Creación de base de datos y tabla en MySQL

```
CREATE DATABASE lucherpan;

USE lucherpan;

CREATE TABLE shittlock (
    ID INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL,
    fecha_registro DATE NOT NULL,
    descripcion_actividad TEXT,
    fecha_limite_actividad DATE,
    cumplida ENUM('si', 'no') DEFAULT 'no'
);
```

Cópialo y pégalo en tu herramienta de gestión de bases de datos o en el editor SQL que estés utilizando. Esta consulta insertará las cuatro filas con los datos especificados en la tabla "shittlock".

-- Primer insert

INSERT INTO shittlock (nombre, email, fecha_registro, descripcion_actividad, fecha_limite_actividad, cumplida)

VALUES ('María López', 'maria@example.com', '2024-07-11', 'Preparar informe mensual', '2024-07-20', 'no');

-- Segundo insert

INSERT INTO shittlock (nombre, email, fecha_registro, descripcion_actividad, fecha_limite_actividad, cumplida)

VALUES ('Pedro Ramirez', 'pedro@example.com', '2024-07-12', 'Revisar diseño del nuevo producto', '2024-07-18', 'si');

-- Tercer insert

INSERT INTO shittlock (nombre, email, fecha_registro, descripcion_actividad, fecha_limite_actividad, cumplida)

VALUES ('Ana Martínez', 'ana@example.com', '2024-07-10', 'Organizar reunión de equipo', '2024-07-15', 'si');

-- Cuarto insert

INSERT INTO shittlock (nombre, email, fecha_registro, descripcion_actividad, fecha_limite_actividad, cumplida)

VALUES ('Carlos Sánchez', 'carlos@example.com', '2024-07-09', 'Actualizar base de datos de clientes', '2024-07-14', 'no');

-- Quinto insert

INSERT INTO shittlock (nombre, email, fecha_registro, descripcion_actividad, fecha_limite_actividad, cumplida)

VALUES ('Laura García', 'laura@example.com', '2024-07-08', 'Preparar presentación para cliente', '2024-07-13', 'si');

Paso 1: Crear el Proyecto

1. Abre tu IDE y crea un nuevo proyecto Java llamado "Examen".

Paso 2: Crear Source Packages

1. En el proyecto, crea dos source packages:
    - Uno llamado `connection`.
    - Otro llamado `Formularios`.

Paso 3: Crear Clases Dentro de Cada Source Package

1. En el source package `connection`, crea una clase llamada `Conexión`.

2.En el source package `Formularios`, crea una clase JFrame Form llamada `Registro`.

- Usa las herramientas de tu IDE para crear un formulario JFrame.

Paso 4: Implementar Lógica y Diseño

- Agrega el código necesario en cada clase según lo que necesites hacer en tu proyecto.

Establecer conexión a base de datos MySQL en Java mediante la clase Conexion en el paquete connection.

```
package connection;


import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;



public class Conexion {

    public static Connection conectar() throws SQLException {
    Connection conexion = null;

    try {
       // Cargar el driver JDBC de MySQL
```

```
            Class.forName("com.mysql.cj.jdbc.Driver");


            // Establecer la conexión con la base de datos
            conexion =
DriverManager.getConnection("jdbc:mysql://localhost:3306/nuclear?useUnicode=true&useJDBCC
ompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC", "root", "");


        } catch (ClassNotFoundException | SQLException e) {
            throw new SQLException("Error en la conexión: " + e.getMessage());
        }


        return conexion;
    }


}
```

---

la segunda source package "Formularios" y la creación de la JFrame Form "Registro":

