UNIVERSIDAD DE CARTAGENA

INGENIERIA DE SOFTWARE

DESARROLLO DE SOFTWARE

PARCIAL FINAL

Presenta:

ANTONIO LUIS DE AVILA PAJARO

Código:

7502420052

Profesora:

JHON CARLOS ARRIETA ARRIETA

Cartagena de Indias, Bolívar NOVIEMBRE, 2025

Universidad de Cartagena
Fundada en 1827

Enlace del Repositorio

https://github.com/Shakespeares1/hexagonal-congresistas-antonio.git

Enlace del video de sustentación

https://www.youtube.com/watch?v=aTBofRBML84

Universidad de Cartagena
Fundada en 1827

INTRODUCCION

Este proyecto corresponde al examen final de la asignatura. El objetivo fue desarrollar una aplicación **Java de escritorio (Standalone, con GUI)** organizada **estrictamente bajo arquitectura Hexagonal**, implementando los flujos **CRUDL** para la entidad asignada durante el semestre: **Congresistas**.

Se construyó una estructura modular con **dominio, puertos, adaptadores y aplicación**, asegurando independencia entre capas y permitiendo que la aplicación funcione con diferentes orígenes de datos.

Universidad de Cartagena
Fundada en 1827

OBJETIVOS

Objetivo General

Diseñar e implementar una aplicación Java con arquitectura Hexagonal que permita realizar operaciones CRUDL sobre la base de datos del caso académico asignado.

Objetivos Específicos

- Implementar un **modelo de dominio puro**, aislado de bases de datos y frameworks.
- Definir puertos (interfaces) de entrada y salida que desacoplen reglas de negocio.
- Construir adaptadores hacia la base de datos, conteniendo conexión, sentencias SQL y ejecución real.
- Integrar una interfaz gráfica simple que permita operar la aplicación desde escritorio.
- Garantizar el correcto funcionamiento del CRUDL con sentencias SQL reales y conexión JDBC.

Universidad de Cartagena
Fundada en 1827

Arquitectura Hexagonal Implementada

La arquitectura hexagonal organiza la aplicación en capas independientes, donde el **dominio** es el centro, y todo lo demás son adaptadores que entran o salen por "puertos".
Esta estructura garantiza:

- Bajo acoplamiento
- Alto nivel de mantenimiento
- Facilidad de pruebas
- Independencia de herramientas externas

A continuación se explica cada una de las capas implementadas en el proyecto.

4.1 Dominio

- Representa el modelo central del negocio, es decir, la información relevante de un Congresista.
- No contiene SQL, ni conexiones, ni elementos de la interfaz gráfica.
- Está completamente libre de dependencias externas (JDBC, Swing, etc.).
- Es una clase pura, enfocada únicamente en los datos y reglas básicas del negocio.
- En arquitectura Hexagonal, el dominio es el núcleo, y esta clase cumple con esa función.

```
public class Congresista {
    private int id;
    private String nombre;
    private String apellido;
    private String email;
    private String pais;

    public Congresista() {}

    public Congresista(int id, String nombre, String apellido, String email, String pais) {
        this.id = id;
```

```
        this.nombre = nombre;
        this.apellido = apellido;
        this.email = email;
        this.pais = pais;
    }
    // Getters y Setters
}
```

Contiene:

- Entidad **Congresista**
- Reglas de negocio mínimas (validaciones)
- Sin importaciones de JDBC ni frameworks

4.2 Puertos

Define **interfaces**:

- CongresistaRepository (puerto de salida)

```java
package com.congreso.ports;

import com.congreso.domain.Congresista;
import java.util.List;

public interface CongresistaRepository {
    void crear(Congresista c);
    void actualizar(Congresista c);
    void eliminar(int id);
    Congresista buscarPorId(int id);
    List<Congresista> listar();
}
```

- CongresistaServicePort (puerto de entrada)

```java
package com.congreso.ports;

import com.congreso.domain.Congresista;
import java.util.List;

public interface CongresistaServicePort {

    void crear(Congresista c);
    void actualizar(Congresista c);
    void eliminar(int id);
    Congresista buscarPorId(int id);
    List<Congresista> listar();
}
```

¿Por qué son puertos?

