Object-Oriented Programming
Semester 2025-III
GetClass - Final Delivery

Alejandro Escobar 20251020094
Jhon Gonzalez 20251020087
Sebastián Zambrano 20251020102

Computer Engineering Program
Universidad Distrital Francisco José de Caldas

1 Introduction — Business Model Focus

GetClasses is a digital marketplace that connects students seeking academic support with qualified tutors. The core business hypothesis is that a centralized platform offering verified profiles, scheduling tools, and trust mechanisms increases match success, platform revenue, and user retention compared to offline alternatives.

Domain Problem and Target Market Students face difficulty finding qualified, available, and affordable tutors. Tutors struggle to reach a wide audience and manage schedules efficiently. The target market includes high-school and university students seeking flexible academic support and freelance tutors aiming to professionalize their services.

Value Proposition and Business Drivers GetClasses provides:

- **Efficient Matching:** filters by subject, availability, rating, and cost.
- **Trust and Credibility:** verified profiles, ratings, and reviews to reduce risk.
- **Operational Simplicity:** tutors manage schedules, sessions, and communication easily.
- **Revenue Generation:** predictable income through booking fees or subscriptions.

Business Justification The platform reduces search friction, increases booking reliability, and enables a two-sided market, directly impacting key metrics: active tutors, conversion rate, and repeat bookings.

