UT2.Manejo de Conectores.
Alberto Colmenar Casas

2. Práctica 2.1 Conexión a MySQL desde Java: nos conectamos a una BD de MySQL (creada previamente) con el driver de JDBC y vemos su contenido por medio del IDE.
3. Práctica 2.2 Panel de control de usuario: login que comprueba si ese usuario y contraseña están en nuestra base de datos.
4. Práctica 2.2 Actualizar contraseña de usuario: update de la contraseña de la BD dado un usuario y contraseña, usando PreparedStatement para poder crear la query parametrizada.
5. Práctica 2.2 Validar contraseña: validamos una contraseña con los distintos parámetros descritos en la práctica usando principalmente el método matches() y expresiones regulares.
6. Práctica 2.3 Consulta y descripción de datos. Base de datos: la clase DatabaseMetaData inicializada con getMetaData() nos aporta los metadatos de la BD conectada anteriormente.
7. Práctica 2.3 Consulta y descripción de datos. Tablas: el usuario elige que tabla quiere ver con un JOptionPane y con DatabaseMetaData obtenemos la información de las tablas.
8. Práctica 2.3 Consulta y descripción de datos. Columnas: con el DatabaseMetaData pedimos esta vez la información de las columnas de una tabla y con el resulset las mostramos.
9. Práctica 2.3 Listado de vuelos: el método devuelve una DefaultTableMode que la construimos gracias al resultSet con la información de la tabla obtenida ejecutando una query con Statement.
10. Práctica 2.3 Listado de pasajeros de un vuelo: obtenemos los códigos de los vuelos con una sentencia a la BD para añadirlos al ComboBox al crear la ventana y que el usuario pueda seleccionar cualquiera.
11. Práctica 2.4 Conexión a Oracle desde Java: por consola escogemos a qué BD quieres conectarte. En este caso en la de Oracle cargamos su driver y nos conectamos a la BD que fue creada anteriormente.

Práctica 2.1 Conexión a MySQL desde Java

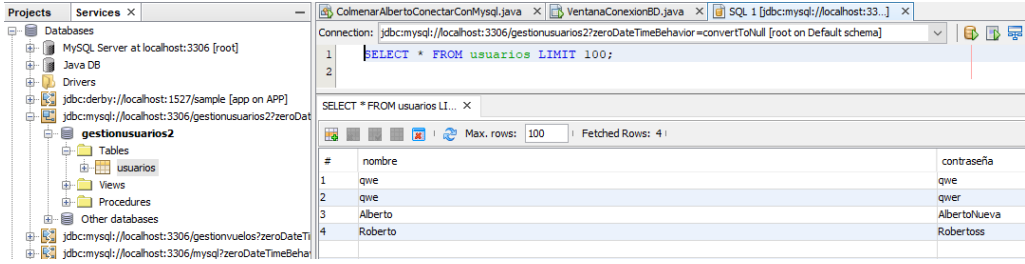
ColmenarAlbertoConectarConMysql.java

```
*/
public class ColmenarAlbertoConectarConMysql {


    public static Connection conexion;

    public void getConnection() {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            conexion = (Connection) DriverManager.getConnection("jdbc:mysql://localhost/gestionusuarios2", "root", "");
            System.out.println("Conexion realizada");
        } catch (ClassNotFoundException ex) {
            System.err.println("Error " + ex.getMessage());
        } catch (SQLException ex) {
            System.err.println("Error de conexión " + ex.getMessage());
        }
    }
    public static void main(String[] args) {
       ColmenarAlbertoConectarConMysql prueba = new ColmenarAlbertoConectarConMysql();
       prueba.getConnection();
    }
```

Usamos el driver JDBC y la clase DriverManager para inicializar un objeto de la clase Connection. Que es el que usaremos para comunicarnos con la BD, este caso de MySQL.

```
Output - ColmenarAlbertoConectarConMysql (run)   Notifications
run:
Conexion realizada
BUILD SUCCESSFUL (total time: 0 seconds)
```

Se ve la base de datos de gestionusuarios2 que tiene una tabla usuarios con un nombre y una contraseña, la utilizaremos en la siguiente práctica.



