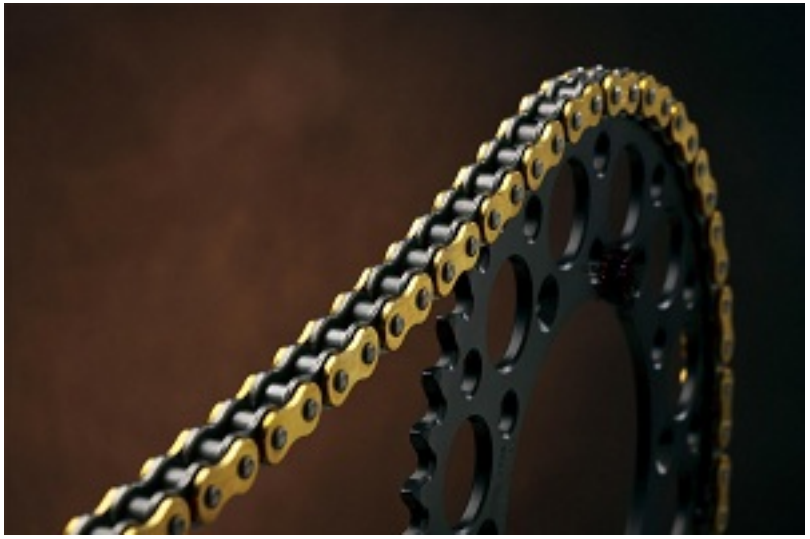
Chain of Responsibility Design Pattern

Decoupling is one of the prominent mantras in software engineering. Chain of responsibility helps to decouple sender of a request and receiver of the request with some trade-offs. Chain of responsibility is a design pattern where a sender sends a request to a chain of objects, where the objects in the chain decide themselves who to honor the request. If an object in the chain decides not to serve the request, it forwards the request to the next object in the chain.



Responsibility is outsourced. In a chain of objects, the responsibility of deciding who to serve the request is left to the objects participating in the chains. It is similar to ‘passing the question in a quiz scenario’. When the quiz

master asks a question to a person, if he doesn't knows the answer, he passes the question to next person and so on. When one person answers the question, the passing flow stops. Sometimes, the passing might reach the last person and still nobody gives the answer.

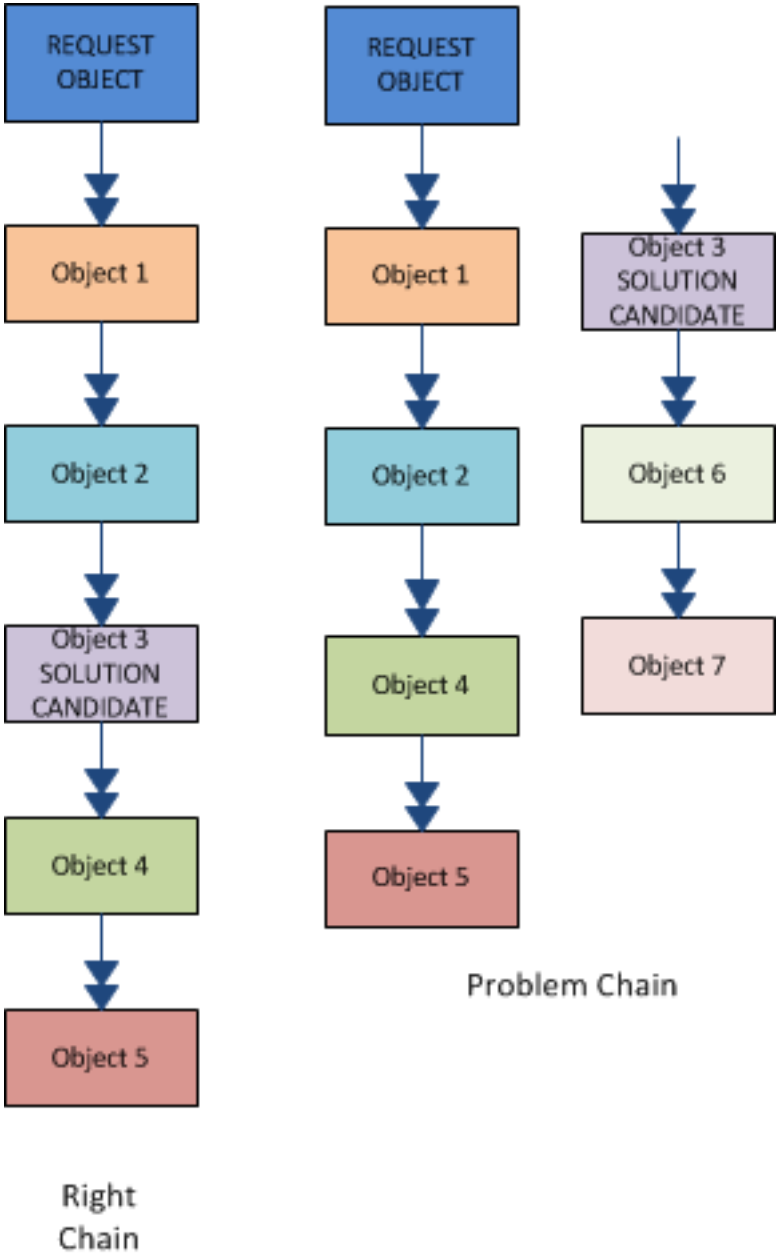
Welcome to behavioral design patterns. This is the first tutorial in behavioral category of our famous design pattern series.

