

**Sistema Automatizado de Agencia de viajes**

**Manual de Técnico**

# **Manual Técnico**

Producto: **Sistema Automatizado de Agencia de viajes**

## **1 OBJETIVOS Y ALCANCES DEL SISTEMA**

A través del desarrollo del sistema de agencia de viaje se pretende automatizar los procesos con respecto a la planificación de vuelos tanto comercial como corporativo, de esta forma el sistema se hará cargo del análisis de los diversos factores que influyen al momento de realizar un paquete de viajes como lo es: tiempo, calidad y precio para facilitar la planificación de los vuelos.

## **2 HERRAMIENTAS UTILIZADAS**

### **Arquitectura:**

Las tecnologías, plataformas y herramientas que se muestran en el esquema de implementación serán instaladas y ejecutadas desde un servidor dedicado.

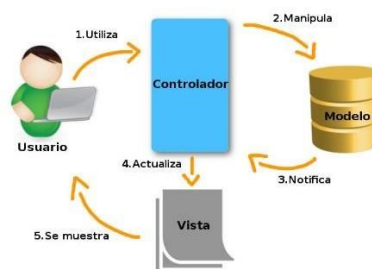
#### **1. NetBeans (Java):**

1.1. NetBeans incluye un editor de código que admite resaltado de sintaxis, autocompletado y refactoring de código. También incluye un depurador, que permite a los desarrolladores encontrar y corregir errores en el código de manera efectiva. Además, NetBeans proporciona herramientas para la gestión de proyectos, la integración de sistemas de control de versiones, y la creación de interfaces gráficas de usuario (GUI, por sus siglas en inglés) para aplicaciones Java.

#### **2. SQLDeveloper**

2.1. SQL Developer permite a los desarrolladores crear, editar y depurar scripts SQL y PL/SQL, así como también permite la creación de objetos de base de datos, como tablas, vistas, procedimientos almacenados y funciones. También proporciona una interfaz de usuario gráfica para administrar bases de datos, lo que facilita la gestión de usuarios, roles, permisos, copias de seguridad y restauraciones de bases de datos.

2.2. SQL Developer es compatible con múltiples sistemas operativos, incluyendo Windows, macOS y Linux, y se puede utilizar tanto como una aplicación independiente como una extensión para el entorno de desarrollo integrado (IDE) de Oracle, JDeveloper.

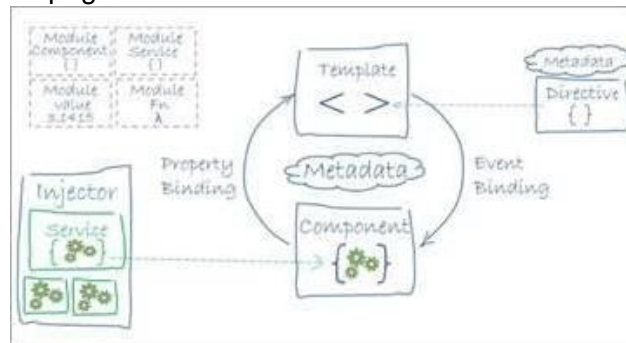


Como se puede observar en la ilustración anterior, se consideran 3 ambientes para poder concretar de forma satisfactoria el desarrollo, así como el control de cambios y/o modificaciones que surjan durante su ejecución. A continuación, se detalla cada ambiente:

1. Ambiente de Desarrollo: Será el entorno inicial en donde se codificará y probarán las funciones requeridas para cada fase, así como la resolución de problemas que puedan surgir durante el desarrollo inicial de las fases.
2. Ambiente de Pre-producción: Este ambiente fue pensado para poder trasladar todas aquellas funcionalidades que sean completadas de forma parcial y que pueden probarse con el cliente para validar que cumpla con sus expectativas.
3. Ambiente de Producción: En este entorno se trasladarán todas aquellas funcionalidades y servicios que hayan sido probadas con el cliente y validadas para poder ser puestas en producción, a este entorno deberá llegar todo aquel código y/o módulo que pasó las pruebas en los dos entornos descritos anteriormente.

#### Capa Vista:

Es la parte de una web o aplicación que conecta e interactúa con los usuarios que la utilizan. Es la parte visible, la que muestra el diseño, los contenidos y la que permite a los visitantes navegar por las diferentes páginas conforme lo deseen.



#### Capa de Control:

- **Capa Modelo:**

Imagen del modelo utilizado



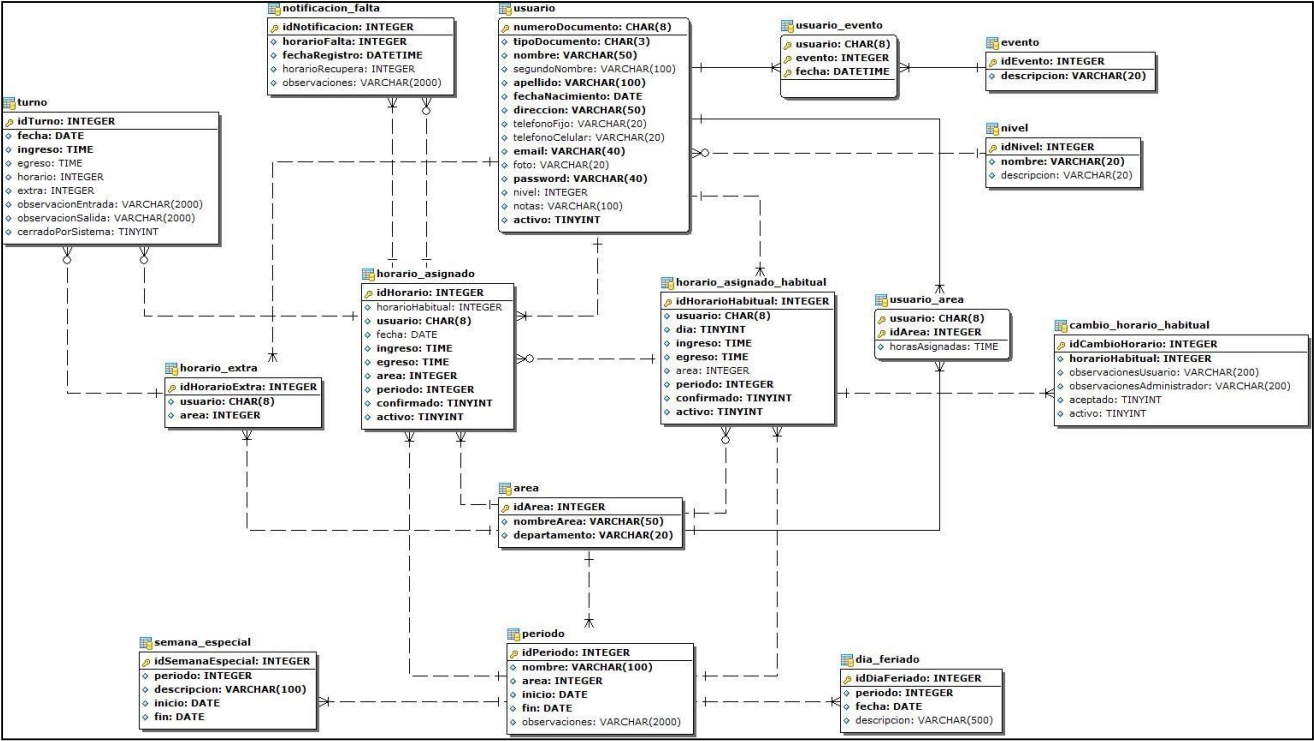
## Servidor

### SQLDeveloper

- SQL Developer es una herramienta de desarrollo de bases de datos de Oracle que se utiliza para crear, administrar y mantener bases de datos relacionales. La herramienta es gratuita y está disponible para su descarga en el sitio web de Oracle.
- SQL Developer proporciona una interfaz gráfica de usuario (GUI) para realizar tareas de administración de bases de datos como crear, modificar y eliminar objetos de base de datos, como tablas, vistas, procedimientos almacenados, funciones, etc. También ofrece la capacidad de escribir y ejecutar consultas SQL para interactuar con los datos almacenados en la base de datos.
- Además de estas características básicas, SQL Developer también ofrece una serie de características avanzadas como:
  - Soporte para versiones múltiples de bases de datos de Oracle y otros motores de bases de datos relacionales como MySQL, Microsoft SQL Server, etc.
  - Soporte para depuración de código SQL y PL/SQL (lenguaje de programación de Oracle).
  - Capacidades de importación y exportación de datos.
  - Integración con otras herramientas de Oracle como Oracle Data Modeler, Oracle BI Publisher, etc.
  - Capacidades de generación de informes y análisis de datos.

