



|  |  |
| --- | --- |
| Crear Base de datos |  |
| 1. Crea  una base de datos en tu DBMS  llamada  dbbiblioteca  create database dbbiblioteca; |
| 2. Crea un esquema scbiblioteca, en la base de datos dbbiblioteca  create schema scbiblioteca |
| 3. Crea  la relación cliente, en el esquema anterior  CREATE TABLE scbiblioteca.cliente (     nif character(10) PRIMARY KEY,     nombre character varying(50) NOT NULL,     apellidos character varying(50) NOT NULL,  ); |

**Vista**

Iniciemos con la vista

Vamos a crear una interfaz, que tenga en cuadros de texto los valores de nif, nombre y apellidos.

En la parte media, una tabla que despliegue los datos de la relación cliente

En la parte inferior, los botones Añadir, Borrar y Editar

**1.- Crear la clase de la vista, la cual será un JFrame**

public class VistaBiblioteca extends JFrame {

**2.- Declaramos los atributos de nuestra forma**  
    
    //CONTENEDOR PRINCIPAL  
    private JPanel contenedor;  
   
    //DECLARACIÓN DE LAS ETIQUETAS  
    private JLabel lblNombre;  
    private JLabel lblApellido;  
    private JLabel lblNIF;  
   
    //DECLARACIÓN DE LOS CUADROS DE TEXTO  
    protected JTextField txtNombre;  
    protected JTextField txtApellido;  
    protected JTextField txtNIF;  
   
    //DECLARACIÓN DE LOS BOTONES  
    protected JButton btnAdd;  // Agregar  
    protected JButton btnDel;  // Borrar  
    protected JButton btnUpd; // Editar  
    protected JButton btnExit; // Salir  
   
    //DECLARACIÓN DE LOS OBJETOS PARA LA TABLA  
    private JScrollPane scroll; //Panel de scroll que contiene la tabla  
    protected Object[][] datos; //Cuerpo de la tabla  
    protected String[] cabecera;    //Cabecera de la tabla  
    protected DefaultTableModel dtm;//Modelo de la tabla, el cual es el predeterminado  
    protected JTable tabla; //Tabla

**3.- Implementamos el constructor, agregando cada uno de los atributos declarados anteriormente en el contenedor principal, en una posición determinada**

    VistaBiblioteca() {  
        setBounds(100, 100, 450, 400);//Dimensiones del frame  
        setTitle("USUARIO DE LA BIBLIOTECA");    //Barra de título del frame  
        setDefaultCloseOperation(EXIT\_ON\_CLOSE);    //Acción al pulsar salir  
   
        //INSTANCIAR EL CONTENEDOR PRINCIPAL Y AÑADIRLO AL FRAME  
        contenedor = new JPanel();  
        getContentPane().add(contenedor);  
   
        //USAR EL LAYOUTMANAGER SpringLayout  
        SpringLayout sp = new SpringLayout();  
        contenedor.setLayout(sp);  
   
**4 .- Agregamos las etiquetas**

**A) Etiqueta nombre**  
        lblNombre = new JLabel("Nombre:");  //Crear la instancia  
        contenedor.add(lblNombre);      //Añadirlo al contenedor

       // ESTABLECER LOS CONSTRAINTS PARA UBICAR LA ETIQUETA NOMBRE  
        sp.putConstraint(SpringLayout.NORTH, lblNombre, 10, SpringLayout.NORTH, contenedor);  
        sp.putConstraint(SpringLayout.WEST, lblNombre,  10, SpringLayout.WEST, contenedor);  
**B) Etiqueta apellidos**

        lblApellido = new JLabel("Apellidos:");  
        contenedor.add(lblApellido);

       // ESTABLECER LOS CONSTRAINTS PARA UBICAR LA ETIQUETA APELLIDOS  
        sp.putConstraint(SpringLayout.NORTH, lblApellido, 50, SpringLayout.NORTH, contenedor);  
        sp.putConstraint(SpringLayout.WEST, lblApellido,  10, SpringLayout.WEST, contenedor);

