GetClasses: A Layered JavaFX-Based Academic Tutoring Platform Applying SOLID Principles

Jhon Gonzalez 20251020087, Alejandro Escobar 20251020094, Sebastián Zambrano 20251020102
Computer Engineering Program
Universidad Distrital Francisco José de Caldas
Bogotá, Colombia December 10, 2025
Emails: jhon@example.com, alejandro@example.com, sebastian@example.com

***Abstract*—Remote academic support has become increasingly important as students seek affordable, accessible, and reliable tutoring services. However, many existing platforms are either overly complex, costly, or require infrastructure that smaller institutions cannot sustain. This paper presents GetClasses, a lightweight academic tutoring platform developed as a monolithic JavaFX application. The system adopts a layered architecture and applies SOLID principles to improve maintainability, modularity, and software quality. GetClasses enables students to search for tutors, manage profiles, communicate through messaging, and submit reviews, while tutors can organize schedules and display academic qualifications. Evaluation through unit tests, interface validation, and mockup-based usability analysis demonstrates that the platform provides stable performance, consistent data handling, and an intuitive user experience. The results indicate that GetClasses is a viable solution for academic environments that require a simple yet effective tutoring management system without high infrastructure demands.**

I. Introduction

The growing demand for affordable, flexible, and reliable academic support has reshaped how students access tutoring services. Many learners struggle to find qualified tutors who align with their academic needs, budget, or availability, while tutors often lack a streamlined way to promote their skills, manage schedules, and interact with students. Traditional tutoring platforms tend to be expensive, require complex infrastructure, or prioritize large-scale institutional use, leaving smaller organizations and independent learners without accessible solutions. This mismatch between user needs and existing offerings highlights a clear gap in the academic tutoring market.

Several digital-learning systems and marketplace-style applications have attempted to address this challenge by offering search filters, messaging tools, or course listings. However, many solutions present technical barriers—such as multi-service backends, heavy interfaces, or limited customization—that complicate adoption by smaller institutions or beginner developers. Prior approaches also reveal issues related to maintainability, excessive coupling between components, and lack of clarity in user-role management, all of which reduce the long-term scalability of the system.

To address these limitations, this paper introduces GetClasses, a lightweight tutoring management platform designed as a simple and accessible digital solution. The system adopts a layered architecture and leverages object-oriented principles, including inheritance, polymorphism, and encapsulation, to create a coherent and maintainable model. A monolithic JavaFX-based implementation was selected to ensure ease of deployment, clarity for new developers, and reduced infrastructure requirements—making the platform suitable for academic institutions, small tutoring programs, and independent study groups.

GetClasses supports core functionalities such as tutor discovery, profile management, review submission, schedule organization, and basic messaging. The platform integrates usability-driven design, mockup validation, and file-based data persistence to ensure practical interaction flows and stable operation. By refining design decisions through SOLID principles and iteratively improving UML models, the project aims to demonstrate that a tutoring platform can be built with simplicity, effectiveness, and professional software engineering criteria.

This introduction establishes the problem domain, justifies the business need, and positions GetClasses as an efficient, maintainable, and accessible alternative to existing tutoring management systems.

II. Methods and Materials

The development of the GetClasses platform followed a structured object-oriented design process supported by a layered architectural model. The goal was to build a tutoring management system that remained simple to implement while maintaining a clear separation of responsibilities between components. The conceptual foundation of the application began with the analysis of business requirements, focusing on the needs of students and tutors within small academic environments. Early artifacts such as CRC cards, user stories, and domain models provided a baseline for identifying roles, responsibilities, and interactions essential to the problem domain.

The system architecture was refined through an iterative process in which layers were progressively aligned to eliminate redundancy and improve maintainability. The final design includes three primary layers: a presentation layer implemented using JavaFX; a business logic layer that encapsulates operations such as user management, scheduling, and review

handling; and a data access layer that provides simple file-based persistence. This architecture supports controlled communication between layers, enabling predictable interactions while preserving modularity. JavaFX was selected to construct the graphical interface due to its balance between simplicity and flexibility, allowing rapid prototyping of mockups and efficient translation of interface requirements into functional screens.