Práctica 2.2 Panel de control de usuario

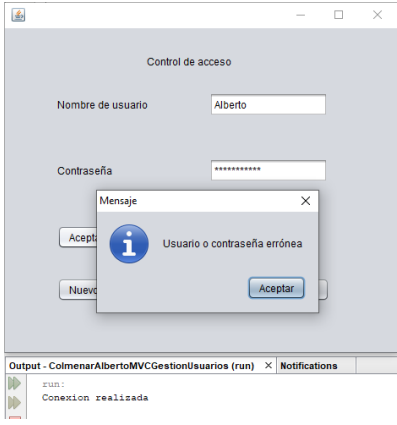
```java
public boolean buscarUsuario(String nombre, String contrasena) {
    boolean encontrado = false;
    if (!(nombre.isEmpty()) || contrasena.isEmpty()) {
        try {
            Statement stmt = (Statement) conexion.createStatement();
            ResultSet resultado = stmt.executeQuery("SELECT * FROM usuarios WHERE nombre='" +
                    nombre + "' and contraseña='" + contrasena + "'");
            if (resultado.next()) {
                encontrado = true;
            }
            resultado.close();
            stmt.close();
        } catch (SQLException ex) {
            System.err.println(ex.getMessage());
        }
    }
    return encontrado;
}
```

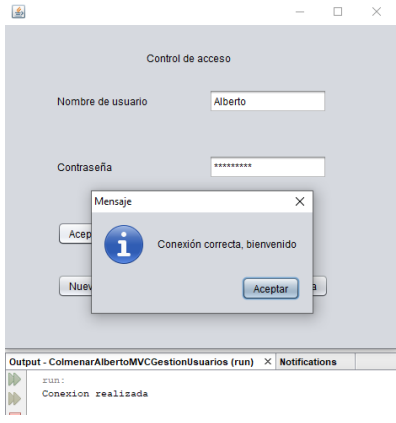
```java
private void botonAceptarActionPerformed(java.awt.event.ActionEvent evt) {
    if (camposRellenados()) {
        String nombre = cajaUsuario.getText();
        String contrasena = String.valueOf(cajaContrasena.getPassword());
        if (mvcGestionUsuarios.buscarUsuario(nombre, contrasena)) {
            JOptionPane.showMessageDialog(this, "Conexión correcta, bienvenido");
        } else {
            JOptionPane.showMessageDialog(this, "Usuario o contraseña errónea");
        }
    } else {
        JOptionPane.showMessageDialog(this, "Rellena los campos");
    }
```

Comprueba si existe el usuario que se quiere logear ejecutando una query.
Usamos la clase Statement para escribir y lanzar la query a la BD previamente conectada.
Capturamos el resultado en un ResultSet y un boolean.

Verifica previamente que los campos nombre y contraseña están rellenados antes de llamar al método anterior.
Al final se muestra un mensaje de bienvenida o erroneo dependiendo de la respuesta de la query del método buscarUsuario().

Los distintos mensajes que puede dar.





Práctica 2.2 Actualizar contraseña de usuario

Flujo normal de la app para cambiar la contraseña del usuario Alberto, previamente tenía Albertoss y después de AlbertoNueva, ya que es una contraseña válida.

```java
public boolean modificarContrasena(String nombre, String contrasena) {
    boolean modificado = false;
    try {
        PreparedStatement sql = conexion.prepareStatement("UPDATE usuarios SET contraseña = ? WHERE nombre = ?");
        sql.setString(1, contrasena);
        sql.setString(2, nombre);
        sql.executeUpdate();
        modificado = true;
        sql.close();
    } catch (SQLException ex) {
        System.err.println(ex.getMessage());
    }
    return modificado;
}

public String contrasenaValida(String contrasena) {
    String errores = "";
    if (!contrasena.matches("\\w*")) {
        errores += "☹ Debe estar formada sólo por letras, números y el carácter \"_\".\n";
    }
    if (contrasena.length() < 8) {
        errores += "☹ Debe tener 8 caracteres como mínimo.\n";
    }
    if (contrasena.matches("^\\d.*")) {
        errores += "☹ No debe comenzar con un carácter numérico.\n";
    }
    if (!contrasena.matches(".*[A-Z]+.*")) {
        errores += "☹ Debe contener al menos una letra en mayúscula.\n";
    }
    return errores;
}
```