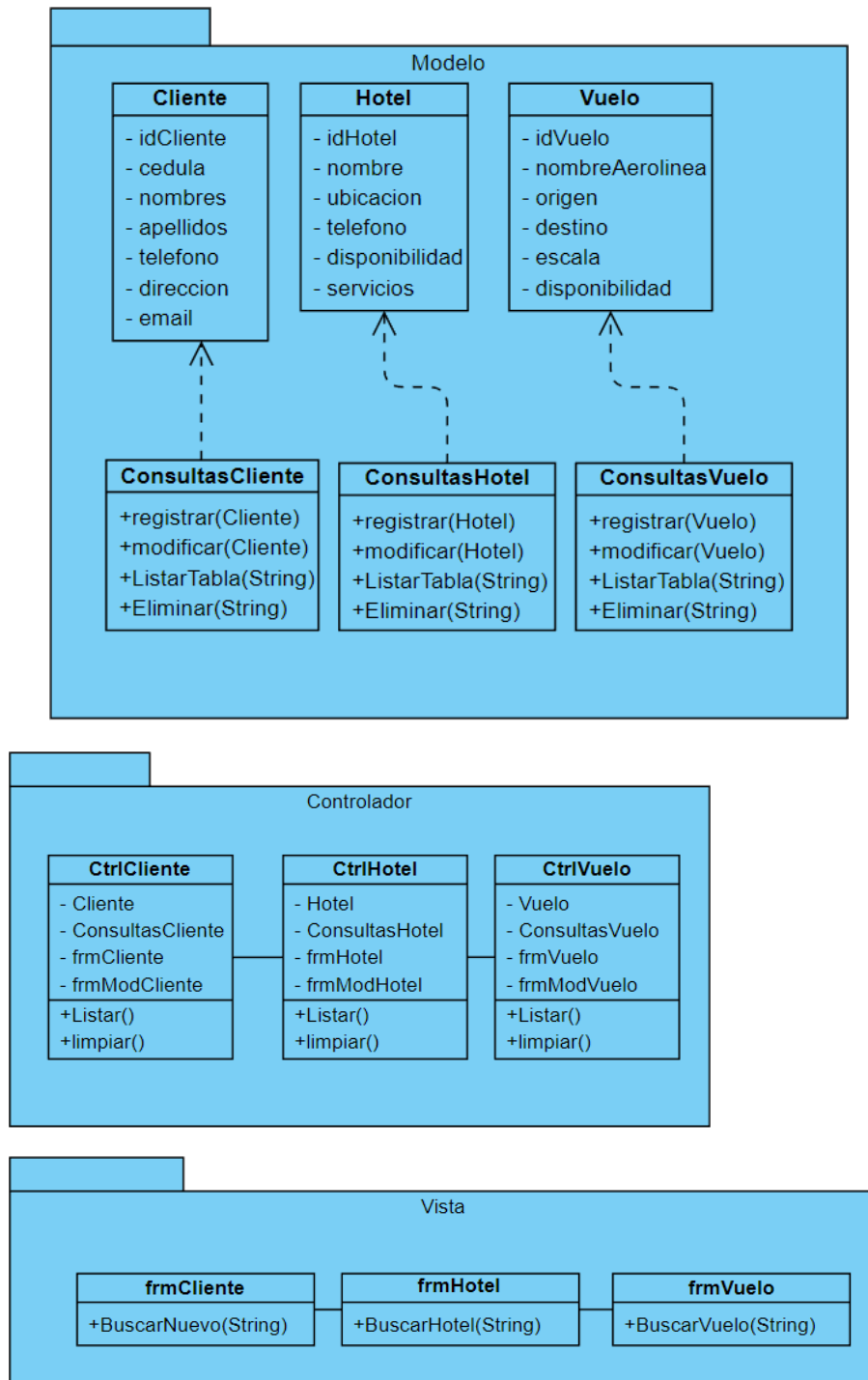


### 3 ENTIDAD RELACIÓN

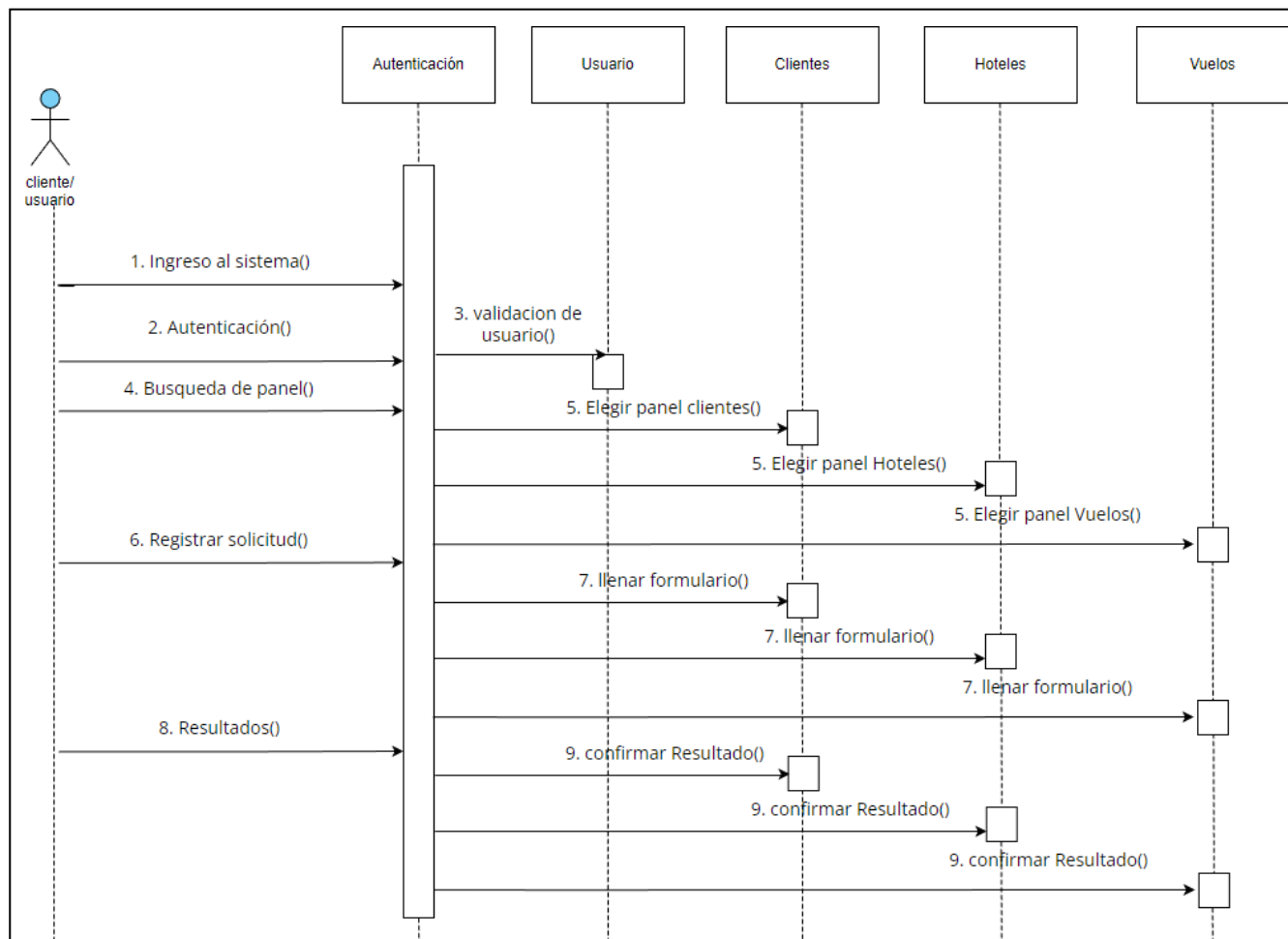


## 4 DIAGRAMAS

### 4.1 DIAGRAMA DE CLASES

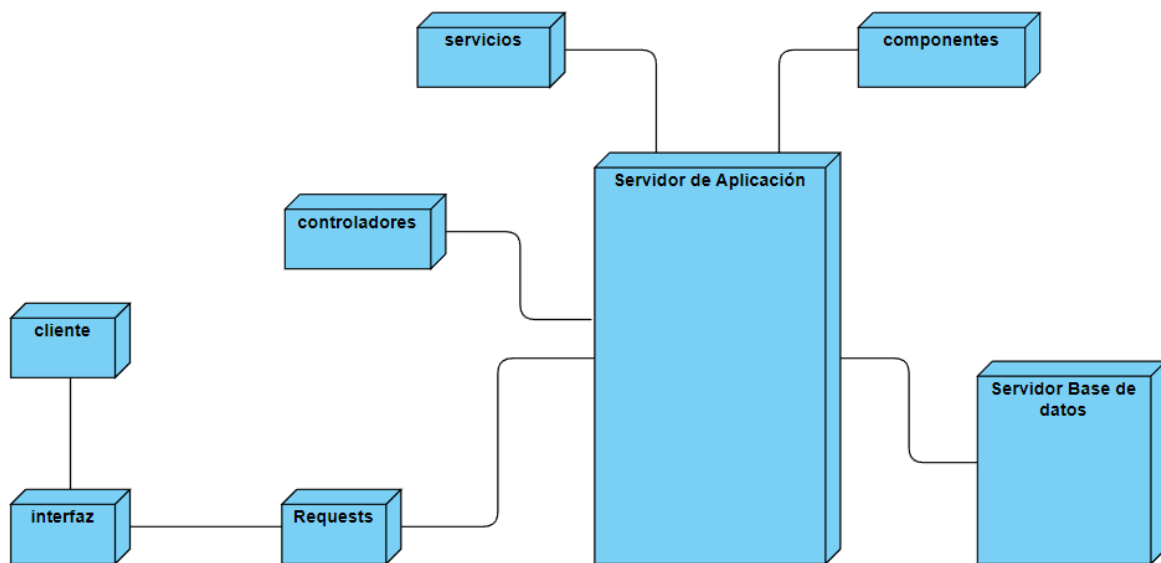


## 4.2 DIAGRAMA DE SECUENCIA





### 4.3 VISTA DE DESPLIEGUE

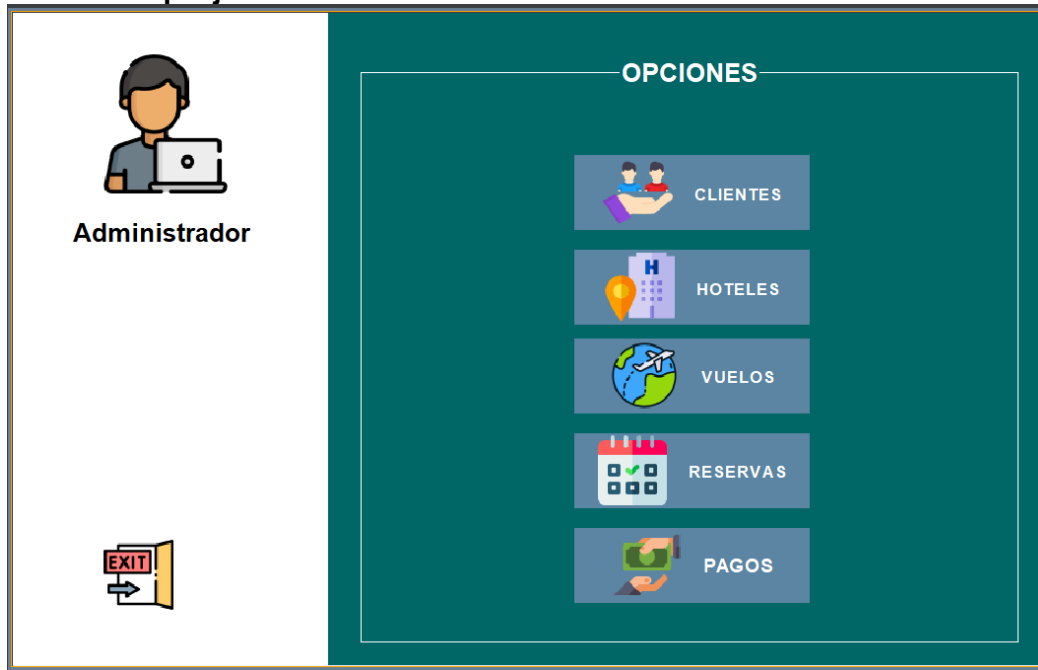


## 5 Decisiones Técnicas

Política	Framework	Versión	Descripción
Interfaces	NetBeans	17.0	Framework a utilizar para crear las interfaces de la aplicación.
Servicios	NodeJs	11.10.0	Entorno para el desarrollo de los diferentes servicios.
Base de Datos	SQLDeveloper	22.2.1	Base de datos a utilizar para la aplicación.

## 6 CODIGO DEL DESARROLLO DEL SISTEMA

### MenuPrincipal.java



Estos métodos definen las acciones que se realizan cuando se interactúa con diferentes elementos de la interfaz de usuario, como seleccionar opciones de menú o hacer clic en botones. Cada método crea instancias de las clases correspondientes y utiliza controladores para realizar acciones específicas, como mostrar ventanas, listar datos y gestionar la interacción con la base de datos.

### FrmLogin.java:



```

private void txtContraseñaKeyPressed(java.awt.event.KeyEvent evt) {
    if (evt.getKeyCode() == KeyEvent.VK_ENTER) {
        ConsultasLogin clo = new ConsultasLogin();
        Usuario usu = new Usuario();
        usu.setUsuario(usuario: txtUsuario.getText());
        usu.setClave(clave: txtContraseña.getText());
        String usuario = txtUsuario.getText();
        String clave = txtContraseña.getText();

        try {
            if (usuario.isEmpty() || clave.isEmpty()) {
                JOptionPane.showMessageDialog(parentComponent: null, message: "Llene los dos campos");
            } else {
                // Validar usuario
                boolean valido = clo.validarUsuario(usuario, clave);
                if (valido) {
                    mostrarMensaje("Bienvenido " + usuario + "!");
                    cerrarVentana();
                } else {
                    mostrarMensaje(mensaje: "Credenciales incorrectas");