- Definen **contratos** que deben cumplir los adaptadores.
- **No tienen implementaciones**, solo describen lo que la aplicación espera del exterior.
- Permiten que la aplicación funcione sin acoplarse a MySQL, JDBC o la GUI.
- El puerto de salida describe cómo la capa de aplicación ve la base de datos.
- El puerto de entrada define las operaciones que los adaptadores primarios (GUI) pueden ejecutar.
- Gracias a los puertos, la aplicación puede cambiar de BD, cambiar la GUI o incluso ser API REST sin tocar el dominio.

Universidad de Cartagena
Fundada en 1827

Adaptadores

Son implementaciones concretas:

- **Adaptador de Persistencia (JDBC MySQL)**
  - Conexión
  - Sentencias SQL
  - ResultSet
  - PreparedStatements
  - CRUDL real
- **Adaptador de entrada (GUI Swing)**
  Ventanas que llaman métodos del servicio.

Aplicación / Servicio

Explicación

- Orquesta los casos de uso (CRUDL) del sistema.
- Implementa el puerto de entrada (CongresistaServicePort).
- Se comunica exclusivamente con el puerto de salida (CongresistaRepository).
- No tiene SQL, ni JDBC, ni Swing — está completamente separada de la infraestructura.
- Es la capa que aplica cualquier regla de negocio futura.

```
public class CongresistaService implements CongresistaServicePort {
  private final CongresistaRepository repository;
   public CongresistaService(CongresistaRepository repository) {
      this.repository = repository;
   }
   @Override
   public void crear(Congresista c) {
      repository.crear(c);
   }
   @Override
```

```
    public void actualizar(Congresista c) {
        repository.actualizar(c);
    }
    @Override
    public void eliminar(int id) {
        repository.eliminar(id);
    }

    @Override
    public Congresista buscarPorId(int id) {
        return repository.buscarPorId(id);
    }

    @Override
    public List<Congresista> listar() {
        return repository.listar();
    }
}
```

Implementa la lógica de los casos de uso:

- Crear congresista
- Listar congresistas
- Actualizar
- Eliminar
- Consultar por país (si aplica)

Adaptador de Entrada – GUI (Swing)

Aquí no hace falta pegar todo el código, solo un fragmento:

```
btnCrear.addActionListener(e -> {
    Congresista c = new Congresista(
        0,
        txtNombre.getText(),
        txtApellido.getText(),
        txtEmail.getText(),
        txtPais.getText()
    );
    service.crear(c);
});
```