**C) Etiqueta NIF**  
        lblNIF = new JLabel("NIF:");  
        contenedor.add(lblNIF);

       // ESTABLECER LOS CONSTRAINTS PARA UBICAR LA ETIQUETA NIF  
        sp.putConstraint(SpringLayout.NORTH, lblNIF, 90, SpringLayout.NORTH, contenedor);  
        sp.putConstraint(SpringLayout.WEST, lblNIF,  10, SpringLayout.WEST, contenedor);

**5) Instanciamos los cuadros de texto y ubicamos en el panel**  
**A) CUADRO DE TEXTO PARA EL NOMBRE**  
        txtNombre       = new JTextField();  
        contenedor.add(txtNombre);  
        sp.putConstraint(SpringLayout.NORTH, txtNombre, 10, SpringLayout.NORTH, contenedor);  
        sp.putConstraint(SpringLayout.WEST, txtNombre, 100, SpringLayout.WEST, contenedor);  
        sp.putConstraint(SpringLayout.EAST, txtNombre, 300, SpringLayout.WEST, contenedor);

**B )CUADRO DE TEXTO PARA EL NIF**  
        txtApellido = new JTextField();  
        contenedor.add(txtApellido);    //añadir al contenedor  
        sp.putConstraint(SpringLayout.NORTH, txtApellido, 50, SpringLayout.NORTH, contenedor);  
        sp.putConstraint(SpringLayout.WEST, txtApellido, 100, SpringLayout.WEST, contenedor);  
        sp.putConstraint(SpringLayout.EAST, txtApellido, 300, SpringLayout.WEST, contenedor);

**C)CUADRO DE TEXTO PARA LOS APELLIDOS**  
        txtNIF      = new JTextField();  
        contenedor.add(txtNIF);  
        sp.putConstraint(SpringLayout.NORTH, txtNIF, 90, SpringLayout.NORTH, contenedor);  
        sp.putConstraint(SpringLayout.WEST, txtNIF, 100, SpringLayout.WEST, contenedor);  
        sp.putConstraint(SpringLayout.EAST, txtNIF, 300, SpringLayout.WEST, contenedor);

**6. Instanciamos la tabla, definimos la cabecera, instanciamos el modelo y agregamos un JScrollPane para la tabla**

        scroll      = new JScrollPane();  
        cabecera    = new String[] {"NIF","NOMBRE","APELLIDOS"};  
        dtm         = new DefaultTableModel(datos,cabecera);  
        tabla       = new JTable(dtm);  
        scroll.setViewportView(tabla);  
        //se coloca el scrollpane...  
        contenedor.add(scroll); //añadir al contenedor  
        sp.putConstraint(SpringLayout.NORTH, scroll, 120, SpringLayout.NORTH, contenedor);  
        sp.putConstraint(SpringLayout.WEST, scroll,   10, SpringLayout.WEST, contenedor);  
        sp.putConstraint(SpringLayout.EAST, scroll,  -10, SpringLayout.EAST, contenedor);  
        sp.putConstraint(SpringLayout.SOUTH, scroll, -50, SpringLayout.SOUTH, contenedor);

**7- Instanciamos botones, los agregamos al contenedor y ubicamos en una posición   
     A) BOTÓN AÑADIR**  
        btnAdd          = new JButton("Añadir");  
        contenedor.add(btnAdd);  
        sp.putConstraint(SpringLayout.SOUTH, btnAdd, -10, SpringLayout.SOUTH, contenedor);//colocarlo  
        sp.putConstraint(SpringLayout.WEST, btnAdd,   35, SpringLayout.WEST, contenedor);  
        **B) BOTÓN BORRAR**  
        btnDel          = new JButton("Borrar");  
        contenedor.add(btnDel);  
        sp.putConstraint(SpringLayout.SOUTH, btnDel, -10, SpringLayout.SOUTH, contenedor);  
        sp.putConstraint(SpringLayout.WEST, btnDel,  145, SpringLayout.WEST, contenedor);  
        **C) BOTÓN MODIFICAR**  
        btnUpd          = new JButton("Editar");  
        contenedor.add(btnUpd);  
        sp.putConstraint(SpringLayout.SOUTH, btnUpd, -10, SpringLayout.SOUTH, contenedor);  
        sp.putConstraint(SpringLayout.WEST, btnUpd,  245, SpringLayout.WEST, contenedor);  
          
