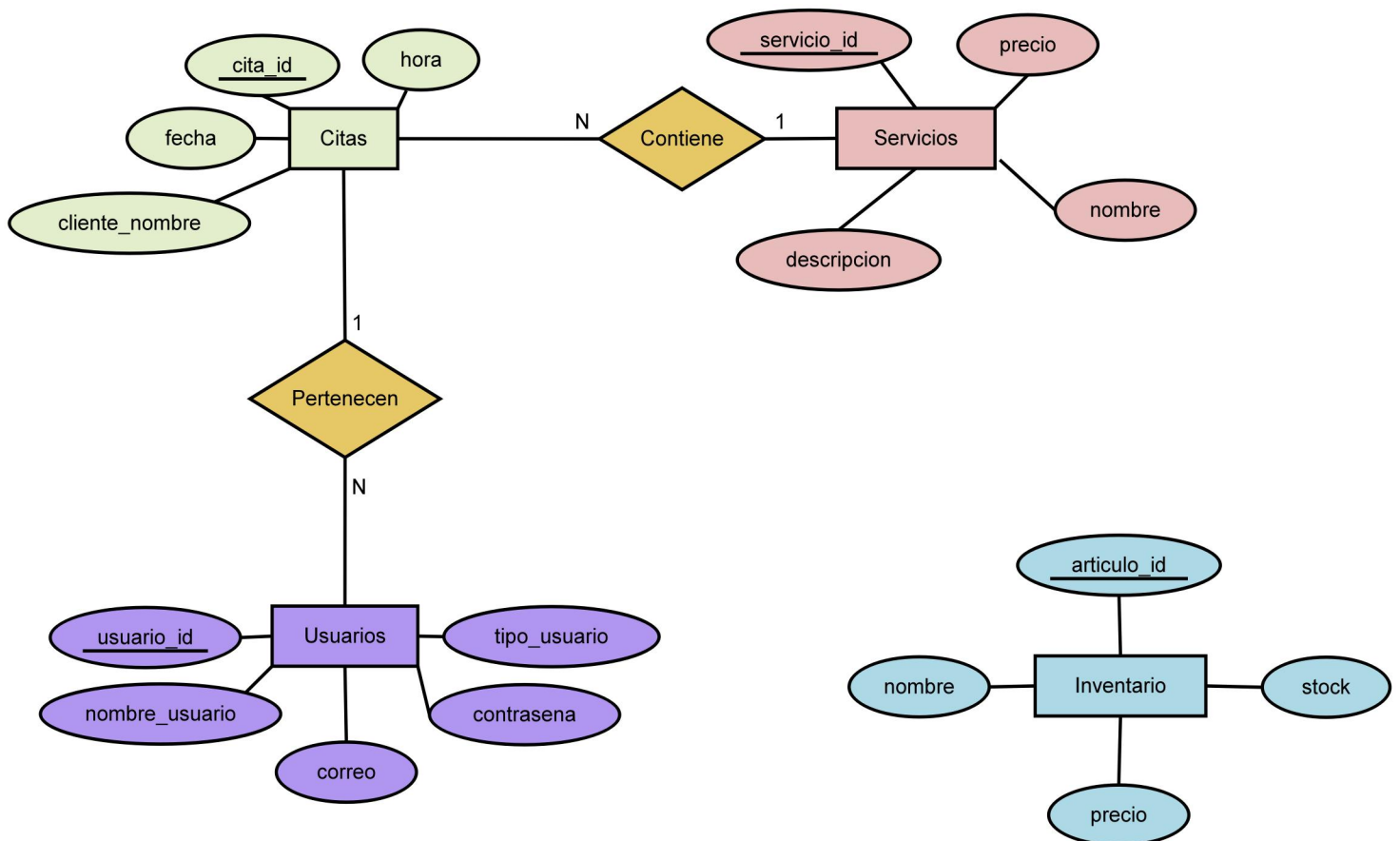
└─ VistaInventario.java

└─ VistaLogin.java

└─ VistaRegistro.java

# BASE DE DATOS

## DIAGRAMA DIA



## ER

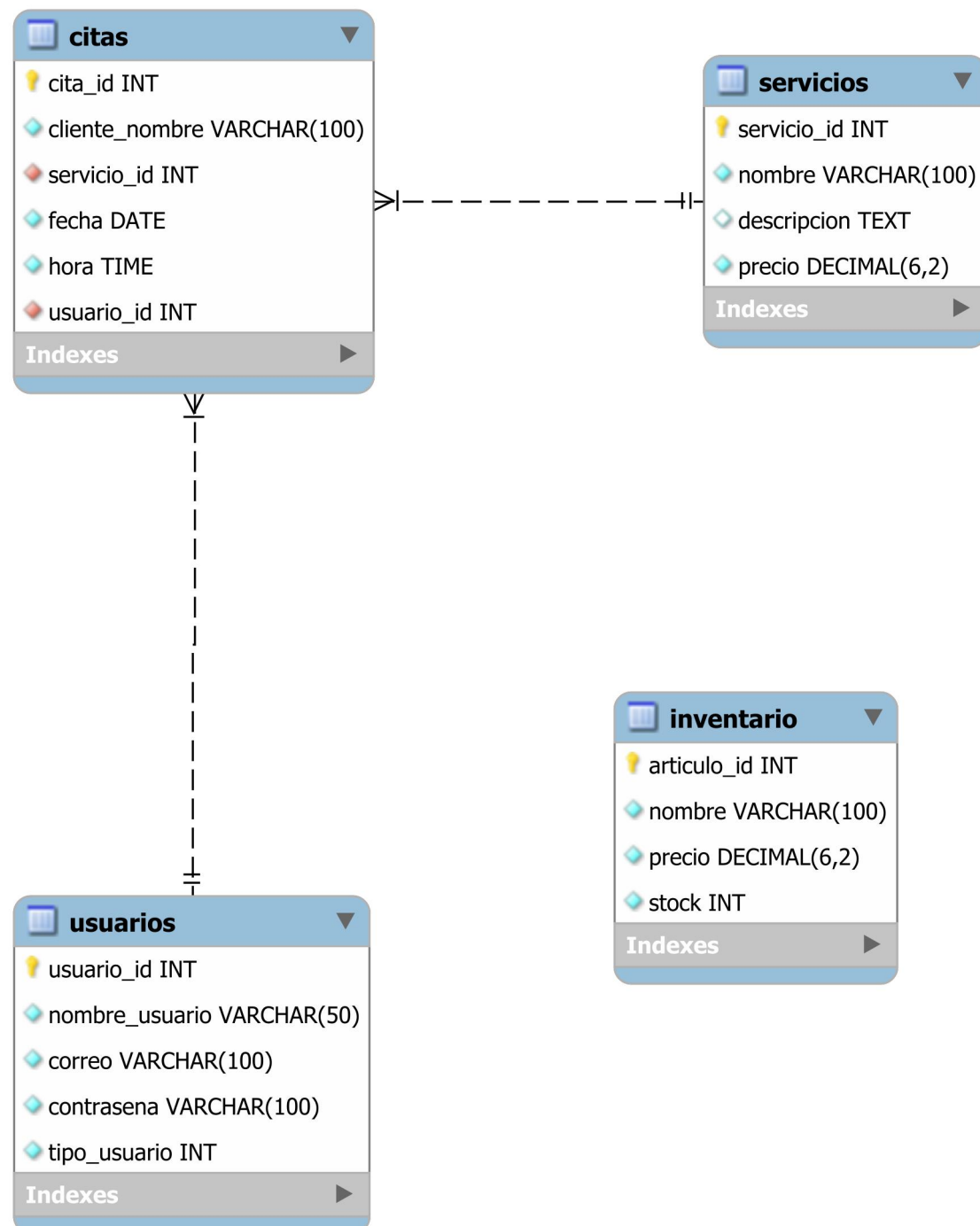
usuarios(#usuario\_id, nombre\_usuario, correo, contraseña, tipo\_usuario);

servicios(#servicio\_id, nombre, descripcion, precio);

inventario(#articulo\_id, nombre, precio, stock);

citas(#cita\_id, cliente\_nombre, fecha, hora, servicio\_id, usuario\_id);

## DIAGRAMA MYSQL WORKBENCH



# VISTAS

## *VISTALOGIN*

La clase VistaLogin es, como su nombre indica, una vista para iniciar sesión en la aplicación TheBarberApp. Permite a los usuarios ingresar sus credenciales, activar la opción de recordar usuario, la cual guarda el nombre del usuario una vez inicia sesión si el check está marcado. Desde esta vista, también podrán acceder al registro si no tienen cuenta. Valida los datos con ConexionBD y comunica los eventos mediante la interfaz LoginListener. Su diseño visual incluye una imagen de fondo y un panel redondeado para mejorar la estética mediante Swing.

## METODOS VISTALOGIN

### **Constructor public VistaLogin()**

- \* Inicializa todos los componentes gráficos del formulario de login.
- \* Carga la imagen de fondo y ajusta los tamaños dinámicamente.
- \* Configura los listeners para botones y enlaces.
- \* Llama al método cargarUsuarioGuardado() al iniciar.

### **public void setLoginListener(LoginListener l)**

Asigna un listener externo para manejar eventos como login exitoso o acceso al formulario de registro.

### **private void guardarUsuario(String usuario)**

Guarda el nombre de usuario en un archivo de texto en el directorio del usuario para recordarlo la próxima vez.

El archivo: recordar.txt se crea en la carpeta del usuario del sistema operativo.

### **private void cargarUsuarioGuardado()**

Carga automáticamente el usuario guardado en el campo de texto si existe el archivo recordar.txt. También marca el checkbox de "Recordar Usuario".

### **private void borrarUsuarioGuardado()**

Elimina el archivo recordar.txt para que no se recuerde más el usuario al cerrar sesión con la opción desmarcada.

### Clase interna - static class RoundedPanel

Clase personalizada que hereda de JPanel para dibujar un panel con esquinas redondeadas y color de fondo semitransparente. Mejora la estética de la interfaz.

### CLASES CON LAS QUE SE CONECTA DIRECTAMENTE

#### ConexionBD

Se utiliza para verificar las credenciales ingresadas por el usuario.

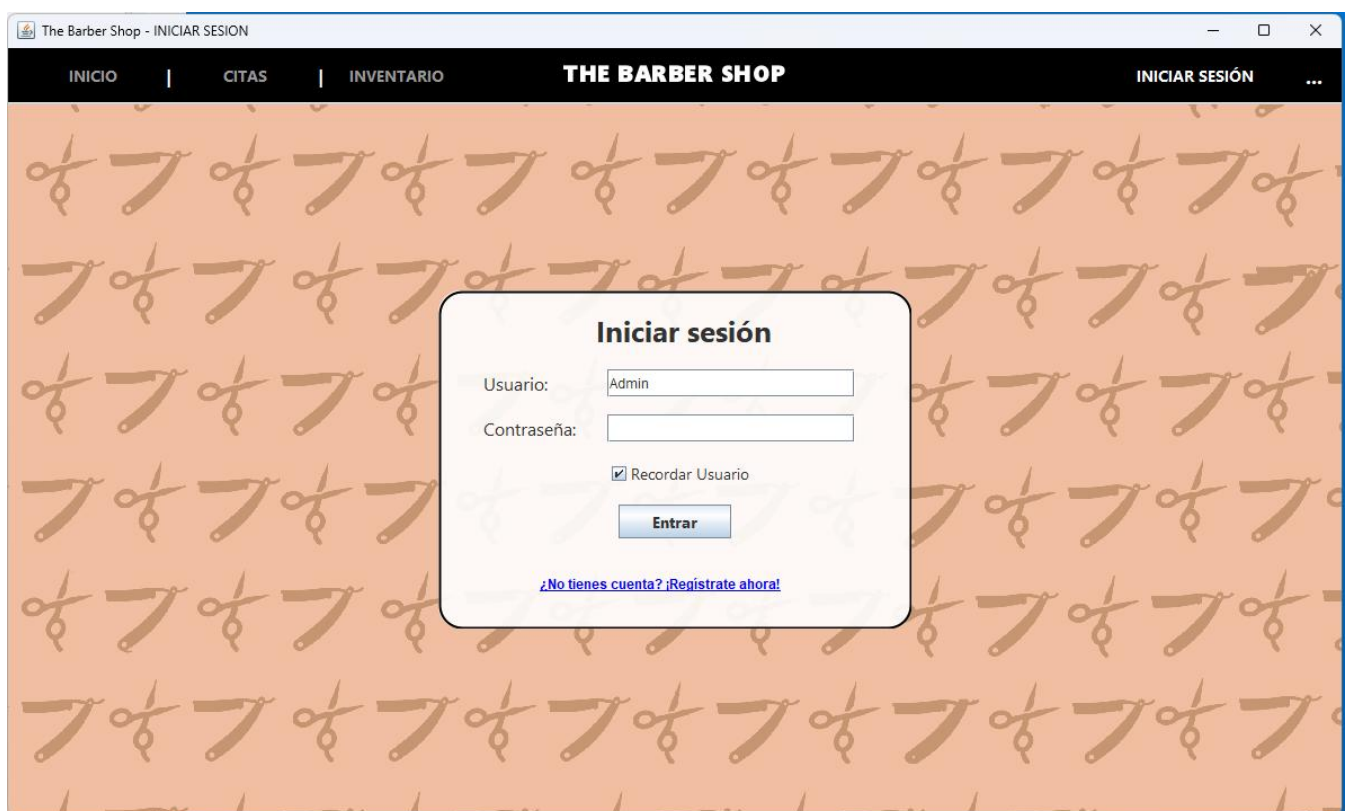
Método llamado: `autenticarUsuario(String usuario, String contrasena)` → devuelve un entero según el tipo de usuario o -1 si no es válido.

#### LoginListener (interfaz interna)

Permite que la clase externa que contenga VistaLogin maneje los eventos:

`onLoginSuccess(String usuario, int tipoUsuario)`  
`onShowRegistro()`

Estas acciones podrían redirigir al usuario al menú principal o a la vista de registro, dependiendo del caso.





## ***VISTAREGISTRO***

La clase VistaRegistro es un componente gráfico (JPanel) de la aplicación TheBarberApp que permite a un nuevo usuario registrarse en el sistema.

Presenta una interfaz visual amigable que solicita nombre de usuario, correo electrónico, contraseña y confirmación de contraseña.

Realiza validaciones básicas, muestra mensajes de error en caso de datos inválidos y, si todo es correcto, registra al usuario en la base de datos mediante la clase ConexionBD.

Además, permite navegar de regreso a la vista de inicio de sesión mediante un enlace interactivo.

## **METODOS VISTAREGISTRO**

### **public VistaRegistro()**

Constructor principal de la clase.

Inicializa todos los componentes visuales de la pantalla de registro:

- \* Crea el fondo personalizado con imagen.
- \* Crea un panel redondeado semitransparente donde se colocan los campos de texto y botones.
- \* Define los listeners para validación, envío de datos y navegación a login.

### **private String hashPassword(String password)**

Recibe una contraseña en texto plano y devuelve su hash usando el algoritmo SHA-256.

### **public void setRegistroListener(RegistroListener l)**

Permite asignar un listener para recibir notificación cuando el registro es exitoso. Facilita cambiar de vista al login desde el contenedor principal cuando el usuario ha terminado el registro.

### **protected void paintComponent(Graphics g) (dentro de RoundedPanel)**

Dibuja el panel redondeado con bordes suavizados. Mejora la apariencia del formulario de registro, dándole un estilo moderno y visualmente atractivo. Esta clase anidada RoundedPanel extiende JPanel y permite el diseño personalizado.

## EVENTOS Y ACCIONES

### **btnRegistrar.addActionListener**

Al hacer clic en el botón "Registrarse", realiza lo siguiente:

- \* Valida los campos.
- \* Verifica el formato del correo.
- \* Compara las contraseñas.
- \* Comprueba si el usuario ya existe (ConexionBD.usuarioYaExiste()).
- \* Hashea la contraseña.
- \* Guarda al usuario en la base de datos (ConexionBD.agregarUsuario()).
- \* Llama al listener para volver a la vista de login.

