**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:** Pamela Chipe, Carlos Jaya, Elkin Pabón
* **NRC:** 23305
* **Fecha de realización:** 17/06/2025

**2. Objetivo del Taller y Desarrollo**

**Objetivo del Taller:**  
  
Aplicar los patrones de diseño estructural Facade y de comportamiento Mediator en la implementación de un sistema de gestión de estudiantes, estructurado en capas, con el fin de reducir la complejidad del acceso a la lógica del negocio, centralizar la coordinación de componentes, mejorar la organización del código y facilitar su mantenimiento y escalabilidad.

**Desarrollo:**  
Requisitos Funcionales

* **RF1:** La aplicación debe permitir agregar un nuevo estudiante con los datos: orden, nombre y edad.
* **RF2:** El sistema debe mostrar todos los estudiantes registrados.
* **RF3:** El usuario debe poder consultar a un estudiante por su número de id.
* **RF4:** El sistema debe permitir modificar los datos de un estudiante existente.
* **RF5:** El sistema debe permitir eliminar un estudiante por su número de id.
* **RF6:** La aplicación debe estar organizada en tres capas: modelo, repositorio, servicio y presentación.

**Facade:**

Es un patrón de diseño estructural que proporciona una interfaz unificada y simplificada para acceder a un conjunto de clases o subsistemas complejos. Su objetivo es ocultar la complejidad interna del sistema al cliente, permitiendo que este interactúe con un solo punto de acceso, mejorando así la legibilidad y el desacoplamiento del código.

**Estructura del proyecto:**  
  
 src/

│

└──ec.edu.espe/

│

├── datos/

│ ├── model/

│ │ └── Estudiante.java

│ │

│ └── repository/

│ └── EstudianteRepository.java

│

├── logica\_negocio/

│ └── EstudianteService.java

│

├── facade/

│ └── EstudianteService.java

│

├── presentacion/

│ └── EstudianteUI.java

│

└── Main.java

1. **datos/ model/ Estudiante.java**

| **package** ec.edu.espe.datos.model;  /\*\*  \* Clase que representa a un estudiante.  \* Contiene los atributos básicos de un estudiante como ID, nombre y edad.  \* Se proporcionan métodos para acceder y modificar estos atributos,  \* así como un método para mostrar la información del estudiante como cadena de texto.  \*/  **public** **class** **Estudiante** {  **private** **int** id;  **private** String nombre;  **private** **int** edad;  /\*\*  \* Constructor vacío requerido para instanciación sin parámetros.  \*/  **public** **Estudiante**() {  }  /\*\*  \* Constructor con parámetros para inicializar un estudiante.  \* @param id Identificador único del estudiante.  \* @param nombre Nombre del estudiante.  \* @param edad Edad del estudiante.  \*/  **public** **Estudiante**(**int** id, String nombre, **int** edad) {  **this**.id = id;  **this**.nombre = nombre;  **this**.edad = edad;  }  **public** **int** **getId**() {  **return** id;  }  **public** **void** **setId**(**int** id) {  **this**.id = id;  }  **public** String **getNombre**() {  **return** nombre;  }  **public** **void** **setNombre**(String nombre) {  **this**.nombre = nombre;  }  **public** **int** **getEdad**() {  **return** edad;  }  **public** **void** **setEdad**(**int** edad) {  **this**.edad = edad;  }  /\*\*  \* Devuelve la representación textual del objeto estudiante.  \* @return Una cadena con los atributos del estudiante.  \*/  **@Override**  **public** String **toString**() {  **return** "Estudiante{" +  "id=" + id +  ", nombre='" + nombre + '\'' +  ", edad=" + edad +  '}';  }  } |
| --- |

