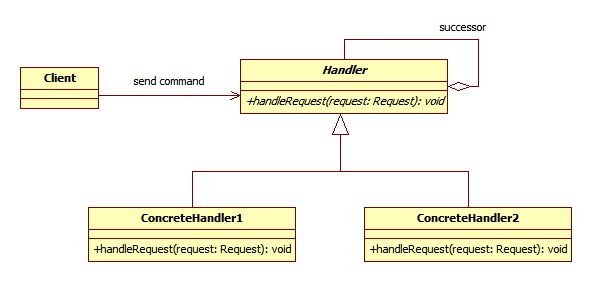
**Chain of Responsibility pattern**

1. **Description**

Chain of responsibility pattern is combined by a group of processing objects and command objects. Each processing object will handle one type of command objects. Other types will be passed to the next processing object.

1. **Implementation**

****

* **Handler** - defines an abstract class for handling requests.
* **ConcreteHandler** - handles the requests when in turn.
* **Client** - sends commands to the first object in the chain that may handle the command.

1. **Advantages and disadvantages**

**Advantages:**

* Reduce coupling of the request handler by dividing the process to many objects.
  + The receiver and the sender do not have knowledge of each other
  + The object in the chain does not know about the structure of the chain
* Add flexibility in assigning responsibilities to objects.
  + Add or change responsibilities for handling a request by adding or changing the chain at run-time

**Disadvantage:**

* Request is not guaranteed
  + A request can have no explicit receiver, so there is no guarantee it will be handled
  + A request can also be unhandled when the chain is not configured properly

**Usage:**

* Has many objects to handle a request
* Issue a request to one of several objects without specifying the receiver explicitly.
* The set of objects that can handle a request should be specified dynamically.

1. **Examples:** Shipping through electronic orders

The methods to process an order online differ from one to another depending on each customer. The number of products, time, money transfer, way of shipment, special cases causes the system to be able to handle all requests. Chain of Responsibility pattern can accept requests with different information and pass them to a chain of order handlers until the suitable handler with the request is found. When special case occurs, new handler is simply added to the system.

1. **Detailed implementation**

One of the worst parts of Chain of Responsibility pattern is that it does not guarantee all requests are handled even if they go through all handlers. This makes the system performed poorly. One way to avoid this is to check whether a request is executed before. If not, then we ignore it and implement handler for that request later.

Furthermore, in order to make Chain of Responsibility more flexible, we should make Request class, as shown in diagram above, an interface. Using this way, when new request appears, we only need to extend the Request class to handle it. Another solution is to use xml file or database to store all possible requests containing data and pass them to handler when the system operates.

Often, Chain of Responsibility is used in conjunction with Composite pattern. This is applied when the handlers or objects becomes complex and developers would like a flexible as well as maintainable system. Each object, representing a group, can contain many other objects, leaf. This creates a whole hierarchy so that the requests can pass through all this to find correct handler.

1. **Alternative**
2. **Reference**

OODesign 2012, ‘Composite Pattern’, OODesign.com, viewed 17 Nov 2012, <http://www.oodesign.com/composite-pattern.html>

OODesign 2012, ‘Chain of Responsibility Pattern’, OODesign.com, viewed 17 Nov 2012, <http://www.oodesign.com/chain-of-responsibility-pattern.html>

CodeProject 2009, ‘Chain of Responsibility Pattern’, codeproject.com, viewed 17 Nov 2012, <http://www.codeproject.com/Articles/41786/Chain-of-Responsibility-Design-Pattern>