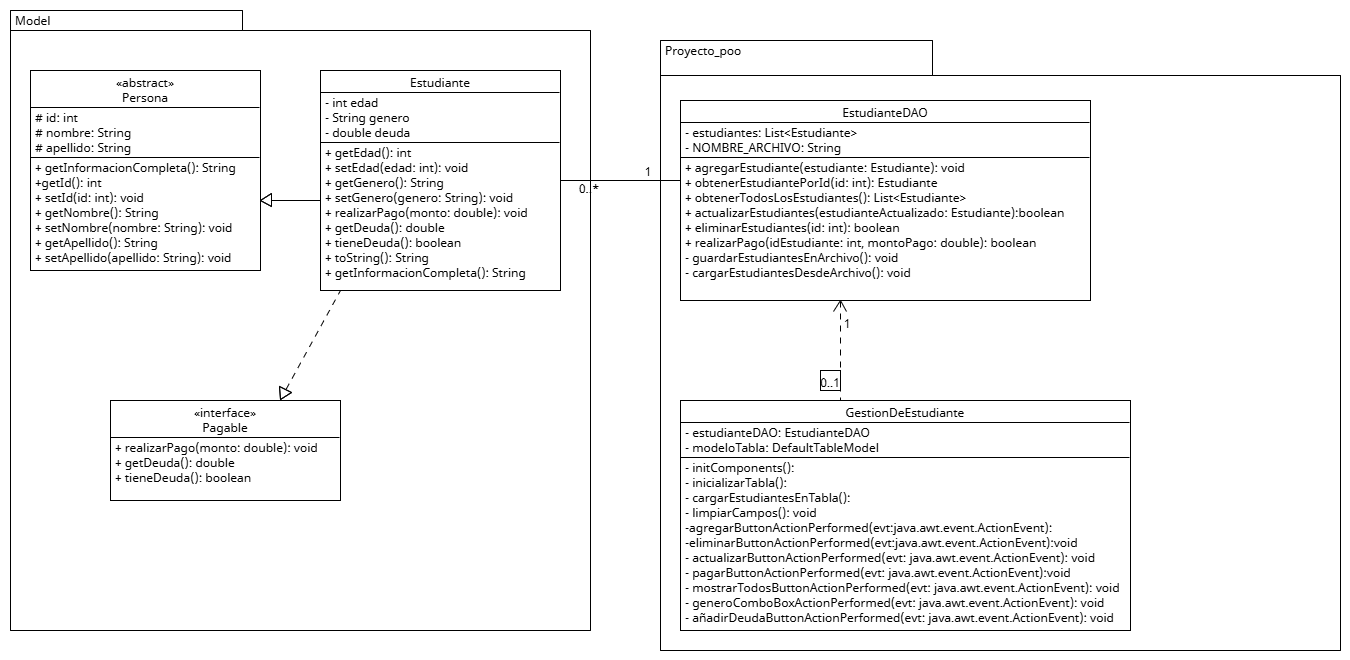
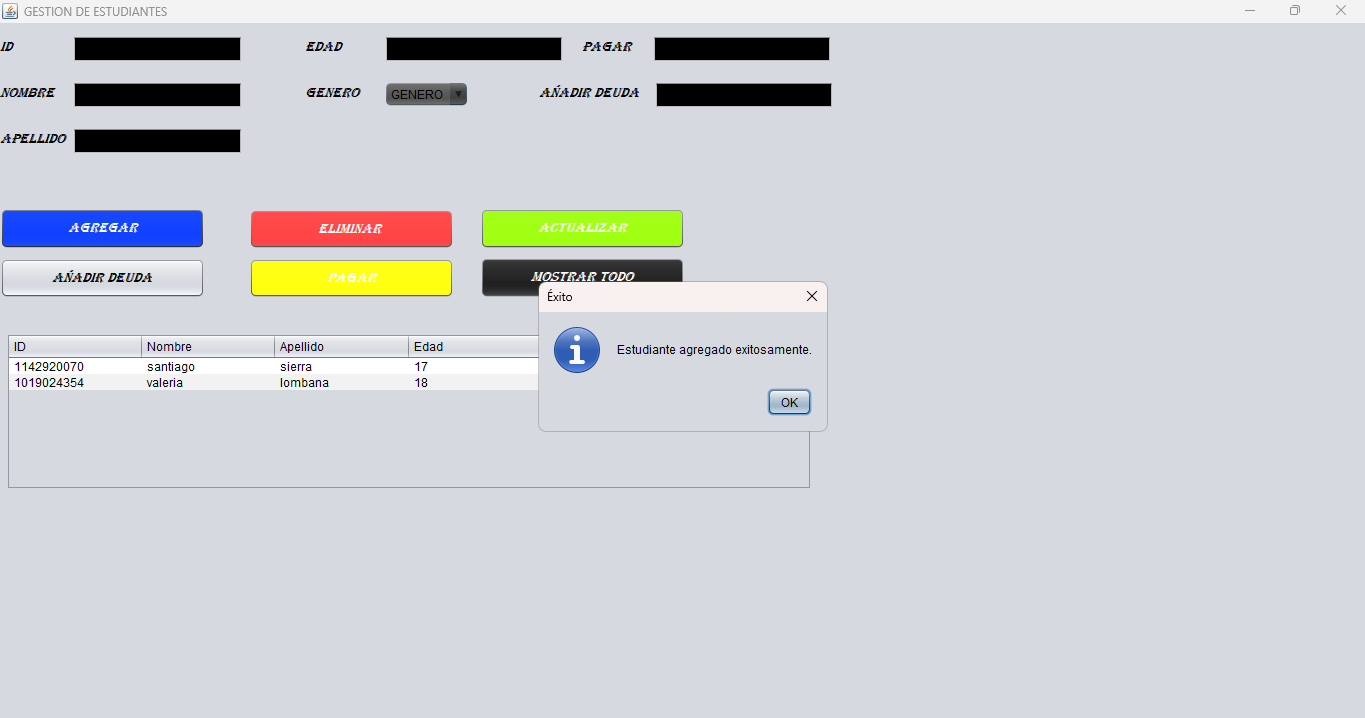
**Desarrollo De Un Software En Java De Un Sistema de Gestión de Estudiantes**

