University Course Registration System

Arnav Adarsh

September 19, 2024

Contents

1 Introduction

This project is a terminal-based university course registration system developed using **Object-Oriented Programming (OOP)** principles. The application supports three types of users:

- **Students**: Register for courses, view schedules, track academic progress.
- **Professors**: Manage courses, view enrolled students.
- **Administrators**: Manage the course catalog, student records, assign professors, and handle complaints.

The aim of this system is to provide a streamlined solution for managing enrollments, schedules, and academic records while adhering to key OOP concepts.

2 OOP Principles in the Project

The project adheres to the four fundamental principles of OOP: **Encapsulation, Inheritance, Polymorphism, and Abstraction**. Each principle is applied effectively within the code, ensuring maintainability, scalability, and reusability.

2.1 Interface

Explanation: Interfaces are used to define a set of functionalities that must be implemented by any class that uses the interface. In this project, the interface `UserInterface` ensures that all user types (Student, Professor, Administrator) implement the necessary methods for login and viewing their respective options.

Implementation: The interface `UserInterface` defines abstract methods like `login()` and `viewOptions()`, which are then implemented differently for `Student`, `Professor`, and `Administrator` classes.

2.2 Classes and Objects

Explanation: Classes are the blueprint for creating objects, which are instances of the class. This project includes classes for each user type (Student, Professor, Administrator) as well as courses, complaints, and academic records.

Implementation:

- **Student Class**: Handles student-specific actions like viewing available courses, registering, dropping courses, and tracking academic progress.
- **Professor Class**: Allows professors to manage courses and view enrolled students.
- **Administrator Class**: Handles the overall management of courses, student records, and complaints.

2.3 Encapsulation

Explanation: Encapsulation involves bundling the data (variables) and methods (functions) that manipulate the data within one unit (class) and restricting access to some components to ensure control and data security.

Implementation: Private variables are used within classes, such as `password` and `studentGrades`. Getter and setter methods are used to control access to these private variables, ensuring data security.

2.4 Inheritance

Explanation: Inheritance allows a class to inherit methods and attributes from another class. In this project, common functionalities are abstracted into a base class.

Implementation: A base class `User` is created, and classes like `Student`, `Professor`, and `Administrator` inherit common properties such as `email`, `name`, and the `login()` method.

2.5 Polymorphism

Explanation: Polymorphism allows one method to behave differently based on the object that is calling it. The same method can perform different functions in different scenarios.

Implementation: The `viewOptions()` method is overridden in each class (`Student`, `Professor`, and `Administrator`) to display user-specific options, showcasing polymorphism.

2.6 Abstraction

Explanation: Abstraction involves hiding the complex implementation details and showing only the necessary functionalities to the user.

Implementation: Only the high-level functionalities like "register course" or "manage courses" are exposed to the users, while complex operations like data validation and storage are hidden in the backend.

3 System Roles and Functionalities

3.1 Student Functionalities

- **View Available Courses**: Students can see all available courses for the current semester, including course details.
- **Register for Courses**: Students can register for courses, ensuring they meet prerequisites and do not exceed the credit limit.
- **View Schedule**: Students can see their weekly course schedule, including timings and professor details.

- **Track Academic Progress**: Students can view grades and calculate SGPA and CGPA based on completed courses.
- **Drop Courses**: Students can drop a course anytime during the semester.
- **Submit Complaints**: Students can submit complaints related to schedule clashes or other issues.

3.2 Professor Functionalities

- **Manage Courses**: Professors can update course details such as timings, prerequisites, and enrollment limits.
- **View Enrolled Students**: Professors can see a list of students enrolled in their courses and access their academic standing and contact information.

3.3 Administrator Functionalities

- **Manage Course Catalog**: Administrators can view, add, and delete courses.
- **Manage Student Records**: Administrators can view and update student records and grades.
- **Assign Professors to Courses**: Administrators assign professors to courses.
- **Handle Complaints**: Administrators manage student complaints and update their status.

4 How to Run the Code

4.1 Setup

- Ensure you have Java installed on your system.
- Clone the repository or download the ZIP file.
- Navigate to the project directory.

4.2 Compile the Code

Use the following command to compile all Java files:

```
javac *.java
```

4.3 Run the Application

Use the following command to run the application:

```
java Main
```

Follow the menu-driven interface to log in as a student, professor, or administrator.

5 Explanation for Beginners

For beginners, understanding the code can be simplified by breaking it into smaller parts:

- **Classes and Objects**: Think of classes as blueprints and objects as real-world entities. For example, a `Student` class is the blueprint, and an individual student like "John Doe" is an object.
- **Methods**: Methods are actions that an object can perform. For example, `registerCourse()` allows a student object to register for a course.
- **Encapsulation**: We use encapsulation to protect sensitive data. For instance, a student's grades are private and can only be accessed or modified through specific methods.

6 Challenges and Assumptions

- **Credit Limit**: The system enforces a maximum credit limit of 20 credits per semester. Courses can be either 2 or 4 credits.
- **Simplified Prerequisite System**: Students can only register for courses in their current semester, provided they have completed the necessary prerequisites.
- **Complaints Handling**: Complaints are managed by administrators, with statuses being updated manually.

This project demonstrates key OOP principles while providing a functional solution to course registration in a university setting.