

Taller: Creación de un CRUD

Aplicando Arquitectura de 3 Capas y Patrón
Modelo-Vista-Controlador (MVC)

Integrantes:

Dennison Chalacan
Jeffrey Manobanda
Jhordy Marcillo

Estructura del Proyecto:

Aplicación CRUD de Estudiante (ID, nombres, edad)

Tecnologías:

Java Swing, Arquitectura 3 Capas, Patrón MVC

Fecha: 20 de noviembre de 2025

Índice

1. Objetivo del Taller	2
2. Estructura del Proyecto	2
3. Descripción de las Capas	3
3.1. Capa Modelo (Model)	3
3.2. Capa de Acceso a Datos (Repository)	3
3.3. Capa Lógica de Negocio (Service)	3
3.4. Capa Presentación – Swing (Vista y Controlador)	3
4. Flujo de Interacción MVC + 3 Capas	3
5. Actividades del Taller	4
5.1. Actividad 1: Crear el paquete datos/model	4
5.2. Actividad 2: Implementar datos/repository	4
5.3. Actividad 3: Crear logica_negocio/EstudianteService.java	5
5.4. Actividad 4: Crear presentacion/EstudianteUI.java en Swing	6
6. Interfaz Gráfica del CRUD	8

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
```

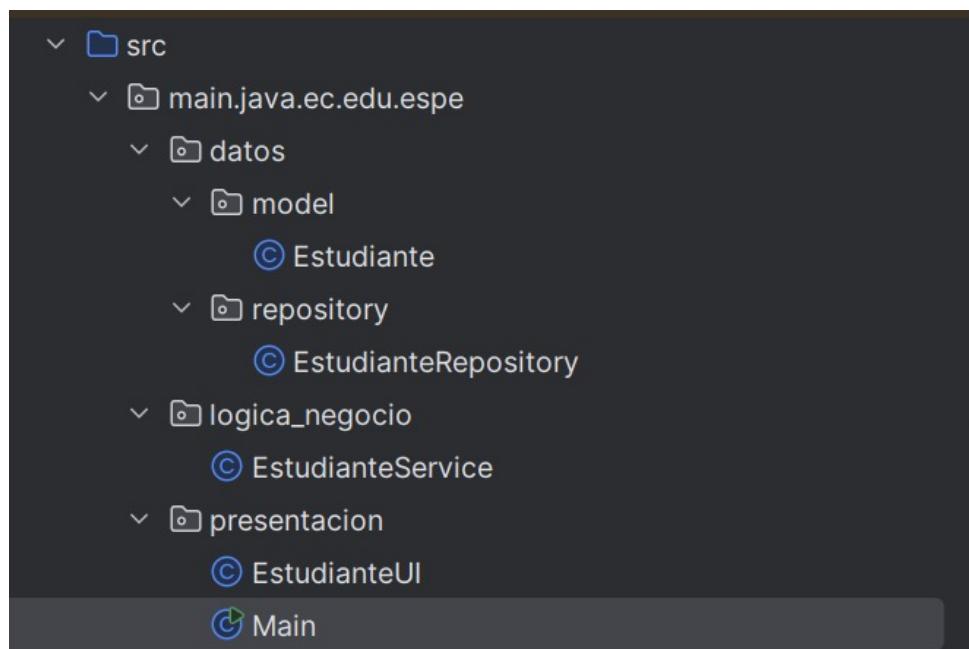


Figura 1: Estructura del proyecto