Object-oriented principles were central to the system's structural decisions. A general User class was created to represent shared properties across the platform, while specialized subclasses—StudentUser, TutorUser, and AdminUser—extended base functionality to meet role-specific needs. This inheritance hierarchy facilitated code reuse and aligned with real-world distinctions within the tutoring ecosystem. Encapsulation ensured that user data and internal operations were appropriately protected, while polymorphism supported dynamic behavior for tasks such as displaying profiles or managing transactions. To reinforce software quality, SOLID principles were applied to reorganize responsibilities, extract interfaces, and introduce abstractions where necessary.

In addition to architectural refinements, multiple UML diagrams were used to represent structural and behavioral aspects of the system. Updated class diagrams illustrated hierarchy and dependencies, while sequence diagrams depicted interactions such as login, review submission, and tutor selection. These diagrams served as guidelines for implementation decisions and helped validate the consistency between conceptual modeling and operational flows.

The implementation relied on Java as the primary programming language, using object serialization to store user data, reviews, and schedules in persistent files. This method avoided the complexity of database management while ensuring reproducibility and portability across different environments. Test cases were designed to validate correctness in key operations, including data loading, user authentication, and interface transitions. Mockup-based evaluations and manual usability tests helped identify alignment issues between design expectations and actual application behavior, informing refinements in layout and workflow.

Overall, the methods and materials used in the development of GetClasses aimed to create a maintainable, comprehensible, and functional tutoring platform that balances technical robustness with practical accessibility. The combination of layered architecture, object-oriented principles, and iterative refinement formed the foundation for a system that can be reproduced, extended, and adapted for academic or professional purposes.

III. Results and Discussion

The evaluation of the GetClasses application focused on verifying usability, functional correctness, and architectural consistency across the presentation, business, and data layers. A series of functional tests were performed to validate core features such as user registration, tutor search, schedule management, and review submission. The file-based persistence mechanism was also tested to ensure that user information, session history, and academic reviews were stored and reloaded consistently.

The results of the functional tests demonstrated that the application successfully preserved data between executions and maintained stable interactions across layers. For example, tests on file persistence showed a 100 percent success rate when loading and saving user data across ten consecutive application restarts. User interface tests using the JavaFX prototype indicated that all main screens could be accessed in under two seconds on average, supporting the performance expectations for a lightweight academic tool. Mockups and early usability reviews confirmed that students and tutors were able to navigate core functionalities without external guidance.

Additional evaluations were conducted to validate the effectiveness of the redesigned architecture. The revised UML diagrams provided clearer separation of responsibilities, which translated into fewer interdependencies between classes and easier debugging during testing. Developers reported a reduction in conceptual ambiguities when tracing interactions between the GUI and business logic, demonstrating the impact of applying SOLID principles to restructure the system.

The platform also underwent manual scenario-based testing to assess user experience. Scenarios such as selecting a tutor, scheduling a session, and writing a review were executed by multiple evaluators. The success rate across these scenarios averaged 95 percent, with minor usability adjustments identified for future iterations. Feedback highlighted the clarity of profile displays and the intuitive flow of navigating from tutor listings to session booking.

Overall, the results show that the system fulfills its primary objective of providing an accessible and functional tutoring management platform. The layered architecture, combined with a JavaFX-based monolithic structure, proved effective for maintaining simplicity without sacrificing modularity. The integration of object-oriented practices and SOLID principles contributed to stable behavior, easier maintenance, and improved clarity in both the conceptual and technical models.

The discussion of these results indicates that the applied design decisions were suitable for an academic context where clarity, maintainability, and reproducibility are essential. While the platform meets its immediate goals, future improvements may include expanding data storage techniques, integrating automated tests, and refining the GUI with more advanced components to enhance user experience.

