

University Course Registration System

Project Report

Submitted by:

Paishal Sharma

24BAS10052

Course:

Java Programming, VITyarthi

Date:

November 24, 2025

Contents

1	Introduction	2
2	Problem Statement	2
3	Functional Requirements	2
4	Non-functional Requirements	2
5	System Architecture	2
6	Design Diagrams	3
6.1	Use Case Diagram	3
6.2	Workflow Diagram	3
6.3	Class Diagram (Simplified)	3
7	Design Decisions & Rationale	4
8	Implementation Details	4
9	Results	4
10	Testing Approach	5
11	Challenges Faced	5
12	Learnings & Key Takeaways	5
13	Future Enhancements	5
14	References	5

1 Introduction

The University Course Registration System is a Java-based Command Line Interface (CLI) application designed to modernize and simplify the academic enrollment process. By leveraging Object-Oriented Programming (OOP) principles, the system allows students to view course catalogs, register for classes, and manage their schedules digitally. The application ensures data persistence using a file-based database approach, simulating a real-world registration environment.

2 Problem Statement

In many educational institutions, manual course registration is error-prone and inefficient. Students often lack real-time visibility into course capacity, leading to over-enrollment or scheduling conflicts. Administrators struggle to maintain accurate rosters without a centralized digital tool. This project addresses these issues by automating enrollment rules, enforcing capacity limits, and providing a persistent record of student registrations.

3 Functional Requirements

The system fulfills the following core functional requirements:

- **User Authentication:** Secure Login and Sign-Up functionality for students.
- **Course Catalog Viewing:** Display of available courses, credits, and remaining seats.
- **Registration:** Logic to enroll students in courses while checking for duplicates and capacity limits.
- **Schedule Management:** Ability for students to view their registered courses and withdraw if necessary.
- **Data Persistence:** Automatic saving and loading of student and course data to local text files.

4 Non-functional Requirements

- **Maintainability:** Data must survive application restarts (achieved via CSV file storage).
- **Usability:** Clear and intuitive CLI menus with helpful prompts.
- **Reliability:** The system handles invalid inputs gracefully without crashing.
- **Performance:** Instant feedback for user actions and efficient file parsing.

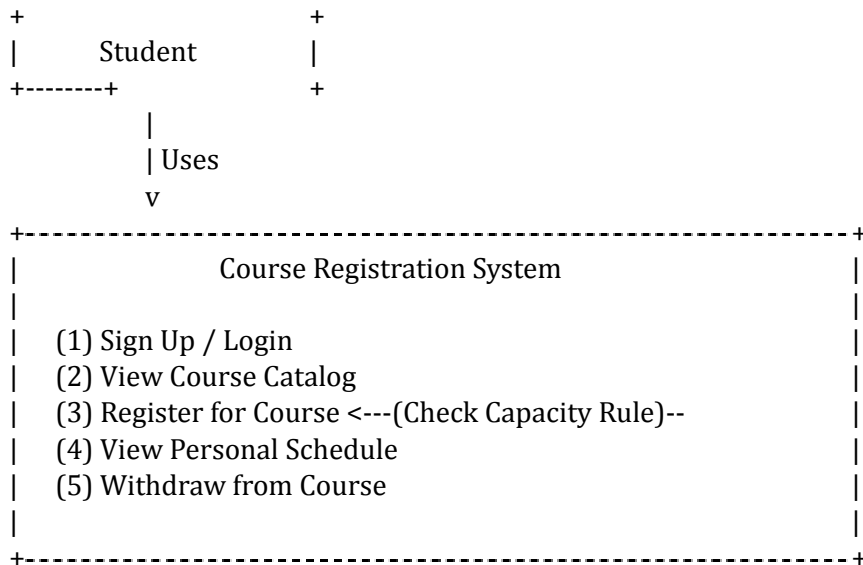
5 System Architecture

The project follows a Service-Oriented Architecture (SOA) / Model-View-Controller (MVC) pattern to ensure separation of concerns:

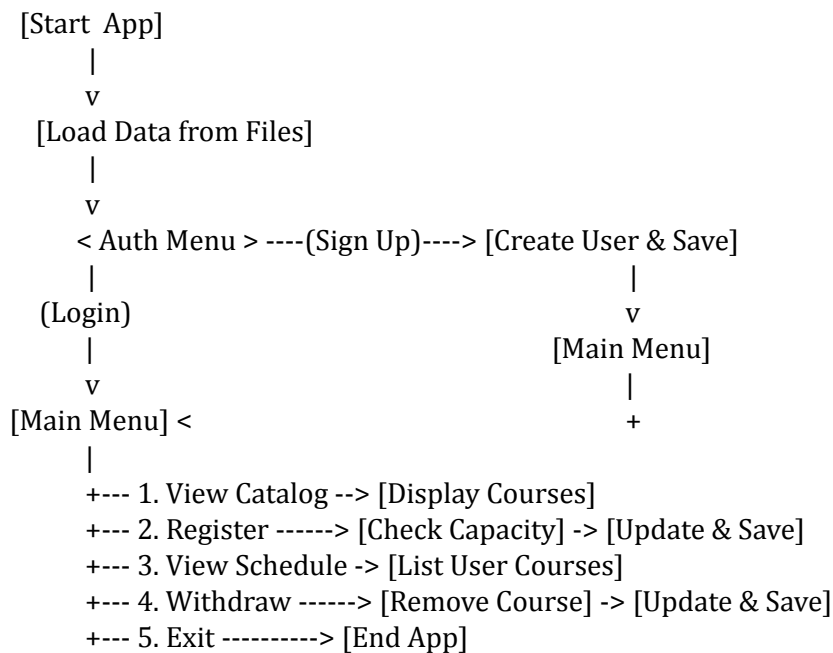
- **Models ('src.models'):** POJOs (Plain Old Java Objects) representing 'Student' and 'Course'.
- **Service ('src.services'):** Contains business logic and file I/O operations. It acts as the bridge between the data and the user interface.
- **View/Controller ('src.Main'):** Handles user input, displays menus, and invokes service methods.

