DOCUMENTATION WORKSHOP-4 PROJECT OBJECT-ORIENTED PROGRAMMING: NOTEHUB (NOTES APP)

Juan Carlos Córdoba Asprilla – 20242020047

Sebastián Camilo Sánchez Cárdenas- 20242020086

Junio de 2025

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Object-oriented Programming

Ing. Carlos Andrés Sierra

INTRODUCTION

Note Hub is a note-taking application project born from the need to create an environment in which to organize basic aspects of daily life. The inspiration for its creation was taken from the already world-renowned note-taking application Notion. Our goal is to take the fundamental functionalities of the Notion application ecosystem (Notes, Calendar, Reminders, Schedule Creator) and unify them into a single app that will also be easier to use. To achieve this, the aim is to make a simple design of the graphical interface of the app and give a deep focus to the functionalities.

OBJECTIVES

To ensure that the project's direction is as desired, a series of objectives were established.

GENERAL OBJETIVE

Develop a mobile app that offers combined note-taking, reminder, schedule creation, and annotation functionality, accessible offline, compatible with Android devices, and lightweight and intuitive for all types of users.

SPECIFIC OBJETIVES

- Implement a note-taking system that allows users to create, edit, and manage up to 100 notes without visible errors.
- Incorporate reminder and calendar annotation creation, allowing users to add events with dates and descriptions and correctly notify them.
- Implement a system that creates and manages a customizable weekly schedule.
- Design four different sections (Notes, Reminders, Schedule, Calendar) within the app, with a clear and easy-to-navigate interface.
- Separation of the system into layers (GUI, logic, data).
- Design of a persistence mechanism.

REQUERIMENTS

Functional Requirements

- The app must allow users to create, edit, and delete notes, reminders, calendar events, and schedules.
- The system must allow interaction through a graphical interface based on Java Swing.
- Visually show how actions are completed within the app.
- correct operation without internet

Non-functional Requirements

- Performance:

Each operation must respond in less than 2 seconds to ensure a smooth experience.

- Scalability:

Support at least 100 active notes and simultaneous use of multiple functions without visible degradation. The system must allow interaction through a graphical interface based on Java Swing.

- Minimal learning curve:

Clean, minimalist interface with a learning time of less than 5 minutes.

- Compatibility: The app must operate correctly on low- and high-end Android devices, without requiring special hardware.

TECHNICAL INFORMATION OF THE PROJECT AND DESIGN

USER STORIES

User stories establish the basic needs of certain types of users and in turn propose how the program should meet them and what their behavior should be when they interact.

User history	**TITLE** Student Work	**PRIORITY** **HIGH**
	As: Student	
	I want: create a note about an activity they left at my university	
	To: remember to do it	
Acceptance Criteria:	**Given** that I am in the note's application	
	When I enter a title and content for the new note	
	Then The note must be saved correctly and must be visible in its respective place	

User history	**TITLE** remind	**PRIORITY** **HIGH**
	As: Runner	
	I want: Create a reminder	
	To: remind me to get my leg checked after a marathon.	
Acceptance Criteria:	**Given** that I already got my leg checked,	
	When I log into the app and check the reminder,	
	Then the reminder should disappear once it's checked as complete.	

User history	**TITLE** Event	**PRIORITY** **HIGH**
	As: Event Planner	
	I want to remove certain reminder notes.	
	To: organize my future events	
Acceptance Criteria:	**Given** that I selected a note or reminder for deletion Then confirming the deletion, **Then** The note or task should no longer appear in the app.	

User history	**TITLE** Happy Birthday	**PRIORITY** **LOW - MID**
	As: Boss	
	I want to put the dates I'll pay my employees on the annual calendar.	
	To: be able to know exactly how much time is left until I prepare their payments.	
Acceptance Criteria:	**Given** that I added the notes to the calendar, **When** I log out and log back in to the app, **Then** I should still be able to see the dates I wrote down.	

User history	**TITLE** Reminders	**PRIORITY** **HIGH**
	As: Event Planner	
	I want to re program postponed event.	
	To: remind myself of the new date I need to attend.	
Acceptance Criteria:	**Given** I'm in the reminders option **When** I select the reminder where I listed my event again, **Then** the content should be updated to the last time I rewrote it.	