1. **datos/ repositoy/ EstudianteRepository.java**

| package ec.edu.espe.datos.repository;  import ec.edu.espe.datos.model.Estudiante;  import java.util.ArrayList;  import java.util.Comparator;  import java.util.List;  import java.util.Optional;  /\*\*  \* Clase que simula un repositorio en memoria para almacenar estudiantes.  \* Permite realizar operaciones CRUD sobre la lista de estudiantes.  \*/  public class EstudianteRepository {  private final List<Estudiante> estudiantes = new ArrayList<>();  /\*\*  \* Guarda un nuevo estudiante en la lista.  \* @param estudiante Objeto Estudiante a guardar.  \*/  public void guardar(Estudiante estudiante) {  estudiantes.add(estudiante);  }  /\*\*  \* Retorna todos los estudiantes ordenados por ID ascendente.  \* @return Lista ordenada de estudiantes.  \*/  public List<Estudiante> obtenerTodos() {  return estudiantes.stream()  .sorted(Comparator.comparingInt(Estudiante::getId))  .toList();  }  /\*\*  \* Busca un estudiante por su ID.  \* @param id Identificador del estudiante a buscar.  \* @return Optional con el estudiante si se encuentra.  \*/  public Optional<Estudiante> buscarPorId(int id) {  return estudiantes.stream()  .filter(e -> e.getId() == id)  .findFirst();  }  /\*\*  \* Actualiza los datos de un estudiante existente.  \* @param estudianteActualizado Objeto con los nuevos datos.  \*/  public void actualizar(Estudiante estudianteActualizado) {  buscarPorId(estudianteActualizado.getId()).ifPresent(estudiante -> {  estudiante.setNombre(estudianteActualizado.getNombre());  estudiante.setEdad(estudianteActualizado.getEdad());  });  }  /\*\*  \* Elimina un estudiante de la lista por su ID.  \* @param id Identificador del estudiante a eliminar.  \* @return true si fue eliminado; false si no se encontró.  \*/  public boolean eliminarPorId(int id) {  return estudiantes.removeIf(e -> e.getId() == id);  }  } |
| --- |

1. **facade/ EstudianteFacade.java**

| **package** ec.edu.espe.facade;  **import** **ec.edu.espe.datos.model.Estudiante**;  **import** **ec.edu.espe.logica\_negocio.EstudianteService**;  **import** **java.util.List**;  /\*\*  \* Facade proporciona una interfaz simplificada para gestionar estudiantes.  \* Encapsula la lógica de negocio y oculta los detalles del servicio.  \*/  **public** **class** **EstudianteFacade** {  **private** **final** EstudianteService estudianteService;  /\*\*  \* Constructor que inicializa el servicio de estudiantes.  \*/  **public** **EstudianteFacade**() {  **this**.estudianteService = **new** EstudianteService();  }  /\*\*  \* Agrega un nuevo estudiante.  \* @param id ID del estudiante.  \* @param nombre Nombre del estudiante.  \* @param edad Edad del estudiante.  \*/  **public** **void** **agregarEstudiante**(**int** id, String nombre, **int** edad) {  estudianteService.crearEstudiante(id, nombre, edad);  }  /\*\*  \* Obtiene la lista de todos los estudiantes.  \* @return Lista de estudiantes.  \*/  **public** List<Estudiante> **obtenerTodosEstudiantes**() {  **return** estudianteService.obtenerTodos();  }  /\*\*  \* Busca un estudiante por ID.  \* @param id ID del estudiante.  \* @return Objeto Estudiante o null si no se encuentra.  \*/  **public** Estudiante **buscarEstudiantePorId**(**int** id) {  **return** estudianteService.buscarPorId(id);  }  /\*\*  \* Actualiza los datos de un estudiante.  \* @param id ID del estudiante.  \* @param nuevoNombre Nuevo nombre.  \* @param nuevaEdad Nueva edad.  \*/  **public** **void** **actualizarEstudiante**(**int** id, String nuevoNombre, **int** nuevaEdad) {  estudianteService.actualizarEstudiante(id, nuevoNombre, nuevaEdad);  }  /\*\*  \* Elimina un estudiante por su ID.  \* @param id ID del estudiante.  \*/  **public** **void** **eliminarEstudiante**(**int** id) {  estudianteService.eliminarEstudiante(id);  }  } |
| --- |

