Universidad Mayor de San Andres

Facultad de Ciencias Puras y Naturales

Carrera de Informática

CONTROL ACADÉMICO

PROGRAMACIÓN II (INF-121)

Proyecto N°11

Integrantes:

Choque Valencia Delma Fernanda
Quispe Lopez Lizbeth
Sandoval Quenta Bryan Daniel
Tinta Lopez Grissel Noemí
Yujra Paye Alejandro Andrés

Docente: Lic. Rosalía López Montalvo

LA PAZ - BOLIVIA

2025

1. Resumen del proyecto

El proyecto "Control Académico" tiene como finalidad resolver la necesidad de gestionar de manera organizada y eficiente los registros académicos en una institución educativa. Este sistema permite almacenar, consultar y controlar la información de estudiantes, docentes, asistencias y notas, todo esto mediante una interfaz gráfica amigable desarrollada en Java.

El sistema brinda funcionalidades como:

- Registro de datos personales y académicos de estudiantes y docentes.
- Asignación de materias a docentes y estudiantes.
- Registro de asistencias por materia.
- Evaluación de estudiantes con asignación de notas según tipo de parcial.
- Visualización y actualización de los datos almacenados.

Para el desarrollo del sistema se utilizaron las siguientes tecnologías:

- Lenguaje: Java
- IDE: NetBeans
- Interfaz gráfica: Swing
- Control de versiones y repositorio: GitHub
- Modelado UML: draw.io

Se aplicaron principios de la Programación Orientada a Objetos como herencia, composición, agregación, encapsulamiento, abstracción, polimorfismo, genericidad y persistencia de objetos, los cuales se reflejan en la estructura de clases del sistema. Además, se diseñó una solución extensible y mantenible que integra funcionalidades clave mediante una arquitectura modular y clara.

2. Objetivo

Diseñar e implementar un sistema orientado a objetos que permita gestionar de forma eficiente los registros académicos de una institución, aplicando los principios de la Programación Orientada a Objetos (POO) y utilizando persistencia de datos mediante archivos .dat.

2.1. Objetivos Específicos

- Aplicar conceptos fundamentales de POO como **herencia**, **composición**, **agregación**, **polimorfismo**, **abstracción**, **excepciones, encapsulamiento** y **clases genéricas**.
- Utilizar al menos un **patrón de diseño** para mejorar la organización del sistema y facilitar su escalabilidad (por ejemplo, Singleton para la gestión centralizada de archivos).
- Implementar la **persistencia de datos** utilizando archivos binarios (.dat) para guardar y recuperar información relacionada a estudiantes, docentes, asistencias y notas.
- Simular una interacción funcional con el usuario a través de una interfaz gráfica desarrollada con **Java Swing**, permitiendo registrar, modificar, y visualizar información.

3. Análisis del problema

3.1. Descripción del contexto

En muchas instituciones educativas, la gestión académica se realiza de manera manual o con herramientas poco especializadas, lo cual genera desorganización, pérdida de información y dificultades para hacer un seguimiento adecuado del rendimiento estudiantil. Este proyecto busca digitalizar el proceso de control académico mediante un sistema que registre y administre datos clave de estudiantes, docentes, asistencias y notas, todo bajo los principios de la Programación Orientada a Objetos.

El sistema permitirá llevar un control más ordenado y automatizado del historial académico de los estudiantes, incluyendo su información personal, materias inscritas, notas asignadas por docentes y asistencias registradas por materia.

3.2. Requisitos funcionales

- Registrar estudiantes con sus datos personales, carrera, registro universitario y materias inscritas.
- Registrar docentes, especificando su tipo (Contratado, Titular o Coordinador), sueldo y materias asignadas.
- Registrar notas por materia y tipo de parcial, incluyendo el docente evaluador.
- Registrar asistencias por estudiante, con fecha, materia y estado de asistencia.
- Mostrar información de estudiantes y docentes, con sus respectivos detalles y datos relacionados.

- Guardar y cargar datos de estudiantes y docentes mediante archivos .dat (persistencia binaria).

3.3. Requisitos no funcionales

- El sistema debe tener una interfaz gráfica fácil de usar (usabilidad).
- El sistema debe estar desarrollado en Java (tecnología requerida).
- Los datos deben almacenarse en archivos .dat (persistencia).

