

**UNIVERSIDAD DE LAS FUERZAS ARMADAS
ESPE**



NOMBRES: MIDEROS SAMIR, MIRANDA ALISON, MORÁN DAVID,
VIVANCO GABRIEL

NRC: 27837

FECHA: 19/11/2025

Tema:

“Taller CRUD estudiantes”

Análisis y diseño del software

PERIODO 2025-2026

Taller: Creación de un CRUD (ID, Nombres, Edad) aplicando Arquitectura de 3 Capas y Patrón Modelo–Vista–Controlador (MVC)

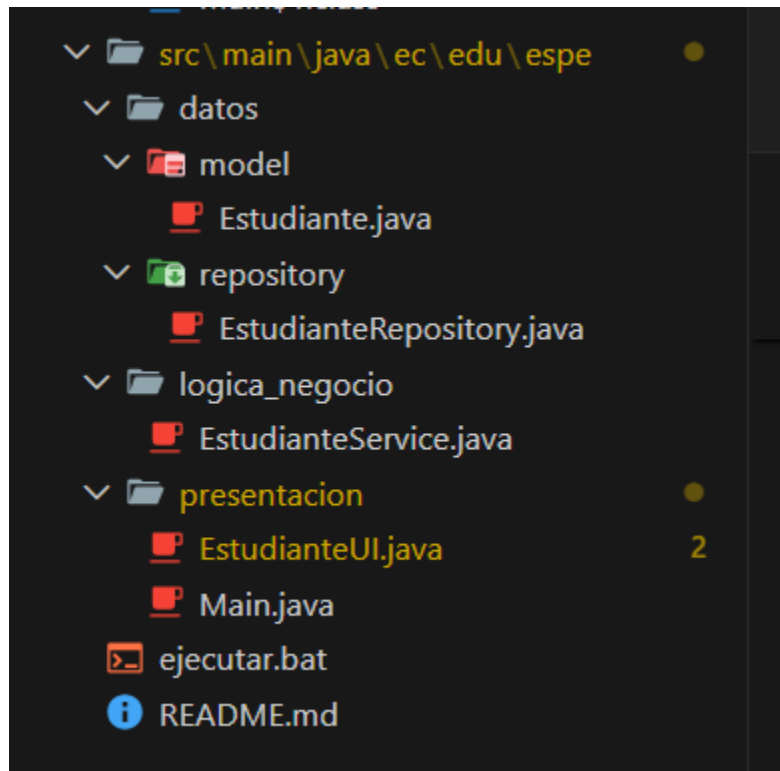
1. Objetivo del Taller

Construir una aplicación CRUD de Estudiante (ID, nombres, edad) utilizando la arquitectura de 3 capas y el patrón de diseño Modelo–Vista–Controlador (MVC) con una interfaz gráfica en Java Swing.

2. Estructura del Proyecto

La aplicación deberá organizarse en la siguiente estructura de paquetes:

```
src/main/java/ec/edu/espe/  
├── datos/  
│   ├── model/  
│   │   └── Estudiante.java  
│   └── repository/  
│       └── EstudianteRepository.java  
├── logica_negocio/  
│   └── EstudianteService.java  
└── presentacion/  
    ├── EstudianteUI.java  
    └── Main.java
```



3. Descripción de las Capas

3.1 Capa Modelo (Model)

Contiene la clase Estudiante.java, que representa el dominio con sus atributos: ID, nombres y edad. No contiene lógica de negocio ni acceso a datos.

```
    * Clase modelo que representa un estudiante
    * Contiene los atributos básicos: ID, nombres y edad
    */
public class Estudiante {
    private int id;
    private String nombres;
    private int edad;

    /**
     * Constructor por defecto
     */
    public Estudiante() {
    }

    /**
     * Constructor con parámetros
     * @param id Identificador único del estudiante
     * @param nombres Nombres completos del estudiante
     * @param edad Edad del estudiante
     */
    public Estudiante(int id, String nombres, int edad) {
        this.id = id;
        this.nombres = nombres;
        this.edad = edad;
    }
}
```

```

// Getters
public int getId() {
    return id;
}

public String getNombres() {
    return nombres;
}

public int getEdad() {
    return edad;
}

```

3.2 Capa de Acceso a Datos (Repository)

EstudianteRepository.java administra las operaciones CRUD utilizando una colección interna (ArrayList) o almacenamiento simple en archivo. Opcionalmente puede implementarse como Singleton.

```

package ec.edu.espe.datos.repository;

import ec.edu.espe.datos.model.Estudiante;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

/**
 * Repositorio para gestionar las operaciones CRUD de estudiantes
 * Utiliza ArrayList para almacenamiento interno
 * Implementa patrón Singleton para una única instancia
 */
public class EstudianteRepository {

    // Instancia única del repositorio (Singleton)
    private static EstudianteRepository instance;

    // Lista interna para almacenar estudiantes
    private List<Estudiante> estudiantes;

