Chain Of Responsibility Pattern

In chain of responsibility, sender sends a request to a chain of objects. The request can be handled by any object in the chain.

A Chain of Responsibility Pattern says that just **"avoid coupling the sender of a request to its receiver by giving multiple objects a chance to handle the request".** For example, an ATM uses the Chain of Responsibility design pattern in money giving process.

In other words, we can say that normally each receiver contains reference of another receiver. If one object cannot handle the request then it passes the same to the next receiver and so on.

Advantage of Chain of Responsibility Pattern

* It reduces the coupling.
* It adds flexibility while assigning the responsibilities to objects.
* It allows a set of classes to act as one; events produced in one class can be sent to other handler classes with the help of composition.

Usage of Chain of Responsibility Pattern:

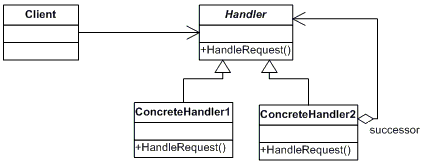
It is used:

* When more than one object can handle a request and the handler is unknown.
* When the group of objects that can handle the request must be specified in dynamic way.

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.

[](http://contribute.geeksforgeeks.org/wp-content/uploads/chain.gif)

* **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.