### **lblLogin.addMouseListener**

Al hacer clic en el texto "¿Ya tienes cuenta?", invoca el listener para cambiar de vista a la pantalla de inicio de sesión.

### **addComponentListener**

Centra el panel de registro cada vez que la ventana cambia de tamaño, para mantener el diseño visual centrado.

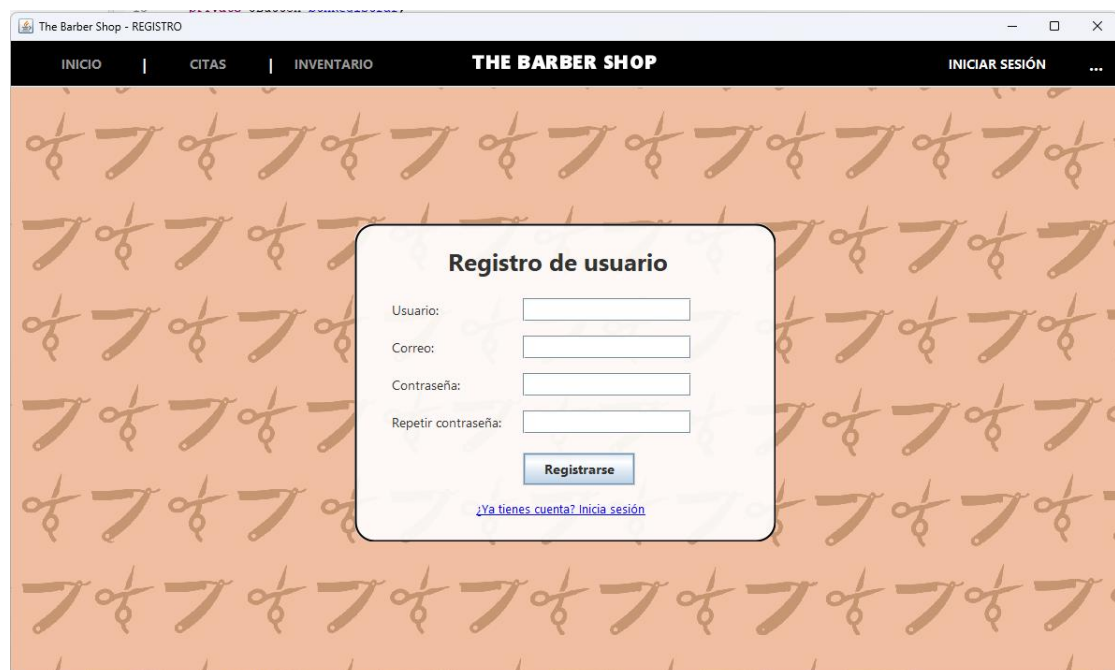
## CLASES CON LAS QUE SE CONECTA DIRECTAMENTE

### **ConexionBD**

Para verificar si el usuario o correo ya existen y para registrar el nuevo usuario en la base de datos.

### **RegistroListener (interfaz interna)**

Cambia la vista al login una vez que el usuario se ha registrado correctamente. Esto se realiza a través del RegistroListener. Esto asegura que la navegación entre vistas sea controlada desde el contenedor principal.



## ***VISTAINICIO***

La clase VistaInicio.java es la ventana principal de la aplicación de gestión de barbería. Esta clase extiende JFrame y gestiona la navegación entre distintas vistas mediante un CardLayout, como las pantallas de inicio, citas, inventario, login y registro.

Además, muestra un panel de inicio personalizado con dos tablas que informan:

- \* Los artículos del inventario con menor stock.
- \* Las citas programadas para el día actual del usuario logueado.

También incorpora un menú lateral con botones estilizados para cambiar de vista, una imagen de fondo, y un menú superior con el nombre del usuario en sesión y un botón para cerrar sesión.

## **METODOS VISTAINICIO**

### **VistaInicio() (Constructor)**

Es el método principal que se ejecuta al instanciar la clase. Inicializa todos los componentes gráficos, como el fondo, el menú lateral, las vistas disponibles en el CardLayout, el panel de bienvenida, y establece el diseño visual general de la ventana.

### **setUserSession(int usuarioid, String nombre)**

Establece los datos de sesión del usuario que ha iniciado sesión. Muestra su nombre en la barra superior y activa el botón de cerrar sesión. También se encarga de actualizar las tablas del panel de inicio para mostrar los datos correspondientes a ese usuario.

### **cambiarVista(String nombreVista)**

Permite cambiar entre vistas internas del CardLayout (por ejemplo, cambiar a la vista de citas o inventario). También fuerza la actualización de los datos mostrados en el panel de inicio cuando se selecciona esa vista.

### **cerrarSesion()**

Este método se llama cuando el usuario decide cerrar sesión. Restaura el estado inicial del sistema, elimina los datos de sesión, limpia las tablas del panel de inicio y devuelve la aplicación a la vista de login.

### **crearPanelInicio()**

Construye y devuelve el panel visual que se muestra como vista principal de bienvenida. Este panel incluye las tablas de "Citas del día" y "Artículos con menor stock", así como sus respectivos títulos estilizados y componentes visuales redondeados.

### **crearTablaEstilizada(DefaultTableModel modelo)**

Recibe un modelo de tabla y genera un JTable personalizado con diseño estilizado, utilizando renderizadores y ajustes de visualización para dar una mejor presentación a los datos.

### **actualizarTablasInicio()**

Este método es el encargado de consultar la base de datos para obtener las citas del día y los artículos con bajo stock. Luego, carga estos datos en las tablas correspondientes del panel de inicio. Solo se ejecuta si hay un usuario con sesión activa.

### **EVENTOS Y ACCIONES**

Todos los botones del menú lateral tienen un ActionListener que cambia la vista visible del CardLayout.

El botón "Cerrar sesión" también tiene un ActionListener, pero realiza acciones adicionales como limpieza de estado y retorno al login.

Al establecer la sesión del usuario (mediante login o registro), se desencadena de forma automática una actualización de los datos mostrados en el panel de inicio.

Los eventos son:

- \* Click en el botón "Inicio".
- \* Click en el botón "Citas".
- \* Click en el botón "Inventario".
- \* Click en el botón "Cerrar sesión".
- \* Evento implícito: Cambio de sesión con setUserSession()

### **CLASES CON LAS QUE SE CONECTA DIRECTAMENTE**

#### **ConexionBD**

Para verificar si el usuario o correo ya existen y para registrar el nuevo usuario en la base de datos.

#### **RegistroListener (interfaz interna)**

Cambia la vista al login una vez que el usuario se ha registrado correctamente. Esto se realiza a través del RegistroListener. Esto asegura que la navegación entre vistas sea controlada desde el contenedor principal.

#### **VistaLogin y VistaRegistro**

Son las pantallas de acceso y registro de usuarios. Se integran en el CardLayout y son mostradas desde la vista principal según corresponda.

#### **VistaCitas y VistaInventario**

Representan las secciones del sistema donde se gestionan las citas y el inventario. También están incluidas en el CardLayout de la ventana principal y son accesibles mediante los botones del menú lateral.

#### **SesionUsuario**

Es la clase encargada de mantener los datos de la sesión del usuario actual, como su ID y su nombre. VistaInicio utiliza esta clase para mostrar información personalizada.

#### **ConexionBD**

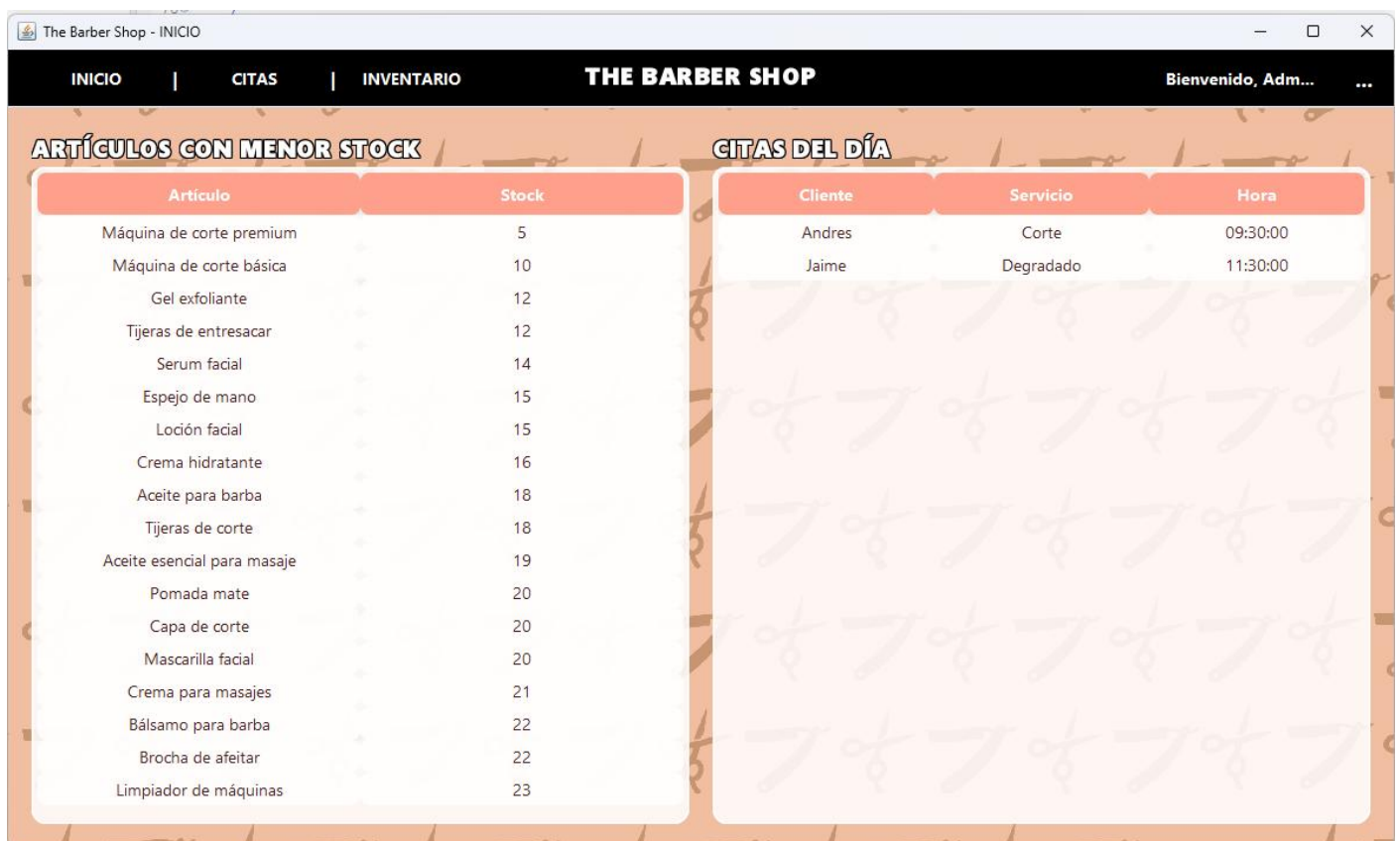
Se utiliza para realizar consultas a la base de datos, específicamente para obtener los artículos con menor stock y las citas del día.

### RoundedPanel y RoundedCellRenderer

Son clases utilitarias que permiten aplicar un diseño visual más atractivo a los paneles y celdas de las tablas, con bordes redondeados y estilos personalizados.

### OutlineLabel

Es una clase para crear etiquetas con texto contorneado, utilizadas en los títulos de las secciones del panel de inicio.



## ***VISTACITAS***

La clase VistaCitas forma parte de la interfaz gráfica de la aplicación TheBarberApp y representa la vista principal de gestión de citas del usuario. Esta pantalla muestra un calendario interactivo a la izquierda y, a la derecha, una tabla con las citas del día seleccionado.

Funcionalidades clave:

- \* Permite ver las citas del día actual o de cualquier día seleccionado en el calendario.
- \* Permite editar una cita con doble clic.
- \* Permite eliminar una cita dejando pulsado sobre ella medio segundo (long press).
- \* Los elementos se adaptan automáticamente al tamaño de la ventana.
- \* Se conecta con la base de datos a través de ConexionBD y muestra solo las citas del usuario actualmente logueado (SesionUsuario.usuarioId).

## **METODOS VISTACITAS**

### **public VistaCitas()**

Constructor que inicializa toda la interfaz gráfica de la vista de citas: el panel de fondo, el calendario, la tabla de citas, los botones de navegación.

Establece los listeners para redimensionar y para eventos del ratón y carga automáticamente las citas del día actual al iniciar.

### **@Override**

### **public void fechaSeleccionada(LocalDate fecha)**

Método implementado desde la interfaz Calendario.FechaSeleccionadaListener. Se activa cuando el usuario selecciona una fecha en el calendario y llama a cargarTusCitas(fecha) para cargar las citas del día seleccionado.

### **private LocalDate citaFecha()**

Método auxiliar que devuelve la última fecha seleccionada en el calendario.

### **public void cargarTusCitas(LocalDate fecha)**

Consulta la base de datos para obtener las citas de un usuario específico y una fecha concreta, además, refresca la información de la tabla de citas con los resultados.

Se conecta con ConexionBD.obtenerCitasPorUsuarioYFecha() y utiliza el ID del usuario actual de SesionUsuario.

## EVENTOS Y ACCIONES

### **mousePressed**

Detecta una pulsación sobre una fila de la tabla e inicia un temporizador para eliminar la cita si se mantiene presionado.

### **mouseReleased**

Cancela el temporizador de eliminación si se suelta el botón del ratón antes de tiempo.

### **mouseClicked**

Detecta doble clic sobre una fila de la tabla y abre el diálogo de edición de la cita seleccionada.

### **componentResized** (de VistaCitas)

Ajusta el tamaño del panel de fondo y del panel principal al cambiar el tamaño del contenedor principal.

### **componentResized** (de panelCompleto)

Reorganiza el calendario y la tabla de citas según el nuevo tamaño del panel contenedor.

### **fechaSeleccionada**

Se activa al elegir una fecha en el calendario y recarga las citas del usuario para ese día.

## CLASES CON LAS QUE SE CONECTA DIRECTAMENTE

### **ConexionBD**

Para consultar la base de datos y obtener las citas del usuario según la fecha.

### **Calendario**

Para mostrar el calendario interactivo.

- \* Usa el método `setFechaSeleccionadaListener()` para reaccionar a cambios en la fecha seleccionada.
- \* Usa `obtieneUltimaFechaSeleccionada()` para obtener la fecha seleccionada en el momento.

### **Cita**

Representa cada cita mostrada en la tabla. Se usa para extraer el nombre del cliente, el servicio y la hora.

### **DialogoEditarCita**

Se abre al hacer doble clic en una fila de la tabla, permitiendo al usuario editar los datos de una cita.