2 Technical Design — UML and SOLID

UML Class Diagram

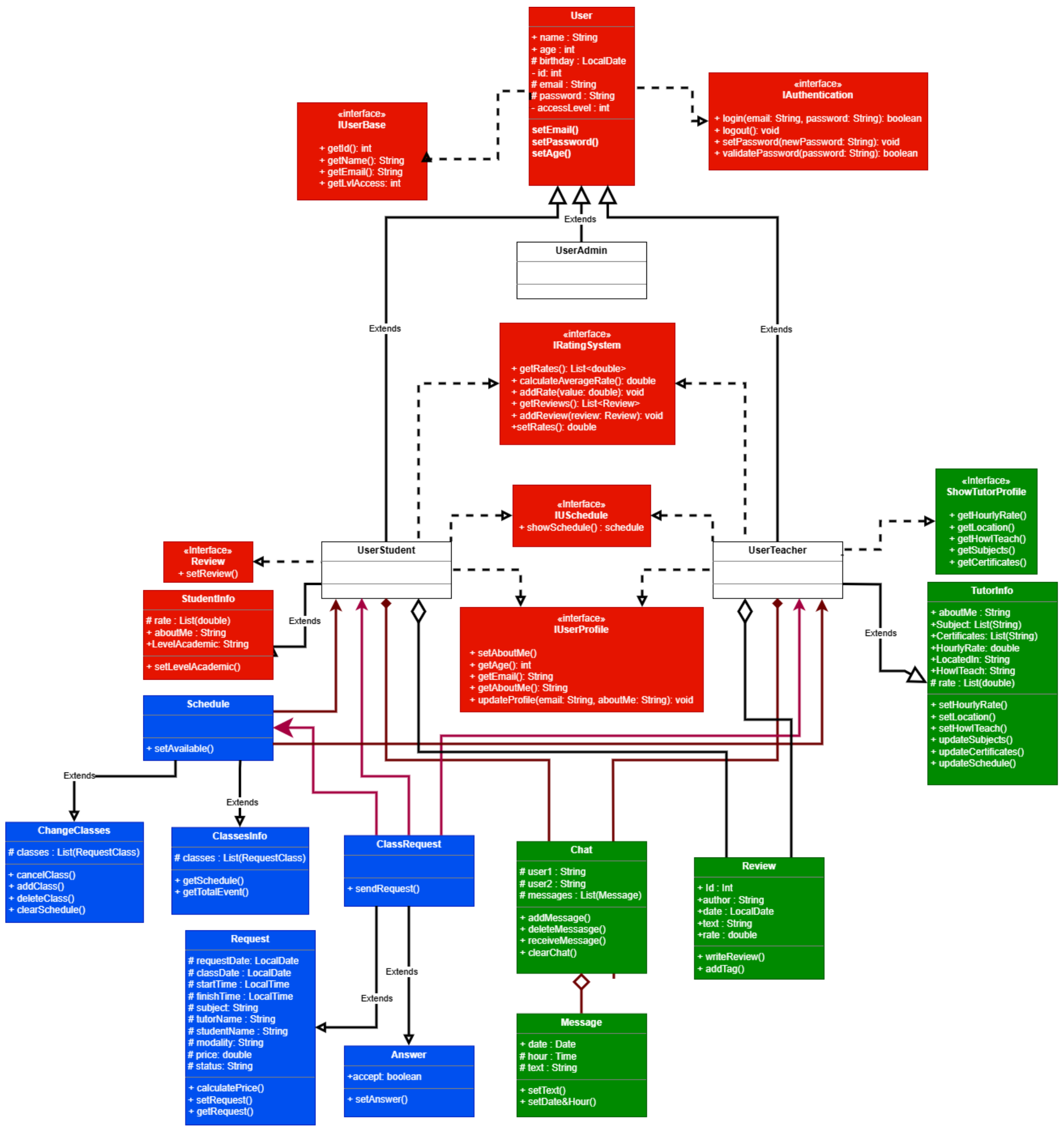
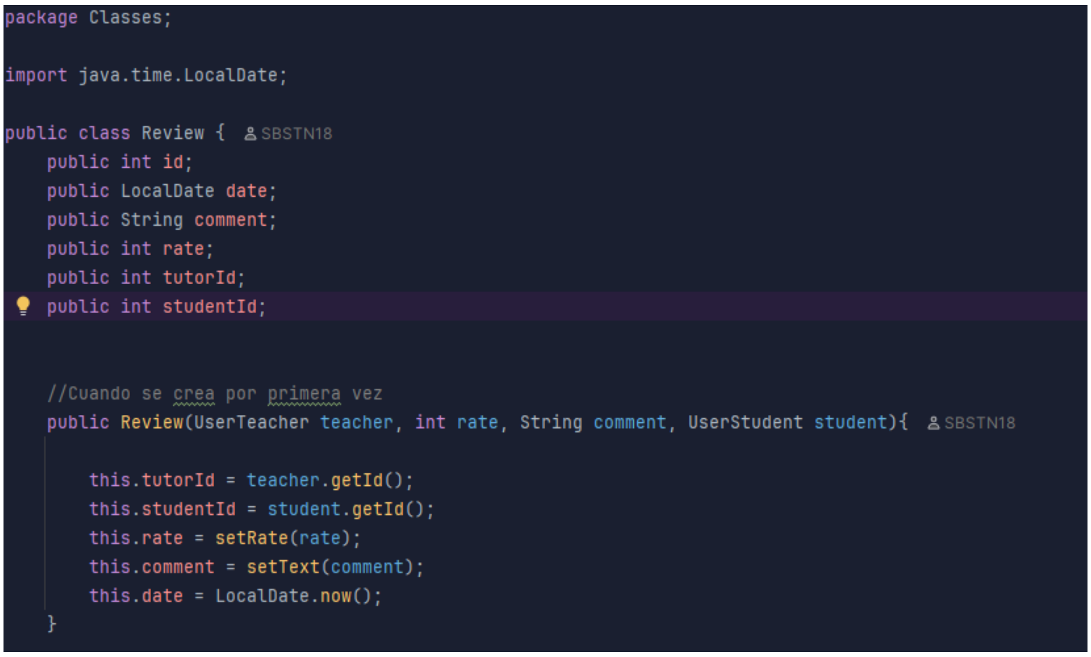


Figure 1: UML class diagram reflecting SOLID principles.

Key Modeling Decisions

- **User** — abstract base class with id, name, email, password; authentication behavior encapsulated.
- **TutorUser** — aggregates `Schedule`, `Reviews`, and `TutorInfo`.
- **StudentUser** and **AdminUser** — specialized subclasses.
- **ClassRequest** — composes `Request` and `Answer`, with `sendRequest()` as primary responsibility.
- **Chat** — associates two `User` instances and contains a list of `Message` objects.

SOLID Principles — Examples



```java
package Classes;

import java.time.LocalDate;

public class Review {  SBSTN18
    public int id;
    public LocalDate date;
    public String comment;
    public int rate;
    public int tutorId;
    public int studentId;


    //Cuando se crea por primera vez
    public Review(UserTeacher teacher, int rate, String comment, UserStudent student){  SBSTN18

        this.tutorId = teacher.getId();
        this.studentId = student.getId();
        this.rate = setRate(rate);
        this.comment = setText(comment);
        this.date = LocalDate.now();
    }
```

Figure 2: Code example: Class Review and his constructor.