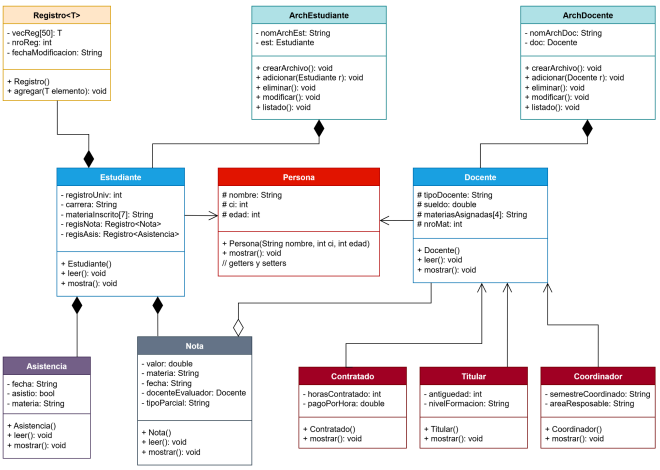
3.4. Casos de uso

- Registrar Estudiante: El usuario ingresa los datos del estudiante y el sistema los almacena, incluyendo materias inscritas.
- Registrar Docente: Se introducen los datos del docente y su tipo (Contratado, Titular, Coordinador), junto con sus materias.
- Registrar Nota: El sistema permite asignar una nota a un estudiante por una materia específica, indicando el docente que evaluó.
- Registrar Asistencia: Se registra si un estudiante asistió o no a una clase específica de una materia en determinada fecha.
- Visualizar Datos: El usuario puede consultar la información de estudiantes y docentes desde la interfaz gráfica.
- Guardar/Cargar Datos: Se genera un archivo .dat al guardar los datos y se puede cargar dicho archivo al iniciar la aplicación para continuar con la información previa.

4. Diseño del sistema

Clases y jerarquía

- **Diagrama UML**



- **Tabla de clases**

Clase	**Atributos principales**	**Métodos clave**
ArchEstudiante	nomErchEst, est	crearArchivo(), adicionar(), eliminar(), modificar(), listado()
ArchDocente	nomArchDoc, doc	crearArchivo(), adicionar(), eliminar(), modificar(), listado()
Persona	nombre, ci, edad	mostrar()
Estudiante	registroUniv, carrera, materiasInstcrito[7], regisNota, regisAsis	leer(), mostrar()
Docente	tipoDocente, sueldo, materiasAsignadas[4], nroMat	leer(), mostrar()
Nota	valor, materia, fecha, docenteEvaluador, tipoParcial	nota(), leer(), mostrar()
Asistencia	fecha, asistio, materia	asistencia(),leer(), mostrar()
Registro<T>	vecReg[50]: T, nroReg, fechaModificacion	registro(), agregar(T elemento)
Contratado	horasContratado, pagoPorHora	contratado(), mostrar()
Titular	antiguedad, nivelFormacion	titular(), mostrar()
Coordinador	semestreCoordinado,areaResponsable	coordinador(), mostrar()

4.1. Relaciones

- Herencia:

  - Estudiante y Docente heredan de la clase base Persona, lo que permite reutilizar los atributos comunes como nombre, ci, y edad.

- Contratado, Titular y Coordinador son subclases de Docente, especializando sus características según el tipo de contrato del docente.

- Composición

  - Estudiante contiene instancias de Registro<Nota> y Registro<Asistencia>: estos registros están íntimamente ligados a la vida del estudiante.
  - ArchEstudiante tiene una relación de composición con Estudiante.
  - ArchDocente tiene una relación de composición con Docente.
  - Estudiante tiene una relación de composición con Registro<T>.

- Agregación

  - En Nota, el atributo docenteEvaluador es un objeto de tipo Docente, lo que indica una agregación: el docente evalúa, pero no depende del objeto nota para existir.
  - Lo mismo sucede con materiasAsignadas[] dentro de Docente y materiasInscritas[] en Estudiante, que son arreglos de strings independientes.

- Interacción entre clases

  - Estudiante y Docente actúan como las clases principales.
  - Los estudiantes y docentes interactúan a través del registro de notas y asistencias.
  - Las clases Nota y Asistencia se registran dentro de objetos Registro<T>, una clase genérica reutilizable.

5. **Desarrollo**

Estructura general del código:

1. **Clases y Subclases:**

Clases:

Persona

Registro<T>

Nota

Asistencia

ArchEstudiante

ArchDocente

Subclases:

Estudiante (Subclase de Persona)

Docente (Subclase de Persona)

Contratado (Subclase de Docente)

Titular (Subclase de Docente)

Coordinador (Subclase de Docente)

2. **Uso de genéricos**

Registro<T>

3. **Métodos sobrescritos**

Método: +mostrar(): void

Clase base: Persona

Clases que lo sobrescriben: Estudiante, Docente, Contratado, Titular, Coordinador

Método: +leer(): void

Clase base: Persona

Clases que lo sobrescriben: Estudiante, Docente

4. Validaciones

Clase Estudiante:

- En el método “adicionarMateria(String materia)” se aplica la validación al verificar si el estudiante ya tiene 7 materias inscritas con la condicional (nroMaterias == 7), en caso de que se exceda, muestra el mensaje "Maximo de materias inscrito".