o

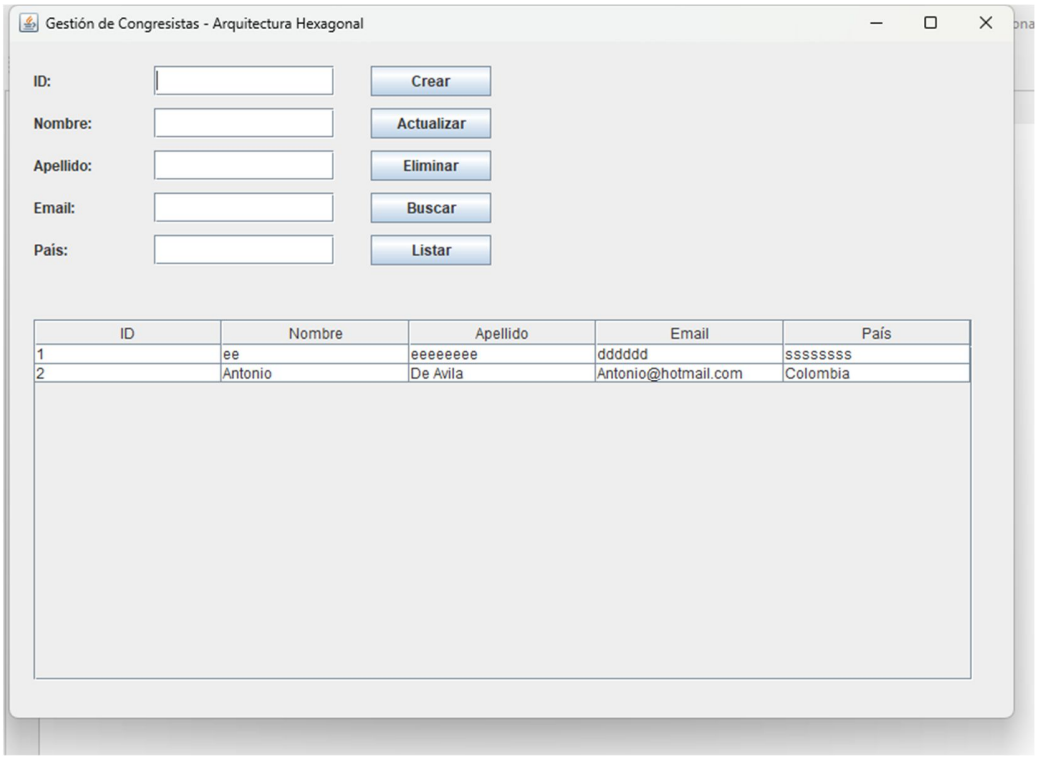
```
private void cargarTabla() {
    tableModel.setRowCount(0);
    List<Congresista> lista = service.listar();
    for (Congresista c : lista) {
        tableModel.addRow(new Object[]{
            c.getId(), c.getNombre(), c.getApellido(), c.getEmail(), c.getPais()
        });
    }
```

Explicación – ¿Por qué es un Adaptador de Entrada?

- Permite que el usuario interactúe con la aplicación mediante botones y formularios.
- Llama a los métodos del servicio (puerto de entrada).
- No tiene lógica de negocio, ni SQL.
- Solo captura datos, los envía al servicio y muestra resultados.

- Puede reemplazarse por otra interfaz (Web, móvil, API REST) sin afectar el dominio

Ensamblaje – Clase Main




```
public class MainApp {

    public static void main(String[] args) {

        MySQLCongresistaRepository repo = new MySQLCongresistaRepository();
        CongresistaService service = new CongresistaService(repo);

        SwingUtilities.invokeLater(() -> {
            CongresistaFrame frame = new CongresistaFrame(service);
            frame.setVisible(true);
        });
    }
```

}

Explicación – ¿Cómo se conecta todo?

- El **adaptador de base de datos** se crea primero y se pasa al servicio.
- El **servicio** es construido con el puerto de salida ya implementado.
- La **GUI** recibe el servicio como puerto de entrada.
- De esta forma se construye el “hexágono” completo.
- No hay dependencias circulares:
    - Dominio no depende de BD
    - Dominio no depende de GUI
    - GUI no sabe nada de MySQL

Script SQL usado en la Aplicación

Copia y pega tu script aquí. Lo puedes generar desde Workbench con:
Server → Data Export → Export to Self-Contained File
o
Clic derecho a la BD → Table Data Export Wizard → Export as SQL Script.

Si quieres, te lo genero si me mandas el nombre de la BD.