```

```

/**
 * Constructor privado para implementar Singleton
 */
private EstudianteRepository() {
    this.estudiantes = new ArrayList<>();
    inicializarDatosPrueba(); // Datos iniciales para pruebas
}

/**
 * Método para obtener la única instancia del repositorio
 */
public static EstudianteRepository getInstance() {
    if (instance == null) {
        instance = new EstudianteRepository();
    }
    return instance;
}

/**
 * Agregar un nuevo estudiante al repositorio
 * @param estudiante El estudiante a agregar
 * @return true si se agregó exitosamente, false si ya existe el
ID
 */
public boolean agregar(Estudiante estudiante) {
    // Verificar que no exista un estudiante con el mismo ID
    if (buscarPorId(estudiante.getId()).isPresent()) {
        return false; // Ya existe un estudiante con ese ID
    }
    return estudiantes.add(estudiante);
}

```

3.3 Capa Lógica de Negocio (Service)

EstudianteService.java aplica reglas de negocio como validaciones (ej., edad > 0, ID no repetido) y delega las operaciones CRUD al repositorio.

```

package ec.edu.espe.logica_negocio;

import ec.edu.espe.datos.model.Estudiante;
import ec.edu.espe.datos.repository.EstudianteRepository;

```

```

import java.util.List;
import java.util.Optional;

/**
 * Servicio de lógica de negocio para estudiantes
 * Contiene las validaciones y reglas de negocio
 * Delega las operaciones CRUD al repositorio
 */
public class EstudianteService {

    private EstudianteRepository estudianteRepository;

    /**
     * Constructor que inicializa el repositorio
     */
    public EstudianteService() {
        this.estudianteRepository =
EstudianteRepository.getInstance();
    }

    /**
     * Agregar un nuevo estudiante con validaciones de negocio
     * @param estudiante El estudiante a agregar
     * @return Resultado de la operación con mensaje
     */
    public ResultadoOperacion agregarEstudiante(Estudiante
estudiante) {
        // Validar datos del estudiante
        ResultadoOperacion validacion =
validarEstudiante(estudiante);
        if (!validacion.isExito()) {
            return validacion;
        }

        // Verificar que el ID no esté duplicado
        if (estudianteRepository.existeId(estudiante.getId())) {
            return new ResultadoOperacion(false, "Error: Ya existe
un estudiante con el ID " + estudiante.getId());

```

```

    }

    // Intentar agregar al repositorio
    boolean agregado = estudianteRepository.agregar(estudiante);
    if (agregado) {
        return new ResultadoOperacion(true, "Estudiante agregado exitosamente");
    } else {
        return new ResultadoOperacion(false, "Error: No se pudo agregar el estudiante");
    }
}

```

3.4 Capa Presentación – Swing (Vista y Controlador)

EstudianteUI.java representa la interfaz gráfica del usuario. Incluye:

- Campos de texto para ID, nombres y edad.

```

public class EstudianteUI extends JFrame {

    // Componentes de la interfaz
    private JTextField txtId;           ← CAMPOS DE TEXTO
    private JTextField txtNombres;
    private JTextField txtEdad;
    private JButton btnGuardar;        ← BOTONES CRUD
    private JButton btnEditar;
    private JButton btnEliminar;
    private JButton btnLimpiar;
    private JButton btnNuevo;
    private JTable tblEstudiantes;    ← JTABLE COMO GRID
    private DefaultTableModel modeloTabla;

    // Servicio de lógica de negocio
    private EstudianteService estudianteService; ← ENVÍA SOLICITUDES AL SERVICE
}

```

- Botones para las acciones CRUD

```

// ID
panel.add(new JLabel("ID:"), gbc);
txtId = new JTextField(10);          ← CAMPO ID

// Nombres
panel.add(new JLabel("Nombres:"), gbc);
txtNombres = new JTextField(20);     ← CAMPO NOMBRES

// Edad
panel.add(new JLabel("Edad:"), gbc);
txtEdad = new JTextField(10);        ← CAMPO EDAD

// Panel de botones
btnNuevo = new JButton("Nuevo");    ← BOTONES CRUD
btnGuardar = new JButton("Guardar");
btnEditar = new JButton("Editar");
btnEliminar = new JButton("Eliminar");

```

- Una JTable que actúa como 'grid' para mostrar los estudiantes.

```
// Crear modelo de tabla
String[] columnas = {"ID", "Nombres", "Edad"};    ← GRID DE ESTUDIANTES
modeloTabla = new DefaultTableModel(columnas, 0);

// Crear tabla
tblEstudiantes = new JTable(modeloTabla);          ← JTABLE COMO GRID
```

- La UI envía las solicitudes al Service y actualiza la vista según resultados.

```
// Llamar al servicio según el modo                ← ENVÍA AL SERVICE
ResultadoOperacion resultado;
if (modoEdicion) {
    resultado = estudianteService.editarEstudiante(estudiante);
} else {
    resultado = estudianteService.agregarEstudiante(estudiante);
}

// Mostrar resultado                                ← ACTUALIZA VISTA
if (resultado.isExito()) {
    cargarDatos();                                   ← ACTUALIZA SEGÚN RESULTADOS
    limpiarFormulario();
}
```

