**ADMINISTRACIÓN DE BASES DE DATOS**
**GRUPO:** 231301
**SEMESTRE:** 2023 - 2024 / 1

UNIVERSIDAD AUTÓNOMA DE SAN LUIS POTOSÍ
FACULTAD DE INGENIERÍA
ÁREA DE CIENCIAS DE LA COMPUTACIÓN

# MANUAL PROYECTO POSTGRESQL - JAVA

GARCÍA MENDIETA GUILLERMO
QUINTANILLA ESCALANTE JUAN DIEGO

# Contenido

## Descripción del Sistema

Este sistema de gestión para un aeropuerto ha sido implementado utilizando la plataforma Netbeans con el lenguaje de programación Java. Proporciona una interfaz de usuario amigable que permite realizar diversas operaciones sobre las entidades del aeropuerto. A continuación, se detallan las características principales:

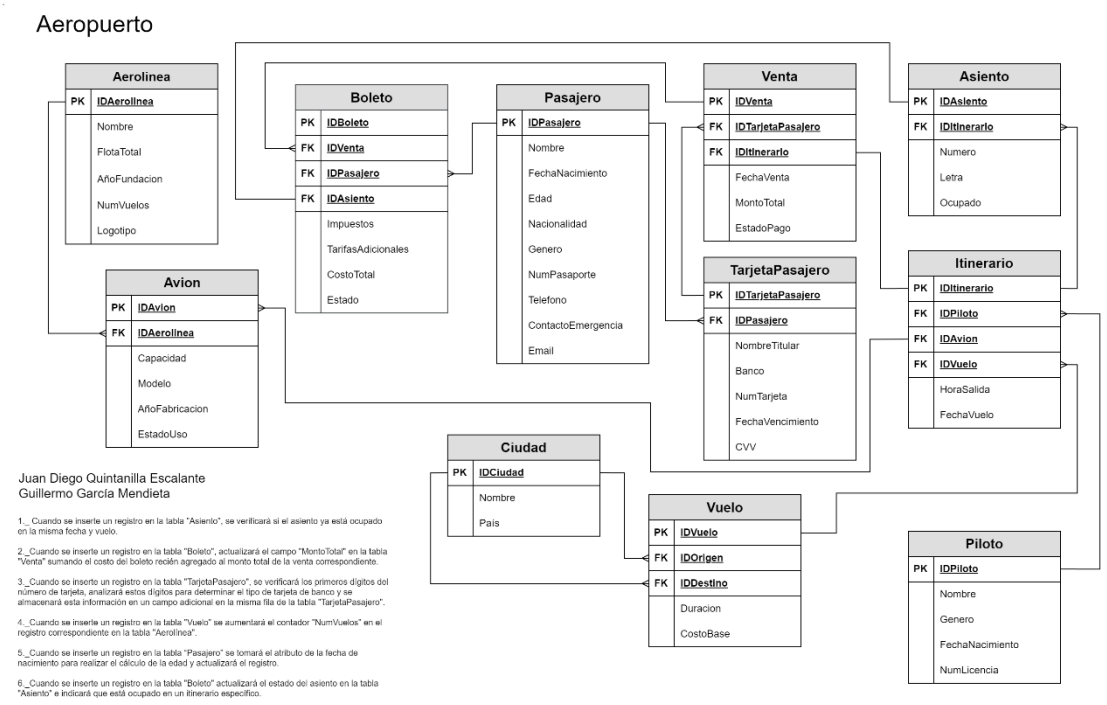
1. **Aerolínea:**
    - Formulario para dar de alta, modificar, eliminar y visualizar aerolíneas.
    - Interfaz gráfica para ingresar y mostrar información clave como nombre, flota total, año de fundación, número de vuelos y logotipo.
2. **Avión:**
    - Módulo de gestión de aviones con capacidad para asociarlos a una aerolínea.
    - Interfaz para registrar detalles como capacidad, modelo, año de fabricación y estado de uso.
    - Integración con la entidad de aerolínea.
3. **Piloto:**
    - Ventana para administrar la información de pilotos, permitiendo acciones como agregar, editar, eliminar y ver detalles.
    - Formulario para ingresar nombre, género, fecha de nacimiento y número de licencia, con validaciones incorporadas.
4. **Ciudad:**
    - Sección dedicada a la administración de ciudades y países.
    - Interfaz para ingresar y mostrar nombre de la ciudad y país.
5. **Vuelo:**
    - Panel de control para gestionar vuelos y visualizar información existente.
    - Formulario para ingresar origen, destino, duración, costo base etc.
6. **Itinerario:**
    - Módulo interactivo para manejar itinerarios de vuelo, con opciones para vincular pilotos, aviones y vuelos específicos.
    - Controles visuales para establecer la hora de salida y la fecha del vuelo.
7. **Pasajero:**
    - Interfaz para administrar datos de pasajeros.

- Formulario para ingresar detalles como nombre, fecha de nacimiento, nacionalidad, género, número de pasaporte, teléfono, contacto de emergencia y correo electrónico.

8. **Tarjeta Pasajero:**
    - Sección dedicada a la gestión de tarjetas asociadas a pasajeros, con opciones para agregar y ver detalles.
    - Formulario para ingresar nombre del titular, banco, número de tarjeta, fecha de vencimiento y código de seguridad.
9. **Asiento:**
    - Vista que permite controlar asientos en vuelos, con opciones para agregar y visualizar información.
    - Formulario para ingresar número de asiento, letra y estado de ocupación.
10. **Venta:**
    - Módulo de registro de transacciones de venta, con funciones de agregar y mostrar detalles.
    - Formulario para ingresar la fecha de venta, monto total y estado de pago.
11. **Boleto:**
    - Sección para administrar boletos, con opciones para agregar y visualizar información.
    - Formulario para ingresar impuestos, tarifas adicionales, costo total y estado del boleto.

Este sistema ha sido diseñado con una interfaz gráfica intuitiva utilizando Netbeans en Java, lo que facilita la interacción del usuario con las diferentes entidades y operaciones del aeropuerto. Además, garantiza la integridad de los datos mediante validaciones y relaciones establecidas en la capa de lógica del programa.

## Diagrama de la base de datos




## Script de la Base de datos

```
-- Database: Aeropuerto
-- Database: Aeropuerto

-- DROP DATABASE IF EXISTS "Aeropuerto";

CREATE DATABASE "Aeropuerto"
    WITH
    OWNER = postgres
    ENCODING = 'UTF8'
    LC_COLLATE = 'Spanish_Mexico.1252'
    LC_CTYPE = 'Spanish_Mexico.1252'
    TABLESPACE = pg_default
    CONNECTION LIMIT = -1
    IS_TEMPLATE = False;

CREATE SCHEMA InfoAerolinea;
CREATE SCHEMA InfoPasajero;

------------ TABLAS InfoAerolinea ------------
CREATE TABLE InfoAerolinea.Aerolinea
(
    idAerolinea BIGSERIAL NOT NULL,
```

```
    Nom_Aerolinea VARCHAR(200) NOT NULL,
    FlotaTotal INT NOT NULL,
    AñoFundacion INT NOT NULL,
    NumVuelos INT NOT NULL,
    Logotipo BYTEA NOT NULL,
    CONSTRAINT PK_AEROLINEA PRIMARY KEY (idAerolinea)
);

CREATE TABLE InfoAerolinea.Piloto
(
    idPiloto BIGSERIAL NOT NULL,
    Nom_Piloto VARCHAR(200) NOT NULL,
    Genero VARCHAR(20) NOT NULL,
    FechaNacimiento DATE NOT NULL,
    NumLicencia BIGINT NOT NULL,
    CONSTRAINT PK_PILOTO PRIMARY KEY (idPiloto)
);
ALTER TABLE InfoAerolinea.Piloto
ADD CONSTRAINT UQ_NUMLIC UNIQUE (NumLicencia);

CREATE TABLE InfoAerolinea.Ciudad
(
    idCiudad BIGSERIAL NOT NULL,
    Nom_Ciudad VARCHAR(200) NOT NULL,
    Pais VARCHAR(200) NOT NULL,
    CONSTRAINT PK_CIUDAD PRIMARY KEY (idCiudad)
);
ALTER TABLE InfoAerolinea.Ciudad
ADD CONSTRAINT UQ_CIUDADPAIS UNIQUE (Nom_Ciudad, Pais);

CREATE TABLE InfoAerolinea.Avion
(
    idAvion BIGSERIAL NOT NULL,
    idAerolinea BIGINT NOT NULL,
    Capacidad INT NOT NULL,
    Modelo VARCHAR(200) NOT NULL,
    AñoFabricacion INT NOT NULL,
    EstadoUso BOOLEAN NOT NULL,
    CONSTRAINT PK_AVION PRIMARY KEY (idAvion),
    CONSTRAINT FK_AEROLINEA FOREIGN KEY (idAerolinea) REFERENCES
InfoAerolinea.Aerolinea (idAerolinea)
);

CREATE TABLE InfoAerolinea.Vuelo
(
```

```
    idVuelo BIGSERIAL NOT NULL,
    idOrigen BIGINT NOT NULL,
    idDestino BIGINT NOT NULL,
    DuracionHoras INT NOT NULL,
    CostoBase MONEY NOT NULL,
    CONSTRAINT PK_VUELO PRIMARY KEY (idVuelo),
    CONSTRAINT FK_ORIGEN FOREIGN KEY (idOrigen) REFERENCES
InfoAerolinea.Ciudad (idCiudad),
    CONSTRAINT FK_DESTINO FOREIGN KEY (idDestino) REFERENCES
InfoAerolinea.Ciudad (idCiudad)
);
ALTER TABLE InfoAerolinea.Vuelo
ADD CONSTRAINT UQ_IDORGIDDEST UNIQUE (idOrigen, idDestino);

CREATE TABLE InfoAerolinea.Itinerario
(
    idItinerario BIGSERIAL NOT NULL,
    idPiloto BIGINT NOT NULL,
    idAvion BIGINT NOT NULL,
    idVuelo BIGINT NOT NULL,
    HoraSalida TIME NOT NULL,
    FechaVuelo DATE NOT NULL,
    CONSTRAINT PK_ITINERARIO PRIMARY KEY (idItinerario),
    CONSTRAINT FK_PILOTO FOREIGN KEY (idPiloto) REFERENCES
InfoAerolinea.Piloto (idPiloto),
    CONSTRAINT FK_AVION FOREIGN KEY (idAvion) REFERENCES InfoAerolinea.Avion
(idAvion),
    CONSTRAINT FK_VUELO FOREIGN KEY (idVuelo) REFERENCES InfoAerolinea.Vuelo
(idVuelo)
);
ALTER TABLE InfoAerolinea.Itinerario
ADD CONSTRAINT UQ_PILOTODIA UNIQUE (idPiloto, FechaVuelo);
select * from InfoAerolinea.Itinerario

------------ TABLAS InfoPasajero ------------
CREATE TABLE InfoPasajero.Pasajero
(
    idPasajero BIGSERIAL NOT NULL,
    Nom_Pasajero VARCHAR(200) NOT NULL,
    FechaNacimiento DATE NOT NULL,
    Edad INT,
    Nacionalidad VARCHAR(50) NOT NULL,
    Genero VARCHAR(20) NOT NULL,
    NumPasaporte VARCHAR(15) NOT NULL,
    Telefono BIGINT NOT NULL,
```

```
    ContactoEmergencia BIGINT NOT NULL,
    Email VARCHAR(200) NOT NULL,
    CONSTRAINT PK_PASAJERO PRIMARY KEY (idPasajero)
);
ALTER TABLE InfoPasajero.Pasajero
ADD CONSTRAINT UQ_NUMPASAPORTEPASAJERO UNIQUE (NumPasaporte);
ALTER TABLE InfoPasajero.Pasajero
ADD CONSTRAINT UQ_NUMTELPASAJERO UNIQUE (Telefono);
ALTER TABLE InfoPasajero.Pasajero
ADD CONSTRAINT UQ_EMAILPASAJERO UNIQUE (Email);

CREATE TABLE InfoPasajero.TarjetaPasajero
(
    idTarjetaPasajero BIGSERIAL NOT NULL,
    idPasajero BIGINT NOT NULL,
    NombreTitular VARCHAR(200) NOT NULL,
    Banco VARCHAR(200) NOT NULL,
    NumTarjeta BIGINT NOT NULL,
    FechaVencimiento DATE NOT NULL,
    CVV INT NOT NULL,
    CONSTRAINT PK_TARJETAPASAJERO PRIMARY KEY (idTarjetaPasajero),
    CONSTRAINT FK_PASAJERO1 FOREIGN KEY (idPasajero) REFERENCES
InfoPasajero.Pasajero (idPasajero)
)
ALTER TABLE InfoPasajero.TarjetaPasajero
ADD CONSTRAINT UQ_IDPASAJNUMTARJ UNIQUE (idPasajero, NumTarjeta);

CREATE TABLE InfoPasajero.Asiento
(
    idAsiento BIGSERIAL NOT NULL,
    idItinerario BIGINT NOT NULL,
    Num_Asiento INT NOT NULL,
    Letra CHAR(1) NOT NULL,
    Ocupado BOOLEAN NOT NULL,
    CONSTRAINT PK_ASIENTO PRIMARY KEY (idAsiento),
    CONSTRAINT FK_ITINERARIO1 FOREIGN KEY (idItinerario) REFERENCES
InfoAerolinea.Itinerario (idItinerario)
)
ALTER TABLE InfoPasajero.Asiento
ADD CONSTRAINT UQ_ASIENTO UNIQUE (idItinerario, Num_Asiento, Letra);

CREATE TABLE InfoPasajero.Venta
(
    idVenta BIGSERIAL NOT NULL,
    idTarjetaPasajero BIGINT NOT NULL,
```

```
    idItinerario BIGINT NOT NULL,
    FechaVenta TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    MontoTotal MONEY,
    EstadoPago BOOLEAN NOT NULL,
    CONSTRAINT PK_VENTA PRIMARY KEY (idVenta),
    CONSTRAINT FK_TARJETAPASAJERO FOREIGN KEY (idTarjetaPasajero) REFERENCES
InfoPasajero.TarjetaPasajero (idTarjetaPasajero),
    CONSTRAINT FK_ITINERARIO2 FOREIGN KEY (idItinerario) REFERENCES
InfoAerolinea.Itinerario (idItinerario)
);
ALTER TABLE InfoPasajero.Venta
ADD CONSTRAINT UQ_TJPASAJEROITINERARIO UNIQUE (idTarjetaPasajero,
idItinerario);
SELECT * FROM InfoPasajero.Venta

CREATE TABLE InfoPasajero.Boleto
(
    idBoleto BIGSERIAL NOT NULL,
    idVenta BIGINT NOT NULL,
    idPasajero BIGINT,
    idAsiento BIGINT NOT NULL,
    Impuestos REAL NOT NULL,
    TarifasAdicionales MONEY NOT NULL,
    CostoTotal MONEY NOT NULL,
    Estado BOOLEAN NOT NULL,
    CONSTRAINT PK_BOLETO PRIMARY KEY (idBoleto),
    CONSTRAINT FK_VENTA FOREIGN KEY (idVenta) REFERENCES InfoPasajero.Venta
(idVenta),
    CONSTRAINT FK_PASAJERO2 FOREIGN KEY (idPasajero) REFERENCES
InfoPasajero.Pasajero (idPasajero),
    CONSTRAINT FK_ASIENTO FOREIGN KEY (idAsiento) REFERENCES
InfoPasajero.Asiento (idAsiento)
);
SELECT * FROM InfoPasajero.Boleto

------------------------------------ TRIGGERS ---------------------------
----------------------
-- Trigger 1 --
CREATE OR REPLACE FUNCTION TR_CUPOASIENTO_FUNCTION()
RETURNS TRIGGER AS $$
BEGIN
    -- Verifica si la clave foránea se ha insertado más de 8 veces.
    IF (SELECT COUNT(*) FROM InfoPasajero.Asiento WHERE idItinerario =
NEW.idItinerario) > 8 THEN
        -- Si ya se insertó 8 veces, entonces genera un error.
```

```
        RAISE EXCEPTION 'No se pueden insertar más de 8 registros con el
mismo itinerario';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER TR_CUPOASIENTO
AFTER INSERT OR UPDATE ON InfoPasajero.Asiento
FOR EACH ROW
EXECUTE FUNCTION TR_CUPOASIENTO_FUNCTION();


-- Trigger 2 --
CREATE OR REPLACE FUNCTION TR_ACTUALIZAMONTOTOTAL1_FUNCTION()
RETURNS TRIGGER AS $$
DECLARE
    CostoBoleto MONEY;
BEGIN
    -- Obtener el costo del boleto del registro eliminado
    CostoBoleto = OLD.CostoTotal;

    -- Actualizar el campo "MontoTotal" en la tabla "Venta" restando el
costo del boleto
    UPDATE InfoPasajero.Venta
    SET MontoTotal = MontoTotal - CostoBoleto
    WHERE idVenta = OLD.idVenta;

    -- Obtener el costo del boleto del registro recién insertado
    CostoBoleto = NEW.CostoTotal;

    -- Actualizar el campo "MontoTotal" en la tabla "Venta" sumando el costo
del boleto
    UPDATE InfoPasajero.Venta
    SET MontoTotal = MontoTotal + CostoBoleto
    WHERE idVenta = NEW.idVenta;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER TR_ACTUALIZAMONTOTOTAL1
AFTER INSERT OR UPDATE ON InfoPasajero.Boleto
FOR EACH ROW
EXECUTE FUNCTION TR_ACTUALIZAMONTOTOTAL1_FUNCTION();
```

```sql
-- Trigger 3 --
-- Trigger TR_ACTUALIZAMONTOTOTAL2 --
CREATE OR REPLACE FUNCTION TR_ACTUALIZAMONTOTOTAL2_FUNCTION()
RETURNS TRIGGER AS $$
DECLARE
    CostoBoleto MONEY;
BEGIN
    -- Obtener el costo del boleto del registro eliminado
    CostoBoleto = OLD.CostoTotal;

    -- Actualizar el campo "MontoTotal" en la tabla "Venta" restando el
costo del boleto
    UPDATE InfoPasajero.Venta
    SET MontoTotal = MontoTotal - CostoBoleto
    WHERE idVenta = OLD.idVenta;

    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER TR_ACTUALIZAMONTOTOTAL2
AFTER DELETE ON InfoPasajero.Boleto
FOR EACH ROW
EXECUTE FUNCTION TR_ACTUALIZAMONTOTOTAL2_FUNCTION();


-- Trigger 4 --
CREATE OR REPLACE FUNCTION TR_BANCOTARJETA_FUNCTION()
RETURNS TRIGGER AS $$
DECLARE
    NumeroTarjeta VARCHAR(16);
    BancoI VARCHAR(50);
BEGIN
    -- Obtener el número de tarjeta insertado o actualizado
    NumeroTarjeta := SUBSTRING(CAST(NEW.NumTarjeta AS VARCHAR) FROM
LENGTH(CAST(NEW.NumTarjeta AS VARCHAR)) FOR 1);

    -- Analizar los últimos dígitos para determinar el banco
    CASE NumeroTarjeta
        WHEN '1' THEN BancoI := 'Santander';
        WHEN '2' THEN BancoI := 'Bancomer';
        WHEN '3' THEN BancoI := 'Banorte';
        WHEN '4' THEN BancoI := 'BanBajio';
        WHEN '5' THEN BancoI := 'Scotiabank';
```

```
        WHEN '6' THEN BancoI := 'Banco Azteca';
        ELSE BancoI := 'Desconocido';
    END CASE;

    -- Actualizar el campo Banco en la misma fila de la tabla
    NEW.Banco := BancoI;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER TR_BANCOTARJETA
BEFORE INSERT OR UPDATE ON InfoPasajero.TarjetaPasajero
FOR EACH ROW
EXECUTE FUNCTION TR_BANCOTARJETA_FUNCTION();


-- Triggers 5 --
CREATE OR REPLACE FUNCTION TR_ACTUALIZARFLOTA1_FUNCTION()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.idAerolinea IS DISTINCT FROM OLD.idAerolinea THEN
        -- Restar 1 a la antigua Aerolínea
        UPDATE InfoAerolinea.Aerolinea
        SET FlotaTotal = FlotaTotal - 1,
            NumVuelos = NumVuelos - (SELECT COUNT(*) FROM
InfoAerolinea.Itinerario WHERE idAvion = OLD.idAvion)
        WHERE idAerolinea = OLD.idAerolinea;

        -- Sumar 1 a la nueva Aerolínea
        UPDATE InfoAerolinea.Aerolinea
        SET FlotaTotal = FlotaTotal + 1,
            NumVuelos = NumVuelos + (SELECT COUNT(*) FROM
InfoAerolinea.Itinerario WHERE idAvion = NEW.idAvion)
        WHERE idAerolinea = NEW.idAerolinea;
    END IF;
    RETURN NEW;
END;

$$ LANGUAGE plpgsql;
CREATE TRIGGER TR_ACTUALIZARFLOTA1
AFTER INSERT OR UPDATE OF idAerolinea ON InfoAerolinea.Avion
FOR EACH ROW
EXECUTE FUNCTION TR_ACTUALIZARFLOTA1_FUNCTION();
```

```sql
-- Trigger 6 --
CREATE OR REPLACE FUNCTION TR_ACTUALIZARFLOTA2_FUNCTION()
RETURNS TRIGGER AS $$
BEGIN
    -- Restar 1 a la Aerolínea
    UPDATE InfoAerolinea.Aerolinea
    SET FlotaTotal = FlotaTotal - 1,
        NumVuelos = NumVuelos - (SELECT COUNT(*) FROM
InfoAerolinea.Itinerario WHERE idAvion = OLD.idAvion)
    WHERE idAerolinea = OLD.idAerolinea;

    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER TR_ACTUALIZARFLOTA2
AFTER DELETE ON InfoAerolinea.Avion
FOR EACH ROW
EXECUTE FUNCTION TR_ACTUALIZARFLOTA2_FUNCTION();

-- Trigger 7 --
CREATE OR REPLACE FUNCTION TR_ACTUALIZARNUMVUELOS1_FUNCTION()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.idAvion IS DISTINCT FROM OLD.idAvion THEN
        UPDATE InfoAerolinea.Aerolinea AS T
        SET NumVuelos = T.NumVuelos - 1
        FROM InfoAerolinea.Avion AS S
        WHERE S.idAerolinea = T.idAerolinea AND S.idAvion = OLD.idAvion;

        UPDATE InfoAerolinea.Aerolinea AS T
        SET NumVuelos = T.NumVuelos + 1
        FROM InfoAerolinea.Avion AS S
        WHERE S.idAerolinea = T.idAerolinea AND S.idAvion = NEW.idAvion;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER TR_ACTUALIZARNUMVUELOS1
AFTER INSERT OR UPDATE OF idAvion ON InfoAerolinea.Itinerario
FOR EACH ROW
EXECUTE FUNCTION TR_ACTUALIZARNUMVUELOS1_FUNCTION();

-- Trigger 8 --
```

```sql
CREATE OR REPLACE FUNCTION TR_ACTUALIZARNUMVUELOS2_FUNCTION()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE InfoAerolinea.Aerolinea AS T
    SET NumVuelos = T.NumVuelos - 1
    FROM InfoAerolinea.Avion AS S
    WHERE S.idAerolinea = T.idAerolinea AND S.idAvion = OLD.idAvion;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER TR_ACTUALIZARNUMVUELOS2
AFTER DELETE ON InfoAerolinea.Itinerario
FOR EACH ROW
EXECUTE FUNCTION TR_ACTUALIZARNUMVUELOS2_FUNCTION();

-- Trigger 9 --
CREATE OR REPLACE FUNCTION TR_CALCULAREDADPASAJERO_FUNCTION()
RETURNS TRIGGER AS $$
DECLARE
    edad INT;
BEGIN
    edad := EXTRACT(YEAR FROM age(current_date, NEW.FechaNacimiento));
    NEW.Edad := edad;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER TR_CALCULAREDADPASAJERO
BEFORE INSERT OR UPDATE OF
FechaNacimiento ON InfoPasajero.Pasajero
FOR EACH ROW
EXECUTE FUNCTION TR_CALCULAREDADPASAJERO_FUNCTION();


-- Triggers 10 --
-- Trigger TR_ESTADOASIENTO1 --
CREATE OR REPLACE FUNCTION TR_ESTADOASIENTO1_FUNCTION()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.idAsiento IS DISTINCT FROM OLD.idAsiento THEN
        -- Actualizar el estado del asiento a NO ocupado en el itinerario
pasado
        UPDATE InfoPasajero.Asiento
        SET Ocupado = FALSE
```

```
        WHERE idAsiento = OLD.idAsiento;

        -- Actualizar el estado del asiento a ocupado en el itinerario
específico
        UPDATE InfoPasajero.Asiento
        SET Ocupado = TRUE
        WHERE idAsiento = NEW.idAsiento;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER TR_ESTADOASIENTO1
AFTER INSERT OR UPDATE OF idAsiento ON InfoPasajero.Boleto
FOR EACH ROW
EXECUTE FUNCTION TR_ESTADOASIENTO1_FUNCTION();

-- Trigger 11 --
-- Trigger TR_ESTADOASIENTO2 --
CREATE OR REPLACE FUNCTION TR_ESTADOASIENTO2_FUNCTION()
RETURNS TRIGGER AS $$
BEGIN
    -- Actualizar el estado del asiento a NO OCUPADO en el itinerario pasado
    UPDATE InfoPasajero.Asiento
    SET Ocupado = FALSE
    WHERE idAsiento = OLD.idAsiento;

    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER TR_ESTADOASIENTO2
AFTER DELETE ON InfoPasajero.Boleto
FOR EACH ROW
EXECUTE FUNCTION TR_ESTADOASIENTO2_FUNCTION();


------------------------------------ USUARIOS ---------------------------
----------------------

CREATE ROLE administrador WITH LOGIN PASSWORD '232323ADM';
GRANT CONNECT ON DATABASE "Aeropuerto" TO administrador;
GRANT ALL PRIVILEGES ON DATABASE "Aeropuerto" TO administrador;
-- Otorgar permisos en el esquema InfoAerolinea
GRANT USAGE, CREATE ON SCHEMA InfoAerolinea TO administrador;
```

```
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA InfoAerolinea
TO administrador;
GRANT USAGE ON ALL SEQUENCES IN SCHEMA InfoAerolinea TO administrador;
-- Otorgar permisos en el esquema InfoPasajero
GRANT USAGE, CREATE ON SCHEMA InfoPasajero TO administrador;
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA InfoPasajero TO
administrador;
GRANT USAGE ON ALL SEQUENCES IN SCHEMA InfoPasajero TO administrador;


CREATE ROLE aerolinea_staff WITH LOGIN PASSWORD '235491AERS';
GRANT CONNECT ON DATABASE "Aeropuerto" TO aerolinea_staff;
GRANT USAGE ON SCHEMA InfoAerolinea TO aerolinea_staff;
GRANT USAGE ON SCHEMA InfoPasajero TO aerolinea_staff;
GRANT USAGE ON ALL SEQUENCES IN SCHEMA InfoAerolinea TO aerolinea_staff;
GRANT USAGE ON ALL SEQUENCES IN SCHEMA InfoPasajero TO aerolinea_staff;
GRANT SELECT, INSERT, UPDATE ON ALL TABLES IN SCHEMA InfoAerolinea TO
aerolinea_staff;
GRANT SELECT, UPDATE ON ALL TABLES IN SCHEMA InfoPasajero TO
aerolinea_staff;
GRANT INSERT, DELETE ON ALL TABLES IN SCHEMA InfoAerolinea TO
aerolinea_staff;
REVOKE ALL PRIVILEGES ON TABLE InfoAerolinea.Ciudad FROM aerolinea_staff;


CREATE ROLE pasajero_service WITH LOGIN PASSWORD '094312PASER';
GRANT CONNECT ON DATABASE "Aeropuerto" TO pasajero_service;
GRANT USAGE ON SCHEMA InfoPasajero TO pasajero_service;
GRANT USAGE ON SCHEMA InfoAerolinea TO pasajero_service;
GRANT USAGE ON ALL SEQUENCES IN SCHEMA InfoPasajero TO pasajero_service;
GRANT USAGE ON ALL SEQUENCES IN SCHEMA InfoAerolinea TO pasajero_service;
GRANT SELECT, INSERT, UPDATE ON ALL TABLES IN SCHEMA InfoPasajero TO
pasajero_service;
GRANT SELECT, UPDATE ON ALL TABLES IN SCHEMA InfoAerolinea TO
pasajero_service;
REVOKE DELETE ON ALL TABLES IN SCHEMA InfoPasajero FROM pasajero_service;
REVOKE INSERT, DELETE ON ALL TABLES IN SCHEMA InfoAerolinea FROM
pasajero_service;

------------------------------------ CHECK CONSTRAINT -------------------
----------------------

ALTER TABLE InfoPasajero.Pasajero
ADD CONSTRAINT CHECK_EDAD CHECK (Edad > 0 AND Edad <= 110);
```

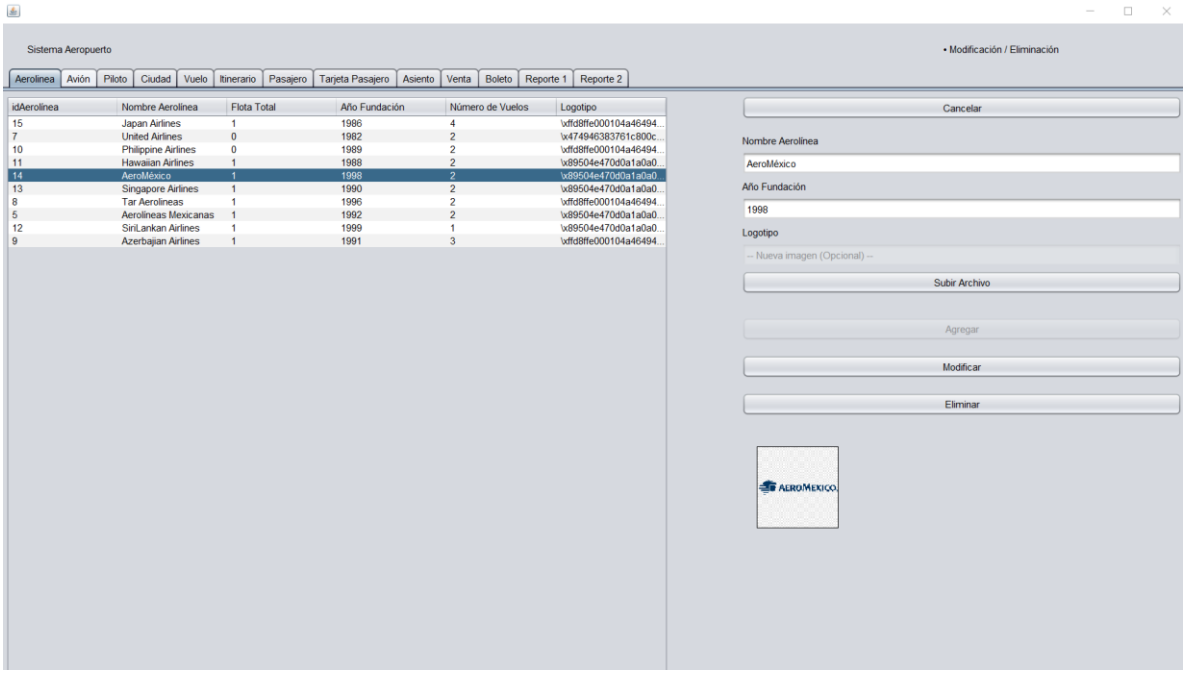
```sql
ALTER TABLE InfoPasajero.TarjetaPasajero
ADD CONSTRAINT CHECK_BANCO CHECK (Banco IN
('Santander','Bancomer','Banorte','BanBajio','Scotiabank','Banco
Azteca','Desconocido'));

ALTER TABLE InfoAerolinea.Piloto
ADD CONSTRAINT CK_EDAD CHECK (FechaNacimiento <= CURRENT_DATE - INTERVAL '25
years');

ALTER TABLE InfoAerolinea.Avion
ADD CONSTRAINT CHK_ANIOFABRICACION
CHECK (AñoFabricacion <= EXTRACT(YEAR FROM CURRENT_DATE) AND AñoFabricacion
>= 1905);
```

## Explicación de las ventanas

En el Proyecto, se tienen ventanas para cada tabla en la base de datos, en donde se nos permite insertar (excepto en la pestaña boleto), modificar y eliminar registros en la tabla seleccionada, se utiliza un *JTabbedPane* para hacer el cambio de pestaña y tabla a manipular en la base de datos




Proyecto ejecutando la pestaña inicial Aerolínea

## Variables de la clase Aeropuerto

```
//Cadenas para establecer la conexion con el servidor, se definen en el
constructor
    private String HOST = "localhost";
    private String PUERTO = "5432";
    private String DB = "Aeropuerto";
    private String USER = "postgres";
    private String PASS = "postgres";
    public String URL = "jdbc:postgresql://" + HOST + ":" + PUERTO + "/" +
DB;
    private Connection CONN = null;
    //Cadena para guardar el ID de un registro que se quiera
modificar/eliminar
    public String currentID = "";
    //Cadena para guardar una imagen en bytes, se usa para hacer la lectura
y mostrar la imagen en un pictureBox
    public byte[] bytesImagen;

    restrictChar listener;
```

```
    // Lista con IDs de itinerarios
    List<Integer> listIDItinerario = new ArrayList<>();
    // Lista con IDs de tarjeta pasajero
    List<Integer> listIDTrjetaPasajero = new ArrayList<>();
    //Lista con IDs de asientos para modificacion; guarda ids de asiento que
aun no estan ocupados en cierto itinerario
    List<Integer> listIDAsiento = new ArrayList<>();
    //Lista con IDs de asientos de datagridview "Boleto"; guarda ids de
asientos de los regsitros boleto
    List<Integer> listIDAsiento2 = new ArrayList<>();
    // Lista con IDs de vuelos
    List<Integer> listIDVuelo = new ArrayList<>();
    // Lista con Nombres de Pilotos
    List<String> listNomPilotos = new ArrayList<>();
```

## Métodos generales de la clase Aeropuerto

### Constructor de la clase Aeropuerto

El constructor es el punto de entrada de la clase **Aeropuerto** en Java. Al ser invocado, inicializa la instancia de la clase y realiza una serie de acciones para preparar el entorno de trabajo. Dentro del constructor, se llevan a cabo las siguientes operaciones:

- **initComponents()**: Este método se encarga de inicializar todos los componentes visuales asociados a la interfaz de usuario del programa.
- **this.listener = new RestrictChar()**: Se crea una instancia de la clase **RestrictChar** para manejar la restricción de caracteres en los campos de texto.
- Asignación de manejadores de eventos a diversos controles de texto (**JTextField**) y un control de número (**JSpinner**). Estos manejadores están encargados de responder a eventos de pulsación de teclas en los respectivos controles, limitando los caracteres que podrán recibir.
- **abrirVentanaLogin()**: Este método es llamado para abrir la ventana de inicio de sesión. Se espera hasta que el nombre de usuario (**USER**) no esté vacío, asegurando que se haya iniciado sesión antes de continuar.
- La estructura condicional verifica si el usuario es un "pasajero_service". En ese caso, establece el índice seleccionado en el objeto **JTabbedPane** (**JTP_Ventanas**) al índice 6. De lo contrario, se realiza una consulta de datos relacionados con la tabla aerolínea mediante el método **consultaDatosAerolinea()**.

```
public AEROPUERTO() {
        initComponents();

        this.listener = new restrictChar();
        tb_NombreAero.addKeyListener(listener);
```

```
        tb_ModeloAvion.addKeyListener(listener);
        tb_NombrePiloto.addKeyListener(listener);
        tb_NombreCiudad.addKeyListener(listener);
        tb_PaisCiudad.addKeyListener(listener);
        tb_NombrePasajero.addKeyListener(listener);
        tb_NacionalidadPasajero.addKeyListener(listener);
        tb_NombreTarPasajero.addKeyListener(listener);

        abrirVentanaLogin();
        while(USER.equals(""))
        {
            abrirVentanaLogin();
        }

        if(USER.equals("pasajero_service"))
        {
            JTP_Ventanas.setSelectedIndex(6);
        }else{
            consultaDatosAerolinea();
        }
    }
```

## Método abrirVentanaLogin

Esta función **abrirVentanaLogin()** en Java crea y muestra una ventana de inicio de sesión.