                }
            }
        } catch (SQLException ex) {
            mostrarMensaje(mensaje: "Error al conectar a la base de datos");
        }
    }
}

```

Este código captura el evento de presionar la tecla Enter en el campo de contraseña, verifica las credenciales ingresadas y muestra mensajes correspondientes en función del resultado de la validación.

#### CtrlLogin.java

```

public void actionPerformed(java.awt.event.ActionEvent evt) {
    if (evt.getSource() == frm1.btnEntrar) {
        String usuario;
        usuario = frm1.txtUsuario.getText();
        String clave;
        clave = frm1.txtContraseña.getText();

        try {
            if (usuario.isEmpty() || clave.isEmpty()) {
                JOptionPane.showMessageDialog(parentComponent: null, message: "Llene los dos campos");
            } else {
                // Validar usuario
                boolean valido;
                valido = clo.validarUsuario(usuario, clave);
                if (valido) {
                    frm1.mostrarMensaje("Bienvenido " + usuario + "!");
                    frm1.cerrarVentana();
                } else {
                    frm1.mostrarMensaje(mensaje: "Credenciales incorrectas");
                }
            }
        } catch (SQLException ex) {
            frm1.mostrarMensaje(mensaje: "Error al conectar a la base de datos");
        }
    }
}

```

Este código captura el evento de acción del botón "Entrar", verifica las credenciales ingresadas y muestra mensajes correspondientes en función del resultado de la validación.

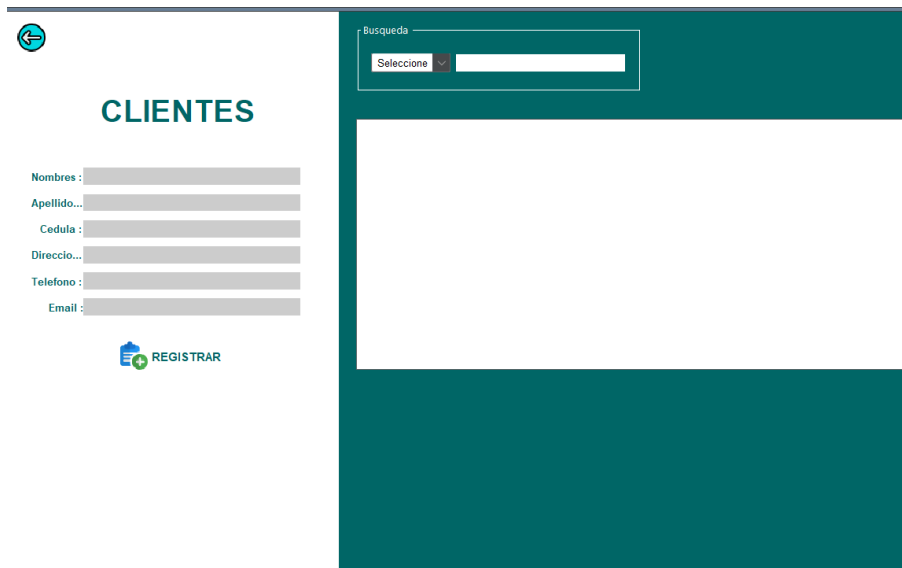
## Código main:

```
1 package Main;
2
3 import Vista.*;
4 /**
5  *
6  * @author staly
7  */
8 public class Main {
9
10    /**
11     *
12     * @param args
13     */
14    public static void main(String[] args) {
15        FrmLogin frml;
16        frml = new FrmLogin();
17        frml.setVisible(b: true);
18    }
19 }
```

Este es el código donde comienza la ejecución de todo el proyecto.