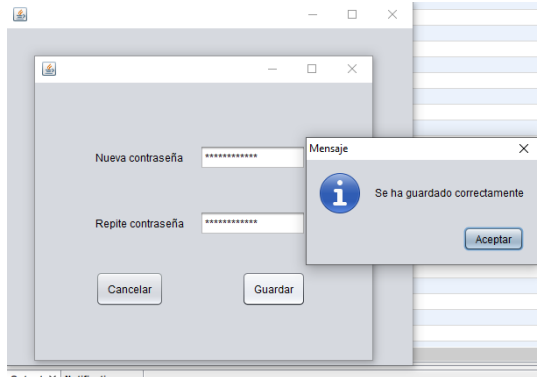
Con una instancia de la clase PreparedStatement construimos el update parametrizado. Añadimos los parámetros y ejecutamos la query con executeQuery(). Devolvemos un boolean para mostrar un mensaje después.

```java
private void botonGuardarActionPerformed(java.awt.event.ActionEvent evt) {
    String contrasenal = String.valueOf(cajaContrasenal.getPassword());
    String contrasenaValida = mvcGestionUsuarios.contrasenaValida(contrasenal
    boolean contrasenaComprobada = true;
    if (!contrasenaValida.equals("")) {
        JOptionPane.showMessageDialog(this, contrasenaValida);
        cajaContrasenal.setText("");
        cajaContrasena2.setText("");
        contrasenaComprobada = false;
        return;
    }
    String contrasena2 = String.valueOf(cajaContrasena2.getPassword());
    if (!contrasenal.equals(contrasena2)) {
        JOptionPane.showMessageDialog(this, "Las dos contraseñas no son iguales");
        cajaContrasena2.setText("");
        contrasenaComprobada = false;
    }
    if (contrasenaComprobada) {
        boolean guardado = mvcGestionUsuarios.modificarContrasena(nombre, contrasenal);
        if (guardado) {
            JOptionPane.showMessageDialog(this, "Se ha guardado correctamente");
            this.dispose();
        } else {
            JOptionPane.showMessageDialog(this, "Error al guardar la contraseña");
        }
    }
}
```

La contraseña nueva se valida (siguiente diapositiva) y se comprueba que se escribe las dos veces iguales.
Después se ejecuta el método anterior y se muestra el mensaje afirmativo o negativo.

```
SELECT * FROM usuarios LIMIT 100;
```

#	nombre	contraseña
1	qwe	qwe
2	qwe	qwer
3	Alberto	Albertoss
4	Roberto	Robertoss




```
ColmenarAlbertoMVCGestionUsuarios (run)
run:
Conexion realizada
```

```
SELECT * FROM usuarios LIMIT 100;
```

#	nombre	contraseña
1	qwe	qwe
2	qwe	qwer
3	Alberto	AlbertoNueva
4	Roberto	Robertoss

Práctica 2.2 Validar contraseña

Este método de validar se usa tanto en la siguiente imagen que se ejecuta al darle a la tecla Intro cuando escribes tu contraseña al logearte como al cambiar la contraseña (diapositiva anterior).