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.

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.

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.

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.
- Botones para las acciones CRUD.
- Una `JTable` que actúa como 'grid' para mostrar los estudiantes.

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

4. Flujo de Interacción MVC + 3 Capas

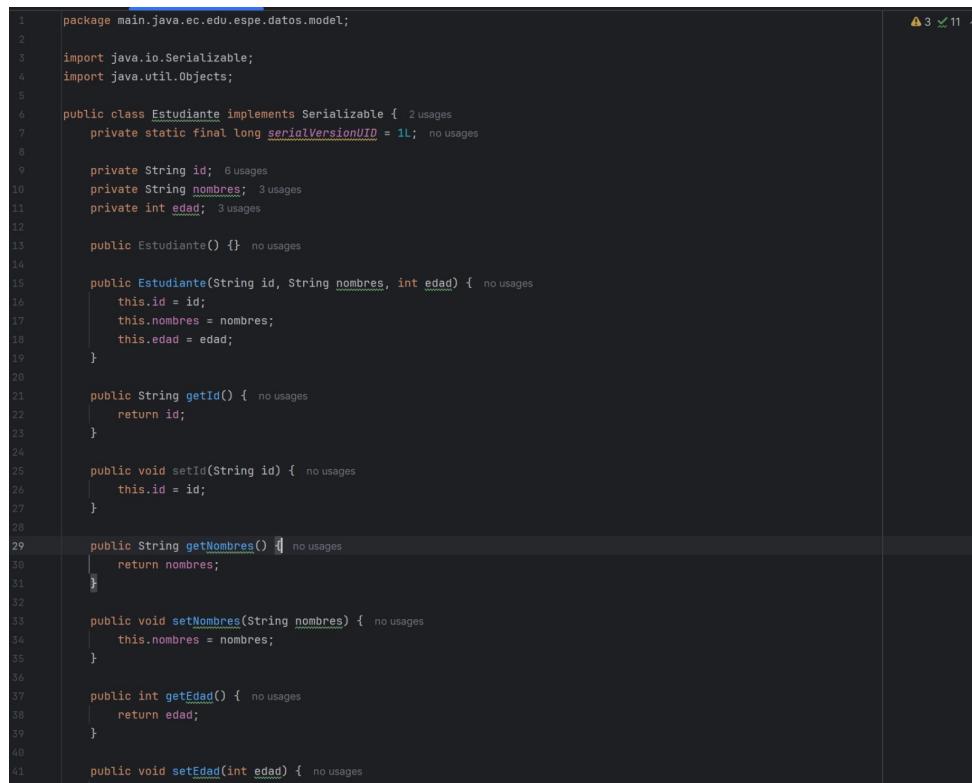
1. El usuario interactúa con `EstudianteUI`.
2. La UI llama a `EstudianteService`.
3. El Service valida datos y llama a `EstudianteRepository`.
4. El Repository ejecuta las operaciones CRUD y devuelve resultados.
5. La UI actualiza el formulario y el grid.

5. Actividades del Taller

5.1. Actividad 1: Crear el paquete datos/model

Crear la clase `Estudiante.java` con:

- Atributos: id, nombres, edad
- Constructor, getters y setters.



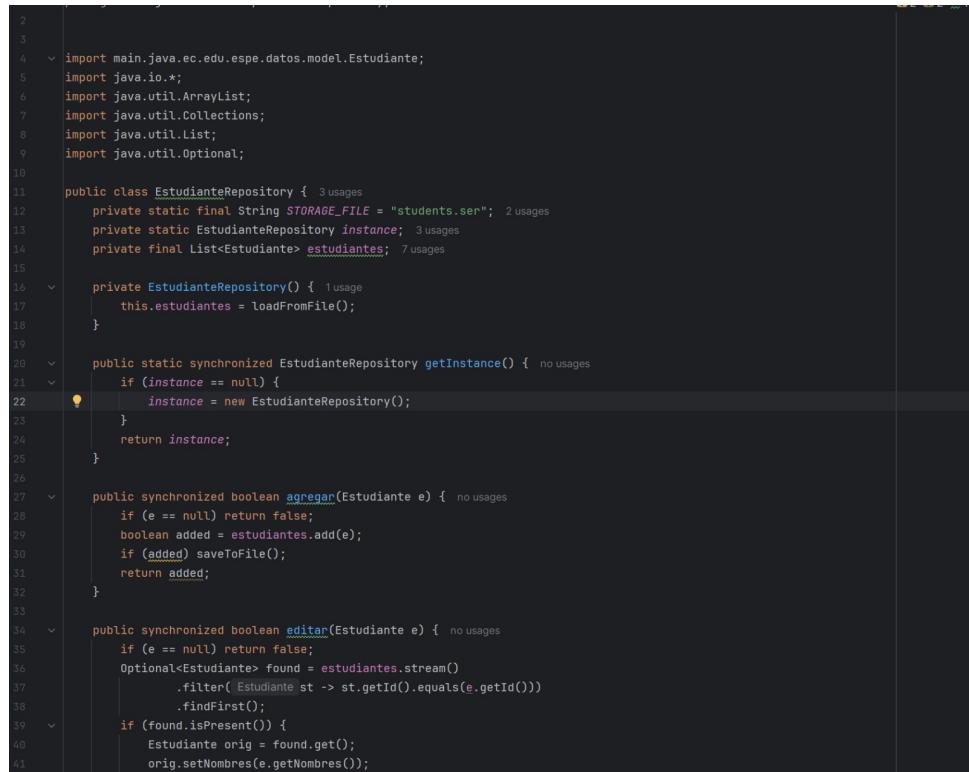
```
1 package main.java.ec.edu.espe.datos.model;
2
3 import java.io.Serializable;
4 import java.util.Objects;
5
