Universidad de las Fuerzas Armadas ESPE

Departamento: Ciencias de la Computación

Carrera: Ingeniería de Software

Taller académico Nª: 3

---

1. Información General

- **Asignatura:** Análisis y Diseño de Software
- **Apellidos y nombres de los estudiantes:**
  - Chavez Oscullo Klever Enrique
  - Guacan Rivera Alexander David
  - Trejo Duque Alex Fernando
- **NRC:** 23305
- **Fecha de realización: 12/06/2025**

---

2. Objetivo del Taller y Desarrollo

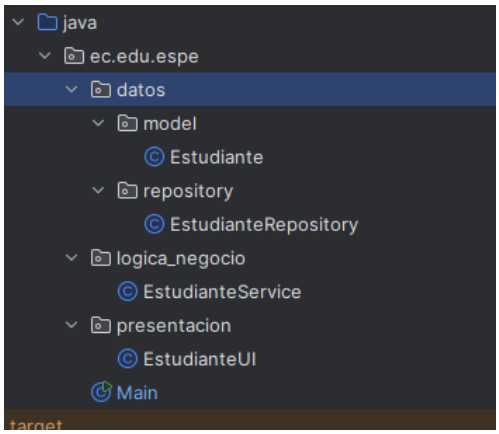
Objetivo del Taller:

Desarrollar en un entorno IDE de Java una aplicación que implemente un CRUD (Crear, Leer, Actualizar, Eliminar) para gestionar datos de estudiantes (id, nombre y edad), utilizando el patrón de arquitectura en 3 capas. Se busca demostrar cómo interactúan las capas entre sí y cómo se ejecuta la lógica del sistema desde la clase principal (Main.java).

Desarrollo:

Para el desarrollo del taller se hizo uso del editor de código IntelliJ IDEA junto con el jdk 21.0.6. El programa va a implementar una arquitectura de 3 capas (datos, logica_negocio, presentacion). A continuación, se presenta la arquitectura:

Arquitectura del proyecto



ec.edu.espe/

|--Main.java // Clase principal para iniciar la aplicación

|--datos/

| |--model/

| |--Estudiante.java // Clase de datos del estudiante.

| |--repository/

| |--EstudianteRepository.java // repositorio para gestion de datos

|

|--logica_negocio/

| |--EstudianteService.java //logica de negocio para manejar estudiantes

|

|--presentacion/

| |--EstudianteUI.java // /Interfaz de usuario para gestionar estudiantes

1. **Capa 1 datos**

   Esta capa se encarga de definir las estructuras de datos y gestionar el acceso a la información del sistema.

   (model/Estudiante.java)

   Representa el modelo de datos del estudiante, encapsulando atributos como identificador único, nombre y edad. Proporciona métodos para acceder y modificar estos atributos, asegurando un manejo controlado de la información, y un método toString que genera una representación textual del objeto

```java
package ec.edu.espe.datos.model;

/**
 * Representa un estudiante con un ID, nombre y edad.
 */
public class Estudiante {
    /**
     * ID único del estudiante.
     */
    private int id;

    /**
     * Nombre del estudiante.
     */
    private String nombre;

    /**
     * Edad del estudiante.
     */
    private int edad;

    /**
     * Constructor para crear una instancia de Estudiante.
     *
     * @param id     ID único del estudiante.
     * @param nombre Nombre completo del estudiante.
     * @param edad   Edad del estudiante.
     */
    public Estudiante(int id, String nombre, int edad) {
        this.id = id;
        this.nombre = nombre;
        this.edad = edad;
    }

    /**
     * Obtiene el ID del estudiante.
     *
     * @return ID del estudiante.
     */
    public int getId() {
        return id;
    }

    /**
     * Actualiza el ID del estudiante.
     *
     * @param id Nuevo ID del estudiante.
     */
    public void setId(int id) {
        this.id = id;
    }

    /**
```

```
     * Obtiene el nombre del estudiante.
     *
     * @return Nombre del estudiante.
     */
    public String getNombre() {
        return nombre;
    }

    /**
     * Actualiza el nombre del estudiante.
     *
     * @param nombre Nuevo nombre del estudiante.
     */
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    /**
     * Obtiene la edad del estudiante.
     *
     * @return Edad del estudiante.
     */
    public int getEdad() {
        return edad;
    }

    /**
     * Actualiza la edad del estudiante.
     *
     * @param edad Nueva edad del estudiante.
     */
    public void setEdad(int edad) {
        this.edad = edad;
    }

    /**
     * Devuelve una representación en forma de cadena del objeto Estudiante.
     *
     * @return Cadena que describe al estudiante con su ID, nombre y edad.
     */
    @Override
    public String toString() {
        return String.format("ID: %d - Nombre: %s - Edad: %d", id, nombre, edad);
    }
}
```