ColmenarAlbertoMVCGestionUsuarios.java | VentanaVerificar.java

Source | History

```
82      public String contrasenaValida(String contrasena) {
83          String errores = "";
84          if (!contrasena.matches("\\w*")) {
85              errores += "Debe estar formada sólo por letras, números y el carácter \"_\".\n";
86          }
87          if (contrasena.length() < 8) {
88              errores += "Debe tener 8 caracteres como mínimo.\n";
89          }
90          if (contrasena.matches("^\\d.*")) {
91              errores += "No debe comenzar con un carácter numérico.\n";
92          }
93          if (!contrasena.matches(".*[A-Z]+.*")) {
94              errores += "Debe contener al menos una letra en mayúscula.\n";
95          }
96          return errores;
97      }
98
99  }
```

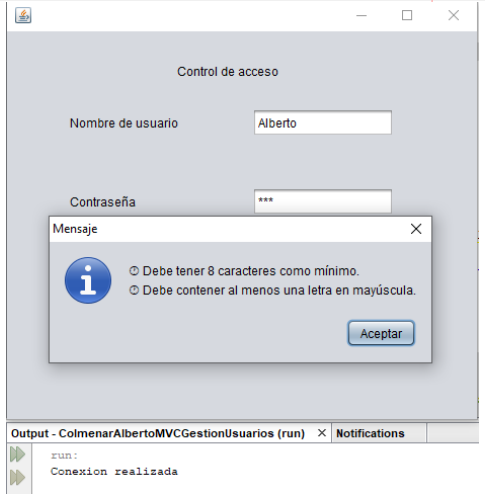
ColmenarAlbertoMVCGestionUsuarios.java | VentanaVerificar.java

Source | Design | History

```
209     private void cajaContrasenaActionPerformed(java.awt.event.ActionEvent evt) {
210         if (camposRellenados()) {
211             String contrasena = String.valueOf(cajaContrasena.getPassword());
212             String contrasenaValida = mvcGestionUsuarios.contrasenaValida(contrasena);
213             if (contrasenaValida.equals("")) {
214                 habilitarBotones();
215             } else {
216                 JOptionPane.showMessageDialog(this, contrasenaValida);
217             }
218
219         }
220     }
```

Método que comprueba que una contraseña es válida usando matches() y expresiones regulares.
Si es válida devuelve un String vacío, sino se van concatenando los diferentes errores en la contraseña para luego mostrarlos en un mensaje para el usuario.

Posible ejecución donde vemos que nuestra contraseña no tiene 8 caracteres ni ninguna mayúscula.



Práctica 2.3 Consulta y descripción de datos. Base de datos

ColmenarAlbertoMVCVuelos.java | ColmenarAlbertoVentanaDescripcion.java

Source | History

```
59      public String informaBD() {
60          String datos = "";
61          try {
62              DatabaseMetaData dbmd = conexion.getMetaData();
63              datos = String.format("NOMBRE: %s%n"
64                      + "DRIVER: %s%n"
65                      + "URL: %s%n"
66                      + "USUARIO: %s%n",
67                      dbmd.getDatabaseProductName(), dbmd.getDriverName(), dbmd.getURL(), dbmd.getUserName());
68              ResultSet tablas = dbmd.getTables(nombreBD, nombreBD, null, null);
69              while (tablas.next()) {
70                  datos += "TABLA: " + tablas.getString("TABLE_NAME") + "\n";
71              }
72          } catch (SQLException ex) {
73              System.err.println("Error " + ex.getMessage());
74          }
75          return datos;
76      }
```

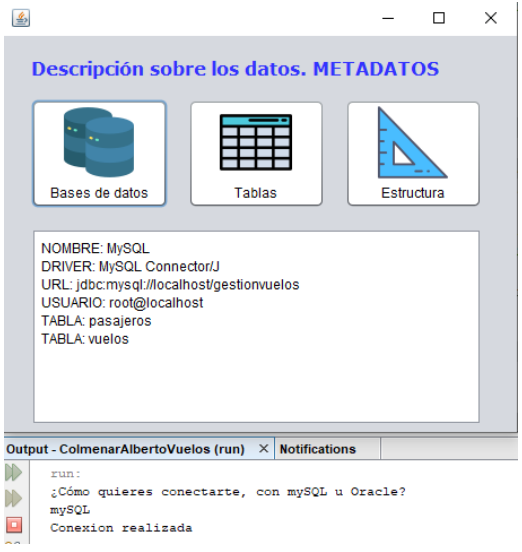
Con el método getMetaData() de Connection inicializamos una instancia de DatabaseMetaData que contiene la información de la BD, previamente conectada, como su nombre, driver, tablas. Retorna un String con toda la información pedida.