Clase ArchEstudiante:

- En el método “crear()” se aplica la validación al verificar si el archivo ya existe con f.exists(), y en caso afirmativo muestra el mensaje "El archivo ya existe."
- En el método “eliminar(int regUni)” se aplica la validación al comparar el registro universitario con “est.getRegistroUniversitario() == regUni”, eliminando solo si hay coincidencia.
- En el método “modificar(int regUni)” se aplica la validación al buscar al estudiante por su registro universitario antes de proceder con la modificación.

Clase ArchDocente:

- En el método “crear()” se aplica la validación al verificar si el archivo ya existe mediante f.exists().
- En el método “eliminar(int ci)” se aplica la validación al comparar el CI del docente con “doc.getCi() == ci”, permitiendo la eliminación sólo si coincide.
- En el método “modificar(int ci)” se aplica la validación al buscar al docente por su CI antes de realizar la modificación.

Clase Registro<T>:

- En el método “agregar(T elemento)” se aplica la validación al verificar si hay espacio disponible en el arreglo con la condición (nroReg < vecReg.length), y

en caso contrario muestra el mensaje "No se puede agregar más elementos. Capacidad llena."

5. **Persistencia con archivos (.txt o .json)**

   El sistema implementa la persistencia de datos mediante archivos binarios (.dat) utilizando serialización de objetos en Java. Las clases ArchEstudiante y ArchDocente se encargan de gestionar el almacenamiento de información, empleando ObjectOutputStream para guardar los objetos serializados y ObjectInputStream para recuperarlos.

6. **Aplicación de Patrones de Diseño**

Para el desarrollo de la interfaz gráfica del sistema, aplicamos el **patrón de diseño Singleton** con el objetivo de garantizar que **solo exista una instancia** de la ventana principal durante la ejecución del programa.

Esto es especialmente útil en entornos donde se requiere evitar la creación de múltiples ventanas redundantes, lo que podría afectar la usabilidad o generar errores lógicos en la interacción con el usuario.

El patrón Singleton se implementó en la clase **`SistemaAcademicoGUI`** asegurando que el constructor sea privado por lo cual proporcionamos un método estático que devuelve la única instancia de nuestra clase.

Ventajas obtenidas:

1. **Control de instancias**: se evita que se creen múltiples interfaces duplicadas.

2. **Ahorro de memoria**: se usa una única instancia global.

3. **Consistencia**: todas las partes del programa acceden a la misma GUI.

7. **Persistencia de datos**

   7.1 Descripción del Formato Usado

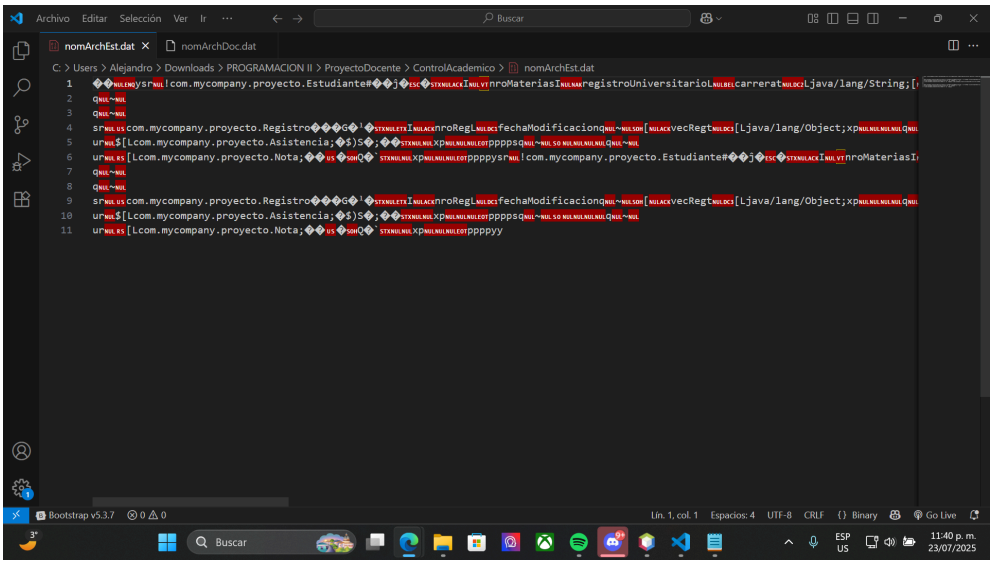
   Se utilizó el formato binario, usando la serialización de objetos mediante ObjectOutputStream para la escritura y la lectura(EOFException), la extensión de los archivos realizados donde se genera la persistencia es “**.**dat”.