    **D) BOTÓN SALIR**  
        btnExit            = new JButton("Salir");  
        contenedor.add(btnExit);  
        sp.putConstraint(SpringLayout.SOUTH, btnExit, -10,  
                        SpringLayout.SOUTH, contenedor);  
        sp.putConstraint(SpringLayout.WEST, btnExit, 355,  
                        SpringLayout.WEST,contenedor);

**8.- Se hace visible el Frame**  
  
        setVisible(true);

**9.- Terminamos el constructor**  
    }

|  |
| --- |
| **10- Implementamos el método conectaControlador que permitirá conectar la vista con su controlador.**  **11- Se agrega a cada botón un ActionCommand, para identificar el botón pulsado** |
| public void conectaControlador(  ControladorBiblioteca c  ){           btnAdd.addActionListener(c);         btnAdd.setActionCommand("INSERTAR");           btnDel.addActionListener(c);         btnDel.setActionCommand("BORRAR");           btnUpd.addActionListener(c);         btnUpd.setActionCommand("MODIFICAR");                  btnExit.addActionListener(c);         btnExit.setActionCommand("SALIR");           tabla.addMouseListener(c);         //sólo se permite pulsar una fila a la vez.         //tabla.setSelectionMode(ListSelectionModel.SINGLE\_SELECTION);     } |
| **12. Finalizamos la vista**  } |
| **Importamos las bibliotecas utilizadas en la vista** |
| **import javax.swing.JFrame; import javax.swing.JPanel; import javax.swing.SpringLayout;** |
| **import javax.swing.JLabel; import javax.swing.JTextField; import javax.swing.JButton;** |
| **import javax.swing.JScrollPane; import javax.swing.table.DefaultTableModel; import javax.swing.JTable;** |

