University Course Registration System

Overview

This project implements a **University Course Registration System** in Java, following Object-Oriented Programming (OOP) principles. It allows students, professors, and administrators to manage their respective tasks efficiently. The system incorporates **generic programming**, **object classes**, and **exception handling** to create a flexible and error-resilient platform.

Features

Student Features

- **View Available Courses**: Displays a list of courses that students can enroll in.
- **Register for Courses**: Allows students to register for courses, checking for prerequisites and conflicts.
- **View Schedule**: Students can view their current course schedule.
- **Drop Courses**: Provides students with an option to drop courses before specific deadlines.
- **Submit Complaints**: Students can file complaints or raise issues about the system or their courses.

Professor Features

- **View Assigned Courses**: Professors can see the courses assigned to them.
- **Manage Course Materials**: Allows professors to upload and manage course content such as assignments or study materials.
- **Assign Teaching Assistants (TAs)**: Professors can assign TAs to help manage their courses.

Administrator Features

- **Manage Professors**: Administrators can assign or remove professors from courses.
- **Manage Course Catalog**: Allows administrators to add or remove courses from the system.

Key Concepts

1. Generic Programming

The system uses **generic programming** to manage collections of data, providing flexibility in how objects are handled and stored. For instance, the use of generic collections like `ArrayList<T>` ensures that a single class can manage different types of data (students, courses, professors) efficiently. Here's an example of how generic programming is used:

java
Copy code

```
public class CourseManager<T> {
    private List<T> courses;

    public CourseManager() {
        this.courses = new ArrayList<>();
    }

    public void addCourse(T course) {
        courses.add(course);
    }

    public T getCourse(int index) {
        return courses.get(index);
    }
}
```

The `CourseManager` class is generic, making it versatile for different types of objects such as courses, students, or professors.

2. Object Classes

This system models real-world entities using **object classes**, ensuring that each entity (e.g., `Student`, `Professor`, `Course`, `Administrator`) has its own properties and methods. The use of OOP principles such as inheritance and polymorphism helps streamline interactions between different entities.

Student Class

The `Student` class represents the student entity, encapsulating data and operations like registering for courses and viewing schedules. Here is a simplified version of the `Student` class:

java
Copy code
```
public class Student {
    private String studentId;
    private List<Course> enrolledCourses;

    public Student(String studentId) {
        this.studentId = studentId;
        this.enrolledCourses = new ArrayList<>();
    }

    public void registerForCourse(Course course) {
        // Logic to register a student for a course
    }

    public List<Course> viewSchedule() {
        return enrolledCourses;
    }
}
```

3. Exception Handling

To make the system more robust and handle unexpected situations, **exception handling** has been integrated. Custom exceptions are used for specific error conditions, such as invalid logins or issues with course registration.

InvalidLoginException

This custom exception handles incorrect login attempts by professors:

java
Copy code
```
public class InvalidLoginException extends Exception {
    public InvalidLoginException(String message) {
        super(message);
    }
```

```
}
```

When a professor tries to log in with incorrect credentials, the system throws an `InvalidLoginException`, ensuring a clear and descriptive error message is provided.

How to Use

1. **Setup**: Clone or download the project from the repository.
2. **Compile and Run**: Use a Java IDE (such as IntelliJ or Eclipse) to compile and run the project.
3. **Login**: You can log in as a student, professor, or administrator using provided credentials.
4. **Manage Courses**: Depending on your role, the system allows you to register for courses (student), manage courses (professor), or assign courses (administrator).
5. **Error Handling**: In cases where you encounter errors (e.g., invalid login or registration issues), the system will display an appropriate message.

Future Enhancements

- **SGPA Calculation**: Integrate a feature that calculates the student's Semester Grade Point Average (SGPA) based on completed courses.
- **Advanced Search**: Enable advanced filtering of available courses based on prerequisites, timing, and professor.
- **Enhanced Security**: Add encryption for sensitive information like login credentials.

Conclusion

This project demonstrates the use of OOP principles, generic programming, and exception handling in building a university course registration system. It ensures a flexible and scalable design, allowing for future expansion while maintaining strong error-handling mechanisms.