Tecniche di Progettazione: Design Patterns

GoF: Chain Of Responsibility

Chain Of Responsibility

▸ Intent

  ▸ 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
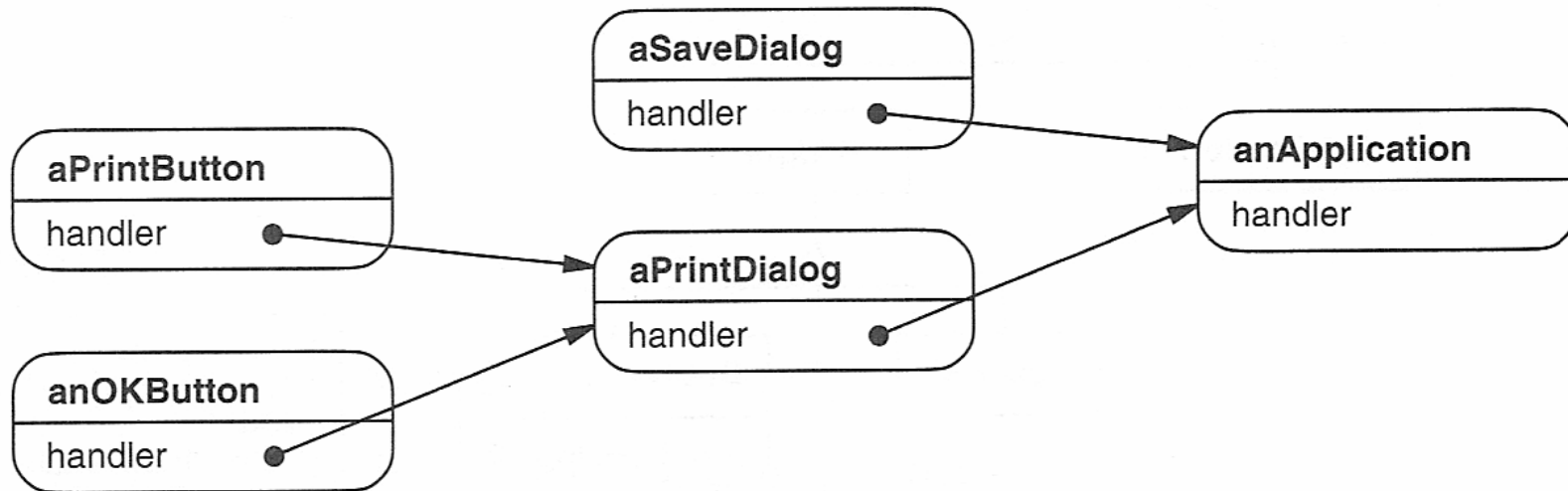
▸ Example

  ▸ Help information on an interface

  ▸ Organize from most specific to general (da chi sa fare meno cose a chi ne sa fare di più)

  ▸ Pattern decouples object that initiates request from the object the ultimately provides the help (non è il cliente che gira tra gli sportelli, ma la sua ruchiesta)
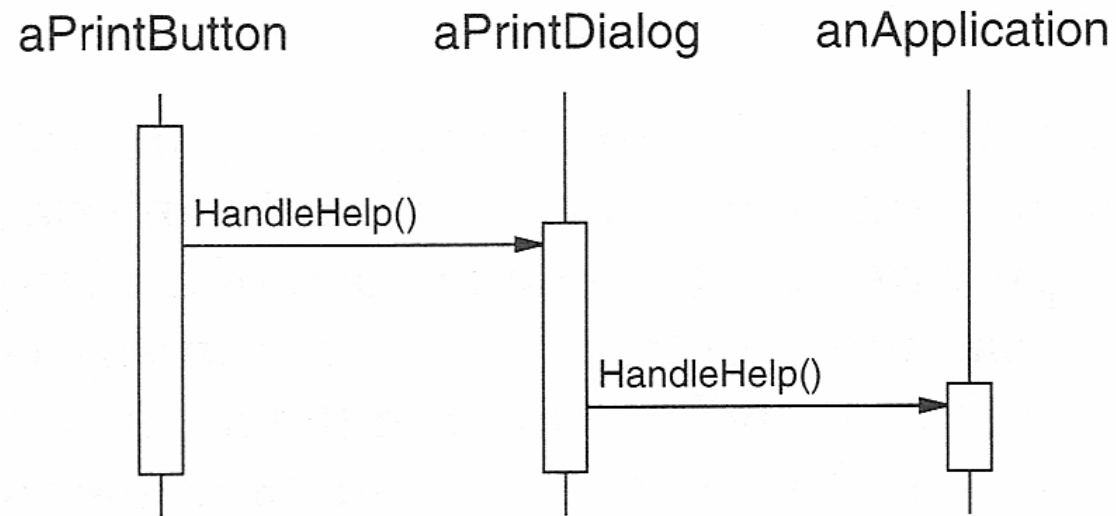
▸