### **DialogoEliminarCita**

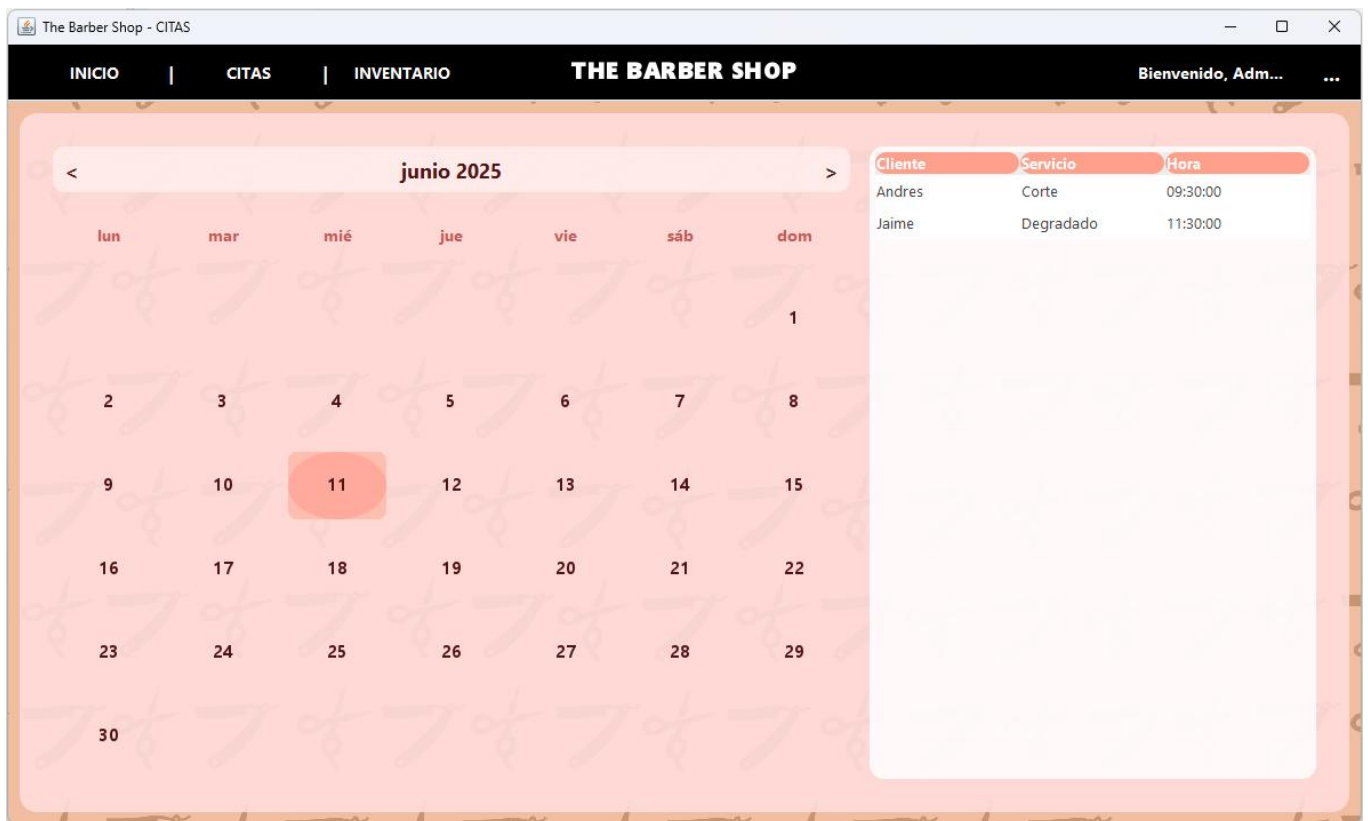
Se abre al mantener pulsado una fila medio segundo y permite al usuario confirmar la eliminación de una cita.

### SesionUsuario

Proporciona el `userId` del usuario actualmente logueado, se usa para filtrar las citas que se muestran.

### RoundedCellRenderer (de Calendario)

Se usa para personalizar el diseño de las celdas de la tabla, haciéndolas más estilizadas.





## ***VISTAINVENTARIO***

La clase VistaInventario forma parte de la interfaz gráfica de la aplicación TheBarberApp y representa la vista principal para la gestión del inventario de productos disponibles. Esta pantalla muestra una tabla estilizada con todos los artículos registrados en la base de datos, así como botones para agregar, editar o eliminar productos.

Funcionalidades clave:

- \* Muestra todos los artículos del inventario registrados por el usuario actual.
- \* Permite agregar un nuevo artículo al inventario.
- \* Permite editar los datos de un artículo seleccionado.
- \* Permite eliminar un artículo seleccionado.
- \* Los botones Editar y Eliminar solo se activan cuando hay un artículo seleccionado.
- \* Los elementos visuales (tabla y botones) se adaptan automáticamente al tamaño de la ventana.
- \* Estilización visual con fondo personalizado, panel translúcido redondeado y celdas estilizadas.
- \* Se conecta con la base de datos a través de ConexionBD y filtra por el ID del usuario activo (SesionUsuario.usuarioId).

## **METODOS VISTAINVENTARIO**

### **public VistaInventario()**

Constructor que inicializa la interfaz gráfica de la vista de inventario:

- \* Crea y configura el panel de fondo con imagen.
- \* Crea un panel semitransparente y redondeado que contiene la tabla y los botones.
- \* Carga automáticamente todos los artículos del usuario actual.
- \* Añade listeners para eventos de selección de filas, clics en botones y redimensionamiento de la ventana.

### **public void cargarArticulos()**

Consulta la base de datos a través de ConexionBD.obtenerArticulosPorUsuario() y actualiza la tabla con todos los artículos del usuario actualmente logueado (SesionUsuario.usuarioId).

### **private void ajustarColumnas()**

Método auxiliar que ajusta el ancho de las columnas de la tabla para adaptarse automáticamente al tamaño del componente, garantizando una visualización adecuada.

### **private void actualizarEstadoBotones()**

Activa o desactiva los botones Editar y Eliminar según si hay una fila seleccionada en la tabla.

### **private void configurarTabla()**

Aplica un renderizador personalizado (RoundedCellRenderer) a las celdas de la tabla para que se muestren con bordes redondeados, color blanco semitransparente y sin bordes visibles.

## EVENTOS Y ACCIONES

### **mouseClicked** (sobre la tabla)

Detecta clics sobre la tabla. Al hacer clic sobre una fila:

- \* Se selecciona el artículo correspondiente.
- \* Se activan los botones Editar y Eliminar.

### **btnAgregar** (ActionListener)

Al pulsar el botón Agregar, se abre el diálogo DialogoAgregarArticulo, donde el usuario puede introducir los datos de un nuevo producto. Al cerrar el diálogo, la tabla se actualiza automáticamente.

### **btnEditar** (ActionListener)

Al pulsar el botón Editar, si hay una fila seleccionada, se abre DialogoEditarArticulo con los datos del artículo. Tras la edición, la tabla se recarga para reflejar los cambios.

### **btnEliminar** (ActionListener)

Al pulsar el botón Eliminar, si hay una fila seleccionada, se abre el DialogoEliminarArticulo para confirmar la acción. Si se elimina el artículo, la tabla se actualiza.

### **componentResized** (del contenedor principal)

Ajusta dinámicamente el tamaño del panel de fondo, del panel contenedor y de la tabla según el tamaño actual de la ventana.

## CLASES CON LAS QUE SE CONECTA DIRECTAMENTE

### **ConexionBD**

Se usa para interactuar con la base de datos:

obtenerArticulosPorUsuario(): obtener los artículos del usuario actual.  
agregarArticulo(), editarArticulo(), eliminarArticulo(): modificar el inventario.

### **Articulo**

Representa cada artículo del inventario, incluyendo nombre, descripción, stock y precio.

### **DialogoAgregarArticulo**

Se abre al pulsar el botón Agregar. Permite ingresar los datos de un nuevo producto.

### **DialogoEditarArticulo**

Se abre al pulsar Editar con una fila seleccionada. Permite modificar los datos del artículo.

### **DialogoEliminarArticulo**

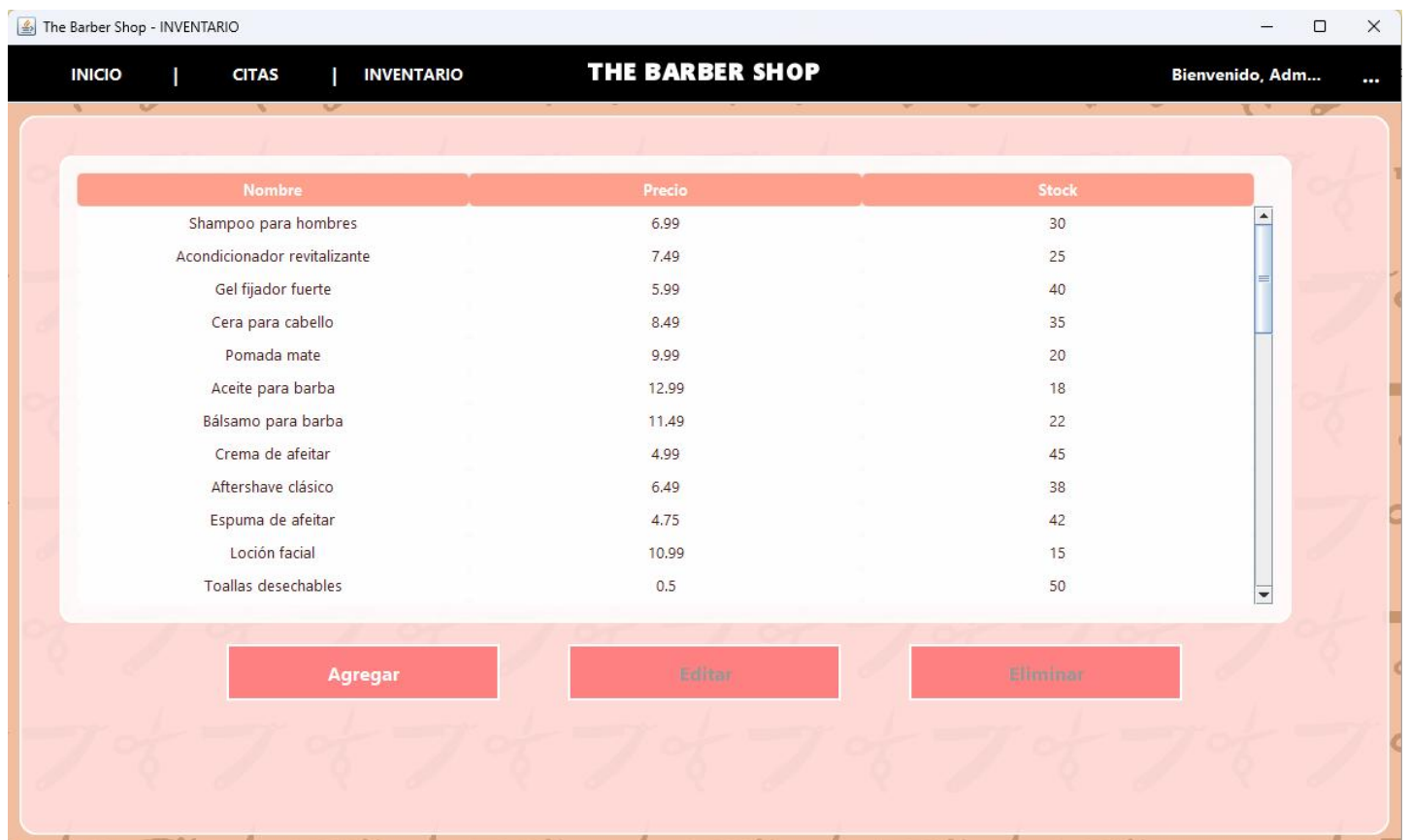
Se abre al pulsar Eliminar con una fila seleccionada. Permite confirmar o cancelar la eliminación del artículo.

### SesionUsuario

Proporciona el `usuarioId` del usuario actualmente logueado. Este ID se usa para filtrar los artículos del inventario que pertenecen a ese usuario.

### RoundedCellRenderer

Se utiliza para personalizar el diseño de las celdas de la tabla, haciéndolas visualmente consistentes con el resto de la aplicación, especialmente la vista de citas.



The screenshot shows a web application window titled "The Barber Shop - INVENTARIO". The interface has a dark header with navigation links: "INICIO", "CITAS", and "INVENTARIO" (which is active). The main content area has a light pink background with a repeating pattern of scissors and combs. It features a table with three columns: "Nombre", "Precio", and "Stock". Below the table are three red buttons: "Agregar", "Editar", and "Eliminar".

Nombre	Precio	Stock
Shampoo para hombres	6.99	30
Acondicionador revitalizante	7.49	25
Gel fijador fuerte	5.99	40
Cera para cabello	8.49	35
Pomada mate	9.99	20
Aceite para barba	12.99	18
Bálsamo para barba	11.49	22
Crema de afeitar	4.99	45
Aftershave clásico	6.49	38
Espuma de afeitar	4.75	42
Loción facial	10.99	15
Toallas desechables	0.5	50

# DIALOGOS CITAS

## *DIALOGOCREARCITA*

La clase DialogoCrearCita forma parte de la interfaz gráfica de la aplicación TheBarberApp y representa el formulario emergente que permite al usuario registrar una nueva cita en un día específico del calendario. Este diálogo aparece al hacer doble clic en un día desde el componente calendario, y está diseñado con el mismo estilo visual que el resto de pantallas de la aplicación.

Funcionalidades clave:

- \* Permite introducir el nombre del cliente de forma manual.
- \* Permite seleccionar uno de los servicios ofrecidos por la barbería (corte, tinte, arreglo de barba, etc.).
- \* Muestra automáticamente las horas disponibles en franjas de media hora entre las 10:00–14:00 y 17:00–21:00.
- \* Se conecta con la base de datos a través de ConexionBD para obtener los servicios y verificar las horas disponibles según el día.
- \* Al guardar la cita, actualiza automáticamente las vistas VistaCitas y VistaInicio para reflejar el cambio.

## METODOS DIALOGOCREARCITA

**public DialogoCrearCita(JFrame parent, ConexionBD conexion, LocalDate fecha, int usuarioid, VistaCitas vistaCitas)**

Inicializa la interfaz del formulario de nueva cita, recibe la ventana principal como parent, la conexión a base de datos, la fecha seleccionada, el ID del usuario actual y una referencia a VistaCitas para poder actualizarla tras la creación.

**public boolean isConfirmado()**

Devuelve true si la cita se creó con éxito.

Este método puede ser usado externamente para comprobar si se confirmó la acción dentro del diálogo.

## EVENTOS Y ACCIONES

**btnGuardar (ActionListener)**

Valida que el nombre del cliente tenga al menos tres letras y que se haya seleccionado una hora. Luego obtiene el servicio elegido y registra la cita en la base de datos mediante `conexion.agregarCita(...)`.

También actualiza VistaCitas y VistaInicio para mostrar los cambios de forma inmediata.

### **btnCancelar (ActionListener)**

Cierra el formulario sin realizar ninguna acción si el usuario pulsa "Cancelar".

El combo de horas (cbHora) muestra únicamente las horas disponibles para la fecha seleccionada. Estas horas se generan dinámicamente en franjas de 30 minutos, entre las 10:00–14:00 y 17:00–21:00, y luego se filtran llamando al método `conexion.consultarHorasDisponibles(String fecha)` para excluir las ya ocupadas.

## **CLASES CON LAS QUE SE CONECTA DIRECTAMENTE**

### **ConexionBD**

Utiliza `obtenerServicios()` para llenar el combo de servicios y `consultarHorasDisponibles()` para mostrar solo las horas libres. También utiliza `agregarCita()` para insertar la nueva cita en la base de datos.

### **VistaCitas**

Después de guardar una nueva cita, llama a `cargarTusCitas(fecha)` para que la tabla de citas se actualice automáticamente.

### **VistaInicio**

Si está disponible, invoca `actualizarTablasInicio()` para refrescar la tabla de citas del día y el stock bajo.

### **SesionUsuario**

Se emplea para identificar qué usuario ha iniciado sesión y asociar la cita con su `usuarioid`.



## ***DIALOGOEDITARCITA***

La clase DialogoEditarCita implementa un cuadro de diálogo modal que permite al usuario editar los datos de una cita previamente registrada en el sistema. Este formulario forma parte del módulo de gestión de citas de la aplicación TheBarberApp, y proporciona una interfaz gráfica igual a DialogoCrearCita, para actualizar los campos: cliente, servicio y hora.

Funcionalidades clave:

- \* Carga inicial de datos: Precarga datos actuales en campos, para facilitar la edición (cliente, servicio, hora).
- \* Servicios: Obtiene lista actualizada desde base de datos, para que el usuario pueda cambiar.
- \* Horas disponibles: Incluye la hora actual y filtra las disponibles para ese día para evitar conflictos.