- Segunda source package: Formularios
- JFrame Form: Registro
- Componentes:
    - Tabla (JTable) con nombre de variable "Tabla"
    - Cuadro de texto (JTextField) para búsqueda con nombre de variable "txtBuscar"

```
package Formularios;



//************************************//


import connection.Conexion; // Importa la clase de conexión personalizada

import java.sql.Connection; // Importa la clase Connection de java.sql para manejar conexiones JDBC

import java.sql.PreparedStatement; // Importa la clase PreparedStatement de java.sql para ejecutar consultas SQL parametrizadas

import java.sql.ResultSet; // Importa la clase ResultSet de java.sql para manejar resultados de consultas

import java.sql.SQLException; // Importa SQLException para manejar excepciones relacionadas con SQL

import java.util.logging.Level; // Importa Level para niveles de registro

import java.util.logging.Logger; // Importa Logger para manejar registros de eventos

import javax.swing.JOptionPane; // Importa JOptionPane para mostrar mensajes de diálogo

import javax.swing.table.DefaultTableModel; // Importa DefaultTableModel para manejar el modelo de tabla en Swing



//************************************//
```

```
public class Registro extends javax.swing.JFrame {



 //*************************************************//


    private Connection conexion;
    private DefaultTableModel modelo;



 //*************************************************//
```

```
public Registro() {

    initComponents();

    //*************************************************//

    setLocationRelativeTo(null);
    try {
        conexion = Conexion.conectar();
        Mostrar("");
    } catch (SQLException ex) {
        Logger.getLogger(Registro.class.getName()).log(Level.SEVERE, null, ex);
        JOptionPane.showMessageDialog(this, "Error al conectar a la base de datos");
    }

    //*************************************************//

}
```

```
//**********************Consulta de la Tabla*************************//


   public void Mostrar(String Nombre)  {


        modelo = new DefaultTableModel();
        modelo.addColumn("ID");
        modelo.addColumn("Nombre");
        modelo.addColumn("Email");
        modelo.addColumn("Fecha de registro");
        modelo.addColumn("Descripción de actividades");
        modelo.addColumn("Fecha límite de actividades");
        modelo.addColumn("Cumplida");


         Tabla.setModel(modelo);


        String sql = Nombre.isEmpty() ? "SELECT * FROM shittlock" : "SELECT * FROM shittlock WHERE
Nombre LIKE ?";
```

```
        try (PreparedStatement ps = conexion.prepareStatement(sql)) {
            if (!Nombre.isEmpty()) {
                ps.setString(1, "%" + Nombre + "%");
            }
            ResultSet rs = ps.executeQuery();

            while (rs.next()) {
                Object[] fila = new Object[7];
                for (int i = 0; i < 7; i++) {
                    fila[i] = rs.getObject(i + 1);
                }
                modelo.addRow(fila);
            }
        } catch (SQLException ex) {
            Logger.getLogger(Registro.class.getName()).log(Level.SEVERE, null, ex);
            JOptionPane.showMessageDialog(this, "Error al ejecutar la consulta");
        }
    }



    //*****************************************************************//
```

```
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jScrollPane1 = new javax.swing.JScrollPane();
        Tabla = new javax.swing.JTable();
        txtBuscar = new javax.swing.JTextField();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

        Tabla.setModel(new javax.swing.table.DefaultTableModel(
            new Object [][] {

            },
            new String [] {
                "ID", "Nombre", "Email", "Fecha de registro", "Descripcion de actividades", "Fecha limite de actividades", "Cumplida"
            }
        ));
        jScrollPane1.setViewportView(Tabla);

        txtBuscar.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyReleased(java.awt.event.KeyEvent evt) {
                txtBuscarKeyReleased(evt);
            }
        });
```

```
        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createSequentialGroup()
                        .addGap(14, 14, 14)
                        .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 790, javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addGroup(layout.createSequentialGroup()
                        .addGap(300, 300, 300)
                        .addComponent(txtBuscar, javax.swing.GroupLayout.PREFERRED_SIZE, 182, javax.swing.GroupLayout.PREFERRED_SIZE)))
                .addContainerGap(20, Short.MAX_VALUE))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
                .addContainerGap(394, Short.MAX_VALUE)
                .addComponent(txtBuscar, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(27, 27, 27)
                .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 137, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(46, 46, 46))
        );

        pack();
    }// </editor-fold>
```

```
private void txtBuscarKeyReleased(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
```