1. **logica\_negocio/ 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 de servicio que encapsula la lógica de negocio relacionada con los estudiantes.  \* Utiliza el repositorio para acceder y modificar los datos.  \*/  **public** **class** **EstudianteService** {  **private** **final** EstudianteRepository estudianteRepository;  /\*\*  \* Constructor que inicializa el repositorio de estudiantes.  \*/  **public** **EstudianteService**() {  **this**.estudianteRepository = **new** EstudianteRepository();  }  /\*\*  \* Crea un nuevo estudiante y lo guarda en el repositorio.  \* @param id ID del estudiante.  \* @param nombre Nombre del estudiante.  \* @param edad Edad del estudiante.  \*/  **public** **void** **crearEstudiante**(**int** id, String nombre, **int** edad) {  Estudiante nuevo = **new** Estudiante(id, nombre, edad);  estudianteRepository.guardar(nuevo);  }  /\*\*  \* Retorna todos los estudiantes ordenados por ID.  \* @return Lista de estudiantes.  \*/  **public** List<Estudiante> **obtenerTodos**() {  **return** estudianteRepository.obtenerTodos();  }  /\*\*  \* Busca un estudiante por su ID.  \* @param id ID del estudiante.  \* @return El estudiante encontrado o null si no existe.  \*/  **public** Estudiante **buscarPorId**(**int** id) {  **return** estudianteRepository.buscarPorId(id).orElse(**null**);  }  /\*\*  \* Actualiza los datos de un estudiante.  \* @param id ID del estudiante.  \* @param nuevoNombre Nuevo nombre del estudiante.  \* @param nuevaEdad Nueva edad del estudiante.  \*/  **public** **void** **actualizarEstudiante**(**int** id, String nuevoNombre, **int** nuevaEdad) {  Estudiante actualizado = **new** Estudiante(id, nuevoNombre, nuevaEdad);  estudianteRepository.actualizar(actualizado);  }  /\*\*  \* Elimina un estudiante del repositorio por su ID.  \* @param id ID del estudiante.  \*/  **public** **void** **eliminarEstudiante**(**int** id) {  estudianteRepository.eliminarPorId(id);  }  } |
| --- |

1. **presentacion/ EstudianteUI.java**

| **package** ec.edu.espe.presentacion;  **import** **ec.edu.espe.datos.model.Estudiante**;  **import** **ec.edu.espe.facade.EstudianteFacade**;  **import** **javax.swing.\***;  **import** **javax.swing.table.DefaultTableModel**;  **import** **java.awt.\***;  **import** **java.util.List**;  /\*\*  \* Interfaz gráfica para gestionar estudiantes usando facade.  \* Permite crear, ver, actualizar y eliminar registros.  \*/  **public** **class** **EstudianteUI** **extends** JFrame {  **private** **final** EstudianteFacade estudianteFacade;  **private** **final** JTextField txtId = **new** JTextField(**5**);  **private** **final** JTextField txtNombre = **new** JTextField(**10**);  **private** **final** JTextField txtEdad = **new** JTextField(**5**);  **private** **final** DefaultTableModel tableModel = **new** DefaultTableModel(**new** Object[]{"ID", "Nombre", "Edad"}, **0**);  **private** **final** JTable tablaEstudiantes = **new** JTable(tableModel);  /\*\*  \* Constructor que configura la ventana y sus componentes.  \*/  **public** **EstudianteUI**() {  estudianteFacade = **new** EstudianteFacade();  setTitle("Gestión de Estudiantes");  setSize(**600**, **400**);  setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  setLocationRelativeTo(**null**);  JPanel panelCampos = **new** JPanel();  panelCampos.add(**new** JLabel("ID:"));  panelCampos.add(txtId);  panelCampos.add(**new** JLabel("Nombre:"));  panelCampos.add(txtNombre);  panelCampos.add(**new** JLabel("Edad:"));  panelCampos.add(txtEdad);  JButton btnCrear = **new** JButton("Crear");  JButton btnActualizar = **new** JButton("Actualizar");  JButton btnEliminar = **new** JButton("Eliminar");  JButton btnVerTodo = **new** JButton("Ver Todo");  btnCrear.addActionListener(e -> crearEstudiante());  btnActualizar.addActionListener(e -> actualizarEstudiante());  btnEliminar.addActionListener(e -> eliminarEstudiante());  btnVerTodo.addActionListener(e -> mostrarEstudiantes());  JPanel panelBotones = **new** JPanel();  panelBotones.add(btnCrear);  panelBotones.add(btnActualizar);  panelBotones.add(btnEliminar);  panelBotones.add(btnVerTodo);  JScrollPane scrollPane = **new** JScrollPane(tablaEstudiantes);  setLayout(**new** BorderLayout());  add(panelCampos, BorderLayout.NORTH);  add(panelBotones, BorderLayout.CENTER);  add(scrollPane, BorderLayout.SOUTH);  }  /\*\*  \* Lógica para crear un nuevo estudiante desde la UI.  \*/  **private** **void** **crearEstudiante**() {  **int** id = Integer.parseInt(txtId.getText());  String nombre = txtNombre.getText();  **int** edad = Integer.parseInt(txtEdad.getText());  estudianteFacade.agregarEstudiante(id, nombre, edad);  limpiarCampos();  mostrarEstudiantes();  }  /\*\*  \* Lógica para actualizar un estudiante.  \*/  **private** **void** **actualizarEstudiante**() {  **int** id = Integer.parseInt(txtId.getText());  String nombre = txtNombre.getText();  **int** edad = Integer.parseInt(txtEdad.getText());  estudianteFacade.actualizarEstudiante(id, nombre, edad);  limpiarCampos();  mostrarEstudiantes();  }  /\*\*  \* Lógica para eliminar un estudiante por ID.  \*/  **private** **void** **eliminarEstudiante**() {  **int** id = Integer.parseInt(txtId.getText());  estudianteFacade.eliminarEstudiante(id);  limpiarCampos();  mostrarEstudiantes();  }  /\*\*  \* Muestra todos los estudiantes en la tabla.  \*/  **private** **void** **mostrarEstudiantes**() {  tableModel.setRowCount(**0**);  List<Estudiante> estudiantes = estudianteFacade.obtenerTodosEstudiantes();  **for** (Estudiante est : estudiantes) {  tableModel.addRow(**new** Object[]{est.getId(), est.getNombre(), est.getEdad()});  }  }  /\*\*  \* Limpia los campos de entrada.  \*/  **private** **void** **limpiarCampos**() {  txtId.setText("");  txtNombre.setText("");  txtEdad.setText("");  }  } |
| --- |