i. Repository/EstudianteRepository.java

Actúa como el componente de acceso a datos, facilitando la interacción con la información de los estudiantes. Se encarga de operaciones como almacenamiento, recuperación y actualización de datos, simulando un mecanismo de persistencia o colección en memoria

```
package ec.edu.espe.datos.repository;

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

import java.util.ArrayList;
import java.util.List;

/**
 * Clase que actúa como repositorio para gestionar las operaciones CRUD
 * relacionadas con los estudiantes.
 */
public class EstudianteRepository {
```

```java
    /**
     * Lista que almacena los objetos de tipo Estudiante.
     */
    private List<Estudiante> estudiantes;

    /**
     * Constructor de la clase EstudianteRepository.
     * Inicializa un repositorio vacío para almacenar estudiantes.
     */
    public EstudianteRepository() {
        this.estudiantes = new ArrayList<>();
    }

    /**
     * Agrega un nuevo estudiante al repositorio.
     *
     * @param estudiante El estudiante que se desea agregar.
     */
    public void agregarEstudiante(Estudiante estudiante) {
        estudiantes.add(estudiante);
    }

    /**
     * Obtiene una lista con todos los estudiantes almacenados.
     *
     * @return Una nueva lista que contiene todos los estudiantes.
     */
    public List<Estudiante> obtenerTodos() {
        return new ArrayList<>(estudiantes);
    }

    /**
     * Busca un estudiante en el repositorio utilizando su ID.
     *
     * @param id El identificador del estudiante que se desea buscar.
     * @return El objeto Estudiante si se encuentra, de lo contrario, retorna
null.
     */
    public Estudiante buscarPorId(int id) {
        return estudiantes.stream()
                .filter(e -> e.getId() == id)
                .findFirst()
                .orElse(null);
    }

    /**
     * Actualiza la información de un estudiante existente en el repositorio.
     * Si el ID del estudiante existe, reemplaza su información.
     *
     * @param estudiante El estudiante con la información actualizada.
     */
    public void actualizarEstudiante(Estudiante estudiante) {
        for (int i = 0; i < estudiantes.size(); i++) {
            if (estudiantes.get(i).getId() == estudiante.getId()) {
                estudiantes.set(i, estudiante);
                break;
            }
        }
```

```
    }

    /**
     * Elimina un estudiante del repositorio utilizando su ID.
     *
     * @param id El identificador del estudiante que se desea eliminar.
     */
    public void eliminarEstudiante(int id) {
        estudiantes.removeIf(e -> e.getId() == id);
    }
}
```

2. **Capa 2: Lógica de Negocio**

   Esta capa contiene la lógica que rige las reglas y procesos del sistema, actuando como intermediaria entre los datos y la presentación. Implementa la lógica de negocio relacionada con la gestión de estudiantes. Se encarga de procesar las operaciones definidas por las reglas del sistema, coordinando las interacciones con la capa de datos para garantizar un comportamiento coherente