ColmenarAlbertoMVCVuelos.java | ColmenarAlbertoVentanaDescripcion.java

Source | Design | History

```
117     private void botonBDActionPerformed(java.awt.event.ActionEvent evt) {
118         String datos = MVCVuelos.informaBD();
119         areaDatos.setText(datos);
120     }
```

La información pedida se vuelca directamente en el TextArea de debajo de los botones.



Práctica 2.3 Consulta y descripción de datos. Tablas

ColmenarAlbertoMVCVuelos.java

Source | History

```
78  public String informaTabla(String nombreTabla) {
79      String datos = "";
80      try {
81          DatabaseMetaData dbmd = conexion.getMetaData();
82          ResultSet tabla = dbmd.getTables(nombreBD, nombreBD, nombreTabla, null);
83          while (tabla.next()) {
84
85              datos += String.format("CATALOGO: %s%n"
86                      + "ESQUEMA: %s%n"
87                      + "NOMBRE TABLA: %s%n"
88                      + "TIPO: %s%n",
89                      tabla.getString("TABLE_CAT"),
90                      tabla.getString("TABLE_SCHEM"),
91                      tabla.getString("TABLE_NAME"),
92                      tabla.getString("TABLE_TYPE"));
93          }
94      } catch (SQLException ex) {
95          System.err.println("Error " + ex.getMessage());
96      }
97      return datos;
98  }
```

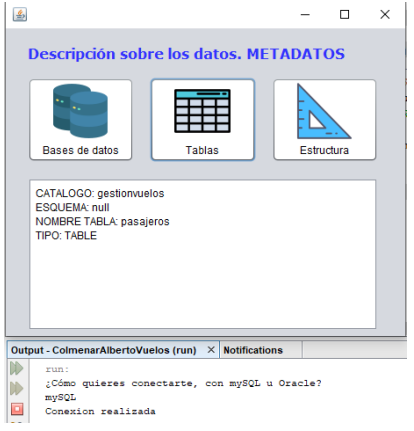
Con una instancia de la clase DatabaseMetaData y el método getMetaData de Connection obtenemos la información de la tabla pasada por parámetro.
Gracias al resultset podemos mostrar los distintos datos que pedía el enunciado.

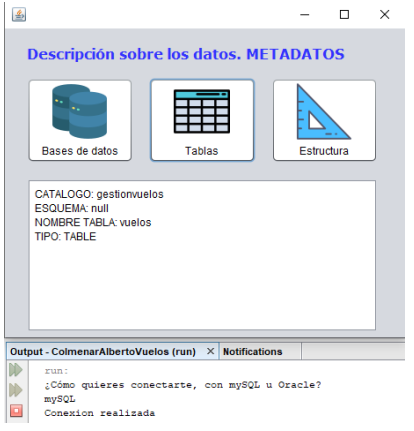
ColmenarAlbertoMVCVuelos.java | ColmenarAlbertoVentanaDescripcion.java

Source | Design | History

```
122     private void botonTablaActionPerformed(java.awt.event.ActionEvent evt) {
123         // muestra un Dialog con los nombres de las tablas para que el usuario escoja
124         String[] opciones = {"PASAJEROS", "VUELOS"};
125         int seleccionado = JOptionPane.showOptionDialog(this, "¿Qué tabla quieres ver?", "Información tabla",
126                 JOptionPane.DEFAULT_OPTION, INFORMATION_MESSAGE, null, opciones, opciones[0]);
127         if (seleccionado > -1) {
128             String datos = MVCVuelos.informaTabla(opciones[seleccionado]);
129             areaDatos.setText(datos);
130         }
```

Creamos un JOptionPane con las opciones de los nombres de las tablas pasajeros y vuelos para que el usuario pueda elegir cual ver, llamar al método anterior y volcar su retorno en el TextArea.





