

CONTENIDOS

Impreso el marzo 28, 2021

Aplicación Gestión de clientes (Guiada)

Vamos a realizar una aplicación para gestionar el alta, baja y actualización de una lista de clientes utilizando el lenguaje Java para su realización y con persistencia de datos mediante MySQL.

Para realizar la aplicación se ha utilizado el IDE Eclipse con el pluging de WindowBulder para la realización de los elementos de la interfaz gráfica.

Se ha utilizado el patrón de MVC para la estructura de la clase y los patrones de diseño DAO y el modelo Singleton para la capa de persistencia.

Id	DNI	Nombre	Apellidos
1	sdsasdas	dasdasd	
2	sdsasdas	dasdasd	
3	asdasdasd	asdasd	
6	58554554J	Antonio	Otero Veiga
11	343434	María	Alonso
12	25252552g	Anotnio	Otero Veiga
17	ertertet	ertert	
18	sdsasdas	dasdasd	
19	wer	wer	
20	asasasas	asasasas	
21	mmmmmm	mmmmm	
22	jjjjj	jjjjj	
23	asdasdasd	asdasdasdas	
24	ñññññ	ñññññ	

La aplicación consta de las siguientes clases:

Inicio: Clase controladora con el main que contiene el código para lanzar la aplicación.

Altas: Clase del paquete Vista, con los elementos de la interfaz.

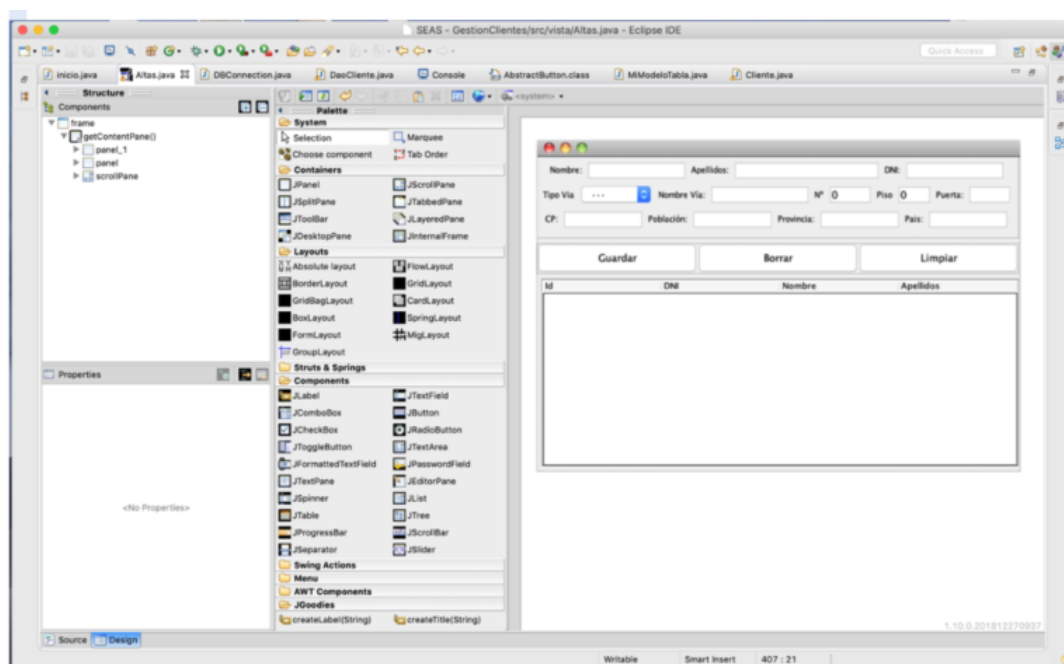
Clientes: Clase del modelo de datos, donde se define la estructura del cliente.

DaoCliente: Clase con los métodos de persistencia para la gestión de la información en la BD. Siguiendo el modelo Singleton.

Descripción de las clases y métodos más importantes.

Vista

Para realizar la interfaz se ha utilizado el plugin para Eclipse windowBulder, con el cual se ha ido incorporando los paneles y elementos para interactuar con el usuario.



La interfaz esta compuesta principalmente de elementos Text Field además de un combobox y una tabla.

