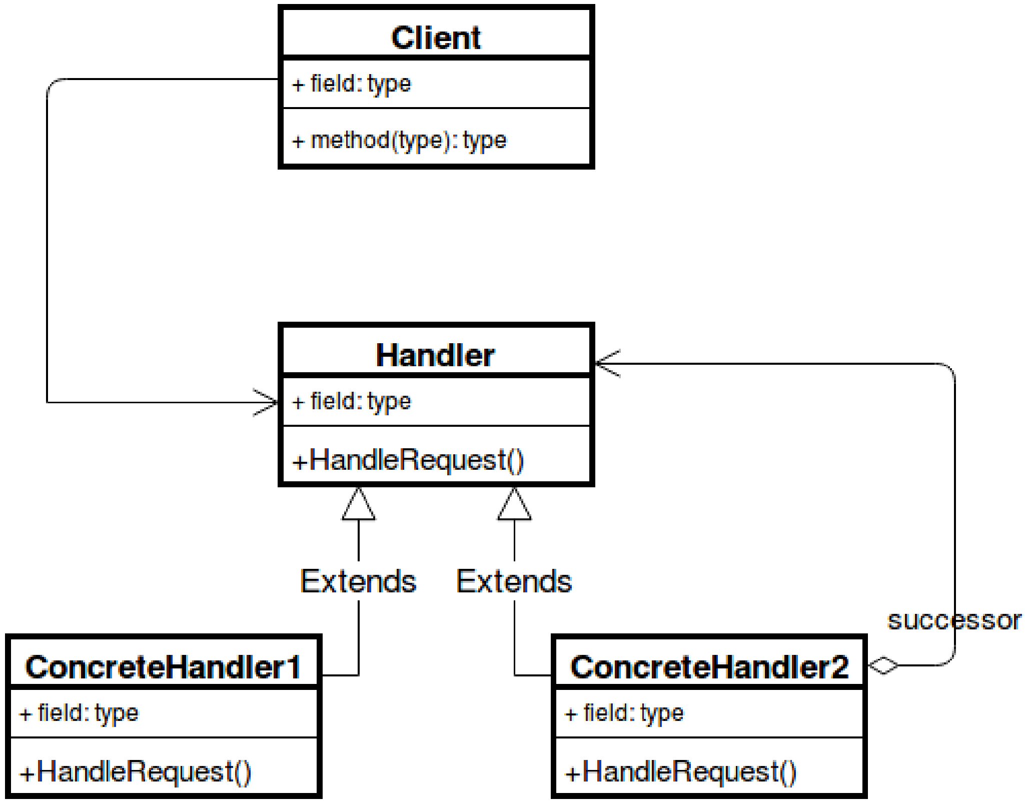
Chain of Responsibility Design Pattern

Chain of responsibility pattern is used to achieve loose coupling in software design where a request from client is passed to a chain of objects to process them. Later, the object in the chain will decide themselves who will be processing the request and whether the request is required to be sent to the next object in the chain or not.

Where and When Chain of Responsibility pattern is applicable :

- When you want to decouple a request's sender and receiver
- Multiple objects, determined at runtime, are candidates to handle a request
- When you don't want to specify handlers explicitly in your code
- When you want to issue a request to one of several objects without specifying the receiver explicitly.

This pattern is recommended when multiple objects can handle a request and the handler doesn't have to be a specific object. Also, handler is determined at runtime. Please note that that a request not handled at all by any handler is a valid use case.



- **Handler :** This can be an interface which will primarily recieve the request and dispatches the request to chain of handlers. It has reference of only first handler in the chain and does not know anything about rest of the handlers.
- **Concrete handlers :** These are actual handlers of the request chained in some sequential order.
- **Client :** Originator of request and this will access the handler to handle it.

How to send a request in the application using the Chain of Responsibility

The Client in need of a request to be handled sends it to the chain of handlers which are classes that extend the Handler class. Each of the handlers in the chain takes its turn at trying to handle the request it receives from the client.

If ConcreteHandler1 can handle it, then the request is handled, if not it is sent to the handler ConcreteHandler2, the next one in the chain.

Lets see an Example of Chain of Responsibility Design Pattern:

```
interface Chain
{
    public abstract void setNext(Chain nextInChain);
    public abstract void process(Number request);
}

class Number
{
    private int number;

    public Number(int number)
    {
        this.number = number;
    }

    public int getNumber()
    {
        return number;
    }

}

class NegativeProcessor implements Chain
{
    private Chain nextInChain;

    public void setNext(Chain c)
    {
        nextInChain = c;
    }

    public void process(Number request)
    {
        if (request.getNumber() < 0)
        {
            System.out.println("NegativeProcessor : " + request.getNumber());
        }
        else
        {
            nextInChain.process(request);
        }
    }
}

class ZeroProcessor implements Chain
{

    private Chain nextInChain;

    public void setNext(Chain c)
    {
        nextInChain = c;
    }

    public void process(Number request)
    {
        if (request.getNumber() == 0)
        {
            System.out.println("ZeroProcessor : " + request.getNumber());
        }
        else
        {
            nextInChain.process(request);
        }
    }
}

class PositiveProcessor implements Chain
{

    private Chain nextInChain;

    public void setNext(Chain c)
    {
        nextInChain = c;
    }
```

```
        public void process(Number request)
        {
            if (request.getNumber() > 0)
            {
                System.out.println("PositiveProcessor : " + request.getNumber());
            }
            else
            {
                nextInChain.process(request);
            }
        }
    }

    class TestChain
    {
        public static void main(String[] args) {
            //configure Chain of Responsibility
            Chain c1 = new NegativeProcessor();
            Chain c2 = new ZeroProcessor();
            Chain c3 = new PositiveProcessor();
            c1.setNext(c2);
            c2.setNext(c3);

            //calling chain of responsibility
            c1.process(new Number(90));
            c1.process(new Number(-50));
            c1.process(new Number(0));
            c1.process(new Number(91));
        }
    }
```

Output :

```
PositiveProcessor : 90
NegativeProcessor : -50
ZeroProcessor : 0
PositiveProcessor : 91
```

Advantages of Chain of Responsibility Design Pattern

- To reduce the coupling degree. Decoupling it will request the sender and receiver.
- Simplified object. The object does not need to know the chain structure.
- Enhance flexibility of object assigned duties. By changing the members within the chain or change their order, allow dynamic adding or deleting responsibility.
- Increase the request processing new class of very convenient.

DisAdvantages of Chain of Responsibility Design Pattern

- The request must be received not guarantee.
- The performance of the system will be affected, but also in the code debugging is not easy may cause cycle call.
- It may not be easy to observe the characteristics of operation, due to debug.

This article is contributed by **Saket Kumar**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Recommended Posts:

Design Patterns | Set 1 (Introduction)

Design Patterns | Set 2 (Factory Method)

Command Pattern

Observer Pattern | Set 1 (Introduction)

Observer Pattern | Set 2 (Implementation)

Singleton Design Pattern | Implementation

Decorator Pattern | Set 1 (Background)

The Decorator Pattern | Set 2 (Introduction and Design)

Decorator Pattern | Set 3 (Coding the Design)

Strategy Pattern | Set 1 (Introduction)

Strategy Pattern | Set 2 (Implementation)

Adapter Pattern

Iterator Pattern

Curiously recurring template pattern (CRTP)

Flyweight Design Pattern

Article Tags : Design Pattern

1

To-do Done

1.6

Based on **3** vote(s)

Feedback/ Suggest Improvement Add Notes Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

Share this post!

▲