**Descripción Funcional**

Este proyecto es una aplicación de escritorio desarrollada en Java con interfaz gráfica (Swing) que permite gestionar la información de estudiantes. Ofrece funcionalidades esenciales para el mantenimiento de un registro académico básico, incluyendo la capacidad de agregar nuevos estudiantes, actualizar sus datos existentes, eliminar registros, gestionar pagos de deudas y visualizar la lista completa de estudiantes. La aplicación asegura la persistencia de los datos al guardar y cargar la información en un archivo de texto, lo que permite que los datos no se pierdan entre sesione

Diagrama UML

Aplicación Funcionando



### A. Fundamentos de la Programación Orientada a Objetos (POO)

* **Uso de objetos, clases, atributos y métodos:** El proyecto hace uso de **clases** como Estudiante, Persona, EstudianteDAO, y GestionDeEstudiante. Cada clase define **atributos** (ej., id, nombre, apellido, edad, genero, deuda en Estudiante) y **métodos** (ej., getNombre(), setEdad(), realizarPago()) apropiados a su responsabilidad. Se crean

**objetos** de estas clases (ej., instancias de Estudiante y EstudianteDAO) para realizar las operaciones del sistema.

* **Constructores personalizados:** La clase Estudiante incluye constructores personalizados. Por ejemplo,

Estudiante(int id, String nombre, String apellido, int edad, String genero, double deuda) para una inicialización completa, y Estudiante(int id, String nombre, String apellido, int edad, String genero) que sobrecarga el constructor para inicializar la deuda a 0.0 por defecto.