4. Flujo de Interacción MVC + 3 Capas

- El usuario interactúa con EstudianteUI.

```
// Evento del botón Guardar
btnGuardar.addActionListener(e -> guardarEstudiante()); ← 1. USUARIO INTERACTÚA

// Evento del botón Editar
btnEditar.addActionListener(e -> {
    if (tblEstudiantes.getSelectedRow() != -1) {
        modoEdicion = true;
        establecerEstadoEdicion();
    }
});
```

- La UI llama a EstudianteService.

```
// Crear objeto estudiante
Estudiante estudiante = new Estudiante(id, nombres, edad);

// Llamar al servicio según el modo                ← 2. UI LLAMA AL SERVICE
ResultadoOperacion resultado;
if (modoEdicion) {
    resultado = estudianteService.editarEstudiante(estudiante);
} else {
    resultado = estudianteService.agregarEstudiante(estudiante);
}
```

- El Service valida datos y llama a EstudianteRepository.

```
public ResultadoOperacion agregarEstudiante(Estudiante estudiante) {
    // Validar datos del estudiante ← 3. SERVICE VALIDA DATOS
    ResultadoOperacion validacion = validarEstudiante(estudiante);
    if (!validacion.isExito()) {
        return validacion;
    }

    // Verificar que el ID no esté duplicado
    if (estudianteRepository.existeId(estudiante.getId())) {
        return new ResultadoOperacion(false, "Error: Ya existe...");
    }

    // Intentar agregar al repositorio ← 3. SERVICE LLAMA AL REPO
    boolean agregado = estudianteRepository.agregar(estudiante);
}
```

- El Repository ejecuta las operaciones CRUD y devuelve resultados.

```
public boolean agregar(Estudiante estudiante) { ← 4. REPOSITORY EJECUTA CI
    // Verificar que no exista un estudiante con el mismo ID
    if (buscarPorId(estudiante.getId()).isPresent()) {
        return false; // Ya existe un estudiante con ese ID
    }
    return estudiantes.add(estudiante); ← 4. DEVUELVE RESULTADOS
}
```

- La UI actualiza el formulario y el grid.

```
// Mostrar resultado
if (resultado.isExito()) {
    JOptionPane.showMessageDialog(this,
        resultado.getMensaje(),
        "Éxito",
        JOptionPane.INFORMATION_MESSAGE);

    cargarDatos(); ← 5. UI ACTUALIZA EL GRI
    limpiarFormulario(); ← 5. UI ACTUALIZA EL FORI
    modoEdicion = false;
    establecerEstadoInicial();
}
```

CAPTURA DE FUNCIONAMIENTO DEL PROGRAMA

Gestión de Estudiantes - CRUD con Arquitectura 3 Capas

Datos del Estudiante

ID:

Nombres:

Edad:

Nuevo

Guardar

Editar

Eliminar

Limpiar

Lista de Estudiantes

ID	Nombres	Edad
1	Juan Pérez	20
2	María González	22
3	Carlos López	19

Total de estudiantes: 3

Agregar un nuevo estudiante:

Datos del Estudiante

ID:

4

Nombres:

Gabriel Vivanco

Edad:

22

Nuevo

Guardar

Editar

Eliminar

Limpiar

Lista de Estudiantes

ID	Nombres	Edad
1	Juan Pérez	20
2	María González	22
3	Carlos López	19
4	Gabriel Vivanco	22

Editar un estudiante:

Gestión de Estudiantes - CRUD con Arquitectura 3 Capas

Datos del Estudiante

ID: 3

Nombres: Samir Mideros

Edad: 22

Nuevo **Guardar** **Editar** **Eliminar** **Limpiar**

Lista de Estudiantes

ID	Nombres	Edad
1	Juan Pérez	20
2	María González	22
3	Samir Mideros	22
4	Gabriel Vivanco	22

Eliminar un estudiante:

Gestión de Estudiantes - CRUD con Arquitectura 3 Capas

Datos del Estudiante

ID: 3

Nombres: Samir Mideros

Edad: 22

Nuevo **Guardar** **Editar** **Eliminar** **Limpiar**

Lista de Estudiantes

ID	Nombres	Edad
1	Juan P	20
2	María	22
3	Samir	22
4	Gabrie	22

Confirmar Eliminación

¿Está seguro de que desea eliminar este estudiante?

Yes **No**

Gestión de Estudiantes - CRUD con Arquitectura 3 Capas

Datos del Estudiante

ID:

Nombres:

Edad:

Lista de Estudiantes

ID	Nombres	Edad
1	Juan Pérez	20
2	María González	22
4	Gabriel Vivanco	22

- Rúbrica de Evaluación (20 puntos)

Criterio	Descripción	Puntaje
Estructura del proyecto	Implementa correctamente la arquitectura de 3 capas.	0-5
Modelo, Servicio y Repositorio	Clases correctamente implementadas con separación de responsabilidades.	0-5
Interfaz gráfica Swing	Formulario funcional y tabla desplegando datos.	0-5
Aplicación del patrón MVC	La vista no contiene lógica de negocio ni acceso a datos.	0-5