1. **Main.java**

| **package** ec.edu.espe;  **import** **ec.edu.espe.presentacion.EstudianteUI**;  **import** **javax.swing.\***;  /\*\*  \* Clase principal que inicia la aplicación de gestión de estudiantes.  \*/  **public** **class** **Main** {  /\*\*  \* Método main que lanza la interfaz gráfica.  \* @param args Argumentos de línea de comandos.  \*/  **public** **static** **void** **main**(String[] args) {  SwingUtilities.invokeLater(() -> {  EstudianteUI ventana = **new** EstudianteUI();  ventana.setVisible(**true**);  });  }  } |
| --- |

**Características:**

| Patrón de diseño Facade | |
| --- | --- |
| 1 | Ofrece una sola interfaz para usar varias clases al mismo tiempo. |
| 2 | Oculta los detalles internos del sistema y muestra solo lo necesario. |
| 3 | Ayuda a que el código del usuario sea más simple y fácil de mantener. |
| 4 | Sirve como una puerta de entrada para controlar todo el sistema desde un solo lugar. |

**Mediator:** El patrón Mediator centraliza la comunicación entre objetos, evitando que se conecten directamente. Así, reduce el acoplamiento y facilita la coordinación entre componentes del sistema.

**Estructura del proyecto:**  
  
src/

│

└──ec.edu.espe/

│

├── datos/

│ ├── model/

│ │ └── Estudiante.java

│ │

│ └── repository/

│ └── EstudianteRepository.java

│

├── logica\_negocio/

│ └── EstudianteService.java

│

├── mediator/

│ ├── EstudianteMediator.java

│ └── EstudianteMediatorImpl.java

│

├── presentacion/

│ └── EstudianteUI.java

│

└── Main.java

1. **datos/ model/ Estudiante.java**

| **package** ec.edu.espe.datos.model;  **public** **class** **Estudiante** {  // Atributos privados del estudiante (encapsulamiento).  **private** **int** id; // Identificador único del estudiante.  **private** String nombre; // Nombre del estudiante.  **private** **int** edad; // Edad del estudiante.  // Constructor de la clase Estudiante.  // Permite crear un objeto Estudiante con un id, nombre y edad específicos.  **public** **Estudiante**(**int** id, String nombre, **int** edad) {  **this**.id = id;  **this**.nombre = nombre;  **this**.edad = edad;  }  // Métodos getter y setter para el atributo 'id'.  // Permiten obtener y modificar el valor del id de forma controlada.  **public** **int** **getId**() {  **return** id;  }  **public** **void** **setId**(**int** id) {  **this**.id = id;  }  // Métodos getter y setter para el atributo 'nombre'.  **public** String **getNombre**() {  **return** nombre;  }  **public** **void** **setNombre**(String nombre) {  **this**.nombre = nombre;  }  // Métodos getter y setter para el atributo 'edad'.  **public** **int** **getEdad**() {  **return** edad;  }  **public** **void** **setEdad**(**int** edad) {  **this**.edad = edad;  }  // Método toString sobrescrito.  // Devuelve una representación en texto del objeto Estudiante.  // Útil para imprimir la información del estudiante de forma legible.  **@Override**  **public** String **toString**() {  **return** "ID: " + id + ", Nombre: " + nombre + ", Edad: " + edad;  }  } |
| --- |