## Código Cliente

frmCliente.java



```

private void tblClienteMouseClicked(java.awt.event.MouseEvent evt) {
    Cliente c = new Cliente();
    ConsultasCliente modC = new ConsultasCliente();
    frmModCliente frmMod = new frmModCliente();
    CtrlCliente ctrlC = new CtrlCliente(modC, modC, frmC, this, frmMod);
    int column = tblCliente.getColumnModel().getColumnIndexAtX(evt.getX());
    int row = evt.getY() / tblCliente.getRowHeight();
    if (row < tblCliente.getRowCount() && row >= 0 && column < tblCliente.getColumnCount() && column >= 0) {
        Object value = tblCliente.getValueAt(row, column);
        if (value instanceof JButton boton) {
            if (boton.getName().equals("m")) {
                int seleccion = this.tblCliente.getSelectedRow();
                frmMod.txtIdCliente.setText(s.tblCliente.getValueAt(row: seleccion, column: 0).toString());
                frmMod.txtCedula.setText(s.tblCliente.getValueAt(row: seleccion, column: 1).toString());
                frmMod.txtNombres.setText(s.tblCliente.getValueAt(row: seleccion, column: 2).toString());
                frmMod.txtApellidos.setText(s.tblCliente.getValueAt(row: seleccion, column: 3).toString());
                frmMod.txtTelefono.setText(s.tblCliente.getValueAt(row: seleccion, column: 4).toString());
                frmMod.txtDireccion.setText(s.tblCliente.getValueAt(row: seleccion, column: 5).toString());
                frmMod.txtEmail.setText(s.tblCliente.getValueAt(row: seleccion, column: 6).toString());
                frmMod.setVisible(s: true);
            }
            if (boton.getName().equals("e")) {
                int fila = this.tblCliente.getSelectedRow();
                String codigo = tblCliente.getValueAt(row: fila, column: 0).toString();
                if (fila < 0) {
                    JOptionPane.showMessageDialog(parentComponent: null, message: "Debe de seleccionar un registro de la tabla ", title: "AVISO", messageType: JOptionPane.INFORMATION_MESSAGE);
                } else {
                    ConsultasCliente.Eliminar(id: codigo);
                    JOptionPane.showMessageDialog(parentComponent: null, message: "Registro Eliminado!");
                }
                ctrlC.Listar();
            }
        }
    }
}

```

El código maneja eventos de clic en los botones dentro de la tabla de clientes y realiza acciones como mostrar una ventana de modificación de cliente o eliminar un cliente de la lista.

```

public void BuscarNuevo(String buscar) {
    this.tblCliente.setDefaultRenderer(columnClass: Object.class, new Render());
    DefaultTableModel md = new DefaultTableModel() {
        public boolean isCellEditable(int row, int column) {
            return false;
        }
    };
    this.tblCliente.setRowHeight(rowHeight: 20);
    JButton BOTONModificar = new JButton(text: "Modificar");
    BOTONModificar.setName(name: "m");
    JButton BOTONEliminar = new JButton(text: "Eliminar");
    BOTONEliminar.setName(name: "e");

    String sql = "";
    buscar_box = (String) cb_buscar.getSelectedItemAt();
    switch (buscar_box) {
        case "Cedula" -> sql = "Select*from CLIENTE where CEDULA like'" + buscar + "'ORDER BY ID";
        case "Nombre" -> sql = "Select*from CLIENTE where NOMBRES like'" + buscar + "' ORDER BY ID";
        case "Apellido" -> sql = "Select*from CLIENTE where APELLIDOS like'" + buscar + "'ORDER BY ID";
        case "Direccion" -> sql = "Select*from CLIENTE where DIRECCION like'" + buscar + "'ORDER BY ID";
        default -> {
        }
    }

    ResultSet rs = ConsultasCliente.ListarTabla(consulta: sql);
    md.setColumnIdentifiers(new Object[]{"Id", "Cedula", "Nombres", "Apellidos", "Telefono", "Direccion", "Email", "", ""});
    try {
        while (rs.next()) {
            md.addRow(new Object[]{rs.getInt(string: "Id"), rs.getString(string: "Cedula"), rs.getString(string: "Nombres"), rs.getString(string: "Apellidos"),
                rs.getString(string: "Telefono"), rs.getString(string: "Direccion"), rs.getString(string: "Email"), BOTONModificar, BOTONEliminar});
            this.tblCliente.setModel(dataModel: md);
        }
    } catch (Exception e) {
        System.out.println(s: e);
    }
}

```

el método BuscarNuevo realiza una búsqueda en la tabla de clientes según un criterio específico y muestra los resultados en la tabla tblCliente, utilizando un modelo de tabla personalizado y renderizadores especiales para los botones "Modificar" y "Eliminar".

## Ctrlcliente.java

```
public void actionPerformed(ActionEvent e) {
    //Guardar Cliente
    if (e.getSource() == frmC.btnGuardar) {
        mod.setCedula(cedula: frmC.txtCedula.getText());
        mod.setNombres(nombres: frmC.txtNombres.getText());
        mod.setApellidos(apellidos: frmC.txtApellidos.getText());
        mod.setTelefono(telefono: frmC.txtTelefono.getText());
        mod.setDireccion(direccion: frmC.txtDireccion.getText());
        mod.setEmail(email: frmC.txtEmail.getText());
        String email = frmC.txtEmail.getText().trim();
        String cedula = frmC.txtCedula.getText().trim();
        String telefono = frmC.txtTelefono.getText().trim();
        try {
            if (isValidEmail(email) && validarCedulatelefono(cedula) && validarCedulatelefono(telefono)) {
                if (modC.registrar(c: mod)) {
                    JOptionPane.showMessageDialog(parentComponent: null, message: "Registro Guardado");
                    limpiar();
                    Listar();
                } else {
                    JOptionPane.showMessageDialog(parentComponent: null, message: "Error al Guardar");
                }
            } else {
                JOptionPane.showMessageDialog(parentComponent: null, message: "Revise si los datos ingresados son correctos");
            }
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(parentComponent: null, message: ex.getMessage(), title: "Error", messageType: JOptionPane.ERROR_MESSAGE);
        }
    }

    if (e.getSource() == frmMod.btnActualizar) {
        mod.setIdCliente(idcliente: Integer.parseInt(s: frmMod.txtIdCliente.getText()));
        mod.setCedula(cedula: frmMod.txtCedula.getText());
        mod.setNombres(nombres: frmMod.txtNombres.getText());
        mod.setApellidos(apellidos: frmMod.txtApellidos.getText());
        mod.setTelefono(telefono: frmMod.txtTelefono.getText());
        mod.setDireccion(direccion: frmMod.txtDireccion.getText());
        mod.setEmail(email: frmMod.txtEmail.getText());
        String email = frmMod.txtEmail.getText().trim();
        String telefono = frmMod.txtTelefono.getText().trim();
        try {
            if (isValidEmail(email) && validarCedulatelefono(cedula: telefono)) {
                if (modC.modificar(c: mod)) {
                    JOptionPane.showMessageDialog(parentComponent: null, message: "Registro Actualizado");
                    frmMod.setVisible(s: false);
                    Listar();
                }
            }
        }
    }
}
```

el método actionPerformed maneja los eventos de los botones "Guardar" y "Actualizar" en la interfaz gráfica y realiza las acciones correspondientes, como guardar un nuevo registro de cliente en la base de datos o actualizar un registro existente. También realiza validaciones en los datos ingresados antes de realizar las operaciones en la base de datos.

```
public void Listar() {
    frmC.tblCliente.setDefaultRenderer(columnClass: Object.class, new Render());
    DefaultTableModel md = new DefaultTableModel() {
        public boolean isCellEditable(int row, int column) {
            return false;
        }
    };
    frmC.tblCliente.setRowHeight(rowHeight: 20);
    JButton BOTONModificar = new JButton(text: "Modificar");
    BOTONModificar.setName(name: "m");
    JButton BOTONEliminar = new JButton(text: "Eliminar");
    BOTONEliminar.setName(name: "e");
    ResultSet rs = ConsultasCliente.ListarTabla(consulta: "select * from CLIENTE");
    md.setColumnIdentifiers(new Object[]{"Id", "Cedula", "Nombres", "Apellidos", "Telefono", "Direccion", "Email", "", ""});
    try {
        while (rs.next()) {
            md.addRow(new Object[]{rs.getInt(string: "id"), rs.getString(string: "Cedula"), rs.getString(string: "Nombres"), rs.getString(string: "Apellidos"),
                rs.getString(string: "Telefono"), rs.getString(string: "Direccion"), rs.getString(string: "Email"), BOTONModificar, BOTONEliminar});
            frmC.tblCliente.setModel(dataModel: md);
        }
    } catch (Exception e) {
        System.out.println(s: e);
    }
}
```

El método Listar obtiene los registros de clientes de la base de datos y los muestra en la tabla frmC.tblCliente de la interfaz gráfica, utilizando un modelo de tabla personalizado y renderizadores especiales para los botones "Modificar" y "Eliminar".