Este código llama a una función o método llamado Mostrar pasándole como argumento el texto obtenido del cuadro de texto txtBuscar.

```
//*********************************************//

Mostrar(txtBuscar.getText());

//*********************************************//
```

```
    }


    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
         * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(Registro.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        } catch (InstantiationException ex) {
```

```
java.util.logging.Logger.getLogger(Registro.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(Registro.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(Registro.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        }
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new Registro().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JTable Tabla;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JTextField txtBuscar;
    // End of variables declaration
}
```

2 PASO

Una lista de componentes y nombres de variables correspondientes para un formulario (JFrame):

Campos de texto:

ID: txtId

Nombre: txtNombre

Email: txtEmail

Fecha de Registro: txtfecha_registro

Descripción de actividad: txtdescripcion_actividad

Fecha límite de actividad: txtfecha_limite_actividad

Combobox:

- ¿Cumplida?: cbo_cumplida

Botones:

Guardar: btnGuardar

Modificar: btnModificar

Eliminar: btnEliminar

Limpiar: btnLimpiar

3 PASO:

Este código ejecuta las funciones `Agregar()` y `Mostrar("")` cuando se hace clic en el botón "Agregar" (`btnGuardar`).

```
//****************************************//
Agregar();
Mostrar("");
//****************************************//
```

La función Agregar() guarda datos ingresados en un formulario en una base de datos MySQL usando PreparedStatement. Si se guarda correctamente, muestra un mensaje y limpia una tabla en la interfaz.

```
//*************Agregar************************//


 void Agregar() {
        String nombre = txtNombre.getText().trim();
        String email = txtEmail.getText().trim();
        String fechaRegistro = txtfecha_registro.getText().trim();
```

```
        String descripcionActividad = txtdescripcion_actividad.getText().trim();
        String fechaLimiteActividad = txtfecha_limite_actividad.getText().trim();
        String cumplida = cbo_cumplida.getSelectedItem().toString().trim();

        try {
            if (nombre.isEmpty() || email.isEmpty() || fechaRegistro.isEmpty() || descripcionActividad.isEmpty()
                    || fechaLimiteActividad.isEmpty() || cumplida.isEmpty()) {
                JOptionPane.showMessageDialog(null, "Faltan ingresar datos");
                return; // Sale del método si faltan datos
            }

            // Preparar la consulta con PreparedStatement
            String sql = "INSERT INTO  shittlock(nombre, email, fecha_registro, descripcion_actividad, fecha_limite_actividad, cumplida) VALUES (?, ?, ?, ?, ?, ?)";
            PreparedStatement pstmt = conexion.prepareStatement(sql);
            pstmt.setString(1, nombre);
            pstmt.setString(2, email);
            pstmt.setString(3, fechaRegistro);
            pstmt.setString(4, descripcionActividad);
            pstmt.setString(5, fechaLimiteActividad);
            pstmt.setString(6, cumplida);

            // Ejecutar la consulta
            int filasAfectadas = pstmt.executeUpdate();
            if (filasAfectadas > 0) {
                JOptionPane.showMessageDialog(null, "Nueva entrada registrada exitosamente");
                limpiarTabla(); // Método para limpiar los campos después de la inserción
            } else {
                JOptionPane.showMessageDialog(null, "Error al registrar la entrada");
```

```
            }

            pstmt.close(); // Cerrar PreparedStatement

        } catch (SQLException e) {
            JOptionPane.showMessageDialog(null, "Error en la conexión: " + e.getMessage());
        }
    }

    void limpiarTabla() {
        DefaultTableModel modelo = (DefaultTableModel) Tabla.getModel();
        while (modelo.getRowCount() > 0) {
            modelo.removeRow(0);
        }
    }

    //*********************************************//
```