7.2 Clase encargada de lectura/escritura

CLASE	ARCHIVO QUE GESTIONA	MÉTODO CLAVE
ArchEstudiante	estudiantes.dat	crear(), adicionar(), listar(), modificar(), buscar()
ArchDocente	docentes.dat	crear(), adicionar(), listar(), modificar(), buscar()

7.3 Ejemplo de Archivo Generado

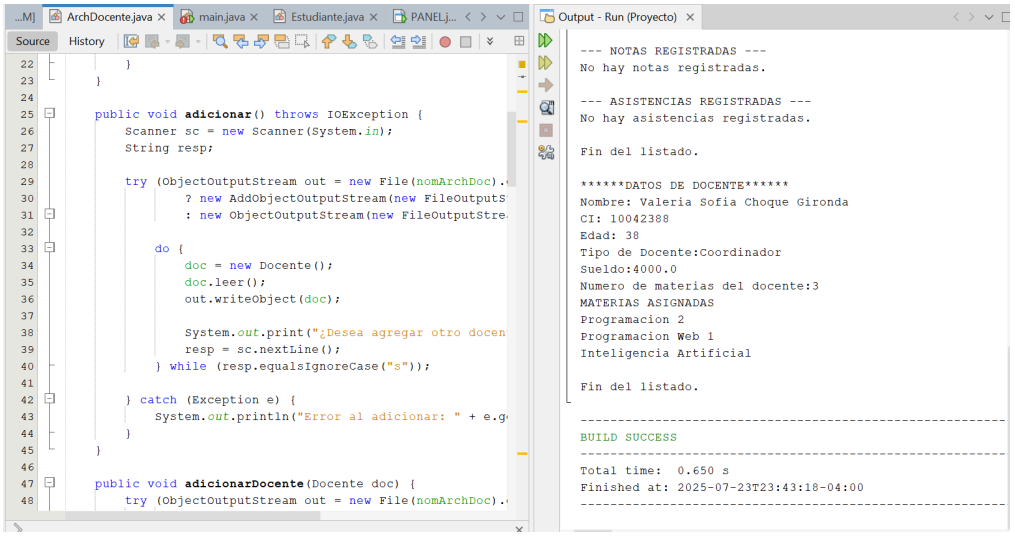
En este punto se aplicó, la información de Estudiantes y Docentes donde se guarda el archivo en binario, utilizando la serialización de objetos, la cual eso nos permite que cada objeto se transforme en una secuencia de datos. Para ello se utiliza ObjectOutputStream, que se encarga de escribir estos datos en un archivo de extensión .dat.



8. Pruebas y validación

Casos de prueba realizados:

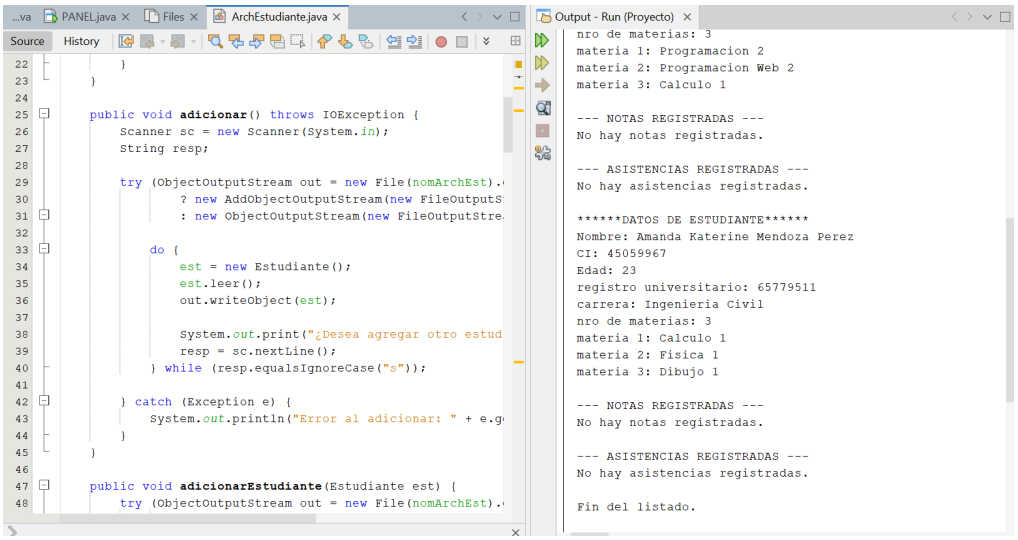
ArchDocente:



```
...M] ArchDocente.java × main.java × Estudiante.java × PANEL.j...
Source History
22          }
23      }
24
25      public void adicionar() throws IOException {
26          Scanner sc = new Scanner(System.in);
27          String resp;
28
29          try (ObjectOutputStream out = new File(nomArchDoc).
30                  ? new AddObjectOutputStream(new FileOutputS
31                  : new ObjectOutputStream(new FileOutputStre
32
33              do {
34                  doc = new Docente();
35                  doc.leer();
36                  out.writeObject(doc);
37
38                  System.out.print("¿Desea agregar otro docen
39                  resp = sc.nextLine();
40              } while (resp.equalsIgnoreCase("s"));
41
42          } catch (Exception e) {
43              System.out.println("Error al adicionar: " + e.g
44          }
45      }
46
47      public void adicionarDocente(Docente doc) {
48          try (ObjectOutputStream out = new File(nomArchDoc).
```

```
Output - Run (Proyecto) ×

--- NOTAS REGISTRADAS ---
No hay notas registradas.

--- ASISTENCIAS REGISTRADAS ---
No hay asistencias registradas.

Fin del listado.

******DATOS DE DOCENTE******
Nombre: Valeria Sofia Choque Gironda
CI: 10042388
Edad: 38
Tipo de Docente:Coordinador
Sueldo:4000.0
Numero de materias del docente:3
MATERIAS ASIGNADAS
Programacion 2
Programacion Web 1
Inteligencia Artificial

Fin del listado.

------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------
Total time:  0.650 s
Finished at: 2025-07-23T23:43:18-04:00
------------------------------------------------------------
```

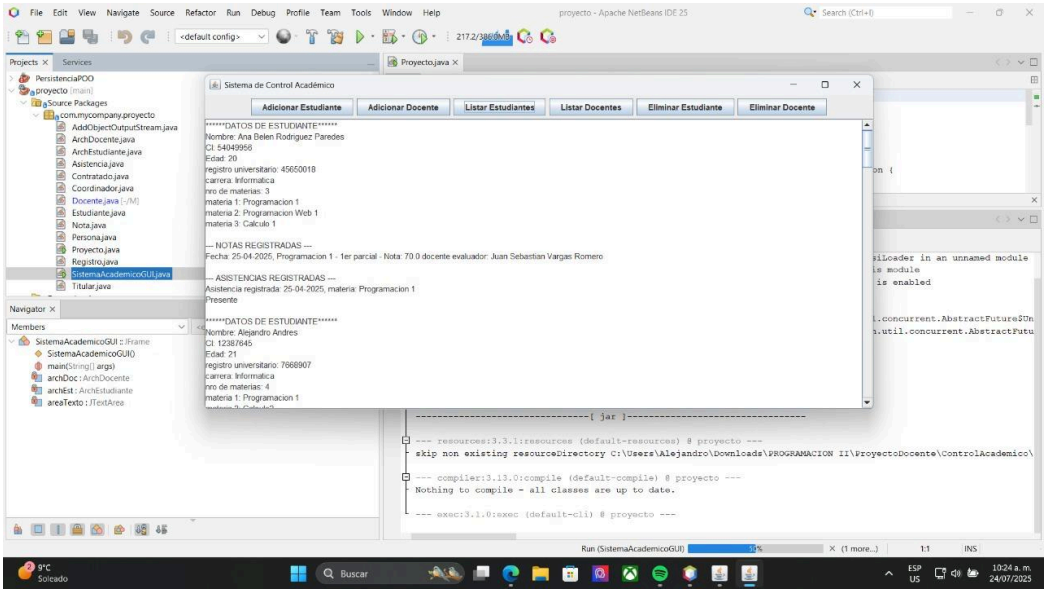
ArchEstudiante:

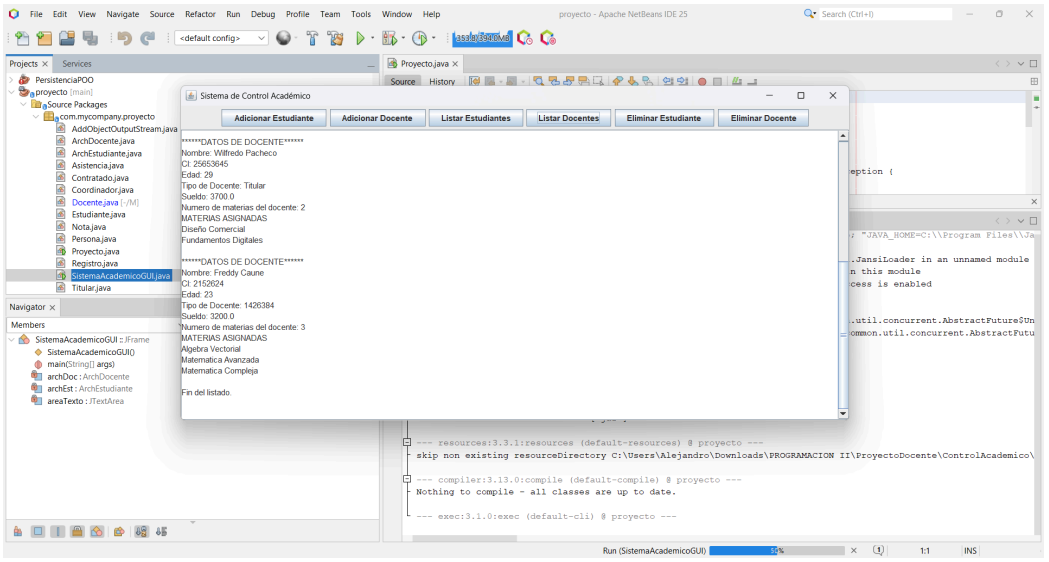


```
...va PANEL.java × Files × ArchEstudiante.java ×
Source History
22          }
23      }
24
25      public void adicionar() throws IOException {
26          Scanner sc = new Scanner(System.in);
27          String resp;
28
29          try (ObjectOutputStream out = new File(nomArchEst).
30                  ? new AddObjectOutputStream(new FileOutputS
31                  : new ObjectOutputStream(new FileOutputStre
32
33              do {
34                  est = new Estudiante();
35                  est.leer();
36                  out.writeObject(est);
37
38                  System.out.print("¿Desea agregar otro estud
39                  resp = sc.nextLine();
40              } while (resp.equalsIgnoreCase("s"));
41
42          } catch (Exception e) {
43              System.out.println("Error al adicionar: " + e.g
44          }
45      }
46
47      public void adicionarEstudiante(Estudiante est) {
48          try (ObjectOutputStream out = new File(nomArchEst).
```

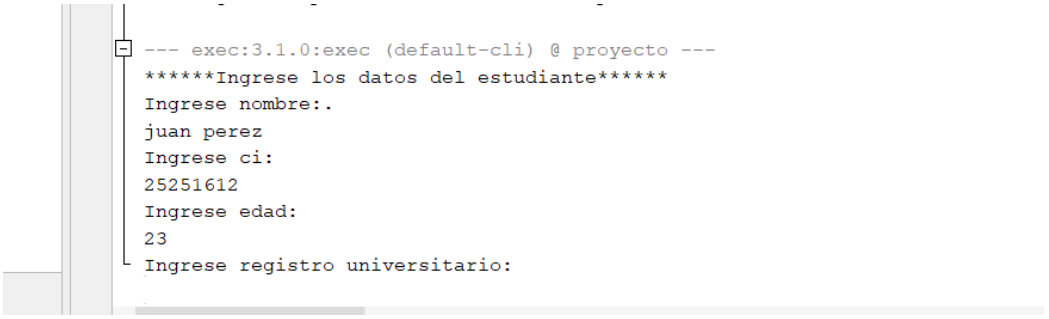
```
Output - Run (Proyecto) ×
nro de materias: 3
materia 1: Programacion 2
materia 2: Programacion Web 2
materia 3: Calculo 1

--- NOTAS REGISTRADAS ---
No hay notas registradas.

--- ASISTENCIAS REGISTRADAS ---
No hay asistencias registradas.

******DATOS DE ESTUDIANTE******
Nombre: Amanda Katerine Mendoza Perez
CI: 45059967
Edad: 23
registro universitario: 65779511
carrera: Ingenieria Civil
nro de materias: 3
materia 1: Calculo 1
materia 2: Fisica 1
materia 3: Dibujo 1

--- NOTAS REGISTRADAS ---
No hay notas registradas.

--- ASISTENCIAS REGISTRADAS ---
No hay asistencias registradas.

Fin del listado.
```