Práctica 2.3 Consulta y descripción de datos. Columnas

ColmenarAlbertoMVCVuelos.java | ColmenarAlbertoVentanaDescripcion.java

Source | History

```
101     public String informaColumnas(String nombreTabla) {
102         String datos = "";
103         try {
104             DatabaseMetaData dbmd = conexion.getMetaData();
105             ResultSet columnas = dbmd.getColumns(null, nombreBD, nombreTabla, null);
106 
107             while (columnas.next()) {
108                 datos += String.format("NOMBRE: %s%n"
109                         + "TIPO: %s%n"
110                         + "TAMAÑO: %s%n"
111                         + "ES NULA: %s%n%n",
112                         columnas.getString("COLUMN_NAME"),
113                         columnas.getString("TYPE_NAME"),
114                         columnas.getString("COLUMN_SIZE"),
115                         columnas.getString("IS_NULLABLE"));
116             }
117         } catch (SQLException ex) {
118             System.err.println("Error " + ex.getMessage());
119         }
120         return datos;
121     }
```

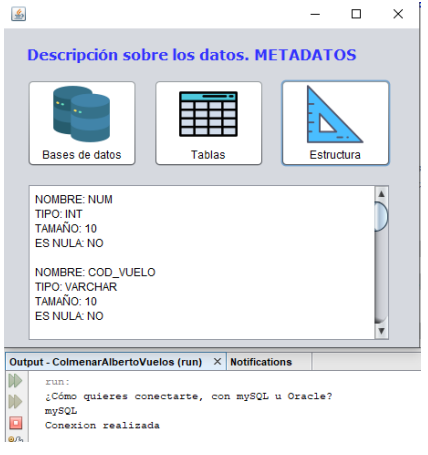
Obtenemos el DatabaseMetaData como anteriormente, esta vez en el resulset nos quedamos con las columnas de una tabla y concatenamos la información pedida.

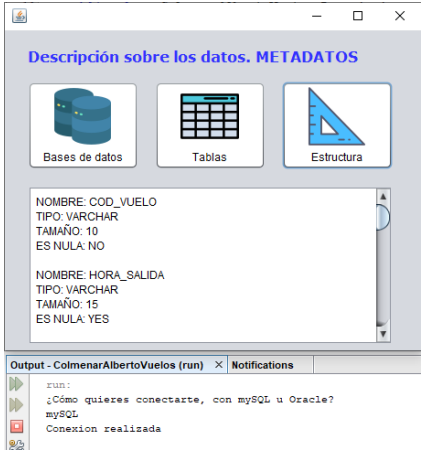
ColmenarAlbertoMVCVuelos.java | ColmenarAlbertoVentanaDescripcion.java

Source | Design | History

```
133     private void botonEstructuraActionPerformed(java.awt.event.ActionEvent evt) {
134         // muestra un Dialog con los nombres de las tablas para que el usuario escoja
135         String[] opciones = {"PASAJEROS", "VUELOS"};
136         int seleccionado = JOptionPane.showOptionDialog(this, "¿Qué tabla quieres ver?", "Información tabla",
137                 JOptionPane.DEFAULT_OPTION, INFORMATION_MESSAGE, null, opciones, opciones[0]);
138         if (seleccionado > -1) {
139             String datos = MVCVuelos.informaColumnas(opciones[seleccionado]);
140             areaDatos.setText(datos);
141         }
142     }
```

Misma estructura para pedir al usuario que elija la tabla y una vez ejecutado el método se vuelca la información de en TextArea.