4 PASO

Maneja el evento de clic sobre una fila en una tabla. Si una fila está seleccionada, obtiene los datos de esa fila y los muestra en campos de texto y un combobox correspondiente en la interfaz gráfica.

```
int fila = Tabla.getSelectedRow();

if (fila == -1) {
    JOptionPane.showMessageDialog(null, "No se ha seleccionado ninguna fila");
} else {
    // Obtener el valor del ID de la fila seleccionada
    int idc = Integer.parseInt(Tabla.getValueAt(fila, 0).toString());

    // Obtener los demás valores de la fila seleccionada
    String nombre = Tabla.getValueAt(fila, 1).toString();
    String email = Tabla.getValueAt(fila, 2).toString();
    String fechaRegistro = Tabla.getValueAt(fila, 3).toString();
    String descripcionActividad = Tabla.getValueAt(fila, 4).toString();
    String fechaLimiteActividad = Tabla.getValueAt(fila, 5).toString();
    String cumplida = Tabla.getValueAt(fila, 6).toString();

    // Mostrar el ID en el campo correspondiente
    txtId.setText(String.valueOf(idc));

    // Mostrar los demás valores en los campos de texto
    txtNombre.setText(nombre);
    txtEmail.setText(email);
    txtfecha_registro.setText(fechaRegistro);
    txtdescripcion_actividad.setText(descripcionActividad);
```

```
    txtfecha_limite_actividad.setText(fechaLimiteActividad);

    // Seleccionar el valor correspondiente en el ComboBox
    cbo_cumplida.setSelectedItem(cumplida);
}
```

5 PASO

Boton Modificar : btnModificar

Llama a la función Modificar() y luego llama a Mostrar("").

```
Modificar();

    Mostrar("");
```

La función Modificar() actualiza un registro en una base de datos MySQL basado en los datos modificados obtenidos de campos de texto y un combobox en la interfaz gráfica.

```
}
void Modificar() {
                // Obtener los valores modificados desde los campos de texto y el ComboBox
                int id = Integer.parseInt(txtId.getText());
                String nombre = txtNombre.getText().trim();
                String email = txtEmail.getText().trim();
                String fechaRegistro = txtfecha_registro.getText().trim();
                String descripcionActividad = txtdescripcion_actividad.getText().trim();
                String fechaLimiteActividad = txtfecha_limite_actividad.getText().trim();
                String cumplida = cbo_cumplida.getSelectedItem().toString().trim();

                try {
                    // Preparar la consulta SQL UPDATE
                    String sql = "UPDATE shittlock SET nombre=?, email=?, fecha_registro=?,
descripcion_actividad=?, fecha_limite_actividad=?, cumplida=? WHERE id=?";
                    PreparedStatement pstmt = conexion.prepareStatement(sql);
                    pstmt.setString(1, nombre);
                    pstmt.setString(2, email);
                    pstmt.setString(3, fechaRegistro);
                    pstmt.setString(4, descripcionActividad);
                    pstmt.setString(5, fechaLimiteActividad);
                    pstmt.setString(6, cumplida);
                    pstmt.setInt(7, id);

                    // Ejecutar la consulta
                    int filasAfectadas = pstmt.executeUpdate();
                    if (filasAfectadas > 0) {
                        JOptionPane.showMessageDialog(null, "Registro actualizado correctamente");
```

```
                limpiarTabla(); // Método para limpiar los campos después de la modificación
            } else {
                JOptionPane.showMessageDialog(null, "Error al actualizar el registro");
            }

            pstmt.close(); // Cerrar PreparedStatement

        } catch (SQLException e) {
            JOptionPane.showMessageDialog(null, "Error en la conexión: " + e.getMessage());
        }
    }
```

6 PASO : Boton Eliminar btnEliminar

Llama a la función Eliminar() y luego llama a Mostrar("").

```
Eliminar();
    Mostrar("");
```