1. **datos/ repositoy/ EstudianteRepository.java**

| **package ec.edu.espe.datos.repository;**  **import ec.edu.espe.datos.model.Estudiante;**  **import java.util.ArrayList;**  **import java.util.List;**  **// Clase que simula un repositorio de estudiantes (almacenamiento en memoria).**  **public class EstudianteRepository {**  **// Lista que almacena los objetos Estudiante.**  **private List<Estudiante> estudiantes;**  **// Constructor que inicializa la lista vacía de estudiantes.**  **public EstudianteRepository() {**  **estudiantes = new ArrayList<>();**  **}**  **// Método para agregar un nuevo estudiante a la lista.**  **public void agregarEstudiante(Estudiante estudiante) {**  **estudiantes.add(estudiante);**  **}**  **// Método para obtener la lista completa de estudiantes.**  **public List<Estudiante> obtenerEstudiantes() {**  **return estudiantes;**  **}**  **// Método para buscar un estudiante por su ID.**  **// Si lo encuentra, lo devuelve; si no, retorna null.**  **public Estudiante obtenerEstudiantePorId(int id) {**  **for (Estudiante estudiante : estudiantes) {**  **if (estudiante.getId() == id) {**  **return estudiante;**  **}**  **}**  **return null;**  **}**  **// Método para actualizar los datos de un estudiante existente.**  **// Se busca por ID y se actualizan su nombre y edad si existe.**  **public void actualizarEstudiante(Estudiante estudiante) {**  **Estudiante estudianteExistente = obtenerEstudiantePorId(estudiante.getId());**  **if (estudianteExistente != null) {**  **estudianteExistente.setNombre(estudiante.getNombre());**  **estudianteExistente.setEdad(estudiante.getEdad());**  **}**  **}**  **// Método para eliminar un estudiante por su ID.**  **// Si el estudiante existe, se elimina de la lista.**  **public void eliminarEstudiante(int id) {**  **Estudiante estudiante = obtenerEstudiantePorId(id);**  **if (estudiante != null) {**  **estudiantes.remove(estudiante);**  **}**  **}**  **}** |
| --- |

1. **mediator/ EstudianteMediator.java**

| **package** [ec.edu](http://ec.edu).espe.mediator;  **import** **ec.edu.espe.datos.model.Estudiante**;  **import** **java.util.List**;  /\*\*  \* Interfaz que define las operaciones que se pueden realizar sobre los estudiantes.  \* Actúa como un punto de acceso unificado o fachada para coordinar la lógica de negocio.  \* Puede facilitar desacoplamiento entre la presentación (UI/API) y la lógica interna.  \*/  **public** **interface** **EstudianteMediator** {  // Método para crear un nuevo estudiante.  **void** **crearEstudiante**(**int** id, String nombre, **int** edad);  // Método para actualizar los datos de un estudiante existente.  **void** **actualizarEstudiante**(**int** id, String nombre, **int** edad);  // Método para eliminar un estudiante según su ID.  **void** **eliminarEstudiante**(**int** id);  // Método para obtener un estudiante específico por ID.  Estudiante **obtenerEstudiante**(**int** id);  // Método para obtener la lista de todos los estudiantes registrados.  List<Estudiante> **obtenerTodosEstudiantes**();  } |
| --- |