TextField:

La inserción de los Text Fields se ha realizado con el plugin, donde se ha cambiado el nombre (variable) del elemento y se ha puesto un label para identificarlo.

```
JLabel lblNombre = new JLabel("Nombre:");
lblNombre.setFont(new Font("Lucida Grande", Font.PLAIN, 10));
panel.add(lblNombre);

textField_nombre = new JTextField();
panel.add(textField_nombre);
textField_nombre.setColumns(10);
```

Combo

La realización del combo se ha realizado insertando el elemento e incorporando los elementos en el Array del código.

```
comboTipovia = new JComboBox();
comboTipovia.setFont(new Font("Lucida Console", Font.PLAIN, 10));
comboTipovia.setModel(new DefaultComboBoxModel(new String[] { "...", "Calle", "Plaza", "Avenida" }));
panel.add(comboTipovia);
```

Tabla

En el caso de la tabla se ha realizado un método para poder reutilizar el código cada vez que se modifican los datos al realizar una actualización, inserción o borrado de un elemento.

```
private void pintarTabla() throws SQLException {
    table.setModel(new DefaultTableModel(
        new Object[][] {
        },
        new String[] {
            "Id", "DNI", "Nombre", "Apellidos"
        }
    ));

    ArrayList<Cliente> lista = new ArrayList<Cliente>();
    lista = (ArrayList<Cliente>) DaoCliente.getInstance().obtenerLista();

    for(Cliente c : lista) {
        Object[] fila = new Object[4];

        fila[0] = c.getId();
        fila[1] = c.getDni();
        fila[2] = c.getNombre();
        fila[3] = c.getApellidos();

        ((DefaultTableModel) table.getModel()).addRow(fila);
    }
}
```

Para las acciones nos encontramos con tres botones:

Guardar

```
// Boton guardar cliente
JButton btnGuardar = new JButton("Guardar");
btnGuardar.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

        guardado();

    }
});
panel_1.add(btnGuardar);
```

Como podemos observar en el código anterior, el evento del botón hace una llamada al método de guardado en la misma clase.

```
private void guardado() {
    //System.out.println("Boton Pulsado");
    //System.out.println(textField_dni.getText());

    if(textField_dni.getText().length() == 0 || textField_nombre.getText().length() == 0) {
        JOptionPane.showMessageDialog(null, "No se puede insertar sin DNI y Nombre");
        System.out.println(comboTipovia.getSelectedIndex());
    }else{
        cliente.setDni(textField_dni.getText());
        cliente.setNombre(textField_nombre.getText());
        cliente.setApellidos(textField_apellidos.getText());
        cliente.setNombreVia(textField_nombreVia.getText());
        cliente.setNumero(Integer.parseInt(textField_numero.getText()));
        cliente.setPiso(Integer.parseInt(textField_piso.getText()));
        cliente.setPuerta(textField_puerta.getText());
        cliente.setCp(textField_cp.getText());
        cliente.setProvincia(textField_provincia.getText());
        cliente.setPoblacion(textField_poblacion.getText());
        cliente.setPais(textField_pais.getText());
        cliente.setTipoVia(comboTipovia.getSelectedIndex());

        try {
            //JOptionPane.showMessageDialog(null, cliente.getId());

            if(cliente.getId() > 0) {
                cliente.update();
            }else {
                cliente.insertar();
            }

            pintarTabla();
            limpiarCampos();
        } catch (SQLException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
    }
}
```

En la función vemos como realiza tanto el guardado como la actualización según sea el caso. Para ello diferencia la acción mediante la comprobación del ID. En el caso de la actualización primero se selecciona un elemento de la lista mostrada en la tabla, con la cual se obtiene el Id del elemento seleccionado. Esto se produce al hacer click en un elemento de la lista mostrada y capturando el evento.

```
table.setSurrendersFocusOnKeystroke(true);
table.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {

        llenarCampos();

    }
});
```

Que realiza la llamada al método de llenar campos, con el cual partiendo del Id del objeto seleccionado obtiene el dato almacenado en la fila seleccionada y rellena con los datos obtenidos de la base de datos cada campo del formulario.

```
public void llenarCampos() {
    DefaultTableModel tm = (DefaultTableModel) table.getModel();
    String dato=String.valueOf(tm.getValueAt(table.getSelectedRow(),0));
    //JOptionPane.showMessageDialog(null, dato);

    try {
        cliente = DaoCliente.getInstance().obtenerPorId(Integer.parseInt(dato));

        textField_dni.setText(cliente.getDni());
        textField_nombre.setText(cliente.getNombre());
        textField_apellidos.setText(cliente.getApellidos());
        textField_nombreVia.setText(cliente.getNombreVia());
        textField_numero.setText(String.valueOf(cliente.getNumero()));
        textField_piso.setText(String.valueOf(cliente.getPiso()));
        textField_puerta.setText(cliente.getPuerta());
        textField_cp.setText(cliente.getCp());
        textField_provincia.setText(cliente.getProvincia());
        textField_poblacion.setText(cliente.getPoblacion());
        textField_pais.setText(cliente.getPais());
        comboTipovia.setSelectedIndex(cliente.getTipoVia());

    } catch (NumberFormatException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    } catch (SQLException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
}
```

