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

Juan Carlos Córdoba Asprilla – 20242020047

Sebastián Camilo Sánchez Cárdenas- 20242020086

July 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 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 a series of objectives were established.

SPECIFIC OBJETIVES

- Structure the application into four distinct sections (Notes, Reminders, Schedule, Calendar) to provide a clear, organized, and easy-to-navigate interface.

- Develop a note-taking system that allows users to create, edit, and manage a variety of notes, ensuring efficient information management without visible errors through an intuitive interface.

- Implement a calendar reminders and annotations module so users can record events with dates and descriptions, ensuring proper notification through the system.

- Design a customizable weekly schedule creation and management system that facilitates user time management by integrating simple editing functions.

- Separate the system architecture into layers (graphical interface, business logic, and data access) to improve software maintainability and scalability.
- Design a data persistence mechanism that ensures reliable storage and retrieval of user information throughout the application's use.

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 when actions are completed within the app.
- The application must continue to function properly even if the device on which it is running does not have an internet connection.

Non-functional Requirements

- Graphical Environment:
The system must allow interaction through a graphical interface based on Java Swing.

- Performance:
Operations within the desktop application must have a low response time.

- Multitasking:
Allow multiple system functions to be performed simultaneously.

- Minimal learning curve:
The interface must be clean and minimalist, allowing the user to fully learn how it works in a short time.

- Compatibility:
The application must work correctly on devices running Windows operating systems.

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
– Display graphical interface with text area and buttons – Create content objects (Notes, Reminder) – Manage button actions and visual layout	Note, Reminder, Content

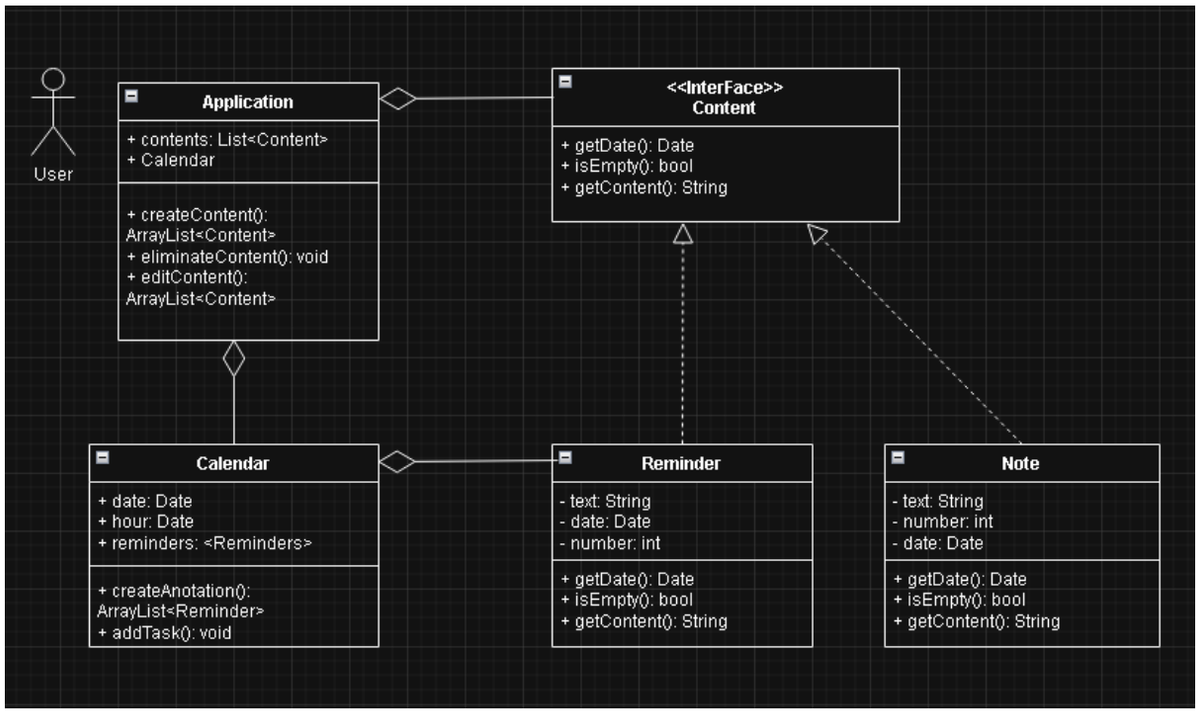
INTERFACE(Content)	
RESPÓNSABILITY	COLABORATOR
– Define the abstraction for content types – Provide methods for the Note and Reminder classes through inheritance	Notes, Reminder, Application