Abre una ventana de inicio de sesión (**Login**), la centra en la pantalla, la hace visible y luego obtiene el nombre de usuario (**USER**) y la contraseña (**PASS**) ingresados en la ventana de inicio de sesión.

```
private void abrirVentanaLogin() {
    Login ventanaLogin = new Login(this, true);
    ventanaLogin.setLocationRelativeTo(null);  // Centra la ventana en la
pantalla
    ventanaLogin.setVisible(true);
    USER = ventanaLogin.getUsr();
    PASS = ventanaLogin.getPass();
}
```

## Método JTP_VentanasStateChanged

Este método **JTP_VentanasStateChanged** en Java es un manejador de eventos que responde a cambios en el estado del componente **JTabbedPane** llamado **JTP_Ventanas**.

Responde a cambios en el índice seleccionado del componente **JTabbedPane**. Utiliza un bloque de instrucciones **switch** para realizar acciones específicas según la pestaña activa. Gestiona la interfaz de usuario, activando/desactivando controles y realizando consultas a la base de datos

correspondientes a cada caso. Además, realiza verificaciones de acceso basadas en el tipo de usuario (**USER**) y muestra mensajes de acceso denegado cuando es necesario.

```
private void JTP_VentanasStateChanged(javax.swing.event.ChangeEvent evt)
{//GEN-FIRST:event_JTP_VentanasStateChanged
    int selectedIndex = JTP_Ventanas.getSelectedIndex();
    switch (selectedIndex) {
        case 0:
            //AEROLINEA
            if(USER.equals("pasajero_service"))
            {
                JTP_Ventanas.setSelectedIndex(6);
                JOptionPane.showMessageDialog(this, "ACCESO DENEGADO");
                break;
            }
            btn_backAerolineaActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
            consultaDatosAerolinea();
            break;
        case 1:
            //AVIÓN
            if(USER.equals("pasajero_service"))
            {
                JTP_Ventanas.setSelectedIndex(6);
                JOptionPane.showMessageDialog(this, "ACCESO DENEGADO");
                break;
            }
            JCB_AerolineaAvion.removeAllItems();
            LlenarCBavion();
            btn_backAvionActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
            consultaDatosAvion();
            break;
        case 2:
            //PILOTO
            if(USER.equals("pasajero_service"))
            {
                JTP_Ventanas.setSelectedIndex(6);
                JOptionPane.showMessageDialog(this, "ACCESO DENEGADO");
                break;
            }
            JCB_GeneroPiloto.removeAllItems();
            JCB_GeneroPiloto.addItem("Masculino");
            JCB_GeneroPiloto.addItem("Femenino");
            JCB_GeneroPiloto.addItem("Otro");
```

```
            btn_backPilotoActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
            consultaDatosPiloto();
            break;
        case 3:
            //CIUDAD
            if(USER.equals("pasajero_service")||USER.equals("aerolinea_staff
"))
            {
                JTP_Ventanas.setSelectedIndex(6);
                JOptionPane.showMessageDialog(this, "ACCESO DENEGADO");
                break;
            }
            btn_backCiudadActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
            consultaDatosCiudad();
            break;
        case 4:
            //VUELO
            if(USER.equals("pasajero_service"))
            {
                JTP_Ventanas.setSelectedIndex(6);
                JOptionPane.showMessageDialog(this, "ACCESO DENEGADO");
                break;
            }
            JCB_CiudOrgVuelo.removeAllItems();
            JCB_CiudDestVuelo.removeAllItems();
            LlenarCBvuelo();
            btn_backVueloActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
            consultaDatosVuelo();
            break;
        case 5:
            //ITINERARIO
            if(USER.equals("pasajero_service"))
            {
                JTP_Ventanas.setSelectedIndex(6);
                JOptionPane.showMessageDialog(this, "ACCESO DENEGADO");
                break;
            }
            JCB_PilotoItinerario.removeAllItems();
            JCB_VueloItinerario.removeAllItems();
            JCB_AvionItinerario.removeAllItems();
            listIDVuelo.clear();
            LlenarCBitinerario();
```

```
            btn_backItinerarioActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
            consultaDatosItinerario();
            break;
        case 6:
            //PASAJERO
            if(USER.equals("aerolinea_staff"))
            {
                JTP_Ventanas.setSelectedIndex(0);
                JOptionPane.showMessageDialog(this, "ACCESO DENEGADO");
                break;
            }
            JCB_GeneroPasajero.removeAllItems();
            JCB_GeneroPasajero.addItem("Masculino");
            JCB_GeneroPasajero.addItem("Femenino");
            JCB_GeneroPasajero.addItem("Otro");
            btn_backPasajeroActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
            ConsultaDatosPasajero();
            break;
        case 7:
            //TARJETA-PASAJERO
            if(USER.equals("aerolinea_staff"))
            {
                JTP_Ventanas.setSelectedIndex(0);
                JOptionPane.showMessageDialog(this, "ACCESO DENEGADO");
                break;
            }
            JCB_PasajeroTarPasajero.removeAllItems();
            LlenarCBTarjPasaj();
            btn_backTarPasajeroActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
            ConsultaDatosTarPasajero();
            break;
        case 8:
            //ASIENTO
            if(USER.equals("aerolinea_staff"))
            {
                JTP_Ventanas.setSelectedIndex(0);
                JOptionPane.showMessageDialog(this, "ACCESO DENEGADO");
                break;
            }
            JCB_ItinerarioAsiento.removeAllItems();
            JCB_LetraAsiento.removeAllItems();
            JCB_LetraAsiento.addItem("A");
```

```
            JCB_LetraAsiento.addItem("B");
            listIDItinerario.clear();
            llenarCBAsiento();
            btn_backAsientoActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
            ConsultaDatosAsiento();
            break;
        case 9:
            //VENTA
            if(USER.equals("aerolinea_staff"))
            {
                JTP_Ventanas.setSelectedIndex(0);
                JOptionPane.showMessageDialog(this, "ACCESO DENEGADO");
                break;
            }
            JCB_TarjetaVenta.removeAllItems();
            JCB_ItinerarioVenta.removeAllItems();
            listIDItinerario.clear();
            listIDTrjetaPasajero.clear();
            LlenarCBVenta();
            btn_backVentaActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
            ConsultaVenta();
            break;
        case 10:
            //BOLETO
            if(USER.equals("aerolinea_staff"))
            {
                JTP_Ventanas.setSelectedIndex(0);
                JOptionPane.showMessageDialog(this, "ACCESO DENEGADO");
                break;
            }
            backboleto();
            ConsultaBoleto();
            break;
        case 11:
            //Reporte 1
            LlenarCBR1();
            JCB_PilotoR1.setSelectedIndex(-1);

            DefaultTableModel modelR1 = (DefaultTableModel)
JT_R1.getModel();
            modelR1.setRowCount(0);

            TF_HoraSalIR1.setText("00:00");
```

```
            JL_Count.setText("");
            break;
        case 12:
            //Reporte 2
            DefaultTableModel modelR2 = (DefaultTableModel)
JT_R2.getModel();
            modelR2.setRowCount(0);

            tb_CostBaseR2.setText("");
            break;
    }
}//GEN-LAST:event_JTP_VentanasStateChanged
```

### Método ConnectBD

Establece una conexión a la base de datos PostgreSQL utilizando JDBC. La función carga el controlador PostgreSQL (**org.postgresql.Driver**), luego utiliza la clase **DriverManager** para obtener una conexión a la base de datos utilizando la URL de conexión, nombre de usuario (USER), y contraseña (PASS) proporcionados. Si hay algún error durante el proceso, se muestra un cuadro de diálogo emergente con el mensaje de error. Finalmente, la función devuelve el objeto de conexión (**Connection**).

```
public Connection ConnectBD() {
    Connection conne = null;
    try {
        Class.forName("org.postgresql.Driver");
        conne = DriverManager.getConnection(URL, USER, PASS);
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(null, ex);
    }
    return conne;
}
```

### Método Query1

Realiza una consulta de modificación en la base de datos mediante la ejecución de la instrucción SQL proporcionada como parámetro. La función establece una conexión a la base de datos llamando al método **ConnectBD()**. Luego, crea un objeto **Statement** y ejecuta la consulta con **executeUpdate()**. Después de ejecutar la consulta, cierra el **Statement** y la conexión a la base de datos. Si la operación es exitosa, devuelve 1; de lo contrario, muestra un cuadro de diálogo emergente con el mensaje de error y la consulta que causó el problema, y devuelve -1.

```
public int Query1(String Query) {
    try {
        CONN = ConnectBD();
        java.sql.Statement Stmnt = CONN.createStatement();
```

```
        Stmnt.executeUpdate(Query);
        Stmnt.close();
        CONN.close();
        return 1;
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(null, ex+Query);
        return -1;
    }
}
```

### Método Query2

Realiza una consulta SQL que incluye la cláusula **RETURNING** para obtener el ID generado automáticamente después de insertar un registro en la base de datos. La función establece una conexión a la base de datos llamando al método **ConnectBD()**. Luego, crea un objeto **Statement** y ejecuta la consulta con **executeQuery()**. Después de ejecutar la consulta, verifica si hay resultados en el conjunto de resultados (**ResultSet**). Si hay resultados, obtiene el ID generado y cierra los recursos asociados (ResultSet, Statement y la conexión a la base de datos) antes de devolver el ID generado. Si no hay resultados, cierra los recursos y devuelve -1 para indicar que no se pudo obtener el ID. Si hay algún error durante la ejecución, muestra un cuadro de diálogo emergente con el mensaje de error y devuelve -1 para indicar un error en la operación.

```
public int Query2(String Query, String id) {
    try {
        CONN = ConnectBD();
        java.sql.Statement Stmnt = CONN.createStatement();

        // Ejecutar la consulta SQL con el RETURNING para obtener el ID
        ResultSet rs = Stmnt.executeQuery(Query + " RETURNING "+id);

        if (rs.next()) {
            int generatedId = rs.getInt(id);
            rs.close();
            Stmnt.close();
            CONN.close();
            return generatedId;
        } else {
            rs.close();
            Stmnt.close();
            CONN.close();
            return -1; // No se pudo obtener el ID
        }
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(null, ex);
        return -1; // Error en la operación
    }
```

```
}
```

# Métodos para el esquema InfoAerolinea

## Ventana Aerolínea

### Método consultaDatosAerolinea

El método **consultaDatosAerolinea** se encarga de realizar una consulta a la base de datos para obtener información relacionada con las aerolíneas. Al abrir una conexión a la base de datos, ejecuta una consulta SQL que selecciona datos específicos de la tabla **InfoAerolinea.Aerolinea**. Los resultados de la consulta se procesan y se llenan las filas de un JTable (**JT_Aerolinea**) con la información obtenida, organizada en columnas como el ID de la aerolínea, el nombre, la flota total, el año de fundación, el número de vuelos y el logotipo.

```
public void consultaDatosAerolinea()
{
    try{
        CONN = ConnectBD();
        java.sql.Statement Stmnt = CONN.createStatement();
        String Query = "SELECT * FROM InfoAerolinea.Aerolinea";
        String[]Data = new String[6];
        ResultSet Columns = Stmnt.executeQuery (Query);
        DefaultTableModel model = (DefaultTableModel)
JT_Aerolinea.getModel();

        // Eliminar todas las filas existentes
        model.setRowCount(0);
        while(Columns.next())
        {
            Data[0]=Columns.getString(1);
            Data[1]=Columns.getString(2);
            Data[2]=Columns.getString(3);
            Data[3]=Columns.getString(4);
            Data[4]=Columns.getString(5);
            Data[5]=Columns.getString(6);
            model.addRow(Data);
        }
        Stmnt.close();
        CONN.close();
    }catch (Exception ex) {
        JOptionPane.showMessageDialog(null, ex);
    }
}
```

## Método JT_AerolineaMouseClicked

El método **JT_AerolineaMouseClicked** responde al evento de clic del mouse en una celda del JTable (**JT_Aerolinea**). Al hacer clic en una celda específica, este método realiza una serie de acciones para preparar la interfaz de usuario para la modificación o eliminación de la información de la aerolínea seleccionada.

Algunas de las acciones realizadas por este método incluyen la visibilidad de ciertos controles, la actualización de campos de texto y la visualización de la imagen asociada a la aerolínea en un Label (**lbl_image**).

```
private void JT_AerolineaMouseClicked(java.awt.event.MouseEvent evt) {//GEN-
FIRST:event_JT_AerolineaMouseClicked
    lbl_Modifica.setVisible(true);
    btn_Agregar_Aerolinea.setEnabled(false);
    btn_Eliminar_Aerolinea.setEnabled(true);
    btn_Modificar_Aerolinea.setEnabled(true);
    btn_backAerolinea.setEnabled(true);

    int filaSeleccionada = JT_Aerolinea.getSelectedRow();
    currentID = JT_Aerolinea.getValueAt(filaSeleccionada, 0).toString();
    String NomAero = JT_Aerolinea.getValueAt(filaSeleccionada,
1).toString();
    String Flota = JT_Aerolinea.getValueAt(filaSeleccionada, 2).toString();
    String AnioF = JT_Aerolinea.getValueAt(filaSeleccionada, 3).toString();
    String NumVuel = JT_Aerolinea.getValueAt(filaSeleccionada,
4).toString();
    String Log = JT_Aerolinea.getValueAt(filaSeleccionada, 5).toString();

    tb_NombreAero.setText(NomAero);
    tb_AnioFAero.setText(AnioF);
    tb_Image.setText( "-- Nueva imagen (Opcional) --");

    // CARGAR IMAGEN
    bytesImagen = ObtenerBytesDesdeBaseDeDatos(currentID);
    if (bytesImagen != null) {
        ImageIcon imagen = new ImageIcon(bytesImagen);
        // Escalar la imagen para que se ajuste al JLabel
        imagen = new
ImageIcon(imagen.getImage().getScaledInstance(lbl_image.getWidth(),
lbl_image.getHeight(), Image.SCALE_REPLICATE));
        lbl_image.setIcon(imagen);
    }
}//GEN-LAST:event_JT_AerolineaMouseClicked
```

### Método ObtenerBytesDesdeBaseDeDatos

Realiza una conexión a la base de datos llamando al método **DriverManager.getConnection()** y prepara una declaración SQL utilizando **PreparedStatement**. La consulta selecciona la columna "Logotipo" de la tabla "Aerolinea" donde el ID de la aerolínea coincide con el proporcionado como parámetro (**idAerolinea**). Ejecuta la consulta mediante **executeQuery()**, y si hay resultados en el conjunto de resultados (**ResultSet**), obtiene los bytes de la columna "Logotipo". Luego, cierra los recursos asociados (ResultSet, PreparedStatement y la conexión a la base de datos) antes de devolver el array de bytes que representa la imagen. Si hay algún error durante la ejecución, imprime la traza del error y devuelve **null**.

```
public byte[] ObtenerBytesDesdeBaseDeDatos(String idAerolinea) {
    byte[] bytesImagen = null;
    try {
        Connection conn = DriverManager.getConnection(URL, USER, PASS);
        PreparedStatement pstmt = conn.prepareStatement("SELECT Logotipo
FROM InfoAerolinea.Aerolinea WHERE idAerolinea = "+idAerolinea);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            bytesImagen = rs.getBytes("Logotipo");
        }
        rs.close();
        pstmt.close();
        conn.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return bytesImagen;
}
```

### Método btn_backAerolineaActionPerformed

El método **btn_backAerolineaActionPerformed** se encarga de restablecer la interfaz de usuario a un estado predeterminado cuando se hace clic en el botón de retroceso (**btn_backAerolinea**). Este método desactiva ciertos controles, oculta etiquetas y restablece los campos de entrada de texto y la imagen asociada a la aerolínea.

```
private void btn_backAerolineaActionPerformed(java.awt.event.ActionEvent
evt) {//GEN-FIRST:event_btn_backAerolineaActionPerformed
    lbl_Modifica.setVisible(false);
    btn_Agregar_Aerolinea.setEnabled(true);
    btn_Eliminar_Aerolinea.setEnabled(false);
    btn_Modificar_Aerolinea.setEnabled(false);
    btn_backAerolinea.setEnabled(false);
    tb_NombreAero.setText("");
```

```
    tb_AnioFAero.setText("");
    tb_Image.setText("Añade una imagen");
    lbl_image.setText("Imagen");
    lbl_image.setIcon(null);
    bytesImagen = null;
    currentID = "";
    //openFileDialog1.Reset();
}//GEN-LAST:event_btn_backAerolineaActionPerformed
```

### Método btn_Agregar_AerolineaActionPerformed

El método **btn_Agregar_AerolineaActionPerformed** responde al evento de clic en el botón de agregar aerolínea (**btn_Agregar_Aerolinea**). Este método realiza la inserción de una nueva aerolínea en la base de datos, utilizando la información proporcionada en los campos de entrada de texto (**tb_NombreAero**, **tb_AnioFAero** y **tb_Image**).

Verifica que los campos **tb_Nombre**, **tb_AnioF** y **tb_Image** no estén vacíos antes de proceder con la inserción.

Construye una consulta SQL para realizar la inserción en la tabla **InfoAerolinea.Aerolinea**.

Después de la inserción, se actualiza la visualización de las aerolíneas mediante la llamada a **consultaDatosAerolinea**.

Se restablece la interfaz de usuario a un estado predeterminado mediante la llamada a **btn_backAerolineaActionPerformed**

```
private void btn_Agregar_AerolineaActionPerformed(java.awt.event.ActionEvent
evt) {//GEN-FIRST:event_btn_Agregar_AerolineaActionPerformed
    if (tb_NombreAero.getText().equals("") ||
tb_AnioFAero.getText().equals("") || tb_Image.getText().equals("Añade una
imagen")) {
        String mensaje = "Ingresa correctamente los datos";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }
    try {
        // Establecer la conexión a la base de datos
        Connection conn = DriverManager.getConnection(URL, USER, PASS);

        // Consulta SQL para la inserción (con un parámetro para la imagen)
        String query = "INSERT INTO InfoAerolinea.Aerolinea (Nom_Aerolinea,
FlotaTotal, AñoFundacion, NumVuelos, Logotipo) VALUES (?, 0, ?, 0, ?)";

        // Preparar la declaración SQL
```

```
        PreparedStatement pstmt = conn.prepareStatement(query);
        pstmt.setString(1, tb_NombreAero.getText());  // Asignar el nombre
desde el JTextField
        pstmt.setInt(2, Integer.parseInt(tb_AnioFAero.getText()));  //
Convertir a entero y asignar el año desde el JTextField
        pstmt.setBytes(3, bytesImagen);  // Asignar los bytes de la imagen

        // Ejecutar la consulta de inserción
        int filasAfectadas = pstmt.executeUpdate();

        // Verificar si la inserción fue exitosa
        if (filasAfectadas > 0) {
            System.out.println("Inserción exitosa.");
        } else {
            System.out.println("La inserción no fue exitosa.");
        }

        // Cerrar la conexión y la declaración
        pstmt.close();
        conn.close();
    } catch (SQLException e) {
        e.printStackTrace();
        String mensaje = "No se pudo insertar\n";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }
    consultaDatosAerolinea();
    btn_backAerolineaActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
}//GEN-LAST:event_btn_Agregar_AerolineaActionPerformed
```

### Método btn_Modificar_AerolineaActionPerformed

El método **btn_Modificar_AerolineaActionPerformed** responde al evento de clic en el botón de modificar (**btn_Modificar_Aerolinea**). Este método lleva a cabo la actualización de la información de una aerolínea en la base de datos, basándose en los valores proporcionados en los campos de entrada de texto (**tb_NombreAero**, **tb_AnioFAero**). La actualización puede incluir la modificación del logotipo si se proporciona una nueva imagen.

Verifica que los campos **tb_NombreAero**, **tb_AnioFAero** no estén vacíos antes de proceder con la modificación.

Construye una consulta SQL para realizar la actualización en la tabla **InfoAerolinea.Aerolinea**, considerando si se debe o no modificar el logotipo.

Después de la actualización, se actualiza la visualización de las aerolíneas mediante la llamada a **consultaDatosAerolinea**.

Se restablece la interfaz de usuario a un estado predeterminado mediante la llamada a **btn_backAerolineaActionPerformed**

```
private void
btn_Modificar_AerolineaActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_btn_Modificar_AerolineaActionPerformed
    if (tb_NombreAero.getText().equals("") ||
tb_AnioFAero.getText().equals("") || currentID.equals("")) {
        String mensaje = "Selecciona correctamente un registro y sus nuevos
datos";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }

    String query = "";
    if (!tb_Image.getText().equals("-- Nueva imagen (Opcional) --")) {
        query = "UPDATE InfoAerolinea.Aerolinea SET Nom_Aerolinea = ?,
AñoFundacion = ?, Logotipo = ? WHERE idAerolinea = " + currentID;

        try {
            // Establecer la conexión a la base de datos
            Connection conn = DriverManager.getConnection(URL, USER, PASS);
            // Preparar la declaración SQL
            PreparedStatement pstmt = conn.prepareStatement(query);
            pstmt.setString(1, tb_NombreAero.getText());  // Asignar el
nombre desde el JTextField
            pstmt.setInt(2, Integer.parseInt(tb_AnioFAero.getText()));  //
Convertir a entero y asignar el año desde el JTextField
            pstmt.setBytes(3, bytesImagen);  // Asignar los bytes de la
imagen

            // Ejecutar la consulta de inserción
            pstmt.executeUpdate();

            // Cerrar la conexión y la declaración
            pstmt.close();
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
            String mensaje = "No se pudo modificar\n";
            int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
```

```
            JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
            return;
        }
    } else {
        query = "UPDATE InfoAerolinea.Aerolinea SET Nom_Aerolinea = '" +
tb_NombreAero.getText() + "', AñoFundacion = " + tb_AnioFAero.getText() + "
WHERE idAerolinea = " + currentID;
        Query1(query);
    }
    consultaDatosAerolinea();
    btn_backAerolineaActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
}//GEN-LAST:event_btn_Modificar_AerolineaActionPerformed
```

## Método btn_Eliminar_AerolineaActionPerformed

El método **btn_Eliminar_AerolineaActionPerformed** responde al evento de clic en el botón de eliminar (**btn_Eliminar_Aerolinea**). Este método se encarga de eliminar la información de la aerolínea seleccionada de la base de datos.

Verifica que currentID no estén vacíos antes de proceder con la eliminación.

Construye una consulta SQL para realizar la eliminación en la tabla **InfoAerolinea.Aerolinea** utilizando el ID de la aerolínea actual (**currentID**).

Después de la eliminación, se actualiza la visualización de las aerolíneas mediante la llamada a **consultaDatosAerolinea**.

Se restablece la interfaz de usuario a un estado predeterminado mediante la llamada a **btn_backAerolineaActionPerformed.**

```
private void
btn_Eliminar_AerolineaActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_btn_Eliminar_AerolineaActionPerformed
    if (currentID.equals("")) {
        String mensaje = "Selecciona correctamente un registro";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }

    String query = "DELETE FROM InfoAerolinea.Aerolinea WHERE idAerolinea =
" + currentID;
    if(Query1(query)==-1)
    {
```

```
        String mensaje = "No se pudo eliminar porque aparece \ncomo clave
foranea en la tabla avión";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }
    consultaDatosAerolinea();
    btn_backAerolineaActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
}//GEN-LAST:event_btn_Eliminar_AerolineaActionPerformed
```

### Método btn_ImagenActionPerformed

Al hacer clic en el botón, se crea una instancia de **JFileChooser** para permitir al usuario seleccionar un archivo de imagen. El diálogo se configura para mostrar solo archivos con extensiones "jpg" y "png". El resultado de la elección del archivo se verifica, y si el usuario selecciona un archivo (resultado igual a **JFileChooser.APPROVE_OPTION**), se obtiene la referencia al archivo seleccionado.

Se extrae la ruta absoluta del archivo y se asigna al campo de texto **tb_Image**. Luego, se procede a convertir la imagen a bytes. Esto se logra utilizando **FileInputStream** para leer el archivo y **ByteArrayOutputStream** para escribir los bytes en un array de bytes (**bytesImagen**).

```
private void btn_ImagenActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_btn_ImagenActionPerformed
        JFileChooser seleccionarArchivo = new JFileChooser();
        seleccionarArchivo.setDialogTitle("Seleccionar Imagen");
        seleccionarArchivo.setFileFilter(new
FileNameExtensionFilter("Archivos de imagen", "jpg", "png"));
        int resultado = seleccionarArchivo.showOpenDialog(this);

        if (resultado == JFileChooser.APPROVE_OPTION) {
            File archivoSeleccionado = seleccionarArchivo.getSelectedFile();
            String path = archivoSeleccionado.getAbsolutePath();

            // Asigna el nombre del archivo seleccionado al JTextField
tb_Image
            tb_Image.setText(path);

            // Convierte la imagen a bytes
            try {
                FileInputStream fis = new
FileInputStream(archivoSeleccionado);
                ByteArrayOutputStream bos = new ByteArrayOutputStream();
                byte[] buf = new byte[1024];
```

```
                for (int readNum; (readNum = fis.read(buf)) != -1;) {
                    bos.write(buf, 0, readNum);
                }
                bytesImagen = bos.toByteArray();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }//GEN-LAST:event_btn_ImagenActionPerformed
```

### Método tb_AnioFAeroKeyTyped

El evento **KeyTyped** se utiliza para restringir la entrada de caracteres en el campo de texto a solo dígitos y limitar la longitud a 4 caracteres. Al escribir en el campo, se verifica si el carácter es un número o la tecla de borrar (**\b**). Si no es un número o la tecla de borrar, el evento se consume, evitando que el carácter se muestre en el campo.

Adicionalmente, se verifica si ya hay 4 caracteres en el campo. Si es así, se consume el evento para evitar que se ingresen más caracteres.

```
private void tb_AnioFAeroKeyTyped(java.awt.event.KeyEvent evt) {//GEN-
FIRST:event_tb_AnioFAeroKeyTyped
    char c = evt.getKeyChar();
    // Verificar si el carácter es un número o la tecla de borrar
    if (!Character.isDigit(c) && c != '\b') {
        evt.consume(); // Consumir el evento si no es un número o la tecla
de borrar
    }

    // Verificar si ya hay 4 caracteres en el campo
    if (tb_AnioFAero.getText().length() >= 4) {
        evt.consume(); // Consumir el evento si ya hay 4 caracteres
    }
}//GEN-LAST:event_tb_AnioFAeroKeyTyped
```

## Ventana Piloto

### Método consultaDatosPiloto

El método **consultaDatosPiloto** realiza una consulta a la base de datos para obtener información detallada sobre los pilotos. Al establecer una conexión a la base de datos, ejecuta una consulta SQL que selecciona datos específicos de la tabla **InfoAerolinea.Piloto**. Los resultados de esta consulta se procesan y organizan para poblar las filas de un **DefaultTableModel** llamado **model**. Cada fila en el **DefaultTableModel** representa un piloto y muestra detalles como el ID del piloto, el nombre, el género, la fecha de nacimiento y el número de licencia.

Durante el procesamiento de los resultados de la consulta, la fecha de nacimiento se formatea para mostrarla en el formato "dd/MM/yyyy".

```
public void consultaDatosPiloto()
{
    try{
        CONN = ConnectBD();
        java.sql.Statement Stmnt = CONN.createStatement();
        String Query = "SELECT * FROM InfoAerolinea.Piloto";
        String[]Data = new String[5];
        ResultSet Columns = Stmnt.executeQuery (Query);
        DefaultTableModel model = (DefaultTableModel) JT_Piloto.getModel();

        // Eliminar todas las filas existentes
        model.setRowCount(0);
        while(Columns.next())
        {
            Data[0]=Columns.getString(1);
            Data[1]=Columns.getString(2);
            Data[2]=Columns.getString(3);
            Data[3]=Columns.getString(4);////
            Data[4]=Columns.getString(5);
            model.addRow(Data);
        }
        Stmnt.close();
        CONN.close();
    }catch (Exception ex) {
        JOptionPane.showMessageDialog(null, ex);
    }
}
```

## Método btn_Agregar_PilotoActionPerformed

El método **btn_Agregar_PilotoActionPerformed** responde al evento de clic en el botón de agregar piloto (**btn_Agregar_Piloto**). Su función principal es gestionar la inserción de nuevos pilotos en la base de datos, asegurando que los datos proporcionados cumplan con ciertos criterios.

