









**🧩 What is the Mediator Pattern?**

The **Mediator Pattern** defines an object (the **Mediator**) that **centralizes complex communications and control logic** between objects (called **Colleagues**) so that they **don’t refer to each other directly**.

It **promotes loose coupling** by keeping objects from referring to each other explicitly.

**🧠 Real-World Analogy**

Think of an **air traffic controller** (Mediator).  
Planes (Colleagues) do **not communicate directly** with each other — they talk via the **controller**.

**✅ Use Cases of Mediator Pattern**

| **Use Case** | **Example** |
| --- | --- |
| ✅ GUI components | Button click updates textbox and label |
| ✅ Chat applications | Users send messages via a central server |
| ✅ Workflow engines | Steps communicate via workflow engine, not directly |
| ✅ Game development | Characters or objects interact via central game manager |
| ✅ Messaging systems | Components communicate via message broker |

**🧰 Components of Mediator Pattern**

| **Component** | **Role** |
| --- | --- |
| **Mediator (interface)** | Declares communication methods |
| **ConcreteMediator** | Implements how objects interact |
| **Colleague** | Objects communicating via the mediator |

**✅ Advantages**

| **Advantage** | **Description** |
| --- | --- |
| ✅ **Loose coupling** | Components don’t need to know about each other |
| ✅ **Centralized logic** | All coordination is in one place |
| ✅ **Improved maintainability** | Easier to update interactions |
| ✅ **Scalability** | Add new components without affecting others |

**❌ Disadvantages**

| **Disadvantage** | **Description** |
| --- | --- |
| ❌ **Complex Mediator** | Mediator can become large and difficult to maintain |
| ❌ **Single point of failure** | All communication goes through one class |
| ❌ **Can violate SRP** | Mediator may handle too many responsibilities |

**✅ Summary Table**

| **Feature** | **Description** |
| --- | --- |
| Type | Behavioral pattern |
| Key Concept | Centralizes communication between components |
| When to Use | Many components interact in complex ways |
| Java Analogy | java.util.Timer (mediates scheduling) |
| Alternatives | Observer Pattern (for 1-to-many updates) |

## ✅ Java Example: Chat Room (Basic)

### 1. ****Mediator Interface****

java

CopyEdit

public interface ChatMediator {

void sendMessage(String msg, User user);

void addUser(User user);

}

### 2. ****Concrete Mediator****

java

CopyEdit

import java.util.ArrayList;

import java.util.List;

public class ChatRoom implements ChatMediator {

private List<User> users = new ArrayList<>();

public void addUser(User user) {

users.add(user);

}

public void sendMessage(String msg, User sender) {

for (User user : users) {

if (user != sender) {

user.receive(msg);

}

}

}

}

### 3. ****Colleague (User)****

java

CopyEdit

public abstract class User {

protected ChatMediator mediator;

protected String name;

public User(ChatMediator mediator, String name) {

this.mediator = mediator;

this.name = name;

}

public abstract void send(String msg);

public abstract void receive(String msg);

}

### 4. ****Concrete Colleague****

java

CopyEdit

public class ChatUser extends User {

public ChatUser(ChatMediator mediator, String name) {

super(mediator, name);

}

public void send(String msg) {

System.out.println(this.name + " sends: " + msg);

mediator.sendMessage(msg, this);

}

public void receive(String msg) {

System.out.println(this.name + " receives: " + msg);

}

}

### 5. ****Client Code****

java

CopyEdit

public class MediatorPatternDemo {

public static void main(String[] args) {

ChatMediator chatRoom = new ChatRoom();

User user1 = new ChatUser(chatRoom, "Alice");

User user2 = new ChatUser(chatRoom, "Bob");

User user3 = new ChatUser(chatRoom, "Charlie");

chatRoom.addUser(user1);

chatRoom.addUser(user2);

chatRoom.addUser(user3);

user1.send("Hello, everyone!");

}

}