User history	**TITLE** Events	**PRIORITY** **MID-HIGH**
	As: User	
	I want to mark special dates or events on the calendar	
	To: don't forget important dates	
Acceptance Criteria:	**Given** that I want to add an event **When** I enter a date and description **Then** the event should be saved and visible in the integrated calendar.	

User history	**TITLE** schedule	**PRIORITY** **HIGH**
	As: Student	
	I want to create my class schedule	
	To: Remember what days I have class	
Acceptance Criteria:	**Given I** am in the schedules section **When** I log in to the app **Then** the information provided is reflected in the weekly calendar and should remain there every time I log in or out of the app.	

CRC CARDS

The CRC cards are a tool that helps define the responsibilities of the program's classes as well as the classes that collaborate with it to carry out certain functionalities correctly.

CLASS (Application)	
RESPÓNSABILITY	COLABORATOR
• Stores the contents of features such as notes or reminders and handles content editing functions within the app and displays the graphical interface	• Interface(content)

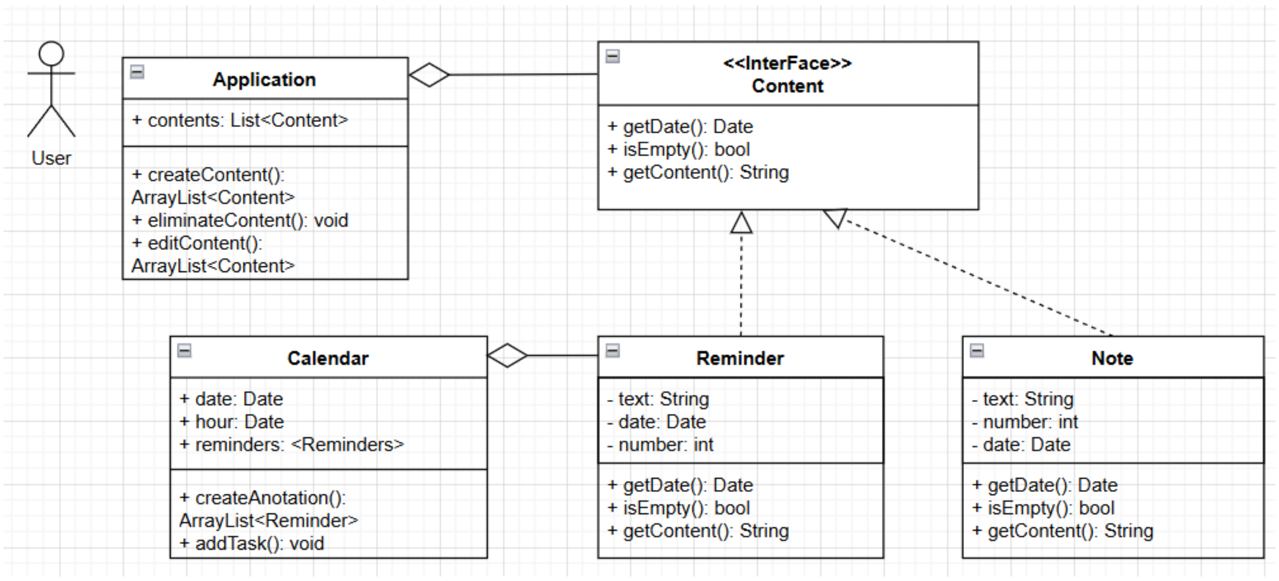
CLASS (Note)	
RESPÓNSABILITY	COLABORATOR
• Represents a single note • Stores and obtain the info of a note	• Interface(content) • application

CLASS (Calendar)	
RESPÓNSABILITY	COLABORATOR
• represents a calendar in which notes on its dates are stored	• Reminder