## ConsultasClienteTest.java

The screenshot displays an IDE window titled 'ConsultasClienteTest.java'. The 'Source' tab is active, showing the following Java code:

```
28  */
29  @Test
30  public void testRegistrar() {
31      //ESTABLECE LA CONEXION A LA BASE DE DATOS
32      CallableStatement ps = null;
33      Connection con = getConnection();
34
35      //SENTENCIA SQL
36      String sql = "(CALL REGISTRAR_CLIENTE(INCREMENTAIDCLIENTE.NEXTVAL,?,?,?,?,?))"; //Insertando datos en la tabla CLIENTE
37
38      try {
39          ps = (CallableStatement) con.prepareCall(sql);
40          //NUMERO DE CEDULA DEL CLIENTE
41          ps.setString(1, "0954310736");
42          //NOMBRES DEL CLIENTE
43          ps.setString(2, "Maximiliano Stalin");
44          //APELLIDOS DEL CLIENTE
45          ps.setString(3, "Cabrera Gamboa");
46          //NUMERO DE TELEFONO DEL CLIENTE
47          ps.setString(4, "0962712966");
48          //DIRECCION DOMICILIARIA DEL CLIENTE
49          ps.setString(5, "Samborondon-Guayas");
50          //EMAIL DEL CLIENTE
51          ps.setString(6, "maximilianocabrera885@gmail.com");
52          ps.execute();
53
54      } catch (SQLException e) {
55          System.out.println(e);
56      } finally {
57          try {
58              con.close();
59          } catch (SQLException e) {
```

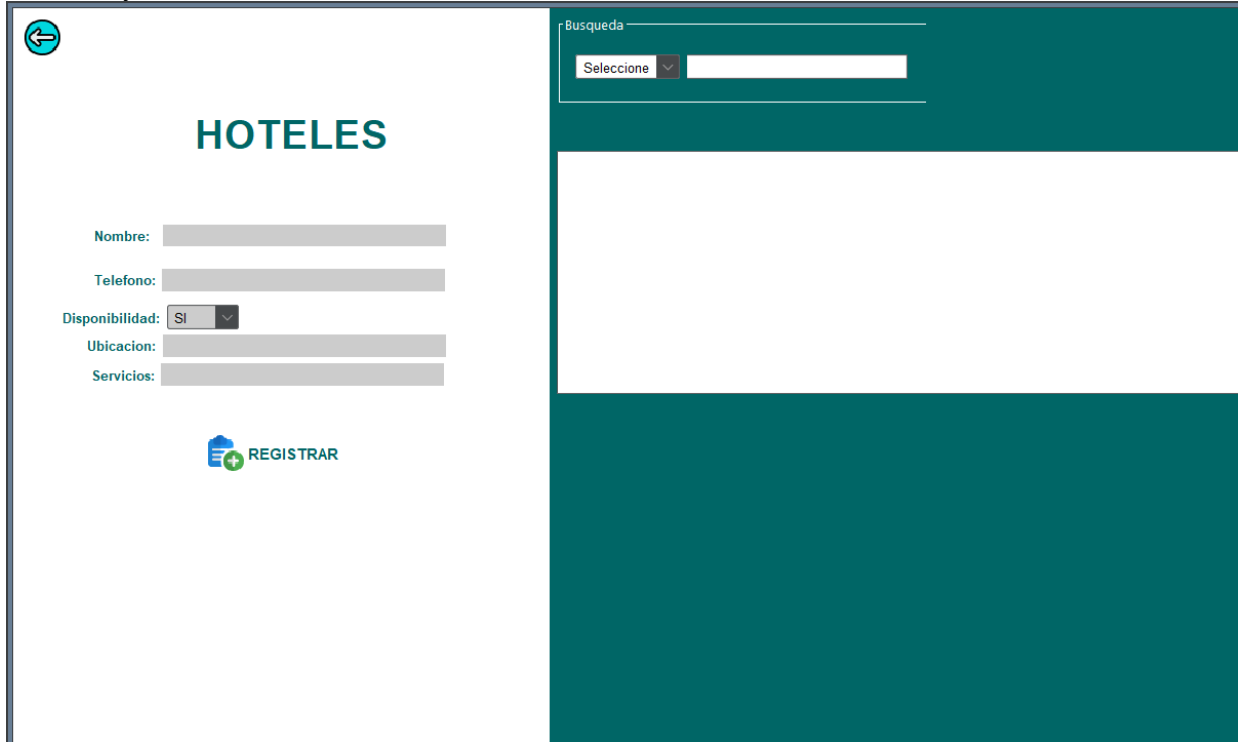
The 'Test Results' tab is also visible, showing the following information:

- Modelo.ConsultasClienteTest.testRegistrar X
- Tests passed: 100.00 %
- The test passed. (2,305 s)
- Conexion Exitosa

El método `testRegistrar` establece una conexión a la base de datos, prepara y ejecuta una llamada a un procedimiento almacenado para registrar un nuevo cliente en la tabla `CLIENTE`. Se establecen los valores de los parámetros de entrada y se capturan las excepciones que puedan ocurrir durante la ejecución.

## Código Hotel

frmHotel.java



```
private void tblHotelMouseClicked(java.awt.event.MouseEvent evt) {  
    Hotel h = new Hotel();  
    ConsultasHotel modH = new ConsultasHotel();  
    frmModHotel frmMod = new frmModHotel();  
    CtrlHotel ctrlc = new CtrlHotel(mod: h, modH, frmH: this, frmMod);  
    int column = tblHotel.getColumnModel().getColumnIndexAtX(sPosition: evt.getX());  
    int row = evt.getY() / tblHotel.getRowHeight();  
    if (row < tblHotel.getRowCount() && row >= 0 && column < tblHotel.getColumnCount() && column >= 0) {  
        Object value = tblHotel.getValueAt(row, column);  
        if (value instanceof JButton boton) {  
            if (boton.getName().equals(asObject: "m")) {  
                int seleccion = this.tblHotel.getSelectedRow();  
                frmMod.txtIdHotel.setText(s: tblHotel.getValueAt(row: seleccion, column: 0).toString());  
                frmMod.txtNombre.setText(s: tblHotel.getValueAt(row: seleccion, column: 1).toString());  
                frmMod.txtUbicacion.setText(s: tblHotel.getValueAt(row: seleccion, column: 2).toString());  
                frmMod.txtTelefono.setText(s: tblHotel.getValueAt(row: seleccion, column: 3).toString());  
                frmMod.txtDisponibilidad.setText(s: tblHotel.getValueAt(row: seleccion, column: 4).toString());  
                frmMod.txtServicios.setText(s: tblHotel.getValueAt(row: seleccion, column: 5).toString());  
                frmMod.setVisible(b: true);  
            }  
            if (boton.getName().equals(asObject: "e")) {  
                int fila = this.tblHotel.getSelectedRow();  
                String codigo = tblHotel.getValueAt(row: fila, column: 0).toString();  
                if (fila < 0) {  
                    JOptionPane.showMessageDialog(parentComponent: null, message: "Debe de seleccionar un registro de la tabla ", title: "AVISO", messageType: JOptionPane.WARNING_MESSAGE);  
                } else {  
                    ConsultasHotel.Eliminar(id: codigo);  
                    JOptionPane.showMessageDialog(parentComponent: null, message: "Registro Eliminado!");  
                    ctrlc.Listar();  
                }  
            }  
        }  
    }  
}
```

El código maneja el evento de clic en la tabla `tblHotel` y realiza diferentes acciones según el botón seleccionado (modificar o eliminar). Permite mostrar la información de un hotel en una ventana de modificación y eliminar un hotel seleccionado de la tabla.



```

public void BuscarHotel(String buscar) {
    this.tblHotel.setDefaultRenderer(new DefaultTableRenderer());
    DefaultTableModel md = new DefaultTableModel() {
        public boolean isCellEditable(int row, int column) {
            return false;
        }
    };

    this.tblHotel.setRowHeight(20);
    JButton BOTONModificar = new JButton(text: "Modificar");
    BOTONModificar.setName(name: "m");
    JButton BOTONEliminar = new JButton(text: "Eliminar");
    BOTONEliminar.setName(name: "e");

    String sql = "";
    buscar_box = (String) cb_buscar.getSelectedItem();
    if (buscar_box.equals(anObject: "Nombre")) {
        sql = "Select*from HOTEL where NOMBRE like'" + buscar + "%'ORDER BY IDHOTEL";
    } else if (buscar_box.equals(anObject: "Ubicacion")) {
        sql = "Select*from HOTEL where UBICACION like'" + buscar + "%' ORDER BY IDHOTEL";
    }

    ResultSet rs = ConsultasCliente.ListarTabla(consulta: sql);
    md.setColumnIdentifiers(new Object[]{"ID", "Nombre", "Ubicacion", "Telefono", "Disponibilidad", "Servicios", "", ""});
    try {
        while (rs.next()) {
            md.addRow(new Object[]{rs.getInt(string: "IDHOTEL"), rs.getString(string: "Nombre"), rs.getString(string: "Ubicacion"), rs.getString(string: "Telefono"),
                rs.getString(string: "Disponibilidad"), rs.getString(string: "Servicios"), BOTONModificar, BOTONEliminar});
            this.tblHotel.setModel(dataModel: md);
        }
    } catch (Exception e) {
        System.out.println(e);
    }
}

```

El código realiza una búsqueda en la tabla de hoteles y actualiza la tabla con los resultados obtenidos, mostrando los hoteles que cumplen con el criterio de búsqueda especificado (nombre o ubicación).