CLASS (Note)	
RESPÓNSABILITY	COLABORATOR
– Store note text, date, and ID – Provide formatted note content – Implement Content interface – Allow checking if the note is empty	Application, Content

CLASS(Reminder)	
RESPÓNSABILITY	COLABORATOR
– Store reminder text, date, and ID – Provide content and date information – Implement Content interface – Allow validation of empty state	Content, Calendar, Application

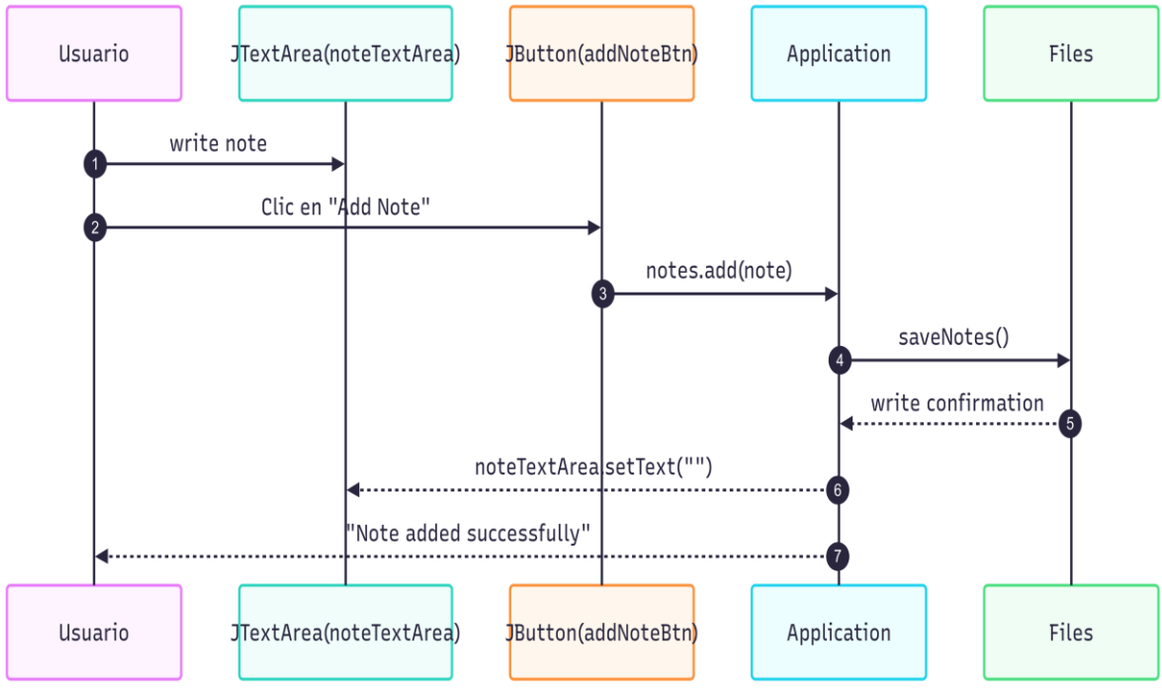
CLASS (Calendar)	
RESPÓNSABILITY	COLABORATOR
– Store a list of reminders – Add tasks (reminders) to the calendar	Reminder, Application

Class diagram



Sequence diagrams (GUI and Logic)

