Memoria del Proyecto
Gestión de Alumnos
RA 1

Aitor Jury Rodríguez. 2º DAM.
Programación de Servicios y Procesos - RA1.

Índice

1. Introducción

El proyecto consiste en una aplicación en Java para la gestión y almacenamiento de alumnos mediante archivos de texto. Permite almacenar, consultar, modificar y eliminar información de alumnos usando ficheros .txt como medio de persistencia.

El programa se ejecuta en consola y sigue un diseño modular, donde la clase principal controla la interacción con el usuario, y los procesos externos (ProcesoAlmacenamiento y ProcesoDevolucion) se encargan de la lectura y escritura de datos.

2. Objetivos

Los objetivos del proyecto son:

- Crear un sistema de gestión de alumnos que funcione en consola.
- Implementar almacenamiento persistente usando archivos de texto individuales por alumno.
- Validar los datos de entrada (nombre, apellidos, DNI, fecha de nacimiento y nota media).
- Permitir realizar operaciones CRUD:
    - Create: Insertar alumno.
    - Read: Buscar por DNI u obtener todos los alumnos.
    - Update: Modificar alumno.
    - Delete: Eliminar alumno.
- Separar responsabilidades: la clase Principal gestiona la interacción y validación, los procesos externos (ProcesoAlmacenamiento y ProcesoDevolucion) solo leen o escriben archivos.
- Implementar mejoras opcionales (puntos extra): validación de fecha, eliminación de alumno, modificación de alumno, lista ordenada y presentación en terminal.