En términos generales:

- Verifica que los campos **tb_NombrePiloto**, **tb_NumLicPiloto**, **JCB_GeneroPiloto**, **JC_FechaNacPiloto**, y **tb_NumLicPiloto** no estén vacíos o en un estado inválido antes de proceder con la inserción.
- Calcula la fecha de nacimiento del piloto a partir de la información proporcionada en el **JCalendar**.

- Construye una consulta SQL para realizar la inserción en la tabla **InfoAerolinea.Piloto** utilizando la información proporcionada.
- Maneja situaciones de error mediante el uso de mensajes de advertencia, informando al usuario sobre posibles problemas, como la falta de datos, edad insuficiente o la existencia de una licencia duplicada.
- Llama a **consultaDatosPiloto** para reflejar los cambios en la interfaz.
- Restablece la interfaz de usuario a un estado predeterminado mediante la llamada a **btn_backPilotoActionPerformed**.

```
private void btn_Agregar_PilotoActionPerformed(java.awt.event.ActionEvent
evt) {//GEN-FIRST:event_btn_Agregar_PilotoActionPerformed
    if (tb_NombrePiloto.getText().equals("") ||
tb_NumLicPiloto.getText().equals("") || JCB_GeneroPiloto.getSelectedIndex()
== -1 || JC_FechaNacPiloto.getDate() == null ||
tb_NumLicPiloto.getText().length() != 10) {
        String mensaje = "Ingresa correctamente los datos";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }

    JCalendar fechaNac = JC_FechaNacPiloto;
    Date fechaSeleccionada = new Date(fechaNac.getDate().getTime());
    String Query = "INSERT INTO InfoAerolinea.Piloto (Nom_Piloto, Genero,
FechaNacimiento, NumLicencia) VALUES ('" + tb_NombrePiloto.getText() + "','"
+ JCB_GeneroPiloto.getSelectedItem() + "','" + fechaSeleccionada + "'," +
tb_NumLicPiloto.getText() + ")";
    if(Query1(Query)==-1)
    {
        String mensaje = "No se pudo agregar \nLa fecha de nacimiento no
cumple con la restricción de edad ó\nEl numero de licencia se repite con
otro registro";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }
    consultaDatosPiloto();
    btn_backPilotoActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
}//GEN-LAST:event_btn_Agregar_PilotoActionPerformed
```

## Método btn_Modificar_PilotoActionPerformed

El método **btn_Modificar_PilotoActionPerformed** responde al evento de clic en el botón de modificar piloto (**btn_Modificar_Piloto**). Su función principal es gestionar la actualización de la información de un piloto en la base de datos, asegurando que los datos proporcionados cumplan con ciertos criterios.

En términos generales:

- Verifica que los campos **tb_NombrePiloto**, **tb_NumLicPiloto**, **JCB_GeneroPiloto**, **JC_FechaNacPiloto**, y **currentID** no estén vacíos o en un estado inválido antes de proceder con la modificación.
- Calcula la fecha de nacimiento del piloto a partir de la información proporcionada en el **JCalendar**.
- Construye una consulta SQL para realizar la actualización en la tabla **InfoAerolinea.Piloto** utilizando la información proporcionada.
- Maneja situaciones de error mediante el uso de mensajes de advertencia, informando al usuario sobre posibles problemas, como la falta de datos, edad insuficiente o la existencia de una licencia duplicada.
- Llama a **consultaDatosPiloto** para reflejar los cambios en la interfaz.
- Restablece la interfaz de usuario a un estado predeterminado mediante la llamada a **btn_backPilotoActionPerformed**.

```
private void btn_Modificar_PilotoActionPerformed(java.awt.event.ActionEvent
evt) {//GEN-FIRST:event_btn_Modificar_PilotoActionPerformed
    if (tb_NombrePiloto.getText().equals("") ||
tb_NumLicPiloto.getText().equals("") || JCB_GeneroPiloto.getSelectedIndex()
== -1 || JC_FechaNacPiloto.getDate() == null || currentID.equals("") ||
tb_NumLicPiloto.getText().length() != 10) {
        String mensaje = "Selecciona correctamente un registro y sus nuevos
datos";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }

    JCalendar fechaNac = JC_FechaNacPiloto;
    Date fechaSeleccionada = new Date(fechaNac.getDate().getTime());
    String Query = "UPDATE InfoAerolinea.Piloto SET Nom_Piloto = '" +
tb_NombrePiloto.getText() + "', Genero = '" +
JCB_GeneroPiloto.getSelectedItem() + "', FechaNacimiento='" +
fechaSeleccionada + "', NumLicencia=" + tb_NumLicPiloto.getText() + "  WHERE
idPiloto = " + currentID;
```

```
    if(Query1(Query)==-1)
    {
        String mensaje = "No se pudo modificar \nLa fecha de nacimiento no
cumple con la restricción de edad ó\nEl numero de licencia se repite con
otro registro";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }
    consultaDatosPiloto();
    btn_backPilotoActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
}//GEN-LAST:event_btn_Modificar_PilotoActionPerformed
```

## Método btn_Eliminar_PilotoActionPerformed

El método **btn_Eliminar_PilotoActionPerformed** responde al evento de clic en el botón de eliminar piloto (**btn_Eliminar_Piloto**). Su función principal es gestionar la eliminación de la información de un piloto en la base de datos, asegurando que los datos proporcionados cumplan con ciertos criterios.

En términos generales:

- Verifica que la variable **currentID** no esté vacía.
- Construye una consulta SQL para realizar la eliminación en la tabla **InfoAerolinea.Piloto** utilizando el ID del piloto actual (**currentID**).
- Maneja situaciones de error mediante el uso de mensajes de advertencia, informando al usuario sobre la necesidad de seleccionar un registro antes de intentar eliminarlo.
- Llama a **consultaDatosPiloto** para reflejar los cambios en la interfaz, actualizando la visualización de los pilotos.

```
private void btn_Eliminar_PilotoActionPerformed(java.awt.event.ActionEvent
evt) {//GEN-FIRST:event_btn_Eliminar_PilotoActionPerformed
    if (currentID.equals("")) {
        String mensaje = "Selecciona correctamente un registro";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }

    String query = "DELETE FROM InfoAerolinea.Piloto WHERE idPiloto = " +
currentID;
    if(Query1(query)==-1)
```

```
    {
        String mensaje = "No se pudo eliminar porque aparece \ncomo clave
foranea en otra tabla";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }
    consultaDatosPiloto();
    btn_backPilotoActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
}//GEN-LAST:event_btn_Eliminar_PilotoActionPerformed
```

### Método btn_backPilotoActionPerformed

El método **btn_backPilotoActionPerformed** se encarga de restablecer la interfaz de usuario a un estado predeterminado cuando se hace clic en el botón de retroceso (**btn_backPiloto**). Su función principal es desactivar ciertos controles, ocultar etiquetas y restablecer los campos de entrada de texto asociados a la información de un piloto.

```
private void btn_backPilotoActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_btn_backPilotoActionPerformed
    lbl_Modifica.setVisible(false);
    btn_Agregar_Piloto.setEnabled(true);
    btn_Eliminar_Piloto.setEnabled(false);
    btn_Modificar_Piloto.setEnabled(false);
    btn_backPiloto.setEnabled(false);
    tb_NombrePiloto.setText("");
    tb_NumLicPiloto.setText("");
    JCB_GeneroPiloto.setSelectedIndex(-1); // Seleccionar "Ninguno"
    Calendar calendar = Calendar.getInstance();
    java.util.Date fechaActual = calendar.getTime();
    // Establecer la fecha actual en el JDateChooser
    JC_FechaNacPiloto.setDate(fechaActual);
    currentID = "";
}//GEN-LAST:event_btn_backPilotoActionPerformed
```

### Método JT_PilotoMouseClicked

El método **JT_PilotoMouseClicked** responde al evento de clic del mouse en una celda del **JTable** (**JT_Piloto**). Al hacer clic en una celda específica, este método realiza una serie de acciones para preparar la interfaz de usuario para la modificación o eliminación de la información del piloto seleccionado.

```
private void JT_PilotoMouseClicked(java.awt.event.MouseEvent evt) {//GEN-
FIRST:event_JT_PilotoMouseClicked
    lbl_Modifica.setVisible(true);
```

```
    btn_Agregar_Piloto.setEnabled(false);
    btn_Eliminar_Piloto.setEnabled(true);
    btn_Modificar_Piloto.setEnabled(true);
    btn_backPiloto.setEnabled(true);

    int filaSeleccionada = JT_Piloto.getSelectedRow();
    currentID = JT_Piloto.getValueAt(filaSeleccionada, 0).toString();
    String NomPilot = JT_Piloto.getValueAt(filaSeleccionada, 1).toString();
    String Genero = JT_Piloto.getValueAt(filaSeleccionada, 2).toString();
    String FechaNac = JT_Piloto.getValueAt(filaSeleccionada, 3).toString();
    String NumLic = JT_Piloto.getValueAt(filaSeleccionada, 4).toString();

    SimpleDateFormat formatoFecha = new SimpleDateFormat("yyyy-MM-dd");
    java.util.Date fechaSeleccionada = null;
    try {
        fechaSeleccionada = formatoFecha.parse(FechaNac);
    } catch (ParseException ex) {
        Logger.getLogger(AEROPUERTO.class.getName()).log(Level.SEVERE, null,
ex);
    }

    tb_NombrePiloto.setText(NomPilot);
    tb_NumLicPiloto.setText(NumLic);
    JC_FechaNacPiloto.setDate(fechaSeleccionada);
    for (int i = 0; i < JCB_GeneroPiloto.getItemCount(); i++) {
        String item = (String) JCB_GeneroPiloto.getItemAt(i);

        if (item.equals(Genero)) {
            JCB_GeneroPiloto.setSelectedIndex(i);
            break;
        }
    }
}//GEN-LAST:event_JT_PilotoMouseClicked
```

### Método tb_NumLicPilotoKeyTyped

El método **tb_NumLicPilotoKeyTyped** responde al evento de escribir en el campo de texto **tb_NumLicPiloto**. Su función principal es verificar si el carácter es un número o la tecla de borrar. Además, verifica si ya hay 10 caracteres en el campo.

```
private void tb_NumLicPilotoKeyTyped(java.awt.event.KeyEvent evt) {//GEN-
FIRST:event_tb_NumLicPilotoKeyTyped
    char c = evt.getKeyChar();
    // Verificar si el carácter es un número o la tecla de borrar
    if (!Character.isDigit(c) && c != '\b') {
```

```
        evt.consume(); // Consumir el evento si no es un número o la tecla
de borrar
    }

    // Verificar si ya hay 10 caracteres en el campo
    if (tb_NumLicPiloto.getText().length() >= 10) {
        evt.consume(); // Consumir el evento si ya hay 10 caracteres
    }
}//GEN-LAST:event_tb_NumLicPilotoKeyTyped
```

## Ventana Ciudad

### Método btn_Agregar_CiudadActionPerformed

El método btn_Agregar_CiudadActionPerformed responde al evento de clic en el botón de agregar ciudad (btn_Agregar_Ciudad). Este método se encarga de verificar que se hayan ingresado los datos necesarios antes de proceder con la inserción de una nueva ciudad en la base de datos. Si la inserción tiene éxito, actualiza la visualización de las ciudades mediante la llamada al método consultaDatosCiudad, y restablece la interfaz de usuario a un estado predeterminado mediante la llamada al método btn_backCiudadActionPerformed.

```
private void btn_Agregar_CiudadActionPerformed(java.awt.event.ActionEvent
evt) {//GEN-FIRST:event_btn_Agregar_CiudadActionPerformed
    if (tb_NombreCiudad.getText().equals("") ||
tb_PaisCiudad.getText().equals("")) {
        String mensaje = "Selecciona correctamente un registro y sus nuevos
datos";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }

    String Query = "INSERT INTO InfoAerolinea.Ciudad (Nom_Ciudad, Pais)
VALUES ('" + tb_NombreCiudad.getText() + "','" + tb_PaisCiudad.getText() +
"')";
    if(Query1(Query)==-1)
    {
        String mensaje = "No se pudo agregar\nEse registro ya existe";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }
    consultaDatosCiudad();
```

```
    btn_backCiudadActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
}//GEN-LAST:event_btn_Agregar_CiudadActionPerformed
private void btn_Agregar_CiudadActionPerformed(java.awt.event.ActionEvent
evt) {//GEN-FIRST:event_btn_Agregar_CiudadActionPerformed
    if (tb_NombreCiudad.getText().equals("") ||
tb_PaisCiudad.getText().equals("")) {
        String mensaje = "Selecciona correctamente un registro y sus nuevos
datos";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }

    String Query = "INSERT INTO InfoAerolinea.Ciudad (Nom_Ciudad, Pais)
VALUES ('" + tb_NombreCiudad.getText() + "','" + tb_PaisCiudad.getText() +
"')";
    if(Query1(Query)==-1)
    {
        String mensaje = "No se pudo agregar\nEse registro ya existe";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }
    consultaDatosCiudad();
    btn_backCiudadActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
}//GEN-LAST:event_btn_Agregar_CiudadActionPerformed
```

### Método btn_Modificar_CiudadActionPerformed

El método btn_Modificar_CiudadActionPerformed responde al evento de clic en el botón de modificación para las ciudades (btn_Modificar_Ciudad). Este método realiza la actualización de la información de una ciudad en la base de datos, verificando previamente que se hayan ingresado los nuevos valores y que se haya seleccionado un registro. Si la actualización tiene éxito, se llama al método consultaDatosCiudad para actualizar la visualización de las ciudades en el formulario, y luego se restablece la interfaz de usuario a un estado predeterminado mediante la llamada al método btn_backCiudadActionPerformed.

```
private void btn_Modificar_CiudadActionPerformed(java.awt.event.ActionEvent
evt) {//GEN-FIRST:event_btn_Modificar_CiudadActionPerformed
    if (tb_NombreCiudad.getText().equals("") ||
tb_PaisCiudad.getText().equals("") || currentID.equals("")) {
```

```
        String mensaje = "Selecciona correctamente un registro y sus nuevos
datos";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }

    String Query = "UPDATE InfoAerolinea.Ciudad SET Nom_Ciudad = '" +
tb_NombreCiudad.getText() + "', Pais = '" + tb_PaisCiudad.getText() + "'
WHERE idCiudad = " + currentID;
    if(Query1(Query)==-1)
    {
        String mensaje = "No se pudo modificar\nEse registro ya existe";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }
    consultaDatosCiudad();
    btn_backCiudadActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
}//GEN-LAST:event_btn_Modificar_CiudadActionPerformed
```

### Método btn_Elimina_ CiudadActionPerformed

El método btn_Eliminar_CiudadActionPerformed responde al evento de clic en el botón de eliminación para las ciudades (btn_Eliminar_Ciudad). Este método se encarga de verificar que se haya seleccionado un registro antes de proceder con la eliminación. Luego, construye una consulta SQL para eliminar la información de la ciudad seleccionada de la tabla InfoAerolinea.Ciudad. Si la eliminación tiene éxito, actualiza la visualización de las ciudades mediante la llamada al método consultaDatosCiudad, y restablece la interfaz de usuario a un estado predeterminado mediante la llamada al método btn_backCiudadActionPerformed.

```
private void btn_Eliminar_CiudadActionPerformed(java.awt.event.ActionEvent
evt) {//GEN-FIRST:event_btn_Eliminar_CiudadActionPerformed
    if (currentID.equals("")) {
        String mensaje = "Selecciona correctamente un registro";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }

    String query = "DELETE FROM InfoAerolinea.Ciudad WHERE idCiudad = " +
currentID;
```

```
    if(Query1(query)==-1)
    {
        String mensaje = "No se pudo eliminar porque aparece \ncomo clave
foranea en otra tabla";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }
    consultaDatosCiudad();
    btn_backCiudadActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
}//GEN-LAST:event_btn_Eliminar_CiudadActionPerformed
```

### Método btn_backCiudadActionPerformed

El método btn_backCiudadActionPerformed se encarga de restablecer la interfaz de usuario a un estado predeterminado cuando se hace clic en el botón de retroceso (btn_backCiudad). Este método oculta ciertos controles, deshabilita y habilita botones según sea necesario, y limpia los campos de entrada de texto.

```
private void btn_backCiudadActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_btn_backCiudadActionPerformed
    lbl_Modifica.setVisible(false);
    btn_Agregar_Ciudad.setEnabled(true);
    btn_Eliminar_Ciudad.setEnabled(false);
    btn_Modificar_Ciudad.setEnabled(false);
    btn_backCiudad.setEnabled(false);
    tb_NombreCiudad.setText("");
    tb_PaisCiudad.setText("");
    currentID = "";
}//GEN-LAST:event_btn_backCiudadActionPerformed
```

### Método JT_CiudadMouseClicked

El método JT_CiudadMouseClicked responde al evento de clic del mouse en una celda de la tabla (JT_Ciudad). Al hacer clic en una celda específica, este método realiza una serie de acciones para preparar la interfaz de usuario para la modificación o eliminación de la información de la ciudad seleccionada. Muestra ciertos controles, deshabilita y habilita botones según sea necesario, y recupera la información de la ciudad seleccionada para mostrarla en campos de entrada de texto.

```
private void JT_CiudadMouseClicked(java.awt.event.MouseEvent evt) {//GEN-
FIRST:event_JT_CiudadMouseClicked
    lbl_Modifica.setVisible(true);
    btn_Agregar_Ciudad.setEnabled(false);
    btn_Eliminar_Ciudad.setEnabled(true);
```

```
    btn_Modificar_Ciudad.setEnabled(true);
    btn_backCiudad.setEnabled(true);

    int filaSeleccionada = JT_Ciudad.getSelectedRow();
    currentID = JT_Ciudad.getValueAt(filaSeleccionada, 0).toString();
    String NomCiudad = JT_Ciudad.getValueAt(filaSeleccionada, 1).toString();
    String Pais = JT_Ciudad.getValueAt(filaSeleccionada, 2).toString();

    tb_NombreCiudad.setText(NomCiudad);
    tb_PaisCiudad.setText(Pais);
}//GEN-LAST:event_JT_CiudadMouseClicked
```

## Método consultaDatosCiudad

El método consultaDatosCiudad en Java realiza una consulta a la base de datos para obtener información relacionada con las ciudades. Abre una conexión a la base de datos, ejecuta una consulta SQL que selecciona todos los datos de la tabla InfoAerolinea.Ciudad, y procesa los resultados para llenar las filas de un componente visual tipo tabla (en este caso, JT_Ciudad) con la información obtenida. Los datos se organizan en columnas, incluyendo el ID de la ciudad, el nombre de la ciudad y el país al que pertenece.

```
public void consultaDatosCiudad()
{
    try{
        CONN = ConnectBD();
        java.sql.Statement Stmnt = CONN.createStatement();
        String Query = "SELECT * FROM InfoAerolinea.Ciudad";
        String[]Data = new String[3];
        ResultSet Columns = Stmnt.executeQuery (Query);
        DefaultTableModel model = (DefaultTableModel) JT_Ciudad.getModel();

        // Eliminar todas las filas existentes
        model.setRowCount(0);
        while(Columns.next())
        {
            Data[0]=Columns.getString(1);
            Data[1]=Columns.getString(2);
            Data[2]=Columns.getString(3);
            model.addRow(Data);
        }
        Stmnt.close();
        CONN.close();
    }catch (Exception ex) {
        JOptionPane.showMessageDialog(null, ex);
    }
}
```

## Ventana Avión

### Método LlenarCBavion

El método LlenarCBavion se encarga de poblar un ComboBox (JCB_AerolineaAvion) con información relacionada a las aerolíneas. Al establecer una conexión con la base de datos, ejecuta una consulta SQL que selecciona el año de fundación y el nombre de las aerolíneas desde la tabla InfoAerolinea.Aerolinea. Los resultados de la consulta se procesan y se añaden al ComboBox en el formato "Nombre de la Aerolínea (Año de Fundación)". Finalmente, se cierra la conexión a la base de datos.

```
public void LlenarCBavion()
    {
        try {
            Connection conn = DriverManager.getConnection(URL, USER, PASS);
            Statement statement = conn.createStatement();
            String query = "SELECT AñoFundacion, Nom_Aerolinea FROM
InfoAerolinea.Aerolinea";
            ResultSet resultSet = statement.executeQuery(query);
            while (resultSet.next()) {
                String aerolineaInfo = resultSet.getString("Nom_Aerolinea")
+ " (" + resultSet.getString("AñoFundacion") + ")";
                JCB_AerolineaAvion.addItem(aerolineaInfo);
            }
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
```

### Método btn_Agregar_AvionActionPerformed

Este método se activa cuando se hace clic en el botón de agregar avión en la interfaz de usuario. Comienza verificando si los campos esenciales (modelo, año de fabricación y aerolínea seleccionada) están completos. Si falta alguna información, se muestra un mensaje informativo. Si la información requerida está presente, se procede a construir una consulta SQL para insertar un nuevo avión en la base de datos ("InfoAerolinea.Avion"). La consulta incluye la información de la aerolínea a la que pertenece el avión, la capacidad (fija en 8 según el código), el modelo, el año de fabricación y el estado de uso. La inclusión del estado de uso (**EstadoUso**) se determina según si la casilla de verificación **JCHBX_ActivoAvion** está seleccionada. Antes de ejecutar la inserción, se verifica si el año de fabricación está dentro de los límites aceptados mediante la llamada a **Query1(query)**. Si hay algún problema, se muestra un mensaje informativo y se cancela la operación de inserción. En caso contrario, se ejecuta la inserción, se refresca la visualización de datos de avión mediante **consultaDatosAvion()**, y se realiza una acción de "regresar" (**btn_backAvionActionPerformed**) para actualizar la interfaz de usuario.

```
private void btn_Agregar_AvionActionPerformed(java.awt.event.ActionEvent
evt) {//GEN-FIRST:event_btn_Agregar_AvionActionPerformed
    if (tb_ModeloAvion.getText().equals("") ||
tb_AnioFAvion.getText().equals("") || JCB_AerolineaAvion.getSelectedIndex()
== -1) {
        String mensaje = "Ingresa correctamente los datos";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }

    String query = "";
    String selectedItem = (String) JCB_AerolineaAvion.getSelectedItem();
    String nomAero = selectedItem.split("\\s+\\(", 2)[0];
    String idAero = getAerolinea(nomAero, false);

    if(JCHBX_ActivoAvion.isSelected())
    {
        query = "INSERT INTO InfoAerolinea.Avion (idAerolinea, Capacidad,
Modelo, AñoFabricacion, EstadoUso) VALUES (" + idAero + ", 8, '" +
tb_ModeloAvion.getText() + "', " + tb_AnioFAvion.getText() + ", TRUE)";
    }else
    {
        query = "INSERT INTO InfoAerolinea.Avion (idAerolinea, Capacidad,
Modelo, AñoFabricacion, EstadoUso) VALUES (" + idAero + ", 8, '" +
tb_ModeloAvion.getText() + "', " + tb_AnioFAvion.getText() + ", FALSE)";
    }

    if(Query1(query)==-1)
    {
        String mensaje = "No se pudo agregar porque \nel año de fabricacion
es esta fuera de los limites";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }
    consultaDatosAvion();
    btn_backAvionActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
}//GEN-LAST:event_btn_Agregar_AvionActionPerformed
```

### Método getAerolinea

Este método realiza una consulta a la base de datos para obtener información sobre una aerolínea, ya sea mediante su nombre o su identificador. La bandera (**flag**) determina el tipo de búsqueda: si es **false**, busca el identificador (**idAerolinea**) a partir del nombre de la aerolínea proporcionado (**val**); si es **true**, busca el nombre de la aerolínea y el año de fundación a partir del identificador proporcionado (**val**). El resultado se devuelve como una cadena formateada, que puede ser el identificador de la aerolínea o el nombre de la aerolínea junto con su año de fundación. Cualquier excepción de SQL que ocurra se imprime en la consola, y el valor de retorno es una cadena vacía si no se encuentra ninguna coincidencia en la base de datos.

```
private String getAerolinea(String val, boolean flag) {
    String aeroValue = "";
    try {
        Connection conexion = DriverManager.getConnection(URL, USER, PASS);
        //conexion.setAutoCommit(false);
        String Query = "";

        if (!flag) {
            Query = "SELECT idAerolinea FROM InfoAerolinea.Aerolinea WHERE
Nom_Aerolinea = '"+val+"'";
        } else {
            Query = "SELECT Nom_Aerolinea, AñoFundacion FROM
InfoAerolinea.Aerolinea WHERE idAerolinea = "+val;
        }

        PreparedStatement pstmt = conexion.prepareStatement(Query);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            if (!flag) {
                aeroValue = rs.getString("idAerolinea");
            } else {
                aeroValue = rs.getString("Nom_Aerolinea") + " (" +
rs.getString("AñoFundacion") + ")";
            }
        }

        rs.close();
        pstmt.close();
        //conexion.commit();
        conexion.close();
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
    return aeroValue;
}
```

## Método btn_Modificar_AvionActionPerformed

Este método se activa cuando se hace clic en el botón de modificar avión en la interfaz de usuario. Inicia verificando si los campos esenciales (modelo, año de fabricación, aerolínea seleccionada y un registro actual) están completos. Si falta alguna información, se muestra un mensaje informativo. Si la información requerida está presente, se procede a construir una consulta SQL para actualizar la información del avión en la base de datos ("InfoAerolinea.Avion"). La consulta incluye la información del modelo, año de fabricación, estado de uso y la aerolínea a la que pertenece el avión. La actualización del estado de uso (**EstadoUso**) se determina según si la casilla de verificación **JCHBX_ActivoAvion** está seleccionada. Antes de ejecutar la actualización, se verifica si el año de fabricación está dentro de los límites aceptados mediante la llamada a **Query1(query)**. Si hay algún problema, se muestra un mensaje informativo y se cancela la operación de modificación. En caso contrario, se ejecuta la actualización, se refresca la visualización de datos de avión mediante **consultaDatosAvion()**, y se realiza una acción de "regresar" (**btn_backAvionActionPerformed**) para actualizar la interfaz de usuario.

```
private void btn_Modificar_AvionActionPerformed(java.awt.event.ActionEvent
evt) {//GEN-FIRST:event_btn_Modificar_AvionActionPerformed
    if (tb_ModeloAvion.getText().equals("") ||
tb_AnioFAvion.getText().equals("") || JCB_AerolineaAvion.getSelectedIndex()
== -1 || currentID.equals("")) {
        String mensaje = "Selecciona correctamente un registro y sus nuevos
datos";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }

    String query = "";
    String selectedItem = (String) JCB_AerolineaAvion.getSelectedItem();
    String nomAero = selectedItem.split("\\s+\\(", 2)[0];
    String idAero = getAerolinea(nomAero, false);

    if(JCHBX_ActivoAvion.isSelected())
    {
        query = "UPDATE InfoAerolinea.Avion SET idAerolinea = " + idAero +
", Modelo = '" + tb_ModeloAvion.getText() + "', AñoFabricacion = " +
tb_AnioFAvion.getText() + ", EstadoUso = TRUE WHERE idAvion = " + currentID;
    }else
    {
        query = "UPDATE InfoAerolinea.Avion SET idAerolinea = " + idAero +
", Modelo = '" + tb_ModeloAvion.getText() + "', AñoFabricacion = " +
tb_AnioFAvion.getText() + ", EstadoUso = FALSE WHERE idAvion = " +
currentID;
```

```
    }

    if(Query1(query)==-1)
    {
        String mensaje = "No se pudo modificar porque \nel año de
fabricacion es esta fuera de los limites";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }
    consultaDatosAvion();
    btn_backAvionActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
}//GEN-LAST:event_btn_Modificar_AvionActionPerformed
```

### Método btn_Eliminar_AvionActionPerformed

Este método se activa cuando se hace clic en el botón de eliminar avión en la interfaz de usuario. Comienza verificando si hay un identificador de avión actualmente seleccionado (**currentID**). Si no hay una selección válida, se muestra un mensaje informativo. Si hay una selección, se construye una consulta SQL para eliminar el avión correspondiente en la base de datos ("InfoAerolinea.Avion") utilizando el identificador actual. Antes de ejecutar la eliminación, se verifica si el avión seleccionado no está siendo utilizado como clave foránea en otras tablas mediante la llamada a **Query1(query)**. Si existe alguna restricción de clave foránea, se muestra un mensaje informativo y se cancela la operación de eliminación. En caso contrario, se ejecuta la eliminación, se actualiza la visualización de datos de avión mediante la llamada a **consultaDatosAvion()**, y se realiza una acción de "regresar" (**btn_backAvionActionPerformed**) para actualizar la interfaz de usuario

```
private void btn_Eliminar_AvionActionPerformed(java.awt.event.ActionEvent
evt) {//GEN-FIRST:event_btn_Eliminar_AvionActionPerformed
    if (currentID.equals("")) {
        String mensaje = "Selecciona correctamente un registro";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }

    String query = "DELETE FROM InfoAerolinea.Avion WHERE idAvion = " +
currentID;
    if(Query1(query)==-1)
    {
        String mensaje = "No se pudo eliminar porque aparece \ncomo clave
foranea en otra tabla";
```

```
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }
    consultaDatosAvion();
    btn_backAvionActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
}//GEN-LAST:event_btn_Eliminar_AvionActionPerformed
```

### Método btn_backAvionActionPerformed

Este método se activa cuando se hace clic en el botón de retroceso en la interfaz de usuario relacionada con la gestión de aviones. Su función principal es restablecer la interfaz a un estado predeterminado después de realizar operaciones como agregar, modificar o eliminar aviones. Oculta un componente de etiqueta (**lbl_Modifica**), habilita o deshabilita ciertos botones según la situación (habilita el botón de agregar y deshabilita los botones de eliminar, modificar y retroceso), y restablece los valores de varios campos y variables a sus valores predeterminados. Este método asegura una interfaz limpia y lista para nuevas operaciones después de realizar cambios en la gestión de aviones.

```
private void btn_backAvionActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_btn_backAvionActionPerformed
    lbl_Modifica.setVisible(false);
    btn_Agregar_Avion.setEnabled(true);
    btn_Eliminar_Avion.setEnabled(false);
    btn_Modificar_Avion.setEnabled(false);
    btn_backAvion.setEnabled(false);
    tb_ModeloAvion.setText("");
    tb_AnioFAvion.setText("");
    JCB_AerolineaAvion.setSelectedIndex(-1); // Seleccionar "Ninguno"
    JCHBX_ActivoAvion.setSelected(false);
    currentID = "";
}//GEN-LAST:event_btn_backAvionActionPerformed
```

### Método JT_AvionMouseClicked

Este método se activa cuando se hace clic en una fila de la tabla **JT_Avion** en la interfaz de usuario. Su función principal es actualizar la interfaz para reflejar la información del avión seleccionado. Muestra un componente de etiqueta (**lbl_Modifica**), habilita los botones de eliminar y modificar, y deshabilita el botón de agregar. También habilita el botón de retroceso. Luego, recupera la información de la fila seleccionada, como el identificador del avión (**currentID**), la aerolínea, el modelo, el año de fabricación y el estado de uso. Estos valores se utilizan para llenar los campos correspondientes en la interfaz de usuario (**tb_ModeloAvion**, **tb_AnioFAvion**, **JCHBX_ActivoAvion**, y **JCB_AerolineaAvion**).

```
private void JT_AvionMouseClicked(java.awt.event.MouseEvent evt) {//GEN-
FIRST:event_JT_AvionMouseClicked
    lbl_Modifica.setVisible(true);
    btn_Agregar_Avion.setEnabled(false);
    btn_Eliminar_Avion.setEnabled(true);
    btn_Modificar_Avion.setEnabled(true);
    btn_backAvion.setEnabled(true);

    int filaSeleccionada = JT_Avion.getSelectedRow();
    currentID = JT_Avion.getValueAt(filaSeleccionada, 0).toString();
    String Aero = JT_Avion.getValueAt(filaSeleccionada, 1).toString();
    String Modelo = JT_Avion.getValueAt(filaSeleccionada, 3).toString();
    String AnioFab = JT_Avion.getValueAt(filaSeleccionada, 4).toString();
    String Activo = JT_Avion.getValueAt(filaSeleccionada, 5).toString();

    tb_ModeloAvion.setText(Modelo);
    tb_AnioFAvion.setText(AnioFab);
    if(Activo.equals("f"))
    {
        JCHBX_ActivoAvion.setSelected(false);
    }else
    {
        JCHBX_ActivoAvion.setSelected(true);
    }

    for (int i = 0; i < JCB_AerolineaAvion.getItemCount(); i++) {
        String item = (String) JCB_AerolineaAvion.getItemAt(i);

        if (item.equals(Aero)) {
            JCB_AerolineaAvion.setSelectedIndex(i);
            break;
        }
    }
}//GEN-LAST:event_JT_AvionMouseClicked
```