#### CtrlHotel.java

```

public void actionPerformed(ActionEvent e) {
    //Guardar Hotel
    if (e.getSource() == frmH.btnGuardar) {
        mod.setNombre(nombre: frmH.txtNombre.getText());
        mod.setUbicacion(ubicacion: frmH.txtUbicacion.getText());
        mod.setTelefono(telefono: frmH.txtTelefono.getText());
        mod.setDisponibilidad((String) frmH.txtDisponibilidad.getSelectedItem());
        mod.setServicios(servicios: frmH.txtServicios.getText());
        String telefono = frmH.txtTelefono.getText().trim();
        try {
            // Verificar si alguno de los campos de texto está vacío
            if (validarCedulaTelefono(telefono)) {
                if (modH.registrar(h: mod)) {
                    JOptionPane.showMessageDialog(parentComponent: null, message: "Registro Guardado");
                    limpiar();
                    Listar();
                } else {
                    JOptionPane.showMessageDialog(parentComponent: null, message: "Error al Guardar");
                }
            } else {
                JOptionPane.showMessageDialog(parentComponent: null, message: "Revise si los datos ingresados son correctos");
            }
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(parentComponent: null, "Error al guardar los datos: " + ex.getMessage());
        }
    }

    if (e.getSource() == frmMod.btnActualizar) {
        mod.setIdHotel(idHotel: Integer.parseInt(s: frmMod.txtIdHotel.getText()));
        mod.setNombre(nombre: frmMod.txtNombre.getText());
        mod.setUbicacion(ubicacion: frmMod.txtUbicacion.getText());
        mod.setTelefono(telefono: frmMod.txtTelefono.getText());
        mod.setDisponibilidad(disponibilidad: frmMod.txtDisponibilidad.getText());
        mod.setServicios(servicios: frmMod.txtServicios.getText());

        if (modH.modificar(h: mod)) {
            JOptionPane.showMessageDialog(parentComponent: null, message: "Registro Actualizado");
            frmMod.setVisible(b: false);
            Listar();
        } else {
            JOptionPane.showMessageDialog(parentComponent: null, message: "Error al actualizar");
        }
    }
}

```

El código maneja los eventos de acción para los botones "Guardar" y "Actualizar" en los formularios de ingreso y modificación de hoteles. Realiza validaciones de datos y realiza las operaciones correspondientes de guardar o actualizar los registros de hoteles en la base de datos. Además, muestra mensajes de confirmación o error según sea necesario.

```
public void Listar() {
    frmH.tblHotel.setDefaultRenderer(columnClass: Object.class, new Render());
    DefaultTableModel md = new DefaultTableModel() {
        public boolean isCellEditable(int row, int column) {
            return false;
        }
    };
    frmH.tblHotel.setRowHeight(rowHeight: 20);
    JButton BOTONModificar = new JButton(text: "Modificar");
    BOTONModificar.setName(name: "m");
    JButton BOTONEliminar = new JButton(text: "Eliminar");
    BOTONEliminar.setName(name: "e");
    ResultSet rs = ConsultasCliente.ListarTabla(consulta: "select * from HOTEL ORDER BY IDHOTEL");
    md.setColumnIdentifiers(new Object[]{"ID", "Nombre", "Ubicacion", "Telefono", "Disponibilidad", "Servicios", "", ""});
    try {
        while (rs.next()) {
            md.addRow(new Object[]{rs.getInt(string: "IDHOTEL"), rs.getString(string: "Nombre"), rs.getString(string: "Ubicacion"), rs.getString(string: "Telefono"),
                rs.getString(string: "Disponibilidad"), rs.getString(string: "Servicios"), BOTONModificar, BOTONEliminar});
            frmH.tblHotel.setModel(dataModel: md);
        }
    } catch (Exception e) {
        System.out.println(e);
    }
}
```

el código consulta los datos de la tabla "HOTEL" en la base de datos y los muestra en una tabla en el formulario frmHotel. También configura botones en cada fila para permitir la modificación y eliminación de registros de hotel.

## ConsultasHotelTest.java

The screenshot shows an IDE with two tabs: 'CtrlHotel.java' and 'ConsultasHotelTest.java'. The 'ConsultasHotelTest.java' tab is active, displaying a Java class with a test method. The code is as follows:

```

25  * insertar un nuevo Hotel
26  */
27  @Test
28  public void testRegistrar() {
29      //ESTABLECE LA CONEXION A LA BASE DE DATOS
30      @SuppressWarnings("UnusedAssignment")
31      CallableStatement ps = null;
32      Connection con = getConnection();
33
34      //SENTENCIA SQL
35      String sql = "(CALL REGISTRAR_HOTEL(INCREMENTADOIDHOTEL.NEXTVAL,?,?,?,?))"; //Insertando datos en la tabla HOTEL
36
37      try {
38          ps = (CallableStatement) con.prepareCall(string: sql);
39          //NOMBRE DEL HOTEL
40          ps.setString(1, string: "Boston Hotel");
41          //DIRECCION DEL HOTEL
42          ps.setString(2, string: "Padre Vicente Solano Vargas, Guayaquil 090312");
43          //TELEFONO DEL HOTEL
44          ps.setString(3, string: "04230-8015");
45          //DISPONIBILIDAD DEL HOTEL
46          ps.setString(4, string: "si");
47          //SERVICIOS DEL HOTEL
48          ps.setString(5, string: "Estacionamiento, WIFI, Desayuno");
49          ps.execute();
50
51      } catch (SQLException e) {
52          System.out.println(e);
53      } finally {
54          try {
55              con.close();
56          } catch (SQLException e) {
57
58          }
59      }
60  }

```

Below the code editor, the 'Test Results' window is open, showing the test results for 'Modelo.ConsultasHotelTest.testModificar X'. The results indicate that all tests passed successfully.

Test Results	Conexion Exitosa
Tests passed: 100.00 %	
The test passed. (0.771 s)	

En este clase muestra los test de cada método implementado en el formHotel, verifica que todo los botones y acciones corran sin problemas.

## Código Vuelos

frmVuelo.java

←

Busqueda

Seleccione

VUELOS

Aerolínea: Aeroméxico

Destino:

Disponibilidad... Si

Origen:

Escala:

REGISTRAR

```
private void tblVueloMouseClicked(java.awt.event.MouseEvent evt) {  
    Vuelo h = new Vuelo();  
    ConsultasVuelo modV = new ConsultasVuelo();  
    frmModVuelo frmMod = new frmModVuelo();  
    CtrlVuelo ctrl = new CtrlVuelo(mod: h, modV, frmV: this, frmMod);  
    int column=tblVuelo.getColumnModel().getColumnIndexAtX(aPosition: evt.getX());  
    int row = evt.getY()/tblVuelo.getRowHeight();  
    if (row<tblVuelo.getRowCount() && row>= 0 && column<tblVuelo.getColumnCount() && column>=0) {  
        Object value=tblVuelo.getValueAt(row, column);  
        if (value instanceof JButton boton) {  
            if (boton.getName().equals(aObject: "M")) {  
                int seleccion=this.tblVuelo.getSelectedRow();  
                frmMod.txtIdVuelo.setText(s: tblVuelo.getValueAt(row: seleccion, column: 0).toString());  
                frmMod.txtAerolinea.setText(s: tblVuelo.getValueAt(row: seleccion, column: 1).toString());  
                frmMod.txtOrigen.setText(s: tblVuelo.getValueAt(row: seleccion, column: 2).toString());  
                frmMod.txtDestino.setText(s: tblVuelo.getValueAt(row: seleccion, column: 3).toString());  
                frmMod.txtEscala.setText(s: tblVuelo.getValueAt(row: seleccion, column: 4).toString());  
                frmMod.txtDisponibilidad.setText(s: tblVuelo.getValueAt(row: seleccion, column: 5).toString());  
                frmMod.setVisible(s: true);  
            }  
            if (boton.getName().equals(aObject: "E")) {  
                int fila =this.tblVuelo.getSelectedRow();  
                String codigo=tblVuelo.getValueAt(row: fila, column: 0).toString();  
                if(fila<0){  
                    JOptionPane.showMessageDialog(parentComponent: null, message: "Debe de seleccionar un registro de la tabla ",title: "AVISO",messageType: JOptionPane.  
                )else{//caso contrario eliminar registro  
                    ConsultasVuelo.Eliminar(id: codigo);  
                    JOptionPane.showMessageDialog(parentComponent: null, message: "Registro Eliminado!");  
                    ctrl.Listar();  
                }  
            }  
        }  
    }  
}
```