1. **mediator/ EstudianteMediatorImpl.java**

| **package** ec.edu.espe.mediator;.  **import** **ec.edu.espe.datos.model.Estudiante**;  **import** **ec.edu.espe.logica\_negocio.EstudianteService**;  **import** **java.util.List**;  /\*\*  \* Implementación concreta de la interfaz EstudianteMediator.  \* Esta clase actúa como intermediaria entre una capa superior (como una interfaz de usuario o API)  \* y la lógica del negocio gestionada por EstudianteService.  \*/  **public** **class** **EstudianteMediatorImpl** **implements** EstudianteMediator {  // Atributo privado que mantiene una referencia al servicio de estudiantes.  **private** EstudianteService estudianteService;  // Constructor que instancia el servicio al momento de crear el mediador.  **public** **EstudianteMediatorImpl**() {  estudianteService = **new** EstudianteService();  }  // Implementación del método para crear un nuevo estudiante.  // Delegamos la lógica al servicio.  **@Override**  **public** **void** **crearEstudiante**(**int** id, String nombre, **int** edad) {  estudianteService.agregarEstudiante(id, nombre, edad);  }  // Implementación del método para actualizar un estudiante.  // Se crea un nuevo objeto con los datos y se pasa al servicio.  **@Override**  **public** **void** **actualizarEstudiante**(**int** id, String nombre, **int** edad) {  estudianteService.actualizarEstudiante(id, nombre, edad);  }  // Implementación del método para eliminar un estudiante por ID.  **@Override**  **public** **void** **eliminarEstudiante**(**int** id) {  estudianteService.eliminarEstudiante(id);  }  // Implementación del método para obtener un estudiante por su ID.  **@Override**  **public** Estudiante **obtenerEstudiante**(**int** id) {  **return** estudianteService.obtenerEstudiantePorId(id);  }  // Implementación del método para obtener la lista de todos los estudiantes.  **@Override**  **public** List<Estudiante> **obtenerTodosEstudiantes**() {  **return** estudianteService.obtenerEstudiantes();  }  } |
| --- |

1. **logica\_negocio/ 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 representa el servicio de estudiantes.  // Esta clase actúa como intermediaria entre el controlador (interfaz de usuario o API) y el repositorio (datos).  **public** **class** **EstudianteService** {  // Atributo privado que referencia al repositorio de estudiantes.  **private** EstudianteRepository estudianteRepository;  // Constructor que instancia el repositorio.  **public** **EstudianteService**() {  estudianteRepository = **new** EstudianteRepository();  }  // Método para agregar un nuevo estudiante usando datos individuales.  // Crea un objeto Estudiante y lo pasa al repositorio para su almacenamiento.  **public** **void** **agregarEstudiante**(**int** id, String nombre, **int** edad) {  Estudiante estudiante = **new** Estudiante(id, nombre, edad);  estudianteRepository.agregarEstudiante(estudiante);  }  // Método que devuelve la lista completa de estudiantes.  **public** List<Estudiante> **obtenerEstudiantes**() {  **return** estudianteRepository.obtenerEstudiantes();  }  // Método que devuelve un estudiante específico según su ID.  **public** Estudiante **obtenerEstudiantePorId**(**int** id) {  **return** estudianteRepository.obtenerEstudiantePorId(id);  }  // Método para actualizar los datos de un estudiante.  // Crea un objeto Estudiante con los nuevos valores y lo pasa al repositorio.  **public** **void** **actualizarEstudiante**(**int** id, String nombre, **int** edad) {  Estudiante estudiante = **new** Estudiante(id, nombre, edad);  estudianteRepository.actualizarEstudiante(estudiante);  }  // Método para eliminar un estudiante del sistema usando su ID.  **public** **void** **eliminarEstudiante**(**int** id) {  estudianteRepository.eliminarEstudiante(id);  }  } |
| --- |