## **MÉTODOS DIALOGOEDITARCITA**

**public DialogoEditarCita(JFrame parent, ConexionBD conexion, Cita cita)**

Inicializa la interfaz del formulario para editar la cita, de la misma forma y con los mismos elementos visuales que DialogoCrearCita.

**public boolean isConfirmado()**

Devuelve true si la cita se editó con éxito.

Este método puede ser usado externamente para comprobar si se confirmó la acción dentro del diálogo.

## **EVENTOS Y ACCIONES**

**btnGuardar (ActionListener)**

- \* Invoca el método conexion.modificarCita(...) pasando datos nuevos.
- \* Usa SesionUsuario.usuarioId para el usuario autenticado.
- \* Muestra mensaje de éxito o error.
- \* Cierra diálogo si se guarda con éxito.

**btnCancelar (ActionListener)**

Cierra el formulario sin realizar ninguna acción si el usuario pulsa "Cancelar".

El combo de horas (cbHora) muestra únicamente las horas disponibles para la fecha seleccionada. Estas horas se generan dinámicamente en franjas de 30 minutos, entre las 10:00–14:00 y 17:00–21:00, y luego se filtran llamando al método conexion.consultarHorasDisponibles(String fecha) para excluir las ya ocupadas.

## CLASES CON LAS QUE SE CONECTA DIRECTAMENTE

### ConexionBD

Es la clase encargada de la conexión y gestión de la base de datos. Desde DialogoEditarCita se utilizan sus métodos para:

- \* Obtener la lista de servicios disponibles (`conexion.obtenerServicios()`).
- \* Consultar las horas libres de un día específico excluyendo la cita actual (`conexion.obtenerHorasLibres(fecha, cita)`).
- \* Modificar la cita existente en la base de datos (`conexion.modificarCita(...)`).

### Cita

Representa la entidad cita con todos sus atributos (fecha, hora, cliente, servicio, id, usuarioid, etc.). En el diálogo se usa un objeto Cita para cargar los datos actuales que se quieren editar y para hacer referencia a la cita a modificar en la base de datos.

### VistaCitas

Vista principal que muestra las citas del usuario. Tras modificar una cita, se recomienda llamar a `vistaCitas.cargarTusCitas(fecha)` para refrescar la tabla de citas y mostrar los cambios inmediatamente.

### VistaInicio

Vista principal de la aplicación que puede contener resúmenes y otras tablas.

Se puede actualizar tras modificar la cita mediante `vistaInicio.actualizarTablasInicio()` para reflejar cambios en otras partes de la interfaz.

### SesionUsuario

Clase estática que mantiene información del usuario autenticado, en particular el usuarioid que se usa para asociar la cita con el usuario que la crea o modifica.



## ***DIALOGOELIMINARCITA***

Mostrar un diálogo modal de confirmación para eliminar una cita específica con un estilo visual parecido a los anteriores. Si el usuario acepta, se llama al método correspondiente en ConexionBD para realizar el borrado.

## **MÉTODOS DIALOGOELIMINARCITA**

**public DialogoEliminarCita(JFrame parent, ConexionBD conexion, int citald, String descripcion)**

Crea un cuadro de diálogo modal que pregunta al usuario si desea eliminar la cita indicada. También hace uso de la clase RoundedPanel de VistaLogin para mantener la estética visual.

**isConfirmado()**

Retorna un booleano que indica si el usuario confirmó la eliminación de la cita. Se utiliza desde la clase llamadora para saber si debe refrescar los datos.

## **EVENTOS Y ACCIONES**

**btnEliminar (ActionListener)**

- \* Ejecuta conexion.eliminarCita(citald);
- \* Marca la eliminación como confirmada (confirmado = true) y cierra el diálogo.

**btnCancelar (ActionListener)**

Cierra el formulario sin realizar ninguna acción si el usuario pulsa "Cancelar".



## CLASES CON LAS QUE SE CONECTA DIRECTAMENTE

### ConexionBD

Encargada de realizar la operación de eliminación en la base de datos.  
Método usado: `conexion.eliminarCita(citaId)`.

### VistaLogin.RoundedPanel

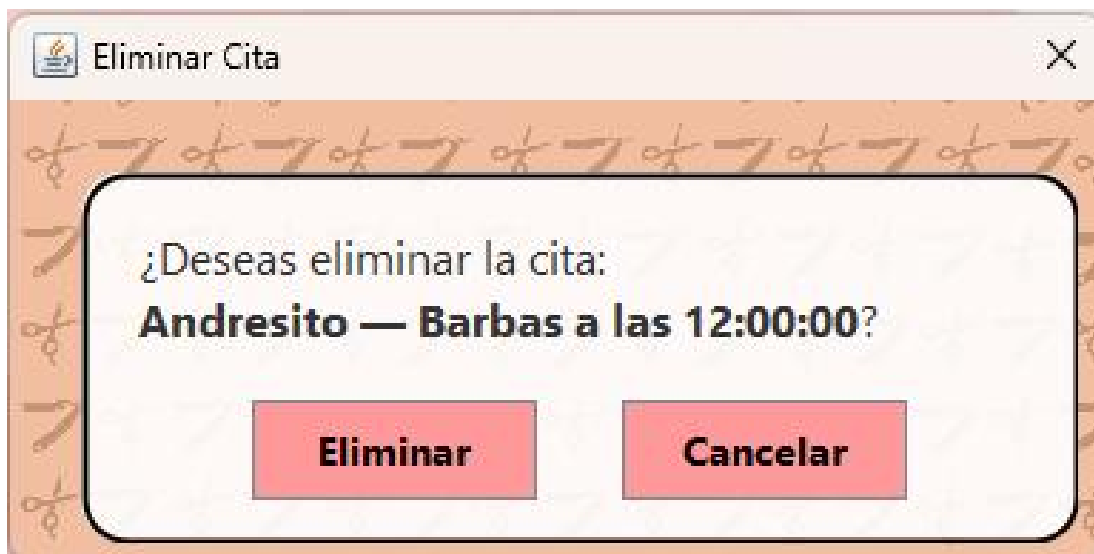
Componente visual reutilizado que proporciona un panel con bordes redondeados y fondo translúcido.

### VistaCitas

Clase que gestiona la tabla de citas. Aunque no se pasa directamente, desde allí se invoca este diálogo.

### VistaInicio

Puede llamarse `actualizarTablasInicio()` para mantener sincronizados los datos en la vista inicial tras la eliminación.



# DIALOGOS INVENTARIO

## *DIALOGOAGREGARPRODUCTO*

Muestra un cuadro de diálogo modal que permite ingresar un nuevo producto al inventario. Realiza validación de datos antes de enviarlos a la base de datos y actualiza automáticamente la vista principal si procede.

## METODOS DIALOGOAGREGARPRODUCTO

**public DialogoAgregarProducto(JFrame parent, ConexionBD conexion)**

Inicializa y configura la interfaz para agregar un nuevo producto y se encarga de mostrar campos de entrada para nombre, precio y stock. También Realiza validaciones básicas antes de insertar en la base de datos.

**public boolean isConfirmado()**

Retorna true si el producto fue agregado correctamente. Puede usarse desde otras clases para decidir si recargar los datos de la tabla.

## EVENTOS Y ACCIONES

**btnGuardar (ActionListener)**

Valida que el nombre tenga al menos 2 caracteres alfanuméricos, que el precio sea un número mayor que 0 y que el stock sea un número entero igual o mayor que 0.

Llama a conexion.agregarArticulo(...) para insertar el nuevo producto.

Muestra mensaje de éxito y actualiza la vista si se invoca desde VistaInicio. También actualiza VistaCitas y VistaInicio para mostrar los cambios de forma inmediata.

**btnCancelar (ActionListener)**

Cierra el formulario sin realizar ninguna acción si el usuario pulsa "Cancelar".

## CLASES CON LAS QUE SE CONECTA DIRECTAMENTE

### ConexionBD

Realiza la inserción del nuevo producto en la base de datos.  
Método utilizado: `agregarArticulo(String nombre, double precio, int stock)`.

### VistaLogin.RoundedPanel

Componente visual reutilizado que proporciona un panel con bordes redondeados y fondo translúcido.

### VistaInicio

Si el diálogo se abrió desde esta clase, llama a `actualizarTablasInicio()` al agregar un producto para refrescar los datos.



The image shows a Java Swing dialog box titled "Agregar Producto". Inside the dialog, there is a rounded rectangular panel with a light orange background featuring a repeating pattern of scissors. The panel is titled "Nuevo Producto". It contains three text input fields, each preceded by a label: "Nombre:", "Precio:", and "Stock:". Below these fields are two buttons: "Guardar" (Save) and "Cancelar" (Cancel). The dialog box has a standard title bar with a close button (X) in the top right corner.

## ***DIALOGOEDITARPRODUCTO***

Muestra un cuadro de diálogo modal que permite editar un producto ya existente en la BD. Realiza validación de datos antes de enviarlos a la base de datos y actualiza automáticamente la vista principal si procede.

### **METODOS DIALOGOEDITARPRODUCTO**

**public DialogoEditarProducto(JFrame parent, ConexionBD conexion, Producto producto)**

Inicializa el diálogo modal para editar un producto. Recibe como parámetros el JFrame padre, una instancia de la conexión a base de datos y el producto que se desea modificar

Configura el diseño gráfico, carga los datos del producto en los campos de texto, aplica estilos visuales y define los comportamientos de los botones Guardar y Cancelar.

**public boolean isConfirmado()**

Retorna true si el producto fue agregado correctamente. Puede usarse desde otras clases para decidir si recargar los datos de la tabla.

### **EVENTOS Y ACCIONES**

**btnGuardar (ActionListener)**

Se validan los campos introducidos (nombre, precio, stock).

Si los datos son válidos, se llama al método modificarArticulo() de la clase ConexionBD, pasando el ID del producto y los nuevos valores.

Se establece el atributo confirmado en true.

Si la ventana padre es una instancia de VistaInicio, se actualizan las tablas de productos.

Finalmente, se cierra el diálogo.

**btnCancelar (ActionListener)**

Cierra el formulario sin realizar ninguna acción si el usuario pulsa "Cancelar".

## CLASES CON LAS QUE SE CONECTA DIRECTAMENTE

### ConexionBD

Esta clase gestiona la conexión con la base de datos y proporciona los métodos necesarios para operar con los datos.


En este caso, DialogoEditarProducto llama al método `modificarArticulo(int id, String nombre, double precio, int stock)` para actualizar los datos del producto en la base de datos.

### Producto

Representa la entidad producto que se está editando. Se utiliza para precargar los datos actuales en los campos del formulario (nombre, precio y stock), y para obtener su ID al momento de guardar los cambios.

### VistalInicio

Si el diálogo es invocado desde la vista principal (VistalInicio), al confirmar la edición, se llama al método `actualizarTablasInicio()` de esta clase para que refleje los cambios realizados en la tabla de productos.



Editar Producto

### Editar Producto

Nombre:

Precio:

Stock:

## ***DIALOGOELIMINARPRODUCTO***

Esta clase define un cuadro de diálogo emergente que solicita al usuario la confirmación antes de eliminar un producto del inventario. Muestra el nombre del producto seleccionado y ofrece dos opciones: confirmar la eliminación o cancelar la operación. Si se confirma, se ejecuta la eliminación en la base de datos.

## **MÉTODOS DIALOGOELIMINARPRODUCTO**

**DialogoEliminarProducto(JFrame parent, ConexionBD conexion, int productoid, String descripcion)**

Constructor que inicializa el diálogo modal de confirmación para eliminar un producto. Recibe como parámetros la ventana padre, la conexión a la base de datos, el ID del producto a eliminar y su descripción para mostrarla en el mensaje.

**isConfirmado()**

Método que devuelve un valor booleano indicando si el usuario ha confirmado la eliminación (true) o ha cancelado la operación (false). Se utiliza para que la clase que llama al diálogo pueda saber el resultado tras cerrarse.

### **EVENTOS Y ACCIONES**

**btnEliminar (ActionListener)**

Llama al método eliminarArticulo(int id) de la clase ConexionBD y cierra el diálogo.

**btnCancelar (ActionListener)**

Cierra el formulario sin realizar ninguna acción si el usuario pulsa "Cancelar".

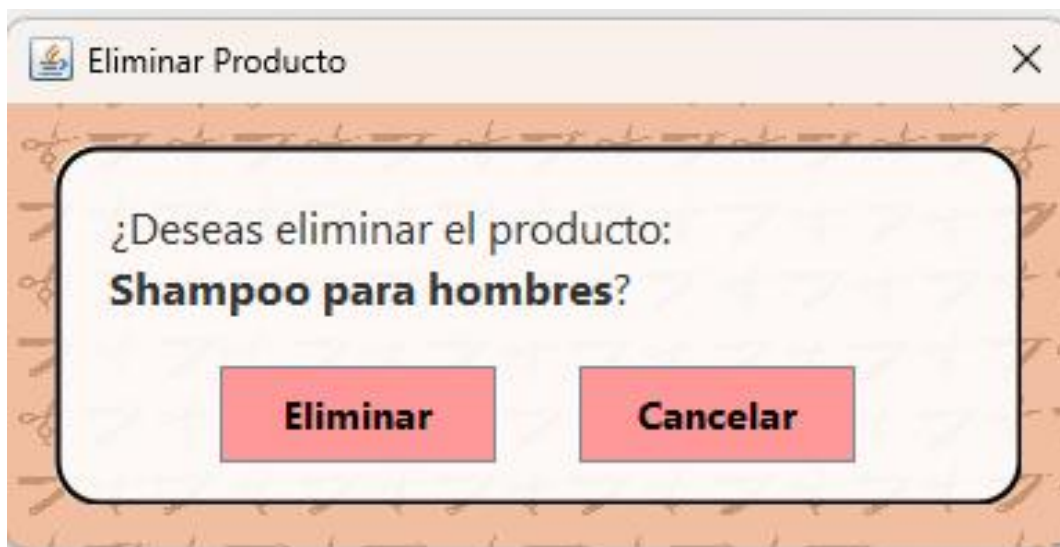
## CLASES CON LAS QUE SE CONECTA DIRECTAMENTE

### ConexionBD

Proporciona el método `eliminarArticulo(int id)` que elimina el producto con el identificador proporcionado de la base de datos.

### VistaLogin.RoundedPanel

Componente visual reutilizado que proporciona un panel con bordes redondeados y fondo translúcido.



# CALENDARIO

Panel personalizado que muestra un calendario mensual con botones para cambiar de mes, días interactivos y la capacidad de seleccionar fechas. Permite la creación de citas mediante doble clic en un día y notifica la fecha seleccionada mediante un listener.

## MÉTODOS CALENDARIO

### **Calendario(JFrame framePadre)**

Constructor que inicializa el panel calendario, configura la fecha actual, crea el encabezado con botones para cambiar de mes y establece el panel de días. Recibe la ventana padre para centrar futuros diálogos.

### **void actualizarCalendario()**

Método privado que refresca el contenido del calendario: limpia el panel de días, añade las etiquetas de los días de la semana, rellena con paneles para cada día del mes y actualiza el título con el mes y año actual.

### **void setFechaSeleccionadaListener(FechaSeleccionadaListener listener)**

Permite registrar un listener externo que será notificado cuando el usuario seleccione una fecha distinta.

### **LocalDate obtieneUltimaFechaSeleccionada()**

Devuelve la última fecha seleccionada en el calendario.

### **private JButton crearBoton(String texto)**

Método auxiliar privado que crea y devuelve un botón estilizado con efectos visuales al pasar el cursor, usado para los botones de navegación de meses.



## EVENTOS Y ACCIONES

### Cambio de mes (Botones < y >) :

Al pulsar el botón <, se ejecuta un ActionListener que resta 1 al mes actual (`mesActual = mesActual.minusMonths(1)`).