Práctica 2.3 Listado de vuelos

ColmenarAlbertoMVCVuelos.java | ColmenarAlbertoVentanaVuelos.java

Source | History

```
123     public DefaultTableModel generarTablaVuelos() {
124         DefaultTableModel tabla = new DefaultTableModel();
125         try {
126             Statement stmt = conexion.createStatement();
127             ResultSet consulta = stmt.executeQuery("SELECT cod_vuelo,"
128                     + " procedencia, destino FROM vuelos");
129             tabla.addColumn("COD_VUELO");
130             tabla.addColumn("PROCEDENCIA");
131             tabla.addColumn("DESTINO");
132             Object datos[] = new Object[3];
133
134             while (consulta.next()) {
135                 for (int i = 0; i < 3; i++) {
136                     datos[i] = consulta.getObject(i + 1);
137                 }
138                 tabla.addRow(datos);
139             }
141             if (stmt != null) {
142                 stmt.close();
143             }
144         } catch (SQLException ex) {
145             System.err.println(ex.getMessage());
146         }
147         return tabla;
    }
```

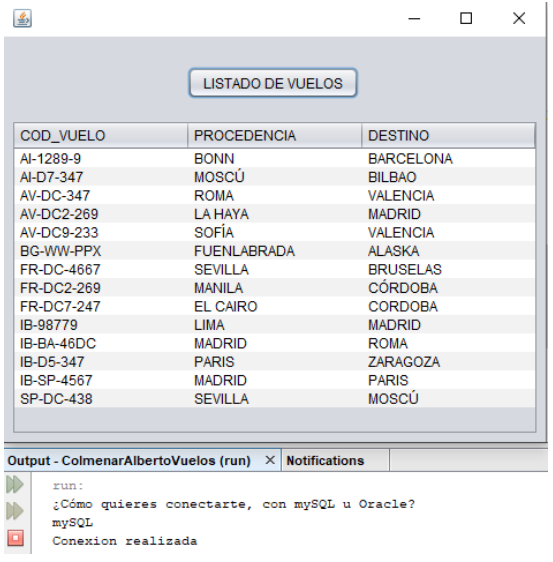
Creamos un Statement con la conexión previa y ejecutamos una query para obtener todos los vuelos.
Como queremos mostrarlo en formato tabla nos hacemos una DefaultTableModel con las columnas que queremos y recorremos el resultSet añadiendole la información.

ColmenarAlbertoMVCVuelos.java | ColmenarAlbertoVentanaVuelos.java

Source | Design | History

```
98
99      private void botonListadoActionPerformed(java.awt.event.ActionEvent evt) {
100         DefaultTableModel tabla = MVCVuelos.generarTablaVuelos();
101         tablaVuelos.setModel(tabla);
102     }
```

Como el método nos devuelve ya la información en formato tabla la volcamos a la interfaz que es una tabla de Swing.



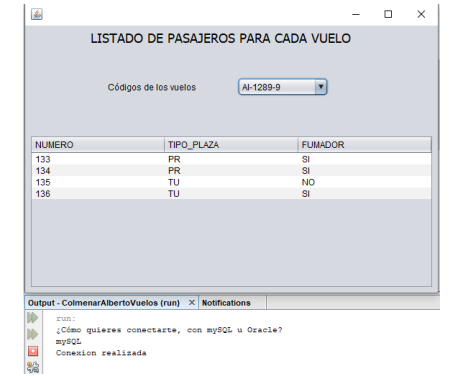
Práctica 2.3 Listado de pasajeros de un vuelo

ColmenarAlbertoMVCVuelos.java | ColmenarAlbertoVentanaPasajeros.java

Source | History

```
149     public ArrayList obtenerListaCodigo() {
150         ArrayList<String> lista = new ArrayList();
151         String cod;
152         try {
153             Statement stmt = conexion.createStatement();
154             ResultSet resultado = stmt.executeQuery("SELECT COD_VUELO FROM vuelos");
155             while (resultado.next()) {
156                 cod = resultado.getString("COD_VUELO");
157                 lista.add(cod);
158             }
159         } catch (SQLException ex) {
160             System.err.println(ex.getMessage());
161         }
162         return lista;
163     }
```

Creamos un arrayList para poder añadirlo al ComboBox con los códigos de los vuelos obtenidas con un Statement ejecutando la query y recorriendolo añadiéndolo a la lista.