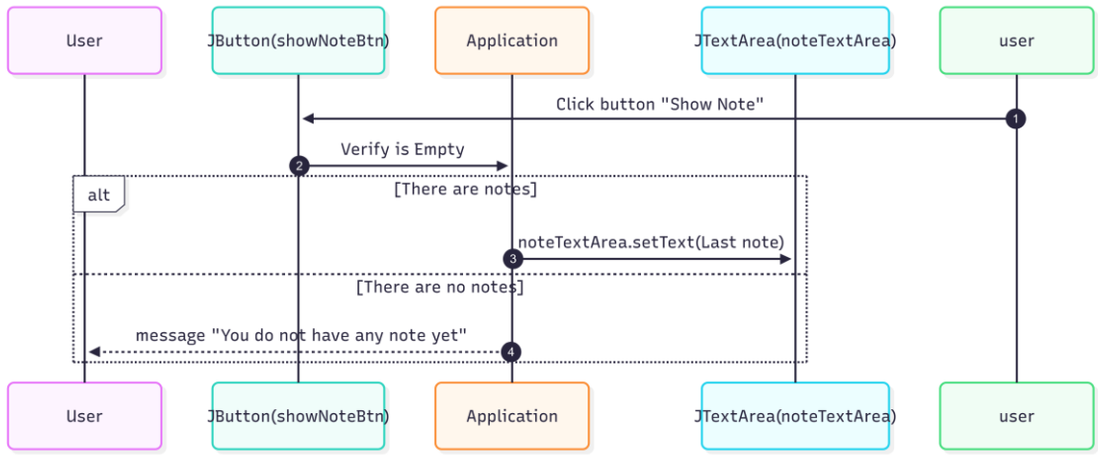
Sequence diagram Create Note



Sequence:

1. The user writes a note in the text area.
2. The user clicks the **"Add Note"** button.
3. The note is added to the `notes` list in the `Application`.
4. The `saveNotes()` function is called to write the updated list to the `notes.txt` file.
5. If saved successfully:
    a. The text area is cleared (`noteTextArea.setText("")`).
    b. A message is displayed: **"Note added successfully"**.

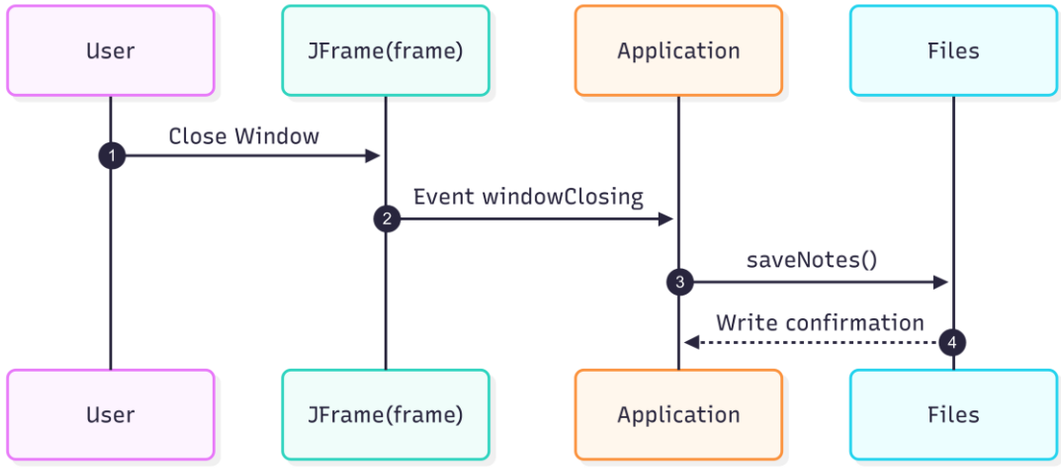
Sequence diagram Show Notes



Sequence:

1. The user clicks the **"Show Note"** button.
2. The system checks if the `notes` list is not empty.
    If there are notes:
        The last note is displayed in the text area.
    If not:
        A message appears: **"You do not have any note yet"**.

Sequence diagram Close application and save notes



Sequence:

1. The user closes the application window.
2. This triggers the `windowClosing` event.
3. The `Application` calls `saveNotes()` to persist the data.
4. Once saved, a confirmation is assumed.