Al pulsar el botón >, se ejecuta un ActionListener que suma 1 al mes actual (`mesActual = mesActual.plusMonths(1)`).

En ambos casos, tras modificar el mes, se invoca el método `actualizarCalendario()`, que:

- \* Elimina los paneles de días anteriores.
- \* Calcula el número de días del nuevo mes y su posición en la semana.
- \* Genera nuevos `DiaPanel` para cada día del mes.
- \* Vuelve a dibujar el calendario actualizado en pantalla.

### Selección de fecha:

Cada día visible del calendario es representado por un componente personalizado llamado `DiaPanel`.

Al hacer clic simple (una sola vez con el botón izquierdo) sobre un `DiaPanel`:

- \* Se actualiza la variable `fechaSeleccionada` con la fecha de ese día.
- \* Se llama al método del listener `fechaSeleccionada(fecha)` si ha sido previamente asignado mediante `setFechaSeleccionadaListener(...)`.
- \* Se invoca `repaint()`, lo que:
- \* Re-dibuja el calendario para destacar visualmente la nueva fecha seleccionada.

También actualiza el color del panel para diferenciarlo del resto.

### Crear cita:

Dentro de la clase interna `DiaPanel`, se detectan eventos de ratón con un `MouseAdapter`.

Al hacer doble clic con el botón izquierdo sobre un día (evento `mouseClicked` con `getClickCount() == 2` y `SwingUtilities.isLeftMouseButton(e)`), se ejecuta el método `abrirDialogoCrearCita(LocalDate fecha)`, que:

- \* Crea una nueva instancia del diálogo modal `DialogoCrearCita`.
- \* Este diálogo se abre centrado sobre la ventana principal (`framePadre`) y recibe como parámetro la fecha correspondiente al día pulsado.
- \* Permite al usuario introducir datos de una nueva cita para ese día, que se almacenará posteriormente en la base de datos.

## CLASES INTERNAS

### interface FechaSeleccionadaListener

Interfaz para notificar a objetos externos cuando el usuario selecciona una fecha en el calendario.

### private class DiaPanel extends JPanel

Panel que representa un día específico en el calendario. Gestiona la detección de clics simples y dobles, cambia su apariencia si es la fecha actual o la seleccionada y abre el diálogo para crear citas.

### static class RoundedPanel extends JPanel

Panel con bordes redondeados y color de fondo personalizado usado para el diseño del panel superior y otros elementos.

## CLASES CON LAS QUE SE CONECTA DIRECTAMENTE

### ConexionBD

Para consultar la base de datos (validación de usuario).

### DialogoCrearCita

Se instancia para permitir crear una cita en la fecha seleccionada al hacer doble clic.

### SesionUsuario

Para obtener el ID del usuario activo.

### VistaCitas

Se pasa como referencia en el diálogo de creación de cita para posibles actualizaciones.



# CONEXIÓNBD

Clase encargada de gestionar la conexión con la base de datos y realizar todas las operaciones de lectura y escritura relacionadas con usuarios, artículos, citas y servicios. Centraliza la lógica de acceso a datos en la aplicación TheBarberApp.

Permite registrar usuarios, validar credenciales, gestionar inventario, controlar citas asociadas a usuarios y convertir entre IDs y nombres de servicios.

## MÉTODOS CONEXIÓNBD

### USUARIOS

**boolean validarCredenciales(String usuario, String contrasena)**

Comprueba si existe un usuario con el nombre y contraseña dados en la base de datos.

**int obtenerIdUsuario(String nombreUsuario)**

Devuelve el ID asociado a un nombre de usuario registrado.

**boolean registrarUsuario(String usuario, String contrasena)**

Registra un nuevo usuario si no existe uno con el mismo nombre

### SERVICIOS

**ArrayList<String> obtenerNombresServicios()**

Devuelve los nombres de todos los servicios disponibles.

**int obtenerIdServicioPorNombre(String nombre)**

Devuelve el ID correspondiente a un nombre de servicio.

**String obtenerNombreServicioPorId(int idServicio)**

Devuelve el nombre de un servicio a partir de su ID.

## INVENTARIO

### **ArrayList<Articulo> obtenerArticulos()**

Devuelve todos los artículos registrados.

### **boolean agregarArticulo(Articulo articulo)**

Inserta un nuevo artículo en la base de datos.

### **boolean actualizarArticulo(Articulo articulo)**

Modifica los datos de un artículo existente.

### **boolean eliminarArticulo(int idArticulo)**

Elimina un artículo por su ID.

### **ArrayList<Articulo> obtenerArticulosMenorStock(int limite)**

Devuelve los artículos con menor cantidad en stock (ordenados y limitados).

## CITAS

### **ArrayList<Cita> obtenerCitasPorUsuario(int idUsuario)**

Devuelve todas las citas asociadas a un usuario específico.

### **ArrayList<Cita> obtenerCitasPorFecha(int idUsuario, LocalDate fecha)**

Devuelve las citas de un usuario para un día determinado.

### **boolean agregarCita(Cita cita)**

Inserta una nueva cita en la base de datos.

### **boolean actualizarCita(Cita cita)**

Modifica una cita existente.

### **boolean eliminarCita(int idCita)**

Elimina una cita por su ID.

## EVENTOS Y ACCIONES

Esta clase no gestiona directamente eventos de la interfaz, pero responde a acciones del usuario ejecutadas en otras vistas (como registrar, iniciar sesión, añadir una cita, etc.) devolviendo los datos necesarios o ejecutando las operaciones requeridas sobre la base de datos.

## CLASES CON LAS QUE SE CONECTA DIRECTAMENTE

### **SesionUsuario**

Usa los métodos de validación y obtención de ID de usuario para controlar la sesión actual.

### **VistaLogin, VistaRegistro**

Validan o registran usuarios usando los métodos correspondientes de esta clase.

### **VistaInventario**

Obtiene, inserta, actualiza y elimina artículos mediante los métodos de inventario.

### **VistaCitas, VistaInicio, Calendario**

Llaman a los métodos relacionados con las citas para mostrar, crear o modificar registros del usuario activo.

### **DialogoCrearCita, DialogoEditarCita, DialogoEliminarCita**

Se conectan para gestionar el alta, modificación y eliminación de citas, así como para traducir nombres/IDs de servicios.

### **DialogoAgregarProducto**

Utiliza agregarArticulo(...) para insertar un nuevo artículo en la base de datos.

### **DialogoEditarProducto**

Utiliza modificarArticulo(...) para actualizar artículos seleccionados.

### **DialogoEliminarProducto**

Llama a eliminarArticuloPorId(...) para eliminar un artículo del inventario.

# ROUNDEDCELLRENDERER

Renderizador personalizado de celdas para tablas (JTable) que aplica bordes redondeados y estilos visuales tanto para celdas normales como para encabezados. Mejora la estética de las tablas al pintar manualmente el fondo con colores personalizados y suavizado gráfico.

## ***MÉTODOS ROUNDEDCELLRENDERER***

### **RoundedCellRenderer(Color backgroundColor, boolean isHeader)**

Constructor que inicializa el renderizador con un color de fondo específico y una indicación de si se usará en celdas de encabezado. Establece la opacidad a false para permitir un repintado manual.

### **Component getTableCellRendererComponent(...)**

Sobrescribe el método de DefaultTableCellRenderer para personalizar el contenido de la celda. Si es un encabezado, cambia la fuente a negrita y el color del texto a blanco, si no es encabezado, utiliza fuente normal y adapta el color del texto según si la celda está seleccionada o no.

### **void paintComponent(Graphics g)**

Sobrescribe el método de pintado del componente para dibujar un fondo con bordes redondeados y aplica suavizado (antialiasing). También dibuja un rectángulo redondeado con el color de fondo indicado.

Cambia el color si la fila está seleccionada.

Luego llama a super.paintComponent(g) para dibujar el contenido del texto.

## **EVENTOS Y ACCIONES**

### **Renderizado de celdas**

Cada vez que una celda de la tabla necesita ser pintada, se ejecuta este renderizador.

### **Selección de filas**

Si la fila es seleccionada, el color de fondo cambia para resaltarla visualmente.

### **Detección de encabezados**

Si se indica que es un encabezado (isHeader = true), se ajustan fuente y color de texto apropiadamente.

## **CLASES CON LAS QUE SE CONECTA DIRECTAMENTE**

### **JTable**

Utiliza JTable como destino del renderizado para aplicar estilos por celda.

### **SwingUtilities**

Para detectar el componente JTable al que pertenece la celda durante el renderizado.

# SESIÓNUSUARIO

Clase auxiliar estática que mantiene la información de la sesión del usuario activo durante la ejecución de la aplicación. Permite acceder globalmente al nombre y al ID del usuario autenticado, así como cerrar la sesión de forma centralizada.

## ***MÉTODOS SESIONUSUARIO***

### **public static String nombreUsuario**

Campo estático que almacena el nombre del usuario actualmente autenticado. Puede ser null si no hay sesión activa.

### **public static int usuariold**

Campo estático que almacena el ID del usuario autenticado, obtenido al iniciar sesión. Por defecto vale -1 cuando no hay sesión activa.

### **public static void cerrarSesion()**

Método estático que limpia los datos de la sesión. Restablece nombreUsuario a null y usuariold a -1, indicando que ya no hay usuario activo.

## **EVENTOS Y ACCIONES**

### **Inicio de sesión exitoso**

Al autenticarse un usuario, otras clases (como VistaLogin) asignan el nombre y el ID del usuario en estos campos para mantener el estado de sesión.

### **Cierre de sesión**

Se llama a cerrarSesion() al cerrar la aplicación o al cerrar sesión explícitamente, para borrar los datos del usuario activo.



## **CLASES CON LAS QUE SE CONECTA DIRECTAMENTE**

### **VistaInicio, VistaCitas,VistaLogin**

Establece los valores de nombreUsuario y usuariold al autenticar correctamente al usuario, iniciando así la sesión. También consultan SesionUsuario.usuariold para mostrar únicamente los datos asociados al usuario activo, como citas o artículos relevantes.

### **DialogoCrearCita, DialogoEditarCita, DialogoEliminarCita**

Utilizan usuariold para asociar, validar o restringir la edición y eliminación de citas pertenecientes al usuario autenticado.

### **ConexionBD**

Accede a SesionUsuario.usuariold para ejecutar operaciones SQL filtradas según el contexto de la sesión actual del usuario.

# CONTROLADOR

La clase Controlador actúa como coordinador entre las distintas vistas principales de la aplicación (VistaInicio, VistaCitas, VistaInventario). Centraliza la gestión de eventos de navegación del menú lateral y controla el cambio de vistas en el CardLayout de VistaInicio.

## ***MÉTODOS CONTROLADOR***

### **Controlador(VistaInicio vi, VistaCitas vc, VistaInventario vi2)**

Constructor que inicializa las referencias a las vistas y registra los ActionListener en los botones del menú. Define la vista inicial como "INICIO".

### **actionPerformed(ActionEvent e)**

Responde a los eventos de los botones del menú. Cambia de vista según el botón pulsado (Inicio, Citas o Inventario) y actualiza la interfaz visualmente.

### **resetMenuButtons()**

Restaura los botones del menú a su estilo por defecto (sin borde ni negrita).

### **setSelectedMenu(String menu)**

Resalta visualmente el botón correspondiente al menú activo, aplicando un borde blanco inferior y negrita, y actualiza el título del JFrame principal.

## **EVENTOS Y ACCIONES**

### **Navegación entre vistas mediante los botones del menú lateral**

El Controlador gestiona los eventos de todos los botones de navegación presentes en las tres vistas (VistaInicio, VistaCitas y VistaInventario). Cuando el usuario hace clic en cualquiera de estos botones, la clase identifica la acción y cambia la vista activa en el CardLayout de VistaInicio, permitiendo una transición fluida entre las secciones de la aplicación (Inicio, Citas e Inventario).

### **Actualización visual del botón de menú seleccionado**

Cada vez que se selecciona una vista, el botón correspondiente del menú se destaca visualmente con un borde inferior blanco de 8 píxeles y un cambio en la tipografía a negrita. Este efecto se aplica de forma consistente en todas las vistas para mantener una retroalimentación visual clara del estado actual de navegación, mejorando la usabilidad.

### **Reflejo del menú activo en el título de la ventana**

El título de la ventana principal (JFrame) se actualiza dinámicamente para mostrar la sección activa de la aplicación, por ejemplo: "The Barber Shop - CITAS". Esto proporciona una referencia visual adicional al usuario sobre su ubicación actual dentro de la aplicación, favoreciendo la orientación y la coherencia.

### **CLASES CON LAS QUE SE CONECTA DIRECTAMENTE**

#### **VistaInicio**

Contenedor principal con CardLayout. Gestiona los cambios de vista y visualiza el menú lateral.

#### **VistaCitas**

Panel con la gestión de citas. Proporciona botones de navegación que el Controlador escucha.

#### **VistaInventario**

Panel de gestión de artículos. También contiene botones de menú gestionados por esta clase.

# CLASES POJO

## *PRODUCTO*

La clase Producto representa un artículo disponible en el inventario de la barbería, como productos capilares o utensilios. Es una estructura de datos que encapsula la información básica asociada a cada producto y permite su manipulación en operaciones CRUD (crear, leer, actualizar, eliminar).

## MÉTODOS PRODUCTO

**Producto(int id, String nombre, double precio, int stock)**

Constructor completo que se usa cuando ya se conoce el ID del producto (por ejemplo, al recuperar datos de la base de datos).

**Producto(String nombre, double precio, int stock)**

Constructor parcial que asume que el ID aún no está asignado. Ideal para crear nuevos productos antes de insertarlos en la base de datos.

### Getters y Setters

Proporcionan acceso controlado a todos los atributos privados de la clase (getId(), setId(), getNombre(), etc.), garantizando encapsulamiento y flexibilidad.

## CLASES CON LAS QUE SE CONECTA DIRECTAMENTE

**VistaInventario y sus diálogos (agregar, editar, eliminar)**

Utilizan instancias de Producto para mostrar, crear o actualizar productos en la interfaz gráfica.

### ConexionBD

Interactúa con la base de datos para convertir registros en objetos Producto y viceversa.

## ***CITA***

La clase Cita representa una reserva realizada por un cliente en la barbería. Incluye toda la información necesaria para gestionar una cita: cliente, servicio, fecha, hora y el usuario que la creó. Este modelo se utiliza para cargar, mostrar, filtrar y almacenar citas en la base de datos.

## **METODOS CITA**

**public Cita(int id, String clienteNombre, int servicioid, String servicioNombre, String fecha, String hora, int usuarioid)**

Recibe todos los parámetros necesarios para construir una cita completa, incluyendo los campos redundantes como el nombre del servicio (útil para mostrar en tablas o formularios sin necesidad de hacer nuevas consultas).

### **Getters**

Proporcionan acceso controlado a todos los atributos privados de la clase (getId(), setId(), getClienteNombre(), etc.), garantizando encapsulamiento y flexibilidad.

No se incluyen setters, lo cual refuerza la inmutabilidad de las citas tras su creación.

## **CLASES CON LAS QUE SE CONECTA DIRECTAMENTE**

### **VistaCitas y sus diálogos (agregar, editar, eliminar)**

Utilizan objetos Cita para mostrar los datos en la interfaz gráfica, así como para transferirlos entre los distintos componentes de la aplicación.

### **ConexionBD**

Gestiona las operaciones de base de datos: inserción, modificación, eliminación y consulta de citas.

# BARBERAPP (MAIN)

La clase TheBarberApp es el punto de inicio de la aplicación de escritorio. Contiene el método main, encargado de lanzar la interfaz gráfica principal (VistalInicio) de forma segura en el hilo de eventos de Swing.

## ***MÉTODOS BARBERAPP***

### **public static void main(String[] args)**

Utiliza SwingUtilities.invokeLater(...) para asegurar que la interfaz gráfica se construya y manipule en el Event Dispatch Thread (EDT), como recomienda la especificación de Swing.