Single Responsibility Principle

```java
// Aggregates reviews; only handles review logic
//Cuando se crea por primera vez
    public Review(UserTeacher teacher, int rate, String comment,
       UserStudent student){

        this.tutorId = teacher.getId();
        this.studentId = student.getId();
        this.rate = setRate(rate);
        this.comment = setText(comment);
        this.date = LocalDate.now();
    }

    //Cuando se carga de DB
    public Review(int id, int tutorId, int studentId, int score,
       String comment, LocalDate date){
        this.id = id;
        this.tutorId = tutorId;
        this.studentId = studentId;
        this.rate = score;
        this.comment = comment;
        this.date = date;
    }
```

Justification Interfaces enable loose coupling (DIP), while classes maintain single responsibility (SRP). Liskov Substitution is respected via abstract `User` and its subclasses. Open/Closed is applied to `ReviewsManager` — new behavior extends class without modifying existing code.

3 Boundaries and Interactions

Narrative

- UI (JavaFX) triggers events; controllers handle domain logic.
- Domain services operate on entities and call repositories.
- Persistence layer (JSON prototype / SQLite) handles data storage.
- External services (Payment, Email) invoked on domain events.

4 Requirements — Functional and Non-Functional

Functional Requirements

- Student registration, tutor registration, booking, messaging, review creation, schedule management.

Non-Functional Requirements — Quality Attributes

1. **Performance:** Sub-second response for search operations in local datasets to ensure smooth user experience.
2. **Scalability:** Layered architecture allows server-based deployment in the future.
3. **Maintainability:** Code is modular, unit-testable, and documented.
4. **Security:** Passwords are hashed (BCrypt); sensitive data is encrypted and validated.
5. **Reliability:** SQLite transactions ensure atomic writes; JSON fallback allows safe prototyping.
6. **Usability:** Standard UI patterns; main tasks achievable in 3 clicks.

5 User Stories — Given / When / Then

ID: 1	**Tutor Registration**
Priority:	High
Effort (hrs):	6
Description:	As a tutor, I want to register on the platform by entering my personal and professional information so that I can offer tutoring sessions.
Acceptance Criteria:	The tutor account is successfully created after entering valid information.

Table 1: User Story 1 – Tutor Registration

ID: 2	**Tutor Profile Picture**
Priority:	Medium
Effort (hrs):	3
Description:	As a tutor, I want to upload a profile picture so that students can identify me easily.
Acceptance Criteria:	The uploaded image appears correctly on the tutor's profile page.

Table 2: User Story 2 – Tutor Profile Picture

ID: 3	**Tutor Search**
Priority:	High
Effort (hrs):	8
Description:	As a student, I want to search for tutors by subject, rate, or language so that I can find the best match for my learning needs.
Acceptance Criteria:	Tutors matching the selected filters are displayed correctly in the search results.

Table 3: User Story 3 – Tutor Search

ID: 4	**Tutor Listing**
Priority:	High
Effort (hrs):	5
Description:	As a student, I want to view a list of available tutors so that I can compare their profiles and select one.
Acceptance Criteria:	The system displays a complete list of tutors with names, subjects, and ratings.

Table 4: User Story 4 – Tutor Listing

ID: 5	**Chat with Tutor**
Priority:	High
Effort (hrs):	6
Description:	As a student, I want to chat in real time with tutors so that I can clarify doubts before booking a session.
Acceptance Criteria:	The chat allows real-time message exchange between tutor and student.

Table 5: User Story 5 – Chat with Tutor

ID: 6	**Tutor Rates Student**
Priority:	Medium
Effort (hrs):	3
Description:	As a tutor, I want to rate students after a session to maintain platform quality and accountability.
Acceptance Criteria:	The tutor can submit a rating and review after completing a class.

Table 6: User Story 6 – Tutor Rates Student

ID: 7	**Student Rates Tutor**
Priority:	Medium
Effort (hrs):	3
Description:	As a student, I want to rate tutors after a session so that other users can make informed decisions.
Acceptance Criteria:	The student can rate the tutor and leave a comment after class.

Table 7: User Story 7 – Student Rates Tutor

ID: 8	**View Ratings**
Priority:	Medium
Effort (hrs):	3
Description:	As a user, I want to view tutor and student ratings so that I can evaluate trust and performance.
Acceptance Criteria:	Ratings and feedback are visible and linked to corresponding user profiles.

