

[JAVA TUTORIAL](#)[#INDEX POSTS](#)[#INTERVIEW QUESTIONS](#)[RESOURCES](#)[HIRE ME](#)[DOWNLOAD ANDROID APP](#)[CONTRIBUTE](#)**Subscribe to Download Java Design Patterns eBook****DOWNLOAD NOW**[HOME](#) » [JAVA](#) » [DESIGN PATTERNS](#) » [MEDIATOR DESIGN PATTERN IN JAVA](#)

# Mediator Design Pattern in Java

APRIL 4, 2018 BY [PANKAJ](#) — [21 COMMENTS](#)

Mediator [design pattern](#) is one of the behavioral design pattern, so it deals with the behaviors of objects. Mediator design pattern is used to provide a centralized communication medium between different objects in a system.

## Table of Contents [\[hide\]](#)

- [1 Mediator Design Pattern](#)
  - [1.1 Mediator Pattern Interface](#)
  - [1.2 Mediator Pattern Colleague Interface](#)
  - [1.3 Concrete Mediator](#)
  - [1.4 Mediator Design Pattern Concrete Colleague](#)
  - [1.5 Mediator Pattern Example Client Program Code](#)
  - [1.6 Mediator Pattern Class Diagram](#)
  - [1.7 Mediator Pattern Example in JDK](#)
  - [1.8 Mediator Design Pattern Important Points](#)

## Mediator Design Pattern



According to GoF, mediator pattern intent is:

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.

Mediator design pattern is very helpful in an enterprise application where multiple objects are interacting with each other. If the objects interact with each other directly, the system components are tightly-coupled with each other that makes higher maintainability cost and not hard to extend. Mediator pattern focuses on provide a mediator between objects for communication and help in implementing lose-coupling between objects.

Air traffic controller is a great example of mediator pattern where the airport control room works as a mediator for communication between different flights. Mediator works as a router between objects and it can have it's own logic to provide way of communication.

The system objects that communicate 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.

For our example, we will try to implement a chat application where users can do group chat. Every user will be identified by it's name and they can send and receive messages. The message sent by any user should be received by all the other users in the group.

## Mediator Pattern Interface

First of all we will create Mediator interface that will define the contract for concrete mediators.

ChatMediator.java

```
package com.journaldev.design.mediator;

public interface ChatMediator {

    public void sendMessage(String msg, User user);

    void addUser(User user);
}
```

## Mediator Pattern Colleague Interface

Users can send and receive messages, so we can have User interface or abstract class. I am creating User as abstract class like below.

User.java

```
package com.journaldev.design.mediator;

public abstract class User {
    protected ChatMediator mediator;
    protected String name;

    public User(ChatMediator med, String name){
        this.mediator=med;
        this.name=name;
    }

    public abstract void send(String msg);

    public abstract void receive(String msg);
}
```

Notice that User has a reference to the mediator object, it's required for the communication between different users.

## Concrete Mediator

Now we will create concrete mediator class, it will have a list of users in the group and provide logic for the communication between the users.

ChatMediatorImpl.java

```
package com.journaldev.design.mediator;

import java.util.ArrayList;
import java.util.List;

public class ChatMediatorImpl implements ChatMediator {

    private List<User> users;

    public ChatMediatorImpl(){
        this.users=new ArrayList<>();
    }

    @Override
    public void addUser(User user){
        this.users.add(user);
    }

    @Override
    public void sendMessage(String msg, User user) {
        for(User u : this.users){
            //message should not be received by the user sending it
        }
    }
}
```

## Mediator Design Pattern Concrete Colleague

Now we can create concrete User classes to be used by client system.

UserImpl.java

```
package com.journaldev.design.mediator;

public class UserImpl extends User {

    public UserImpl(ChatMediator med, String name) {
        super(med, name);
    }

    @Override
    public void send(String msg){
        System.out.println(this.name+": Sending Message="+msg);
        mediator.sendMessage(msg, this);
    }

    @Override
}
```

```
        public void receive(String msg) {  
            System.out.println(this.name+": Received Message:"+msg);  
        }  
    }  
}
```

Notice that send() method is using mediator to send the message to the users and it has no idea how it will be handled by the mediator.

## Mediator Pattern Example Client Program Code

Let's test this our chat application with a simple program where we will create mediator and add users to the group and one of the user will send a message.