Highlights of Chain of Responsibility

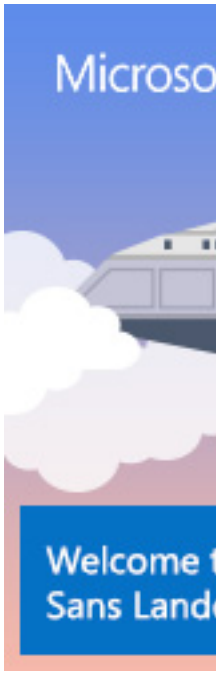
- Sender will not know which object in the chain will serve its request.
- Every node in chain will have the responsibility to decide, if they can serve the request.
- If node decides to forward the request, it should be capable of choosing the next node and forward it.
- There is a possibility where none of the node may serve the request.

Problems in Chain of Responsibility

There may be scenarios where a node is capable of solving the request but may not get a chance for it. Though there is a candidate who can solve the problem, but since nobody forwarded the request to it, it was not given a chance to serve and final result is the request goes unattended failure. This happens because of improper chain sequence. A chain sequence may not be suitable for all scenarios.



In object oriented design generally, every object is responsible for all its behaviour. Behaviour of an object is not transferred to other objects and is enclosed within itself. In chain of responsibility, some percentage of behaviour is offloaded to third party objects.



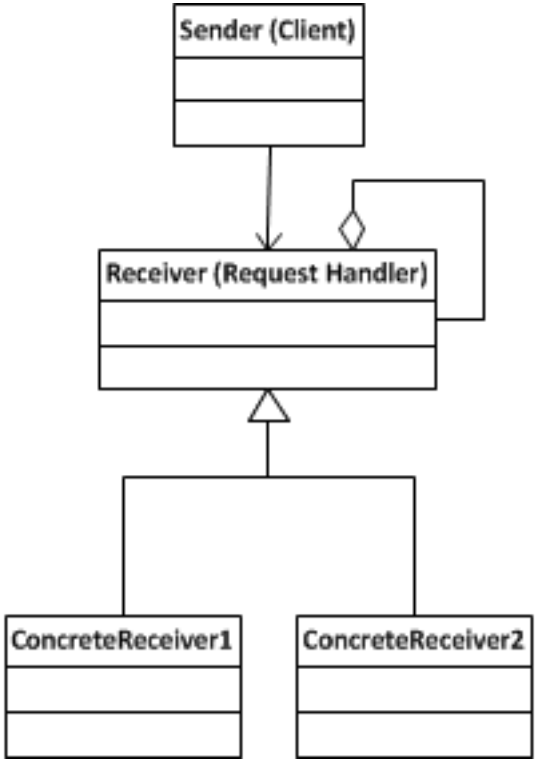
Chain of Responsibility Example

When thinking about nice examples for chain of responsibility pattern following list came to my mind. Coin sorting machine, ATM money dispenser, Servlet Filter and finally java's own Exception Handling mechanism. We know exception handling better than anybody else and we are daily living with it. This qualifies as the best example for chain of responsibility.

We may have sequence of exceptions listed in catch statements and when there is an exception thrown, the

catch list is scanned one by one from top. If first exception in catch can handle it the job is done, else the responsibility is moved to next in line and so on till it reaches finally block if we have one.

UML Diagram for Chain of Responsibility



Example Java Code

Following example code gives a sample implementation of chain of responsibility.

Chain.java

This is the interface that acts as a chain link.

```
package com.javapapers.designpattern.chainofresponsibility;

public interface Chain {

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

Number.java

This class is the request object.