   a. EstudianteService.java

```
package ec.edu.espe.logica_negocio;

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

import java.util.List;

/**
 * Clase que contiene la lógica de negocio para la gestión de estudiantes.
 * Esta clase actúa como un intermediario entre la capa de datos y el resto
 * de la aplicación.
 */
public class EstudianteService {
    /**
     * Referencia al repositorio donde se almacenan los datos de los estudiantes.
     */
    private EstudianteRepository repository;

    /**
     * Constructor de la clase EstudianteService.
     * Inicializa la instancia de EstudianteRepository.
     */
    public EstudianteService() {
        this.repository = new EstudianteRepository();
    }

    /**
     * Agrega un nuevo estudiante utilizando su ID, nombre y edad.
     *
     * @param id     Identificador único del estudiante.
     * @param nombre El nombre del estudiante.
     * @param edad   La edad del estudiante.
     */
    public void agregarEstudiante(int id, String nombre, int edad) {
        Estudiante estudiante = new Estudiante(id, nombre, edad);
        repository.agregarEstudiante(estudiante);
    }
```

```java
    /**
     * Obtiene la lista de todos los estudiantes almacenados.
     *
     * @return Una lista de objetos Estudiante.
     */
    public List<Estudiante> obtenerTodos() {
        return repository.obtenerTodos();
    }

    /**
     * Busca un estudiante utilizando su identificador único.
     *
     * @param id Identificador del estudiante que se desea buscar.
     * @return El objeto Estudiante si se encuentra, de lo contrario, retorna
null.
     */
    public Estudiante buscarPorId(int id) {
        return repository.buscarPorId(id);
    }

    /**
     * Actualiza la información de un estudiante utilizando su ID, un nuevo
nombre y edad.
     *
     * @param id     Identificador único del estudiante.
     * @param nombre Nuevo nombre del estudiante.
     * @param edad   Nueva edad del estudiante.
     */
    public void actualizarEstudiante(int id, String nombre, int edad) {
        Estudiante estudiante = new Estudiante(id, nombre, edad);
        repository.actualizarEstudiante(estudiante);
    }

    /**
     * Elimina un estudiante del repositorio utilizando su identificador único.
     *
     * @param id El identificador del estudiante que se desea eliminar.
     */
    public void eliminarEstudiante(int id) {
        repository.eliminarEstudiante(id);
    }
}
```

3. **Capa 3: Presentación**

   Esta capa se ocupa de la interfaz de usuario, proporcionando la interacción directa con el usuario final. Ofrece una interfaz de usuario nos permite realizar operaciones como visualización, creación y modificación de datos de estudiantes

