**Sequence of handlers processing an event one after other**

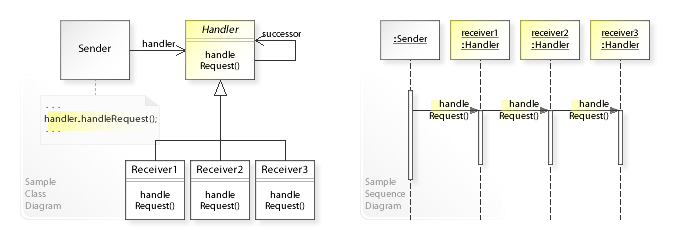
**A chain of components who all get a chance to process a command or query, optionally having default processing implementation and an ability to terminate the processing chain.**

**Chain of Responsibility** is a behavioral design pattern that lets you pass requests along a chain of handlers. Upon receiving a request, each handler decides either to process the request or to pass it to the next handler in the chain.

**GOF Definition**

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

Class Diagram



## ****Participants in the solution****

**1) 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.

**2) Concrete handlers**: These are actual handlers of the request chained in some sequential order.

**3) Client**: Originator of request and this will access the handler to handle it.

## ****Disadvantages****

And now that we've seen how interesting Chain of Responsibility is, let's keep in mind some drawbacks:

* Mostly, it can get broken easily:
  + if a processor fails to call the next processor, the command gets dropped
  + if a processor calls the wrong processor, it can lead to a cycle
* It can create deep stack traces, which can affect performance
* It can lead to duplicate code across processors, increasing maintenance

### Chain of Responsibility Pattern Examples in JDK

* java.util.logging.Logger#log()
* javax.servlet.Filter#doFilter()
* try catch block

**Command Query Separation**

Command – asking for an action or change (example- Please set your attack value to 2)

Query – asking for information (example -please give me your attack value)

CQS – having separate means of sending commands and queries to

## Relations with Other Patterns

* [**Chain of Responsibility**](https://refactoring.guru/design-patterns/chain-of-responsibility)**,**[**Command**](https://refactoring.guru/design-patterns/command)**,**[**Mediator**](https://refactoring.guru/design-patterns/mediator) and[**Observer**](https://refactoring.guru/design-patterns/observer) address various ways of connecting senders and receivers of requests:
  + Chain of Responsibility passes a request sequentially along a dynamic chain of potential receivers until one of them handles it.
  + Command establishes unidirectional connections between senders and receivers.
  + Mediator eliminates direct connections between senders and receivers, forcing them to communicate indirectly via a mediator object.
  + Observer lets receivers dynamically subscribe to and unsubscribe from receiving requests.