**Mediator Design Pattern:** A mediator design pattern is one of the important and widely used behavioral design patterns. Mediator enables the decoupling of objects by introducing a layer in between so that the interaction between objects happens via the layer. Real Life

* If the objects interact with each other directly, the system components are tightly coupled with each other making higher maintainability cost and not hard to extend.
* The mediator pattern focuses on providing a mediator between objects for communication and helps in implementing loose coupling between objects.

Example: In an airport, there are multiple airplanes that need to communicate and coordinate their movements to avoid collisions and ensure safe takeoffs and landings. Without a centralized system, direct communication between airplanes could lead to chaos and increased risk*.*

Online Auction system also an another example for Mediator design pattern.

**When to use the Mediator Design Pattern**

* **Complex Communication:** Your system involves a set of objects that need to communicate with each other in a complex manner, and you want to avoid direct dependencies between them.
* **Loose Coupling:** You want to promote loose coupling between objects, allowing them to interact without knowing the details of each other’s implementations.
* **Centralized Control:** You need a centralized mechanism to coordinate and control the interactions between objects, ensuring a more organized and maintainable system.
* **Changes in Behavior:** You anticipate changes in the behavior of components, and you want to encapsulate these changes within the mediator, preventing widespread modifications.
* **Enhanced Reusability:** You want to reuse individual components in different contexts without altering their internal logic or communication patterns.

**When not to use the Mediator Design Pattern**

* **Simple Interactions:** The interactions between components are straightforward, and introducing a mediator would add unnecessary complexity.
* **Single Responsibility Principle (SRP):** Each component has a single responsibility, and introducing a mediator might violate the Single Responsibility Principle, leading to less maintainable code.
* **Performance Concerns:** Introducing a mediator could introduce a performance overhead, especially in situations where direct communication between components is more efficient.
* **Small Scale Applications:** In small-scale applications with a limited number of components, the overhead of implementing a mediator might outweigh its benefits.
* **Over-Engineering:** Avoid using the Mediator pattern if it seems like an over-engineered solution for the specific requirements of your system. Always consider the trade-offs and the specific needs of your application.



