Development of a Desktop Note-Taking App Based on Object-Oriented Programming

Juan Carlos Córdoba Asprilla, Sebastián Camilo Sánchez Cárdenas
Faculty of Engineering - Systems Engineering Program
Universidad Distrital Francisco José de Caldas
Emails: {jccordoba, scsanchez}@udistrital.edu.co

***Abstract*—This paper presents the design and development of a simplified desktop note-taking application inspired by Notion, created using Object-Oriented Programming (OOP) principles. The project aims to provide offline capabilities, modular structure, and core functionalities such as notes, reminders, scheduling, and calendar integration. The paper describes the methodology, design architecture, and implementation process, highlighting the application of OOP principles and the use of CRC cards, UML diagrams, and local storage for persistence.**

***Index Terms*—Object-Oriented Programming, desktop application, note-taking, UML, SOLID, Java Swing**

I. Introduction

In the contemporary digital age, productivity applications have become integral to educational and professional environments. Popular tools like Notion, Evernote, Microsoft OneNote, and Google Keep offer comprehensive platforms for taking notes, organizing information, managing tasks, and collaborating across devices. These tools are widely adopted due to their cloud-based flexibility, rich feature sets, and integration with broader ecosystems. For instance, Notion combines document editing, task management, and relational databases in a unified interface [1]. Microsoft OneNote integrates deeply with the Microsoft Office Suite, enabling synchronized notebooks across devices, while Evernote emphasizes tagging, searchability, and cross-platform access.

Despite their strengths, such platforms often pose challenges for users in academic contexts or low-resource environments. Their steep learning curves, overcomplexity for basic tasks, reliance on internet access, and data storage on third-party servers may be problematic for users seeking simplicity, privacy, or offline functionality. For example, students requiring quick access to class schedules or reminders may not need collaborative features, cloud syncing, or integrations. In addition, minimalistic tools that offer core functionality without distractions are increasingly sought after for focused individual use.

To address these limitations, we propose **NoteHub**, a lightweight, desktop-based note-taking application designed with a clear emphasis on usability, offline operation, and modular extensibility. NoteHub aims to replicate essential functionality such as note creation, local reminders, a calendar for events, and a customizable weekly schedule, all without internet dependency or complex setup. The application targets individual users who value simplicity, privacy, and performance over feature bloat.

The design and implementation of NoteHub are grounded in the principles of Object-Oriented Programming (OOP). These include modular class design, encapsulation, interface abstraction, and polymorphism, which together promote code reuse, separation of concerns, and long-term maintainability. The application is structured around a three-layered architecture: the presentation layer (GUI) built with Java Swing, the business logic layer (handling note and reminder operations), and a data access layer (managing local storage through Java I/O). Java was selected for its portability, rich standard libraries, and familiarity in educational contexts.

Development followed Object-Oriented Analysis and Design (OOAD) techniques, including the creation of CRC (Class-Responsibility-Collaborator) cards, UML class and sequence diagrams, and the application of SOLID principles for clean architecture. These design artifacts supported modular decomposition and enabled collaborative iteration throughout the development process. In particular, the use of interface-based abstraction via a common `Content` interface allowed diverse content types—such as notes and reminders—to share common operations while remaining decoupled.

This paper presents a technical overview of NoteHub's initial development cycle. It describes the problem context, outlines the design methodology, and discusses the system's architecture and features. The contribution lies in demonstrating how educationally guided software engineering techniques can be applied to develop a practical, maintainable tool that solves a real-world problem in a constrained setting.

II. Methods and Materials

A. Development Methodology

The development of NoteHub followed a structured, workshop-based methodology grounded in Object-Oriented Analysis and Design (OOAD). The process was iterative and divided into four key phases: requirements gathering, system design, implementation, and validation. In each phase, specific deliverables were defined to ensure that the application progressed in a modular and testable manner. The early stages focused on identifying the users' real needs through structured *user stories*, which provided a behavioral blueprint of how the application should respond to different interactions. This

This project is part of the Object-Oriented Programming course led by Professor Carlos Andrés Sierra.

narrative-driven approach allowed for a user-centered design, where each functionality—such as creating notes, setting reminders, or saving scheduled events—was traceable to a concrete scenario.