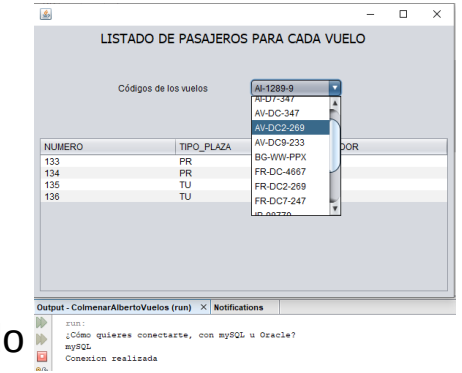


ColmenarAlbertoMVCVuelos.java | ColmenarAlbertoVentanaPasajeros.java

Source | Design | History

```
14      public ColmenarAlbertoVentanaPasajeros() {
15          initComponents();
16          MVCVuelos.getConnection();
17          this.cargarListaCodigos();
18      }
19
20      public void cargarListaCodigos() {
21          ArrayList<String> lista = new ArrayList();
22          lista = MVCVuelos.obtenerListaCodigo();
23          for (String cod : lista) {
24              this.comboCodigos.addItem(cod);
25          }
26      }
27
```

Al crear la ventana hacemos la conexión a la BD y cargamos los códigos al ComboBox.



El código que esté seleccionado mostramos sus pasajeros

ColmenarAlbertoMVCVuelos.java | ColmenarAlbertoVentanaPasajeros.java

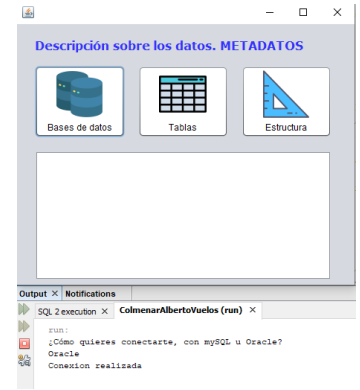
Source | Design | History

```
120     private void comboCodigosActionPerformed(java.awt.event.ActionEvent evt) {
121         DefaultTableModel tabla = MVCVuelos.generarTablaPasajeros(String.valueOf(comboCodigos.getSelectedItem()));
122         tablaPasajeros.setModel(tabla);
123     }
```

Práctica 2.4 Conexión a Oracle desde Java

```java
public void getConnection() {
    Scanner in = new Scanner(System.in);
    String bd = "";
    // pregunta como quiere conectarse
    while ((!bd.equals("mySQL")) && !bd.equals("Oracle")) {
        System.out.println("¿Cómo quieres conectarte, con mySQL u Oracle?");
        bd = in.nextLine();
    }

    if (bd.equals("mySQL")) {
        try {
            nombreBD = "gestionvuelos";
            Class.forName("com.mysql.cj.jdbc.Driver");
            conexion = (Connection) DriverManager.getConnection("jdbc:mysql://localhost/gestionvuelos", "root", "");
            System.out.println("Conexion realizada");
        } catch (ClassNotFoundException ex) {
            System.err.println("Error " + ex.getMessage());
        } catch (SQLException ex) {
            System.err.println("Error de conexión " + ex.getMessage());
        }
    } else if (bd.equals("Oracle")) {
        nombreBD = "SYSTEM";
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            conexion = (Connection) DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE", "system", "Dam-2022");
            System.out.println("Conexion realizada");
        } catch (ClassNotFoundException ex) {
            System.err.println("Error " + ex.getMessage());
        } catch (SQLException ex) {
            System.err.println("Error de conexión " + ex.getMessage());
        }
```

Se muestran la conexión de Oracle (debajo)
El usuario escribe por consola ‘Oracle’ para conectarse a ella.
Se carga el driver de Oracle y se conecta a la BD previamente creada.



Se muestra el contenido de las dos tablas y de la conexión de Oracle.