Crea una instancia de VistalInicio, que es el JFrame principal de la aplicación y contiene la lógica para cambiar entre vistas (login, inicio, citas, inventario, etc.).

Muestra la ventana principal de la aplicación invocando setVisible(true) sobre vistalInicio.

## **CLASES CON LAS QUE SE CONECTA DIRECTAMENTE**

### **VistalInicio**

Utilizan objetos Cita para mostrar los datos en la interfaz gráfica, así como para transferirlos entre los distintos componentes de la aplicación.

### **Controlador**

Gestiona los eventos de los botones del menú y controla la navegación entre vistas.

### **ConexionBD**

Gestiona las operaciones de base de datos: inserción, modificación, eliminación y consulta de citas.

# FUNCIONAMIENTO DE LA APP

Al iniciar la aplicación lo primero que nos encontraremos será la pantalla de login, desde esta podemos iniciar sesión o registrarnos haciendo click en el enlace, lo cual nos llevará a la ventana de registro para crear un nuevo usuario que no sea Administrador, ya que Admin solo tenemos este.

Para iniciar sesión se nos pide el nombre de usuario y contraseña, mientras que para registrarnos también se nos pide el correo.

Una vez dentro de la aplicación, la primera ventana a la que tendremos acceso será “INIICIO”, en la cual tendremos dos tablas, a la izquierda tenemos la tabla relacionada con el inventario, la cual es visible para todos los usuarios a modo de recordatorio de los productos que se están agotando o hay que reponer.

En la derecha, nos saldrán las citas que tengamos agendadas con este usuario en el día de hoy.

The screenshot shows the 'The Barber Shop - INICIO' app interface. At the top, there is a navigation bar with three tabs: 'INICIO', 'CITAS', and 'INVENTARIO'. The 'INICIO' tab is selected. Below the navigation bar, the title 'THE BARBER SHOP' is displayed. On the right side of the header, there is a greeting 'Bienvenido, Adm...' and a menu icon. The main content area is divided into two sections. The left section is titled 'ARTÍCULOS CON MENOR STOCK' and contains a table with two columns: 'Artículo' and 'Stock'. The right section is titled 'CITAS DEL DÍA' and contains a table with three columns: 'Cliente', 'Servicio', and 'Hora'. The background of the app has a subtle pattern of hair clippers.

Artículo	Stock
Máquina de corte premium	5
Máquina de corte básica	10
Gel exfoliante	12
Tijeras de entresacar	12
Serum facial	14
Espejo de mano	15
Loción facial	15
Crema hidratante	16
Aceite para barba	18
Tijeras de corte	18
Aceite esencial para masaje	19
Pomada mate	20
Capa de corte	20
Mascarilla facial	20
Crema para masajes	21
Bálsamo para barba	22
Brocha de afeitarse	22
Limpiador de máquinas	23

Cliente	Servicio	Hora
Prueba	Corte	11:30:00

Para cambiar entre las vistas tenemos la barra de menú arriba para cambiar las pestañas.

IMPORTANTE: La pestaña Inventario estará disponible o no dependiendo de si el usuario es Admin o no, esto se identifica según “tipousuario”.

#### VISTA ADMIN

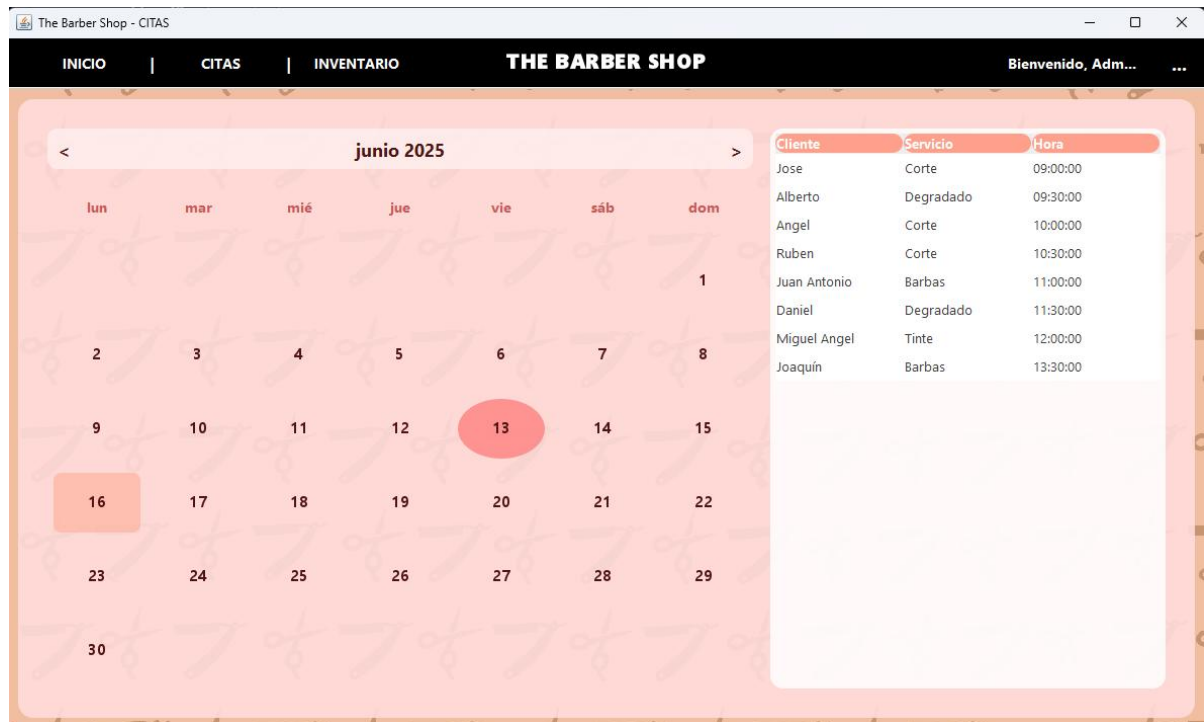


#### VISTA USUARIO ESTANDAR





Si hacemos click en el botón de navegación de "CITAS", nos abrirá la ventana de gestión de las citas, la cual incluye una tabla en la cual consultamos las citas por el día marcado en el calendario y el usuario activo.



Las acciones de esta clase son:

#### **Doble Click en un día del calendario**

Se ejecuta el "DialogoCrearCita", donde podremos rellenar la información de la nueva cita.

**Crear Cita**

**Nueva Cita**

Fecha: 16/06/2025

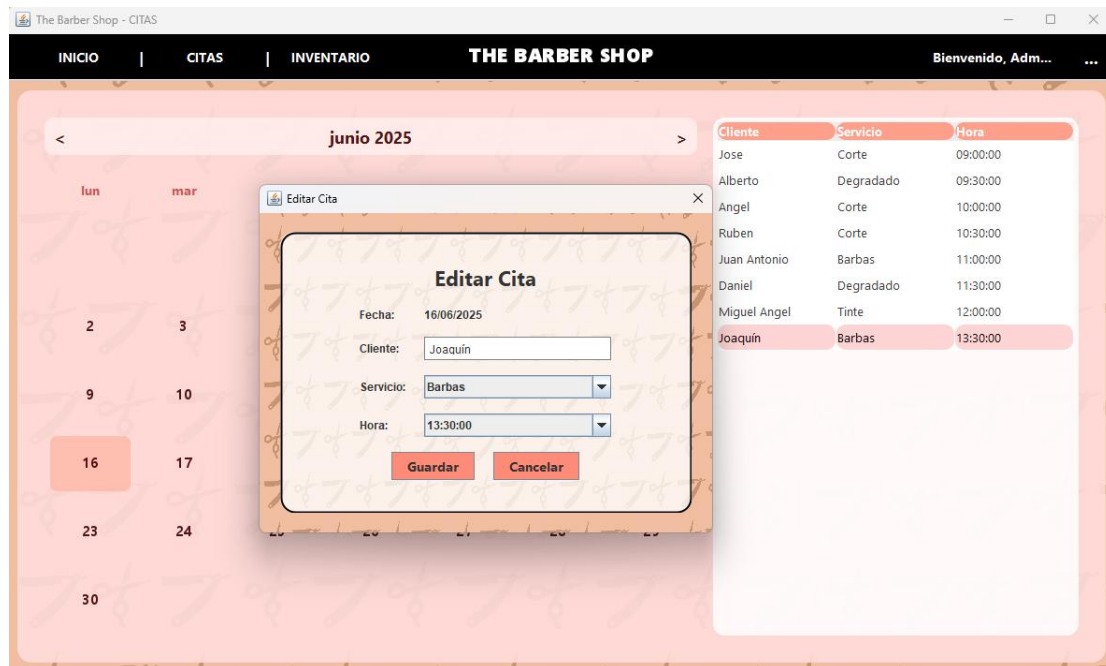
Cliente:

Servicio:

Hora:

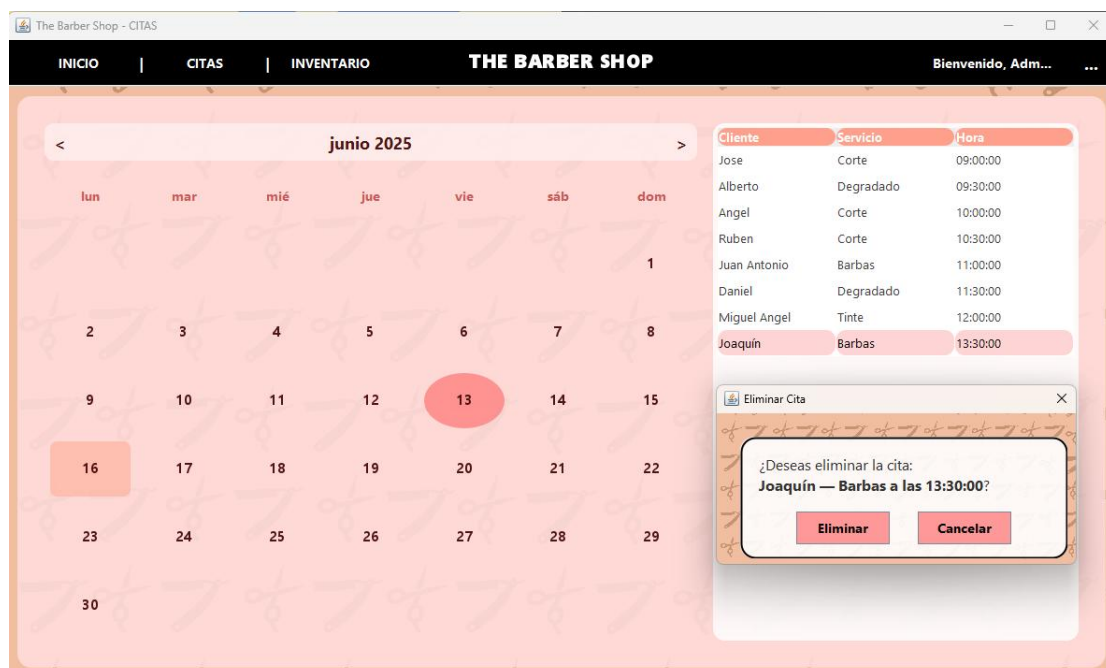
### Doble click en una cita de la tabla

Al hacer doble click en una fila de la tabla que contiene las citas del día seleccionado, se nos abre “DialogoEditarCita” con la información de la cita seleccionada precargada y lista para editarse.

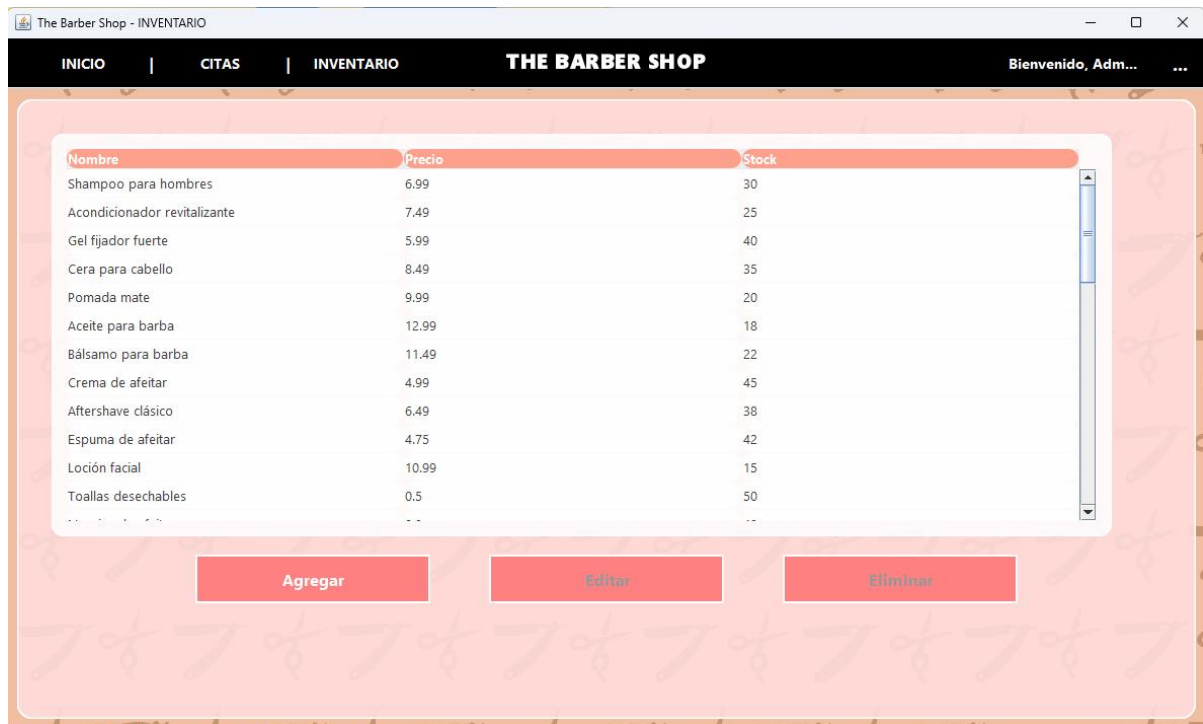


### Mantener click sobre una cita de la tabla

Esta acción llama a “DialogoEliminarCita” el cual nos pide confirmación para eliminar la cita seleccionada.



Si hacemos click en el botón de navegación de "INVENTARIO", nos abrirá la ventana de gestión del inventario, al cual solo tendrá acceso el/los usuarios administradores (tipousuario == 1)



Nombre	Precio	Stock
Shampoo para hombres	6.99	30
Acondicionador revitalizante	7.49	25
Gel fijador fuerte	5.99	40
Cera para cabello	8.49	35
Pomada mate	9.99	20
Aceite para barba	12.99	18
Bálsamo para barba	11.49	22
Crema de afeitar	4.99	45
Aftershave clásico	6.49	38
Espuma de afeitar	4.75	42
Loción facial	10.99	15
Toallas desechables	0.5	50
--	--	--

**Agregar** **Editar** **Eliminar**

Tendremos acceso a una tabla con todos los productos que vendemos o necesitamos comprar para trabajar, debajo de esta, tenemos 3 botones.

### Agregar

Añade un producto a través de "DialogoAñadirProducto" una vez metamos los datos de este. Si seleccionamos un producto en la tabla, se habilitan los demás botones.