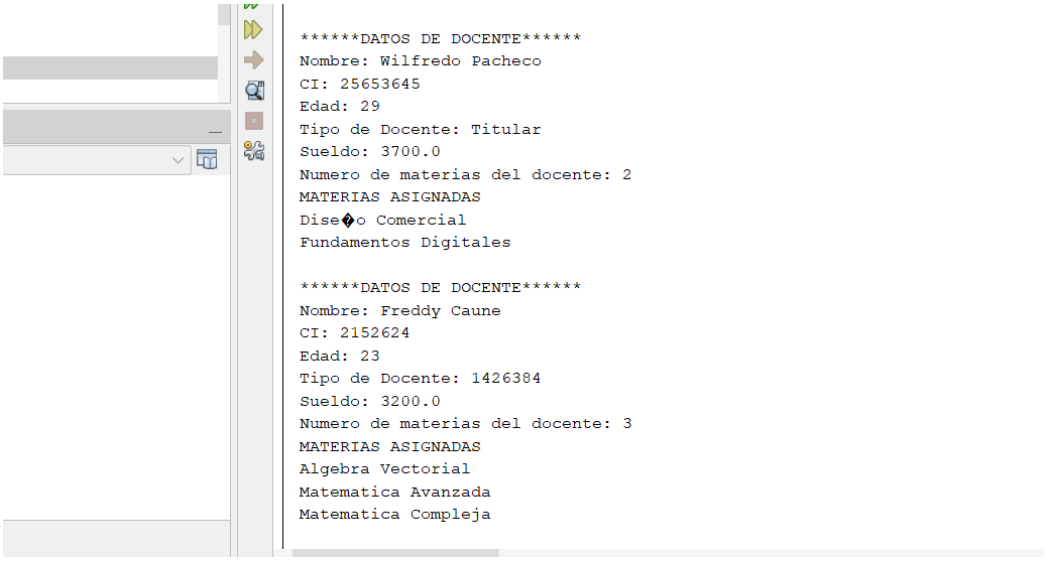
Capturas de pantalla de la Interfaz:





Ejemplos de entradas y salidas esperadas:





9. Conclusiones

- **Reflexionar sobre el uso de patrones y cómo facilitaron el diseño.**

El desarrollo del proyecto Control Académico facilitó aplicar de forma práctica los conceptos de la Programación Orientada a Objetos.
La organización del código facilito la comprensión y reutilización para lograr una estructura funcional como la clase base Persona de la cual heredan Estudiante y Docente permitiendo compartir atributos, la clase Registro<T>facilitando el orden y reutilización.Este proyecto no solo ayudó a reforzar los conocimientos aprendidos durante la materia, sino que también permitió entender por qué es importante crear programas bien organizados, claros y fáciles de mejorar en el futuro.

- **Ventajas observadas en la estructura del sistema.**

La estructura de la Programación orientada a objetos, junto con la división en clases específicas según el diagrama UML, permitió un desarrollo ordenado y una gestión clara de las responsabilidades de cada clase.

- **Qué se aplicó de los contenidos de la materia**

Así también durante el desarrollo, se aplicó contenidos clave de la materia en el desarrollo del proyecto se aplicaron varios conceptos fundamentales vistos en la materia, tales como programación orientada a objetos, diseño de clases y diagramas UML, manejo de archivos y patrones de diseño básicos.
Jerarquía de Clases Clara:

La herencia (Persona → Estudiante/Docente → Tipos de Docente) permitió reutilizar atributos y métodos.

Persistencia Flexible:
Las clases ArchEstudiante y ArchDocente manejan archivos de forma genérica, aunque podrían mejorarse con DAO.

Componentes Desacoplados:
La clase Registro<T> separa la gestión de notas/asistencias de la lógica de Estudiante.

- **Qué se podría mejorar o ampliar**

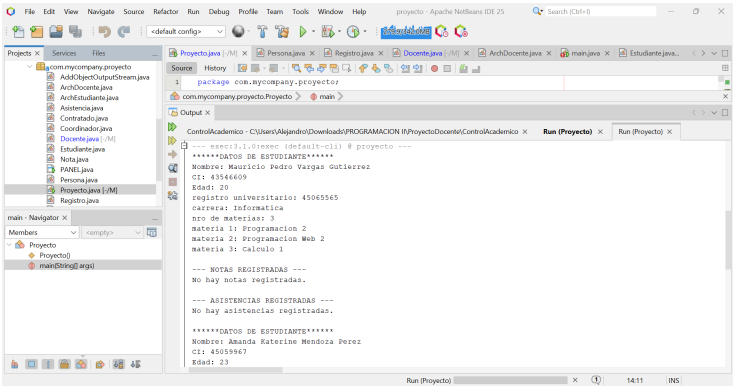
Para futuros desarrollos se podría implementar una interfaz gráfica para mejorar la interacción con el usuario y hacer el sistema más amigable. También se podrían aplicar patrones de diseño más avanzados para optimizar la estructura y facilitar la integración de nuevas

