**The Mediator design pattern** helps different objects communicate with each other without them needing to know too much about each other. Imagine a chat room where people can talk to each other, but they don't directly talk to each other. Instead, they send their messages to a central person (the mediator) who then relays the messages to the intended recipients. This way, everyone talks to the mediator, and the mediator handles the distribution of messages.

**Mediator Interface (IChatMediator):**

Think of this as the central "chat room manager." It has a rule: anyone who wants to send a message must use the SendMessage method.

**Concrete Mediator (ChatRoom):**

This is like the actual chat room in our analogy. It implements the IChatMediator rule, which means it knows how to take messages and pass them to the right people.

**Colleague Classes (User):**

These are the people in our chat room analogy. Each person (user) knows how to send a message, but they don't need to know how the message is delivered to others. They send messages to the mediator, who then makes sure the messages get to the right people.

**Client Code (Main):**

Here, we're creating the chat room (mediator) and users (people) and having them interact. Users are sending messages, but they're doing it through the chat room (mediator). This keeps things organized and ensures that users don't need to worry about finding each other.

In simple English, the Mediator pattern helps objects communicate without talking directly to each other. It's like having a central hub (the mediator) that handles messages between objects. This makes communication more organized and reduces the need for objects to know too much about each other. Just like in a chat room, you send your message to the mediator, and the mediator takes care of passing it on to the right recipients.

**Step 1: Define the Mediator Interface**

The IChatMediator interface defines the mediator's contract. In this case, it has a single method SendMessage(User sender, string message) that mediates the communication between users.

public interface IChatMediator

{

void SendMessage(User sender, string message);

}

**Step 2: Implement the Concrete Mediator**

The ChatRoom class implements the IChatMediator interface. It is responsible for facilitating communication between users by relaying messages.

public class ChatRoom : IChatMediator

{

public void SendMessage(User sender, string message)

{

Console.WriteLine($"{sender.Name} sends message: {message}");

}

}

**Step 3: Create Colleague Classes**

The User class represents the colleagues (participants) in the communication. It holds the user's name and a reference to the mediator. The SendMessage method within the User class delegates sending messages to the mediator.

public class User

{

public string Name { get; }

private IChatMediator \_mediator;

public User(string name, IChatMediator mediator)

{

Name = name;

\_mediator = mediator;

}

public void SendMessage(string message)

{

\_mediator.SendMessage(this, message);

}

}

**Step 4: Client Code (Main Program)**

In the Main method, you create instances of the mediator (ChatRoom) and users (User) and associate each user with the same mediator. Users can send messages using the mediator without needing to know the details of how the messages are relayed.

static void Main(string[] args)

{

IChatMediator chatRoom = new ChatRoom();

User user1 = new User("User 1", chatRoom);

User user2 = new User("User 2", chatRoom);

User user3 = new User("User 3", chatRoom);

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

user2.SendMessage("Hi there!");

user3.SendMessage("Hey, folks!");

}

**Mediator Pattern**: The Mediator pattern promotes loose coupling between objects by introducing a mediator that handles interactions between objects. Colleague objects (in this case, User instances) do not communicate directly with each other but instead communicate through the mediator (IChatMediator), reducing direct dependencies.

**Step 1**: The IChatMediator interface defines the mediator's role, which is to mediate communication between users by relaying messages.

**Step 2**: The ChatRoom class is a concrete implementation of the mediator interface. It handles the actual message relaying, decoupling users from each other.

**Step 3**: The User class represents colleagues. It has a reference to the mediator (\_mediator) through which it communicates. The SendMessage method delegates the message-sending responsibility to the mediator.

**Step 4**: In the Main method, users are created and associated with the same mediator (chatRoom). When users send messages, they do so by calling SendMessage on the mediator, which in turn relays the messages to other users.

By using the Mediator pattern, your code achieves better separation of concerns, easier extensibility, and reduced dependencies between objects communicating with each other. This pattern is especially useful when you have a complex communication structure where many objects interact, and you want to centralize and control the interactions.