1. Behavioral Design Patterns

The focus of behavioral patterns is on the interactions and communication between objects. These patterns help to efficiently handle tasks and notifications in the Task Scheduler system.

Use Case 1: Observer Pattern (Task Conflict Notifications)

When a new task is added to the Task Scheduler, it may be necessary to update many components. For example, notifications (such as Conflict Notifiers and Email Notifiers) should be issued to several observers in the event of a scheduling conflict. The Observer Pattern can be integrated to alert various components of task conflicts.

Implemntation of the task sheduler:

- Upon adding a new task, the system looks for conflicts.
- When a conflict occurs, observers (such Conflict Notifier and Email Notifier) are alerted.

Use Case 2: Strategy Pattern (Task Sorting Strategies)

The Task Scheduler supports many sorting methods (e.g., by priority level, by start time). You can switch between different sorting mechanisms according to user desire by using the Strategy Pattern.

Implementation in Task Scheduler:

- Permit sorting based on priority or start time.

2. Creational Design Patterns:

Creational patterns deal with object creation mechanisms. Here, the Singleton and Factory patterns can be used in your Task Scheduler.

Use Case 3: Factory Pattern (Task Creation Factory)

To construct tasks with varying priorities or kinds (such as HighPriorityTask and LowPriorityTask), you need a flexible method. The logic for creating tasks can be assigned to a factory by using the Factory Pattern. Implementation in your Task Scheduler: To instantiate distinct job types, use a factory.

Implementation in Task Scheduler:

- Use a factory to instantiate different task types.

Use Case 4: Singleton Pattern (Schedule Manager)

To maintain consistency in task management, you should only have one instance of the ScheduleManager in your Task Scheduler. There will only be one instance of the schedule manager created if the Singleton Pattern is followed.

Implementation in Task Scheduler:

- Make sure that the ScheduleManager is the only one.

3. **Structural Design Patterns**:

Structural patterns focus on how objects are composed. In your Task Scheduler, the **Adapter** and Decorator patterns can be useful.

Use Case 5: Adapter Pattern (External Task Source Integration)

Suppose you need to import tasks into your Task Scheduler from an external API that offers tasks in a different format. To convert the external task data format to your internal task format, utilize the Adapter Pattern.

Implementation in Task Scheduler:

- An adapter for converting task data from outside sources to internal ones.

Use Case 6: Decorator Pattern (Task Completion Status)

You may want to add more functionality to your tasks in your task scheduler, such the ability to mark them as "completed" without changing the original Task class. You can dynamically add this functionality using the Decorator Pattern.

Implementation in Task Scheduler:

- A decorator can be used to give a task the status "completed".