**Modelo**

|  |  |
| --- | --- |
| **1. Declara la clase ModeloBiblioteca** | |
| public class ModeloBiblioteca { | |
| **2. Declara los atributos y asigna las credenciales de tu dbms** | |
| private String  host     = "localhost";     private String  usuario     = "postgres";     private String  clave       = "admin";     private int     puerto      = 5432;     private String  servidor    = "";      private String baseDatos;     private static Connection conexion  = null; | |
| **3. Crea tu constructor, el cual recibirá como parámetro el nombre de la base de datos con la cual trabajará** | |
| **public ModeloBiblioteca(String baseDatos){         this.baseDatos = baseDatos;         ConexionBd();     }** | |
| **4. Crea el método para crear la conexión a la base de datos.** | | | |
| **protected void ConexionBd(){         this.servidor="jdbc:postgresql://"+host+":"+ puerto+"/"+baseDatos;           //Registrar el driver         try {            Class.forName("org.postgresql.Driver");         } catch (ClassNotFoundException e) {             System.err.println("ERROR AL REGISTRAR EL DRIVER "+ e);             System.exit(0); //parar la ejecución         }           //Establecer la conexión con el servidor         try {             conexion = DriverManager.getConnection(this.servidor,                         this.usuario, this.clave);         } catch (SQLException e) {             System.err.println("ERROR AL CONECTAR CON EL SERVIDOR");             System.exit(0); //parar la ejecución         }         System.out.println("Conectado a "+baseDatos);     }       //Devuelve el objeto Connection que se usará en la clase Controller     private Connection getConexion() {         return conexion;     }** | | | |
| **4. Implementamos los métodos para insertar, actualizar, eliminar datos de los clientes** |  | |
| protected boolean insertCliente(Cliente c){         //Objeto para ejecutar las instrucciones en la base de datos         PreparedStatement ps;         String sqlInsertCliente = "insert into scbiblioteca.cliente values (?,?,?);";         try{             //Preparar la instrucción             ps  = getConexion().prepareStatement(sqlInsertCliente);             //Indicar qué información se pasa al Statement             ps.setString(1, c.getNif());             ps.setString(2, c.getNombre());             ps.setString(3, c.getApellidos());                          //Ejecutar el comando insert             ps.executeUpdate();             return true;         }catch (SQLException e) {             System.err.println("Error en la INSERCIÓN " + e );             return false;         }     } |
| protected boolean deleteCliente(Cliente c){         //Objeto para ejecutar los procedimientos almacenados en la base de datos         PreparedStatement ps;         String sqlDeleteCliente = "delete from scbiblioteca.cliente where nif = ?;";         try{             //Preparar la llamada             ps  = getConexion().prepareStatement(sqlDeleteCliente);              //Indicar qué información se pasa al Statement             ps.setString(1, c.getNif());             //Ejecutar el procedimiento             ps.executeUpdate();             //System.out.println(this.view.dtm.getValueAt(filaPulsada, 0));             return true;         }catch (SQLException e) {             System.err.println("Error en el BORRADO "+ e);             return false;         }     }  protected boolean updateCliente(Cliente c){  //Objeto para ejecutar los procedimientos almacenados en la base de datos  PreparedStatement ps;  String sqlUpdateCliente = "update scbiblioteca.cliente set nombre = ?, apellidos = ? where nif = ?;";  try{  //Preparar la llamada  ps = getConexion().prepareStatement(sqlUpdateCliente);    //Indicar qué información se pasa al procedimiento  ps.setString(1, c.getNombre());  ps.setString(2, c.getApellidos());  ps.setString(3, c.getNif());  //Ejecutar el procedimiento  ps.executeUpdate();  //System.out.println(this.view.dtm.getValueAt(filaPulsada, 0));  return true;  }catch (SQLException e) {  System.err.println("Error en la MODIFICACION");  return false;  }  } |

|  |
| --- |
| **5. Implementar los métodos para consultar y listar datos de clientes** |
| protected Cliente selectCliente(Cliente c){         //Objeto para ejecutar los procedimientos almacenados en la base de datos         PreparedStatement ps;         //Objeto para recoger los datos devueltos por el procedimiento almacenado         ResultSet rs;                  // Objeto con el cliente encontrado         Cliente clienteEncontrado= null;                      String sqlConsulta = "select nif, nombre, apellidos from scbiblioteca.cliente where nif = ?;";         try{             //Preparar el statement             ps  = getConexion().prepareStatement(sqlConsulta);                                     //Indicar qué información se pasa al statement             ps.setString(1, c.getNif());             //Ejecutar el procedimiento             rs  = ps.executeQuery();             //Cargar los datos devueltos en una instancia cliente             if(rs.next()){                 clienteEncontrado = new Cliente(rs.getString(1), rs.getString(2),rs.getString(3));                        }                      }catch (SQLException e) {             System.err.println("Error al CARGAR UN CLIENTE "+ e);         }         return clienteEncontrado;     } |

|  |
| --- |
| **Método que regresa en una  lista, los clientes registrados en la bd** |
| protected List<Cliente> listCliente(){         PreparedStatement ps;         //Objeto para recoger los datos devueltos         ResultSet rs;          //Obtener datos de todos los clientes         String consultaSQL = "Select nif, nombre, apellidos from scbiblioteca.cliente;";         // Objeto List que contendrá todos los clientes         List<Cliente> clientes = new ArrayList<Cliente>();         try {             //Preparar el statement con la consulta SQL             ps  = getConexion().prepareStatement(consultaSQL);                                    //Ejecutarla y obtiene en rs el resultado             rs  = ps.executeQuery();                          //Recorrer el resultado para crear instancias de Cliente             while(rs.next()){                 Cliente c = new Cliente();                 //Añadir registro a registro en el vector                 c.setNif(rs.getString("nif"));                 c.setNombre(rs.getString("nombre"));                 c.setApellidos(rs.getString("apellidos"));                  // agregar a la lista cada uno de los clientes                 clientes.add(c);             }           } catch (SQLException e) {             System.err.println("Error al CARGAR DATOS " + e);         }         return clientes;     } |