Flow

▸ Request is passed a long chain of objects until one handles it

Flow

▸ First Object receives the request and either handles it or forwards it to the next candidate

▸ Example

 ▸ User clicks help on a PrintButton within a PrintDialog

Applicability
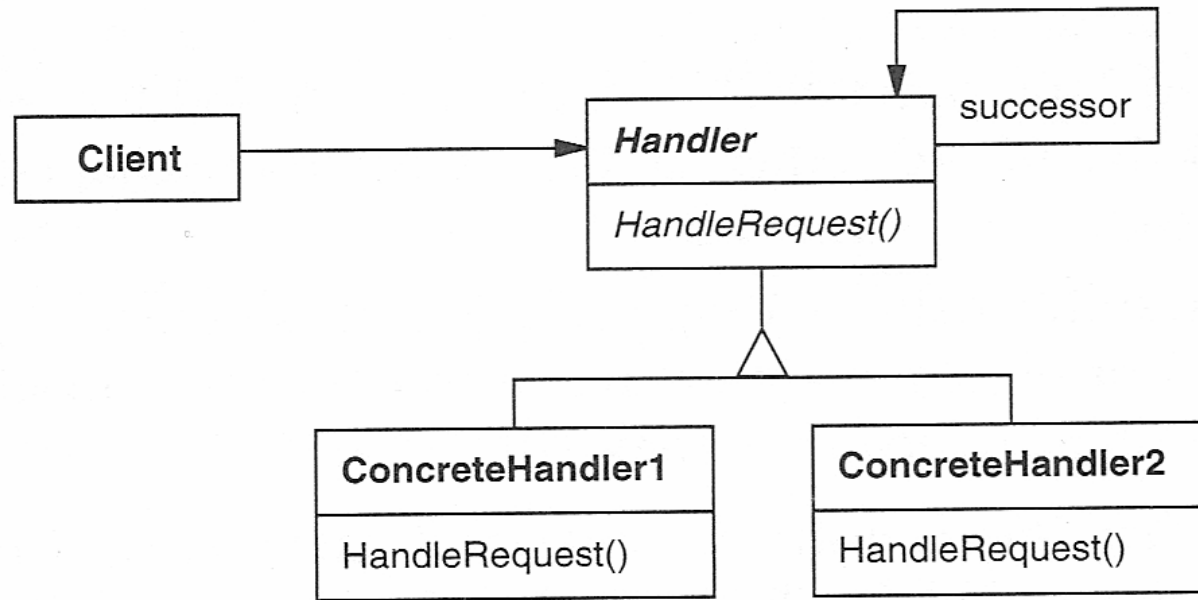
▸ Use Chain of Responsibility when

  ▸ More than one object may handle a request and the handler isn't known a priori.

  ▸ You want to issue a request to one of several objects without specifying the receiver explicitly

  ▸ The Set of objects than can handle a request should be specified dynamically

Structure
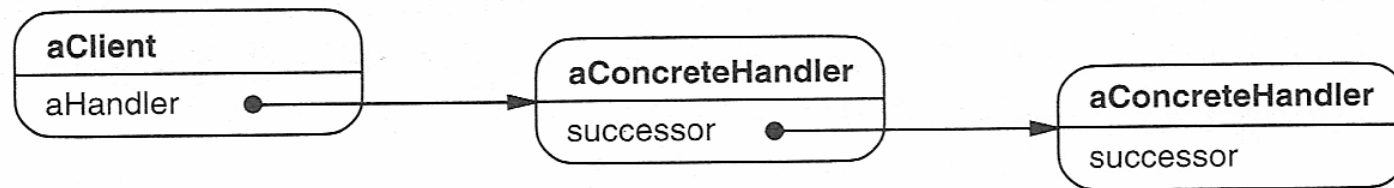
Participants

▸ **Handler**

 ▸ Defines an interface for handling request

 ▸ Optional implements the successor link

▸ **ConcreteHandler**

 ▸ Handles requests it is responsible for

 ▸ Can access its successor

 ▸ Forwards requests it does not handle

▸ **Client**

 ▸ Initiates the request to a (usually the first) ConcreteHandler object on the chain