   a. EstudianteUI.java

```java
package ec.edu.espe.presentacion;

import ec.edu.espe.datos.model.Estudiante;
import ec.edu.espe.logica_negocio.EstudianteService;
import javafx.application.Platform;
import javafx.scene.control.Button;
```

```java
import javafx.scene.control.Label;
import javafx.scene.control.TextArea;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;
import javafx.geometry.Insets;
import javafx.scene.layout.VBox;
import javafx.scene.Scene;

/**
 * Clase que representa la interfaz gráfica de usuario (UI) para la gestión de estudiantes.
 * Permite realizar operaciones como agregar, buscar, actualizar, eliminar y listar estudiantes.
 */
public class EstudianteUI {
    private EstudianteService service;
    private TextField idField;
    private TextField nombreField;
    private TextField edadField;
    private TextArea listaEstudiantes;
    private Label mensajeLabel;

    /**
     * Constructor de la clase EstudianteUI.
     * Inicializa el servicio de lógica de negocios para la gestión de estudiantes.
     */
    public EstudianteUI() {
        this.service = new EstudianteService();
    }

    /**
     * Método que configura y muestra la ventana principal de la aplicación.
     *
     * @param primaryStage La ventana principal de la aplicación JavaFX.
     */
    public void mostrarVentana(Stage primaryStage) {
        primaryStage.setTitle("Gestión de Estudiantes");

        // Crear un diseño GridPane para los controles de la interfaz
        GridPane grid = new GridPane();
        grid.setPadding(new Insets(10)); // Márgenes
        grid.setVgap(10); // Espaciado vertical
        grid.setHgap(10); // Espaciado horizontal

        // Campos de entrada
        Label idLabel = new Label("ID:");
        idField = new TextField();
        Label nombreLabel = new Label("Nombre:");
        nombreField = new TextField();
        Label edadLabel = new Label("Edad:");
        edadField = new TextField();

        // Botones
        Button agregarButton = new Button("Agregar");
        Button buscarButton = new Button("Buscar");
        Button actualizarButton = new Button("Actualizar");
```

```
        Button eliminarButton = new Button("Eliminar");
        Button listarButton = new Button("Listar Todos");
        Button salirButton = new Button("Salir");

        // Área de texto para mostrar la lista de estudiantes
        listaEstudiantes = new TextArea();
        listaEstudiantes.setEditable(false); // Solo lectura
        listaEstudiantes.setPrefHeight(200); // Altura preferida

        // Etiqueta para mostrar mensajes al usuario
        mensajeLabel = new Label();

        // Agregar componentes al GridPane
        grid.add(idLabel, 0, 0);
        grid.add(idField, 1, 0);
        grid.add(nombreLabel, 0, 1);
        grid.add(nombreField, 1, 1);
        grid.add(edadLabel, 0, 2);
        grid.add(edadField, 1, 2);
        grid.add(agregarButton, 0, 3);
        grid.add(buscarButton, 1, 3);
        grid.add(actualizarButton, 0, 4);
        grid.add(eliminarButton, 1, 4);
        grid.add(listarButton, 0, 5);
        grid.add(salirButton, 1, 5);
        grid.add(listaEstudiantes, 0, 6, 2, 1); // Área de texto ocupa dos
columnas
        grid.add(mensajeLabel, 0, 7, 2, 1);

        // Configurar las acciones de los botones
        agregarButton.setOnAction(e -> agregarEstudiante());
        buscarButton.setOnAction(e -> buscarEstudiante());
        actualizarButton.setOnAction(e -> actualizarEstudiante());
        eliminarButton.setOnAction(e -> eliminarEstudiante());
        listarButton.setOnAction(e -> mostrarEstudiantes());
        salirButton.setOnAction(e -> Platform.exit()); // Cierra la aplicación

        // Configurar la escena y mostrar la ventana
        VBox root = new VBox(grid); // Contenedor principal
        Scene scene = new Scene(root, 400, 500); // Crear escena
        primaryStage.setScene(scene); // Asignar escena a la ventana
        primaryStage.show(); // Mostrar la ventana
    }

    /**
     * Agrega un nuevo estudiante basado en los datos ingresados por el usuario.
     * Verifica que los campos ID y edad sean numéricos.
     */
    private void agregarEstudiante() {
        try {
            int id = Integer.parseInt(idField.getText());
            String nombre = nombreField.getText();
            int edad = Integer.parseInt(edadField.getText());
            service.agregarEstudiante(id, nombre, edad); // Agregar estudiante al
servicio
            mensajeLabel.setText("Estudiante agregado con éxito.");
            limpiarCampos();
            mostrarEstudiantes(); // Refrescar la lista de estudiantes
```

```
        } catch (NumberFormatException e) {
            mensajeLabel.setText("Error: ID y edad deben ser números."); //
Manejo de error numérico
        }
    }

    /**
     * Muestra la lista de estudiantes en el área de texto.
     */
    private void mostrarEstudiantes() {
        StringBuilder sb = new StringBuilder();
        for (Estudiante e : service.obtenerTodos()) {
            sb.append(e.toString()).append("\n"); // Agregar cada estudiante a la
lista
        }
        listaEstudiantes.setText(sb.toString());
        mensajeLabel.setText("Lista actualizada.");
    }

    /**
     * Busca un estudiante por su ID basado en el input del usuario.
     * Si lo encuentra, muestra los datos en los campos de entrada.
     */
    private void buscarEstudiante() {
        try {
            int id = Integer.parseInt(idField.getText());
            Estudiante estudiante = service.buscarPorId(id);
            if (estudiante != null) {
                nombreField.setText(estudiante.getNombre());
                edadField.setText(String.valueOf(estudiante.getEdad()));
                mensajeLabel.setText("Estudiante encontrado.");
            } else {
                mensajeLabel.setText("Estudiante no encontrado.");
            }
        } catch (NumberFormatException e) {
            mensajeLabel.setText("Error: ID debe ser un número.");
        }
    }

    /**
     * Actualiza los datos de un estudiante existente basado en el input del
usuario.
     */
    private void actualizarEstudiante() {
        try {
            int id = Integer.parseInt(idField.getText());
            String nombre = nombreField.getText();
            int edad = Integer.parseInt(edadField.getText());
            Estudiante estudiante = service.buscarPorId(id);
            if (estudiante != null) {
                service.actualizarEstudiante(id, nombre, edad);
                mensajeLabel.setText("Estudiante actualizado con éxito.");
                mostrarEstudiantes();
                limpiarCampos();
            } else {
                mensajeLabel.setText("Estudiante no encontrado.");
            }
        } catch (NumberFormatException e) {
```

```
            mensajeLabel.setText("Error: ID y edad deben ser números.");
        }
    }

    /**
     * Elimina un estudiante basado en su ID ingresado por el usuario.
     */
    private void eliminarEstudiante() {
        try {
            int id = Integer.parseInt(idField.getText());
            Estudiante estudiante = service.buscarPorId(id);
            if (estudiante != null) {
                service.eliminarEstudiante(id);
                mensajeLabel.setText("Estudiante eliminado con éxito.");
                mostrarEstudiantes();
                limpiarCampos();
            } else {
                mensajeLabel.setText("Estudiante no encontrado.");
            }
        } catch (NumberFormatException e) {
            mensajeLabel.setText("Error: ID debe ser un número.");
        }
    }

    /**
     * Limpia los campos de entrada de texto.
     */
    private void limpiarCampos() {
        idField.clear();
        nombreField.clear();
        edadField.clear();
    }
}
```