6 Design Diagrams

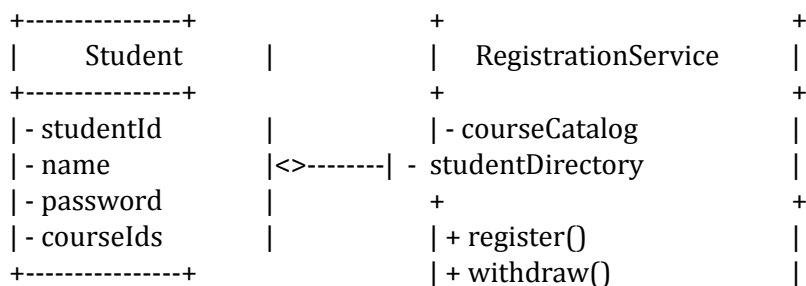
6.1 Use Case Diagram

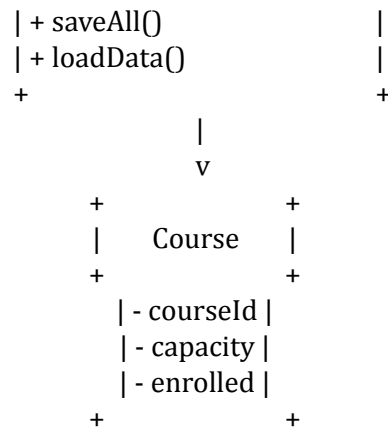


6.2 Workflow Diagram



6.3 Class Diagram (Simplified)





7 Design Decisions & Rationale

- **File-Based Storage (CSV):** Chosen over a complex SQL database to keep the project lightweight and portable. This allows the application to run on any machine with Java installed without re- quiring database server setup.
- **CLI Interface:** A command-line interface was selected to focus purely on business logic and backend implementation logic rather than spending time on UI/CSS libraries.
- **Service Layer Pattern:** Business logic was isolated in 'RegistrationService' to make the code testable and modular.

8 Implementation Details

The project is implemented in Java (JDK 17). Key implementation details include:

- **Collections Framework:** Used 'ArrayList' to manage runtime lists of students and courses.
- **Java Streams:** Used for filtering lists (e.g., finding a student by ID).
- **File I/O:** 'BufferedReader' and 'BufferedWriter' are used for reading and writing '.txt' files in CSV format.
- **Exception Handling:** Custom exceptions are thrown for business rule violations (e.g., "Course is full").

9 Results

The application successfully simulates the registration flow.

- **Successful Registration:** Students can enroll in courses, and the "Seats Available" count decreases immediately.
- **Persistence:** Upon restarting the application, previous enrollments are correctly loaded.
- **Validation:** The system correctly rejects attempts to register for full courses or duplicate enroll- ments.

10 Testing Approach

A custom unit testing suite was developed (dependency-free) to verify core logic:

- **Unit Tests:** Located in 'src.test.RegistrationServiceTest'.
- **Test Cases Covered:**
 1. Successful registration increments student course count.
 2. Duplicate registration throws 'IllegalStateException'.
 3. Registration for a full course throws 'IllegalStateException'.

11 Challenges Faced

- **Data Persistence:** syncing the in-memory 'List' state with the text files on the disk required careful management of 'load()' and 'save()' method calls to ensure no data was lost on exit.
- **Object Relationships:** Handling relationships between 'Student' and 'Course' when loading from CSV required storing IDs rather than full objects to avoid circular dependencies during deserialization.

12 Learnings & Key Takeaways

- Gained practical experience with Java File I/O and data serialization.
- Understood the importance of separating the View (Main) from the Logic (Service).
- Learned how to write unit tests to verify business rules automatically.

13 Future Enhancements

- **Database Integration:** Migrate from text files to a MySQL or SQLite database for better scalability.
- **GUI:** Implement a graphical user interface using JavaFX or Swing.
- **Admin Features:** Add menu options for Admins to create new courses dynamically at runtime.

14 References

- Official Java Documentation (Oracle)
- Course Syllabus and Lecture Notes