Fragmentos de Código por Capa

Por qué es de la capa dominio:

- No hay JDBC
- No hay SQL
- No hay frameworks
- Solo lógica pura

Capa Dominio

La capa de dominio contiene los modelos puros del negocio.
En este proyecto, la entidad **Congresista** representa la información fundamental del caso de uso y está completamente desacoplada de detalles técnicos.public class Congresista {

package com.congreso.domain;

```
public class Congresista {
    private int id;
    private String nombre;
    private String apellido;
    private String email;
    private String pais;

    public Congresista() {}

    public Congresista(int id, String nombre, String apellido, String email, String pais) {
        this.id = id;
        this.nombre = nombre;
        this.apellido = apellido;
        this.email = email;
        this.pais = pais;
    }

    // Getters y Setters
}
```

Universidad de Cartagena
Fundada en 1827

Justificación

- El dominio representa el núcleo del negocio sin depender de frameworks, bases de datos o interfaces gráficas.
- La clase es completamente independiente y puede ser utilizada por cualquier tecnología.
- Cumple los principios de Clean Architecture y Hexagonal Architecture: el núcleo no depende de detalles externos.

Puertos (Interfaces)

Los puertos definen **contratos**, no implementaciones.
Actúan como la frontera entre el dominio y los adaptadores.

4.2.1 Puerto de Salida – CongresistaRepository

Este puerto define las operaciones necesarias para acceder a la base de datos: crear, actualizar, eliminar, buscar y listar congresistas.
Este contrato permite que el dominio no dependa directamente de MySQL ni de ninguna tecnología de almacenamiento.

```
package com.congreso.ports;


import com.congreso.domain.Congresista;

import java.util.List;


public interface CongresistaRepository {

   void crear(Congresista c);

   void actualizar(Congresista c);

   void eliminar(int id);

   Congresista buscarPorId(int id);

   List<Congresista> listar();

}
```

Puerto de Entrada – CongresistaServicePort

Este puerto define las operaciones que la capa de aplicación ofrecerá a los adaptadores de entrada (por ejemplo, la GUI).

```
package com.congreso.ports;


import com.congreso.domain.Congresista;

import java.util.List;


public interface CongresistaServicePort {

   void crear(Congresista c);

   void actualizar(Congresista c);

   void eliminar(int id);

   Congresista buscarPorId(int id);

   List<Congresista> listar();
```

Justificación

- Los puertos definen qué necesita la aplicación y cómo se comunica con el exterior.
- No tienen código SQL ni implementaciones concretas.
- Permiten intercambiar infraestructura sin modificar el dominio.
- Son fundamentales para cumplir el principio de inversión de dependencias.

Universidad de Cartagena
Fundada en 1827

Capa de Aplicación – Servicio

```java
package com.congreso.application;

import com.congreso.domain.Congresista;
import com.congreso.ports.CongresistaRepository;
import com.congreso.ports.CongresistaServicePort;
import java.util.List;

public class CongresistaService implements CongresistaServicePort {

    private final CongresistaRepository repository;

    public CongresistaService(CongresistaRepository repository) {
        this.repository = repository;
    }

    @Override
    public void crear(Congresista c) {
        // Aquí podríamos agregar validaciones antes de guardar
        repository.crear(c);
    }

    @Override
    public void actualizar(Congresista c) {
        repository.actualizar(c);
    }

    @Override
    public void eliminar(int id) {
        repository.eliminar(id);
    }

    @Override
    public Congresista buscarPorId(int id) {
        return repository.buscarPorId(id);
    }

    @Override
    public List<Congresista> listar() {
        return repository.listar();
    }
}
```

Justificación

- Orquesta los casos de uso CRUDL.
- Aplica reglas de negocio antes de llamar a los adaptadores.
- No depende de SQL ni JDBC: depende únicamente de una **interfaz**.
- Permite intercambiar MySQL por otra BD sin cambiar el servicio.

¿Por qué la GUI pertenece a la capa de Adaptadores?

- No contiene lógica de negocio
- No ejecuta SQL
- No accede directamente a la base de datos
- Se comunica exclusivamente con el servicio mediante el **puerto de entrada**
- Es totalmente reemplazable por otra interfaz (Web, móvil, REST, etc.)