### Método ConsultaDatosAvion

Este método realiza una consulta a la base de datos para obtener información detallada sobre los aviones almacenados. Utiliza una conexión establecida a la base de datos y ejecuta una consulta SQL para seleccionar todos los registros de la tabla "InfoAerolinea.Avion". Los resultados se procesan y se insertan en un modelo de tabla (**JT_Avion**) para su presentación. Cualquier excepción que ocurra durante la ejecución se captura y se muestra mediante un cuadro de diálogo de advertencia.

```
public void consultaDatosAvion()
{
```

```
    try{
        CONN = ConnectBD();
        java.sql.Statement Stmnt = CONN.createStatement();
        String Query = "SELECT * FROM InfoAerolinea.Avion";
        String[]Data = new String[6];
        ResultSet Columns = Stmnt.executeQuery (Query);
        DefaultTableModel model = (DefaultTableModel) JT_Avion.getModel();

        // Eliminar todas las filas existentes
        model.setRowCount(0);
        while(Columns.next())
        {
            Data[0]=Columns.getString(1);
            Data[1]=getAerolinea(Columns.getString(2), true);
            Data[2]=Columns.getString(3);
            Data[3]=Columns.getString(4);
            Data[4]=Columns.getString(5);
            Data[5]=Columns.getString(6);
            model.addRow(Data);
        }
        Stmnt.close();
        CONN.close();
    }catch (Exception ex) {
        JOptionPane.showMessageDialog(null, ex);
    }
}
```

## Método tb_AnioFAvionKeyTyped

Este método se activa cuando se realiza una pulsación de tecla en el campo de año de fabricación (**tb_AnioFAvion**). Su función principal es garantizar que solo se ingresen dígitos y que la longitud del campo no exceda los cuatro caracteres (un año típico). Al detectar el carácter de la tecla presionada, verifica si es un dígito o la tecla de borrar (**\b**). Si el carácter no es un dígito ni la tecla de borrar, consume el evento, evitando que se muestre en el campo. Luego, verifica si la longitud del campo es igual o mayor a cuatro caracteres y consume el evento en caso afirmativo, limitando así la longitud del campo a cuatro caracteres.

```
private void tb_AnioFAvionKeyTyped(java.awt.event.KeyEvent evt) {//GEN-
FIRST:event_tb_AnioFAvionKeyTyped
    char c = evt.getKeyChar();
    // Verificar si el carácter es un número o la tecla de borrar
    if (!Character.isDigit(c) && c != '\b') {
        evt.consume(); // Consumir el evento si no es un número o la tecla
de borrar
    }
```

```
    // Verificar si ya hay 4 caracteres en el campo
    if (tb_AnioFAvion.getText().length() >= 4) {
        evt.consume(); // Consumir el evento si ya hay 4 caracteres
    }
}//GEN-LAST:event_tb_AnioFAvionKeyTyped
```

## Ventana Vuelo

### Método LlenarCBvuelo

Este método se encarga de llenar los JComboBox (**JCB_CiudOrgVuelo** y **JCB_CiudDestVuelo**) con información sobre ciudades disponibles para vuelos. Establece una conexión a la base de datos y ejecuta una consulta SQL para seleccionar los nombres de las ciudades y sus respectivos países desde la tabla "InfoAerolinea.Ciudad". Luego, recorre los resultados de la consulta y construye una cadena que combina el nombre de la ciudad y su país. Estas cadenas se agregan como elementos a los JComboBox de ciudades de origen y destino. Si se produce alguna excepción durante el proceso, se imprime la traza de la excepción.

```
public void LlenarCBvuelo()
{
    try {
        Connection conn = DriverManager.getConnection(URL, USER, PASS);
        Statement statement = conn.createStatement();
        String query = "SELECT Nom_Ciudad, Pais FROM InfoAerolinea.Ciudad";
        ResultSet resultSet = statement.executeQuery(query);
        while (resultSet.next()) {
            String CiudadInfo = resultSet.getString("Nom_Ciudad") + " - " +
resultSet.getString("Pais");
            JCB_CiudOrgVuelo.addItem(CiudadInfo);
            JCB_CiudDestVuelo.addItem(CiudadInfo);
        }
        conn.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

### Método btn_Agregar_VueloActionPerformed

Este método se activa cuando se hace clic en el botón de agregar vuelo en la interfaz de usuario. Comienza verificando si los campos esenciales (costo base, ciudades de origen y destino seleccionadas) están completos y si las ciudades de origen y destino son diferentes. Si falta alguna información o las ciudades de origen y destino son iguales, se muestra un mensaje informativo. Si la información requerida está presente, se procede a construir una consulta SQL para insertar un nuevo vuelo en la base de datos ("InfoAerolinea.Vuelo"). La consulta incluye los identificadores de

las ciudades de origen y destino, la duración del vuelo y el costo base. Antes de ejecutar la inserción, se verifica si las ciudades de origen y destino son diferentes y si el costo base es numérico mediante las funciones **getCiudad** y **Query1**. Si hay algún problema, se muestra un mensaje informativo y se cancela la operación de inserción. En caso contrario, se ejecuta la inserción, se refresca la visualización de datos de vuelo mediante **consultaDatosVuelo()**, y se realiza una acción de "regresar" (**btn_backVueloActionPerformed**) para actualizar la interfaz de usuario.

```
private void btn_Agregar_VueloActionPerformed(java.awt.event.ActionEvent
evt) {//GEN-FIRST:event_btn_Agregar_VueloActionPerformed
    if (tb_CostBaseVuelo.getText().equals("") ||
JCB_CiudOrgVuelo.getSelectedIndex() == JCB_CiudDestVuelo.getSelectedIndex()
|| JCB_CiudDestVuelo.getSelectedIndex() == -1 ||
JCB_CiudOrgVuelo.getSelectedIndex() == -1) {
        String mensaje = "Ingresa correctamente los datos";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }

    //Funcion para obtener el idCiudad a traves del contenido del combobox
    String idCdorg = getCiudad((String) JCB_CiudOrgVuelo.getSelectedItem(),
false);
    //Funcion para obtener el idCiudad a traves del contenido del combobox
    String idCddst = getCiudad((String) JCB_CiudDestVuelo.getSelectedItem(),
false);
    String Query = "INSERT INTO InfoAerolinea.Vuelo (idOrigen, idDestino,
DuracionHoras, CostoBase) VALUES (" + idCdorg + "," + idCddst + "," +
JS_DuracionVuelo.getValue() + ", " + tb_CostBaseVuelo.getText() + ")";
    Query1(Query);
    consultaDatosVuelo();
    btn_backVueloActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
}//GEN-LAST:event_btn_Agregar_VueloActionPerformed
```

## Método getCiudad

Este método se encarga de obtener información sobre una ciudad a partir de su identificador o de su nombre y país. La bandera (**flag**) determina el tipo de búsqueda: si es **false**, busca el identificador (**idCiudad**) a partir del nombre y país proporcionados (**val**); si es **true**, busca el nombre de la ciudad y el país a partir del identificador proporcionado (**val**).

Para la búsqueda por nombre y país, el método divide la cadena **val** en dos partes (nombre de la ciudad y país), construye una consulta SQL y ejecuta la consulta para obtener el identificador de la

ciudad. Para la búsqueda por identificador, simplemente realiza una consulta SQL para obtener el nombre de la ciudad y el país.

La información recuperada se devuelve como una cadena formateada, que puede ser el identificador de la ciudad o el nombre de la ciudad junto con el país. Si ocurre alguna excepción durante el proceso, se imprime la traza de la excepción.

```
public String getCiudad(String val, boolean flag) {
    String cdCalue = "";
    try (Connection conexion = DriverManager.getConnection(URL, USER, PASS))
{
        String consulta;
        if (!flag) {
            String[] partes = val.split("-");
            String ciudad = partes[0].trim();
            String pais = partes[1].trim();

            consulta = "SELECT idCiudad FROM InfoAerolinea.Ciudad WHERE
Nom_Ciudad = '"+ciudad+"' AND Pais = '"+pais+"'";
            PreparedStatement pstmt = conexion.prepareStatement(consulta);
            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) {
                cdCalue = rs.getString("idCiudad");
            }
        } else {
            consulta = "SELECT Nom_Ciudad, Pais FROM InfoAerolinea.Ciudad
WHERE idCiudad = "+val;
            PreparedStatement pstmt = conexion.prepareStatement(consulta);
            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) {
                String nomCiudad = rs.getString("Nom_Ciudad");
                String nomPais = rs.getString("Pais");
                cdCalue = nomCiudad + " - " + nomPais;
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return cdCalue;
}
```

### Método btn_Modificar_VueloActionPerformed

Este método se activa cuando se hace clic en el botón de modificar vuelo en la interfaz de usuario. Comienza verificando si los campos esenciales (costo base, ciudades de origen y destino seleccionadas, duración del vuelo y un registro actual) están completos. Si falta alguna información

o las ciudades de origen y destino son iguales, se muestra un mensaje informativo. Si la información requerida está presente, se procede a construir una consulta SQL para actualizar la información del vuelo en la base de datos ("InfoAerolinea.Vuelo"). La consulta incluye los identificadores de las ciudades de origen y destino, la duración del vuelo y el costo base. Antes de ejecutar la actualización, se verifica si las ciudades de origen y destino son diferentes y si el costo base es numérico mediante las funciones **getCiudad** y **Query1**. Si hay algún problema, se muestra un mensaje informativo y se cancela la operación de modificación. En caso contrario, se ejecuta la actualización, se refresca la visualización de datos de vuelo mediante **consultaDatosVuelo()**, y se realiza una acción de "regresar" (**btn_backVueloActionPerformed**) para actualizar la interfaz de usuario.

```
private void btn_Modificar_VueloActionPerformed(java.awt.event.ActionEvent
evt) {//GEN-FIRST:event_btn_Modificar_VueloActionPerformed
    if (tb_CostBaseVuelo.getText().equals("") ||
JCB_CiudOrgVuelo.getSelectedIndex() == JCB_CiudDestVuelo.getSelectedIndex()
|| JCB_CiudDestVuelo.getSelectedIndex() == -1 ||
JCB_CiudOrgVuelo.getSelectedIndex() == -1 || currentID.equals("")) {
        String mensaje = "Selecciona correctamente un registro y sus nuevos
datos";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }
    //Funcion para obtener el idCiudad a traves del contenido del combobox
    String idCdorg = getCiudad((String) JCB_CiudOrgVuelo.getSelectedItem(),
false);
    //Funcion para obtener el idCiudad a traves del contenido del combobox
    String idCddst = getCiudad((String) JCB_CiudDestVuelo.getSelectedItem(),
false);
    String Query = "UPDATE InfoAerolinea.Vuelo SET idOrigen = " + idCdorg +
", idDestino = " + idCddst + ", DuracionHoras = " +
JS_DuracionVuelo.getValue() + ", CostoBase = " + tb_CostBaseVuelo.getText()
+ " WHERE idVuelo = " + currentID;
    Query1(Query);
    consultaDatosVuelo();
    btn_backVueloActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
}//GEN-LAST:event_btn_Modificar_VueloActionPerformed
```

## Método btn_Eliminar_VueloActionPerformed

Este método se activa cuando se hace clic en el botón de eliminar vuelo en la interfaz de usuario. Comienza verificando si hay un identificador de vuelo actualmente seleccionado (**currentID**). Si no hay una selección válida, se muestra un mensaje informativo. Si hay una selección, se construye una consulta SQL para eliminar el vuelo correspondiente en la base de datos

("InfoAerolinea.Vuelo") utilizando el identificador actual. Antes de ejecutar la eliminación, se verifica si el vuelo seleccionado no está siendo utilizado como clave foránea en otras tablas mediante la llamada a **Query1(query)**. Si existe alguna restricción de clave foránea, se muestra un mensaje informativo y se cancela la operación de eliminación. En caso contrario, se ejecuta la eliminación, se actualiza la visualización de datos de vuelo mediante **consultaDatosVuelo()**, y se realiza una acción de "regresar" (**btn_backVueloActionPerformed**) para actualizar la interfaz de usuario.

```
private void btn_Eliminar_VueloActionPerformed(java.awt.event.ActionEvent
evt) {//GEN-FIRST:event_btn_Eliminar_VueloActionPerformed
    if (currentID.equals("")) {
        String mensaje = "Selecciona correctamente un registro";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }

    String query = "DELETE FROM InfoAerolinea.Vuelo WHERE idVuelo = " +
currentID;
    if(Query1(query)==-1)
    {
        String mensaje = "No se pudo eliminar porque aparece \ncomo clave
foranea en otra tabla";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }
    consultaDatosVuelo();
    btn_backVueloActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
}//GEN-LAST:event_btn_Eliminar_VueloActionPerformed
```

### Método btn_backVueloActionPerformed

Este método se activa al hacer clic en el botón de retroceso (**btn_backVuelo**). Oculta la etiqueta de modificación (**lbl_Modifica**), habilita el botón de agregar vuelo (**btn_Agregar_Vuelo**), y deshabilita los botones de eliminar y modificar vuelo (**btn_Eliminar_Vuelo** y **btn_Modificar_Vuelo**). Limpia los campos y selecciones en la interfaz, reiniciando el estado para nuevas operaciones.

```
private void btn_backVueloActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_btn_backVueloActionPerformed
    lbl_Modifica.setVisible(false);
    btn_Agregar_Vuelo.setEnabled(true);
    btn_Eliminar_Vuelo.setEnabled(false);
```

```
    btn_Modificar_Vuelo.setEnabled(false);
    btn_backVuelo.setEnabled(false);
    tb_CostBaseVuelo.setText("");
    JS_DuracionVuelo.setValue(1);
    JCB_CiudOrgVuelo.setSelectedIndex(-1); // Seleccionar "Ninguno"
    JCB_CiudDestVuelo.setSelectedIndex(-1); // Seleccionar "Ninguno"
    currentID = "";
}//GEN-LAST:event_btn_backVueloActionPerformed
```

### Método JT_VueloMouseClicked

Este método se activa cuando se hace clic en una fila de la tabla **JT_Vuelo** en la interfaz de usuario. Su función principal es actualizar la interfaz para reflejar la información del vuelo seleccionado. Muestra un componente de etiqueta (**lbl_Modifica**), habilita los botones de eliminar y modificar, y deshabilita el botón de agregar. También habilita el botón de retroceso.

Luego, recupera la información de la fila seleccionada, como el identificador del vuelo (**currentID**), las ciudades de origen y destino, la duración del vuelo en horas y el costo base del vuelo. Estos valores se utilizan para llenar los campos correspondientes en la interfaz de usuario (**tb_CostBaseVuelo**, **JS_DuracionVuelo**, **JCB_CiudOrgVuelo** y **JCB_CiudDestVuelo**).

Se realizan ajustes en la presentación de los datos, como limpiar el formato monetario del costo base y convertir la duración del vuelo a un formato adecuado para el componente **JSpinner**. Además, se seleccionan automáticamente en los JComboBox las ciudades de origen y destino correspondientes al vuelo seleccionado.

```
private void JT_VueloMouseClicked(java.awt.event.MouseEvent evt) {//GEN-
FIRST:event_JT_VueloMouseClicked
    lbl_Modifica.setVisible(true);
    btn_Agregar_Vuelo.setEnabled(false);
    btn_Eliminar_Vuelo.setEnabled(true);
    btn_Modificar_Vuelo.setEnabled(true);
    btn_backVuelo.setEnabled(true);

    int filaSeleccionada = JT_Vuelo.getSelectedRow();
    currentID = JT_Vuelo.getValueAt(filaSeleccionada, 0).toString();
    String CDorigen = JT_Vuelo.getValueAt(filaSeleccionada, 1).toString();
    String CDdestino = JT_Vuelo.getValueAt(filaSeleccionada, 2).toString();
    String DuracionHoras = JT_Vuelo.getValueAt(filaSeleccionada,
3).toString();
    String CostoBase = JT_Vuelo.getValueAt(filaSeleccionada, 4).toString();

    String valorLimpio = CostoBase.replaceAll("[\\$,]", "");
    tb_CostBaseVuelo.setText(valorLimpio);

    int valorEntero = Integer.parseInt(DuracionHoras);
    JS_DuracionVuelo.setValue(valorEntero);
```

```
    for (int i = 0; i < JCB_CiudOrgVuelo.getItemCount(); i++) {
        String item = (String) JCB_CiudOrgVuelo.getItemAt(i);

        if (item.equals(CDorigen)) {
            JCB_CiudOrgVuelo.setSelectedIndex(i);
            break;
        }
    }

    for (int i = 0; i < JCB_CiudDestVuelo.getItemCount(); i++) {
        String item = (String) JCB_CiudDestVuelo.getItemAt(i);

        if (item.equals(CDdestino)) {
            JCB_CiudDestVuelo.setSelectedIndex(i);
            break;
        }
    }
}//GEN-LAST:event_JT_VueloMouseClicked
```

## Método consultaDatosVuelo

Este método realiza una consulta a la base de datos para obtener información detallada sobre vuelos almacenados. Utiliza una conexión establecida a la base de datos y ejecuta una consulta SQL para seleccionar todos los registros de la tabla "InfoAerolinea.Vuelo". Los resultados se procesan y se insertan en un modelo de tabla (**JT_Vuelo**) para su presentación. Cada fila en la tabla representa un vuelo e incluye información sobre las ciudades de origen y destino, la fecha de salida y la hora de salida. Cualquier excepción que ocurra durante la ejecución se captura y se muestra mediante un cuadro de diálogo de advertencia.

```
public void consultaDatosVuelo()
{
    try{
        CONN = ConnectBD();
        java.sql.Statement Stmnt = CONN.createStatement();
        String Query = "SELECT * FROM InfoAerolinea.Vuelo";
        String[]Data = new String[5];
        ResultSet Columns = Stmnt.executeQuery (Query);
        DefaultTableModel model = (DefaultTableModel) JT_Vuelo.getModel();

        // Eliminar todas las filas existentes
        model.setRowCount(0);
        while(Columns.next())
```

```
        {
            Data[0]=Columns.getString(1);
            Data[1]=getCiudad(Columns.getString(2),true);
            Data[2]=getCiudad(Columns.getString(3),true);
            Data[3]=Columns.getString(4);
            Data[4]=Columns.getString(5);
            model.addRow(Data);
        }
        Stmnt.close();
        CONN.close();
    }catch (Exception ex) {
        JOptionPane.showMessageDialog(null, ex);
    }
}
```

### Método tb_CostBaseVueloKeyTyped

Este método se activa cada vez que se presiona una tecla en el campo de texto **tb_CostBaseVuelo** de la interfaz de usuario. Su función principal es restringir la entrada de caracteres en este campo para garantizar que solo se ingresen valores válidos para el costo base del vuelo.

```
private void tb_CostBaseVueloKeyTyped(java.awt.event.KeyEvent evt) {//GEN-
FIRST:event_tb_CostBaseVueloKeyTyped
    char c = evt.getKeyChar();

    // Permite solo dígitos y un punto decimal
    if (!Character.isDigit(c) && c != '.') {
        evt.consume();
    }

    // Asegura que solo haya un punto decimal
    if (c == '.' && tb_CostBaseVuelo.getText().contains(".")) {
        evt.consume();
    }

    // Permite solo dos decimales
    String text = tb_CostBaseVuelo.getText();
    if (text.contains(".") && text.split("\\.").length > 1 &&
text.split("\\.")[1].length() >= 2 && c != KeyEvent.VK_BACK_SPACE) {
        evt.consume();
    }
}//GEN-LAST:event_tb_CostBaseVueloKeyTyped
```

## Ventana Itinerario

### Método LlenarCBItinerario

Este método se encarga de llenar los JComboBox en la interfaz de usuario con información relevante para el itinerario:

- **Llenar ComboBox para Piloto:** Se realiza una consulta a la base de datos para obtener la información de los pilotos, y se añade al JComboBox **JCB_PilotoItinerario** el ID del piloto junto con su nombre.
- **Llenar ComboBox para Avión:** Se realiza una consulta a la base de datos para obtener la información de los aviones. Se añaden al JComboBox **JCB_AvionItinerario** los ID del avión, su modelo y la aerolínea a la que pertenece, pero solo se añaden aquellos aviones que están en uso (**EstadoUso** es verdadero).
- **Llenar ComboBox para Vuelo:** Se realiza una consulta a la base de datos para obtener la información de los vuelos. Se añaden al JComboBox **JCB_VueloItinerario** la ciudad de origen y destino de cada vuelo.

```
public void LlenarCBitinerario() {
    try {
        // Conexión a la base de datos (asegúrate de manejar excepciones)
        Connection conn = DriverManager.getConnection(URL, USER, PASS);

        // Llenar ComboBox para Piloto
        String queryPiloto = "SELECT idPiloto, Nom_Piloto FROM
InfoAerolinea.Piloto";
        try (PreparedStatement pstmtPiloto =
conn.prepareStatement(queryPiloto); ResultSet rsPiloto =
pstmtPiloto.executeQuery()) {

            while (rsPiloto.next()) {
                JCB_PilotoItinerario.addItem(rsPiloto.getString("idPiloto")
+ " - " + rsPiloto.getString("Nom_Piloto"));
            }
        }

        // Llenar ComboBox para Avión
        String queryAvion = "SELECT idAvion, idAerolinea, Modelo, EstadoUso
FROM InfoAerolinea.Avion";
        try (PreparedStatement pstmtAvion =
conn.prepareStatement(queryAvion); ResultSet rsAvion =
pstmtAvion.executeQuery()) {
            while (rsAvion.next()) {
                if (rsAvion.getBoolean("EstadoUso")) {
                    String aerolinea =
getAerolinea(rsAvion.getString("idAerolinea"), true);
```

```
                    String[] partes = aerolinea.split("\\(");
                    if (partes.length > 0) {
                        aerolinea = partes[0].trim();
                    }
                    JCB_AvionItinerario.addItem(rsAvion.getString("idAvion")
+ " - " + rsAvion.getString("Modelo") + " - " + aerolinea);
                }
            }
        }

        // Llenar ComboBox para Vuelo
        String queryVuelo = "SELECT idVuelo, idOrigen, idDestino FROM
InfoAerolinea.Vuelo";
        String[] partes;
        try (PreparedStatement pstmtVuelo =
conn.prepareStatement(queryVuelo); ResultSet rsVuelo =
pstmtVuelo.executeQuery()) {

            while (rsVuelo.next()) {
                String origen = getCiudad(rsVuelo.getString("idOrigen"),
true);
                partes = origen.split("-");
                if (partes.length > 0) {
                    origen = partes[0].trim();
                }
                String destino = getCiudad(rsVuelo.getString("idDestino"),
true);
                partes = destino.split("-");
                if (partes.length > 0) {
                    destino = partes[0].trim();
                }
                JCB_VueloItinerario.addItem(origen + " - " + destino);
                listIDVuelo.add(rsVuelo.getInt("idVuelo"));
            }
        }

    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

### Método btn_Agregar_ItinerarioActionPerformed

Este método maneja el evento de agregar un nuevo itinerario. Al igual que el método de modificación, realiza varias verificaciones para asegurarse de que se ingresen correctamente los datos del itinerario y que estos datos sean válidos.

1. Verifica si la hora ingresada (**TF_HoraSalItinerario.getText()**) es válida usando el método **ValidarHora**.
2. Verifica si se ha seleccionado una fecha (**JC_FechaVueloItinerario.getDate()**).
3. Verifica si se ha ingresado la hora de salida (**TF_HoraSalItinerario.getText()**).
4. Verifica si se han seleccionado piloto, vuelo y avión en los respectivos ComboBox (**JCB_PilotoItinerario.getSelectedIndex()**, **JCB_VueloItinerario.getSelectedIndex()**, **JCB_AvionItinerario.getSelectedIndex()**).

Si alguna de estas condiciones no se cumple, se muestra un mensaje informativo indicando al usuario que ingrese correctamente los datos.

Si todas las validaciones son exitosas, se obtienen los IDs correspondientes para piloto, avión y vuelo utilizando funciones como **getPiloto**, **getAvion** y **getVuelo**. Luego, se construye la consulta SQL para insertar un nuevo registro en la tabla **InfoAerolinea.Itinerario**. Antes de ejecutar la consulta, se verifica si la inserción es posible para evitar conflictos, por ejemplo, si se intenta asignar dos vuelos al mismo piloto en el mismo día.

Si la inserción se realiza con éxito, se vuelve a consultar y actualizar los datos del itinerario, y se restablecen los elementos de la interfaz gráfica a su estado predeterminado.

```
private void
btn_Agregar_ItinerarioActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_btn_Agregar_ItinerarioActionPerformed
    if
(!ValidarHora(TF_HoraSalItinerario.getText())||JC_FechaVueloItinerario.getDa
te() ==
null||TF_HoraSalItinerario.getText().equals("")||JCB_PilotoItinerario.getSel
ectedIndex() == -1 || JCB_VueloItinerario.getSelectedIndex() == -1 ||
JCB_AvionItinerario.getSelectedIndex() == -1) {
        String mensaje = "Ingresa correctamente los datos";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }

    JCalendar fechaNac = JC_FechaVueloItinerario;
    Date fechaSeleccionada = new Date(fechaNac.getDate().getTime());
    String PilotoID = getPiloto((String)
JCB_PilotoItinerario.getSelectedItem(), false);
```

```
    String AvionID = getAvion((String)
JCB_AvionItinerario.getSelectedItem(), false);
    String VueloID = getVuelo((String)
JCB_VueloItinerario.getSelectedItem(),false);

    String Query = "INSERT INTO InfoAerolinea.Itinerario (idPiloto, idAvion,
idVuelo, HoraSalida, FechaVuelo) VALUES (" + PilotoID + "," + AvionID + ","
+ VueloID + ", '" + TF_HoraSalItinerario.getText() + "', '"+
fechaSeleccionada + "')";
    if(Query1(Query)==-1)
    {
        String mensaje = "Ocurrió una excepción: \nNo se puede agergar dos
vuelos del mismo piloto en el mismo dia";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }
    consultaDatosItinerario();
    btn_backItinerarioActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
}//GEN-LAST:event_btn_Agregar_ItinerarioActionPerformed
```

### Método btn_Modificar_ItinerarioActionPerformed

Este método maneja el evento de modificación de un itinerario. Primero, realiza varias verificaciones para asegurarse de que se hayan seleccionado correctamente los datos del itinerario y que estos datos sean válidos.

1. Verifica si la hora ingresada (**TF_HoraSalItinerario.getText()**) es válida usando el método **ValidarHora**.
2. Verifica si se ha seleccionado una fecha (**JC_FechaVueloItinerario.getDate()**).
3. Verifica si se ha ingresado la hora de salida (**TF_HoraSalItinerario.getText()**).
4. Verifica si se han seleccionado piloto, vuelo y avión en los respectivos ComboBox (**JCB_PilotoItinerario.getSelectedIndex()**, **JCB_VueloItinerario.getSelectedIndex()**, **JCB_AvionItinerario.getSelectedIndex()**).
5. Verifica si hay un registro actualmente seleccionado (**currentID**).

Si alguna de estas condiciones no se cumple, se muestra un mensaje informativo indicando al usuario que seleccione correctamente un registro y sus nuevos datos.

Si todas las validaciones son exitosas, se obtienen los IDs correspondientes para piloto, avión y vuelo utilizando funciones como **getPiloto**, **getAvion** y **getVuelo**. Luego, se construye la consulta SQL para actualizar los datos del itinerario con los nuevos valores. Antes de ejecutar la consulta, se

verifica si la modificación es posible para evitar conflictos, por ejemplo, si se intenta asignar dos vuelos al mismo piloto en el mismo día.

Si la modificación se realiza con éxito, se vuelve a consultar y actualizar los datos del itinerario, y se restablecen los elementos de la interfaz gráfica a su estado predeterminado.

```
private void
btn_Modificar_ItinerarioActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_btn_Modificar_ItinerarioActionPerformed
    if
(!ValidarHora(TF_HoraSalItinerario.getText())||JC_FechaVueloItinerario.getDa
te() ==
null||TF_HoraSalItinerario.getText().equals("")||JCB_PilotoItinerario.getSel
ectedIndex() == -1 || JCB_VueloItinerario.getSelectedIndex() == -1 ||
JCB_AvionItinerario.getSelectedIndex() == -1|| currentID.equals("")) {
        String mensaje = "Selecciona correctamente un registro y sus nuevos
datos";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }

    JCalendar fechaNac = JC_FechaVueloItinerario;
    Date fechaSeleccionada = new Date(fechaNac.getDate().getTime());
    String PilotoID = getPiloto((String)
JCB_PilotoItinerario.getSelectedItem(), false);
    String AvionID = getAvion((String)
JCB_AvionItinerario.getSelectedItem(), false);
    String VueloID = getVuelo((String)
JCB_VueloItinerario.getSelectedItem(),false);

    String Query = "UPDATE InfoAerolinea.Itinerario SET idPiloto = " +
PilotoID + ", idAvion = " + AvionID + ",idVuelo = " + VueloID + ",
HoraSalida = '" + TF_HoraSalItinerario.getText() + "', FechaVuelo = '"+
fechaSeleccionada + "' WHERE idItinerario = " + currentID;
    if(Query1(Query)==-1)
    {
        String mensaje = "Ocurrió una excepción: \nNo se puede agergar dos
vuelos del mismo piloto en el mismo dia";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }
    consultaDatosItinerario();
```

```
    btn_backItinerarioActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
}//GEN-LAST:event_btn_Modificar_ItinerarioActionPerformed
```

### Método getPiloto

Este método recupera información sobre un piloto. Si **flag** es **false**, extrae el nombre del piloto de una cadena; si es **true**, realiza una consulta a la base de datos utilizando el ID del piloto. La información obtenida se devuelve en un formato específico. Se manejan excepciones de SQL para garantizar una ejecución segura.

```
private String getPiloto(String val, boolean flag) {
    String pilotoValue = "";
    try (Connection conexion = DriverManager.getConnection(URL, USER, PASS))
{
        if (!flag) {
            int indiceGuionInicio = val.indexOf('-');
            pilotoValue = val.substring(0, indiceGuionInicio).trim();
        } else {

            String Query = "SELECT idPiloto, Nom_Piloto FROM
InfoAerolinea.Piloto WHERE idPiloto = "+val;
            PreparedStatement pstmt = conexion.prepareStatement(Query);
            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) {
                pilotoValue = rs.getString("idPiloto") + " - " +
rs.getString("Nom_Piloto");
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return pilotoValue;
}
```

### Método getAvion

Este método obtiene información sobre un avión, ya sea extrayendo el id de una cadena o consultando la base de datos por su ID. Si **flag** es **false**, extrae el id de la cadena; si es **true**, realiza una consulta a la base de datos y obtiene detalles adicionales, incluido el nombre de la aerolínea asociada al avión. Se manejan excepciones de SQL para garantizar una ejecución segura.

```
private String getAvion(String val, boolean flag) {
    String AvionValue = "";
```

```
    try (Connection conexion = DriverManager.getConnection(URL, USER, PASS))
{
        if (!flag) {
            int indiceGuionInicio = val.indexOf('-');
            AvionValue = val.substring(0, indiceGuionInicio).trim();
        } else {

            String Query = "SELECT idAvion,idAerolinea,Modelo FROM
InfoAerolinea.Avion WHERE idAvion="+val;
            PreparedStatement pstmt = conexion.prepareStatement(Query);
            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) {
                String aerolinea = getAerolinea(rs.getString("idAerolinea"),
true);
                String[] partes = aerolinea.split("\\(");
                if (partes.length > 0) {
                    aerolinea = partes[0].trim();
                }
                AvionValue = rs.getString("idAvion") + " - " +
rs.getString("Modelo") + " - " + aerolinea;
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return AvionValue;
}
```

### Método getVuelo

Este método obtiene información sobre un vuelo, ya sea seleccionando el ID del vuelo desde una lista existente o consultando la base de datos por su ID. Proporciona flexibilidad al obtener información detallada del vuelo, incluyendo nombres de ciudades de origen y destino. Se manejan excepciones de SQL para garantizar una ejecución segura.

```
private String getVuelo(String val, boolean flag) {
    String VueloValue = "";
    try (Connection conexion = DriverManager.getConnection(URL, USER, PASS))
{
        if (!flag) {
            VueloValue =
listIDVuelo.get(JCB_VueloItinerario.getSelectedIndex()).toString();
        } else {
            String[] partes;
            String Query = "SELECT idOrigen, idDestino FROM
InfoAerolinea.Vuelo WHERE idVuelo=" + val;
```

```
            PreparedStatement pstmt = conexion.prepareStatement(Query);
            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) {
                String origen = getCiudad(rs.getString("idOrigen"), true);
                partes = origen.split("-");
                if (partes.length > 0) {
                    origen = partes[0].trim();
                }
                String destino = getCiudad(rs.getString("idDestino"), true);
                partes = destino.split("-");
                if (partes.length > 0) {
                    destino = partes[0].trim();
                }
                VueloValue = origen + " - " + destino;
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return VueloValue;
}
```

### Método btn_Eliminar_ItinerarioActionPerformed

Este método maneja el evento de eliminación de un itinerario. Primero, verifica si se ha seleccionado correctamente un registro. Si no hay un registro seleccionado, se muestra un mensaje de información indicando al usuario que seleccione correctamente un registro.

