Facilitates communications between components or objects (colleagues)

All communications are handled by the Mediator and the colleagues don’t need to know anything about each other.

Components in a system can go in and out at any time

* Chat room participants
* Players in an online game

It makes no sense for them to have direct references to one another .Those references may go dead.

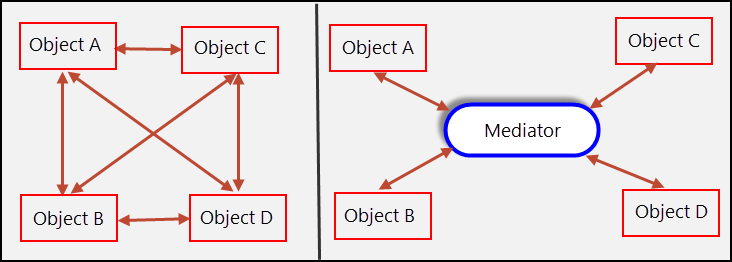
Solution: have them all refer to the same central component that facilitates communication

Mediator is a component that facilitates communication between other components without them necessarily being aware of each other or having direct (reference) access to each other

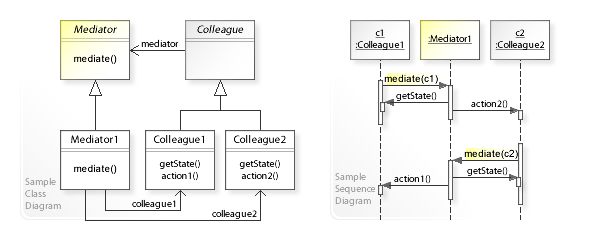
**GOF Definition**:

Allows loose coupling by encapsulating the way disparate sets of objects interact and communicate with each other.  Allows for the actions of each object set to vary independently of one another.

Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently



Class Diagram:



**Participants**

**Mediator** - defines the interface for communication between *Colleague* objects

**ConcreteMediator** - implements the Mediator interface and coordinates communication between *Colleague* objects. It is aware of all of the *Colleagues* and their purposes with regards to inter-communication.

**Colleague** - defines the interface for communication with other *Colleagues* through its *Mediator*

**ConcreteColleague** - implements the Colleague interface and communicates with other *Colleagues* through its *Mediator*

# Mediator vs Observer

The Observer pattern: Defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

The Mediator pattern: Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.

## 1. Without

* **Client1**: Hey **Subject**, when do you change?
* **Client2**: When did you change **Subject**? I have not noticed!
* **Client3**: I know that **Subject** has changed.

## 2. With

* **Clients** are silent.
* Sometime later ...
* **Subject**: Dear **clients**, I have changed!

# Mediator

## 1. Without

* **Client1**: Hey **Taxi1**, take me somewhere.
* **Client2**: Hey **Taxi1**, take me somewhere.
* **Client1**: Hey **Taxi2**, take me somewhere.
* **Client2**: Hey **Taxi2**, take me somewhere.

## 2. With

* **Client1**: Hey **TaxiCenter**, please take me a **Taxi**.
* **Client2**: Hey **TaxiCenter**, please take me a **Taxi**.

**Mediator vs Façade:**

The **facade** only exposes the existing functionality from a different perspective.

The **mediator** "adds" functionality because it combines different existing functionality to create a new one.

Facade differs from Mediator in that it abstracts a subsystem of objects to provide a more convenient interface. Its protocol is unidirectional; that is, Facade objects make requests of the subsystem classes but not vice versa. In contrast, Mediator enables cooperative behavior that colleague objects don't or can't provide, and the protocol is multidirectional.