ChatClient.java

```
package com.journaldev.design.mediator;  
  
public class ChatClient {  
  
    public static void main(String[] args) {  
        ChatMediator mediator = new ChatMediatorImpl();  
        User user1 = new UserImpl(mediator, "Pankaj");  
        User user2 = new UserImpl(mediator, "Lisa");  
        User user3 = new UserImpl(mediator, "Saurabh");  
        User user4 = new UserImpl(mediator, "David");  
        mediator.addUser(user1);  
        mediator.addUser(user2);  
        mediator.addUser(user3);  
        mediator.addUser(user4);  
  
        user1.send("Hi All");  
    }  
}
```

Notice that client program is very simple and it has no idea how the message is getting handled and if mediator is getting user or not.

Output of the mediator pattern example program is:

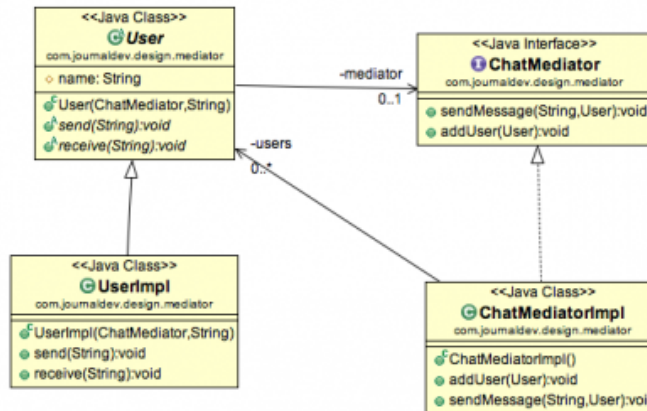
Pankaj: Sending Message=Hi All

Lisa: Received Message:Hi All

Saurabh: Received Message:Hi All

David: Received Message:Hi All

## Mediator Pattern Class Diagram



## Mediator Pattern Example in JDK

- `java.util.Timer` class `scheduleXXX()` methods
- `Java Concurrency Executor` `execute()` method.
- `java.lang.reflect.Method` `invoke()` method.

## Mediator Design Pattern Important Points

- Mediator pattern is useful when the communication logic between objects is complex, we can have a central point of communication that takes care of communication logic.
- Java Message Service (JMS) uses Mediator pattern along with **Observer pattern** to allow applications to subscribe and publish data to other applications.
- We should not use mediator pattern just to achieve loose-coupling because if the number of mediators will grow, then it will become hard to maintain them.

That's all for mediator design pattern and its implementation in java.

**« PREVIOUS**[Iterator Design Pattern in Java](#)**NEXT »**[Memento Design Pattern in Java](#)**About Pankaj**

If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on [Google Plus](#), [Facebook](#) or [Twitter](#). I would love to hear your thoughts and opinions on my articles directly.

Recently I started creating video tutorials too, so do check out my videos on [Youtube](#).

---

FILED UNDER: [DESIGN PATTERNS](#)

**Comments****fatemeh says**[OCTOBER 26, 2017 AT 7:53 AM](#)

hello,please help me  
i want to know, how use jms from mediator pattern?  
thankful

[Reply](#)

**robothy says**

NOVEMBER 22, 2016 AT 10:23 PM

Nice example.

[Reply](#)

**David says**

NOVEMBER 1, 2016 AT 12:27 PM

I knew at this and I can't see the differences between mediator pattern and observe pattern

[Reply](#)

**Hari K Bista says**

AUGUST 5, 2017 AT 2:09 AM

you are right.

[Reply](#)

**Hiccup says**

OCTOBER 2, 2017 AT 10:20 AM

I think the difference between mediator and observer is

mediator : intent is any object want to initiate communication but in observation subject state triggers and it's mainly broadcast..

here i think example is taken is not compete use of mediator. Mediator is not about broadcast always..it's a selective communication. Interested object communicate and that mapping is with the mediator either in the form of hash map or any other data structure...

[Reply](#)

**vinod says**

MAY 18, 2018 AT 11:26 AM

In observer, many objects are interested in the state change of one object. They are not interested in each other. So the relation is one to many.



In mediator, many objects are interested to communicate many other objects. Here the relation is many to many.

[Reply](#)

**Annappa says**

MARCH 25, 2016 AT 5:07 AM

Nice explanation.

