CSE Fest Management System

Strtoryline :
The CSE Fest Management System is designed to help manage a university's annual Computer Science and Engineering festival, which consists of multiple segments or events such as symposium talks, datathons, innovative showcasing, and competitive programming challenges. The system allows participants to register for these segments, receive notifications about event updates, and manage event schedules and results. It is structured using the SOLID principles to ensure scalability, maintainability, and ease of extension for future enhancements.

Storyline Breakdown :

1. Main Event Management (main.py):

   - The entry point of the system where the various event segments are initialized, participants are registered, and the event flow is managed.

   - The program allows the user to interact with the system, view the list of available segments, register for them, and get updates on event schedules and results.

2. Event Segments (segment/):

   - The system handles several segments like Symposium, Datathon, Innovation Showcasing, and Competitive Programming.

   - Each segment has its own script (e.g., symposium.py, datathon.py) that defines how participants interact with it, from registration to event progression.

   - The segments inherit from the SegmentInterface class defined in the interfaces/segment_interface.py file, ensuring consistent structure across all event types.

3. Models (models/):

   - Participant: This model represents a participant's data such as name, contact information, and the segment they are participating in.

   - Event: A general model for handling the details of any event, including time, venue, and associated participants.

   - Registration: This model manages the registration process, ensuring participants are successfully signed up for the correct segment, and handles any associated data.

4. Services (services/):

   - Scheduler: Manages the scheduling of all events, ensuring there are no conflicts between different segment times.

- Notifier: Sends notifications to participants about event updates such as schedule changes or results. It uses the NotificationInterface for flexibility in how notifications are delivered (email, SMS, etc.).
- Results: Handles storing and displaying results for the competitive events, such as the datathon and competitive programming challenges.

5. Interfaces (interfaces/):
    - SegmentInterface: Defines the required methods for any event segment (such as registering a participant and starting the event). This interface ensures that every segment has a consistent structure and behavior.
    - NotificationInterface: Ensures that any notification service used in the system implements the required methods for notifying participants, maintaining flexibility and separation of concerns.

<u>How SOLID Principles Are Followed:</u>

1. Single Responsibility Principle (SRP):
    - Each class and module has one reason to change:
        - The segment modules like symposium.py, datathon.py, etc., handle the specific logic related to that segment only.
        - The models (e.g., participant.py, event.py) focus on the data-related operations and represent entities in the system.
        - The services (e.g., scheduler.py, notifier.py) are focused on auxiliary tasks such as scheduling and sending notifications.
    - This modular approach ensures that each class/module has a single responsibility, making the code easier to maintain and extend.
2. Open/Closed Principle (OCP):
    - The system is open for extension but closed for modification:
        - New event segments can be added (e.g., a new segment like workshops) without modifying existing code, just by implementing new classes that extend SegmentInterface.
        - Similarly, new notification methods (e.g., push notifications) can be added by creating new classes that implement NotificationInterface without changing the core logic of the notifier.
3. Liskov Substitution Principle (LSP):

- Objects of a superclass (in this case, SegmentInterface or NotificationInterface) should be replaceable with objects of a subclass (like Symposium, Datathon, or NotifierEmail) without affecting the program's behavior.
- For example, a NotifierEmail class can be substituted for NotifierSMS in the Notifier service without changing the event registration flow, ensuring the system remains flexible and predictable.

4. Interface Segregation Principle (ISP):
   - The system uses small, specific interfaces rather than large, general ones:
     - SegmentInterface focuses solely on the methods required for segment management (such as register_participant and start_event), and NotificationInterface focuses on notification handling.
   - This ensures that classes only need to implement the methods they actually use, avoiding unnecessary dependencies and keeping the system more modular.
5. Dependency Inversion Principle (DIP):
   - High-level modules (services, segment, etc.) do not depend on low-level modules (like specific notification methods or segment types); instead, both depend on abstractions (interfaces).
   - For example, the Notifier service interacts with the NotificationInterface, not a concrete notification class, which allows the system to be easily adapted to different notification mechanisms (email, SMS, etc.) without changing the core event logic.
   - This abstraction of dependencies ensures that components remain loosely coupled and easier to test and modify.

Conclusion:

The CSE Fest Management System is structured using solid object-oriented principles, making the codebase highly modular, flexible, and extensible. Each component adheres to the SOLID principles, ensuring that the system can grow and change without becoming brittle or difficult to maintain. The use of interfaces (like SegmentInterface and NotificationInterface) guarantees that new segments and notification methods can be added in a predictable manner, while the separation of concerns allows for independent development of features like event scheduling, participant management, and result notification.