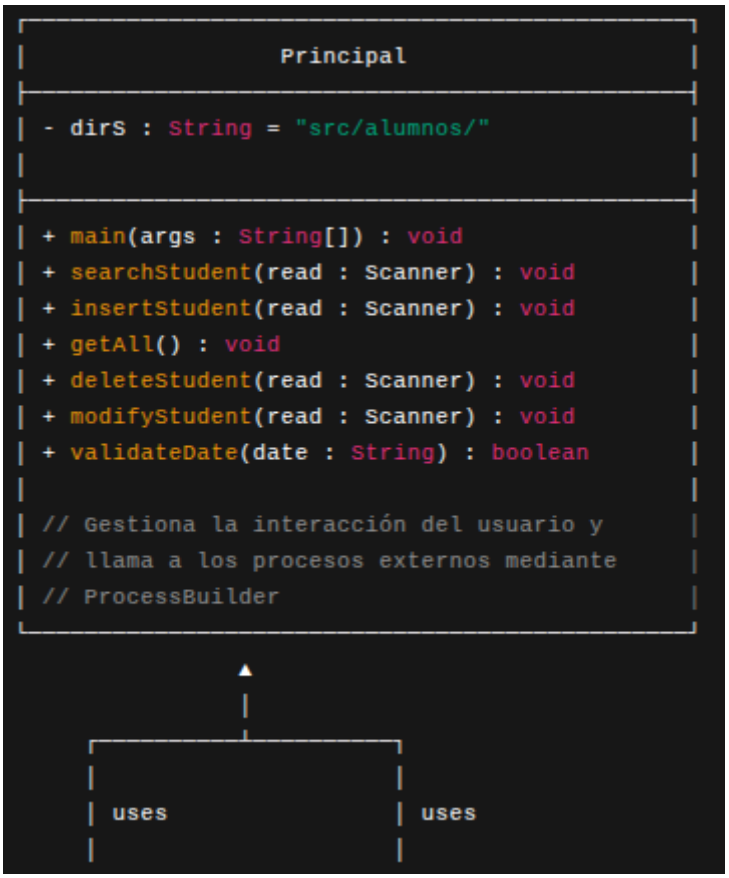
3. Herramientas utilizadas

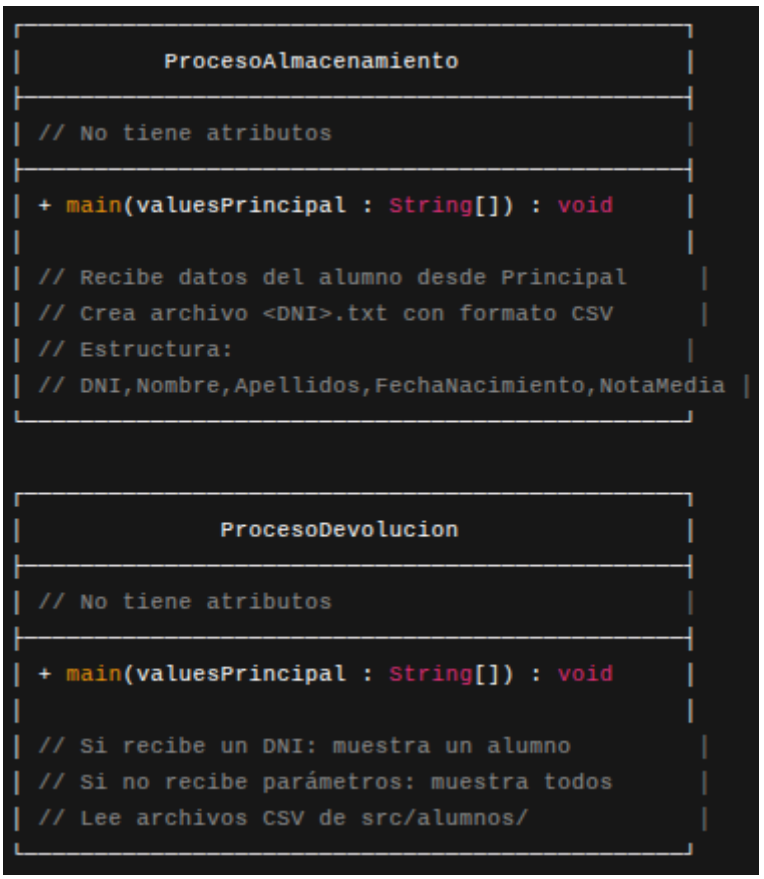
Para el desarrollo del proyecto se han empleado las siguientes herramientas y tecnologías:

- Lenguaje: Java SE 17.
- Entorno de desarrollo (IDE): Visual Studio Code.
- Sistema operativo: Windows 11 y UbuntuMax.
- Librerías y paquetes Java utilizados:
    - java.io → manejo de archivos (File, FileReader, FileWriter, BufferedReader, BufferedWriter).
    - java.time → validación de fechas con LocalDate y DateTimeFormatter.
    - java.util.Scanner → lectura de datos de consola.
    - java.lang.ProcessBuilder → ejecución de procesos externos (java -cp bin Clase Parametros).
- Formato de almacenamiento: CSV (Comma-Separated Values) guardado en ficheros .txt dentro de la carpeta src/alumnos/.
- Ejecución: Terminal integrada de VSCode o consola del sistema (CMD).

4. Diseño (UML y Casos de Uso)

4.1 Diagrama de Clases (UML)





Descripción:

- Principal controla el flujo general del programa y lanza procesos secundarios.
- ProcesoAlmacenamiento guarda los datos en un archivo individual por alumno.
- ProcesoDevolucion lee uno o todos los archivos y muestra la información en consola.

4.2 Casos de Uso

Actores:

- Usuario del sistema (persona que gestiona los alumnos desde consola).

Casos de uso principales:

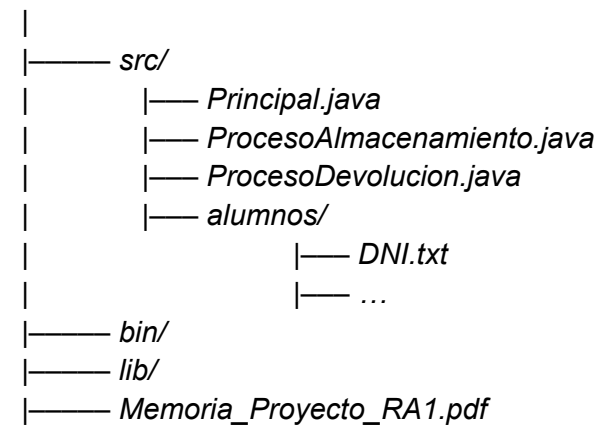
- Insertar alumno: El usuario introduce los datos y el sistema crea un archivo nuevo con su información.
- Buscar alumno por DNI: El sistema localiza y muestra el contenido del archivo asociado al DNI.
- Obtener todos los alumnos: Se listan todos los registros almacenados.
- Eliminar alumno: El sistema borra el archivo correspondiente.
- Modificar alumno: Permite editar la información del alumno manteniendo los campos no modificados.
- Finalizar: Sale del programa de forma ordenada.

5. Estructura del Proyecto

La estructura del proyecto se basa en:

- Carpeta del proyecto: Proyecto_Almacenaje_Alumnos/
- Carpeta base de las clases: Proyecto_Almacenaje_Alumnos/src/
- Carpeta de alumnos: Proyecto_Almacenaje_Alumnos/src/alumnos/
- Clases principales:
    - Principal.java: controla el flujo de la aplicación y la interacción con el usuario.
    - ProcesoAlmacenamiento.java → guarda la información de un alumno en un archivo.
    - ProcesoDevolucion.java → lee y muestra la información de un alumno o todos los alumnos.
- Estructura interna de un archivo de alumno (DNI.txt):

DNI,Nombre,Apellidos,FechaNacimiento,NotaMedia
50360798K,Aitor,Jury Rodriguez,17-06-2004,9.15



6. Descripción de Clases y Funciones

6.1 Principal.java

Clase principal del sistema. Gestiona el menú, validaciones y ejecución de procesos.