La función Eliminar() elimina un registro de una tabla en una base de datos MySQL basado en la fila seleccionada en una tabla de la interfaz gráfica. Luego llama a limpiarTabla() para actualizar la interfaz después de la eliminación.

```
void Eliminar() {
    int filaSeleccionada = Tabla.getSelectedRow();

    if (filaSeleccionada == -1) {
        JOptionPane.showMessageDialog(null, "Por favor, seleccione una fila para eliminar");
        return;
    }

    // Obtener el ID de la fila seleccionada en la tabla
    int id = Integer.parseInt(Tabla.getValueAt(filaSeleccionada, 0).toString());

    try {
        // Preparar la consulta SQL DELETE
        String sql = "DELETE FROM shittlock WHERE id=?";
        PreparedStatement pstmt = conexion.prepareStatement(sql);
        pstmt.setInt(1, id);

        // Ejecutar la consulta
        int filasAfectadas = pstmt.executeUpdate();
        if (filasAfectadas > 0) {
            JOptionPane.showMessageDialog(null, "Registro eliminado correctamente");
            limpiarTabla(); // Actualizar la tabla después de eliminar
        } else {
            JOptionPane.showMessageDialog(null, "Error al eliminar el registro");
        }

        pstmt.close(); // Cerrar PreparedStatement

    } catch (SQLException e) {
```

```
        JOptionPane.showMessageDialog(null, "Error en la conexión: " + e.getMessage());
    }
}
```

7 PASO : BOTON LIMPIAR btnLimpiar

Llama a la función limpiarCampos() para limpiar los campos en la interfaz y luego llama a Mostrar("") para actualizar la visualización.

```
limpiarCampos();
        Mostrar("");
```

La función limpiarCampos() se encarga de restablecer o limpiar todos los campos de texto y el ComboBox en la interfaz gráfica.

```
void limpiarCampos() {
    txtId.setText("");
    txtNombre.setText("");
    txtEmail.setText("");
    txtfecha_registro.setText("");
    txtdescripcion_actividad.setText("");
    txtfecha_limite_actividad.setText("");
    cbo_cumplida.setSelectedIndex(0); // Resetea el ComboBox al primer ítem
}
```

Creación de la base de datos :

```
CREATE DATABASE Nuclear;

USE Nuclear;

CREATE TABLE Shittlock (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL,
    fecha_registro DATE NOT NULL,
    descripcion_actividad TEXT,
    fecha_limite_actividad DATE,
    cumplida ENUM('No', 'Sí')
);
```

id: Es un identificador único para cada registro, se autoincrementa automáticamente.

nombre: Almacena el nombre del usuario o entidad.

email: Almacena la dirección de correo electrónico.

fecha_registro: La fecha en que se registró la actividad.

descripcion_actividad: Una descripción detallada de la actividad.

fecha_limite_actividad: La fecha límite para completar la actividad.

cumplida: Un campo ENUM que puede tener dos valores: 'No' o 'Sí', para indicar si la actividad ha sido cumplida o no.

Este esquema te permitirá almacenar la información necesaria para gestionar las actividades y su cumplimiento dentro de la base de datos "Nuclear".

Para insertar cinco registros de ejemplo en la tabla Shittlock, puedes usar la siguiente serie de sentencias INSERT INTO.

INSERT INTO Shittlock (nombre, email, fecha_registro, descripcion_actividad, fecha_limite_actividad, cumplida)

VALUES

('María López', 'maria@example.com', '2024-07-09', 'Preparar informe trimestral', '2024-07-20', 'No'),

('Carlos Martínez', 'carlos@example.com', '2024-07-09', 'Revisión de código', '2024-07-12', 'Sí'),

('Ana García', 'ana@example.com', '2024-07-08', 'Entrenamiento de nuevos empleados', '2024-07-15', 'No'),

('Pedro Ramírez', 'pedro@example.com', '2024-07-07', 'Desarrollo de prototipo', '2024-07-25', 'No'),

('Luisa Fernández', 'luisa@example.com', '2024-07-06', 'Planificación de proyecto', '2024-07-18', 'Sí');