6 public class Estudiante implements Serializable {
7     private static final long serialVersionUID = 1L;
8
9     private String id;
10    private String nombres;
11    private int edad;
12
13    public Estudiante() {}
14
15    public Estudiante(String id, String nombres, int edad) {
16        this.id = id;
17        this.nombres = nombres;
18        this.edad = edad;
19    }
20
21    public String getId() {
22        return id;
23    }
24
25    public void setId(String id) {
26        this.id = id;
27    }
28
29    public String getNombres() {
30        return nombres;
31    }
32
33    public void setNombres(String nombres) {
34        this.nombres = nombres;
35    }
36
37    public int getEdad() {
38        return edad;
39    }
40
41    public void setEdad(int edad) {
```

Figura 2: Código de la clase `Estudiante.java`

5.2. Actividad 2: Implementar datos/repository

Implementar `EstudianteRepository.java`:

- Crear una lista interna de estudiantes.
- Implementar: `agregar()`, `editar()`, `eliminar()`, `listar()`.

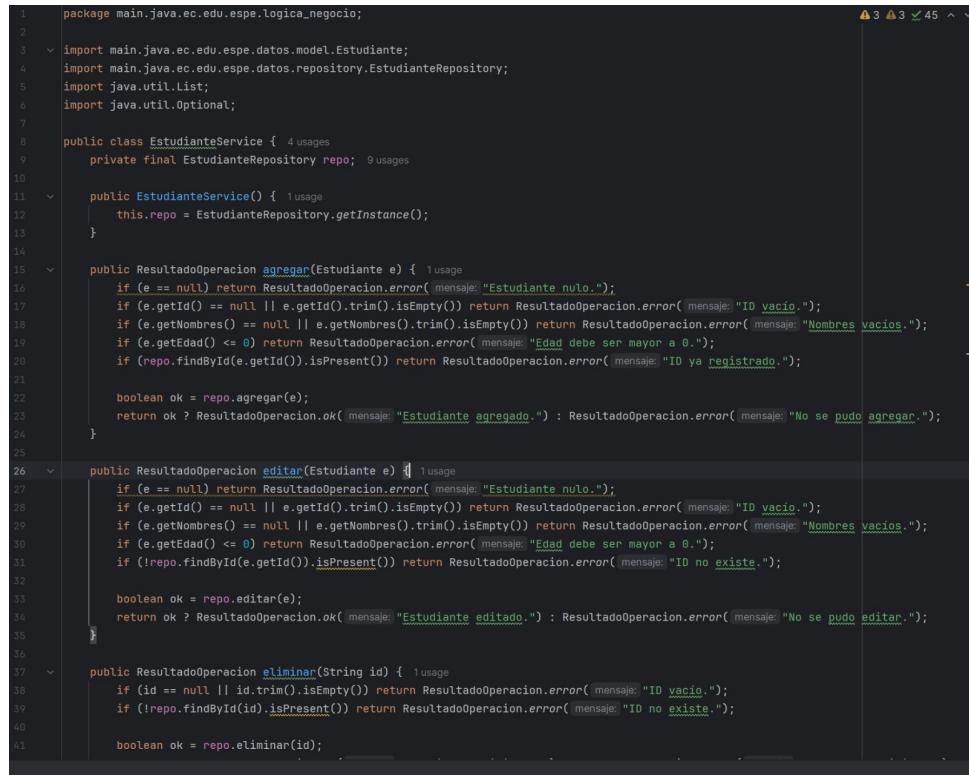