Este código maneja el evento de clic del ratón en la tabla de vuelos. Si se hace clic en el botón "Modificar", muestra un formulario de modificación con los datos del vuelo seleccionado. Si se hace clic en el botón "Eliminar", elimina el vuelo seleccionado de la base de datos y actualiza la lista de vuelos.

```

public void BuscarVuelo(String buscar) {
    this.tblVuelo.setDefaultRenderer(columnClass: Object.class, new Render());
    DefaultTableModel md = new DefaultTableModel() {
        public boolean isCellEditable(int row, int column) {
            return false;
        }
    };
    this.tblVuelo.setRowHeight(rowHeight: 20);
    JButton BOTONModificar= new JButton(text: "Modificar");
    BOTONModificar.setName(name: "M");
    JButton BOTONEliminar= new JButton(text: "Eliminar");
    BOTONEliminar.setName(name: "E");

    String sql = "";
    buscar_box=(String)cb_buscar.getSelectedItemAt();
    switch (buscar_box) {
        case "Aerolineas" -> sql = "Select*from VUELO where NOMBREAEROLINEA like'" + buscar + "%'ORDER BY IDVUELO";
        case "Origen" -> sql = "Select*from VUELO where ORIGEN like'" + buscar + "%' ORDER BY IDVUELO";
        case "Destino" -> sql = "Select*from VUELO where DESTINO like'" + buscar + "%' ORDER BY IDVUELO";
        default -> {
        }
    }
    ResultSet rs = ConsultasCliente.ListarTabla(consulta: sql);
    md.setColumnIdentifiers(new Object[]{"ID", "Aerolineas", "Origen", "Destino", "Escala", "Disponibilidad","",""});
    try {
        while (rs.next()) {
            md.addRow(new Object[]{rs.getInt(string: "IDVUELO"), rs.getString(string: "NOMBREAEROLINEA"), rs.getString(string: "ORIGEN"), rs.getString(string: "DESTINO"),
            rs.getString(string: "ESCALA"), rs.getString(string: "Disponibilidad"),BOTONModificar,BOTONEliminar});
            this.tblVuelo.setModel(dataModel: md);
        }
    } catch (Exception e) {
        System.out.println(e);
    }
}

```

Esta función realiza una búsqueda en la base de datos de vuelos según un criterio seleccionado y actualiza la tabla tblVuelo con los resultados de la búsqueda, mostrando los datos de los vuelos y permitiendo la modificación y eliminación de registros a través de los botones "Modificar" y "Eliminar".

## CtrlVuelo.java

```

public void actionPerformed(ActionEvent e) {
    //Guardar Cliente
    if (e.getSource() == frmV.btnGuardar) {
        if (frmV.txtOrigen.getText().isEmpty() || frmV.txtDestino.getText().isEmpty() || frmV.txtEscala.getText().isEmpty()) {
            JOptionPane.showMessageDialog(parentComponent: null, message: "Por favor, complete todos los campos antes de guardar.");
        } else {
            mod.setNombreAerolinea((String) frmV.txtAerolinea.getSelectedItem());
            mod.setOrigen(origen: frmV.txtOrigen.getText());
            mod.setDestino(destino: frmV.txtDestino.getText());
            mod.setEscala(escala: frmV.txtEscala.getText());
            mod.setDisponibilidad((String) frmV.txtDisponibilidad.getSelectedItem());

            try {
                if (modV.registrar(v: mod)) {
                    JOptionPane.showMessageDialog(parentComponent: null, message: "Registro Guardado");
                    limpiar();
                    Listar();
                } else {
                    JOptionPane.showMessageDialog(parentComponent: null, message: "Error al Guardar");
                }
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(parentComponent: null, "Error al guardar los datos: " + ex.getMessage());
            }
        }
    }

    if (e.getSource() == frmMod.btnActualizar) {
        mod.setIdVuelo(idVuelo: Integer.parseInt(:" frmMod.txtIdVuelo.getText()));
        mod.setNombreAerolinea(nombreAerolinea: frmMod.txtAerolinea.getText());
        mod.setOrigen(origen: frmMod.txtOrigen.getText());
        mod.setDestino(destino: frmMod.txtDestino.getText());
        mod.setEscala(escala: frmMod.txtEscala.getText());
        mod.setDisponibilidad(disponibilidad: frmMod.txtDisponibilidad.getText());

        if (modV.modificar(v: mod)) {
            JOptionPane.showMessageDialog(parentComponent: null, message: "Registro Actualizado");
            frmMod.setVisible(b: false);
            Listar();
        } else {
            JOptionPane.showMessageDialog(parentComponent: null, message: "Error al actualizar");
        }
    }
}

```

Este método maneja los eventos de acción de los botones "Guardar" y "Actualizar". Para el botón "Guardar", verifica la validez de los datos ingresados, guarda los datos en la base de datos y muestra mensajes de confirmación o error. Para el botón "Actualizar", actualiza los datos de un vuelo específico en la base de datos y muestra mensajes de confirmación o error.

```
public void Listar() {
    frmV.tblVuelo.setDefaultRenderer(columnClass: Object.class, new Render());
    DefaultTableModel md = new DefaultTableModel() {
        public boolean isCellEditable(int row, int column) {
            return false;
        }
    };
    frmV.tblVuelo.setRowHeight(rowHeight: 20);
    JButton BOTONModificar = new JButton(text: "Modificar");
    BOTONModificar.setName(name: "m");
    JButton BOTONEliminar = new JButton(text: "Eliminar");
    BOTONEliminar.setName(name: "e");
    ResultSet rs = ConsultasCliente.ListarTabla(consulta: "select * from VUELO ORDER BY IDVUELO");
    md.setColumnIdentifiers(new Object[]{"ID", "Aerolínea", "Origen", "Destino", "Escala", "Disponibilidad", "", ""});
    try {
        while (rs.next()) {
            md.addRow(new Object[]{rs.getInt(string: "IDVUELO"), rs.getString(string: "NOMBREAEROLINEA"), rs.getString(string: "ORIGEN"), rs.getString(string: "DESTINO"),
                rs.getString(string: "ESCALA"), rs.getString(string: "Disponibilidad"), BOTONModificar, BOTONEliminar});
            frmV.tblVuelo.setModel(dataModel: md);
        }
    } catch (Exception e) {
        System.out.println(e);
    }
}
```

Este método se encarga de obtener los datos de los vuelos desde la base de datos, crear un modelo de tabla personalizado, llenar el modelo de tabla con los datos de los vuelos y establecer el modelo de tabla en la tabla de la interfaz de usuario. Esto permite mostrar la lista de vuelos en la tabla con la posibilidad de interactuar con los botones "Modificar" y "Eliminar" para cada vuelo.

## CtrlVuelo.java

```
@Test
public void testRegistrar() {
    //ESTABLECE LA CONEXION A LA BASE DE DATOS
    CallableStatement ps = null;
    Connection con = getConnection();

    //SENTENCIA SQL
    String sql = "(CALL REGISTRAR_VUELO(INCREMENTAIDVUELO.NEXTVAL,?,?,?,?,?))"; //Insertando datos en la tabla VUELO

    try {
        ps = (CallableStatement) con.prepareCall(string: sql);
        //NOMBRE DE LA AEROLINEA
        ps.setString(i: 1, string: "Airlines");
        //ORIGEN DEL VUELO
        ps.setString(i: 2, string: "Guayaquil");
        //DESTINO DEL VUELO
        ps.setString(i: 3, string: "Miami");
        //ESCALA DEL VUELO
        ps.setString(i: 4, string: "Panama");
        //DISPONIBILIDAD DEL VUELO
        ps.setString(i: 5, string: "si");
        ps.execute();
    } catch (SQLException e) {
        System.out.println(e);
    } finally {
        try {
            con.close();
        } catch (SQLException e) {
            System.out.println(e);
        }
    }
}
```

Test Results X

Modelo.ConsultasHotelTestModificar X    Modelo.ConsultasVueloTesttestRegistrar X

Tests passed: 100.00 %    Conexion Exitosa

The test passed. (0.762 s)

Estos métodos de prueba están diseñados para probar las operaciones básicas de inserción, actualización, consulta y eliminación en la tabla de vuelos de la base de datos. Cada método establece la conexión, prepara y ejecuta la sentencia SQL correspondiente, y finalmente cierra la conexión.