Table 8: User Story 8 – View Ratings

ID: 9	**Tutor Auto-Responder**
Priority:	Low
Effort (hrs):	2
Description:	As a tutor, I want to enable an auto-responder when I am unavailable so that students receive immediate feedback.
Acceptance Criteria:	The system automatically sends a pre-defined message when the tutor is offline.

Table 9: User Story 9 – Tutor Auto-Responder

ID: 10	**Contact Support/Admin**
Priority:	High
Effort (hrs):	4
Description:	As a user, I want to contact platform support or administrators to report issues or request assistance.
Acceptance Criteria:	Support requests are sent successfully and confirmation is received.

Table 10: User Story 10 – Contact Support/Admin

6 CRC Cards — Responsibilities (Detailed)

Class: ProfileUpdate	
Responsibility	**Collaborators**
Registers and updates the basic profile information for a user; validates input and triggers persistence mechanisms.	User

Table 11: ProfileUpdate CRC — Detailed responsibilities and collaborators

Class: User	
Responsibility	**Collaborators**
Represents general user attributes and behaviors; manages authentication, profile information, and basic interactions with system services.	ProfileUpdate, Teacher, Student

Table 12: User CRC — Detailed responsibilities and collaborators

Class: TutorInfo	
Responsibility	**Collaborators**
Manages tutor-specific information such as subjects, certifications, hourly rates, and experience; provides data for profile display and business logic.	Teacher

Table 13: TutorInfo CRC — Detailed responsibilities and collaborators

Class: StudentInfo	
Responsibility	**Collaborators**
Manages student-specific information such as enrollment, progress, and preferences; provides data for profile and interaction tracking.	Student

Table 14: StudentInfo CRC — Detailed responsibilities and collaborators

Class: Teacher	
Responsibility	**Collaborators**
Represents tutor user; inherits attributes and behaviors from User; manages personal schedule, class requests, reviews, and payment interactions.	User, TutorInfo, Reviews, ClassRequest, Schedul

Table 15: Teacher CRC — Detailed responsibilities and collaborators

Class: Student	
Responsibility	**Collaborators**
Represents student user; inherits attributes and behaviors from User; manages class bookings, schedule interaction, and reviews.	User, StudentInfo, Reviews, ClassRequest, Sched

Table 16: Student CRC — Detailed responsibilities and collaborators

Class: PaymentSystem	
Responsibility	**Collaborators**
Handles all payment transactions between students and tutors; validates payment methods, processes refunds, and maintains transaction records.	Teacher, Student

Table 17: PaymentSystem CRC — Detailed responsibilities and collaborators

Class: Reviews	
Responsibility	**Collaborators**
Manages creation, modification, and deletion of reviews; aggregates tutor ratings and feedback for system display.	Review, Student, Teacher

Table 18: Reviews CRC — Detailed responsibilities and collaborators

7 GUI — TutorCard and TutorListPanel

TutorCard

`TutorCard` is a JavaFX `BorderPane` that visually represents a tutor's profile. Responsibilities:

- Display name, age, "about me", subjects (buttons), cost per hour, rating, favorite status.
- Emit click events to controller via `TutorCardListener`.

Design justification

- Follows SRP: only rendering, no business logic.
- Listener interface ensures MVC separation; controller handles interactions.
- Modular design supports reuse in `TutorListPanel` and future extensions.

TutorListPanel

- Holds multiple `TutorCard` instances in a scrollable view.
- Responsible for rendering the tutor list and delegating click events.
- Maintains separation of concerns: UI container only, logic handled by controller.