These user stories guided the creation of Class-Responsibility-Collaborator (CRC) cards. This step clarified the responsibilities of each class and the relationships between components. For instance, the `Application` class manages user interface logic, while `Note` and `Reminder` handle data related to user-generated content. Each class in the business logic layer interacts through a common `Content` interface, allowing polymorphic behavior and enabling future extensibility. The use of CRC cards also provided a smooth transition toward Unified Modeling Language (UML) diagrams, which played a critical role in visualizing the architecture.

B. Architecture Overview

NoteHub was built using a classic three-layer architecture, which promotes scalability and clear separation of concerns:

- **Presentation Layer**: This layer was implemented using Java Swing, offering a graphical user interface composed of tabs and panels for organizing different modules such as Notes, Calendar, Schedule, and Reminders. Java Swing components like `JFrame`, `JTabbedPane`, and `JTextArea` provided a robust structure for user interaction.
- **Business Logic Layer**: This layer encapsulates core domain logic. Classes such as `Note`, `Reminder`, and `Calendar` contain the operations necessary to manipulate content. These classes adhere to a shared interface named `Content`, ensuring that any type of content can be processed uniformly.
- **Data Access Layer**: For data persistence, the application uses Java's native `java.io` and `java.nio` libraries. Content such as notes and reminders is written to and read from plain-text files (e.g., `notes.txt`, `reminders.txt`). This file-based approach simplifies storage while supporting full offline functionality.

C. Design Components and CRC Model

Each component was planned and validated using CRC cards. The `Application` class acts as the orchestrator, managing tabs, buttons, and event listeners. The `Note` class is responsible for storing and displaying textual content, while the `Reminder` class handles time-based notifications. Both implement the `Content` interface, which defines methods such as `getContent()` and `getDate()`—allowing polymorphic processing by the application. The `Calendar` class interacts with reminders and offers a basic overview of scheduled events.

This modeling strategy reduced ambiguity during coding and helped align the team around unified responsibilities. By focusing on roles rather than just code structure, each class remained focused and minimal, reducing the risk of feature creep or redundancy.

D. UML and Sequence Diagrams

As the logical foundation of the application solidified, UML diagrams were used to formalize relationships. A UML Class Diagram illustrated inheritance (e.g., `Note` and `Reminder` implementing `Content`), aggregation, and dependencies. Sequence diagrams depicted interactions like saving a note: the user writes a note, clicks "Add," which triggers content validation, file persistence, and UI feedback. These diagrams were created using draw.io and PlantUML, ensuring they could be integrated into both development and documentation workflows.

This visual documentation helped ensure that the software's actual implementation remained faithful to its design. It also played a key role in stakeholder communication and future maintainability.

E. Implementation Tools

NoteHub was developed in Java using Visual Studio Code. The decision to use Java was based on its strong support for object-oriented principles and its mature ecosystem. Version control was managed via Git and GitHub, allowing collaborative development and code history tracking. The graphical interface was built with Java Swing for portability and simplicity. Modeling was done using draw.io for diagrams and Canva for interface mockups.

F. Application of SOLID Principles

Throughout the project, adherence to the SOLID principles of object-oriented programming was a top priority. The **Single Responsibility Principle** was followed by assigning each class a narrowly defined role. For instance, the `Note` class focuses only on note content, without handling UI or persistence. The **Open/Closed Principle** guided the development of the `Content` interface, allowing future content types to be added with minimal changes to existing logic. The **Liskov Substitution Principle** is supported by polymorphism through the interface, ensuring that all content classes can be treated uniformly. The **Interface Segregation Principle** was respected by designing interfaces with minimal required methods, avoiding bloated contracts. Lastly, the **Dependency Inversion Principle** was realized by ensuring that the `Application` class depends on abstractions, not concrete classes.

G. Summary

The methods and materials used in NoteHub's development enabled a well-structured and maintainable application. From story-driven requirements to layered architecture and rigorous modeling, the team was able to prototype a working application that can evolve with future demands. The combination of CRC cards, UML diagrams, and SOLID principles ensured a strong foundation not only for current functionalities but also for future expansions, such as mobile support or more complex scheduling features.

III. Results

The implementation of NoteHub has successfully materialized the conceptual model outlined during the design phase. The application features a functional graphical user interface (GUI) developed in Java Swing, which is divided into modular tabs corresponding to key features: Notes, Reminders, Calendar, and Schedule. Each of these modules is independently operational and interacts seamlessly with the business logic and data persistence layers.

