Overview

This is a **University Course Registration System** designed using Object-Oriented Programming concepts. It simulates functionalities where users (students, professors, and administrators) can interact with a course catalog, manage registrations, grades, and academic records, as well as file complaints.

Key Functionalities:

- **Student Enrollment:** Students can view available courses, register, drop courses, view schedules, and file complaints.
- **Course Management:** Professors and administrators can manage course details, including assigning professors, setting grades, and enrolling students.
- **Complaint System:** Students can file complaints, and administrators can manage and track their status.

Project Structure

The system is organized into multiple packages and classes for effective use of OOP principles.

1. users package

- **User (Abstract Class):**
  - This is the base class for all user types (`Student, Professor, Admin`).
  - It contains common attributes such as email and password.
  - Methods for login, signup, and logout are defined, but each user type customizes the logic (This is due to admin having fixed login credentials as stated in assignment pdf) .
  - It demonstrates **encapsulation** by securing the user email and password and using getter and setter methods.
- **UserActions (Interface):**
  - Defines core methods that any user must implement: `login`, `signup`, and `logout`.

    - Demonstrates **abstraction** by hiding specific user logic but enforcing implementation.

2. student package

- **Student (Class):**
    - It extends the `User` class, representing the student.
    - This manages registered courses, completed courses, grades, and complaints.
    - It contains functionalities for calculating SGPA and CGPA, managing course enrollments, and viewing complaint statuses.
    - It demonstrates **Inheritance** by inheriting login and signup mechanisms from `User`, while customizing student-specific behavior.
    - Uses **polymorphism** when overriding methods such as `login` and `signup` to fit the student context.

3. courses package

- **Course (Class):**
    - This represents a university course with attributes like course code, title, credits, schedule, professor, and enrolled students.
    - manages prerequisites and enrollment logic.
    - **Encapsulates** course data and provides methods to manage course attributes.
- **CourseBook (Class):**
    - it acts as a catalogue for all courses.
    - Students and admin can use it to see available courses for a specific semester.
    - **Encapsulates** the collection of courses and exposes them through controlled access.

4. administrator package

- **Admin (Class):**
    - It extends the `User` class and represents an admin who manages course and student records.

    - Manages adding courses, deleting courses, and assigning professors to courses.
    - It uses **Inheritance** to inherit methods from `User` while implementing admin-specific methods.
    - Demonstrates **polymorphism** when overriding inherited methods for admin roles.

5. complaints package

- **Complaint (Class):**
    - It represents a complaint filed by a student.
    - **Encapsulates** the complaint details such as ID, status, description, and student information.
    - Uses getter to control how complaint data is accessed.
- **ComplaintBook (Class):**
    - Maintains a collection of complaints.
    - Allows admin to view all complaints and students to track their complaint status.
    - Demonstrates **encapsulation** by controlling access to complaint records.

6. helper package

- **InputValidator (Class):**
    - It is responsible for validating user inputs, such as email and password formats.
    - Ensures **abstraction** by hiding complex validation logic and providing simple methods for input validation.

OOP Concepts in Action:

1. Classes & Objects:

- The system is composed of several classes:
- `User`: An abstract base class for all user types
- `Student`: Represents a student in the university
- `Professor`: Represents a professor in the university

- `Admin`: Represents an administrative user
- `Course`: Represents a course offered by the university
- `Complaint`: Represents a complaint filed by a student
- `CourseCatalogue`: Manages the list of all courses
- `ComplaintManager`: Manages all complaints in the system
- `UniversitySystem`: The main class that orchestrates the entire system

Each of these classes **encapsulates** related data and objects of these classes interact to form the complete system.

2. Interfaces

The system uses interfaces to define contracts for classes:

- `UserActions`: Defines the common actions that all users can perform (login, logout, signup)

3. Inheritance:

- Inheritance is used to create a hierarchy of user types:

```
User (abstract class)
├── Student
├── Professor
└── Admin
```

All user types inherit common attributes and methods from the `User` class, allowing for code reuse and establishing a clear relationship between different user types.

4. Polymorphism:

- The `UserActions` interface defines methods that are implemented differently across the user types (`Student`, `Admin`).
- Polymorphism is also used in methods like `login` and `signup`, where each subclass overrides the inherited methods to provide specialized behavior.

5. Encapsulation:

- Sensitive data, such as user credentials and complaint details, are hidden from direct access through private fields.
- Public getter and setter methods provide controlled access to these fields, enforcing **data hiding**.
- Classes like `InputValidator` demonstrate encapsulation by hiding the validation logic behind simple method calls.

6. Abstraction:

- The system provides an abstraction of real-world entities such as users, courses, and complaints.
- By utilizing abstract classes (`User`) and interfaces (`UserActions`), the project hides complex internal details and exposes only necessary operations.

Conclusion

In summary, this university course registration system uses OOP concepts effectively to build the application. The use of inheritance, polymorphism, encapsulation, and abstraction ensures that each entity in the system behaves independently still interacts easily with other entities which enables efficient management of university operations.

(I have created an hardcoded ExampleUsage.java file to test the system quickly. It consists of 9 professors, 4 students and 10 courses.)