Overview of the System

The Course Scheduling System simplifies the enrollment process for students by enabling them to draft, validate, and finalize course schedules efficiently. Designed with flexibility, scalability, and maintainability in mind, the system utilizes key object-oriented principles and design patterns to address common challenges.

Design Patterns

Singleton Pattern

- **Usage**: Implemented in the Authentication class to ensure a single instance manages user credentials.
- **Reason**: Centralized authentication avoids redundancy and ensures consistent access to credentials across the system. The Singleton pattern guarantees controlled, thread-safe access to authentication logic.

Factory Pattern

- **Usage**: Utilized in the CourseManager class to dynamically create various course types (e.g., online courses, lab-based courses).
- **Reason**: Abstracting the course creation process ensures the system can accommodate future expansion, such as adding new course types, without modifying core logic.

Strategy Pattern

- **Usage**: Incorporated in the EnrollmentStrategies module to allow the system to adopt different enrollment rules (e.g., first-come-first-served, merit-based enrollment).

- **Reason**: By separating enrollment strategies, the system adapts seamlessly to changing policies without major code changes.

Observer Pattern

- **Usage**: Designed to notify students about updates, such as when waitlisted courses become available.
- **Reason**: Decoupling the notification mechanism makes the system scalable and easier to maintain when adding new notification channels (e.g., email, SMS).

Dependency Injection

- **Usage**: Applied in components such as PrerequisiteChecker when integrating with Scheduler and CourseManager.
- **Reason**: Loose coupling between dependencies improves flexibility and enhances the system's testability.

Object-Oriented Principles

Encapsulation

- Classes like Student, Course, and Scheduler encapsulate their respective data and expose only necessary methods.
- **Example**: The Student::addCourse method ensures controlled addition of courses to a student's schedule.

Inheritance

- **Usage**: The User class serves as a base class for Student and Instructor, sharing common attributes like username and email.
- **Reason**: This approach promotes code reuse and simplifies class hierarchies.

Polymorphism

- **Usage**: Applied in EnrollmentStrategies to enable different enrollment logic strategies.
- **Reason**: This makes the system extensible for diverse enrollment requirements.

Challenges Addressed

Prerequisite Validation

- **Solution**: The PrerequisiteChecker class ensures students meet course requirements before enrollment.
- **Design Choice**: Decoupling validation logic promotes single responsibility and reusability.

Efficient Draft Schedule Management

- **Solution**: The Scheduler class incorporates methods like addToDraftSchedule and validateDraftSchedule for efficient schedule management.
- **Reason**: Ensures students can finalize or modify their draft schedules effectively before enrollment.

Centralized Course Management

- **Solution**: The CourseManager class provides unified methods for adding, removing, and searching courses.
- **Reason**: Simplifies course management and reduces redundancy.

Trade-offs and Future Enhancements

Trade-offs

1. **Increased Complexity**: Incorporating multiple design patterns added complexity to the implementation but enhanced maintainability and scalability.
2. **Time Constraints**: Due to project deadlines, some features, such as a real-time notification system and advanced analytics, were deferred.

Future Enhancements

1. Implementing the **Observer Pattern** for real-time notifications to students.
2. Expanding the **Factory Pattern** to accommodate diverse course types.
3. Integrating a robust database system for persistent data storage.
4. Refining user interfaces to enhance usability.

Conclusion

The application of design patterns and object-oriented principles ensures the system is flexible, maintainable, and scalable. These design decisions address the immediate requirements while allowing room for future enhancements and expansions.