Luego, se construye una consulta SQL para eliminar el itinerario correspondiente usando el **currentID**. Antes de ejecutar la consulta, se verifica si la eliminación es posible para evitar violaciones de clave foránea. Si la eliminación no es posible debido a que el itinerario está siendo referenciado en otra tabla, se muestra un mensaje informativo y se interrumpe la operación.

Si la eliminación se realiza con éxito, se vuelve a consultar y actualizar los datos del itinerario, y se restablecen los elementos de la interfaz gráfica a su estado predeterminado.

```
private void
btn_Eliminar_ItinerarioActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_btn_Eliminar_ItinerarioActionPerformed
    if (currentID.equals("")) {
        String mensaje = "Selecciona correctamente un registro";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }
```

```
    String query = "DELETE FROM InfoAerolinea.Itinerario WHERE idItinerario
= " + currentID;
    if(Query1(query)==-1)
    {
        String mensaje = "No se pudo eliminar porque aparece \ncomo clave
foranea en otra tabla";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }
    consultaDatosItinerario();
    btn_backItinerarioActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
}//GEN-LAST:event_btn_Eliminar_ItinerarioActionPerformed
```

### Método btn_backItinerarioActionPerformed

Este método gestiona la interfaz del itinerario al retroceder. Restablece los elementos visuales y los campos de entrada de datos a su estado inicial

```
private void btn_backItinerarioActionPerformed(java.awt.event.ActionEvent
evt) {//GEN-FIRST:event_btn_backItinerarioActionPerformed
    lbl_Modifica.setVisible(false);
    btn_Agregar_Itinerario.setEnabled(true);
    btn_Eliminar_Itinerario.setEnabled(false);
    btn_Modificar_Itinerario.setEnabled(false);
    btn_backItinerario.setEnabled(false);

    JCB_PilotoItinerario.setSelectedIndex(-1); // Seleccionar "Ninguno"
    JCB_VueloItinerario.setSelectedIndex(-1); // Seleccionar "Ninguno"
    JCB_AvionItinerario.setSelectedIndex(-1); // Seleccionar "Ninguno"
    TF_HoraSalItinerario.setText("");
    Calendar calendar = Calendar.getInstance();
    java.util.Date fechaActual = calendar.getTime();
    JC_FechaVueloItinerario.setDate(fechaActual);
    currentID = "";
}//GEN-LAST:event_btn_backItinerarioActionPerformed
```

### Método JT_ItinerarioMouseClicked

Este método maneja el evento de clic del ratón en la tabla **JT_Itinerario**. Al seleccionar una fila, se activan varios elementos de la interfaz gráfica y se cargan los datos de esa fila en los elementos correspondientes (como ComboBoxes y campos de texto) para que el usuario pueda ver y potencialmente modificar la información asociada a ese itinerario. Se utiliza un formato específico

para la fecha y se ajusta la hora para omitir los minutos. Además, se manejan excepciones para garantizar un comportamiento robusto.

```
private void JT_ItinerarioMouseClicked(java.awt.event.MouseEvent evt)
{//GEN-FIRST:event_JT_ItinerarioMouseClicked
    lbl_Modifica.setVisible(true);
    btn_Agregar_Itinerario.setEnabled(false);
    btn_Eliminar_Itinerario.setEnabled(true);
    btn_Modificar_Itinerario.setEnabled(true);
    btn_backItinerario.setEnabled(true);

    int filaSeleccionada = JT_Itinerario.getSelectedRow();
    currentID = JT_Itinerario.getValueAt(filaSeleccionada, 0).toString();
    String piloto = JT_Itinerario.getValueAt(filaSeleccionada,
1).toString();
    String avion = JT_Itinerario.getValueAt(filaSeleccionada, 2).toString();
    String vuelo = JT_Itinerario.getValueAt(filaSeleccionada, 3).toString();
    String horavuelo = JT_Itinerario.getValueAt(filaSeleccionada,
4).toString();
    String fechavuelo = JT_Itinerario.getValueAt(filaSeleccionada,
5).toString();

    SimpleDateFormat formatoFecha = new SimpleDateFormat("yyyy-MM-dd");
    java.util.Date fechaSeleccionada = null;
    try {
        fechaSeleccionada = formatoFecha.parse(fechavuelo);
    } catch (ParseException ex) {
        Logger.getLogger(AEROPUERTO.class.getName()).log(Level.SEVERE, null,
ex);
    }
    JC_FechaVueloItinerario.setDate(fechaSeleccionada);
    horavuelo = horavuelo.substring(0, horavuelo.lastIndexOf(":"));
    TF_HoraSalItinerario.setText(horavuelo);

    for (int i = 0; i < JCB_PilotoItinerario.getItemCount(); i++) {
        String item = (String) JCB_PilotoItinerario.getItemAt(i);

        if (item.equals(piloto)) {
            JCB_PilotoItinerario.setSelectedIndex(i);
            break;
        }
    }

    for (int i = 0; i < JCB_VueloItinerario.getItemCount(); i++) {
        String item = (String) JCB_VueloItinerario.getItemAt(i);
```

```
        if (item.equals(vuelo)) {
            JCB_VueloItinerario.setSelectedIndex(i);
            break;
        }
    }

    for (int i = 0; i < JCB_AvionItinerario.getItemCount(); i++) {
        String item = (String) JCB_AvionItinerario.getItemAt(i);

        if (item.equals(avion)) {
            JCB_AvionItinerario.setSelectedIndex(i);
            break;
        }
    }
}//GEN-LAST:event_JT_ItinerarioMouseClicked
```

### Método consultaDatosItinerario

Este método obtiene y muestra información sobre los itinerarios en una tabla (**JT_Itinerario**). Realiza una consulta a la base de datos, actualiza la interfaz eliminando las filas existentes y añadiendo los resultados, y utiliza métodos auxiliares como **getPiloto**, **getAvion**, y **getVuelo** para obtener datos relacionados. El cierre de recursos y el manejo de excepciones garantizan una ejecución segura.

```
public void consultaDatosItinerario()
{
    try{
        CONN = ConnectBD();
        java.sql.Statement Stmnt = CONN.createStatement();
        String Query = "SELECT * FROM InfoAerolinea.Itinerario";
        String[]Data = new String[6];
        ResultSet Columns = Stmnt.executeQuery (Query);
        DefaultTableModel model = (DefaultTableModel)
JT_Itinerario.getModel();

        // Eliminar todas las filas existentes
        model.setRowCount(0);
        while(Columns.next())
        {
            Data[0]=Columns.getString(1);
            Data[1]=getPiloto(Columns.getString(2), true);
            Data[2]=getAvion(Columns.getString(3), true);
            Data[3]=getVuelo(Columns.getString(4), true);
            Data[4]=Columns.getString(5);
            Data[5]=Columns.getString(6);
            model.addRow(Data);
```

```
        }
        Stmnt.close();
        CONN.close();
    }catch (Exception ex) {
        JOptionPane.showMessageDialog(null, ex);
    }
}
```

### Método ValidarHora

Este método verifica si una cadena de texto representa una hora válida en formato de 24 horas. Divide la cadena en horas y minutos, y valida que las horas no superen 24 y los minutos no superen 59. En caso de cualquier discrepancia, muestra un mensaje de error y retorna **false**; de lo contrario, retorna **true**. Proporciona una validación clara y efectiva para horas ingresadas.

```
private boolean ValidarHora(String val) {
    String horaText = val;
    String[] partes = horaText.split(":");

    if (partes.length == 2) {
        int horas = Integer.parseInt(partes[0]);
        int minutos = Integer.parseInt(partes[1]);

        if (horas > 24 || minutos > 59) {
            JOptionPane.showMessageDialog(null, "Por favor, ingrese una hora
válida.");
            return false;
        }
    } else {
        JOptionPane.showMessageDialog(null, "Por favor, ingrese una hora
válida.");
        return false;
    }
    return true;
}
```

### Método TF_HoraSalItinerarioKeyTyped

Este método gestiona el evento de tecla presionada en el campo de texto **TF_HoraSalItinerario** para garantizar que solo se ingresen dígitos y dos puntos, y verifica el formato HH:MM.

```
private void TF_HoraSalItinerarioKeyTyped(java.awt.event.KeyEvent evt)
{//GEN-FIRST:event_TF_HoraSalItinerarioKeyTyped
    char c = evt.getKeyChar();
    String horaActual = TF_HoraSalItinerario.getText();

    // Asegura que solo se ingresen dígitos y dos puntos
```

```
    if (!Character.isDigit(c) && c != ':' || horaActual.length() >= 5) {
        evt.consume();
    }

    // Verifica el formato HH:MM
    if (Character.isDigit(c) && horaActual.length() == 2) {
        TF_HoraSalItinerario.setText(horaActual + ":");
    }

    // Asegura que solo haya un punto y se encuentre en la posición correcta
    if (c == ':' && (horaActual.contains(":") || horaActual.length() >= 5))
{
        evt.consume();
    }
}//GEN-LAST:event_TF_HoraSalItinerarioKeyTyped
```

# Métodos para el esquema InfoPasajero

## Ventana Pasajero

### Método btn_Agregar_PasajeroActionPerformed

El método **btn_Agregar_PasajeroActionPerformed** se activa cuando se hace clic en el botón de agregar pasajero. Realiza varias validaciones, incluyendo la comprobación de un patrón de correo electrónico, longitud de números de pasaporte, teléfono y contacto de emergencia, así como la selección de elementos en campos obligatorios. Después de pasar las validaciones, construye y ejecuta una consulta SQL para insertar un nuevo registro en la tabla **InfoPasajero.Pasajero**. Antes de realizar la inserción, verifica si los datos ingresados, como el número de pasaporte, teléfono o correo electrónico, ya pertenecen a otro pasajero o si se está ingresando una edad inválida. Luego, actualiza la tabla de pasajeros llamando a la función **ConsultaDatosPasajero** y realiza acciones para restablecer la interfaz a su estado original.

```
private void btn_Agregar_PasajeroActionPerformed(java.awt.event.ActionEvent
evt) {//GEN-FIRST:event_btn_Agregar_PasajeroActionPerformed
    String pattern = "^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$";
    Pattern regex = Pattern.compile(pattern);
    String email = tb_EmailPasajero.getText();
    Matcher matcher = regex.matcher(email);
    if (!matcher.matches()||tb_NombrePasajero.getText().equals("") ||
tb_NacionalidadPasajero.getText().equals("")
||tb_NumPassPasajero.getText().equals("")
||tb_NumTelPasajero.getText().equals("")
||tb_ContEmergenciaPasajero.getText().equals("")
||tb_EmailPasajero.getText().equals("") ||
JCB_GeneroPasajero.getSelectedIndex() == -1 || JC_FechaNacPasajero.getDate()
== null || tb_NumPassPasajero.getText().length() != 10||
```

```
tb_NumTelPasajero.getText().length() != 10||
tb_ContEmergenciaPasajero.getText().length() != 10) {
        String mensaje = "Ingresa correctamente los datos";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }

    JCalendar fechaNac = JC_FechaNacPasajero;
    Date fechaSeleccionada = new Date(fechaNac.getDate().getTime());
    String Query = "INSERT INTO InfoPasajero.Pasajero (Nom_Pasajero,
FechaNacimiento, Nacionalidad, Genero, NumPasaporte, Telefono,
ContactoEmergencia, Email) VALUES ('" + tb_NombrePasajero.getText() + "','"
+ fechaSeleccionada + "','" + tb_NacionalidadPasajero.getText() + "','" +
JCB_GeneroPasajero.getSelectedItem() + "','" + tb_NumPassPasajero.getText()
+ "'," + tb_NumTelPasajero.getText() + "," +
tb_ContEmergenciaPasajero.getText() + ",'" + email + "')";
    if(Query1(Query)==-1)
    {
        String mensaje = "No se pudo agregar \nSe esta intentando agregar un
Número de Pasaporte, Teléfono o Email \nque pertenece a otro pasajero\no se
tiene una edad invalida";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }
    ConsultaDatosPasajero();
    btn_backPasajeroActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
}//GEN-LAST:event_btn_Agregar_PasajeroActionPerformed
```

### Método btn_Modificar_PasajeroActionPerformed

El método **btn_Modificar_PasajeroActionPerformed** se activa cuando se hace clic en el botón de modificar pasajero. Realiza varias validaciones, incluyendo la comprobación de un patrón de correo electrónico, longitud de números de pasaporte, teléfono y contacto de emergencia, así como la selección de elementos en campos obligatorios. Después de pasar las validaciones, construye y ejecuta una consulta SQL para actualizar el registro correspondiente en la tabla **InfoPasajero.Pasajero**. Antes de realizar la actualización, verifica si los datos ingresados, como el número de pasaporte, teléfono o correo electrónico, ya pertenecen a otro pasajero o si se está ingresando una edad inválida. Luego, actualiza la tabla de pasajeros llamando a la función **ConsultaDatosPasajero** y realiza acciones para restablecer la interfaz a su estado original.

```
private void
btn_Modificar_PasajeroActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_btn_Modificar_PasajeroActionPerformed
    String pattern = "^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$";
    Pattern regex = Pattern.compile(pattern);
    String email = tb_EmailPasajero.getText();
    Matcher matcher = regex.matcher(email);
    if (currentID.equals("")
||!matcher.matches()||tb_NombrePasajero.getText().equals("") ||
tb_NacionalidadPasajero.getText().equals("")
||tb_NumPassPasajero.getText().equals("")
||tb_NumTelPasajero.getText().equals("")
||tb_ContEmergenciaPasajero.getText().equals("")
||tb_EmailPasajero.getText().equals("") ||
JCB_GeneroPasajero.getSelectedIndex() == -1 || JC_FechaNacPasajero.getDate()
== null || tb_NumPassPasajero.getText().length() != 10||
tb_NumTelPasajero.getText().length() != 10||
tb_ContEmergenciaPasajero.getText().length() != 10) {
        String mensaje = "Selecciona correctamente un registro y sus nuevos
datos";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }

    JCalendar fechaNac = JC_FechaNacPasajero;
    Date fechaSeleccionada = new Date(fechaNac.getDate().getTime());
    String Query = "UPDATE InfoPasajero.Pasajero SET Nom_Pasajero = '" +
tb_NombrePasajero.getText() + "', FechaNacimiento = '" + fechaSeleccionada +
"', Nacionalidad='" + tb_NacionalidadPasajero.getText() + "', Genero='" +
JCB_GeneroPasajero.getSelectedItem() + "', NumPasaporte='" +
tb_NumPassPasajero.getText() + "', Telefono=" + tb_NumTelPasajero.getText()
+ ", ContactoEmergencia=" + tb_ContEmergenciaPasajero.getText() + ",
Email='" + email + "' WHERE idPasajero = " + currentID;
    if(Query1(Query)==-1)
    {
        String mensaje = "No se pudo agregar \nSe esta intentando agregar un
Número de Pasaporte, Teléfono o Email \nque pertenece a otro pasajero\no se
tiene una edad invalida";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }
```

```
    ConsultaDatosPasajero();
    btn_backPasajeroActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
}//GEN-LAST:event_btn_Modificar_PasajeroActionPerformed
```

### Método btn_Eliminar_PasajeroActionPerformed

El método **btn_Eliminar_PasajeroActionPerformed** se activa cuando se hace clic en el botón de eliminar pasajero. Comprueba si hay un registro seleccionado y, en caso afirmativo, construye y ejecuta una consulta SQL para eliminar el registro correspondiente de la tabla **InfoPasajero.Pasajero**. Antes de eliminar, verifica si el registro está siendo utilizado como clave foránea en otra tabla. Luego, actualiza la tabla de pasajeros llamando a la función **ConsultaDatosPasajero** y realiza acciones para restablecer la interfaz a su estado original.

```
private void btn_Eliminar_PasajeroActionPerformed(java.awt.event.ActionEvent
evt) {//GEN-FIRST:event_btn_Eliminar_PasajeroActionPerformed
    if (currentID.equals("")) {
        String mensaje = "Selecciona correctamente un registro";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }

    String query = "DELETE FROM InfoPasajero.Pasajero WHERE idPasajero = " +
currentID;
    if(Query1(query)==-1)
    {
        String mensaje = "No se pudo eliminar porque aparece \ncomo clave
foranea en otra tabla";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }
    ConsultaDatosPasajero();
    btn_backPasajeroActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
}//GEN-LAST:event_btn_Eliminar_PasajeroActionPerformed
```

### Método btn_backPasajeroActionPerformed

Este método oculta y habilita componentes de la interfaz gráfica relacionada con la gestión de pasajeros. Restablece valores y deshabilita botones. La función principal es facilitar la transición entre acciones como agregar, eliminar o modificar información de pasajeros.

```
private void btn_backPasajeroActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_btn_backPasajeroActionPerformed
    lbl_Modifica.setVisible(false);
    btn_Agregar_Pasajero.setEnabled(true);
    btn_Eliminar_Pasajero.setEnabled(false);
    btn_Modificar_Pasajero.setEnabled(false);
    btn_backPasajero.setEnabled(false);

    tb_NombrePasajero.setText("");
    Calendar calendar = Calendar.getInstance();
    java.util.Date fechaActual = calendar.getTime();
    JC_FechaNacPasajero.setDate(fechaActual);
    tb_NacionalidadPasajero.setText("");
    JCB_GeneroPasajero.setSelectedIndex(-1); // Seleccionar "Ninguno"
    tb_NumPassPasajero.setText("");
    tb_NumTelPasajero.setText("");
    tb_ContEmergenciaPasajero.setText("");
    tb_EmailPasajero.setText("");

    currentID = "";
}//GEN-LAST:event_btn_backPasajeroActionPerformed
```

### Método JT_PasajeroMouseClicked

Este método se activa cuando el usuario hace clic en una fila de la tabla de pasajeros (**JT_Pasajero**). Su función principal es habilitar y mostrar ciertos componentes de la interfaz gráfica, así como recuperar y mostrar la información de la fila seleccionada en los campos correspondientes. Esto facilita la modificación o eliminación de datos de pasajeros.

```
private void JT_PasajeroMouseClicked(java.awt.event.MouseEvent evt) {//GEN-
FIRST:event_JT_PasajeroMouseClicked
    lbl_Modifica.setVisible(true);
    btn_Agregar_Pasajero.setEnabled(false);
    btn_Eliminar_Pasajero.setEnabled(true);
    btn_Modificar_Pasajero.setEnabled(true);
    btn_backPasajero.setEnabled(true);

    int filaSeleccionada = JT_Pasajero.getSelectedRow();
    currentID = JT_Pasajero.getValueAt(filaSeleccionada, 0).toString();
    String NomPas = JT_Pasajero.getValueAt(filaSeleccionada, 1).toString();
    String FechaNac = JT_Pasajero.getValueAt(filaSeleccionada,
2).toString();
    String Nac = JT_Pasajero.getValueAt(filaSeleccionada, 4).toString();
    String Genero = JT_Pasajero.getValueAt(filaSeleccionada, 5).toString();
    String NumPas = JT_Pasajero.getValueAt(filaSeleccionada, 6).toString();
    String Tel = JT_Pasajero.getValueAt(filaSeleccionada, 7).toString();
```

```
    String ContEm = JT_Pasajero.getValueAt(filaSeleccionada, 8).toString();
    String Email = JT_Pasajero.getValueAt(filaSeleccionada, 9).toString();

    SimpleDateFormat formatoFecha = new SimpleDateFormat("yyyy-MM-dd");
    java.util.Date fechaSeleccionada = null;
    try {
        fechaSeleccionada = formatoFecha.parse(FechaNac);
    } catch (ParseException ex) {
        Logger.getLogger(AEROPUERTO.class.getName()).log(Level.SEVERE, null,
ex);
    }

    tb_NombrePasajero.setText(NomPas);
    JC_FechaNacPasajero.setDate(fechaSeleccionada);
    tb_NacionalidadPasajero.setText(Nac);
    JCB_GeneroPasajero.setSelectedIndex(-1); // Seleccionar "Ninguno"
    tb_NumPassPasajero.setText(NumPas);
    tb_NumTelPasajero.setText(Tel);
    tb_ContEmergenciaPasajero.setText(ContEm);
    tb_EmailPasajero.setText(Email);

    for (int i = 0; i < JCB_GeneroPasajero.getItemCount(); i++) {
        String item = (String) JCB_GeneroPasajero.getItemAt(i);

        if (item.equals(Genero)) {
            JCB_GeneroPasajero.setSelectedIndex(i);
            break;
        }
    }
}//GEN-LAST:event_JT_PasajeroMouseClicked
```

### Método ConsultaDatosPasajero

La función **ConsultaDatosPasajero** realiza una consulta a la base de datos para obtener información sobre pasajeros. Luego, actualiza un modelo de tabla en un componente de tabla, eliminando las filas existentes y agregando los resultados de la consulta. La información recuperada incluye varios atributos de los pasajeros, como nombre, apellido, género, fecha de nacimiento, etc.

```
public void ConsultaDatosPasajero(){
    try{
        CONN = ConnectBD();
        java.sql.Statement Stmnt = CONN.createStatement();
        String Query = "SELECT * FROM InfoPasajero.Pasajero";
        String[]Data = new String[10];
        ResultSet Columns = Stmnt.executeQuery (Query);
```

```
        DefaultTableModel model = (DefaultTableModel)
JT_Pasajero.getModel();

        // Eliminar todas las filas existentes
        model.setRowCount(0);
        while(Columns.next())
        {
            Data[0]=Columns.getString(1);
            Data[1]=Columns.getString(2);
            Data[2]=Columns.getString(3);
            Data[3]=Columns.getString(4);
            Data[4]=Columns.getString(5);
            Data[5]=Columns.getString(6);
            Data[6]=Columns.getString(7);
            Data[7]=Columns.getString(8);
            Data[8]=Columns.getString(9);
            Data[9]=Columns.getString(10);
            model.addRow(Data);
        }
        Stmnt.close();
        CONN.close();
    }catch (Exception ex) {
        JOptionPane.showMessageDialog(null, ex);
    }
}
```

## Método tb_NumPassPasajeroKeyTyped, tb_ContEmergenciaPasajeroKeyTyped y tb_NumTelPasajeroKeyTyped

Estos son eventos de teclado asociados a tres campos de texto (**tb_ContEmergenciaPasajero**, **tb_NumTelPasajero**, **tb_NumPassPasajero**) en una interfaz gráfica. Estos eventos están diseñados para limitar la entrada de caracteres en estos campos de texto. Aquí está un resumen de cada uno:

1. **tb_ContEmergenciaPasajeroKeyTyped:**
    - Este método se activa cada vez que se presiona una tecla en el campo **tb_ContEmergenciaPasajero**.
    - Verifica si el carácter presionado es un número o la tecla de borrar (**\b**).
    - Consume el evento si el carácter no es un número o si ya hay 10 caracteres en el campo.
2. **tb_NumTelPasajeroKeyTyped:**
    - Este método se activa cada vez que se presiona una tecla en el campo **tb_NumTelPasajero**.

- Verifica si el carácter presionado es un número o la tecla de borrar (**\b**).
- Consume el evento si el carácter no es un número o si ya hay 10 caracteres en el campo.

3. **tb_NumPassPasajeroKeyTyped:**
    - Este método se activa cada vez que se presiona una tecla en el campo **tb_NumPassPasajero**.
    - Verifica si el carácter presionado es una letra o un número.
    - Consume el evento si el carácter no es una letra o número, y convierte las letras a mayúsculas.
    - Consume el evento si ya hay 10 caracteres en el campo.

Estos métodos aseguran que solo se ingresen datos válidos y que se cumpla con las restricciones de longitud en los campos correspondientes.

```
private void tb_ContEmergenciaPasajeroKeyTyped(java.awt.event.KeyEvent evt)
{//GEN-FIRST:event_tb_ContEmergenciaPasajeroKeyTyped
    char c = evt.getKeyChar();
    // Verificar si el carácter es un número o la tecla de borrar
    if (!Character.isDigit(c) && c != '\b') {
        evt.consume(); // Consumir el evento si no es un número o la tecla
de borrar
    }

    // Verificar si ya hay 10 caracteres en el campo
    if (tb_ContEmergenciaPasajero.getText().length() >= 10) {
        evt.consume(); // Consumir el evento si ya hay 10 caracteres
    }
}//GEN-LAST:event_tb_ContEmergenciaPasajeroKeyTyped
```

```
private void tb_NumTelPasajeroKeyTyped(java.awt.event.KeyEvent evt) {//GEN-
FIRST:event_tb_NumTelPasajeroKeyTyped
    char c = evt.getKeyChar();
    // Verificar si el carácter es un número o la tecla de borrar
    if (!Character.isDigit(c) && c != '\b') {
        evt.consume(); // Consumir el evento si no es un número o la tecla
de borrar
    }

    // Verificar si ya hay 10 caracteres en el campo
    if (tb_NumTelPasajero.getText().length() >= 10) {
        evt.consume(); // Consumir el evento si ya hay 10 caracteres
    }
```

```
}//GEN-LAST:event_tb_NumTelPasajeroKeyTyped
```

```
private void tb_NumPassPasajeroKeyTyped(java.awt.event.KeyEvent evt) {//GEN-
FIRST:event_tb_NumPassPasajeroKeyTyped
    char c = evt.getKeyChar();

    // Verificar si el carácter es una letra o un número
    if (!Character.isLetterOrDigit(c)) {
        evt.consume(); // Consumir el evento si no es una letra o número
    }

    // Convertir las letras a mayúsculas
    if (Character.isLetter(c)) {
        c = Character.toUpperCase(c);
        evt.setKeyChar(c);
    }

    // Verificar si ya hay 10 caracteres en el campo
    if (tb_NumPassPasajero.getText().length() >= 10) {
        evt.consume(); // Consumir el evento si ya hay 10 caracteres
    }
}//GEN-LAST:event_tb_NumPassPasajeroKeyTyped
```

## Ventana Tarjeta Pasajero

### Método LlenarCBTarjPasaj

Este método popula un JComboBox (**JCB_PasajeroTarPasajero**) con información relacionada a los pasajeros. Extrae el nombre y los últimos cuatro dígitos del número de pasaporte de los pasajeros almacenados en la base de datos. Luego, agrega estos datos al JComboBox para su visualización en la interfaz gráfica. Esto facilita la selección de pasajeros al realizar operaciones relacionadas con tarjetas de pasajero.

```
public void LlenarCBTarjPasaj() {
    try {
        Connection conexion = DriverManager.getConnection(URL, USER, PASS);
        Statement statement = conexion.createStatement();
        String query = "SELECT Nom_Pasajero, NumPasaporte FROM
InfoPasajero.Pasajero";
        ResultSet resultSet = statement.executeQuery(query);

        while (resultSet.next()) {
```

```
            String digitnumpass =
resultSet.getString("NumPasaporte").substring(resultSet.getString("NumPasapo
rte").length() - 4);
            JCB_PasajeroTarPasajero.addItem(digitnumpass + " - " +
resultSet.getString("Nom_Pasajero"));
        }

        resultSet.close();
        statement.close();
        conexion.close();
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
}
```

### Método btn_Agregar_TarPasajeroActionPerformed

Este método se ejecuta al hacer clic en el botón "Agregar" en la sección de tarjetas de pasajero de la interfaz gráfica. Realiza las siguientes acciones:

1. **Validación de Datos:** Verifica que todos los campos obligatorios estén llenos y que los datos ingresados sean válidos en cuanto a la longitud de los campos de número de tarjeta (**tb_NumTarPasajero**), CVV (**tb_CVVTarPasajero**), y la fecha de vencimiento (**TF_FechaVenTarPasajero**).
2. **Obtención de ID de Pasajero:** Utiliza el método **getPasajero** para obtener el ID del pasajero seleccionado en el cuadro combinado **JCB_PasajeroTarPasajero**.
3. **Formato de Fecha:** Convierte la fecha ingresada en el formato correcto ("dd/MM/yyyy" a "yyyy-MM-dd").
4. **Consulta y Agregado:** Realiza una consulta SQL para insertar los datos de la nueva tarjeta de pasajero en la tabla correspondiente (**InfoPasajero.TarjetaPasajero**). Si la consulta es exitosa, actualiza la tabla de datos de tarjetas de pasajero y realiza la acción de volver (**btn_backTarPasajeroActionPerformed**).
5. **Manejo de Excepciones:** En caso de que la tarjeta ya exista, muestra un mensaje de advertencia y no realiza la inserción.

```
private void
btn_Agregar_TarPasajeroActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_btn_Agregar_TarPasajeroActionPerformed
    if (tb_NombreTarPasajero.getText().equals("") ||
tb_NumTarPasajero.getText().equals("") ||
JCB_PasajeroTarPasajero.getSelectedIndex() == -1 ||
tb_CVVTarPasajero.getText().equals("") ||
TF_FechaVenTarPasajero.getText().equals("") ||
tb_NumTarPasajero.getText().length() != 16 ||
```

```
tb_CVVTarPasajero.getText().length() != 3 ||
TF_FechaVenTarPasajero.getText().length() != 7) {
        String mensaje = "Ingresa correctamente los datos";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }

    String PasajeroID = getPasajero((String)
JCB_PasajeroTarPasajero.getSelectedItem(), false);
    String fechaFinal = "";
    try {
        fechaFinal = "01/" + TF_FechaVenTarPasajero.getText();
        SimpleDateFormat formatoEntrada = new
SimpleDateFormat("dd/MM/yyyy");
        SimpleDateFormat formatoSalida = new SimpleDateFormat("yyyy-MM-dd");
        java.util.Date fecha = formatoEntrada.parse(fechaFinal);
        fechaFinal = formatoSalida.format(fecha);
    } catch (ParseException e) {
        e.printStackTrace();
    }

    String Query = "INSERT INTO InfoPasajero.TarjetaPasajero (idPasajero,
NombreTitular, Banco, NumTarjeta, FechaVencimiento, CVV) VALUES (" +
PasajeroID + ",'" + tb_NombreTarPasajero.getText() + "', 'Desconocido', " +
tb_NumTarPasajero.getText() + ", '" + fechaFinal + "', " +
tb_CVVTarPasajero.getText() + ")";
    if(Query1(Query)==-1)
    {
        String mensaje = "No se pudo agregar \nEsa tarjeta ya existe";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }
    ConsultaDatosTarPasajero();
    btn_backTarPasajeroActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
}//GEN-LAST:event_btn_Agregar_TarPasajeroActionPerformed
```

### Método btn_Modificar_TarPasajeroActionPerformed

Este método se ejecuta al hacer clic en el botón "Modificar" en la sección de tarjetas de pasajero de la interfaz gráfica. Aquí está un resumen de lo que hace:

1. **Validación de Datos:** Verifica que se haya seleccionado un registro y que los datos ingresados para la modificación sean válidos en cuanto a la longitud de los campos de número de tarjeta (**tb_NumTarPasajero**), CVV (**tb_CVVTarPasajero**), y la fecha de vencimiento (**TF_FechaVenTarPasajero**).
2. **Obtención de ID de Pasajero:** Utiliza el método **getPasajero** para obtener el ID del pasajero seleccionado en el cuadro combinado **JCB_PasajeroTarPasajero**.
3. **Formato de Fecha:** Convierte la fecha ingresada en el formato correcto ("dd/MM/yyyy" a "yyyy-MM-dd").
4. **Consulta y Actualización:** Realiza una consulta SQL para actualizar los datos de la tarjeta de pasajero en la tabla correspondiente (**InfoPasajero.TarjetaPasajero**). Si la consulta es exitosa, actualiza la tabla de datos de tarjetas de pasajero y realiza la acción de volver (**btn_backTarPasajeroActionPerformed**).
5. **Manejo de Excepciones:** En caso de que la tarjeta ya exista, muestra un mensaje de advertencia y no realiza la actualización.

```
private void
btn_Modificar_TarPasajeroActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_btn_Modificar_TarPasajeroActionPerformed
    if (currentID.equals("") ||tb_NombreTarPasajero.getText().equals("") ||
tb_NumTarPasajero.getText().equals("") ||
JCB_PasajeroTarPasajero.getSelectedIndex() == -1 ||
tb_CVVTarPasajero.getText().equals("") ||
TF_FechaVenTarPasajero.getText().equals("") ||
tb_NumTarPasajero.getText().length() < 16 ||
tb_CVVTarPasajero.getText().length() < 3 ||
TF_FechaVenTarPasajero.getText().length() < 7) {
        String mensaje = "Selecciona correctamente un registro y sus nuevos
datos";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }

    String PasajeroID = getPasajero((String)
JCB_PasajeroTarPasajero.getSelectedItem(), false);
    String fechaFinal = "";
    try {
        fechaFinal = "01/" + TF_FechaVenTarPasajero.getText();
        SimpleDateFormat formatoEntrada = new
SimpleDateFormat("dd/MM/yyyy");
        SimpleDateFormat formatoSalida = new SimpleDateFormat("yyyy-MM-dd");
        java.util.Date fecha = formatoEntrada.parse(fechaFinal);
```

```
        fechaFinal = formatoSalida.format(fecha);
    } catch (ParseException e) {
        e.printStackTrace();
    }

    String Query = "UPDATE InfoPasajero.TarjetaPasajero SET idPasajero = " +
PasajeroID + ", NombreTitular = '" + tb_NombreTarPasajero.getText() + "',
NumTarjeta = " + tb_NumTarPasajero.getText() + ", FechaVencimiento = '" +
fechaFinal + "', CVV = " + tb_CVVTarPasajero.getText() + " WHERE
idTarjetaPasajero = " + currentID;
    if(Query1(Query)==-1)
    {
        String mensaje = "No se pudo agregar \nEsa tarjeta ya existe";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }
    ConsultaDatosTarPasajero();
    btn_backTarPasajeroActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
}//GEN-LAST:event_btn_Modificar_TarPasajeroActionPerformed
```

## Método getPasajero

Este método **getPasajero** se encarga de obtener información del pasajero desde la base de datos, ya sea utilizando el ID del pasajero o un formato específico que combina los últimos 4 dígitos del pasaporte y el nombre del pasajero. Realiza la conexión a la base de datos, ejecuta consultas SQL, procesa los resultados y retorna la información del pasajero en el formato requerido. Este método es útil para recuperar datos del pasajero según el contexto de uso.

```
private String getPasajero(String val, boolean flag) {
    String pasajeroValue = "";
    try (Connection conexion = DriverManager.getConnection(URL, USER, PASS))
{
        if (!flag) {
            // Separa el contenido del string
            String[] partes = val.split("-");
            String digitpass = partes[0].trim();
            String nompas = partes[1].trim();
            String Query = "SELECT idPasajero FROM InfoPasajero.Pasajero
WHERE NumPasaporte LIKE '%" + digitpass + "' AND Nom_Pasajero = '" + nompas
+ "'";

            PreparedStatement pstmt = conexion.prepareStatement(Query);
            ResultSet rs = pstmt.executeQuery();
```

```
            if (rs.next()) {
                pasajeroValue = rs.getString("idPasajero");
            }
        } else {
            String Query = "SELECT Nom_Pasajero, NumPasaporte FROM
InfoPasajero.Pasajero WHERE idPasajero=" + val;
            PreparedStatement pstmt = conexion.prepareStatement(Query);
            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) {
                String digitnumpass =
rs.getString("NumPasaporte").substring(rs.getString("NumPasaporte").length()
- 4);
                pasajeroValue = digitnumpass + " - " +
rs.getString("Nom_Pasajero");
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return pasajeroValue;
}
```

### Método btn_Eliminar_TarPasajeroActionPerformed

Este método se ejecuta al hacer clic en el botón "Eliminar" en la sección de tarjetas de pasajero de la interfaz gráfica. A continuación, se presenta un resumen de su funcionamiento:

1. **Validación de Selección:** Verifica que se haya seleccionado un registro para eliminar. Si no se ha seleccionado, muestra un mensaje de advertencia y termina la ejecución.
2. **Construcción de Consulta SQL:** Construye una consulta SQL para eliminar la tarjeta de pasajero seleccionada de la tabla correspondiente (**InfoPasajero.TarjetaPasajero**) utilizando el ID actual (**currentID**).
3. **Verificación de Clave Foránea:** Antes de ejecutar la consulta de eliminación, realiza una verificación mediante el método **Query1** para asegurarse de que la tarjeta de pasajero no esté siendo referenciada como clave foránea en otras tablas. Si es una clave foránea, muestra un mensaje de advertencia y termina la ejecución.
4. **Ejecución de Consulta y Actualización de la Interfaz:** Si la verificación de clave foránea es exitosa, ejecuta la consulta de eliminación, actualiza la tabla de datos de tarjetas de pasajero y realiza la acción de volver (**btn_backTarPasajeroActionPerformed**).

```
private void
btn_Eliminar_TarPasajeroActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_btn_Eliminar_TarPasajeroActionPerformed
    if (currentID.equals("")) {
```

```
        String mensaje = "Selecciona correctamente un registro";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }

    String query = "DELETE FROM InfoPasajero.TarjetaPasajero WHERE
idTarjetaPasajero = " + currentID;
    if(Query1(query)==-1)
    {
        String mensaje = "No se pudo eliminar porque aparece \ncomo clave
foranea en otra tabla";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }
    ConsultaDatosTarPasajero();
    btn_backTarPasajeroActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
}//GEN-LAST:event_btn_Eliminar_TarPasajeroActionPerformed
```

### Método btn_backTarPasajeroActionPerformed

Este método realiza acciones específicas cuando se presiona el botón "Volver" en la sección de tarjetas de pasajero de la interfaz gráfica. Su función es restablecer la visibilidad de ciertos elementos y habilitar el botón de agregar (**btn_Agregar_TarPasajero**). Además, establece a "Ninguno" (**JCB_PasajeroTarPasajero.setSelectedIndex(-1)**) la selección en el cuadro combinado de pasajeros y limpia los campos relacionados con la tarjeta de pasajero (**tb_NombreTarPasajero**, **tb_NumTarPasajero**, **tb_CVVTarPasajero**, **TF_FechaVenTarPasajero**). La variable **currentID** se establece en una cadena vacía.

```
private void btn_backTarPasajeroActionPerformed(java.awt.event.ActionEvent
evt) {//GEN-FIRST:event_btn_backTarPasajeroActionPerformed
    lbl_Modifica.setVisible(false);
    btn_Agregar_TarPasajero.setEnabled(true);
    btn_Eliminar_TarPasajero.setEnabled(false);
    btn_Modificar_TarPasajero.setEnabled(false);
    btn_backTarPasajero.setEnabled(false);

    JCB_PasajeroTarPasajero.setSelectedIndex(-1); // Seleccionar "Ninguno"
    tb_NombreTarPasajero.setText("");
    tb_NumTarPasajero.setText("");
    tb_CVVTarPasajero.setText("");
    TF_FechaVenTarPasajero.setText("");
```

```
    currentID = "";
}//GEN-LAST:event_btn_backTarPasajeroActionPerformed
```

### Método JT_TarPasajeroMouseClicked

Este método gestiona la lógica detrás de la interacción del usuario con la tabla de tarjetas de pasajero (**JT_TarPasajero**). Al hacer clic en una fila, se actualiza la interfaz gráfica y se muestran detalles específicos del elemento seleccionado en la tabla, como el ID del registro actual, el nombre del titular, el número de tarjeta, etc. Además, realiza algunas conversiones y formateos, como la transformación de la fecha de vencimiento del formato de la base de datos al formato MM/yyyy.

```
private void JT_TarPasajeroMouseClicked(java.awt.event.MouseEvent evt)
{//GEN-FIRST:event_JT_TarPasajeroMouseClicked
    lbl_Modifica.setVisible(true);
    btn_Agregar_TarPasajero.setEnabled(false);
    btn_Eliminar_TarPasajero.setEnabled(true);
    btn_Modificar_TarPasajero.setEnabled(true);
    btn_backTarPasajero.setEnabled(true);

    int filaSeleccionada = JT_TarPasajero.getSelectedRow();
    currentID = JT_TarPasajero.getValueAt(filaSeleccionada, 0).toString();
    String pasajero = JT_TarPasajero.getValueAt(filaSeleccionada,
1).toString();
    String nomtitular = JT_TarPasajero.getValueAt(filaSeleccionada,
2).toString();
    String numtarj = JT_TarPasajero.getValueAt(filaSeleccionada,
4).toString();
    String fechaven = JT_TarPasajero.getValueAt(filaSeleccionada,
5).toString();
    String cvv = JT_TarPasajero.getValueAt(filaSeleccionada, 6).toString();


    try {
        SimpleDateFormat formatoSalida = new SimpleDateFormat("MM/yyyy");
        SimpleDateFormat formatoEntrada = new SimpleDateFormat("yyyy-MM-
dd");
        java.util.Date fecha = formatoEntrada.parse(fechaven);
        fechaven = formatoSalida.format(fecha);
    } catch (ParseException e) {
        e.printStackTrace();
    }

    tb_NombreTarPasajero.setText(nomtitular);
    tb_NumTarPasajero.setText(numtarj);
```

```
    tb_CVVTarPasajero.setText(cvv);
    TF_FechaVenTarPasajero.setText(fechaven);

    for (int i = 0; i < JCB_PasajeroTarPasajero.getItemCount(); i++) {
        String item = (String) JCB_PasajeroTarPasajero.getItemAt(i);

        if (item.equals(pasajero)) {
            JCB_PasajeroTarPasajero.setSelectedIndex(i);
            break;
        }
    }
}//GEN-LAST:event_JT_TarPasajeroMouseClicked
```

### Método ConsultaDatosTarjPasajero

Este método realiza una consulta a la base de datos para recuperar la información de las tarjetas de pasajero almacenadas. Luego, utiliza un objeto **DefaultTableModel** para mostrar estos datos en un componente **JTable** (**JT_TarPasajero**) en la interfaz gráfica. La información recuperada incluye el ID de la tarjeta, el pasajero asociado (con el método **getPasajero**), el número de la tarjeta, la fecha de expiración y otros detalles relacionados con la tarjeta de pasajero.

```
public void ConsultaDatosTarPasajero(){
    try{
        CONN = ConnectBD();
        java.sql.Statement Stmnt = CONN.createStatement();
        String Query = "SELECT * FROM InfoPasajero.TarjetaPasajero";
        String[]Data = new String[7];
        ResultSet Columns = Stmnt.executeQuery (Query);
        DefaultTableModel model = (DefaultTableModel)
JT_TarPasajero.getModel();

        // Eliminar todas las filas existentes
        model.setRowCount(0);
        while(Columns.next())
        {
            Data[0]=Columns.getString(1);
            Data[1]=getPasajero(Columns.getString(2), true);
            Data[2]=Columns.getString(3);
            Data[3]=Columns.getString(4);
            Data[4]=Columns.getString(5);
            Data[5]=Columns.getString(6);
            Data[6]=Columns.getString(7);
            model.addRow(Data);
        }
        Stmnt.close();
        CONN.close();
```

```
    }catch (Exception ex) {
        JOptionPane.showMessageDialog(null, ex);
    }
}
```

### Método tb_NumTarPasajeroKeyTyped, tb_CVVTarPasajeroKeyTyped y TF_FechaVenTarPasajeroKeyTyped

Estos métodos son *event listeners* para la interfaz gráfica (GUI) de una aplicación Java Swing, probablemente parte de un formulario relacionado con la información de tarjetas de pasajeros. Aquí está un resumen de cada uno:

1. **Método tb_NumTarPasajeroKeyTyped**:
    - Este método restringe la entrada en un campo de texto (**tb_NumTarPasajero**) para que solo permita caracteres numéricos.
    - Evita que se ingresen más de 16 caracteres en el campo.
2. **Método tb_CVVTarPasajeroKeyTyped**:
    - Similar al anterior, restringe la entrada en otro campo de texto (**tb_CVVTarPasajero**) para que solo permita caracteres numéricos.
    - Limita la longitud del campo a 3 caracteres.
3. **Método TF_FechaVenTarPasajeroKeyTyped**:
    - Este método gestiona la entrada de fecha en un formato específico (MM/yy) en un campo de texto (**TF_FechaVenTarPasajero**).
    - Solo permite dígitos y la barra inclinada ("/").
    - Verifica el formato de fecha y asegura que haya un único carácter de barra inclinada y que esté en la posición correcta.

```
private void tb_NumTarPasajeroKeyTyped(java.awt.event.KeyEvent evt) {//GEN-FIRST:event_tb_NumTarPasajeroKeyTyped
    char c = evt.getKeyChar();
    // Verificar si el carácter es un número o la tecla de borrar
    if (!Character.isDigit(c) && c != '\b') {
        evt.consume(); // Consumir el evento si no es un número o la tecla de borrar
    }

    // Verificar si ya hay 10 caracteres en el campo
    if (tb_NumTarPasajero.getText().length() >= 16) {
        evt.consume(); // Consumir el evento si ya hay 16 caracteres
    }
}//GEN-LAST:event_tb_NumTarPasajeroKeyTyped
```

```
private void tb_CVVTarPasajeroKeyTyped(java.awt.event.KeyEvent evt) {//GEN-
FIRST:event_tb_CVVTarPasajeroKeyTyped
    char c = evt.getKeyChar();
    // Verificar si el carácter es un número o la tecla de borrar
    if (!Character.isDigit(c) && c != '\b') {
        evt.consume(); // Consumir el evento si no es un número o la tecla
de borrar
    }

    // Verificar si ya hay 10 caracteres en el campo
    if (tb_CVVTarPasajero.getText().length() >= 3) {
        evt.consume(); // Consumir el evento si ya hay 3 caracteres
    }
}//GEN-LAST:event_tb_CVVTarPasajeroKeyTyped
```

```
private void TF_FechaVenTarPasajeroKeyTyped(java.awt.event.KeyEvent evt)
{//GEN-FIRST:event_TF_FechaVenTarPasajeroKeyTyped
    char c = evt.getKeyChar();
    String fechaActual = TF_FechaVenTarPasajero.getText();

    // Asegura que solo se ingresen dígitos y diagonal
    if (!Character.isDigit(c) && c != '/' || fechaActual.length() >= 7) {
        evt.consume();
    }

    // Verifica el formato MM/yy
    if (Character.isDigit(c) && fechaActual.length() == 2) {
        int mes = Integer.parseInt(fechaActual);
        if (mes < 1 || mes > 12) {
            evt.consume();
        } else {
            TF_FechaVenTarPasajero.setText(fechaActual + "/");
        }
    }

    // Asegura que solo haya un slash y se encuentre en la posición correcta
    if (c == '/' && (fechaActual.contains("/") || fechaActual.length() >=
7)) {
        evt.consume();
    }
}//GEN-LAST:event_TF_FechaVenTarPasajeroKeyTyped
```

## Ventana Asiento

### Método LlenarCBAsiento

Este método **llenarCBAsiento** se encarga de poblar un JComboBox (**JCB_ItinerarioAsiento**) con información detallada sobre los itinerarios disponibles. Realiza una consulta a la base de datos para obtener detalles como la hora de salida, la fecha del vuelo y el ID del vuelo asociado. Utiliza el método **getVuelo** para obtener información más legible sobre el vuelo al que pertenece el itinerario.

Cada elemento del JComboBox se forma concatenando el ID del vuelo, la hora de salida sin segundos y la fecha del vuelo. Además, se almacenan los ID de los itinerarios en la lista **listIDItinerario** para su posterior referencia.

```
public void llenarCBAsiento() {
    try {
        Connection conexion = DriverManager.getConnection(URL, USER, PASS);
        Statement statement = conexion.createStatement();
        String query = "SELECT idItinerario, idVuelo, HoraSalida, FechaVuelo
FROM InfoAerolinea.Itinerario";
        ResultSet rs = statement.executeQuery(query);

        while (rs.next()) {
            String horaCompleta = rs.getString("HoraSalida");
            String horaSinSegundos = horaCompleta.substring(0, 5);

            Date fecha = rs.getDate("FechaVuelo");
            SimpleDateFormat dateFormat = new SimpleDateFormat("MM-dd-
yyyy");
            String nuevaFecha = dateFormat.format(fecha);

            String item = "(" + getVuelo(rs.getString("idVuelo"), true) + ")
- " + horaSinSegundos + " - " + nuevaFecha;
            JCB_ItinerarioAsiento.addItem(item);
            listIDItinerario.add(Integer.parseInt(rs.getString("idItinerario
")));
        }

        rs.close();
        statement.close();
        conexion.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

### Método btn_backAsientoActionPerformed

El método **btn_backAsientoActionPerformed** es un manejador de eventos que se activa cuando se presiona el botón "Atrás" en el contexto de la interfaz de usuario asociada a asientos. Su función es restablecer la interfaz a un estado inicial, ocultando un mensaje de modificación (**lbl_Modifica**) y habilitando el botón "Agregar Asiento". Además, deshabilita los botones relacionados con la modificación y eliminación de asientos (**btn_Eliminar_Asiento** y **btn_Modificar_Asiento**). Restablece los valores de los elementos de la interfaz, como los JComboBox (**JCB_ItinerarioAsiento** y **JCB_LetraAsiento**), el control deslizante (**JS_NumAsiento**), y el casillero de verificación (**JCHBX_OcupadoAsiento**). Finalmente, reinicia la variable **currentID** a un estado vacío.

```
private void btn_backAsientoActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_btn_backAsientoActionPerformed
    lbl_Modifica.setVisible(false);
    btn_Agregar_Asiento.setEnabled(true);
    btn_Eliminar_Asiento.setEnabled(false);
    btn_Modificar_Asiento.setEnabled(false);
    btn_backAsiento.setEnabled(false);

    JCB_ItinerarioAsiento.setSelectedIndex(-1); // Seleccionar "Ninguno"
    JCB_LetraAsiento.setSelectedIndex(-1); // Seleccionar "Ninguno"
    JS_NumAsiento.setValue(1);
    JCHBX_OcupadoAsiento.setSelected(false);

    currentID = "";
}//GEN-LAST:event_btn_backAsientoActionPerformed
```

### Método btn_Agregar_AsientoActionPerformed

El método **btn_Agregar_AsientoActionPerformed** es un manejador de eventos asociado al botón de agregar asiento en la interfaz de usuario. Su función es validar los datos ingresados, como la selección de itinerario (**JCB_ItinerarioAsiento**), la selección de letra de asiento (**JCB_LetraAsiento**), y otros elementos de la interfaz como el control deslizante (**JS_NumAsiento**) y el casillero de verificación (**JCHBX_OcupadoAsiento**).

Si los datos son válidos, el método construye una consulta SQL para insertar un nuevo asiento en la base de datos. La consulta depende del estado del casillero de verificación (**JCHBX_OcupadoAsiento**). Si está marcado, el asiento se considera ocupado; de lo contrario, se considera desocupado. Se realiza una verificación adicional utilizando la función **VerificarCupoItinerario** para asegurarse de que el itinerario seleccionado no esté lleno antes de agregar un nuevo asiento. Si la verificación pasa, se ejecuta la consulta SQL y se actualizan los datos de la interfaz llamando a **ConsultaDatosAsiento()**. Finalmente, se llama a **btn_backAsientoActionPerformed** para restablecer la interfaz.

```
private void btn_Agregar_AsientoActionPerformed(java.awt.event.ActionEvent
evt) {//GEN-FIRST:event_btn_Agregar_AsientoActionPerformed
```

```
    if (JCB_ItinerarioAsiento.getSelectedIndex() == -
1||JCB_LetraAsiento.getSelectedIndex() == -1) {
        String mensaje = "Ingresa correctamente los datos";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }

    String idIT =
listIDItinerario.get(JCB_ItinerarioAsiento.getSelectedIndex()).toString();
    if (VerificarCupoItinerario(idIT)) {
        String query = JCHBX_OcupadoAsiento.isSelected()
                ? "INSERT INTO InfoPasajero.Asiento (idItinerario,
Num_Asiento, Letra, Ocupado) VALUES (" + idIT + ", " +
JS_NumAsiento.getValue() + ", '" + JCB_LetraAsiento.getSelectedItem() + "',
TRUE)"
                : "INSERT INTO InfoPasajero.Asiento (idItinerario,
Num_Asiento, Letra, Ocupado) VALUES (" + idIT + ", " +
JS_NumAsiento.getValue() + ", '" + JCB_LetraAsiento.getSelectedItem() + "',
FALSE)";

        if (Query1(query) == -1) {
            String mensaje = "Ocurrió una excepción: \nEse asiento ya
existe";
            JOptionPane.showMessageDialog(null, mensaje, "Error",
JOptionPane.ERROR_MESSAGE);
        }

        ConsultaDatosAsiento();
        btn_backAsientoActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
    } else {
        String mensaje = "Ese itinerario ya está lleno";
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
JOptionPane.INFORMATION_MESSAGE);
    }
}//GEN-LAST:event_btn_Agregar_AsientoActionPerformed
```

## Método btn_Modificar_AsientoActionPerformed

El método **btn_Modificar_AsientoActionPerformed** es un manejador de eventos asociado al botón de modificar asiento en la interfaz de usuario. Su función es validar los datos seleccionados y los nuevos datos ingresados para modificar un asiento existente en la base de datos.

Primero, se verifica si se ha seleccionado un registro existente (**currentID** no está vacío) y si se han proporcionado datos válidos, como la selección de itinerario (**JCB_ItinerarioAsiento**), la selección de letra de asiento (**JCB_LetraAsiento**), y otros elementos de la interfaz como el control deslizante (**JS_NumAsiento**) y el casillero de verificación (**JCHBX_OcupadoAsiento**).

Luego, se verifica si el nuevo itinerario seleccionado (**idIT**) no está lleno y si el asiento no está asociado a una venta-boleto (se hace referencia a la función **VerificarCupoItinerario** y **VerificarBoletoItinerario**). Si ambas verificaciones pasan, se construye una consulta SQL para actualizar el registro de asiento en la base de datos, considerando el estado del casillero de verificación (**JCHBX_OcupadoAsiento**).

Si la consulta SQL se ejecuta con éxito, se actualizan los datos de la interfaz llamando a **ConsultaDatosAsiento()** y se restablece la interfaz llamando a **btn_backAsientoActionPerformed**. En caso de algún problema durante la ejecución, se muestra un mensaje de error al usuario.

```
private void btn_Modificar_AsientoActionPerformed(java.awt.event.ActionEvent
evt) {//GEN-FIRST:event_btn_Modificar_AsientoActionPerformed
    if (currentID.equals("")||JCB_ItinerarioAsiento.getSelectedIndex() == -
1||JCB_LetraAsiento.getSelectedIndex() == -1) {
        String mensaje = "Selecciona correctamente un registro y sus nuevos
datos";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }

    String idIT =
listIDItinerario.get(JCB_ItinerarioAsiento.getSelectedIndex()).toString();
    if (VerificarCupoItinerario(idIT) /*&&
VerificarBoletoItinerario(currentID)*/) {
        String query = JCHBX_OcupadoAsiento.isSelected() ?
                    "UPDATE InfoPasajero.Asiento SET idItinerario = " + idIT
+ ", Num_Asiento = " + JS_NumAsiento.getValue() + ", Letra = '" +
JCB_LetraAsiento.getSelectedItem() + "', Ocupado = TRUE WHERE idAsiento = "
+ currentID:
                    "UPDATE InfoPasajero.Asiento SET idItinerario = " + idIT
+ ", Num_Asiento = " + JS_NumAsiento.getValue() + ", Letra = '" +
JCB_LetraAsiento.getSelectedItem() + "', Ocupado = FALSE WHERE idAsiento = "
+ currentID;

        if (Query1(query) == -1) {
            String mensaje = "Ocurrió una excepción: \nEse asiento ya
existe, no se modificó";
            JOptionPane.showMessageDialog(null, mensaje, "Error",
JOptionPane.ERROR_MESSAGE);
```

```
        }

        ConsultaDatosAsiento();
        btn_backAsientoActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
    } else {
        String mensaje = "No se pudo modificar porque el nuevo itinerario ya
esta lleno \no el asiento ya esta asociado a una venta-boleto";
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
JOptionPane.INFORMATION_MESSAGE);
    }
}//GEN-LAST:event_btn_Modificar_AsientoActionPerformed
```

### Método btn_Eliminar_AsientoActionPerformed

Se encarga de manejar el evento de clic en el botón "Eliminar" para eliminar un asiento. Verifica si se ha seleccionado un registro, ejecuta una consulta SQL para eliminar el asiento correspondiente y actualiza la tabla de asientos. Además, gestiona posibles errores, como la presencia de claves foráneas que impiden la eliminación. Después de la eliminación, restaura el formulario a su estado inicial.

```
private void btn_Eliminar_AsientoActionPerformed(java.awt.event.ActionEvent
evt) {//GEN-FIRST:event_btn_Eliminar_AsientoActionPerformed
    if (currentID.equals("")) {
        String mensaje = "Selecciona correctamente un registro";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }

    String query = "DELETE FROM InfoPasajero.Asiento WHERE idAsiento = " +
currentID;
    if (Query1(query) == -1) {
        String mensaje = "No se pudo eliminar porque aparece \ncomo clave
foranea en otra tabla";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }
    ConsultaDatosAsiento();
    btn_backAsientoActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
}//GEN-LAST:event_btn_Eliminar_AsientoActionPerformed
```

## Método ConsultaDatosAsiento

Este método realiza una consulta a la base de datos para obtener información sobre los asientos. Luego, utiliza un objeto **DefaultTableModel** asociado a una tabla (**JT_Asiento**) para mostrar los resultados de la consulta. La información incluye detalles sobre el asiento, como su ID, el itinerario al que está asignado, el número del asiento, el tipo de asiento y su estado.

Además, utiliza el método **getItinerario** para obtener información legible sobre el itinerario al que está asignado cada asiento, mejorando así la presentación de la información en la interfaz de usuario.

```
public void ConsultaDatosAsiento() {
    try{
        CONN = ConnectBD();
        java.sql.Statement Stmnt = CONN.createStatement();
        String Query = "SELECT * FROM InfoPasajero.Asiento";
        String[]Data = new String[5];
        ResultSet Columns = Stmnt.executeQuery (Query);
        DefaultTableModel model = (DefaultTableModel) JT_Asiento.getModel();

        // Eliminar todas las filas existentes
        model.setRowCount(0);
        while(Columns.next())
        {
            Data[0]=Columns.getString(1);
            Data[1]=getItinerario(Columns.getString(2));
            Data[2]=Columns.getString(3);
            Data[3]=Columns.getString(4);
            Data[4]=Columns.getString(5);
            model.addRow(Data);
        }
        Stmnt.close();
        CONN.close();
    }catch (Exception ex) {
        JOptionPane.showMessageDialog(null, ex);
    }
}
```

## Método getItinerario

Este método **getItinerario** obtiene información detallada sobre un itinerario específico a partir de su ID. Realiza una consulta a la base de datos para obtener detalles como la hora de salida, la fecha del vuelo y el ID del vuelo asociado. Además, utiliza el método **getVuelo** para obtener información más legible sobre el vuelo al que pertenece el itinerario.

El resultado se devuelve en una cadena que incluye el ID del vuelo, la hora de salida sin segundos y la fecha del vuelo, proporcionando así una descripción completa y formateada del itinerario.

```
private String getItinerario(String val) {
    String itinerarioValue = "";
    try (Connection conexion = DriverManager.getConnection(URL, USER, PASS);
PreparedStatement pstmt = conexion.prepareStatement("SELECT idVuelo,
HoraSalida, FechaVuelo FROM InfoAerolinea.Itinerario WHERE
idItinerario="+val)) {
        try (ResultSet rs = pstmt.executeQuery()) {
            if (rs.next()) {
                String horaCompleta = rs.getString("HoraSalida");
                String horaSinSegundos = horaCompleta.substring(0, 5);

                Date fecha = rs.getDate("FechaVuelo");
                SimpleDateFormat dateFormat = new SimpleDateFormat("MM-dd-
yyyy");
                String nuevaFecha = dateFormat.format(fecha);

                itinerarioValue = "(" + getVuelo(rs.getString("idVuelo"),
true) + ") - " + horaSinSegundos + " - " + nuevaFecha;
            }
        }
    } catch (SQLException e) {
        e.printStackTrace(); // Manejo básico de la excepción. Puedes
personalizarlo según tus necesidades.
    }

    return itinerarioValue;
}
```

### Método VerificarCupoItinerario

La función **VerificarCupoItinerario** se encarga de verificar si hay espacio disponible en un itinerario para asignar más asientos. Toma el ID de un itinerario como parámetro, realiza una consulta a la base de datos para contar la cantidad de asientos asignados en ese itinerario y devuelve **true** si no hay asientos asignados o si la cantidad es menor que 8, indicando que aún hay espacio disponible. La función maneja excepciones de SQL y retorna **false** en caso de error o si no hay espacio disponible; de lo contrario, retorna **true**.

```
private boolean VerificarCupoItinerario(String itinerario) {
    try (Connection conexion = DriverManager.getConnection(URL, USER, PASS))
{
        String consulta = "SELECT COUNT(*) FROM InfoPasajero.Asiento WHERE
idItinerario = " + itinerario;
        try (PreparedStatement pstmt = conexion.prepareStatement(consulta))
{
            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) {
```

```
                int count = rs.getInt(1);
                return count == 0 || count < 8;
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return false;
}
```

### Método VerificarBoletoItinerario

La función **VerificarBoletoItinerario** se encarga de verificar si un asiento específico está asociado a algún boleto en la base de datos. Toma el ID del asiento como parámetro, realiza una consulta a la base de datos para contar la cantidad de boletos asociados a ese asiento y devuelve **true** si no hay boletos asociados, indicando que el asiento está disponible. La función maneja excepciones de SQL y retorna **false** en caso de error o si el asiento está asociado a algún boleto; de lo contrario, retorna **true**.

```
private boolean VerificarBoletoItinerario(String asiento) {
    try (Connection conexion = DriverManager.getConnection(URL, USER, PASS))
{
        String consulta = "SELECT COUNT(*) FROM InfoPasajero.Boleto WHERE
idAsiento = " + asiento;
        try (PreparedStatement pstmt = conexion.prepareStatement(consulta))
{
            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) {
                int count = rs.getInt(1);
                return count == 0;
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return false;
}
```

### Método JT_AsientoMouseClicked

El método **JT_AsientoMouseClicked** maneja el evento de clic en una fila de la tabla de asientos (**JT_Asiento**). Cuando se hace clic en una fila, el método extrae la información relevante de esa fila, como el ID del asiento, el itinerario, el número de asiento, la letra y si está ocupado. Luego, actualiza los componentes visuales en el formulario con esta información. Además, realiza ajustes en la visibilidad de una etiqueta (**lbl_Modifica**) y en el estado de los botones para reflejar la acción de modificación.