Para el botón de borrado llamamos al método delete realizado en la clase Cliente para borrar el registro enviando el id del elemento.

```
public void borrado() {
    DefaultTableModel tm = (DefaultTableModel) table.getModel();
    String dato=String.valueOf(tm.getValueAt(table.getSelectedRow(),0));

    if(cliente.getId() > 0 ) {
        try {
            cliente.delete(Integer.parseInt(dato));
            pintarTabla();
            limpiarCampos();

        } catch (NumberFormatException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        } catch (SQLException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
    }
}
```

Cada vez que se borra, actualiza o se inserta un cliente, debemos vaciar los datos del formulario, por lo que tenemos el método limpiar campos que es ejecutado cada vez que se realiza alguna de estas acciones.


```
private void limpiarCampos() {
    textField_dni.setText(null);
    textField_nombre.setText(null);
    textField_apellidos.setText(null);
    textField_nombreVia.setText(null);
    textField_numero.setText("0");
    textField_piso.setText("0");
    textField_puerta.setText(null);
    textField_cp.setText(null);
    textField_provincia.setText(null);
    textField_poblacion.setText(null);
    textField_pais.setText(null);
    comboTipovia.setSelectedIndex(0);
}
```

Modelo

En el modelo de la aplicación intervienen tres elementos, la clase clientes que define la estructura de un cliente, la Clase DaoClientes que contiene los métodos de manipulación en la base de datos y la propia base de datos.

Base de datos:

La base de datos consta de una sola tabla con los valores de los campos de cada atributo de cliente. No se han respetado todas las formas normales de entidad relación para no hacer la practica demasiado grande.

	#	Nombre
<input type="checkbox"/>	1	id 
<input type="checkbox"/>	2	dni
<input type="checkbox"/>	3	nombre
<input type="checkbox"/>	4	apellidos
<input type="checkbox"/>	5	nombreVia
<input type="checkbox"/>	6	tipoVia
<input type="checkbox"/>	7	numero
<input type="checkbox"/>	8	piso
<input type="checkbox"/>	9	puerta
<input type="checkbox"/>	10	cp
<input type="checkbox"/>	11	provincia
<input type="checkbox"/>	12	poblacion
<input type="checkbox"/>	13	pais