Funciones principales:

- main(String[] args):
    - Crea la carpeta src/alumnos/ si no existe.
    - Muestra el menú principal y procesa la opción seleccionada.
    - Maneja las opciones: buscar, insertar, obtener todos, eliminar, modificar y finalizar.
- searchStudent(Scanner read):
    - Solicita DNI al usuario.
    - Valida que no esté vacío.
    - Llama al proceso ProcesoDevolucion para mostrar la información.
- insertStudent(Scanner read):
    - Solicita datos del alumno: nombre, apellidos, DNI, fecha de nacimiento, nota media.
    - Valida cada campo: no puede estar vacío y la fecha debe cumplir el formato dd-mm-yyyy.
    - Nota media opcional: si no se introduce, se asigna 0 por defecto.
    - Llama al proceso ProcesoAlmacenamiento para guardar el alumno.
- getAll():
    - Llama a ProcesoDevolucion sin parámetros para obtener todos los alumnos.
- deleteStudent(Scanner read):
    - Solicita DNI.
    - Elimina el archivo correspondiente si existe.
- modifyStudent(Scanner read):
    - Solicita DNI y verifica si el alumno existe.
    - Permite modificar los campos, dejando enter para mantener los datos actuales.
    - Sobrescribe el archivo con los nuevos datos.
- validarFecha(String date):
    - Verifica que la fecha tenga formato correcto dd-MM-yyyy y sea válida usando LocalDate.

Validaciones implementadas:

- Nombre, apellidos y DNI no pueden estar vacíos.
- Fecha debe tener formato correcto y ser una fecha real.
- Nota media es opcional y numérica (se asume 0 si se deja vacía).

6.2 ProcesoAlmacenamiento.java

Se encarga de crear o sobrescribir un archivo de alumno.

Funciones:

- Recibe los parámetros (nombre, apellidos, DNI, fecha, nota).
- Crea el archivo DNI.txt dentro de src/alumnos/.

- Escribe dos líneas:
    1. Cabecera con los nombres de los campos.
    2. Datos reales del alumno.
- Muestra mensaje de confirmación en consola.

6.3 ProcesoDevolucion.java

Se encarga de leer y mostrar los datos almacenados.

Dos modos de uso:

1. Un parámetro (DNI): muestra los datos del alumno concreto.
2. Sin parámetros: recorre todos los ficheros de src/alumnos/ y los imprime formateados.

Salida en consola:

==========

DNI: 50360798K
Nombre y apellidos: Jury Rodriguez, Aitor
Fecha de nacimiento: 17-06-2004
Nota media: 9.15

==========

7. Mejoras y puntos extra implementados

Para el proyecto, he implementado las siguientes mejoras para conseguir el mejor resultado posible:

1. Eliminación de alumnos por DNI → Implementada en deleteStudent().
2. Modificación de datos de un alumno por DNI → Implementada en modifyStudent().
3. Validación de fecha → Implementada en Principal.java con validarFecha().
4. Presentación de terminal mejorada → Salida formateada con líneas y separación clara de campos.
5. Separación de responsabilidades → Principal.java gestiona interacción y validación, mientras que los procesos solo leen/escriben archivos.

8. Flujo del Programa

El flujo del programa sigue los siguientes puntos:

1. El programa inicia y verifica o crea la carpeta src/alumnos/.
2. Muestra menú principal con opciones.
3. El usuario selecciona una opción:
    - Buscar → solicita DNI → llama a ProcesoDevolucion con DNI.
    - Insertar → solicita datos → valida → llama a ProcesoAlmacenamiento.
    - Obtener todos → llama a ProcesoDevolucion sin parámetros.
    - Eliminar → solicita DNI → borra archivo si existe.
    - Modificar → solicita DNI → permite cambiar datos → sobrescribe archivo.
    - Finalizar → termina programa.
4. El programa vuelve al menú tras completar cada operación hasta que el usuario elige salir.

9. Conclusión

El proyecto cumple los objetivos planteados: permite gestionar alumnos de forma modular y persistente desde la consola.

Objetivos cumplidos:

- Implementación completa de CRUD.
- Validaciones robustas de datos y fechas.
- Separación de lógica e independencia de procesos.
- Uso de archivos CSV como sistema de almacenamiento estructurado.

Posibles mejoras futuras:

- Exportación de los datos a Excel o Base de Datos.
- Incorporación de una interfaz gráfica (Swing, JavaFX).
- Añadir ordenación o filtrado de alumnos por nota o apellido.

Gracias a la arquitectura modular, el sistema es mantenible, escalable y fácilmente ampliable, cumpliendo con los principios de diseño limpio y responsabilidad única.