# Overview:

* The Mediator design pattern defines an object that encapsulates how a set of objects interact:
  + Promotes loose coupling by keeping objects from referring to each other explicitly.
    - We can reduce the direct interconnections among the objects.
  + Let you vary their interaction independently.
* A Mediator is the one who takes the responsibility of communication among a group of objects.
  + Acts as an intermediary who can track the communication between two objects.
  + The other objects in the system are also aware of this mediator.
    - They know that if they need to communicate among themselves, they need to go through the mediator.
* Used to reduce communication complexity between multiple objects or classes.

# Examples:

* An air traffic controller is a great example:
  + The airport control room works as a mediator for communication between different flights.
  + Mediator works as a router between objects.
    - Has its own logic to provide way of communication.
* In an airplane application, before taking off the flight undergoes a series of checks
  + Checks confirm that all components/ parts (which are dependent to each other) are in perfect condition.
* You could use a mediator for a house of the future where appliances talk to each other.
  + When a user stops hitting the snooze button, the alarm clock tells the coffee maker to start brewing.
  + Turn off the sprinkler 15 minutes before a shower is scheduled.
  + Set the alarm early on trash days.

# JDK Examples:

* Java.util.Timer class scheduleXXX () methods.
* Java Concurrency Executor execute() method
* Java.lang.reflect.Method invoke method.
* Java Message Service (JMS) uses Mediator pattern along with Observer pattern to allow applications to subscribe and publish data to other applications.

# What problems the mediator can solve?

* Object-oriented design encourages the distribution of behavior among objects.
* Such distribution can result in an object structure with many connections between objects.
  + In the worst case, every object ends up knowing about every other.
* Lots of interconnections make it less likely that an object can work without the support of others
  + It can be difficult to change the system’s behavior in any significant way.
    - Since behavior is distributed among many objects.
* If we need to make some kind of change, it becomes a challenging task.
* The mediator pattern will help solve the above problems.

# Advantages:

* Increases the reusability of the objects supported by the mediator by decoupling them from the system.
* Simplifies maintenance of the system by centralizing control logic.
* Simplifies and reduces the variety of messages sent between objects in the system.
* Allows you to replace one object in the structure with a different one without affecting the classes and the interfaces.
* We should not use mediator pattern just to achieve lose-coupling
  + If the number of mediators grows, it the becomes hard to maintain them.

# When to use the Mediator?

* Use the mediator pattern to centralize complex communications and control between related objects.
* When a set of objects communicate in well-defined but complex ways
  + The resulting interdependencies are unstructured and difficult to understand.
* When reusing an object is difficult because it refers to and communicates with many other objects.
* When a behavior that is distributed between several classes should be customizable without a lot of sub-classing.

# Implementation:

# Participants:

* **Mediator:**
  + Defines an interface for communicating with Colleague objects.
* **ConcreteMediator:**
  + Implements cooperative behavior by coordinating Colleague objects.
  + Knows and maintains its colleagues.
* **Colleague classes:**
  + Each Colleague class knows its Mediator object.
  + Each Colleague communicates with its mediator whenever it would have otherwise communicated with another colleague.
* **Colleagues** send and receive requests from a Mediator object
  + Mediator implements the cooperative behavior by routing requests between the appropriate colleagues.
* **The system objects** that communicate with each other are called Colleagues
  + Usually we have an interface or abstract class that provides the contract for communication and then we have concrete implementation of mediators.

# Implementation issues:

* No need to define an abstract Mediator class when colleagues work with only one mediator
  + The abstract coupling that the Mediator class provides lets colleagues work with different Mediator subclasses, and vice versa.
* Colleagues have to communicate with their mediator when an event of interest occurs.
  + One approach is to implement the Mediator as an Observer.
  + Colleague classes act as Subjects
    - Send notifications to the mediator whenever they change state.
      * Mediator responds by propagating the effects of the change to other colleagues.

# Implementing advantages:

* The implementation limits sub-classing:
  + A mediator localizes behavior that otherwise would be distributed among several objects
  + Changing this behavior requires subclassing Mediator only.
    - Colleague classes can be reused as is.
* It decouples colleagues:
  + A mediator promotes loose coupling between colleagues
  + You can vary and reuse Colleague and Mediator classes independently.
* It simplifies object protocols:
  + Replaces many-to-many interactions with one-to-many interactions between the mediator and its colleagues
  + One-to-many relationships are easier to understand, maintain, and extend.
* It abstracts how objects cooperate:
  + Making mediation an independent concept and encapsulating it in an object lets you focus on how objects interact apart from their individual behavior
    - Can help clarify how objects interact in a system.
* It centralizes control:
  + Trades complexity of interaction for complexity in the mediator
    - A mediator can become more complex than any individual colleague.
    - Can make the mediator itself a monolith that is hard to maintain.