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**: This layer was created to store the program files in a .txt

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.

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

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

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

The focus was placed on functionality over visual design

File storage

The notes and reminders that are create by the user are going to store in a .txt file

- A “Save note” method that write a file notes.txt with the information that the user gives, every note is save in a line of the file

- A “Load note” method that read all the lines of the file notes.txt and charge in the program

- A “Save reminder” method that write a file reminders.txt with the information that the user gives, every reminder is save in a line of the file

- A “Load reminder” method that read all the lines of the file reminders.txt and charge in the program

USAGE INSTRUCTIONS

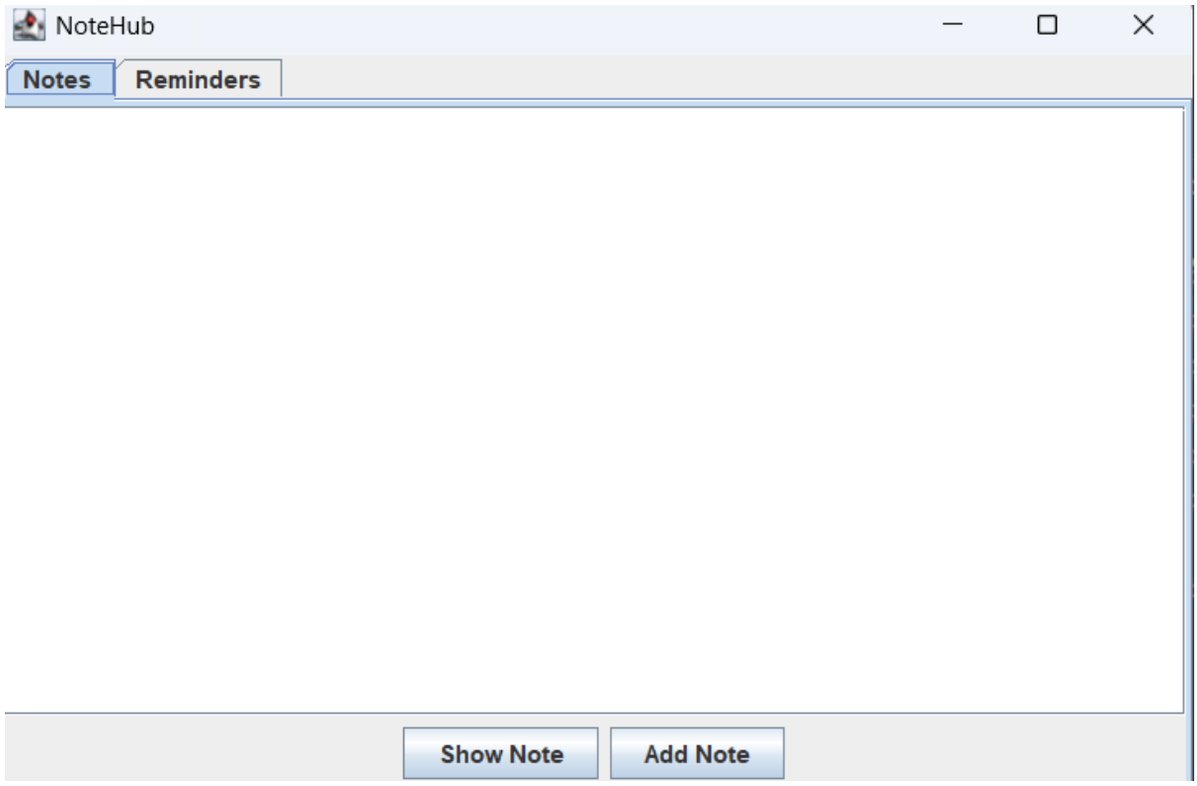
Instructions for downloading and running the program:

https://github.com/3-Sebas-Sanchez/BreezeTask/tree/main/Workshops/Workshop-4#-how-to-run-notehub

Instructions for using the already launched program

Create a note:

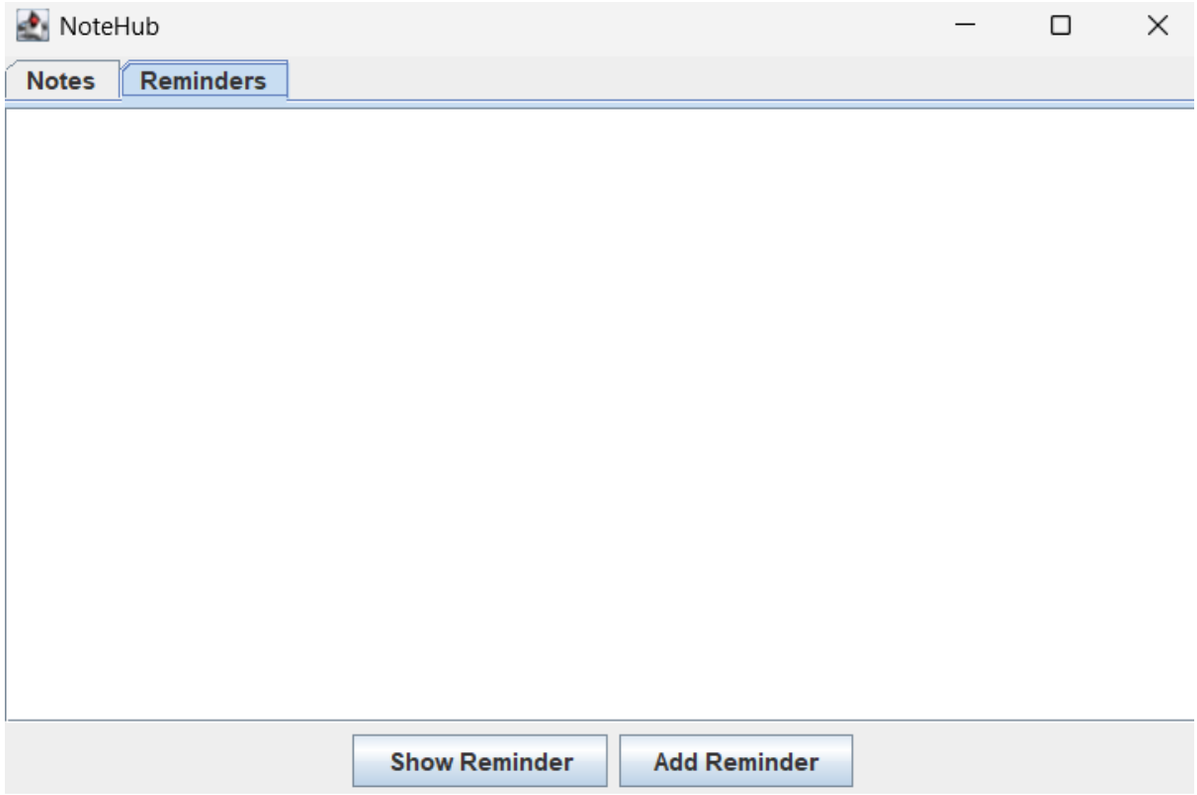
- Go to the "Notes" tab.
- Write the note's content in the text area.
- Click the "Add Note" button to save the note.
- A message will appear indicating that the note was added successfully.



Show the last note:

- Click the "Show Note" button.
- If there are notes available, the last saved note will be displayed.
- If there are no notes, the system will display the message: "You do not have any notes yet."

- Create a reminder:
- Go to the "Reminders" tab.
- Write the reminder in the corresponding area.
- Click "Add Reminder" to save it.
- Show the last reminder:
- Click the "Show Reminder" button.
- The most recent reminder will be displayed if one exists.



Closing the window will automatically save the content.

REFERENCES

Oracle Java Swing Documentation – Java Swing Package javax.swing (Java Platform SE 8)

Oracle Java Documentation – Swing Tutorial https://docs.oracle.com/javase/tutorial/uiswing/

Oracle Java Documentation – Basic I/O https://docs.oracle.com/javase/tutorial/essential/io/