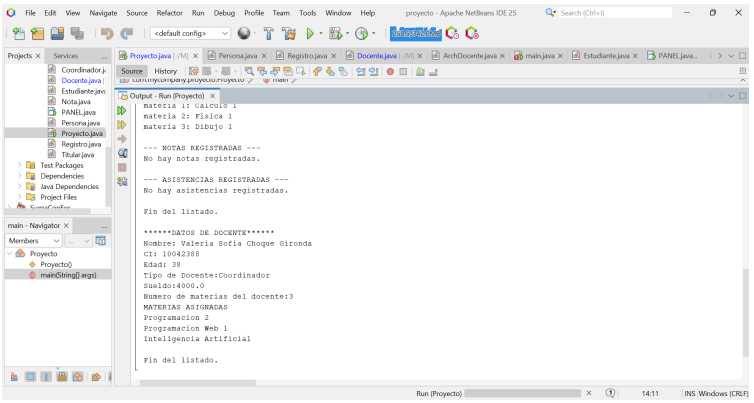
10. Distribución de roles de equipo

ROL/INTEGRANTE	RESPONSABILIDAD PRINCIPAL
Bryan Daniel Sandoval Quenta (Desarrollador Principal, logica)	Encargado de las clases Estudiante y control académico. Apoyo en persistencia de datos, Diagrama UML, Interfaz, apoyo en la creación del informe.
Delma Fernanda Choque Valencia (Desarrolladora de jerarquía docente	Desarrollo de las clases Docente y Coordinador. Participación en persistencia y pruebas, apoyo en la creación del informe.
Alejandro Andrés Yujra Paye (Desarrollador de base Estructural)	Implementación de la clase Persona y la genérica Registro<T>. Manejo de estructuras, apoyo en la creación del informe.
Grissel Noemí Tinta Lopez (Desarrolladora de clases docentes)	Implementación de las clases Contrastado y Titular. Colaboración en el diseño UML, apoyo en la creación del informe.
Lizbeth Quispe Lopez (Desarrolladora de módulos de evaluación)	Desarrollo de las clases Asistencia y Nota. Participación en validaciones y pruebas, apoyo en la creación del Informe.

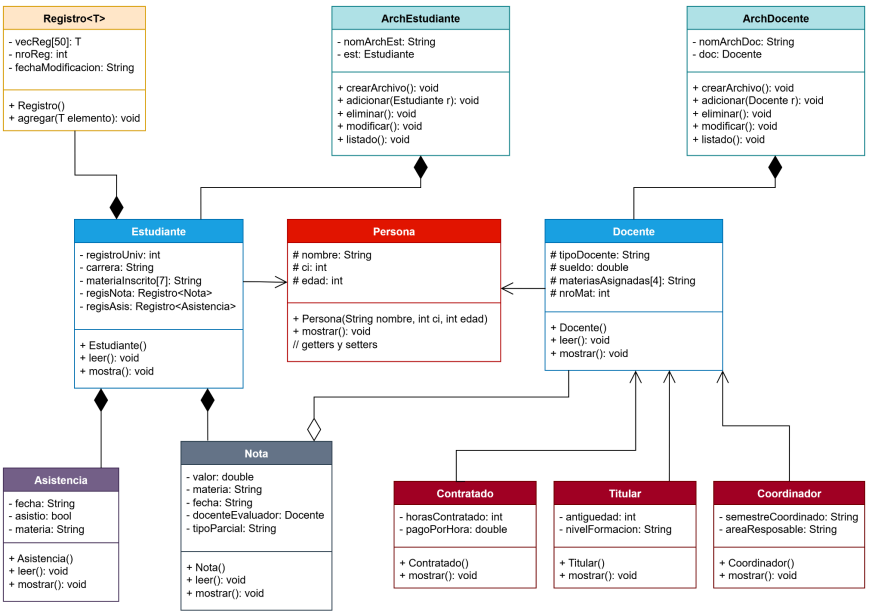
11. Anexos

- **Archivos de prueba:**





- **Diagramas UML:**



– **Link de repositorio GitHub:**

https://github.com/Bryan-Daniel-Sandoval-Quenta/ControlAcademico