Universidad de Cartagena
Fundada en 1827

Adaptador de Base de Datos – JDBC

Código del Adaptador

```
package com.congreso.adapters.db;

import com.congreso.domain.Congresista;
import com.congreso.ports.CongresistaRepository;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class MySQLCongresistaRepository implements CongresistaRepository {

    private final String url = "jdbc:mysql://localhost:3306/congresodb";
    private final String user = "root";
    private final String pass = "";

    private Connection getConnection() throws SQLException {
        return DriverManager.getConnection(url, user, pass);
    }

    @Override
    public void crear(Congresista c) {
        String sql = "INSERT INTO congresistas(nombre, apellido, email, pais)
VALUES(?,?,?,?)";
        try (Connection conn = getConnection();
             PreparedStatement ps = conn.prepareStatement(sql)) {

            ps.setString(1, c.getNombre());
            ps.setString(2, c.getApellido());
```

```
            ps.setString(3, c.getEmail());
            ps.setString(4, c.getPais());
            ps.executeUpdate();

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

    @Override
    public void actualizar(Congresista c) {
        String sql = "UPDATE congresistas SET nombre=?, apellido=?, email=?, pais=?
WHERE id=?";
        try (Connection conn = getConnection();
             PreparedStatement ps = conn.prepareStatement(sql)) {

            ps.setString(1, c.getNombre());
            ps.setString(2, c.getApellido());
            ps.setString(3, c.getEmail());
            ps.setString(4, c.getPais());
            ps.setInt(5, c.getId());
            ps.executeUpdate();

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

    @Override
```

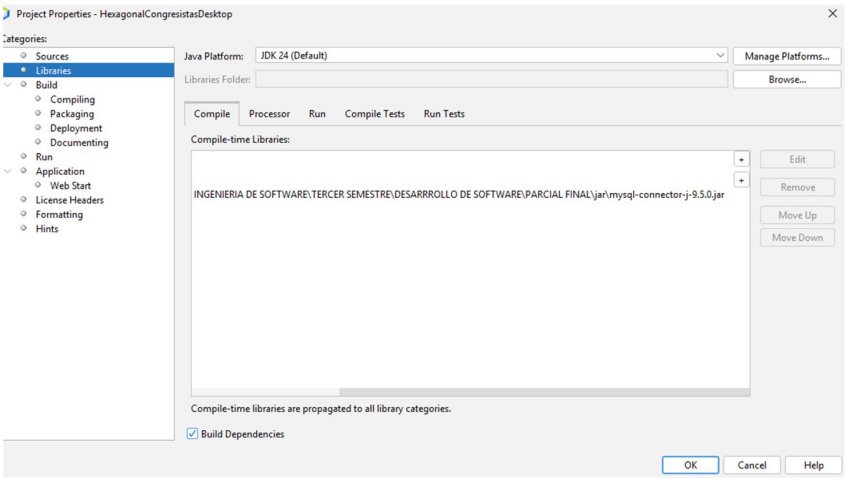
```
public void eliminar(int id) {
    String sql = "DELETE FROM congresistas WHERE id=?";
    try (Connection conn = getConnection();
         PreparedStatement ps = conn.prepareStatement(sql)) {

        ps.setInt(1, id);
        ps.executeUpdate();

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

@Override
public Congresista buscarPorId(int id) {
    String sql = "SELECT * FROM congresistas WHERE id=?";
    Congresista c = null;

    try (Connection conn = getConnection();
         PreparedStatement ps = conn.prepareStatement(sql)) {

        ps.setInt(1, id);
        ResultSet rs = ps.executeQuery();

        if (rs.next()) {
            c = new Congresista(
                    rs.getInt("id"),
                    rs.getString("nombre"),
```

```
                rs.getString("apellido"),
                rs.getString("email"),
                rs.getString("pais")
            );
        }

    } catch (SQLException e) {
        e.printStackTrace();
    }
    return c;
}

@Override
public List<Congresista> listar() {
    String sql = "SELECT * FROM congresistas";
    List<Congresista> lista = new ArrayList<>();

    try (Connection conn = getConnection();
         PreparedStatement ps = conn.prepareStatement(sql);
         ResultSet rs = ps.executeQuery()) {

        while (rs.next()) {
            lista.add(new Congresista(
                rs.getInt("id"),
                rs.getString("nombre"),
                rs.getString("apellido"),
                rs.getString("email"),
                rs.getString("pais")
```

```
                ));
            }

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

Justificación

- Implementa un puerto de salida.
- Contiene SQL real: INSERT, SELECT, UPDATE, DELETE.
- Utiliza JDBC, conexiones y ResultSet.
- Es un detalle de infraestructura totalmente reemplazable.
- No afecta el dominio ni la capa de aplicación.



- Explicación de por qué pertenece a adaptadores secundarios.
- Que contiene SQL REAL y JDBC.

Universidad de Cartagena
Fundada en 1827

Adaptador de Entrada – GUI (Swing)