|  |  |  |
| --- | --- | --- |
| **6. Cerrar la conexión activa a la base de datos** | | |
| protected void closeConexion(){      // verifica que la conexión esté activa         if ( getConexion() != null){             try {                 getConexion().close();             } catch(SQLException e){                 System.err.println("Error al cerrar la bd "+ e);             }         }     } | | |
| **7. Cerramos nuestra clase** | | |
| } | | |
| **8. Importamos las clases usadas** |  |  | |
| import java.sql.Connection; import java.sql.DriverManager; import java.sql.SQLException; import java.sql.ResultSet; import java.sql.PreparedStatement; import java.util.List; import java.util.ArrayList; |  |  | |

**Cliente**

|  |
| --- |
| **¿Y nuestro cliente?** |
| Como te pudiste dar cuenta, en el modelo utilizamos instancias de tipo Cliente, la cual aún no creamos |
| Esta clase tiene los atributos nif, nombre y apellidos sus métodos constructores y métodos sets y gets |

public class Cliente {

private String nif;

private String nombre;

private String apellidos;

public Cliente(String nif, String nombre, String apellidos){

this.nif = nif;

this.nombre = nombre;

this.apellidos = apellidos;

}

public Cliente(){

}

public void setNif (String nif){

this.nif = nif;

}

public void setNombre(String nombre){

this.nombre = nombre;

}

public void setApellidos(String apellidos){

this.apellidos = apellidos;

}

public String getNif(){

return nif;

}

public String getNombre() {

return nombre;

}

public String getApellidos(){

return apellidos;

}

}

**Controlador**

|  |  |
| --- | --- |
| **1. Para iniciar nuestro controlador, declaramos nuestra clase ControladorBiblioteca, la cual implementará ActionListener y MouseListener** | |
| public class ControladorBiblioteca implements ActionListener, MouseListener { | |
| **2. Declaramos sus atributos, que serán la vista y el modelo** | |
| private VistaBiblioteca view;     private ModeloBiblioteca modelo; | |
| **Constructor** |  | |
| **ControladorBiblioteca( VistaBiblioteca view , ModeloBiblioteca modelo){         this.view   = view;         this.modelo = modelo;         cargarTabla();     }** | | |

**4. Implementamos el método ActionPerformed, el cual determinará el botón presionado (a través del ActionCommand, para establecer la acción que. se realizará. Si la acción requiere interacción con la base de datos, se realiza a través del modelo**

    public void actionPerformed(ActionEvent arg0) {  
  
        Cliente c;  
   
        //COMANDO EJECTUADO  
        String comando  = arg0.getActionCommand();   
        //Deberá coincidir con alguno de los parámetros indicados en setActionCommand invocado en la  
        //  clase VistaBiblioteca  
        switch (comando) {  
            case "INSERTAR":  
                c = new Cliente(this.view.txtNIF.getText(), this.view.txtNombre.getText(), this.view.txtApellido.getText());  
                modelo.insertCliente(c);  
            break;  
   
            case "BORRAR":  
                //Obtener qué fila se ha pulsado  
                int filaPulsada = this.view.tabla.getSelectedRow();  
                //Si se ha pulsado una fila  
                if(filaPulsada>=0){  
                    //Se obtiene el nif del cliente