Consequences

▸ **Reduced Coupling**

  ▸ Objects are free from knowing what object handles the request

▸ **Added Flexibility in assigning responsibilities to objects**

  ▸ Can change chain at runtime

  ▸ Can subclass for special handlers

▸ **Receipt is guaranteed**

  ▸ Request could fall off the chain

  ▸ Request could be dropped with bad chain

▸

Implementation

▶ Implementing the successor chain

   ▶ Define new links

      ▶ Can be handled at the base class level

   ▶ Use existing links

      ▶ In case like Composite, can use parent link

      ▶ Sometimes redundant links are needed, if relationship structure differs

Implementation

- **Representing Requests**
  - Hard coded operations
    - handle()
    - Limited in handling requests
  - Encoded request sent to the handler
    - handle(int i)
    - Requires conditionals
    - Requires packing/unpacking arguments
  - Send a Request Objects
    - handle(Request r)
    - Must be able to determine type in handler
    - Can subclass handlers

Related Patterns

▸ Composite

  ▸ Used with Chain of Responsibility so parent can act as a successor

▸ Decorator

  ▸ See next slide

Why would I ever use a Chain of Responsibility over a Decorator?

▸ **CoR: you can break the chain at any point**

  ▸ This is not true of **Decorator**.

  ▸ Decorators can be thought of as executing all at once without any interaction with the other decorators.

▸ Use the Chain of Responsibility pattern when you can conceptualize your program as a chain made up of links, where each link can either handle a request or pass it up the chain.

What's the difference between "Chain of responsibility" and "Strategy"?

▸ They're very different.

▸ **Strategy** is about having a generic interface which you can use to provide different implementations of an algorithm, or several algorithms or pieces of logic which have some common dependencies.

▸ For instance, a CollectionSorter could support a SortingStrategy (merge sort, quick sort, bubble sort). They all have the same interface and purpose, but can do different things.

▸ In some cases you may decide to determine strategy *inside*. Maybe the sorter has some heuristics based on collection size etc.

▸ Most of the time it indeed is *injected from outside*. This is when the pattern really shines: It provides users the ability to override (or provide) behavior. (dependency injection and Inversion of Control.)

What's the difference between "Chain of responsibility" and "Strategy" patterns?

▸ **Chain of responsibility** is about having a chain of objects which usually go from more detailed to more generic. Each of the pieces in chain can provide the answer, but they have different levels of detail.

▸ Popular GOF example is a context help system. When you click on a component in your desktop app, which help to display? First item in chain could look for help for the very component you clicked. Next in chain could try and display help for the whole containing dialog. Next for the application module... and so on.

Design patterns, Laura Semini, Università di Pisa, Dipartimento di Informatica.

Chain-of-responsibility vs. lists of handlers

▸ Problem: refactor a huge(1000 lines) method full of "if" statements.

▸ Solution 1: chain-of-responsibility pattern: base "Handler" class. Then, "Handler1", "Handler2", etc.

▸ Solution 2: base "Handler" class as well, with "Handler1", "Handler2", just like the previous method mentioned.
However, there would be no "getSuccessor" method. Instead, a Collection class with a list of handlers(a Vector, an ArrayList, …).
The handleRequest function would still exist, but it wouldn't propagate the call to the next handlers. It would just process the request or return null.
To handle a request, one would use

▸ for(Handler handle : handlers){
      result = handle.handleRequest(request);
      if(result!=null) return result;
}
throw new CouldNotParseRequestException();

Chain-of-responsibility vs. lists of handlers (cont'd)

▸ Solution 2 makes it easy and clear to manipulate this set of handlers: the collections interface is well known and everybody understands how to iterate over a List or what not.

▸ If the handlers can completely handle a request on their own, Solution 2 is fine. The handlers do not have a reference to other handlers, which makes the handler interface simple. You can add or remove handlers from the middle of the chain.

▸ One problem with Solution 2 is that a handler cannot do pre-processing or post-processing on the request. If this functionality is required, then Chain of Responsibility is better. In CoR, the handler is the one responsible for delegating to the next handler on the chain, so the handler can do pre-processing and/or post-processing, including modifying or replacing the response from the next handler on the chain. In this way, CoR is very similar to Decorator; it's just the intent that's different.

Chain of responsibility and Filters. Are they the same thing ?