```java
package com.congreso.adapters.gui;

import com.congreso.application.CongresistaService;
import com.congreso.domain.Congresista;
import java.util.List;
import javax.swing.*;
import javax.swing.table.DefaultTableModel;

public class CongresistaFrame extends JFrame {

    private final CongresistaService service;
    private JTextField txtId, txtNombre, txtApellido, txtEmail, txtPais;
    private JTable table;
    private DefaultTableModel tableModel;

    public CongresistaFrame(CongresistaService service) {
        this.service = service;
        initComponents();
        cargarTabla();
    }

    private void initComponents() {
        setTitle("Gestión de Congresistas - Arquitectura Hexagonal");
        setSize(850, 600);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(null);

        JLabel lblId = new JLabel("ID:");
        JLabel lblNombre = new JLabel("Nombre:");
        JLabel lblApellido = new JLabel("Apellido:");
        JLabel lblEmail = new JLabel("Email:");
        JLabel lblPais = new JLabel("Pais:");

        txtId = new JTextField();
        txtNombre = new JTextField();
        txtApellido = new JTextField();
        txtEmail = new JTextField();
        txtPais = new JTextField();

        JButton btnCrear = new JButton("Crear");
        JButton btnActualizar = new JButton("Actualizar");
        JButton btnEliminar = new JButton("Eliminar");
        JButton btnBuscar = new JButton("Buscar");
        JButton btnListar = new JButton("Listar");
```

```
        lblId.setBounds(20, 20, 100, 25); txtId.setBounds(120, 20, 150, 25);
        lblNombre.setBounds(20, 55, 100, 25); txtNombre.setBounds(120, 55, 150, 25);
        lblApellido.setBounds(20, 90, 100, 25); txtApellido.setBounds(120, 90, 150, 25);
        lblEmail.setBounds(20, 125, 100, 25); txtEmail.setBounds(120, 125, 150, 25);
        lblPais.setBounds(20, 160, 100, 25); txtPais.setBounds(120, 160, 150, 25);

        btnCrear.setBounds(300, 20, 100, 25);
        btnActualizar.setBounds(300, 55, 100, 25);
        btnEliminar.setBounds(300, 90, 100, 25);
        btnBuscar.setBounds(300, 125, 100, 25);
        btnListar.setBounds(300, 160, 100, 25);

        add(lblId); add(txtId);
        add(lblNombre); add(txtNombre);
        add(lblApellido); add(txtApellido);
        add(lblEmail); add(txtEmail);
        add(lblPais); add(txtPais);

        add(btnCrear); add(btnActualizar); add(btnEliminar); add(btnBuscar); add(btnListar);

        tableModel = new DefaultTableModel(new Object[]{"ID", "Nombre", "Apellido", "Email", "País"}, 0);
        table = new JTable(tableModel);
        JScrollPane scroll = new JScrollPane(table);
        scroll.setBounds(20, 230, 780, 300);
        add(scroll);