The GUI was tested through manual interaction scenarios, where users could enter textual notes, add and edit reminders, and verify that changes were reflected immediately in both the user interface and the local file system. Notes and reminders are persistently stored using plain-text formats in `notes.txt` and `reminders.txt` files, respectively. This guarantees offline functionality and allows for basic data recovery and continuity across sessions.

In terms of architecture, the separation of concerns across layers proved to be highly effective. The `Application` class manages all interactions with the user through a well-defined control structure. Upon user action, such as clicking the "Add Note" button, the content is processed by the relevant business logic class, validated, and saved through the persistence layer. A successful operation triggers feedback mechanisms in the UI, such as confirmation messages or refreshed content views.

From a design validation perspective, the results also include a complete set of UML diagrams and CRC cards that closely mirror the current implementation. These artifacts were essential in ensuring architectural alignment and were used during team reviews to verify the consistency of logic and class responsibilities.

Furthermore, sequence diagrams were created to document dynamic behaviors such as creating, saving, and displaying notes. These diagrams confirmed that the control flow remained consistent with the intended structure and contributed to maintaining code quality during refactoring.

The system also adheres to SOLID principles, with each class fulfilling a unique responsibility and depending only on abstractions. The usage of the shared `Content` interface across different content types (e.g., `Note`, `Reminder`) facilitated polymorphism and simplified integration testing.

summary of the results:

- A fully functional desktop GUI with modules for notes, calendar, reminders, and scheduling.
- Persistence layer implemented through `.txt` files, guaranteeing offline storage.
- Logical architecture in compliance with SOLID principles and modeled via CRC cards.
- UML Class Diagram and Sequence Diagrams that align with the implementation and validate its modularity.

These results validate the feasibility of developing a lightweight, maintainable, and modular personal productivity application using Java and OOP. While additional features and improvements are planned for the future, the current version already demonstrates the robustness and clarity of the system's foundational structure.

IV. Conclusions

The development of NoteHub demonstrates the viability of constructing a modular, extensible, and user-centric desktop application by applying Object-Oriented Programming (OOP) principles and best practices. Throughout the development process, the use of a structured methodology —including the definition of user stories, CRC cards, and UML models— allowed for consistent design decisions and guided the iterative implementation.

The adoption of a three-layered architecture—consisting of presentation, business logic, and data access layers—enabled a clean separation of concerns, which proved critical in maintaining the scalability and readability of the codebase. This separation ensures that future enhancements or feature extensions can be performed with minimal disruption to the existing system structure.

The core objectives of the application were successfully met. NoteHub supports the creation and management of notes, reminders, calendar events, and weekly schedules within a lightweight and offline-capable desktop environment. Persistent storage was achieved using plain-text files, which are simple to manage and validate while allowing the application to function without network dependencies. The system also responds consistently to user interactions through a responsive Java Swing interface.

Moreover, the project served as a practical implementation of OOP design principles.This reinforces not only the maintainability of the system but also its adaptability to future changes or platform migrations.

An important outcome of this project was the ability to apply theoretical programming concepts to a real-world application scenario. The discipline of documenting requirements, planning architecture with UML, and mapping responsibilities through CRC cards created a solid foundation from which development could proceed with clarity and purpose.

In conclusion, NoteHub accomplishes its initial goal of offering a simplified alternative to more complex note-taking applications by focusing on usability, offline access, and architectural soundness. While certain advanced features remain as future improvements, the current state of the system validates the effectiveness of the chosen design and implementation strategy.

Acknowledgment

We thank Professor Carlos Andrés Sierra for his guidance and support throughout the development of this project.

REFERENCES

[1] Notion Labs Inc., *Notion (Version 2.0) [Desktop App]*, 2025. Available: https://www.notion.so

[2] freeCodeCamp, “Los principios SOLID explicados en español,” 2023. Available: https://www.freecodecamp.org/espanol/news/los-principios-solid-explicados-en-espanol/

[3] Styde.net, “Concurrencia y persistencia en programación orientada a objetos,” 2022. Available: https://styde.net/concurrencia-y-persistencia-en-programacion-orientada-a-objetos/

[4] Oracle, “The Java Tutorials – Creating a GUI With Swing.” Available: https://docs.oracle.com/javase/tutorial/uiswing/

[5] draw.io, “UML Class Diagrams – draw.io Blog.” Available: https://www.drawio.com/blog/uml-class-diagrams