Main.java

Actúa como la clase principal de la aplicación, encargada de iniciar el programa. Sirve como el punto de entrada que inicializa y lanza la interfaz de usuario, integrando las capas del sistema para su ejecución.

```
package ec.edu.espe;

import ec.edu.espe.presentacion.EstudianteUI;
import javafx.application.Application;

/**
 * Clase principal de la aplicación que extiende de Application,
 * utilizada para iniciar una aplicación JavaFX.
 */
public class Main extends Application {



    /**
     * Metodo principal (`main`) de la aplicación.
     * Es el punto de entrada de la aplicación Java.
     *
```

```
     * @param args Los argumentos de la línea de comandos.
     */
    public static void main(String[] args) {
        launch(args);
    }

    /**
     * Metodo `start` que se ejecuta al iniciar la aplicación JavaFX.
     * Inicializa e invoca la interfaz de usuario principal de la aplicación.
     *
     * @param primaryStage La ventana principal de la aplicación JavaFX.
     */
    @Override
    public void start(javafx.stage.Stage primaryStage) {
        EstudianteUI ui = new EstudianteUI();
        ui.mostrarVentana(primaryStage);
    }
}
```

Arquitectura en capas

Arquitectura en Capas
1. Divide el sistema en capas jerárquicas, presentación, lógica de negocio y acceso a datos
2. Enfocado en la estructura del sistema y sus dependencias
3. Facilita mantenimiento y escalabilidad al aislar responsabilidades generales
4. Usualmente se aplica a nivel de sistema completo

Conclusión

Este proyecto demuestra la utilidad de aplicar una arquitectura limpia y mantenible en Java, separando responsabilidades y facilitando la escalabilidad del software. El uso de capas permite modificar la lógica o demás funcionalidades implementadas en el programa.

Esta estructura no solo facilita la escalabilidad del software, permitiendo adaptaciones y ampliaciones futuras, sino que también optimiza la modificación de la lógica o las funcionalidades implementadas en el programa sin comprometer su integridad. Gracias a esta organización, se logra un control global del proyecto, asegurando una visión clara de sus componentes y su interacción.

3. Referencias

https://github.com/kechavez07/es.edu.ec.git