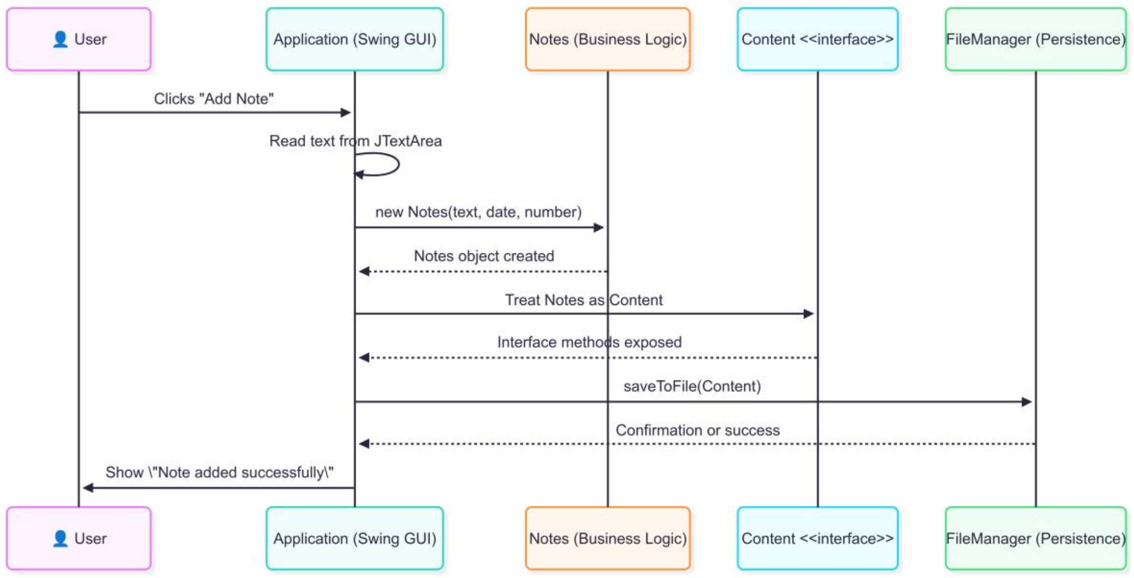
CLASS (Reminder)	
RESPÓNSABILITY	COLABORATOR
• Represents a single reminder • Stores and obtain the info of a reminder	• Calendar • Interface(content) • application

INTERFACE(Content)	
RESPÓNSABILITY	COLABORATOR
• Provides note and reminder classes with the methods they need to define their own content.	• Reminder • note

Class diagram



Sequence diagram (GUI and Logic)



SOLID principles implementation

1. Single Responsibility Principle (SRP)

Each class has a single responsibility.

- Note simply represents a note (text, number, date).
- Reminder simply represents a reminder.
- Calendar manages a calendar with notes (a list of reminders).
- Application handles everything related to creating, editing, or deleting content.

2. Open/Closed Principle (OCP)

The program must be open for extension and closed for modification.

With the Content interface, new classes (such as Event, Task, etc.) can be created by inheriting from the interface, so they don't interfere with or require modification of other classes.

3. Liskov Substitution Principle (LSP)

Subclasses can replace super classes

Since the "content" interface is an abstract data type, and since note and reminder are classes that inherit all methods from this interface and become a concrete data type, both classes can function the same if the content data type needs to be replaced with a concrete data type.

For example, an application receives a list of "content," which is an abstract data type. This content can currently be a note or a reminder.

The important thing about this is that it can be a reminder or a note. Either class can replace the content data type within the application data list because they have the same structure and behavior, and they would not affect the operation of the application class.

4. Interface Segregation Principle (ISP)

The Content interface doesn't implement methods that aren't useful to classes that inherit from it.

It only declares three useful methods and doesn't force classes to implement things they don't use, nor does it overload the interface with multiple methods.

5. Dependency Inversion Principle (DIP)

High-level classes depend on abstractions, not concrete implementations.

The Application class is not only required to receive the specific parameters of notes or reminders, but is also built around the "content" abstraction, which it receives from the "content" interface. This suggests that it can receive content-type data within its list, which could be a note, a reminder, or, in future cases, other types of content such as tasks. There will be no need to modify the application to accept this type of data.

PROGRAM STRUCTURE

Layered Architecture Overview

The application was structured following a layered architecture model, organizing the system into three main layers:

- **Presentation Layer**: This layer was developed using Java Swing and is responsible for the graphical user interface. The `Application` class manages the GUI components, including windows, tabs, buttons, and text areas that allow the user to interact with the system.