1. **presentacion/ EstudianteUI.java**

| **package** ec.edu.espe.presentacion;  **import** **ec.edu.espe.datos.model.Estudiante**;  **import** **ec.edu.espe.mediator.EstudianteMediatorImpl**;  **import** **javax.swing.\***;  **import** **javax.swing.table.DefaultTableModel**;  **import** **java.awt.\***;  **import** **java.util.List**;  /\*\*  \* Clase que define la interfaz gráfica (GUI) para la gestión de estudiantes.  \* Utiliza Swing y sigue el patrón MVC, donde esta clase representa la "Vista" que interactúa con el "Mediador".  \*/  **public** **class** **EstudianteUI** **extends** JFrame {  // Instancia del mediador para acceder a las operaciones de negocio.  **private** EstudianteMediatorImpl mediator;  // Componentes de la interfaz.  **private** JTextField txtId, txtNombre, txtEdad;  **private** JButton btnCrear, btnVer, btnActualizar;  **private** JTable tablaEstudiantes;  **private** DefaultTableModel tableModel;  // Constructor que configura toda la ventana.  **public** **EstudianteUI**() {  setTitle("Gestión de Estudiantes"); // Título de la ventana.  setSize(**600**, **500**); // Tamaño de la ventana.  setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); // Cierra el programa al cerrar ventana.  setLocationRelativeTo(**null**); // Centra la ventana en la pantalla.  setLayout(**new** BorderLayout(**10**, **10**)); // Layout general.  mediator = **new** EstudianteMediatorImpl(); // Inicializa el mediador.  // Panel de entrada de datos  JPanel panelCampos = **new** JPanel(**new** GridBagLayout());  GridBagConstraints gbc = **new** GridBagConstraints();  gbc.insets = **new** Insets(**5**, **5**, **5**, **5**); // Espaciado entre componentes  // Campo ID  gbc.gridx = **0**; gbc.gridy = **0**;  panelCampos.add(**new** JLabel("ID:"), gbc);  gbc.gridx = **1**;  txtId = **new** JTextField(**20**);  panelCampos.add(txtId, gbc);  // Campo Nombre  gbc.gridx = **0**; gbc.gridy = **1**;  panelCampos.add(**new** JLabel("Nombre:"), gbc);  gbc.gridx = **1**;  txtNombre = **new** JTextField(**20**);  panelCampos.add(txtNombre, gbc);  // Campo Edad  gbc.gridx = **0**; gbc.gridy = **2**;  panelCampos.add(**new** JLabel("Edad:"), gbc);  gbc.gridx = **1**;  txtEdad = **new** JTextField(**20**);  panelCampos.add(txtEdad, gbc);  // Panel con botones de acciones  JPanel panelBotones = **new** JPanel(**new** FlowLayout(FlowLayout.CENTER, **10**, **10**));  btnCrear = **new** JButton("Crear Estudiante");  btnVer = **new** JButton("Ver Todos");  btnActualizar = **new** JButton("Actualizar");  panelBotones.add(btnCrear);  panelBotones.add(btnVer);  panelBotones.add(btnActualizar);  // Agrupa campos y botones  JPanel panelPrincipal = **new** JPanel(**new** BorderLayout());  panelPrincipal.add(panelCampos, BorderLayout.NORTH);  panelPrincipal.add(panelBotones, BorderLayout.CENTER);  add(panelPrincipal, BorderLayout.NORTH);  // Tabla para mostrar estudiantes  String[] columnNames = {"ID", "Nombre", "Edad"};  tableModel = **new** DefaultTableModel(columnNames, **0**);  tablaEstudiantes = **new** JTable(tableModel);  JScrollPane scrollPane = **new** JScrollPane(tablaEstudiantes);  add(scrollPane, BorderLayout.CENTER);  // Acciones de los botones  btnCrear.addActionListener(e -> crearEstudiante());  btnVer.addActionListener(e -> verEstudiantes());  btnActualizar.addActionListener(e -> actualizarEstudiante());  }  // Método que permite crear un nuevo estudiante o reemplazar uno existente con el mismo ID  **private** **void** **crearEstudiante**() {  **try** {  **int** id = Integer.parseInt(txtId.getText());  String nombre = txtNombre.getText();  **int** edad = Integer.parseInt(txtEdad.getText());  Estudiante existente = mediator.obtenerEstudiante(id);  **if** (existente != **null**) {  mediator.eliminarEstudiante(id); // Elimina si ya existe uno con el mismo ID  JOptionPane.showMessageDialog(**this**, "Estudiante existente eliminado.");  } **else** {  JOptionPane.showMessageDialog(**this**, "No existe un estudiante con ese ID para eliminar.");  }  mediator.crearEstudiante(id, nombre, edad); // Crea el nuevo  JOptionPane.showMessageDialog(**this**, "Estudiante creado con éxito");  limpiarCampos();  } **catch** (NumberFormatException ex) {  JOptionPane.showMessageDialog(**this**, "Por favor, ingresa datos válidos.");  }  }  // Muestra todos los estudiantes registrados en la tabla  **private** **void** **verEstudiantes**() {  List<Estudiante> estudiantes = mediator.obtenerTodosEstudiantes();  tableModel.setRowCount(**0**); // Limpia la tabla antes de agregar nuevos datos  **for** (Estudiante estudiante : estudiantes) {  tableModel.addRow(**new** Object[]{estudiante.getId(), estudiante.getNombre(), estudiante.getEdad()});  }  }  // Actualiza un estudiante con los datos ingresados  **private** **void** **actualizarEstudiante**() {  **try** {  **int** id = Integer.parseInt(txtId.getText());  String nombre = txtNombre.getText();  **int** edad = Integer.parseInt(txtEdad.getText());  mediator.actualizarEstudiante(id, nombre, edad);  JOptionPane.showMessageDialog(**this**, "Estudiante actualizado con éxito");  limpiarCampos();  } **catch** (NumberFormatException ex) {  JOptionPane.showMessageDialog(**this**, "Por favor, ingresa datos válidos.");  }  }  // Limpia los campos de entrada  **private** **void** **limpiarCampos**() {  txtId.setText("");  txtNombre.setText("");  txtEdad.setText("");  }  // Método principal para ejecutar la aplicación  **public** **static** **void** **main**(String[] args) {  SwingUtilities.invokeLater(() -> {  EstudianteUI ui = **new** EstudianteUI(); // Crea una instancia de la UI  ui.setVisible(**true**); // Muestra la ventana  });  }  } |
| --- |