## Código Conexión

### Conexión.java

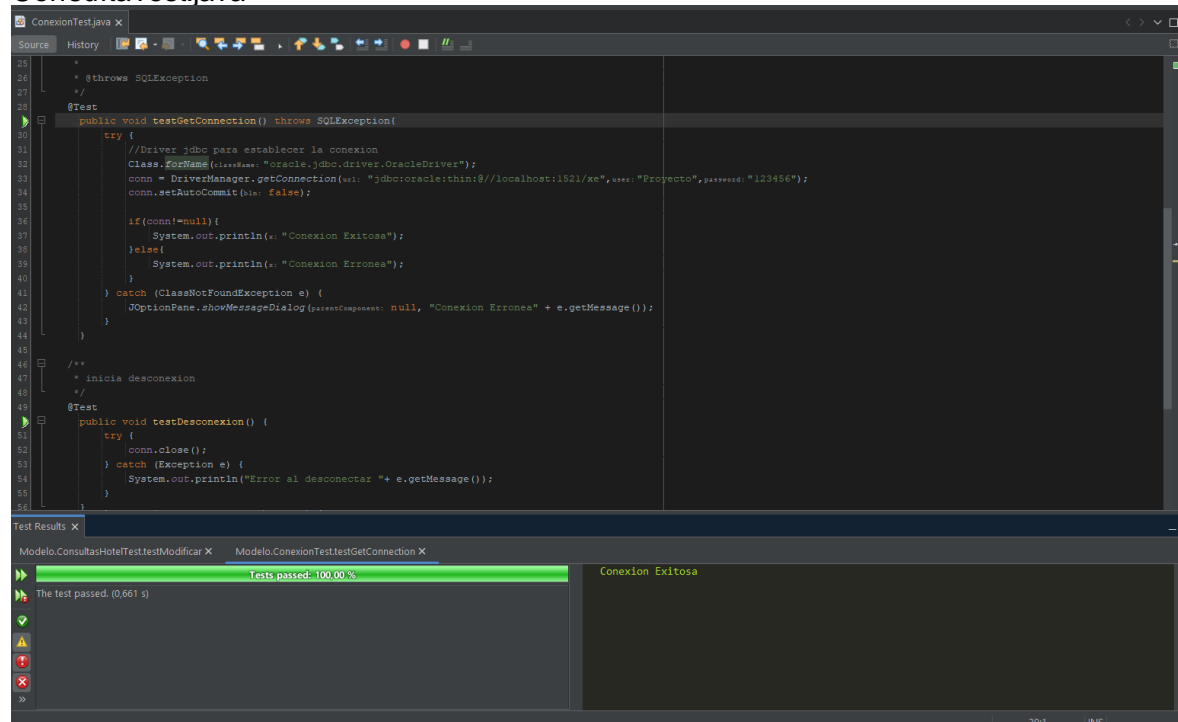
```
public class Conexion
{
    private static Connection conn = null;
    private static final String login = "Proyecto";//Usuario de la Base de Datos
    private static final String password = "123456";//Contraseña de la Base de Datos
    private static final String url = "jdbc:oracle:thin:@//localhost:1521/xe";//url conexion a la base de Datos llamada "Proyecto"
    //jdbc:oracle:thin:@//localhost:1522/XE
    /**
     * @return
     */
    public static Connection getConnection() {
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");//Driver jdbc para establecer la conexion
            conn = DriverManager.getConnection(url, login, password);
            conn.setAutoCommit(false);

            if(conn!=null){
                System.out.println("Conexion Exitosa");
            }else{
                System.out.println("Conexion Erronea");
            }
        } catch (ClassNotFoundException|SQLException e) {
            JOptionPane.showMessageDialog(null, "Conexion Erronea" + e.getMessage());
        }
        return conn;
    }
}

/**
 * mensaje si ocurre error de conexión
 */
public void desconexion() {
    try {
        conn.close();
    } catch (Exception e) {
        System.out.println("Error al desconectar " + e.getMessage());
    }
}
```

La clase Conexion proporciona métodos para establecer y cerrar una conexión a una base de datos Oracle utilizando JDBC, y el método main() se utiliza para probar la conexión llamando al método getConnection().

### ConsultaTest.java



```
ConexionTest.java X
Source History
25 * @throws SQLException
26 */
27 @Test
28 public void testGetConnection() throws SQLException {
29     try {
30         //Driver jdbc para establecer la conexion
31         Class.forName("oracle.jdbc.driver.OracleDriver");
32         conn = DriverManager.getConnection(url, login, password);
33         conn.setAutoCommit(false);
34
35         if(conn!=null){
36             System.out.println("Conexion Exitosa");
37         }else{
38             System.out.println("Conexion Erronea");
39         }
40     } catch (ClassNotFoundException e) {
41         JOptionPane.showMessageDialog(null, "Conexion Erronea" + e.getMessage());
42     }
43 }
44
45 /**
46 * inicia desconexión
47 */
48 @Test
49 public void testDesconexion() {
50     try {
51         conn.close();
52     } catch (Exception e) {
53         System.out.println("Error al desconectar " + e.getMessage());
54     }
55 }
56
Test Results X
Modelo.ConsultasHotelTestModifier X Modelo.ConexionTestTestGetConnection X
Tests passed: 100.00 %
Conexion Exitosa
The test passed. (0.661 s)
```

estos métodos se utilizan para probar la conexión a una base de datos Oracle. El método `testGetConnection()` intenta establecer la conexión y el método `testDesconexion()` cierra la conexión.

## CtrlReservas.java

```
public void actionPerformed(ActionEvent e) {

    //Validaciones campos vacios y fechas
    if (e.getSource() == frmR.btnReservar) {
        if (frmR.dateFechaInicio.getDate() == null || frmR.dateFechaFin.getDate() == null || frmR.combohoteles.getSelectedIndex() == -1 || frmR.txtHabitaciones.getText().isEmpty()) {
            JOptionPane.showMessageDialog(parentComponent: null, message: "Por favor, complete todos los campos antes de Reservar.");
        } else if (!ValidarFecha(fechaSeleccionada: frmR.dateFechaInicio.getDate()) || !ValidarFecha(fechaSeleccionada: frmR.dateFechaFin.getDate())) {
            JOptionPane.showMessageDialog(parentComponent: null, message: "La fecha no puede ser anterior a hoy.");
        } else {
            //código para guardar la reserva
            r.setFechaInicio(fechaInicio: frmR.dateFechaInicio.getDate());
            r.setFechaFin(fechaFin: frmR.dateFechaFin.getDate());
            r.setHotel((String) frmR.combohoteles.getSelectedIndex());
            r.setHabitaciones(habitaciones: Integer.parseInt(s: frmR.txtHabitaciones.getText()));
            r.setPersonas(personas: Integer.parseInt(s: frmR.txtPersonas.getText()));
            r.setCliente(cliente: frmR.txtCliente.getText());
            r.setCedula(cedula: frmR.txtCedula.getText());
            r.setPrecioTotal(precioTotal: Double.parseDouble(s: frmR.txtPrecioTotal.getText()));
            r.setEstado((String) frmR.txtEstado.getSelectedIndex());

            try {
                if (validarCedulaTelefono(cedula: r.getCedula())) {
                    if (cr.registrar(r)) {
                        JOptionPane.showMessageDialog(parentComponent: null, message: "Reserva Realizada ,Tenga un buen dia!");
                        limpiar();
                        Listar();
                    } else {
                        JOptionPane.showMessageDialog(parentComponent: null, message: "Error al realizar la Reserva");
                    }
                } else {
                    throw new CedulaInvalidaException();
                }
            } catch (CedulaInvalidaException ex) {
                JOptionPane.showMessageDialog(parentComponent: null, message: ex.getMessage(), title: "Error", messageType: JOptionPane.ERROR_MESSAGE);
            }
        }
    }

    if (e.getSource() == frmMR.btnActualizar) {
        if (!VerificarCamposCompletos(new JTextField[]{frmMR.txtIdReserva, frmMR.txtHabitaciones,
            frmMR.txtPersonas, frmMR.txtCliente, frmMR.txtCedula,
            frmMR.txtPrecioTotal, frmMR.txtEstado}))
            || frmMR.dateFechaInicio.getDate() == null || frmMR.combohoteles.getSelectedIndex() == -1 || frmMR.dateFechaFin.getDate() == null) {
        }
    }
}
```

este código maneja los eventos de los botones "Reservar" y "Actualizar" de dos formularios diferentes (frmR y frmMR). Realiza validaciones de campos vacíos y fechas, guarda o actualiza reservas según corresponda.

## CtrlPagon.java

```
public void Listar() {
    frmP.tblReservas.setDefaultRenderer(columnClass: Object.class, new Render());
    DefaultTableModel md = new DefaultTableModel() {
        public boolean isCellEditable(int row, int column) {
            return false;
        }
    };
    frmP.tblReservas.setRowHeight(rowHeight: 20);
    JButton BOTONPagar = new JButton(text: "Pagar");
    BOTONPagar.setName(name: "p");
    Connection con = getConexion();
    try {
        CallableStatement stmt = con.prepareCall(stating: "{call listar_reservas_pre_aprobadas(?)}");
        stmt.registerOutParameter(i: 1, is: OracleTypes.CURSOR);
        stmt.execute();
        ResultSet rs = (ResultSet) stmt.getObject(i: 1);
        md.setColumnIdentifiers(new Object[]{"IdReserva", "Cliente", "Cedula", "Hotel", "Habitaciones", "Personas", "Fecha Inicio", "Fecha Fin", "Precio Total", "Estado", "Boton"});
        while (rs.next()) {
            md.addRow(new Object[]{rs.getInt(stating: "IdReserva"), rs.getString(stating: "Cliente"), rs.getString(stating: "Cedula"), rs.getString(stating: "Hotel"),
                rs.getInt(stating: "Habitaciones"), rs.getInt(stating: "Personas"), rs.getDate(stating: "FechaInicio"), rs.getDate(stating: "FechaFin"), rs.getDouble(stating: "PrecioTotal"),
                frmP.tblReservas.getModel().getModel().getRowCount()});
        }
    } catch (Exception e) {
        System.out.println(s: e);
    }
}
```

Este método obtiene las reservas pre-aprobadas de la base de datos y las muestra en una tabla en la interfaz de usuario, incluyendo un botón "Pagar" en cada fila.