La clase Cliente consta de los atributos y los métodos de inserción, actualización, borrado y listado de elementos para realizar el CRUD de los datos.

```
1 package modelo;
2
3 import java.sql.SQLException;
4 import java.util.ArrayList;
5 import java.util.List;
6
7 public class Cliente {
8     private int id;
9     private String dni;
10    private String nombre;
11    private String apellidos;
12    private String nombreVia;
13    private int numero;
14    private int piso;
15    private String puerta;
16    private String cp;
17    private String provincia;
18    private String poblacion;
19    private String pais;
20    private int tipoVia;
21
22
23    /**
24     * Constructor Cliente Vacio
25     */
26    public Cliente() {
27    }
28
29
30    /**
31     * Constructor con parametros
32     *
33     * @param id
34     * @param dni
35     * @param nombre
36     * @param apellidos
37     * @param nombreVia
38     * @param numero
39     * @param piso
40     * @param puerta
41     * @param cp
42     * @param provincia
43     * @param poblacion
44     * @param pais
45     * @param tipoVia
46     */
47
48    public Cliente(int id, String dni, String nombre, String apellidos, String nombreV
49        this.id = id;
50        this.dni = dni;
51        this.nombre = nombre;
52        this.apellidos = apellidos;
53        this.nombreVia = nombreVia;
54        this.numero = numero;
55        this.piso = piso;
56        this.puerta = puerta;
57        this.cp = cp;
58        this.provincia = provincia;
59        this.poblacion = poblacion;
60        this.pais = pais;
61        this.tipoVia = tipoVia;
62    }
63
64    public int getId() {
65        return id;
66    }
67
68    public void setId(int id) {
```

```
69         this.id = id;
70     }
71
72     public String getDni() {
73         return dni;
74     }
75
76     public void setDni(String dni) {
77         this.dni = dni;
78     }
79
80     public String getNombre() {
81         return nombre;
82     }
83
84     public void setNombre(String nombre) {
85         this.nombre = nombre;
86     }
87
88     public String getApellidos() {
89         return apellidos;
90     }
91
92     public void setApellidos(String apellidos) {
93         this.apellidos = apellidos;
94     }
95
96     public String getNombreVia() {
97         return nombreVia;
98     }
99
100    public void setNombreVia(String nombreVia) {
101        this.nombreVia = nombreVia;
102    }
103
104
105    public int getNumero() {
106        return numero;
107    }
108
109    public void setNumero(int numero) {
110        this.numero = numero;
111    }
112
113    public int getPiso() {
114        return piso;
115    }
116
117    public void setPiso(int piso) {
118        this.piso = piso;
119    }
120
121    public String getPuerta() {
122        return puerta;
123    }
124
125    public void setPuerta(String puerta) {
126        this.puerta = puerta;
127    }
128
129    public String getCp() {
130        return cp;
131    }
132
133    public void setCp(String cp) {
134        this.cp = cp;
135    }
136
137    public String getProvincia() {
```



```
138         return provincia;
139     }
140
141     public void setProvincia(String provincia) {
142         this.provincia = provincia;
143     }
144
145     public String getPoblacion() {
146         return poblacion;
147     }
148
149     public void setPoblacion(String poblacion) {
150         this.poblacion = poblacion;
151     }
152
153     public String getPais() {
154         return pais;
155     }
156
157     public void setPais(String pais) {
158         this.pais = pais;
159     }
160
161
162
163     public int getTipoVia() {
164         return tipoVia;
165     }
166
167     public void setTipoVia(int tipoVia) {
168         this.tipoVia = tipoVia;
169     }
170
171
172     /**
173      * Metodo de inserción de cliente. Envía los datos del objeto al metodo de la clas
174      * @throws SQLException
175      */
176
177     public void insertar() throws SQLException {
178
179
180         DaoCliente.getInstance().insert(this);
181
182     }
183
184
185     /**
186      * Metodo de actualización de cliente. Envía los datos del objeto al metodo de la
187      * @throws SQLException
188      */
189     public void update() throws SQLException {
190
191
192         DaoCliente.getInstance().update(this);
193
194     }
195
196
197     /**
198      * Metodo de Borrado de cliente. Envía el id del objeto actual al metodo de la cla
199      * @throws SQLException
200      */
201     public void delete(int id) throws SQLException {
202
203
204         DaoCliente.getInstance().delete(id);
205
206     }
```

```

207
208
209     /**
210      * Metodo para obtener la lista de clientes obtenidos de la base de datos mediante
211      * @return
212      * @throws SQLException
213      */
214
215     public List<Cliente> obtenerLista() throws SQLException{
216
217
218         List<Cliente> lista = DaoCliente.getInstance().obtenerLista();
219
220         return lista;
221     }
222
223
224 }

```

Siguiendo el modelo Singleton y para facilitar el cambio de la capa de persistencia, esta capa solo realiza las llamadas a la clase DAO correspondiente que es la que realizara las operaciones con la base de datos.

```

1  package modelo;
2
3  import java.sql.Connection;
4  import java.sql.Date;
5  import java.sql.PreparedStatement;
6  import java.sql.ResultSet;
7  import java.sql.SQLException;
8  import java.util.ArrayList;
9  import java.util.List;
10
11  public class DaoCliente {
12
13
14      private Connection con = null;
15
16      private static DaoCliente instance = null;
17
18      private DaoCliente() throws SQLException {
19          con = DBConnection.getConnection();
20      }
21
22      public static DaoCliente getInstance() throws SQLException {
23          if (instance == null)
24              instance = new DaoCliente();
25
26          return instance;
27      }
28
29
30      /**
31       *
32       * Metodo que inserta mediante el conector JDBC los datos de un cliente en la Base de
33       *
34       * @param t
35       * @throws SQLException
36       */
37      public void insert(Cliente t) throws SQLException {
38
39          PreparedStatement ps = con.prepareStatement(
40              "INSERT INTO clientes(dni, nombre, apellidos, nombreVia, numero, piso, pue
41              ps.setString(1, t.getDni());
42              ps.setString(2, t.getNombre());