**Agregar Producto**

**Nuevo Producto**

Nombre:

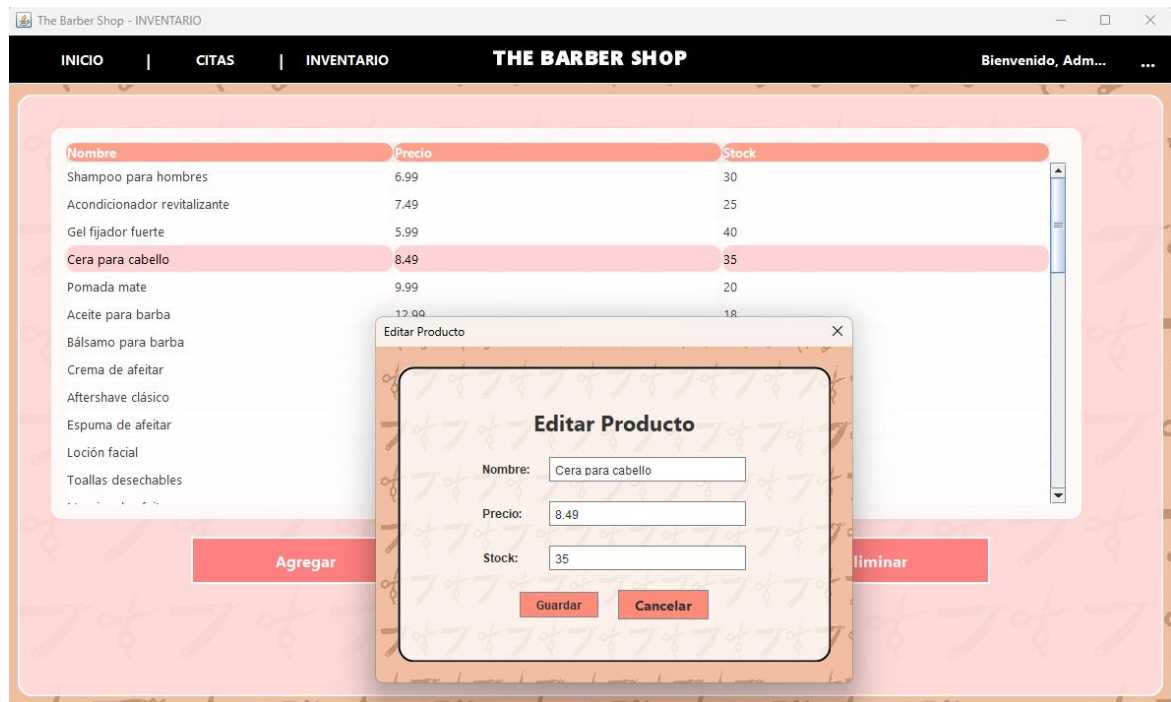
Precio:

Stock:

**Guardar** **Cancelar**

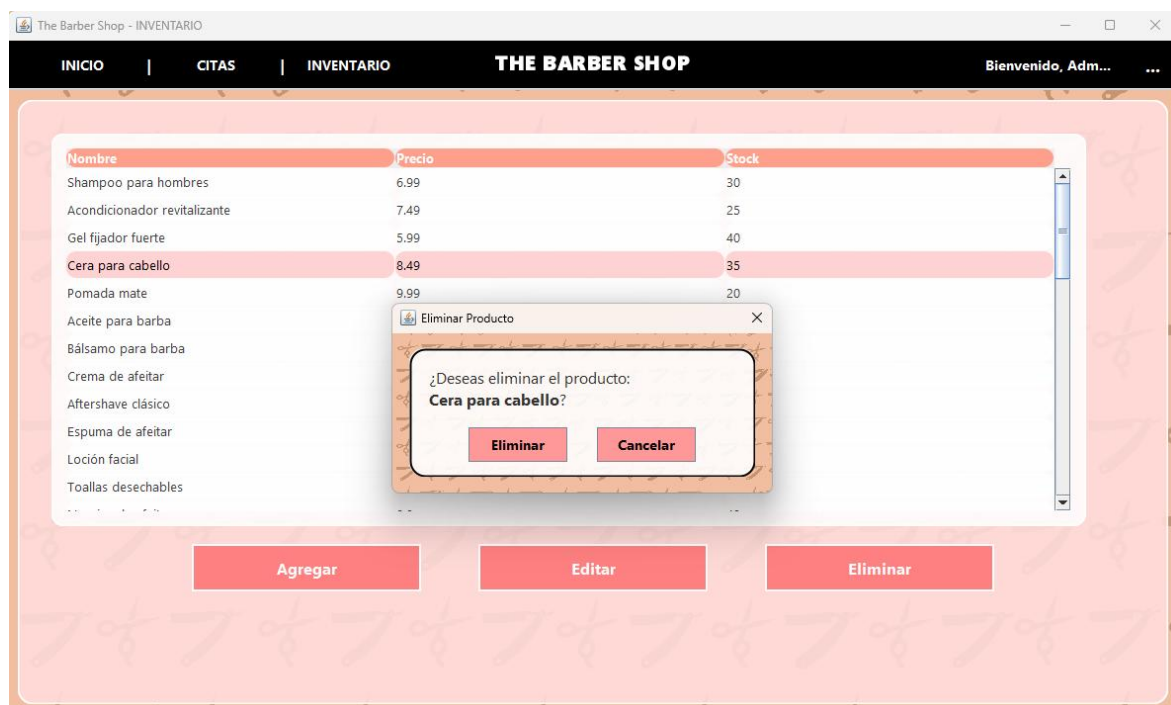
## Editar

Requiere de tener un producto seleccionado en la tabla, llama a “DialogoEditarProducto”, carga la información del producto seleccionado en los campos y permite su edición.



## Eliminar

Requiere de tener un producto seleccionado en la tabla, llama a “DialogoEliminarProducto”, carga la información del producto seleccionado y te pide confirmar su eliminación.



# CONTROL DE ERRORES

El control de errores es una parte fundamental de nuestra aplicación, se encarga de prever que el usuario haga entradas erróneas en los campos con los que pueda interactuar, avisando al usuario de esto e impidiendo errores de datos en la BD.

También es usado para comprobar a nivel de código que parte de este puede fallar, y que tipo de error es, aunque nos centraremos mas en la interacción con el usuario.

Las clases donde la usamos son:

- \* DialogoCrearCita
- \* DialogoEditarCita
- \* DialogoAgregarProducto
- \* DialogoEditarProducto
- \* VistaLogin
- \* VistaRegistro

En esta el usuario tiene campos de texto donde introducir información que necesita tener un formato uniforme para la base de datos, por lo tanto, debemos encargarnos de que este solo pueda meter un tipo de dato esperado.

## VISTALOGIN

En esta clase, se nos pide un usuario y una contraseña, si el usuario y la contraseña se encuentran en la bd pasamos a VistaInicio.

Si los campos están en blanco, o los datos no son correctos:



**Iniciar sesión**

Usuario:

Contraseña:

☒ Recordar Usuario

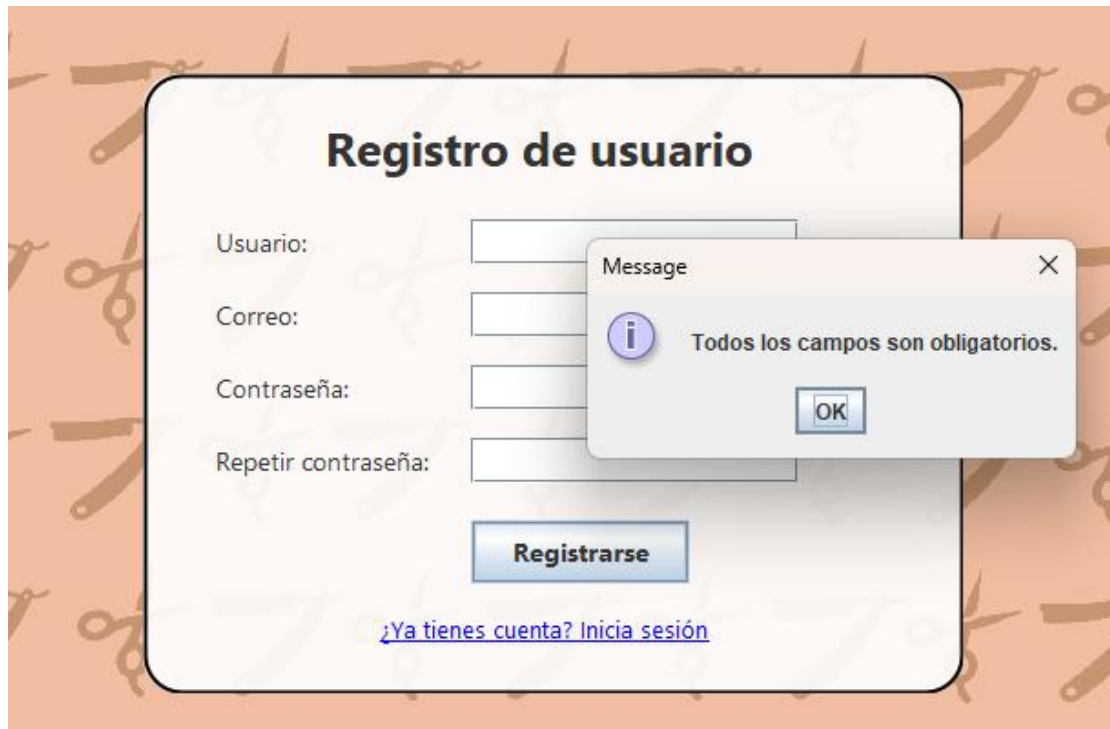
**Entrar**

**Usuario o contraseña incorrecto**

[¿No tienes cuenta? ¡Regístrate ahora!](#)

## VISTAREGISTRO

En esta clase, se nos pide un usuario, un correo, una contraseña y una confirmación de la contraseña, si todos o algún campo esta en blanco:



The screenshot shows a registration form titled "Registro de usuario" on a light orange background with a pattern of hairdressing tools. The form has four input fields: "Usuario:", "Correo:", "Contraseña:", and "Repetir contraseña:". Below these fields is a blue "Registrarse" button. At the bottom of the form is a link: [¿Ya tienes cuenta? Inicia sesión](#). A modal message box is overlaid on the form, displaying the text "Message" at the top, an information icon, and the message "Todos los campos son obligatorios." with an "OK" button.

Si el usuario o el correo ya existe:



The screenshot shows the same registration form as before, but with the "Usuario:" field filled with "Prueba2" and the "Correo:" field filled with "a@gmail.com". The "Contraseña:" and "Repetir contraseña:" fields are filled with four dots. The "Registrarse" button is still visible. A modal message box is overlaid, displaying the text "Message" at the top, an information icon, and the message "El usuario o correo ya está registrado." with an "OK" button.



Si las contraseñas no coinciden:



The image shows a web form titled "Registro de usuario" (User Registration) on a light orange background with a pattern of hairdressing tools. The form has four input fields: "Usuario:" with the value "Prueba3", "Correo:" with the value "aa@gmail.com", "Contraseña:" with four dots, and "Repetir contraseña:" with five dots. Below the fields is a blue "Registrarse" button. At the bottom of the form is a link: [¿Ya tienes cuenta? Inicia sesión](#). A modal message box titled "Message" is overlaid on the right side of the form. It contains an information icon, the text "Las contraseñas no coinciden." (Passwords do not match.), and an "OK" button.

**Registro de usuario**

Usuario:

Correo:

Contraseña:

Repetir contraseña:

**Registrarse**

[¿Ya tienes cuenta? Inicia sesión](#)

**Message**

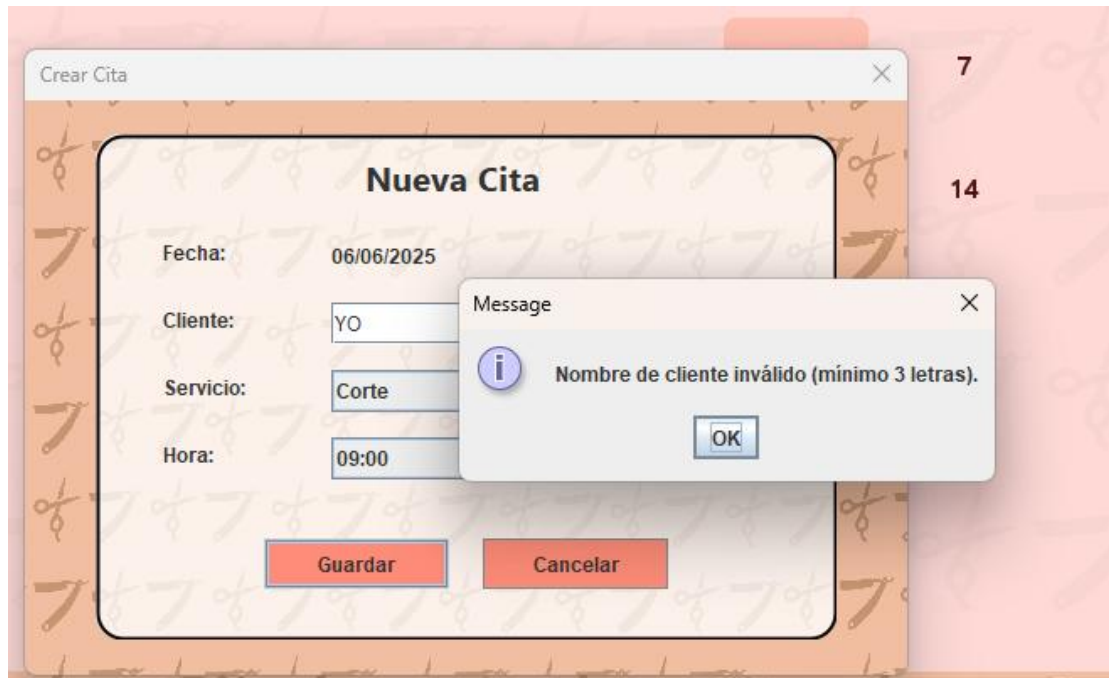
**i** Las contraseñas no coinciden.

**OK**

## ***DIALOGOCREARCITA***

En esta clase el usuario puede interactuar con 3 elementos, 2 Combobox y un JTextField.

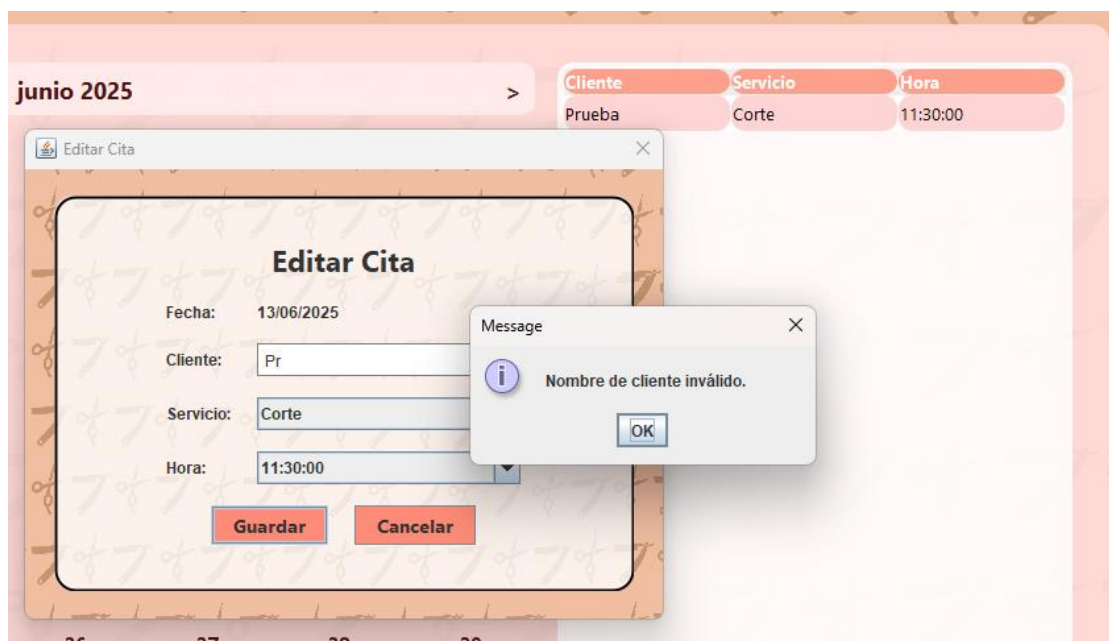
Si en el JTextField “Cliente:” introducimos menos de 3 letras:



## ***DIALOGOEDITARCITA***

En esta clase, al igual que en la anterior, el usuario puede interactuar con 3 elementos, 2 Combobox y un JTextField.