[Reply](#)

**Ashwin says**

SEPTEMBER 27, 2015 AT 7:34 AM

Really crisp and good writeups

[Reply](#)

**Aditya says**

SEPTEMBER 14, 2015 AT 4:27 AM

how mediator pattern is implemented in ExecutorService API. Kindly explain

[Reply](#)

**mohammad says**

JULY 31, 2015 AT 1:55 PM

hi.

tanck you for your nice tutorial.

[Reply](#)

**Mohit Gupta says**

JULY 31, 2015 AT 6:26 AM

Awesome explanation thanks ☐

[Reply](#)

**abc says**

APRIL 24, 2015 AT 5:46 AM

who is the mediator here?

[Reply](#)**Leo says**

MAY 23, 2015 AT 10:35 AM

ChatMediator.

[Reply](#)**Avinash Pandey says**

FEBRUARY 18, 2015 AT 4:12 AM

Your tutorials are nice and explanatory.

[Reply](#)**Ilia Sokolovski says**

JANUARY 20, 2015 AT 9:01 AM

Hey,

That was actually very helpful, Thanks!

[Reply](#)**Lalit Upadheyay says**

DECEMBER 23, 2014 AT 8:34 PM

I think that the line(s) mediator.addUser in ChatClient can be avoided by delegating this call in User constructor by appending one line

```
this.mediator.addUser(this);
```

Also we should implement Mediator as a singleton, to avoid scenario when different mediator instances can be tied to a participating object. This would cause synchronized message relay issue and the implementation will fail to serve its purpose.

[Reply](#)**Rishi says**

MARCH 7, 2016 AT 7:56 AM

Agree 100%

[Reply](#)**venkatesh says**

JUNE 22, 2017 AT 4:52 AM

how would we use multiple group chats if mediator is singleton

[Reply](#)**Shyam says**

MAY 2, 2016 AT 11:19 PM

This is an classic example for people to understand what is the use of mentioned designed pattern.

Later after understanding its up to one how to use his knowledge.

If you are so great at making code perfect, make an tutorial and help others.

Please don't boast too much.

[Reply](#)**Koteshwar says**

SEPTEMBER 7, 2014 AT 4:33 PM

Excellent..Thank you so much ☐

[Reply](#)**Krishna says**

JANUARY 17, 2014 AT 7:48 AM

Thanks for best and easiest example with explanation

[Reply](#)

## Leave a Reply

Your email address will not be published. Required fields are marked \*

Comment

Name \*

Email \*

☐

Save my name, email, and website in this browser for the next time I comment.

---

[DOWNLOAD ANDROID APP](#)

---



---

[DESIGN PATTERNS TUTORIAL](#)

---

## Java Design Patterns

### Creational Design Patterns

- > [Singleton](#)
- > [Factory](#)
- > [Abstract Factory](#)
- > [Builder](#)
- > [Prototype](#)

### Structural Design Patterns

- > [Adapter](#)

- > [Composite](#)
- > [Proxy](#)
- > [Flyweight](#)
- > [Facade](#)
- > [Bridge](#)
- > [Decorator](#)

Behavioral Design Patterns

- > [Template Method](#)
- > [Mediator](#)
- > [Chain of Responsibility](#)
- > [Observer](#)
- > [Strategy](#)
- > [Command](#)
- > [State](#)
- > [Visitor](#)
- > [Interpreter](#)
- > [Iterator](#)
- > [Memento](#)

Miscellaneous Design Patterns

- > [Dependency Injection](#)
- > [Thread Safety in Java Singleton](#)

RECOMMENDED TUTORIALS

Java Tutorials

- > [Java IO](#)
- > [Java Regular Expressions](#)
- > [Multithreading in Java](#)
- > [Java Logging](#)
- > [Java Annotations](#)
- > [Java XML](#)
- > [Collections in Java](#)
- > [Java Generics](#)
- > [Exception Handling in Java](#)
- > [Java Reflection](#)
- > [Java Design Patterns](#)
- > [JDBC Tutorial](#)

Java EE Tutorials

- > [Servlet JSP Tutorial](#)
- > [Struts2 Tutorial](#)
- > [Spring Tutorial](#)
- > [Hibernate Tutorial](#)
- > [Primefaces Tutorial](#)
- > [Apache Axis 2](#)
- > [JAX-RS](#)
- > [Memcached Tutorial](#)

**Oracle Database Tools**

---

**Java Programming Courses**

---

**Open Source Code**

---

**Medical Coding Company**

---

**Medical Coding Outsourcing**

---

**Learn Programming Online**

---

**Employer Interview Questions**

---

**Web Design Tutorials**