```

```

43     ps.setString(3, t.getApellidos());
44     ps.setString(4, t.getNombreVia());
45     ps.setInt(5, t.getNumero());
46     ps.setInt(6, t.getPiso());
47     ps.setString(7,t.getPuerta());
48     ps.setString(8,t.getCp());
49     ps.setString(9,t.getProvincia());
50     ps.setString(10,t.getPoblacion());
51     ps.setString(11,t.getPais());
52     ps.setInt(12, t.getTipoVia());
53
54     ps.executeUpdate();
55
56     ps.close();
57
58 }
59
60
61
62 /**
63  * Metodo que acutaliza los datos de un cliente mediante el id almacenado del mismo.
64  * @param t
65  * @throws SQLException
66  */
67 public void update(Cliente t) throws SQLException {
68
69     if (t.getId() != 0) {
70
71         PreparedStatement ps = con.prepareStatement(
72             "UPDATE clientes SET dni = ?, nombre = ?, apellidos = ?, nombreVia = ?, nu
73
74         ps.setString(1, t.getDni());
75         ps.setString(2, t.getNombre());
76         ps.setString(3, t.getApellidos());
77         ps.setString(4, t.getNombreVia());
78         ps.setInt(5, t.getNumero());
79         ps.setInt(6, t.getPiso());
80         ps.setString(7,t.getPuerta());
81         ps.setString(8,t.getCp());
82         ps.setString(9,t.getProvincia());
83         ps.setString(10,t.getPoblacion());
84         ps.setString(11,t.getPais());
85         ps.setInt(12, t.getTipoVia());
86         ps.setInt(13, t.getId());
87
88         ps.executeUpdate();
89
90         ps.close();
91
92     }
93
94 }
95
96
97
98 /**
99  * Metodo que retorna la lista de clientes almacenados en la base de datos.
100  * @return
101  * @throws SQLException
102  */
103 public List<Cliente> obtenerLista() throws SQLException {
104
105     PreparedStatement ps = con.prepareStatement("SELECT * FROM clientes");
106     ResultSet rs = ps.executeQuery();
107
108     List<Cliente> result = null;
109
110     while (rs.next()) {
111         if (result == null)

```

```

112         result = new ArrayList<>();
113
114         result.add(new Cliente(rs.getInt("id"), rs.getString("dni"),rs.getString("nom
115             rs.getString("puerta"), rs.getString("cp"), rs.getString("provincia"),
116     }
117
118     rs.close();
119     ps.close();
120
121     return result;
122 }
123
124 /*
125  * Metodo que obtiene los datos de un cliente por medio de la identificación del ID
126  */
127 public Cliente obtenerPorId(int id) throws SQLException {
128
129     PreparedStatement ps = con.prepareStatement("SELECT * FROM clientes WHERE id = ?")
130     ps.setInt(1, id);
131     ResultSet rs = ps.executeQuery();
132
133     Cliente result = null;
134
135     if (rs.next()) {
136         result = new Cliente(rs.getInt("id"), rs.getString("dni"),rs.getString("nombr
137             rs.getString("puerta"), rs.getString("cp"), rs.getString("provincia"),
138     }
139
140     rs.close();
141     ps.close();
142
143     return result;
144
145
146
147 }
148
149 /**
150  * Metodo para borrar un cliente de la base de datos proporcionandole el id del mismo.
151  * @param id
152  * @throws SQLException
153  */
154 public void delete(int id) throws SQLException {
155
156     if (id <= 0)
157         return;
158
159     PreparedStatement ps = con.prepareStatement("DELETE FROM Clientes WHERE id = ?");
160     ps.setInt(1, id);
161
162     ps.executeUpdate();
163
164     ps.close();
165 }
166
167
168
169 }

```

Podemos observar como cada uno de los métodos realiza una llamada a los métodos de la clase estática DaoClientes.

En esta clase es donde se construye las sentencias SQL para la manipulación de los datos. Podemos observar como en el constructor de la clase utilizamos el patrón singleton para realizar la conexión llamando a la clase DBConnection.

```
1 package modelo;
2
3 import java.sql.Connection;
4 import java.sql.DriverManager;
5 import java.sql.SQLException;
6 import java.util.Properties;
7
8
9
10 public class DBConnection {
11
12
13     private static final String JDBC_URL = "jdbc:mysql://localhost:3306/GestionClientesEjercici
14
15         private static Connection instance = null;
16
17         private DBConnection() { }
18
19         /**
20          * Metodo que permite la llamada estatica por los metodos del resto de clases, utilizan
21          * @return
22          * @throws SQLException
23          */
24         public static Connection getConnection() throws SQLException {
25             if (instance == null) {
26                 Properties props = new Properties();
27                 props.put("user", "root");
28                 props.put("password", "root");
29                 instance = DriverManager.getConnection(JDBC_URL, props);
30             }
31
32             return instance;
33         }
34
35
36 }
```

Junto a este documento se incluyen los siguientes recursos.

- Proyecto Eclipse con todo el código del programa. [Descargar](#)
- Compilación JAR del proyecto para su ejecución. [AplicacionClientes.jar](#)
- Script SQL de la base de datos. [baseDeDatos.sql](#)
- Documentación en JAVADOC. [Documentación](#)