```
package com.javapapers.designpattern.chainofresponsibility;

public class Number {
        private int number;

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

        public int getNumber() {
                return number;
        }
```

```
}
```

NegativeProcessor.java

This class is a link in chain series.

```
package com.javapapers.designpattern.chainofresponsibility;

public 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);
                }
        }
}
```

ZeroProcessor.java

This class is another link in chain series.

```
package com.javapapers.designpattern.chainofresponsibility;

public 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.getN
umber());
                } else {
                        nextInChain.process(request);
                }
        }
}
```

PositiveProcessor.java

This class is another link in chain series.

```
package com.javapapers.designpattern.chainofresponsibility;

public 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);
                }
        }
}
```

TestChain.java

This class configures the chain of responsibility and executes it.

```
package com.javapapers.designpattern.chainofresponsibility;

public 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(99));
                c1.process(new Number(-30));
                c1.process(new Number(0));
                c1.process(new Number(100));
        }
}
```

Output for chain of responsibility example

PositiveProcessor : 99

NegativeProcessor : -30

ZeroProcessor : 0

PositiveProcessor : 100

Download Source Code

Chain Of Responsibility Java Source Code

Use of Chain of Responsibility in JDK

- javax.servlet.Filter#doFilter()
- java.util.logging.Logger#log

This Behavioral Design Pattern tutorial was added on 05/08/2012.

« Dependency Injection (DI) with Spring

java vs javaw vs javaws »

Comments on “Chain of Responsibility Design Pattern”

saleem *06/08/2012 at 12:22 pm*

excellent and makes very easy to understand the concept, tnx a lot.

Reply

---

vikas *29/05/2014 at 8:53 am*

Thankyou !!!

Reply

Pradeep *06/08/2012 at 12:25 pm*

Thanks Joe for this blog…very well explained.

Reply

Irshad *06/08/2012 at 1:09 pm*

well explained. Thank you. Did not get what logic used for Coin sorting machine, ATM money dispenser?

Reply

Bharani *06/08/2012 at 1:22 pm*

@Irshad: Say if you want to withdraw 2200 Rs, and ATM has 1000, 500 and 100 rupee notes. As per chain of responsibility the request would be sent to object that dispenses 1000 rs note. If 1000 ruppee notes are NOT available then it would be sent to an object that dispenses 500 rs note. The 500Rs object decides to dispense 4 notes and then 100 rs object would be asked to handle the rest. Advantage is, eventhough there are no 1000 rs not avaialble, your req would be still handled by other objects like 500 and 100.

Reply

Shahjahan *06/08/2012 at 2:36 pm*

Hi Bharani

But in ATM when we withdraw 1000 then is in form of one 500 notes and rest are 100 note.

here how it follow the chain rules

Reply

---

sayeed *06/08/2012 at 3:08 pm*

Hi Joe,
It would be better if you put the search box , which is right now in the bottom of the page to above on inside “Stay in touch” box. It will make more easy to search content on your page.

Reply

Ajit *06/08/2012 at 3:15 pm*

Hi Shahjahan,
Here ATM software has been made to implement a customized form of Chain of Responsibility for ATM user convenience. So if you were to withdraw Rs 1000, the first Rs 500 is mandated to be served by the 100 dispenser. Then the balance 500 will follow the Chain of Responsibility pattern.
If it was not customized then you would have starved for the change and probably scold the ATM machine :)
Hope I could clarify.

Reply

Ashish *15/07/2013 at 12:00 am*

best example of Combination of

chain of responsibility with decorator pattern.

where responsibility is shared in a modified manner.

Reply

Anonymous *06/08/2012 at 7:06 pm*

Hi Joe,

Awesome!

-Jacob

Reply

---

vikas *06/08/2012 at 10:01 pm*

Thanks ….

geting the clear understanding on chain of responsibility ….

Reply

Bharani *07/08/2012 at 11:09 am*

@Irshad: The example i've used it for pure illustration purpose to explain application of CoR. In reality, dispenser uses the Chain of responsiblity design pattern along with certain BUSINESS RULES in order to use the notes optimally. Infact what you have pointed out, is an advantage of CoR as it is FLEXIBLE and CUSTOMIZABLE.

Reply

Sandeep *07/08/2012 at 4:27 pm*

Good One .. easy to understand

Reply

Anonymous *08/08/2012 at 12:18 pm*

hi joe ,

its really intresting ,easy, understandable iam so happy for your blog ….

Reply

Mr. Ballem *08/08/2012 at 2:49 pm*

Could you please explain JVM “Runtime Data Areas” in a Nut Shell.

Reply

Prasad *08/08/2012 at 6:16 pm*

could you please tell me, is there any hierarchy we need to follow for learning design patterns. like first this pattern should learn after that this one like that any hierarchy is there?

Reply

Anonymous *09/08/2012 at 12:11 pm*

A very good article that helps me understand so obscure concept! Thanks, Joe!

Reply

Hardik *14/08/2012 at 12:41 am*

Very useful,

Just little question is what is the difference between this design approach and traditional if else approach?

Can you help me to understand it more deeply.

Reply

Anonymous *16/08/2012 at 2:30 pm*

thanks a ton. awesome article

Reply

LittlePhoenix *17/08/2012 at 8:19 am*