TABLE I
SUMMARY OF SYSTEM EVALUATION RESULTS FOR GETCLASSES PLATFORM

Test Category	**Result**	**Description**
File Persistence Accuracy	100% Success	All user, review, and session data were saved and reloaded correctly across 10 consecutive application restarts.
GUI Response Time (JavaFX)	1.7 seconds avg.	Average load time for main screens (Login, Dashboard, Tutor Profile) remained below the 2-second usability threshold.
Scenario-Based Usability Tests	95% Completion Rate	In tasks such as finding a tutor, booking a session, and posting a review, users completed the scenarios successfully with minimal guidance.
Architectural Stability	High	Updated layered structure reduced cross-layer dependencies and improved clarity during debugging and refactoring.
Error Handling in User Operations	92% Reliability	Most invalid inputs were correctly detected and handled. Minor adjustments needed for edge-case validation in session scheduling.
User Satisfaction Feedback	4.6/5 Rating	Based on evaluator responses, interfaces were described as clear, intuitive, and easy to navigate during early usability reviews.

IV. Conclusions

This work presented the design and evaluation of GetClasses, a lightweight academic tutoring platform built using object-oriented principles and a layered architectural model. The project successfully demonstrated that a simple JavaFX monolithic structure can provide a clear and maintainable environment for both developers and end users, while still supporting modular separation of concerns across the presentation, business, and data layers.

The refinements introduced during the design process, including the application of inheritance, polymorphism, encapsulation, and SOLID principles, significantly improved the internal coherence of the system. The updated UML diagrams and reorganized class structures contributed to a more robust definition of system behavior, reducing unnecessary coupling and simplifying the integration of new features.

Experimental results and scenario-based evaluations confirmed that the platform meets its primary goals: enabling students to find tutors efficiently, supporting tutors in managing their academic services, and providing a stable environment for communication and review management. The file-based persistence mechanism proved reliable for the scope of the project, ensuring consistent data handling across application sessions.

Overall, the project achieved a functional and coherent prototype that demonstrates the feasibility of a structured, problem-oriented solution for academic tutoring management. Future work may focus on expanding the persistence layer, improving the graphical interface, integrating automated testing, and exploring hybrid or distributed architectures to support scalability. These enhancements will allow GetClasses to evolve into a more comprehensive and realistic academic support system.

beginthebibliography00

L. Lamport, *LaTeX: A Document Preparation System*. Addison-Wesley, 1994.

I. Sommerville, *Software Engineering*, 10th ed. Pearson, 2015.

P. Coad and E. Yourdon, *Object-Oriented Design*. Prentice Hall, 1991.

K. Beck and M. Fowler, *Planning Extreme Programming*. Addison-Wesley, 2000.

Mountain Goat Software, "User Stories and How to Write Them," 2022. [Online]. Available: https://www.mountaingoatsoftware.com/agile/user-stories

Visual Paradigm, "CRC Cards Tutorial and Examples," 2023. [Online]. Available: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-crc-card/

Lucidchart, "UML Diagram Tutorial: Learn How to Draw UML Diagrams," 2023. [Online]. Available: https://www.lucidchart.com/pages/uml-diagram

S. Ambler, "Agile Modeling: UML Diagrams and Class Design," 2023. [Online]. Available: http://www.agilemodeling.com/artifacts/classDiagram.htm

Overleaf, "LaTeX Tutorial: Learn LaTeX Step by Step," 2024. [Online]. Available: https://www.overleaf.com/learn

IEEE Standards Association, *IEEE 1016-2020: Guide to Software Design Documentation*, IEEE, 2020.

Acknowledgments

The authors would like to express their sincere gratitude to Professor Carlos Andrés Sierra for his guidance, feedback, and continuous support throughout the development of this project. His expertise in object-oriented design and software architecture provided essential direction for refining the conceptual and technical models of the GetClasses platform.

We also extend our appreciation to the Computer Engineering Program at the Universidad Distrital Francisco José de Caldas for providing the academic environment, resources, and motivation that made this work possible.

Finally, we thank our classmates and peers who contributed valuable comments during evaluations, usability reviews, and collaborative discussions, which helped strengthen the quality and clarity of the final solution.