1. **Main.java**

| **package** ec.edu.espe;  /\*\*  \* Clase principal que sirve como punto de entrada del programa.  \* Inicia la interfaz gráfica de usuario para la gestión de estudiantes.  \*/  **import** **ec.edu.espe.presentacion.EstudianteUI**;  **public** **class** **Main** {  // Método main: el punto de inicio de ejecución de la aplicación.  **public** **static** **void** **main**(String[] args) {  // Crea una instancia de la interfaz gráfica.  EstudianteUI estudianteUI = **new** EstudianteUI();  // Hace visible la ventana para que el usuario pueda interactuar.  estudianteUI.setVisible(**true**);  }  } |
| --- |

**Características:**

| Patrón de diseño Mediator | |
| --- | --- |
| 1 | Declara la interfaz Mediator para la comunicación entre componentes (llamados colegas) |
| 2 | Implementa la lógica de coordinación entre los “Colegas” (ConcreteMediator o en este caso, EstudianteMediatorImpl) conoce y mantiene referencias a los objetos colegas |
| 3 | El objetivo es reducir el acoplamiento entre objetos que interactúan entre sí al centralizar la comunicación en un único objeto mediador |
| 4 | Mejora la mantenibilidad y escalabilidad |

**Patrones Estructurales y de Comportamiento:**

| **Estructurales** | **Comportamiento** |
| --- | --- |
| Se encargan de cómo están formadas y conectadas las clases y objetos. | Se encargan de cómo los objetos se comunican y trabajan entre sí. |
| Buscan que el sistema sea más ordenado y fácil de mantener. | Buscan que el sistema sea más flexible y dinámico al ejecutar acciones. |
| Se enfocan en la estructura estática (conexiones, herencia, composición). | Se enfocan en el flujo dinámico (mensajes, eventos, coordinación). |
| Ayudan a ocultar detalles internos y simplificar el uso del sistema. | Ayudan a controlar el comportamiento sin acoplar objetos directamente. |

**3. Conclusiones**

* El patrón *Facade* demostró ser una herramienta eficaz para encapsular la complejidad interna del sistema, permitiendo una interfaz simple y clara para el usuario.
* El desarrollo en capas, favoreció una mejor organización del código, facilitando la lectura, el mantenimiento y la evolución del sistema.
* Este taller permitió no solo aplicar conocimientos teóricos sobre patrones de diseño, sino también adquirir experiencia práctica en la implementación de software estructurado, escalable y adaptable.

**4. Recomendaciones**

* Utilizar el patrón *Facade* cuando se necesite simplificar el acceso a subsistemas complejos, especialmente en sistemas organizados por capas donde el cliente no debe conocer los detalles internos de la lógica de negocio.
* Aplicar patrones como *Mediator* en sistemas donde múltiples objetos deban comunicarse entre sí sin acoplamiento directo, facilitando la coordinación y el mantenimiento del flujo de datos entre componentes.
* Incorporar patrones de diseño desde la fase de análisis y diseño del software, ya que permiten tomar mejores decisiones arquitectónicas, promover el reuso de código y anticiparse a problemas comunes de escalabilidad o mantenimiento.

**5. Referencias**

* Freeman, E., & Robson, E. (2021). *Head First Design Patterns: A Brain-Friendly Guide* (2nd ed.). O'Reilly Media.
* Refactoring Guru. (n.d.). *Facade*.<https://refactoring.guru/design-patterns/facade>
* Microsoft. (2023). *Common web application architectures: Facade pattern*. Microsoft Learn.<https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures>
* Refactoring Guru. (n.d.). *Mediator*. <https://refactoring.guru/design-patterns/mediator>
* Baeldung. (2021). *Mediator design pattern in Java*. <https://www.baeldung.com/java-mediator-pattern>