Chain Of Responsibility

Behavioral Design Pattern :

As per Gang of Four design patterns, the **Chain of Responsibility** pattern is defined as:

"Gives more than one object an opportunity to handle a request by linking receiving objects together."

The **Chain of Responsibility** pattern is a behavioral pattern which is used to avoid coupling the sender of the request to the receiver and gives more than one receiver object the opportunity to handle the request.

We need to keep few important principles in mind while implementing Chain of Responsibility:

1. Each processor in the chain will have its implementation for processing a command
2. Every processor in the chain should have reference to the next processor
3. Each processor is responsible for delegating to the next processor so beware of dropped commands
4. Processors should not form a recursive cycle

   In our example, we don't have a cycle in our chain: *A -> B.* But, if we explicitly set *A* as next processor of *B,* then we end up with a cycle in our chain: *A -> B -> A.* Taking the next processor in the constructor can help with this
5. Only one processor in the chain handles a given command

The **Chain of Responsibility** pattern allows a number of classes to attempt to handle a request independently.

The **Receiver** objects of the requests are free from the order and can be use in any order for processing.

This pattern decouples sender and receiver objects based on type of request.

This pattern defines a chain of receiver objects having the responsibility, depending on run-time conditions, to either handle a request or forward it to the next receiver on the chain.

This pattern helps us avoiding coupling of sender and receiver objects of a requests and allows us to have more than one receiver as well for the request.

- ATM withdrawal using different currency notes is one of the great example of **Chain of Responsibility** pattern.
- Using different types of Loggers in our software is another example of the pattern.
- A call to technical support at different level of discussion is also a good example.

Chain of Responsibility Pattern Example in JDK

Let's see the example of chain of responsibility pattern in JDK and then we will proceed to implement a real life example of this pattern. We know that we can have multiple catch blocks in a **try-catch block** code. Here every catch block is kind of a processor to process that particular exception.
So when any exception occurs in the try block, its send to the first catch block to process. If the catch block is not able to process it, it forwards the request to next object in chain i.e next catch block. If even the last catch block is not able to process it, the exception is thrown outside of the chain to the calling program.

Chain of Responsibility Pattern Examples in JDK
java.util.logging.Logger#log()
javax.servlet.Filter#doFilter()

Points :

The **Chain of Responsibility**, **Command**, **Mediator** and **Observer** offers various ways of connecting senders and receivers of requests.

The **Chain of Responsibility** pattern is used to sequentially pass a request along a dynamic chain of receivers until at least one of them handles it.

The **Command** pattern is used to establish unidirectional connection between sender and receiver objects.

The **Mediator** pattern is used to eliminate direct coupling between sender and receiver objects and force them to communicate indirectly via a mediator object.

The **Observer** pattern allows receivers to dynamically subscribe and unsubscribe for the requests.

PartOne Example :

Validate Request : Authentication through JWT Token , UserNamePassword

PartTwo Example :

ATM Withdrawal money.