                    //int identificador   = (int)this.view.dtm.getValueAt(filaPulsada, 0);  
                    c = new Cliente();  
                    String nif = (String) this.view.dtm.getValueAt(filaPulsada,0);  
                    c.setNif(nif);  
                    modelo.deleteCliente(c);  
                }  
   
            break;  
   
            case "MODIFICAR":  
                //Obtener qué fila se ha pulsadao en la tabla  
                filaPulsada = this.view.tabla.getSelectedRow();  
                //Si se ha pulsado una fila  
                if(filaPulsada>=0){  
                    //Se obtiene el nif de la fila pulsada  
                    c = new Cliente();  
                    String nif = (String) this.view.dtm.getValueAt(filaPulsada,0);  
                    c.setNif(nif);

                    // Se obtienen el nombre y los apellidos de los cuadros de texto  
                    c.setNombre(this.view.txtNombre.getText());  
                    c.setApellidos(this.view.txtApellido.getText());  
                    modelo.updateCliente(c);                      
                }  
            break;  
               
            case "SALIR":

                // Si se seleccionó salir, se invoca el método que cierra nuestra conexión a la bd y cierra el frame  
                 modelo.closeConexion();  
                 this.view.dispose();  
            break;  
              
            default:  
                System.err.println("Comando no definido");  
            break;  
        }  
        //limpiar el formulario  
        limpia();  
   
        //refrescar la tabla  
        cargarTabla();  
    }

**5. Creamos métodos adicionales para limpiar y cargar los datos de la vista y la tabla**

    //Método para limpiar los campos de la ventana  
    private void limpia(){  
        this.view.txtNombre.setText("");  
        this.view.txtApellido.setText("");  
        this.view.txtNIF.setText("");  
    }  
   
    //Método que recarga los datos de la tabla de la base de datos  
    // en la tabla de la clase View

// Método que carga los datos del listado de clientes que se obtuvo del modelo en la tabla

  protected void cargarTabla(){  
   
        Vector<Object> fila;  
   
        //Limpiar los datos de la tabla  
        for(int i=this.view.dtm.getRowCount(); i>0; i--){  
            this.view.dtm.removeRow(i-1);  
        }

       // Listado de los clientes que retornó el modelo  
        List<Cliente> clientes = modelo.listCliente();  
        for(Cliente c: clientes){  
                //Añadir registro a registro en el vector  
                fila    = new Vector<Object>();  
                fila.add(c.getNif());  
                fila.add(c.getNombre());  
                fila.add(c.getApellidos());  
                //Añadir el vector a la tabla de la clase View  
                this.view.dtm.addRow(fila);  
            }  
    }

**6. Implementamos el método mouseClicked. Este médoto obtiene la fila pulsada en la tabla, para buscar los datos en la base de datos a través del modelo y cargarlos en los cuadros de texto correspondientes.**

    public void mouseClicked(MouseEvent arg0) {

        //Recoger qué fila se ha pulsadao en la tabla  
        int filaPulsada = this.view.tabla.getSelectedRow();  
        //Si se ha pulsado una fila  
        if(filaPulsada>=0){  
            //Se recoge el id de la fila marcada  
            Cliente c = new Cliente();  
            String nif= (String) this.view.dtm.getValueAt(filaPulsada, 0);  
            c.setNif(nif);  
            Cliente c2 = modelo.selectCliente(c);  
            if (c2 != null){  
                this.view.txtNIF.setText(c2.getNif());  
                this.view.txtNombre.setText(c2.getNombre());  
                this.view.txtApellido.setText(c2.getApellidos());  
            }

        }  
    }  
**// Se implementan el resto de los métodos declarados en las interfaces**  
    @Override  
    public void mouseEntered(MouseEvent arg0) {}  
    @Override  
    public void mouseExited(MouseEvent arg0) {}  
    @Override  
    public void mousePressed(MouseEvent arg0) {}  
    @Override  
    public void mouseReleased(MouseEvent arg0) {}

**7 .- Se cierra la clase**  
}

|  |
| --- |
| **8.- Importamos las clases utilizadas de las bibliotecas de Java** |
| import java.awt.event.ActionEvent; import java.awt.event.ActionListener; import java.awt.event.MouseEvent; import java.awt.event.MouseListener; |
| import java.util.List; import java.util.ArrayList; import java.util.Vector; |

