Chain of Responsibility

Intent

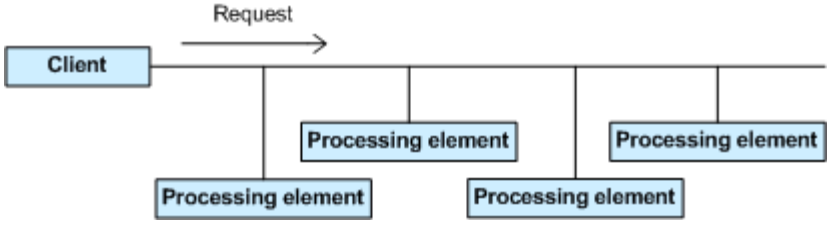
01

- Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.
- Launch-and-leave requests with a single processing pipeline that contains many possible handlers.
- An object-oriented linked list with recursive traversal.

Problem

02

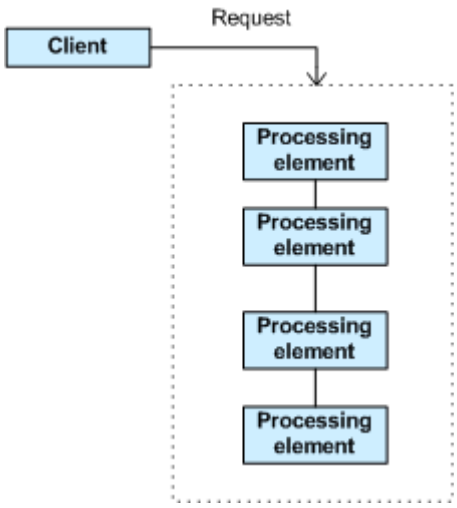
There is a potentially variable number of “handler” or “processing element” or “node” objects, and a stream of requests that must be handled. Need to efficiently process the requests without hard-wiring handler relationships and precedence, or request-to-handler mappings.



Discussion

03

Encapsulate the processing elements inside a “pipeline” abstraction; and have clients “launch and leave” their requests at the entrance to the pipeline.



04

The pattern chains the receiving objects together, and then passes any request messages from object to object until it reaches an object capable of handling the message. The number and type of handler objects isn’t known a priori, they can be configured dynamically. The chaining mechanism uses recursive composition to allow an unlimited number of handlers to be linked.

05

Chain of Responsibility simplifies object interconnections. Instead of senders and receivers maintaining references to all candidate receivers, each sender keeps a single reference to the head of the chain, and each receiver keeps a single reference to its immediate successor in the chain.

06

Make sure there exists a “safety net” to “catch” any requests which go unhandled.

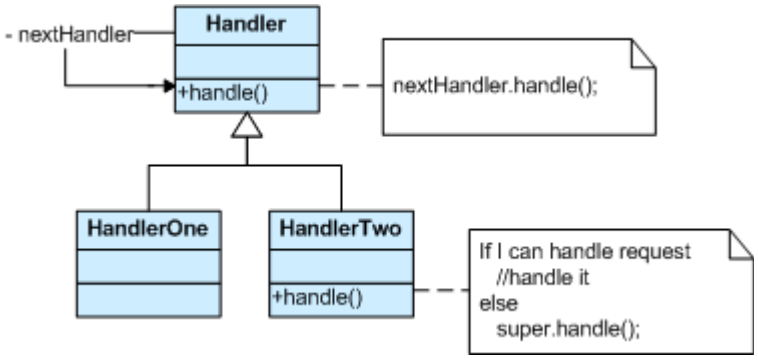
07

Do not use Chain of Responsibility when each request is only handled by one handler, or, when the client object knows which service object should handle the request.

Structure

08

The derived classes know how to satisfy Client requests. If the “current” object is not available or sufficient, then it delegates to the base class, which delegates to the “next” object, and the circle of life continues.



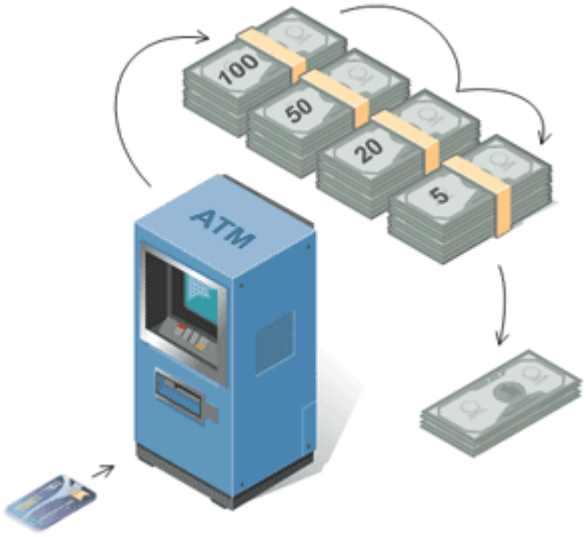
09

Multiple handlers could contribute to the handling of each request. The request can be passed down the entire length of the chain, with the last link being careful not to delegate to a “null next”.

Example

10

The Chain of Responsibility pattern avoids coupling the sender of a request to the receiver by giving more than one object a chance to handle the request. ATM use the Chain of Responsibility in money giving mechanism.



Check list

11

1. The base class maintains a “next” pointer.
2. Each derived class implements its contribution for handling the request.
3. If the request needs to be “passed on”, then the derived

class “calls back” to the base class, which delegates to the “next” pointer.

4. The client (or some third party) creates and links the chain (which may include a link from the last node to the root node).
5. The client “launches and leaves” each request with the root of the chain.
6. Recursive delegation produces the illusion of magic.

Rules of thumb

- Chain of Responsibility, Command, Mediator, and Observer, address how you can decouple senders and receivers, but with different trade-offs. Chain of Responsibility passes a sender request along a chain of potential receivers.
- Chain of Responsibility can use Command to represent requests as objects.
- Chain of Responsibility is often applied in conjunction with Composite. There, a component’s parent can act as its successor.