```
public void process(Number request) {
if (request.getNumber() > 0) {
…
} else {
nextInChain.process(request); ???
}
}
```

Should we check nextInChain not null before call this ? Otherwise it shall throws NullPointerException

Reply

Louftansa *21/08/2012 at 3:50 pm*

Super clear, Joe. Thanks, it may be useful for a problem I'm encountering!!!

Reply

Pawan *25/08/2012 at 11:33 pm*

Just too good.

Reply

Alessandra *28/08/2012 at 7:18 am*

Thanks, Joe.

Your blog will be my first source in this topic. Your explanations are very clarifying.

Reply

Shubhangi *30/08/2012 at 12:34 pm*

Too descriptive and useful

Reply

Anonymous *06/09/2012 at 6:45 pm*

too good continue ur explanations.

Reply

Anonymous *06/10/2012 at 3:42 pm*

Thanks Joe..Very nice & precise description

Reply

Зоран *11/10/2012 at 8:11 pm*

I'm impressed. Your every design pattern I perused. How about you go ahead and explain all the GoF design patterns?
Zoran, Belgrade

Reply

Anonymous *27/10/2012 at 7:07 pm*

pls keep behavioural patterns as soon as possible..please…

Reply

Anonymous *01/11/2012 at 11:55 pm*

goo one !

thanks…

Abhrajyoti

Reply

Anonymous *02/11/2012 at 4:15 pm*

you are cool man!!

Reply

Natasha *09/11/2012 at 11:43 pm*

Really cool!

Thank you So mucn!

Reply

fachhoch *20/12/2012 at 12:27 am*

instead of using the if statement in every chain implementation why not put create abstract and and the condition there her is what I mean

```
public abstract class AbstractChain implements Chain{
private Chain nextInChain;
protected abstract void handle(Number number);
protected abstract boolean canHandle(Number number);
@Override
public void process(Number number) {
if(canHandle(number)){
handle(number);
}else {
nextInChain.process(number);
}

}
}
```

I prefer to use chain like the code above this way my code is easier to read each handle method just worries

about how to handle and not worry about conditions if its eligible to handle can this also be called chain of responsibility?

Reply

Milind *25/01/2013 at 1:13 pm*

excellent, No words to say

Reply

---

kuntal chakrabarti *06/04/2013 at 1:21 am*

Excellent article… I have one question.. Is Interceptors in struts2 uses chain of responsibility design pattern?

Reply

---

www.smithlawtlh.com *25/06/2013 at 9:33 am*

very good, Are you contemplating taking up sport fishing.

Reply

Atul Singh Chauan *06/08/2013 at 12:03 pm*

excellent and very easy to understand the concept, thanks a lot.

Reply

Revathy *14/08/2013 at 1:03 am*

Really Superb. You made us to understand the concept of design pattern, which everyone feels difficult… Really great..

Reply

Aswin *09/10/2013 at 1:35 pm*

awesome joe keep going

Reply

Anonymous *26/11/2013 at 11:28 am*

Very crisp and clear to understand the pattern

Reply

---

shrawan *28/12/2013 at 9:19 pm*

very easy to understand. nice keep it up

Reply

Vinuraj M V *08/01/2014 at 3:55 pm*

Awesome..!!! Thanks Joe..

Reply

L.GANESH *13/03/2014 at 12:02 pm*

Nice post and quick observable. Example is best suitable for demonstration of this pattern

Reply

Gajanan *09/04/2014 at 6:46 pm*

A very good website and Examples are really good

Thanks

Reply

Joe *13/04/2014 at 11:53 am*

Thanks Gajanan.

Reply

Anonymous *18/04/2014 at 1:03 pm*

Excellent

Reply

Your Comment

Name

Email

Post Comment

TUTORIAL MENU

Java

Site Map