```
2
3
4  import main.java.ec.edu.espe.datos.model.Estudiante;
5  import java.io.*;
6  import java.util.ArrayList;
7  import java.util.Collections;
8  import java.util.List;
9  import java.util.Optional;
10
11 public class EstudianteRepository {
12     private static final String STORAGE_FILE = "students.ser";
13     private static EstudianteRepository instance;
14     private final List<Estudiante> estudiantes;
15
16     private EstudianteRepository() {
17         this.estudiantes = loadFromFile();
18     }
19
20     public static synchronized EstudianteRepository getInstance() {
21         if (instance == null) {
22             instance = new EstudianteRepository();
23         }
24         return instance;
25     }
26
27     public synchronized boolean agregar(Estudiante e) {
28         if (e == null) return false;
29         boolean added = estudiantes.add(e);
30         if (added) saveToFile();
31         return added;
32     }
33
34     public synchronized boolean editar(Estudiante e) {
35         if (e == null) return false;
36         Optional<Estudiante> found = estudiantes.stream()
37             .filter( Estudiante st -> st.getId().equals(e.getId()))
38             .findFirst();
39
40         if (found.isPresent()) {
41             Estudiante orig = found.get();
42             orig.setNombres(e.getNombres());
43         }
44     }
45
46     private void saveToFile() {
47         try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(STORAGE_FILE))) {
48             oos.writeObject(estudiantes);
49         } catch (IOException ex) {
50             ex.printStackTrace();
51         }
52     }
53
54     private List<Estudiante> loadFromFile() {
55         List<Estudiante> estudiantes = new ArrayList();
56         try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(STORAGE_FILE))) {
57             estudiantes = (List<Estudiante>) ois.readObject();
58         } catch (IOException | ClassNotFoundException ex) {
59             ex.printStackTrace();
60         }
61         return estudiantes;
62     }
63 }
```

Figura 3: Código del EstudianteRepository.java

5.3. Actividad 3: Crear logica_negocio/EstudianteService.java

- Validar reglas del negocio.
- Delegar todas las operaciones al Repository.