        btnCrear.addActionListener(e -> crear());
        btnActualizar.addActionListener(e -> actualizar());
        btnEliminar.addActionListener(e -> eliminar());
        btnBuscar.addActionListener(e -> buscar());
        btnListar.addActionListener(e -> cargarTabla());
    }

    private void crear() {
        Congresista c = new Congresista(
                0,
                txtNombre.getText(),
                txtApellido.getText(),
                txtEmail.getText(),
                txtPais.getText()
        );
        service.crear(c);
        cargarTabla();
        limpiarCampos();
    }
```

```
    private void eliminar() {
        int id = Integer.parseInt(txtId.getText());
        service.eliminar(id);
        cargarTabla();
        limpiarCampos();
    }

    private void buscar() {
        int id = Integer.parseInt(txtId.getText());
        Congresista c = service.buscarPorId(id);
        if (c != null) {
            txtNombre.setText(c.getNombre());
            txtApellido.setText(c.getApellido());
            txtEmail.setText(c.getEmail());
            txtPais.setText(c.getPais());
        } else {
            JOptionPane.showMessageDialog(this, "No encontrado");
        }
    }

    private void cargarTabla() {
        tableModel.setRowCount(0);
        List<Congresista> lista = service.listar();
        for (Congresista c : lista) {
            tableModel.addRow(new Object[]{
                    c.getId(),
                    c.getNombre(),
                    c.getApellido(),
                    c.getEmail(),
                    c.getPais()
            });
        }
    }

    private void limpiarCampos() {
        txtId.setText("");
        txtNombre.setText("");
        txtApellido.setText("");
        txtEmail.setText("");
        txtPais.setText("");
    }
```

- Botones
- Tabla
- Interacción con el servicio

Universidad de Cartagena
Fundada en 1827

6 Ensamblaje – Clase Main

```java
package com.congreso.main;

import com.congreso.adapters.db.MySQLCongresistaRepository;
import com.congreso.adapters.gui.CongresistaFrame;
import com.congreso.application.CongresistaService;
import javax.swing.SwingUtilities;

public class MainApp {

    public static void main(String[] args) {

        // Adaptador de base de datos (secundario)
        MySQLCongresistaRepository repo = new MySQLCongresistaRepository();

        // Servicio (capa de aplicación)
        CongresistaService service = new CongresistaService(repo);

        // GUI (adaptador primario)
        SwingUtilities.invokeLater(() -> {
            CongresistaFrame frame = new CongresistaFrame(service);
            frame.setVisible(true);
        });
    }
}
```

- Explicación de cómo se conectan servicio + BD + GUI.

Universidad de Cartagena
Fundada en 1827

Script SQL del Proyecto

Script SQL Utilizado

Pegas esto:

```
CREATE DATABASE congresodb;

USE congresodb;

CREATE TABLE congresistas (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100),
    apellido VARCHAR(100),
    email VARCHAR(150),
    pais VARCHAR(80)
);
```

Universidad de Cartagena
Fundada en 1827

Tecnologías utilizadas

- Java 17
- MySQL 8.0
- JDBC – mysql-connector-j-9.5.0.jar
- NetBeans IDE 15+

Universidad de Cartagena
Fundada en 1827

Conclusiones

La arquitectura Hexagonal permitió separar claramente el dominio de la infraestructura y la interfaz gráfica. Esta separación facilita futuras modificaciones y garantiza un diseño limpio, escalable y fácilmente adaptable a otros entornos (REST, Web, etc.).
El CRUDL se ejecutó correctamente sobre MySQL usando JDBC, validando que el sistema cumple los requerimientos del examen.

Universidad de Cartagena
Fundada en 1827

Bibliografía

- Oracle. *Java JDBC Documentation*.
- MySQL. *MySQL Connector/J Developer Guide*.
- Universidad de Cartagena – Material de clase de Bases de Datos.