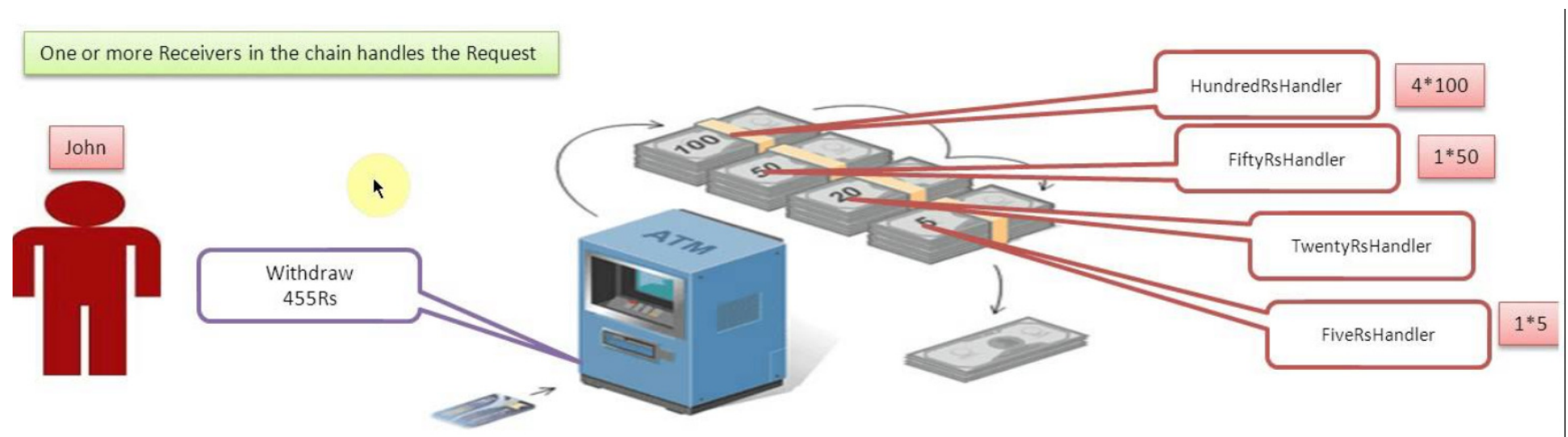▸ Filter pattern is similar to chain of responsibility pattern.

▸ But it is far enough to not mix them.

▸ In filter/interceptor pattern, we <u>have not </u>the notion of responsibility.

▸ So, a filter or an interceptor is more a <u>chain of processing </u>than a <u>chain of responsibility</u>.

▸ (continues)

Design patterns, Laura Semini, Università di Pisa, Dipartimento di Informatica.

Chain of responsibility and Filters. Are they the same thing ?

▸ Filters/interceptors in a chain may have (and have often) no logic or functional relation between them

▸ nodes of a chain of responsibility have always a logic or functional relation between them

   ▸ they have to handle the same concern.

▸ Example,

   ▸ in a chain filter, the first filter may handle logging concern, the second filter, security concern and the last, encoding concern...

▸ In a chain of responsibility, the same concern is handled by all nodes of the chain.

Design patterns, Laura Semini, Università di Pisa, Dipartimento di Informatica.

Ex. ATM use the Chain of Responsibility in money giving mechanism.

One or more Receivers in the chain handles the Request

John

Withdraw 455Rs

ATM

HundredRsHandler — 4*100

FiftyRsHandler — 1*50

TwentyRsHandler

FiveRsHandler — 1*5

Design patterns, Laura Semini, Università di Pisa, Dipartimento di Informatica.

Fun, but not CoR. Discuss

Design patterns, Laura Semini, Università di Pisa, Dipartimento di Informatica.

Homework (no pre- or post-processing)

▸ **When the client of a bank asks for a loan, (s)he asks to:**

  ▸ The desk empoyee, who is entitled of approving loans up to $10.000, otherwise (s)he asks to:

  ▸ The deputy director of the bank branch, who is entitled of approving loans up to $50.000, otherwise (s)he asks to:

  ▸ The director of the bank branch, who is entitled of approving loans up to $200.000, otherwise (s)he asks to:

  ▸ The director of the bank regional area, who is entitled of approving loans up to $1.000.000, otherwise (s)he asks to:

  ▸ …..

Dynamically change the chain (no more deputy director of the bank branch)

A non Homework

▶ Use CoR to write a program that, given a number n, is able to return:

1. the number of primes smaller than n
2. the decomposition in prime factors of n

▶ Hint: build a (or two differet) chain of prime numbers, and assume n is smaller than 50.

▶ Use Builder to build the chain, in a situation where there are two kinds of handlers, and hence two chains (the client decides which chain to build):

▶ one only dealing with prime factorization,
▶ the other only counting the number of primes smaller than n