- **Business Logic Layer**: This layer includes the core classes that define the application logic, such as `Notes`, `Reminder`, and `Calendar`. These classes implement the `Content` interface, ensuring standardized behavior across different content types.

- **Data Access Layer (in progress)**: Currently, a simple `List<String>` is used to store notes in memory , in a future update, a more developed file persistence will be implemented.

Java Swing GUI Implementation

The graphical user interface developed in the Application class performs the following functions:

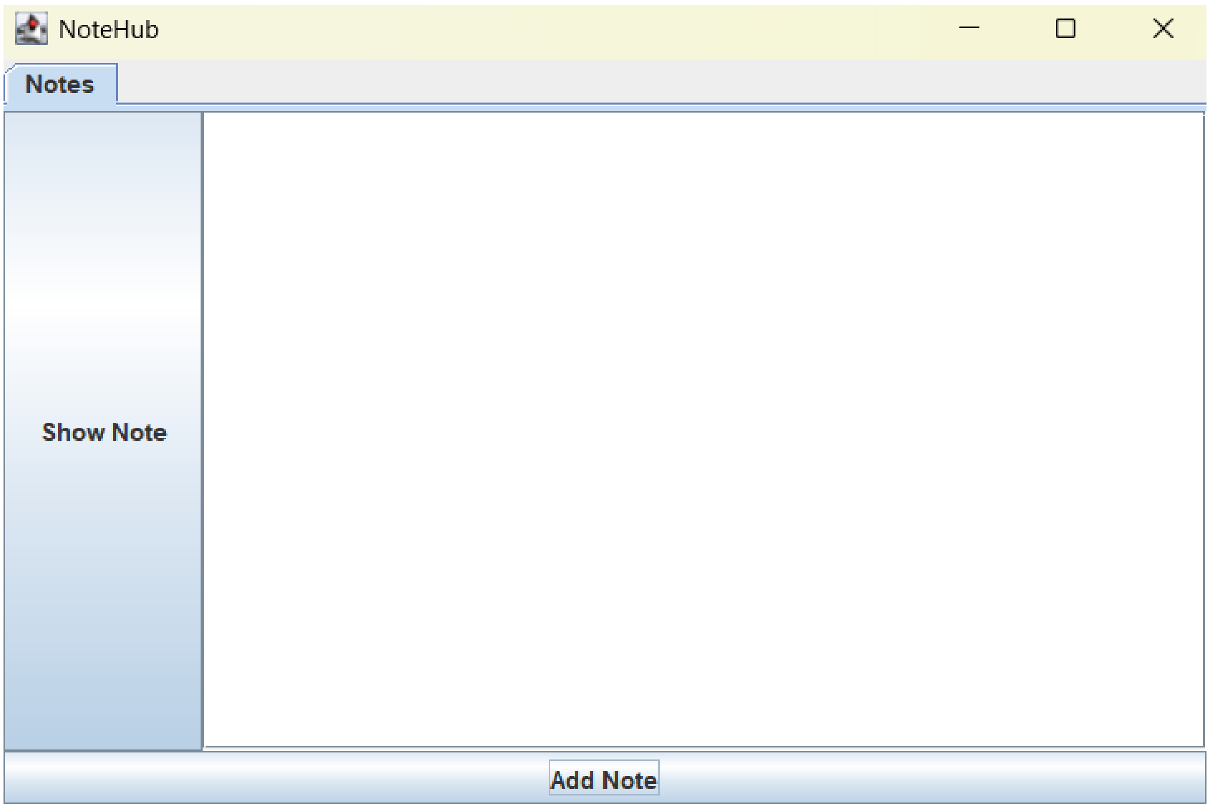
- A window-style interface (JFrame) with tabs (JTabbedPane).

- A note-writing area using JTextArea with a JScrollPane.

- An “Add Note” button that adds the text content to the system.

- A “Show Note” button that displays the written content in the console.

The focus was placed on functionality over visual design


NoteHub
Notes
Show Note
Add Note