```

1 package main.java.ec.edu.espe.logica_negocio;
2
3 import main.java.ec.edu.espe.datos.modelo.Estudiante;
4 import main.java.ec.edu.espe.datos.repository.EstudanteRepository;
5 import java.util.List;
6 import java.util.Optional;
7
8 public class EstudianteService {
9     private final EstudianteRepository repo; 9 usages
10
11     public EstudianteService() { 1 usage
12         this.repo = EstudianteRepository.getInstance();
13     }
14
15     public ResultadoOperacion agregar(Estudiante e) { 1 usage
16         if (e == null) return ResultadoOperacion.error( mensaje: "Estudiante nulo.");
17         if (e.getId() == null || e.getId().trim().isEmpty()) return ResultadoOperacion.error( mensaje: "ID vacio.");
18         if (e.getNombres() == null || e.getNombres().trim().isEmpty()) return ResultadoOperacion.error( mensaje: "Nombres vacios.");
19         if (e.getEdad() <= 0) return ResultadoOperacion.error( mensaje: "Edad debe ser mayor a 0.");
20         if (!repo.findById(e.getId()).isPresent()) return ResultadoOperacion.error( mensaje: "ID ya registrado.");
21
22         boolean ok = repo.agregar(e);
23         return ok ? ResultadoOperacion.ok( mensaje: "Estudiante agregado.") : ResultadoOperacion.error( mensaje: "No se pudo agregar.");
24     }
25
26     public ResultadoOperacion editar(Estudiante e) { 1 usage
27         if (e == null) return ResultadoOperacion.error( mensaje: "Estudiante nulo.");
28         if (e.getId() == null || e.getId().trim().isEmpty()) return ResultadoOperacion.error( mensaje: "ID vacio.");
29         if (e.getNombres() == null || e.getNombres().trim().isEmpty()) return ResultadoOperacion.error( mensaje: "Nombres vacios.");
30         if (e.getEdad() <= 0) return ResultadoOperacion.error( mensaje: "Edad debe ser mayor a 0.");
31         if (!repo.findById(e.getId()).isPresent()) return ResultadoOperacion.error( mensaje: "ID no existe.");
32
33         boolean ok = repo.editar(e);
34         return ok ? ResultadoOperacion.ok( mensaje: "Estudiante editado.") : ResultadoOperacion.error( mensaje: "No se pudo editar.");
35     }
36
37     public ResultadoOperacion eliminar(String id) { 1 usage
38         if (id == null || id.trim().isEmpty()) return ResultadoOperacion.error( mensaje: "ID vacio.");
39         if (!repo.findById(id).isPresent()) return ResultadoOperacion.error( mensaje: "ID no existe.");
40
41         boolean ok = repo.eliminar(id);
42     }
43 }

```

Figura 4: Código del EstudianteService.java

5.4. Actividad 4: Crear presentacion/EstudianteUI.java en Swing

- Construir el formulario (ID, nombres, edad).
- Implementar botón Guardar → llama al Service.
- Actualizar la JTable con los datos.

```
1 package main.java.ec.edu.espe.presentacion;
2
3 import main.java.ec.edu.espe.datos.model.Estudiante;
4 import main.java.ec.edu.espe.logica_negocio.EstudianteService;
5 import main.java.ec.edu.espe.logica_negocio.EstudianteService.ResultadoOperacion;
6
7 import javax.swing.*;
8 import javax.swing.table.DefaultTableModel;
9 import java.awt.*;
10 import java.awt.event.MouseAdapter;
11 import java.awt.event.MouseEvent;
12
13 public class EstudianteUI extends JFrame { no usages
14     private final EstudianteService service; 5 usages
15
16     private JTextField tfId; 11 usages
17     private JTextField tfNombres; 6 usages
18     private JTextField tfEdad; 6 usages
19
20     private JButton btnGuardar; 3 usages
21     private JButton btnEditar; 3 usages
22     private JButton btnEliminar; 3 usages
23     private JButton btnLimpiar; 3 usages
24
25     private JTable table; 5 usages
26     private DefaultTableModel tableModel; 7 usages
27
28     public EstudianteUI() { no usages
29         service = new EstudianteService();
30         initComponents();
31         cargarTabla();
32     }
33
34     private void initComponents() { 1 usage
35         setTitle("CRUD Estudiantes - 3 capas + MVC");
36         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
37         setSize( width: 700, height: 450 );
38         setLocationRelativeTo(null);
39
40         JLabel lblId = new JLabel( text: "ID:" );
41         JLabel lblNombres = new JLabel( text: "Nombres:" );
```

Figura 5: Interfaz gráfica EstudianteUI.java

6. Interfaz Gráfica del CRUD

The screenshot shows a window titled "CRUD Estudiantes - 3 capas + MVC". On the left, there is a form with fields for "ID" (002), "Nombres" (Dennison), and "Edad" (22). Below the form are four buttons: "Guardar", "Editar", "Eliminar", and "Limpiar". On the right, there is a table with three rows:

ID	Nombres	Edad
001	Jeffrey	22
002	Dennison	22
003	Jhordy	22

Figura 6: Interfaz del CRUD

The screenshot shows a window titled "CRUD Estudiantes - 3 capas + MVC". On the left, there is a form with empty fields for "ID", "Nombres", and "Edad". Below the form are four buttons: "Guardar", "Editar", "Eliminar", and "Limpiar". On the right, there is a table with three rows:

ID	Nombres	Edad
001	Jeffrey	22
002	Dennison	22
003	Jhordy	22

Figura 7: Interfaz del CRUD