* **Métodos Get y Set:** Se han implementado métodos **get** y **set** para controlar el acceso y la modificación de los atributos en las clases Persona y Estudiante, asegurando el encapsulamiento de los datos.

### B. Principios de Abstracción y Herencia

* **Clase abstracta y extensión:** La clase Persona es definida como abstract, conteniendo atributos comunes (id, nombre, apellido) y un método abstracto (getInformacionCompleta()). La clase

Estudiante extiende (extends) Persona, implementando el método abstracto y añadiendo sus propios atributos y comportamientos específicos.

* **Herencia para jerarquías:** La relación Estudiante extends Persona establece una jerarquía clara, indicando que un estudiante es una especialización de una persona.
* **Implementación de interfaces:** Se ha creado la interfaz Pagable que define un contrato con métodos como realizarPago(double monto), getDeuda(), y tieneDeuda(). La clase

Estudiante implementa (implements) esta interfaz, obligándose a proporcionar una implementación para estos comportamientos relacionados con la deuda.

### C. Polimorfismo

* **Sobrescritura de métodos (@Override):** El método realizarPago(double monto) de la interfaz Pagable es sobrescrito (@Override) en la clase Estudiante para proporcionar la lógica específica de cómo un estudiante maneja los pagos. También

toString() y getInformacionCompleta() son sobrescritos en Estudiante.

* **Sobrecarga de métodos:** La clase Estudiante demuestra sobrecarga de métodos a través de sus **constructores**. Existen dos constructores con el mismo nombre (

Estudiante) pero que aceptan diferentes conjuntos de parámetros, permitiendo crear objetos Estudiante de distintas maneras.

### D. Colecciones

* **Uso de ArrayList:** La clase EstudianteDAO utiliza un java.util.ArrayList<Estudiante> llamado estudiantes para almacenar, gestionar y acceder a la colección de objetos Estudiante en memoria.

### E. Manejo de Archivos

* **Persistencia de datos:** La clase EstudianteDAO es responsable de la persistencia de los datos de los estudiantes. Los métodos

guardarEstudiantesEnArchivo() y cargarEstudiantesDesdeArchivo() aseguran que los datos se guarden en un archivo (estudiantes.txt) y se carguen al iniciar la aplicación, manteniendo la información entre diferentes ejecuciones del programa.

* **Uso de FileReader, FileWriter, BufferedReader, PrintWriter:** Para el manejo de archivos, se utilizan PrintWriter y FileWriter para escribir los datos de los estudiantes en el archivo, y BufferedReader y FileReader para leer la información, garantizando una correcta manipulación de los flujos de entrada/salida.

### F. Interfaces Gráficas con Java Swing

* **Interfaz de usuario funcional:** La clase GestionDeEstudiante extiende javax.swing.JFrame y constituye la interfaz de usuario principal. Utiliza una variedad de componentes Swing como

JLabel (para etiquetas), JTextField (para entradas de texto como ID, nombre), JButton (para acciones como agregar, eliminar), JComboBox (para selección de género), y JTable (para mostrar la lista de estudiantes).

* **Diseño modular:** La aplicación organiza sus funcionalidades en diferentes clases bien definidas (GestionDeEstudiante para la UI, EstudianteDAO para el acceso a datos, Estudiante para el modelo), lo que contribuye a un diseño modular y facilita la comprensión y el mantenimiento del código.

### G. Modelo de Eventos en Swing

* **Implementación de ActionListener:** Se han implementado múltiples eventos de usuario a través de ActionListener. Cada botón en la interfaz (agregarButton, eliminarButton, actualizarButton, pagarButton, mostrarTodosButton) tiene asociado un ActionListener que responde a la interacción del usuario. Por ejemplo, agregarButtonActionPerformed maneja la adición de un nuevo estudiante al hacer clic en el botón "agregar".