```
private void JT_AsientoMouseClicked(java.awt.event.MouseEvent evt) {//GEN-
FIRST:event_JT_AsientoMouseClicked
    lbl_Modifica.setVisible(true);
    btn_Agregar_Asiento.setEnabled(false);
    btn_Eliminar_Asiento.setEnabled(true);
    btn_Modificar_Asiento.setEnabled(true);
    btn_backAsiento.setEnabled(true);

    int filaSeleccionada = JT_Asiento.getSelectedRow();
    currentID = JT_Asiento.getValueAt(filaSeleccionada, 0).toString();
    String itinerario = JT_Asiento.getValueAt(filaSeleccionada,
1).toString();
    String numasiento = JT_Asiento.getValueAt(filaSeleccionada,
2).toString();
    String letra = JT_Asiento.getValueAt(filaSeleccionada, 3).toString();
    String ocupado = JT_Asiento.getValueAt(filaSeleccionada, 4).toString();

    int valorEntero = Integer.parseInt(numasiento);
    JS_NumAsiento.setValue(valorEntero);

    if(ocupado.equals("f"))
    {
        JCHBX_OcupadoAsiento.setSelected(false);
    }else
    {
        JCHBX_OcupadoAsiento.setSelected(true);
    }

    for (int i = 0; i < JCB_ItinerarioAsiento.getItemCount(); i++) {
        String item = (String) JCB_ItinerarioAsiento.getItemAt(i);

        if (item.equals(itinerario)) {
            JCB_ItinerarioAsiento.setSelectedIndex(i);
            break;
        }
    }

    for (int i = 0; i < JCB_LetraAsiento.getItemCount(); i++) {
        String item = (String) JCB_LetraAsiento.getItemAt(i);

        if (item.equals(letra)) {
            JCB_LetraAsiento.setSelectedIndex(i);
            break;
        }
    }
```

```
}//GEN-LAST:event_JT_AsientoMouseClicked
```

## Ventana Venta

### Método btn_Agregar_VentaActionPerformed

El método **btn_Agregar_VentaActionPerformed** en un formulario de operaciones de venta realiza las siguientes acciones al hacer clic en el botón "Agregar Venta":

- Verifica si los datos esenciales (tarjeta y itinerario) están seleccionados.
- Obtiene los IDs del itinerario y la tarjeta de pasajero seleccionados.
- Construye una consulta SQL para insertar la venta en la base de datos, considerando si la venta está pagada o no.
- Verifica la disponibilidad de asientos en el itinerario seleccionado y otros datos necesarios.
- Si todo está en orden, realiza la inserción de la venta y los boletos asociados en la base de datos.
- Actualiza la vista de ventas.
- Restablece el formulario de ventas.
- Notifica al usuario sobre el éxito o cualquier excepción que ocurra durante el proceso.

```
private void btn_Agregar_VentaActionPerformed(java.awt.event.ActionEvent
evt) {//GEN-FIRST:event_btn_Agregar_VentaActionPerformed
    if (JCB_TarjetaVenta.getSelectedIndex() == -
1||JCB_ItinerarioVenta.getSelectedIndex() == -1) {
        String mensaje = "Ingresa correctamente los datos";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }
    String idIT =
listIDItinerario.get(JCB_ItinerarioVenta.getSelectedIndex()).toString();
    String idTP =
listIDTrjetaPasajero.get(JCB_TarjetaVenta.getSelectedIndex()).toString();
    String query = JCHBX_PagadoVenta.isSelected()?
                    "INSERT INTO InfoPasajero.Venta (idTarjetaPasajero,
idItinerario, MontoTotal, EstadoPago) VALUES (" + idTP + ", " + idIT + ", "
+ 0 + ", TRUE)" :
                    "INSERT INTO InfoPasajero.Venta (idTarjetaPasajero,
idItinerario, MontoTotal, EstadoPago) VALUES (" + idTP + ", " + idIT + ", "
+ 0 + ", FALSE)";
```

```
    if (VerificarDispAsientos(idIT,
Integer.parseInt(JS_NumBoletos.getValue().toString())) &&
JS_IVABoleto.getValue() != null && !tb_TSBoleto.toString().equals("") &&
!tb_TSRBoleto.toString().equals(""))
    {
        int idVenta = Query2(query,"idVenta");
        //System.out.println(idVenta);
        String CostVuel = getCostoBaseVuelo(idIT);
        CostVuel = CostVuel.replaceAll("[\\$,]", "");
        if(idVenta!=-1)
        {
            if (!insertarBoletos(idVenta, idIT, idTP, CostVuel,
Integer.parseInt(JS_NumBoletos.getValue().toString()))
            {
                String mensaje = "Ocurrió una excepción: \nNo se pudo
insertar los boletos";
                int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
                JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
                return;
            }
            ConsultaVenta();
            btn_backVentaActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
        }else
        {
            String mensaje = "Ocurrió una excepción: \nNo se pudo insertar
porque no es posible tener dos ventas\ndel mismo pasajero en el mismo
itinerario";
            int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
            JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
            return;
        }
    }else
    {
        String mensaje = "Ocurrió una excepción: \nNo se pudo insertar
porque no hay suficientes asientos disponibles";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }
}//GEN-LAST:event_btn_Agregar_VentaActionPerformed
```

## Método getCostoBaseVuelo

La función **getCostoBaseVuelo** se encarga de obtener el costo base de un vuelo asociado a un itinerario específico. Aquí tienes una descripción del método:

- **Parámetro**:
  - **idIT**: El identificador del itinerario del cual se desea obtener el costo base del vuelo asociado.
- **Retorno**:
  - Retorna el costo base del vuelo en formato de cadena (**String**).
- **Funcionamiento**:
  - Establece una conexión a la base de datos utilizando la información de conexión proporcionada (URL, usuario, contraseña).
  - Construye una consulta SQL que selecciona el campo **CostoBase** de la tabla **Vuelo** para el itinerario específico.
  - Ejecuta la consulta SQL y obtiene el resultado.
  - Si la consulta devuelve resultados, extrae el valor del campo **CostoBase** y lo retorna como una cadena.
  - En caso de que la consulta no devuelva resultados, o si ocurre alguna excepción durante la ejecución, imprime la traza del error y retorna **null**.

```
private String getCostoBaseVuelo(String idIT) {
    try (Connection conexion = DriverManager.getConnection(URL, USER, PASS))
{
        String consulta = "SELECT V.CostoBase FROM InfoAerolinea.Itinerario
I JOIN InfoAerolinea.Vuelo V ON I.idVuelo = V.idVuelo WHERE I.idItinerario =
"+idIT;
        try (PreparedStatement pstmt = conexion.prepareStatement(consulta))
{
            try (ResultSet rs = pstmt.executeQuery()) {
                if (rs.next()) {
                    String costoBase = rs.getString("CostoBase");
                    return costoBase;
                }
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return null; // Retorna null en caso de error
}
```

## Método VerificarDispAsientos

El método **VerificarDispAsientos** se encarga de verificar si hay suficientes asientos disponibles en un itinerario específico para una cantidad determinada de asientos. A continuación, se presenta una descripción de este método:

- **Parámetros**:
  - **idIT**: El identificador del itinerario para el cual se desea verificar la disponibilidad de asientos.
  - **numAsientos**: El número de asientos que se desea verificar.
- **Funcionamiento**:
  - Establece una conexión a la base de datos utilizando la información de conexión proporcionada (URL, usuario, contraseña).
  - Construye una consulta SQL para contar la cantidad de asientos no ocupados (donde el campo **Ocupado** es **FALSE**) en el itinerario especificado.
  - Ejecuta la consulta SQL y obtiene el resultado.
  - Verifica si la cantidad de asientos no ocupados es mayor o igual al número de asientos que se desea verificar.
  - Retorna **true** si hay suficientes asientos disponibles; de lo contrario, retorna **false**.
- **Manejo de Excepciones**:
  - En caso de que ocurra una excepción durante la conexión a la base de datos o la ejecución de la consulta, se imprime la traza del error.
  - En cualquier caso de excepción, el método retorna **false** por precaución.

```
public boolean VerificarDispAsientos(String idIT, int numAsientos) {
    try (Connection conexion = DriverManager.getConnection(URL, USER, PASS))
{
        String consulta = "SELECT COUNT(*) FROM InfoPasajero.Asiento WHERE
idItinerario = "+idIT+" AND Ocupado = FALSE";
        try (PreparedStatement pstmt = conexion.prepareStatement(consulta))
{
            try (ResultSet rs = pstmt.executeQuery()) {
                if (rs.next()) {
                    int count = rs.getInt(1);
                    return count >= numAsientos;
                }
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
```

```
    }
    return false; // Retorna false en caso de error
}
```

### Método insertarBoletos

La función **insertarBoletos** se encarga de insertar boletos asociados a una venta en la base de datos.

- **Parámetros**:
    - **idVenta**: Identificador de la venta a la cual se asociarán los boletos.
    - **idIT**: Identificador del itinerario al cual pertenecen los asientos.
    - **idTP**: Identificador de la tarjeta de pasajero asociada a la venta.
    - **costVuel**: Costo base del vuelo.
    - **numBoletos**: Número de boletos que se desean insertar.
- **Retorno**:
    - Retorna un valor booleano indicando si la inserción de los boletos fue exitosa (**true**) o no (**false**).
- **Funcionamiento**:
    - Se realizan cálculos para determinar los impuestos, tarifas adicionales y costo total de los boletos.
    - Se obtiene el identificador del pasajero asociado a la tarjeta de pasajero (**idTP**).
    - Se itera sobre el número de boletos y se busca un asiento disponible para cada uno en el itinerario (**idIT**).
    - Se inserta un nuevo boleto en la tabla **InfoPasajero.Boleto** asociado a la venta y al asiento correspondiente.

```
public boolean insertarBoletos(int idVenta, String idIT, String idTP, String
costVuel, int numBoletos) {
    float costV = Float.parseFloat(costVuel);
    float Impuestos = costV * Float.parseFloat("0." +
JS_IVABoleto.getValue().toString());
    float TarAd = Float.parseFloat(tb_TSBoleto.getText()) +
Float.parseFloat(tb_TSRBoleto.getText());
    float CostTot = costV + Impuestos + TarAd;
    try {
        Connection conn = DriverManager.getConnection(URL, USER, PASS);
        String query;
        String idPasajero = "";
```

```
        String idAsiento = "";

        query = "SELECT idPasajero FROM InfoPasajero.TarjetaPasajero WHERE
idTarjetaPasajero = "+idTP;
        try (PreparedStatement pstmt = conn.prepareStatement(query);
ResultSet rs = pstmt.executeQuery()) {
            while (rs.next()) {
                idPasajero = rs.getString("idPasajero");
            }
        }

        for (int i = 0; i < numBoletos; i++) {
            query = "SELECT idAsiento, Ocupado FROM InfoPasajero.Asiento
WHERE idItinerario = " + idIT;
            try (PreparedStatement stmt = conn.prepareStatement(query);
ResultSet asientos = stmt.executeQuery()) {
                while (asientos.next()) {
                    if (asientos.getString("Ocupado").equals("f")) {
                        idAsiento = asientos.getString("idAsiento");
                        break;
                    }
                }
            }
            System.out.println(idPasajero+"-"+idAsiento);
            query = "INSERT INTO InfoPasajero.Boleto (idVenta, idPasajero,
idAsiento, Impuestos, TarifasAdicionales, CostoTotal, Estado) VALUES ("+
idVenta + ", NULL, " + idAsiento + ", "+Impuestos+", "+TarAd+", "+CostTot+",
TRUE)";
            PreparedStatement insertBoleto = conn.prepareStatement(query);
            insertBoleto.executeUpdate();
        }
        return true;
    }catch (SQLException ex) {
        JOptionPane.showMessageDialog(null, ex);
        return false;
    }
}
```

## Método btn_Modificar_VentaActionPerformed

El método **btn_Modificar_VentaActionPerformed** se encarga de manejar la acción de modificar una venta en la interfaz gráfica. Este método es invocado cuando se presiona el botón correspondiente para realizar dicha modificación.

- **Parámetros:**

- **evt**: Objeto que contiene información sobre el evento de acción.

- **Funcionamiento:**
  - Verifica si se ha seleccionado correctamente un registro y si se han ingresado los nuevos datos.
  - Obtiene los identificadores del itinerario (**idIT**) y de la tarjeta de pasajero (**idTP**) seleccionados en los componentes gráficos.
  - Obtiene el número de boletos asociados a la venta actual mediante el método **CountBoletosCurrentID**.
  - Obtiene la fila seleccionada en la tabla de ventas y extrae el itinerario asociado a esa fila.
  - Comprueba si hay suficientes asientos disponibles en el nuevo itinerario (**idIT**) o si el itinerario seleccionado no ha cambiado.
  - Actualiza la información de la venta en la base de datos, incluyendo la tarjeta de pasajero, el itinerario y el estado de pago.
  - Borra los boletos asociados a la venta actual mediante el método **BorrarBoletos**.
  - Actualiza el monto total de la venta a 0.
  - Obtiene el costo base del vuelo asociado al nuevo itinerario mediante el método **getCostoBaseVuelo**.
  - Llama al método **insertarBoletos** para insertar nuevamente los boletos asociados a la venta con la nueva información.
  - Consulta y actualiza la información de las ventas en la interfaz gráfica.
  - Restaura la interfaz gráfica a su estado original.

```
private void btn_Modificar_VentaActionPerformed(java.awt.event.ActionEvent
evt) {//GEN-FIRST:event_btn_Modificar_VentaActionPerformed
    if (currentID.equals("")||JCB_TarjetaVenta.getSelectedIndex() == -
1||JCB_ItinerarioVenta.getSelectedIndex() == -1) {
        String mensaje = "Selecciona correctamente un registro y sus nuevos
datos";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }
    String idIT =
listIDItinerario.get(JCB_ItinerarioVenta.getSelectedIndex()).toString();
    String idTP =
listIDTrjetaPasajero.get(JCB_TarjetaVenta.getSelectedIndex()).toString();
```

```
    int numBoletos = CountBoletosCurrentID();
    DefaultTableModel model = (DefaultTableModel) JT_Venta.getModel();
    int selectedRow = JT_Venta.getSelectedRow();
    String itinerarioSelectedRow = model.getValueAt(selectedRow,
model.findColumn("Itinerario")).toString();

    if (VerificarDispAsientos(idIT, numBoletos)|| itinerarioSelectedRow ==
JCB_ItinerarioVenta.getSelectedItem().toString())
    {
        String query = JCHBX_PagadoVenta.isSelected()?
                    "UPDATE InfoPasajero.Venta SET
idTarjetaPasajero="+idTP+", idItinerario="+idIT+", MontoTotal=0,
EstadoPago=TRUE WHERE idVenta="+currentID:
                    "UPDATE InfoPasajero.Venta SET idTarjetaPasajero=" +
idTP + ", idItinerario=" + idIT + ", MontoTotal=0, EstadoPago=FALSE WHERE
idVenta=" + currentID;
        if (Query1(query)!=-1)
        {
            BorrarBoletos(currentID);
            query = "UPDATE InfoPasajero.Venta SET MontoTotal=0 WHERE
idVenta=" + currentID;
            Query1(query);
            String CostVuel = getCostoBaseVuelo(idIT);
            CostVuel = CostVuel.replaceAll("[\\$,]", "");
            int id = Integer.parseInt(currentID);

            if (!insertarBoletos(id, idIT, idTP, CostVuel, numBoletos))
            {
                String mensaje = "Ocurrió una excepción: \nNo se pudo
insertar los boletos";
                int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
                JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
                return;
            }
            ConsultaVenta();
            btn_backVentaActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
        }

    }else{
        String mensaje = "Ocurrió una excepción: \nNo se pudo modificar
porque no hay suficientes asientos disponibles";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
```

```
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }
}//GEN-LAST:event_btn_Modificar_VentaActionPerformed
```

### Método BorrarBoletos

El método **BorrarBoletos** elimina los boletos asociados a una venta específica. Este método realiza una operación de eliminación en la base de datos para borrar todos los registros de la tabla **InfoPasajero.Boleto** donde el campo **idVenta** coincide con el valor proporcionado en el parámetro **currentID**.

- **Parámetros:**
    - **currentID**: Un identificador que representa la venta cuyos boletos se deben eliminar.
- **Funcionamiento:**
    - Establece una conexión a la base de datos utilizando los parámetros de conexión (**URL**, **USER**, **PASS**).
    - Construye una consulta SQL de eliminación que borra todos los boletos asociados a la venta identificada por **currentID**.
    - Ejecuta la operación de eliminación.
    - Cierra los recursos de conexión (**PreparedStatement**) para liberar los recursos.
    - Maneja excepciones de SQL imprimiendo la traza de errores en la consola.

```
private void BorrarBoletos(String currentID) {
    try {
        Connection conn = DriverManager.getConnection(URL, USER, PASS);
        String query = "DELETE FROM InfoPasajero.Boleto WHERE idVenta = " +
currentID;
        PreparedStatement delBoleto = conn.prepareStatement(query);
        delBoleto.executeUpdate();
        delBoleto.close();
        conn.close();
    } catch (SQLException e) {
        e.printStackTrace();
        return;
    }
}
```

## Método CountBoletosCurrentID

El método **CountBoletosCurrentID** se encarga de contar la cantidad de boletos asociados a una venta específica. Este método realiza una consulta a la base de datos para obtener el conteo de boletos asociados a la venta identificada por **currentID**.

- **Retorno:**
    - Un entero que representa la cantidad de boletos asociados a la venta.
- **Funcionamiento:**
    - Establece una conexión a la base de datos utilizando los parámetros de conexión (**URL**, **USER**, **PASS**).
    - Construye una consulta SQL que cuenta la cantidad de registros en la tabla **InfoPasajero.Boleto** donde el campo **idVenta** coincide con el valor de **currentID**.
    - Ejecuta la consulta y obtiene el resultado.
    - Si hay un resultado, devuelve la cantidad de boletos; de lo contrario, retorna 0.
    - Maneja excepciones de SQL imprimiendo la traza de errores en la consola.

```
private int CountBoletosCurrentID() {
    try (Connection conexion = DriverManager.getConnection(URL, USER, PASS))
{
        String consulta = "SELECT COUNT(*) FROM InfoPasajero.Boleto WHERE
idVenta = " + currentID;
        try (PreparedStatement pstmt = conexion.prepareStatement(consulta))
{
            try (ResultSet rs = pstmt.executeQuery()) {
                if (rs.next()) {
                    int count = rs.getInt(1);
                    return count;
                }
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return 0;
}
```

## Método btn_Eliminar_VentaActionPerformed

El método **btn_Eliminar_VentaActionPerformed** es un manejador de eventos que se ejecuta cuando se hace clic en el botón "Eliminar Venta". Este método realiza la eliminación de una venta seleccionada junto con sus boletos asociados.

- **Parámetros:**

- **evt**: Un objeto **ActionEvent** que representa el evento de clic en el botón.

- **Funcionamiento:**
    - Verifica si **currentID** (identificador de la venta actualmente seleccionada) está vacío.
    - Construye una consulta SQL para eliminar la venta y sus boletos asociados utilizando el identificador **currentID**.
    - Ejecuta la consulta utilizando el método **Query1**.
    - Si la operación de eliminación es exitosa, actualiza la visualización de las ventas llamando a **ConsultaVenta()** y restablece la interfaz llamando a **btn_backVentaActionPerformed**.

```
private void btn_Eliminar_VentaActionPerformed(java.awt.event.ActionEvent
evt) {//GEN-FIRST:event_btn_Eliminar_VentaActionPerformed
    if (currentID.equals("")) {
        String mensaje = "Selecciona correctamente un registro";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }

    String query = "DELETE FROM InfoPasajero.Venta WHERE idVenta = " +
currentID;
    if (Query1(query) == -1) {
        String mensaje = "Ocurrió una excepción\nAún existen boletos
asociados a esa venta";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }
    ConsultaVenta();
    btn_backVentaActionPerformed(new ActionEvent(this,
ActionEvent.ACTION_PERFORMED, null));
}//GEN-LAST:event_btn_Eliminar_VentaActionPerformed
```

### Método btn_backVentaActionPerformed

El método **btn_backVentaActionPerformed** en un formulario de operaciones de venta realiza las siguientes acciones al hacer clic en el botón "Volver":

- Oculta **lbl_Modifica**.
- Habilita **btn_Agregar_Venta**.

- Deshabilita **btn_Eliminar_Venta** y **btn_Modificar_Venta**.
- Deshabilita **btn_backVenta**.
- Restablece controles de entrada, combo boxes y valores relacionados con la venta.
- Establece **currentID** como cadena vacía.

```
private void btn_backVentaActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_btn_backVentaActionPerformed
    lbl_Modifica.setVisible(false);
    btn_Agregar_Venta.setEnabled(true);
    btn_Eliminar_Venta.setEnabled(false);
    btn_Modificar_Venta.setEnabled(false);
    btn_backVenta.setEnabled(false);
    JS_NumBoletos.setEnabled(true);
    JS_IVABoleto.setEnabled(true);
    tb_TSBoleto.setEnabled(true);
    tb_TSRBoleto.setEnabled(true);

    JCB_ItinerarioVenta.setSelectedIndex(-1);
    JCB_TarjetaVenta.setSelectedIndex(-1);
    JS_NumBoletos.setValue(1);
    JCHBX_PagadoVenta.setSelected(false);

    currentID = "";
}//GEN-LAST:event_btn_backVentaActionPerformed
```

## Método JT_VentaMouseClicked

**JT_VentaMouseClicked** es un *event handler* que se activa al hacer clic en una fila de la tabla de ventas (**JT_Venta**). Su función principal es actualizar la interfaz de usuario al seleccionar una venta específica. Esto incluye la visualización de ciertos componentes, la desactivación y activación de botones, y la carga de información relacionada con la venta seleccionada en controles como listas desplegables y casillas de verificación.

```
private void JT_VentaMouseClicked(java.awt.event.MouseEvent evt) {//GEN-
FIRST:event_JT_VentaMouseClicked
    lbl_Modifica.setVisible(true);
    btn_Agregar_Venta.setEnabled(false);
    btn_Eliminar_Venta.setEnabled(true);
    btn_Modificar_Venta.setEnabled(true);
    btn_backVenta.setEnabled(true);
    JS_NumBoletos.setEnabled(false);
    JS_IVABoleto.setEnabled(false);
    tb_TSBoleto.setEnabled(false);
    tb_TSRBoleto.setEnabled(false);
```

```
    int filaSeleccionada = JT_Venta.getSelectedRow();
    currentID = JT_Venta.getValueAt(filaSeleccionada, 0).toString();
    String tarjeta = JT_Venta.getValueAt(filaSeleccionada, 1).toString();
    String itinerario = JT_Venta.getValueAt(filaSeleccionada, 2).toString();
    String pagado = JT_Venta.getValueAt(filaSeleccionada, 5).toString();


    if(pagado.equals("f"))
    {
        JCHBX_PagadoVenta.setSelected(false);
    }else
    {
        JCHBX_PagadoVenta.setSelected(true);
    }

    for (int i = 0; i < JCB_TarjetaVenta.getItemCount(); i++) {
        String item = (String) JCB_TarjetaVenta.getItemAt(i);

        if (item.equals(tarjeta)) {
            JCB_TarjetaVenta.setSelectedIndex(i);
            break;
        }
    }

    for (int i = 0; i < JCB_ItinerarioVenta.getItemCount(); i++) {
        String item = (String) JCB_ItinerarioVenta.getItemAt(i);

        if (item.equals(itinerario)) {
            JCB_ItinerarioVenta.setSelectedIndex(i);
            break;
        }
    }
}//GEN-LAST:event_JT_VentaMouseClicked
```

## Método LlenarCBVenta

El método **LlenarCBVenta** se encarga de llenar los elementos de dos JComboBox en un formulario de venta. Para ello, realiza dos consultas a la base de datos:

1. **Consulta de Itinerarios:**
    - Obtiene información sobre los itinerarios, como el ID del itinerario, el ID del vuelo, la hora de salida y la fecha del vuelo.
    - Formatea la información obtenida y la agrega al JComboBox **JCB_ItinerarioVenta**.

2. **Consulta de Tarjetas de Pasajero:**

    - Obtiene información sobre las tarjetas de pasajero, como el ID de la tarjeta, el nombre del titular y el número de tarjeta.
    - Formatea la información obtenida y la agrega al JComboBox **JCB_TarjetaVenta**.

Además, el método mantiene listas (**listIDItinerario** y **listIDTrjetaPasajero**) que contienen los ID correspondientes a los elementos seleccionados en los JComboBox para su uso posterior.

```
public void LlenarCBVenta() {
    try {
        Connection conn = DriverManager.getConnection(URL, USER, PASS);
        String query = "SELECT idItinerario, idVuelo, HoraSalida, FechaVuelo
FROM InfoAerolinea.Itinerario";
        try (PreparedStatement pstmt = conn.prepareStatement(query);
ResultSet rs = pstmt.executeQuery()) {
            while (rs.next()) {
                String horaCompleta = rs.getString("HoraSalida");
                String horaSinSegundos = horaCompleta.substring(0, 5);

                Date fecha = rs.getDate("FechaVuelo");
                SimpleDateFormat dateFormat = new SimpleDateFormat("MM-dd-
yyyy");
                String nuevaFecha = dateFormat.format(fecha);

                String item = "(" + getVuelo(rs.getString("idVuelo"), true)
+ ") - " + horaSinSegundos + " - " + nuevaFecha;
                JCB_ItinerarioVenta.addItem(item);
                listIDItinerario.add(rs.getInt("idItinerario"));
            }
        }

        query = "SELECT idTarjetaPasajero, NombreTitular, NumTarjeta FROM
InfoPasajero.TarjetaPasajero";
        try (PreparedStatement pstmt = conn.prepareStatement(query);
ResultSet rs = pstmt.executeQuery()) {
            while (rs.next()) {
                String numTarjeta = rs.getString("NumTarjeta");
                String ultimosTresDigitos =
numTarjeta.substring(numTarjeta.length() - 3);
                JCB_TarjetaVenta.addItem(rs.getString("NombreTitular") + " -
" + ultimosTresDigitos);
                listIDTrjetaPasajero.add(rs.getInt("idTarjetaPasajero"));
            }
        }
```

```
        conn.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

### Método ConsultaVenta

El método **ConsultaVenta** se encarga de obtener y mostrar información actualizada sobre las ventas de pasajes en una tabla. Establece conexión con la base de datos, ejecuta una consulta SQL para obtener los datos relevantes de la tabla **InfoPasajero.Venta**, y actualiza la interfaz gráfica eliminando las filas existentes en la tabla y agregando las nuevas. El método maneja posibles errores y muestra mensajes de alerta al usuario en caso de excepciones.

```
public void ConsultaVenta() {
    try {
        CONN = ConnectBD();
        java.sql.Statement Stmnt = CONN.createStatement();
        String Query = "SELECT * FROM InfoPasajero.Venta";
        String[] Data = new String[6];
        ResultSet Columns = Stmnt.executeQuery(Query);
        DefaultTableModel model = (DefaultTableModel) JT_Venta.getModel();

        // Eliminar todas las filas existentes
        model.setRowCount(0);
        while (Columns.next()) {
            Data[0] = Columns.getString(1);
            Data[1] = getTarjetaPasajero(Columns.getString(2));
            Data[2] = getItinerario(Columns.getString(3));
            Data[3] = Columns.getString(4);
            Data[4] = Columns.getString(5);
            Data[5] = Columns.getString(6);
            model.addRow(Data);
        }
        Stmnt.close();
        CONN.close();
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(null, ex);
    }
}
```

### Método getTarjetaPasajero

El método **getTarjetaPasajero** recibe como parámetro el ID de una tarjeta de pasajero (**idTP**). Este método realiza una consulta a la base de datos para obtener el nombre del titular y los últimos tres dígitos del número de tarjeta asociados a esa tarjeta de pasajero. Luego, combina esta información

en un formato específico (**NombreTitular - Últimos tres dígitos**) y retorna este valor. En caso de error, el método retorna **null**.

```
private String getTarjetaPasajero(String idTP) {
    try (Connection conexion = DriverManager.getConnection(URL, USER, PASS))
{
        String consulta = "SELECT NombreTitular, NumTarjeta FROM
InfoPasajero.TarjetaPasajero WHERE idTarjetaPasajero=" + idTP;
        try (PreparedStatement pstmt = conexion.prepareStatement(consulta))
{
            try (ResultSet rs = pstmt.executeQuery()) {
                if (rs.next()) {
                    String numTarjeta = rs.getString("NumTarjeta");
                    String ultimosTresDigitos =
numTarjeta.substring(numTarjeta.length() - 3);
                    String TarjPas = rs.getString("NombreTitular") + " - " +
ultimosTresDigitos;
                    return TarjPas;
                }
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return null; // Retorna null en caso de error
}
```

### Método tb_TSBoletoKeyTyped y Método tb_TSRBoletoKeyTyped

Estos dos métodos (**tb_TSBoletoKeyTyped** y **tb_TSRBoletoKeyTyped**) son *event handlers* que responden al evento de tecla escrita en los campos de texto **tb_TSBoleto** y **tb_TSRBoleto**. Aquí hay un resumen de lo que hacen:

1. **tb_TSBoletoKeyTyped Método:**
    - Se ejecuta cuando se presiona una tecla en el campo de texto **tb_TSBoleto**.
    - Verifica si el carácter ingresado no es un dígito o la tecla de retroceso (**\b**).
    - Consume el evento (no lo procesa) si el carácter no es un dígito o si la longitud del texto ya es de 10 caracteres.
2. **tb_TSRBoletoKeyTyped Método:**
    - Se ejecuta cuando se presiona una tecla en el campo de texto **tb_TSRBoleto**.
    - Verifica si el carácter ingresado no es un dígito o la tecla de retroceso (**\b**).

- Consume el evento (no lo procesa) si el carácter no es un dígito o si la longitud del texto ya es de 10 caracteres.

Estos métodos aseguran que en los campos de texto **tb_TSBoleto** y **tb_TSRBoleto** solo se puedan ingresar caracteres numéricos y que la longitud del texto no exceda los 10 caracteres.

```
private void tb_TSBoletoKeyTyped(java.awt.event.KeyEvent evt) {//GEN-
FIRST:event_tb_TSBoletoKeyTyped
    char c = evt.getKeyChar();
    // Verificar si el carácter es un número o la tecla de borrar
    if (!Character.isDigit(c) && c != '\b') {
        evt.consume(); // Consumir el evento si no es un número o la tecla
de borrar
    }

    // Verificar si ya hay 10 caracteres en el campo
    if (tb_TSBoleto.getText().length() >= 10) {
        evt.consume(); // Consumir el evento si ya hay 10 caracteres
    }
}//GEN-LAST:event_tb_TSBoletoKeyTyped
```

```
private void tb_TSRBoletoKeyTyped(java.awt.event.KeyEvent evt) {//GEN-
FIRST:event_tb_TSRBoletoKeyTyped
    char c = evt.getKeyChar();
    // Verificar si el carácter es un número o la tecla de borrar
    if (!Character.isDigit(c) && c != '\b') {
        evt.consume(); // Consumir el evento si no es un número o la tecla
de borrar
    }

    // Verificar si ya hay 10 caracteres en el campo
    if (tb_TSRBoleto.getText().length() >= 10) {
        evt.consume(); // Consumir el evento si ya hay 10 caracteres
    }
}//GEN-LAST:event_tb_TSRBoletoKeyTyped
```

## Ventana Boleto

### Método ConsultaBoleto

**ConsultaBoleto** es un método que se encarga de realizar una consulta a la base de datos para obtener información sobre los boletos. Aquí hay un resumen de su funcionalidad:

1. **Limpieza de Listas**: Se limpia la lista **listIDAsiento2** para almacenar nuevos identificadores de asientos.