Si en el JTextField “Cliente:” introducimos menos de 3 letras:





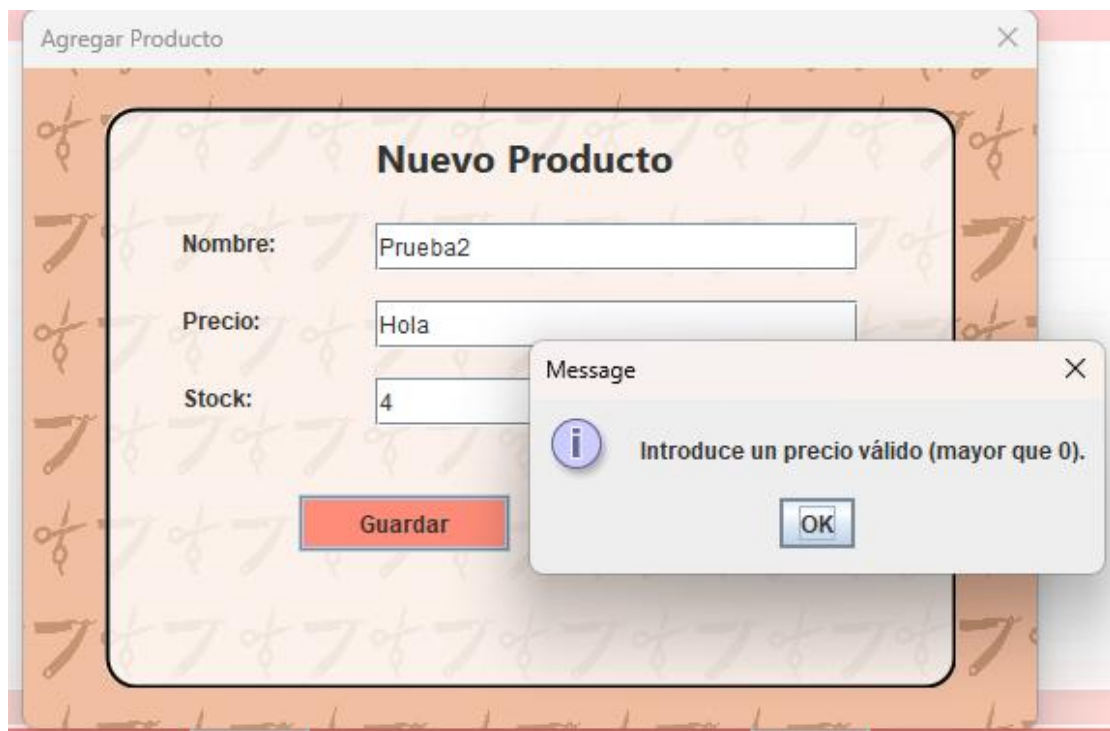
## ***DIALOGOAGREGARPRODUCTO***

En esta clase el usuario puede interactuar con 4 JtextField.

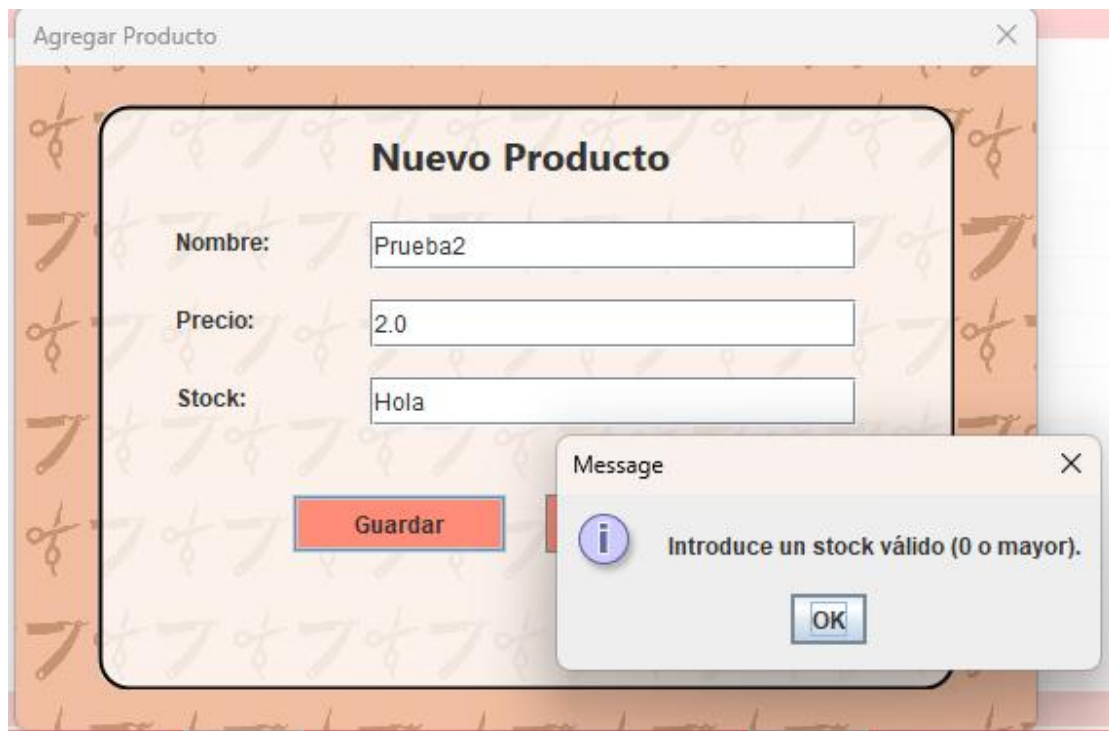
Si el nombre del producto está vacío o tiene menos de dos letras:



Si el precio no es un numero:



Si el Stock no es un numero:



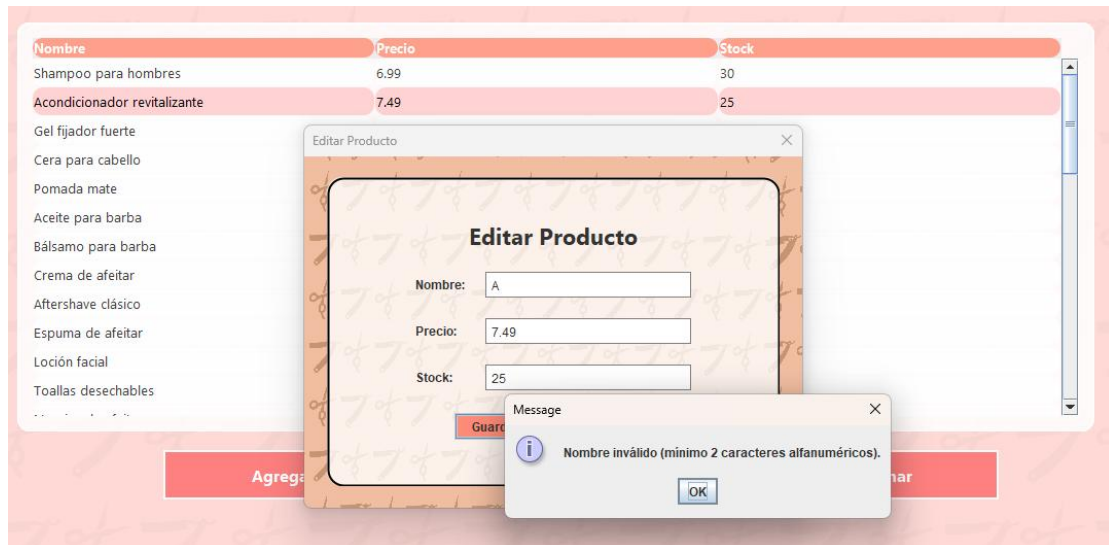
Si el Stock es decimal:



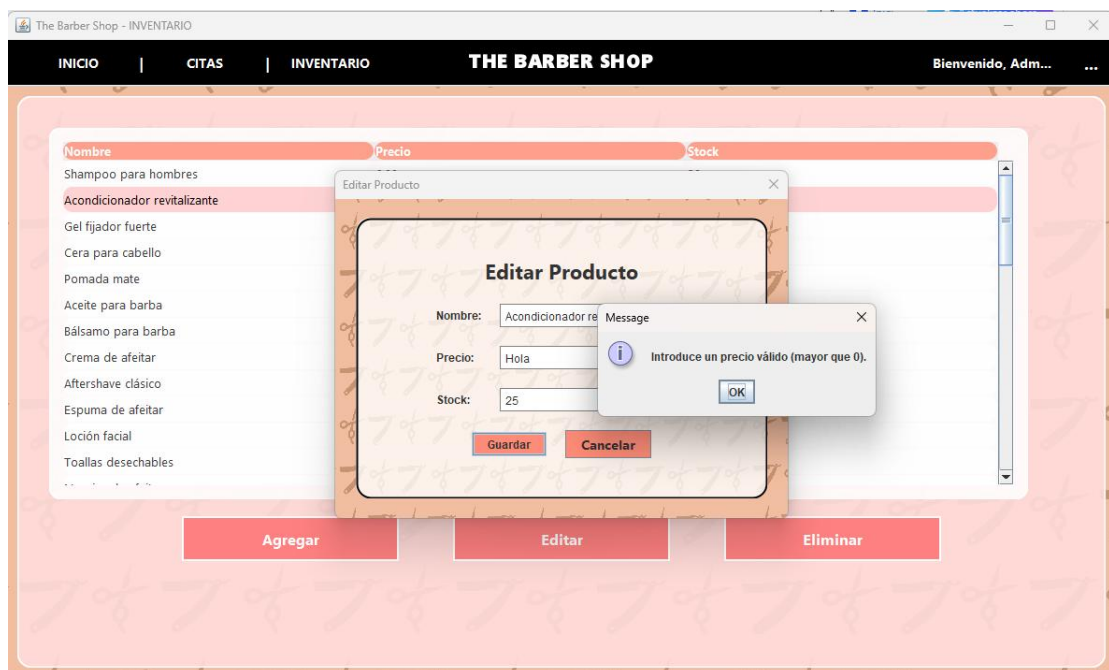
## DIALOGO EDITAR PRODUCTO

En esta clase el usuario puede interactuar con 4 JTextField.

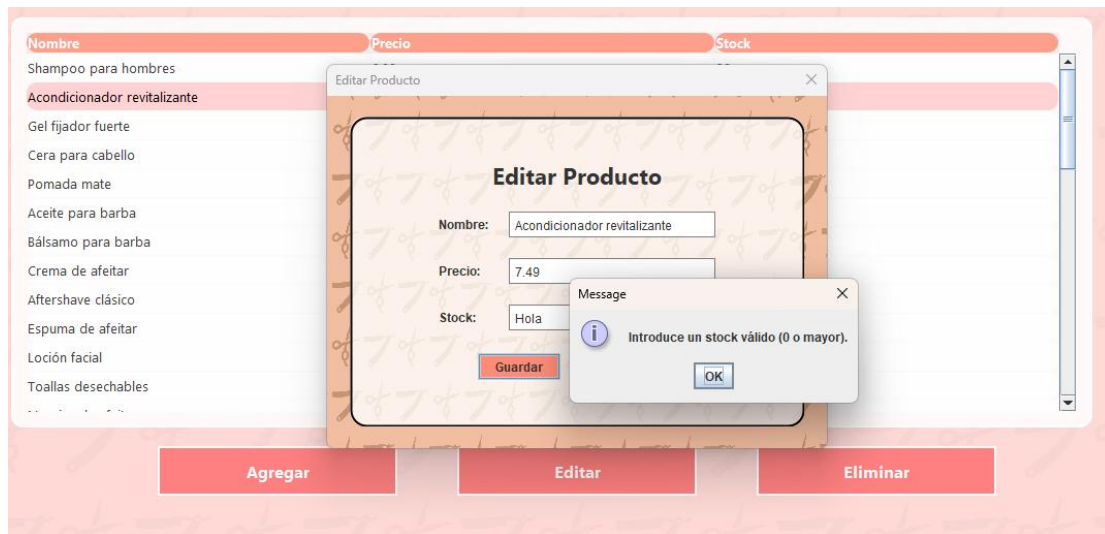
Si el nombre del producto está vacío o tiene menos de dos letras:



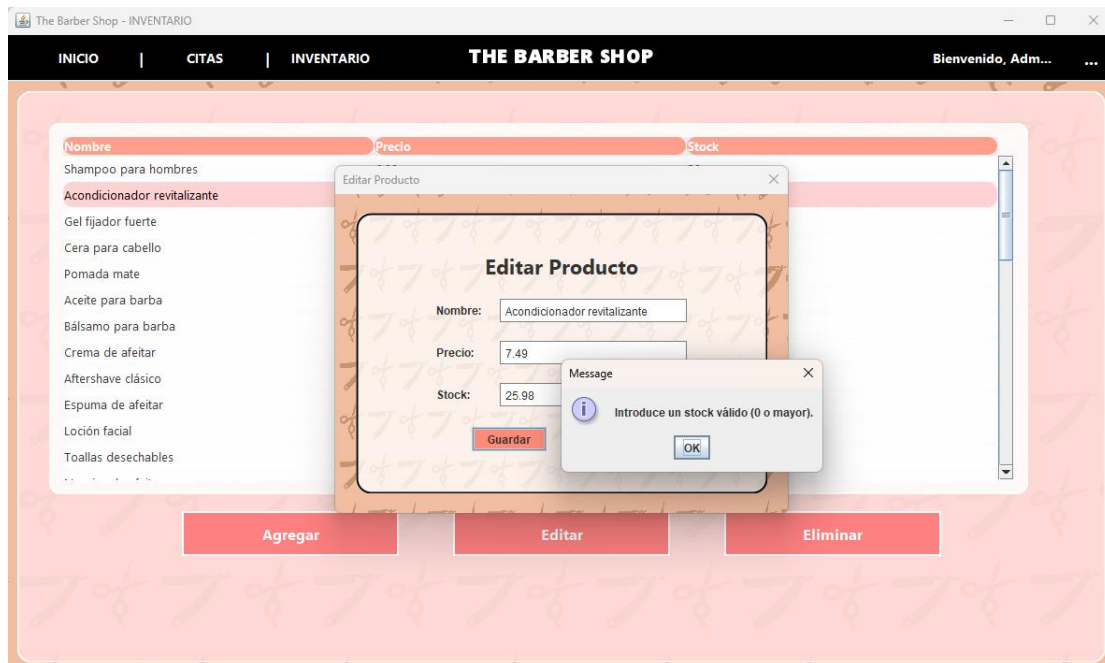
Si el precio no es un numero:



Si el Stock no es un numero:



Si el Stock es decimal:



# CONCLUSIÓN

Crear TheBarberApp ha sido un camino largo, con muchas horas delante del ordenador, momentos de frustración, pero también de orgullo. No ha sido solo un trabajo para entregar, sino un proyecto que antes de empezar bromeaba con hacer para mi cuñado y su peluquería, en el que he puesto todo lo que he ido aprendiendo a lo largo de estos dos años.

He tenido que enfrentar errores, cambiar cosas que pensaba que ya estaban bien, y seguir mejorando poco a poco. También he tenido problemas con el control de versiones ya que no tenía actualizada la copia de seguridad y tuve que rehacer el proyecto en poco tiempo, pero gracias a eso he aprendido muchísimo, no solo sobre programación, sino también sobre cómo organizarme, tener paciencia y no rendirme cuando algo no sale a la primera.

Este proyecto me ha demostrado que soy capaz de sacar adelante algo completo por mí mismo, y eso me da confianza para futuros retos.

# GITHUB

[PROYECTO - THEBARBERAPP](#)