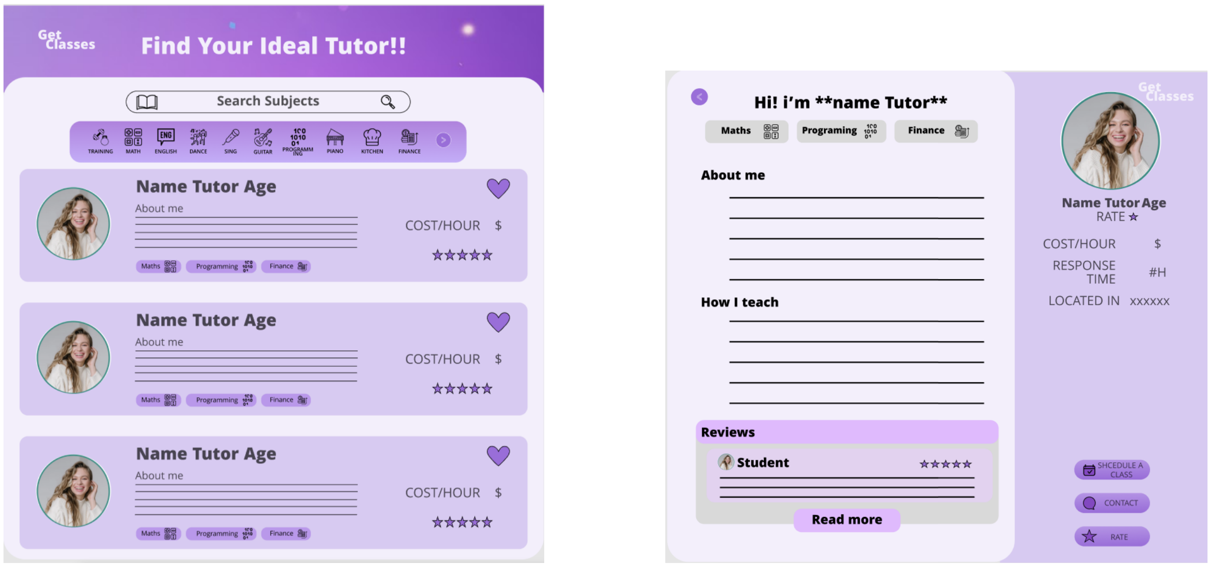


Figure 3: GUI mockups: Main (Left) y profile (right).

8 Presentation Layer — Architecture and Execution

Package Organization

- `com.getclasses.GUI.Controllers` — JavaFX controllers, FXML, views (TutorCard, TutorListPanel).
- `com.getclasses.Classes` — Entities and services (User, ReviewsManager, BookingService).
- `com.getclasses.Database` — Repositories (JSON/SQLite).
- `com.getclasses.GUI` — Repository interfaces, external service adapters.

Runtime Flow

1. Application initializes JavaFX and injects repositories into services.
2. Controllers call services; services manipulate entities and persist via repositories.
3. UI tasks that may block are executed in `Task/Service` threads.

9 Persistence — JSON and SQLite Integration

SQLite Integration

```java
public class ConnectionDB {

    private static final String URL = "jdbc:sqlite:src/main/
       resources/GetclassDB.db";
    public static Connection getConnection() {
        Connection conn = null;
        try {
            conn = DriverManager.getConnection(URL);
            System.out.println("Conexi n establecida con  xito "
               );
        } catch (SQLException e) {
            System.out.println("Error al conectar: " + e.
               getMessage());
        }
        return conn;
    }
}
```

Repositories abstract persistence, enabling UI and domain to remain unchanged when swapping storage.

10 Conclusions

- Architecture respects SOLID and MVC principles.
- TutorCard/TutorListPanel designed for reuse and clear separation of concerns.
- Persistence mechanisms (JSON/SQLite) fully integrated with domain and UI.
- Non-functional requirements are justified with technical reasoning.
- Document reflects business model focus and addresses professor's feedback.

11 References

- Overleaf. (2024). *LaTeX tutorial: Learn LaTeX step by step.* Retrieved from `https://www.overleaf.com/learn`
- Lamport, L. (1994). *LaTeX: A Document Preparation System.* Addison-Wesley.
- Lucidchart. (2023). *UML Diagram Tutorial.* Retrieved from `https://www.lucidchart.com/pages/uml-diagram`
- Ambler, S. W. (2023). *Agile Modeling: UML and Class Design.* Retrieved from `http://www.agilemodeling.com/artifacts/classDiagram.htm`
- Beck, K., & Fowler, M. (2000). *Planning Extreme Programming.* Addison-Wesley.
- Mountain Goat Software. (2022). *User Stories.* Retrieved from `https://www.mountaingoatsoftware.com/agile/user-stories`
- Coad, P., & Yourdon, E. (1991). *Object-Oriented Design.* Prentice Hall.
- Sommerville, I. (2015). *Software Engineering* (10th ed.). Pearson.
- Visual Paradigm. (2023). *CRC Cards Tutorial.* Retrieved from `https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-crc-card/`
- IEEE. (2023). *Guide to Software Design Documentation (IEEE 1016-2020).*