2. **Conexión a la Base de Datos**: Se establece una conexión con la base de datos.
3. **Ejecución de la Consulta SQL**: Se ejecuta una consulta SQL para obtener información de los boletos almacenados en la tabla **InfoPasajero.Boleto**.
4. **Llenado del Modelo de la Tabla**: Los resultados de la consulta se utilizan para llenar el modelo de la tabla **JT_Boleto**. Cada fila de la tabla representa un boleto y contiene información como el ID del boleto, el ID del pasajero asociado, el nombre del pasajero, el ID del asiento, el estado del boleto, el impuesto, las tarifas adicionales, y el costo total.
5. **Almacenamiento de Identificadores de Asientos**: Se almacenan los identificadores de los asientos en la lista **listIDAsiento2**.
6. **Manejo de Excepciones**: Se manejan las excepciones de manera general, mostrando un cuadro de diálogo con el mensaje de error en caso de que ocurra una excepción.

```
public void ConsultaBoleto() {
    try {
        listIDAsiento2.clear();
        CONN = ConnectBD();
        java.sql.Statement Stmnt = CONN.createStatement();
        String Query = "SELECT * FROM InfoPasajero.Boleto";
        String[] Data = new String[8];
        ResultSet Columns = Stmnt.executeQuery(Query);
        DefaultTableModel model = (DefaultTableModel) JT_Boleto.getModel();

        // Eliminar todas las filas existentes
        model.setRowCount(0);
        while (Columns.next()) {
            Data[0] = Columns.getString(1);
            Data[1] = Columns.getString(2);
            if(Columns.getString(3)==null)
            {
                Data[2] = "Por Capturar";
            }else
            {
                Data[2] = getPasajero(Columns.getString(3), true);
            }
            listIDAsiento2.add(Integer.parseInt(Columns.getString(4)));
            Data[3] = getAsiento(Columns.getString(4));
            Data[4] = Columns.getString(5);
            Data[5] = Columns.getString(6);
            Data[6] = Columns.getString(7);
            Data[7] = Columns.getString(8);
            model.addRow(Data);
        }
        Stmnt.close();
```

```
        CONN.close();
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(null, ex);
    }
}
```

### Método getAsiento

**getAsiento** es un método que busca y retorna información sobre un asiento específico en la base de datos. Aquí tienes un resumen de su funcionalidad:

1. **Conexión a la Base de Datos**: Se establece una conexión con la base de datos.
2. **Consulta SQL para Obtener el Asiento**: Se ejecuta una consulta SQL para obtener información sobre el asiento correspondiente al ID de asiento proporcionado.
3. **Procesamiento de Resultados**: Se procesan los resultados de la consulta para obtener información específica del asiento, como el ID del itinerario, la letra del asiento y el número del asiento.
4. **Obtención del Itinerario del Asiento**: Se utiliza el método **getItinerario** para obtener el itinerario asociado al asiento y se formatea para mostrarlo de manera legible.
5. **Formateo y Retorno de la Información del Asiento**: Se formatea la información del asiento (itinerario, letra y número) y se retorna como una cadena.
6. **Manejo de Excepciones**: Se manejan las excepciones de manera general, imprimiendo la traza de errores en caso de que ocurra una excepción.

```
private String getAsiento(String idasiento) {
    try (Connection conexion = DriverManager.getConnection(URL, USER, PASS))
{
        String consulta = "SELECT * FROM InfoPasajero.Asiento WHERE
idAsiento=" + idasiento;
        try (PreparedStatement pstmt = conexion.prepareStatement(consulta))
{
            try (ResultSet rs = pstmt.executeQuery()) {
                if (rs.next()) {
                    String itinerario =
getItinerario(rs.getString("idItinerario"));
                    Matcher matcher =
Pattern.compile("\\(([^)]*)\\)").matcher(itinerario);
                    if (matcher.find()) {
                        itinerario = matcher.group(1);
                    }
                    String asiento = itinerario + " - " +
rs.getString("Letra") + "-" + rs.getString("Num_Asiento");
                    return asiento;
```

```
                }
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return null; // Retorna null en caso de error
}
```

### Método JT_BoletoMouseClicked

El método **JT_BoletoMouseClicked** realiza las siguientes acciones cuando se hace clic en una fila de la tabla **JT_Boleto**:

1. **Limpiar Elementos del ComboBox**: Elimina todos los elementos de los ComboBox **JCB_PasajeroBoleto** y **JCB_AsientoBoleto**.
2. **Habilitar ComboBox**: Habilita los ComboBox **JCB_PasajeroBoleto** y **JCB_AsientoBoleto** para su interacción.
3. **Mostrar Etiqueta y Habilitar Botones**: Hace visible la etiqueta **lbl_Modifica** y habilita los botones **btn_Eliminar_Boleto** y **btn_Modificar_Boleto**.
4. **Obtener Datos de la Fila Seleccionada**: Obtiene la información de la fila seleccionada, incluyendo el identificador del boleto (**currentID**), el pasajero, el asiento y el estado del boleto.
5. **Configuración de Estado del CheckBox**: Configura el estado del **JCHBX_EstadoBoleto** según el estado obtenido de la fila.
6. **Llenar ComboBox de Pasajeros y Asientos**: Llama al método **LlenarCBBoleto** para llenar los ComboBox de pasajeros y asientos con información actualizada, tomando en cuenta el asiento asociado al boleto seleccionado.
7. **Seleccionar Elementos en ComboBox**: Selecciona el pasajero en el ComboBox **JCB_PasajeroBoleto** según la información obtenida de la fila. Además, establece la selección en -1 para ambos ComboBox al final del método.

```
private void JT_BoletoMouseClicked(java.awt.event.MouseEvent evt) {//GEN-
FIRST:event_JT_BoletoMouseClicked

    JCB_PasajeroBoleto.removeAllItems();
    JCB_AsientoBoleto.removeAllItems();
    JCB_PasajeroBoleto.setEnabled(true);
    JCB_AsientoBoleto.setEnabled(true);
    lbl_Modifica.setVisible(true);
    btn_Eliminar_Boleto.setEnabled(true);
    btn_Modificar_Boleto.setEnabled(true);
    JCHBX_EstadoBoleto.setEnabled(true);
```

```
    int filaSeleccionada = JT_Boleto.getSelectedRow();
    currentID = JT_Boleto.getValueAt(filaSeleccionada, 0).toString();
    String pasajero = JT_Boleto.getValueAt(filaSeleccionada, 1).toString();
    String asiento = JT_Boleto.getValueAt(filaSeleccionada, 2).toString();
    String estado = JT_Boleto.getValueAt(filaSeleccionada, 5).toString();


    if(estado.equals("f"))
    {
        JCHBX_EstadoBoleto.setSelected(false);
    }else
    {
        JCHBX_EstadoBoleto.setSelected(true);
    }

    String idAsiento =
listIDAsiento2.get(JT_Boleto.getSelectedRow()).toString();
    LlenarCBBoleto(idAsiento);

    for (int i = 0; i < JCB_PasajeroBoleto.getItemCount(); i++) {
        String item = (String) JCB_PasajeroBoleto.getItemAt(i);
        if (item.equals(pasajero)) {
            JCB_PasajeroBoleto.setSelectedIndex(i);
            break;
        }
    }
    JCB_PasajeroBoleto.setSelectedIndex(-1);
    JCB_AsientoBoleto.setSelectedIndex(-1);
}//GEN-LAST:event_JT_BoletoMouseClicked
```

## Método LlenarCBBoleto

**LlenarCBBoleto** es un método que se encarga de poblar ciertos elementos de la interfaz relacionados con la venta de boletos. Aquí está un resumen de su funcionalidad:

1. **Limpieza de Listas**: Se limpian las listas **listIDAsiento** para almacenar nuevos identificadores de asientos.
2. **Conexión a la Base de Datos**: Se establece una conexión con la base de datos.
3. **Obtención del Itinerario del Asiento**: Se realiza una consulta para obtener el itinerario asociado al asiento proporcionado.

4. **Llenado del ComboBox de Asientos Disponibles**: Se realiza una consulta para obtener los asientos disponibles en el mismo itinerario. La información se agrega al ComboBox **JCB_AsientoBoleto**, y se almacenan los identificadores de los asientos en **listIDAsiento**.
5. **Llenado del ComboBox de Pasajeros**: Se realiza una consulta para obtener información sobre los pasajeros. La información se agrega al ComboBox **JCB_PasajeroBoleto**.
6. **Manejo de Excepciones**: Se manejan las excepciones de SQL, imprimiendo la traza de la excepción en caso de un error.

```
public void LlenarCBBoleto(String Asiento) {
    try {
        listIDAsiento.clear();
        Connection conn = DriverManager.getConnection(URL, USER, PASS);

        String itinerario = "";
        String query = "SELECT idItinerario FROM InfoPasajero.Asiento WHERE
idAsiento=" + Asiento;
        try (PreparedStatement pstmt = conn.prepareStatement(query);
ResultSet rs = pstmt.executeQuery()) {
            while (rs.next()) {
                itinerario = rs.getString("idItinerario");
            }
        }

        query = "SELECT * FROM InfoPasajero.Asiento WHERE idItinerario="+
itinerario + " AND Ocupado=FALSE";
        try (PreparedStatement pstmt = conn.prepareStatement(query);
ResultSet rs = pstmt.executeQuery()) {
            while (rs.next()) {
                itinerario = getItinerario(rs.getString("idItinerario"));
                Matcher matcher =
Pattern.compile("\\(([^)]*)\\)").matcher(itinerario);
                if (matcher.find()) {
                    itinerario = matcher.group(1);
                }
                JCB_AsientoBoleto.addItem(itinerario + " - " +
rs.getString("Letra") + "-" + rs.getString("Num_Asiento"));
                listIDAsiento.add(rs.getInt("idAsiento"));
            }
        }

        query = "SELECT Nom_Pasajero, NumPasaporte FROM
InfoPasajero.Pasajero";
        try (PreparedStatement pstmt = conn.prepareStatement(query);
ResultSet rs = pstmt.executeQuery()) {
```

```
            while (rs.next()) {
                String digitnumpass =
rs.getString("NumPasaporte").substring(rs.getString("NumPasaporte").length()
- 4);
                JCB_PasajeroBoleto.addItem(digitnumpass + " - " +
rs.getString("Nom_Pasajero"));
            }
        }

    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

### Método btn_Modificar_BoletoActionPerformed

El método **btn_Modificar_BoletoActionPerformed** realiza las siguientes acciones al ser ejecutado:

1. **Validación de Selección**: Verifica si se ha seleccionado correctamente un registro antes de continuar. Si no se ha seleccionado, muestra un mensaje informativo y sale del método.
2. **Construcción de la Consulta SQL**: Construye una consulta SQL para actualizar la información del boleto en la base de datos. La construcción de la consulta depende de las selecciones realizadas en los JComboBox (cuadro combinado) y del estado del JCheckBox (casilla de verificación).
3. **Ejecución de la Consulta SQL**: Ejecuta la consulta SQL construida mediante el método **Query1**. Si la ejecución de la consulta falla, muestra un mensaje de error.
4. **Consulta de Boletos**: Llama al método **ConsultaBoleto** para actualizar la tabla de boletos en la interfaz gráfica.
5. **Restauración de la Interfaz**: Llama al método **backboleto** para restaurar la interfaz a su estado original.

```
private void btn_Modificar_BoletoActionPerformed(java.awt.event.ActionEvent
evt) {//GEN-FIRST:event_btn_Modificar_BoletoActionPerformed
    if (currentID.equals("")) {
        String mensaje = "Selecciona correctamente un registro";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }

    String query = "";
```

```
    if(JCB_AsientoBoleto.getSelectedIndex() != -1 &&
JCB_PasajeroBoleto.getSelectedIndex() != -1)
    {
        String idA =
listIDAsiento.get(JCB_AsientoBoleto.getSelectedIndex()).toString();
        String idPas =
getPasajero(JCB_PasajeroBoleto.getSelectedItem().toString(), false);
        query = JCHBX_EstadoBoleto.isSelected() ?
            "UPDATE InfoPasajero.Boleto SET idPasajero=" + idPas + ",
idAsiento = " + idA + ", Estado = TRUE WHERE idBoleto = " +currentID :
            "UPDATE InfoPasajero.Boleto SET idPasajero=" + idPas + ",
idAsiento = " + idA + ", Estado = FALSE WHERE idBoleto = " + currentID;
    }

    if(JCB_AsientoBoleto.getSelectedIndex() != -1 &&
JCB_PasajeroBoleto.getSelectedIndex() == -1)
    {
        String idA =
listIDAsiento.get(JCB_AsientoBoleto.getSelectedIndex()).toString();
        query = JCHBX_EstadoBoleto.isSelected()?
                "UPDATE InfoPasajero.Boleto SET idAsiento = " + idA + ",
Estado = TRUE WHERE idBoleto = " + currentID :
                "UPDATE InfoPasajero.Boleto SET idAsiento = " + idA + ",
Estado = FALSE WHERE idBoleto = " + currentID;

    }

    if(JCB_AsientoBoleto.getSelectedIndex() == -1 &&
JCB_PasajeroBoleto.getSelectedIndex() != -1)
    {
        String idPas =
getPasajero(JCB_PasajeroBoleto.getSelectedItem().toString(), false);
        query = JCHBX_EstadoBoleto.isSelected()?
            "UPDATE InfoPasajero.Boleto SET idPasajero=" + idPas + ", Estado
= TRUE WHERE idBoleto = " + currentID :
            "UPDATE InfoPasajero.Boleto SET idPasajero=" + idPas + ", Estado
= FALSE WHERE idBoleto = " + currentID;
    }

    if (JCB_AsientoBoleto.getSelectedIndex() == -1 &&
JCB_PasajeroBoleto.getSelectedIndex() == -1)
    {
        query = JCHBX_EstadoBoleto.isSelected() ?
                "UPDATE InfoPasajero.Boleto SET Estado = TRUE WHERE idBoleto
= " + currentID :
```

```
                "UPDATE InfoPasajero.Boleto SET Estado = FALSE WHERE
idBoleto = " + currentID;
    }

    if (Query1(query) == -1) {
        String mensaje = "No se pudo modificar";
        JOptionPane.showMessageDialog(null, mensaje, "Error",
JOptionPane.ERROR_MESSAGE);
    }

    ConsultaBoleto();
    backboleto();
}//GEN-LAST:event_btn_Modificar_BoletoActionPerformed
```

### Método btn_Eliminar_BoletoActionPerformed

El método **btn_Eliminar_BoletoActionPerformed** realiza las siguientes acciones al ser ejecutado:

1. **Validación de Selección**: Verifica si se ha seleccionado correctamente un registro antes de continuar. Si no se ha seleccionado, muestra un mensaje informativo y sale del método.
2. **Construcción de la Consulta SQL**: Construye una consulta SQL para eliminar el boleto de la base de datos. La construcción de la consulta utiliza el identificador (**idBoleto**) del boleto actualmente seleccionado.
3. **Ejecución de la Consulta SQL**: Ejecuta la consulta SQL construida mediante el método **Query1**. Si la ejecución de la consulta falla, muestra un mensaje de error.
4. **Consulta de Boletos**: Llama al método **ConsultaBoleto** para actualizar la tabla de boletos en la interfaz gráfica.
5. **Restauración de la Interfaz**: Llama al método **backboleto** para restaurar la interfaz a su estado original.

```
private void btn_Eliminar_BoletoActionPerformed(java.awt.event.ActionEvent
evt) {//GEN-FIRST:event_btn_Eliminar_BoletoActionPerformed
    if (currentID.equals("")) {
        String mensaje = "Selecciona correctamente un registro";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }

    String query = "DELETE FROM InfoPasajero.Boleto WHERE idBoleto = " +
currentID;
    if (Query1(query) == -1) {
        String mensaje = "No se pudo eliminar";
```

```
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }
    ConsultaBoleto();
    backboleto();
}//GEN-LAST:event_btn_Eliminar_BoletoActionPerformed
```

### Método backboleto

El método **backboleto** realiza las siguientes acciones:

1. **Limpiar ComboBox**: Elimina elementos de los ComboBox **JCB_PasajeroBoleto** y **JCB_AsientoBoleto**.
2. **Deshabilitar ComboBox**: Deshabilita ComboBox **JCB_PasajeroBoleto** y **JCB_AsientoBoleto**.
3. **Ocultar Etiqueta y Deshabilitar Botones**: Oculta la etiqueta **lbl_Modifica** y deshabilita los botones **btn_Eliminar_Boleto** y **btn_Modificar_Boleto**.
4. **Deshabilitar CheckBox**: Deshabilita el **JCHBX_EstadoBoleto**.
5. **Limpiar ID Actual**: Establece **currentID** en una cadena vacía.

En resumen, el método **backboleto** restablece la interfaz gráfica a su estado predeterminado después de realizar operaciones en la gestión de boletos.

```
public void backboleto(){
    JCB_PasajeroBoleto.removeAllItems();
    JCB_AsientoBoleto.removeAllItems();
    JCB_PasajeroBoleto.setEnabled(false);
    JCB_AsientoBoleto.setEnabled(false);
    lbl_Modifica.setVisible(false);
    btn_Eliminar_Boleto.setEnabled(false);
    btn_Modificar_Boleto.setEnabled(false);
    JCHBX_EstadoBoleto.setEnabled(false);
    currentID = "";
}
```

## Métodos para Reportes

1. **LlenarCBR1()**
    - Este método llena un JComboBox (**JCB_PilotoR1**) con datos obtenidos de la base de datos. Parece obtener información de pilotos y sus identificadores para mostrarlos en el combo box.

```
public void LlenarCBR1()
```

```
{
    JCB_PilotoR1.removeAllItems();
    listNomPilotos.clear();
    try {
        Connection conn = DriverManager.getConnection(URL, USER, PASS);
        // Llenar ComboBox para Piloto
        String queryPiloto = "SELECT idPiloto, Nom_Piloto FROM
InfoAerolinea.Piloto";
        try (PreparedStatement pstmtPiloto =
conn.prepareStatement(queryPiloto); ResultSet rsPiloto =
pstmtPiloto.executeQuery()) {
            while (rsPiloto.next()) {
                JCB_PilotoR1.addItem(rsPiloto.getString("idPiloto") + " - "
+ rsPiloto.getString("Nom_Piloto"));
                listNomPilotos.add(rsPiloto.getString("Nom_Piloto"));
            }
        }
        conn.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

2. **tb_CostBaseR2KeyTyped(java.awt.event.KeyEvent evt)**

    - Este método es un escuchador de eventos de teclado para un JTextField (**tb_CostBaseR2**). Limita la entrada de caracteres a dígitos y un punto decimal. Además, asegura que solo haya un punto decimal y permite solo dos decimales.

```
private void tb_CostBaseR2KeyTyped(java.awt.event.KeyEvent evt) {//GEN-
FIRST:event_tb_CostBaseR2KeyTyped
    char c = evt.getKeyChar();

    // Permite solo dígitos y un punto decimal
    if (!Character.isDigit(c) && c != '.') {
        evt.consume();
    }

    // Asegura que solo haya un punto decimal
    if (c == '.' && tb_CostBaseR2.getText().contains(".")) {
        evt.consume();
    }

    // Permite solo dos decimales
    String text = tb_CostBaseR2.getText();
```

```
    if (text.contains(".") && text.split("\\.").length > 1 &&
text.split("\\.")[1].length() >= 2 && c != KeyEvent.VK_BACK_SPACE) {
        evt.consume();
    }
}//GEN-LAST:event_tb_CostBaseR2KeyTyped
```

3. **Rep2_consultaActionPerformed(java.awt.event.ActionEvent evt)**

    - Este método maneja el evento de clic en un botón (**Rep2_consulta**). Realiza una consulta a la base de datos utilizando información proporcionada por el usuario (monto base y fecha) y muestra los resultados en una tabla (**JT_R2**).

```
private void Rep2_consultaActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_Rep2_consultaActionPerformed
    if (JC_FechaR2.getDate() == null||tb_CostBaseR2.getText().equals("")) {
        String mensaje = "Ingresa correctamente los datos";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }

    try{
        CONN = ConnectBD();
        java.sql.Statement Stmnt = CONN.createStatement();

        JCalendar fecha = JC_FechaR2;
        Date fechaSeleccionada = new Date(fecha.getDate().getTime());
        String Query = JCHBX_PagadoR2.isSelected()?
                "SELECT\n" +
                "(\n" +
                "        SELECT NombreTitular\n" +
                "        FROM InfoPasajero.TarjetaPasajero\n" +
                "        WHERE idTarjetaPasajero = V.idTarjetaPasajero\n" +
                "    ) AS NombreTitular,\n" +
                "V.MontoTotal,\n" +
                "V.FechaVenta,\n" +
                "I.idItinerario\n" +
                "FROM\n" +
                "InfoPasajero.Venta V\n" +
                "INNER JOIN InfoAerolinea.Itinerario I ON V.idItinerario =
I.idItinerario\n" +
                "WHERE\n" +
                "V.MontoTotal >= "+tb_CostBaseR2.getText()+"::money\n" +
                "AND V.FechaVenta >= '"+fechaSeleccionada+"'\n" +
```

```
                "AND V.EstadoPago = TRUE":
                "SELECT\n" +
                "(\n" +
                "        SELECT NombreTitular\n" +
                "        FROM InfoPasajero.TarjetaPasajero\n" +
                "        WHERE idTarjetaPasajero = V.idTarjetaPasajero\n" +
                "    ) AS NombreTitular,\n" +
                "V.MontoTotal,\n" +
                "V.FechaVenta,\n" +
                "I.idItinerario\n" +
                "FROM\n" +
                "InfoPasajero.Venta V\n" +
                "INNER JOIN InfoAerolinea.Itinerario I ON V.idItinerario =
I.idItinerario\n" +
                "WHERE\n" +
                "V.MontoTotal >= "+tb_CostBaseR2.getText()+"::money\n" +
                "AND V.FechaVenta >= '"+fechaSeleccionada+"'\n" +
                "AND V.EstadoPago = FALSE";

        String[]Data = new String[5];
        ResultSet Columns = Stmnt.executeQuery (Query);
        DefaultTableModel model = (DefaultTableModel) JT_R2.getModel();

        // Eliminar todas las filas existentes
        model.setRowCount(0);
        while(Columns.next())
        {
            Data[0]=Columns.getString(1);
            Data[1]=Columns.getString(2);
            Data[2]=Columns.getString(3);
            Data[3]=getItinerario(Columns.getString(4));
            model.addRow(Data);
        }
        Stmnt.close();
        CONN.close();
    }catch (Exception ex) {
        JOptionPane.showMessageDialog(null, ex);
    }
    tb_CostBaseR2.setText("");
}//GEN-LAST:event_Rep2_consultaActionPerformed
```

4. **Rep1_consultaActionPerformed(java.awt.event.ActionEvent evt)**

    - Este método maneja el evento de clic en un botón (**Rep1_consulta**). Realiza una consulta más compleja a la base de datos utilizando información proporcionada por el

usuario (hora de salida, fecha y piloto) y muestra los resultados en una tabla (**JT_R1**). También actualiza un JLabel (**JL_Count**) con el número de registros encontrados.

```
private void Rep1_consultaActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_Rep1_consultaActionPerformed
    if (!ValidarHora(TF_HoraSalIR1.getText())||JC_FechaR1.getDate() ==
null||TF_HoraSalIR1.getText().equals("")||JCB_PilotoR1.getSelectedIndex() ==
-1) {
        String mensaje = "Ingresa correctamente los datos";
        int tipoMensaje = JOptionPane.INFORMATION_MESSAGE;
        JOptionPane.showMessageDialog(null, mensaje, "Mensaje",
tipoMensaje);
        return;
    }

    try{
        CONN = ConnectBD();
        java.sql.Statement Stmnt = CONN.createStatement();

        JCalendar fecha = JC_FechaR1;
        Date fechaSeleccionada = new Date(fecha.getDate().getTime());
        String nombrePiloto =
listNomPilotos.get(JCB_PilotoR1.getSelectedIndex()).toString();

        String Query =
        "SELECT\n" +
        "P.Nom_Piloto,\n" +
        "I.HoraSalida,\n" +
        "I.FechaVuelo,\n" +
        "A.Modelo AS ModeloAvion,\n" +
        "V.DuracionHoras,\n" +
        "V.CostoBase,\n" +
        "(\n" +
        "        SELECT COUNT(I2.idItinerario)\n" +
        "        FROM InfoAerolinea.Itinerario AS I2\n" +
        "        WHERE I2.idPiloto = ( SELECT idPiloto FROM
InfoAerolinea.Piloto WHERE Nom_Piloto = '"+nombrePiloto+"') AND
I2.FechaVuelo >= '"+fechaSeleccionada+"' AND I2.HoraSalida >=
'"+TF_HoraSalIR1.getText()+"'\n" +
        "    ) AS CantidadVuelos\n" +
        "FROM\n" +
        "InfoAerolinea.Piloto AS P\n" +
        "INNER JOIN\n" +
        "InfoAerolinea.Itinerario AS I ON P.idPiloto = I.idPiloto\n" +
        "INNER JOIN\n" +
        "InfoAerolinea.Avion AS A ON I.idAvion = A.idAvion\n" +
```

```
        "INNER JOIN\n" +
        "InfoAerolinea.Vuelo AS V ON I.idVuelo = V.idVuelo\n" +
        "WHERE\n" +
        "P.Nom_Piloto = '"+nombrePiloto+"' AND I.FechaVuelo >=
'"+fechaSeleccionada+"' AND I.HoraSalida >= '"+TF_HoraSalIR1.getText()+"'\n"
+
        "GROUP BY\n" +
        "P.Nom_Piloto, I.HoraSalida, I.FechaVuelo, A.Modelo,
V.DuracionHoras, V.CostoBase;";

        String[]Data = new String[7];
        ResultSet Columns = Stmnt.executeQuery (Query);
        DefaultTableModel model = (DefaultTableModel) JT_R1.getModel();

        // Eliminar todas las filas existentes
        model.setRowCount(0);
        while(Columns.next())
        {
            Data[0]=Columns.getString(1);
            Data[1]=Columns.getString(2);
            Data[2]=Columns.getString(3);
            Data[3]=Columns.getString(4);
            Data[4]=Columns.getString(5);
            Data[5]=Columns.getString(6);
            JL_Count.setText("Registros encontrados:
"+Columns.getString(7));
            model.addRow(Data);
        }
        Stmnt.close();
        CONN.close();
    }catch (Exception ex) {
        JOptionPane.showMessageDialog(null, ex);
    }

    JCB_PilotoR1.setSelectedIndex(-1);
    TF_HoraSalIR1.setText("00:00");
}//GEN-LAST:event_Rep1_consultaActionPerformed
```

5. **TF_HoraSalIR1KeyTyped(java.awt.event.KeyEvent evt)**

    - Este método es un escuchador de eventos de teclado para un JTextField (**TF_HoraSalIR1**). Limita la entrada de caracteres para que solo se ingresen dígitos y dos puntos, asegura un formato de hora válido (HH:MM) y permite solo una ocurrencia de los dos puntos.

```
private void TF_HoraSalIR1KeyTyped(java.awt.event.KeyEvent evt) {//GEN-
FIRST:event_TF_HoraSalIR1KeyTyped
    char c = evt.getKeyChar();
    String horaActual = TF_HoraSalIR1.getText();

    // Asegura que solo se ingresen dígitos y dos puntos
    if (!Character.isDigit(c) && c != ':' || horaActual.length() >= 5) {
        evt.consume();
    }

    // Verifica el formato HH:MM
    if (Character.isDigit(c) && horaActual.length() == 2) {
        TF_HoraSalIR1.setText(horaActual + ":");
    }

    // Asegura que solo haya un punto y se encuentre en la posición correcta
    if (c == ':' && (horaActual.contains(":") || horaActual.length() >= 5))
{
        evt.consume();
    }
}//GEN-LAST:event_TF_HoraSalIR1KeyTyped
```

## Clase restrictChar

La clase **restrictChar** implementa la interfaz **KeyListener** en Java y proporciona una restricción en la entrada de caracteres en los componentes de la interfaz gráfica que utilizan esta clase como escucha de eventos de teclado. En este caso, se están restringiendo los caracteres **-**, **(**, **)**, y **.**.

Aquí hay una explicación más detallada:

**Métodos:**

1. **keyTyped(KeyEvent e)**
    - Este método se llama cuando se ha detectado que se ha presionado y luego liberado una tecla.
    - **e.getKeyChar()**: Devuelve el carácter asociado con la tecla que se presionó.
    - Se consume el evento (**e.consume()**) si el carácter es **-**, **(**, **)**, o **.**. Consumir el evento significa que el componente no procesará el carácter y no se mostrará en el campo de texto u otro componente.
2. **keyPressed(KeyEvent e) y keyReleased(KeyEvent e)**
    - Estos métodos están presentes porque la clase implementa la interfaz **KeyListener**. Sin embargo, en este caso, no se ha implementado ninguna lógica específica para estos métodos.

```java
package Aeropuerto;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;

/**
 * Clase que implementa la interfaz KeyListener para restringir ciertos caracteres
 * en un componente de la interfaz gráfica de usuario (GUI).
 */
public class restrictChar implements KeyListener {

    /**
     * Este método se llama cuando se ha detectado una tecla presionada en el componente.
     * Se consume el evento si el carácter es '-', '(', ')' o '.' para evitar que se ingrese.
     *
     * @param e Evento de tecla que contiene información sobre la tecla presionada.
     */
    @Override
    public void keyTyped(KeyEvent e) {
        char c = e.getKeyChar();
        if (c == '-' || c == '(' || c == ')' || c == '.') {
            e.consume();
        }
    }

    /**
     * Este método se llama cuando una tecla ha sido presionada y aún no ha sido liberada.
     * No se necesita implementación en este contexto.
     *
     * @param e Evento de tecla que contiene información sobre la tecla presionada.
     */
    @Override
    public void keyPressed(KeyEvent e) {
        // No necesitamos implementar este método, pero debe estar presente
    }

    /**
     * Este método se llama cuando una tecla ha sido liberada.
     * No se necesita implementación en este contexto.
```

```
     *
     * @param e Evento de tecla que contiene información sobre la tecla
liberada.
     */
    @Override
    public void keyReleased(KeyEvent e) {
        // No necesitamos implementar este método, pero debe estar presente
    }
}
```

## Clase Login

Esta clase Java representa una ventana de inicio de sesión (**Login**) en una aplicación de aeropuerto. Aquí hay una explicación de la clase y sus métodos:

**Atributos:**

- **USER**: Almacena el nombre de usuario después de la autenticación.
- **PASS**: Almacena la contraseña después de la autenticación.

**Métodos:**

1. **Constructor:**
    - **public Login(java.awt.Frame parent, boolean modal)**: El constructor de la clase, que inicializa la ventana de inicio de sesión.

```
public Login(java.awt.Frame parent, boolean modal) {
    super(parent, modal);
    initComponents();
}
```

2. **Inicialización de la Interfaz Gráfica:**
    - **private void initComponents()**: Método generado automáticamente que inicializa los componentes de la interfaz gráfica, como botones, campos de texto, etc.
3. **Método de Inicio de Sesión:**
    - **private void btn_IniciarSesionActionPerformed(java.awt.event.ActionEvent evt)**: Este método se activa cuando el botón de inicio de sesión es presionado. Realiza lo siguiente:
        - Obtiene el nombre de usuario y la contraseña ingresados.
        - Compara las credenciales ingresadas con credenciales predefinidas.

- Muestra mensajes de bienvenida o error según las credenciales ingresadas.
- Almacena las credenciales si son correctas y cierra la ventana de inicio de sesión.

```
private void btn_IniciarSesionActionPerformed(java.awt.event.ActionEvent
evt) {//GEN-FIRST:event_btn_IniciarSesionActionPerformed
    // Obtiene el usuario y la contraseña ingresados en los campos de texto
    String usuario = tf_User.getText();
    String contrasena = new String(pf_Pass.getPassword());

    // Lógica de validación de credenciales
    if (usuario.equals("postgres") && contrasena.equals("postgres")) {
        JOptionPane.showMessageDialog(this, "¡Bienvenido postgres!");
        USER = usuario;
        PASS = contrasena;
        this.dispose();
    } else if (usuario.equals("administrador") &&
contrasena.equals("232323ADM")) {
        JOptionPane.showMessageDialog(this, "¡Bienvenido administrador!");
        USER = usuario;
        PASS = contrasena;
        this.dispose();
    } else if (usuario.equals("aerolinea_staff") &&
contrasena.equals("235491AERS")) {
        JOptionPane.showMessageDialog(this, "¡Bienvenido aerolinea_staff!");
        USER = usuario;
        PASS = contrasena;
        this.dispose();
    } else if (usuario.equals("pasajero_service") &&
contrasena.equals("094312PASER")) {
        JOptionPane.showMessageDialog(this, "¡Bienvenido
pasajero_service!");
        USER = usuario;
        PASS = contrasena;
        this.dispose();
    } else {
        JOptionPane.showMessageDialog(this, "Usuario no existente");
        USER = "";
        PASS = "";
        this.dispose();
    }
}//GEN-LAST:event_btn_IniciarSesionActionPerformed
```

4. **Métodos de Obtención de Credenciales:**
    - **String getUsr()**: Devuelve el nombre de usuario.
    - **String getPass()**: Devuelve la contraseña.

```
String getUsr() {
    return USER;
}
String getPass() {
    return PASS;
}
```

5. **Método Principal:**
    - **public static void main(String args[])**: El método principal que crea una instancia de la clase **Login** y la hace visible.

**Uso Principal:**

En el método **btn_IniciarSesionActionPerformed**, se verifica el nombre de usuario y la contraseña ingresados con credenciales predefinidas para diferentes roles (postgreSQL, administrador, personal de aerolínea, servicio de pasajeros). Si las credenciales son correctas, se muestra un mensaje de bienvenida y se almacenan en las variables **USER** y **PASS**.