References
Design Patterns: Elements of Resuable Object-Oriented Software, Gamma, Helm, Johnson, Vlissides, Addison Wesley, 1995, pp. 223-232


Pipe & Filter References


http://www.enterpriseintegrationpatterns.com/PipesAndFilters.html

Detailed Discussion
http://john.cs.olemiss.edu/~hcc/softArch/notes/pipes.html

Intercepting Filter References

Core J2EE Patterns, 2nd Ed. Alur, Crupi, Malks, Sun Microsystems Press, Prentice Hall, 2003, pp 144-165

http://java.sun.com/blueprints/corej2eepatterns/Patterns/InterceptingFilter.html
Chain of Responsibility - Composition on steroids

Dynamically create chain of handlers

Multiple handlers may be able to handle a request

Only one handler actually handles the request

Consequences

Reduced coupling

Added flexibility in assigning responsibilities to objects

Not guaranteed that request will be handled
Finding Methods

test = new Bar();
test.toString();
Context Help System

User clicks on component for help

Tree of handlers
From specific to general
Email Filters in Mail Client

User creates a set of rules
delete
move
modify

Chain the rules

First rule that applies handles the mail
Other Examples

Java 1.0 AWT action(Event)

http://wiki.cs.uiuc.edu/PatternStories/JavaAWT

javax.servlet.Filter


Microsoft Windows global keyboard events

class ChainOfResponsibilityExample {
  public static void main(String[] args) {
    // building the chain of responsibility
    Logger l = new DebugLogger(Logger.DEBUG).setNext(
      new EMailLogger(Logger.ERR).setNext(
        new StderrLogger(Logger.NOTICE) )
    );

    l.message("Entering function x.", Logger.DEBUG);  // handled by DebugLogger
    l.message("Step1 completed.", Logger.NOTICE);      // handled by Debug- and StderrLogger
    l.message("An error has occurred.", Logger.ERR);  // handled by all three Logger
  }
}

abstract class Logger {
    public static int ERR = 3;
    public static int NOTICE = 5;
    public static int DEBUG = 7;
    protected int mask;

    protected Logger next;
    public Logger setNext(Logger l) {
        next = l;
        return this;
    }

    abstract public void message(String msg, int priority);
}

class DebugLogger extends Logger {
    public DebugLogger(int mask) {
        this.mask = mask;
    }

    public void message(String msg, int priority) {
        if (priority <= mask) debug log here
        if (next != null) next.message(msg, priority);
    }
}

class EMailLogger extends Logger {
    public EMailLogger(int mask) { this.mask = mask; }

    public void message(String msg, int priority) {
        if (priority <= mask) send email here;
        if (next != null) next.message(msg, priority);
    }
}
abstract class Logger {
    public static int ERR = 3;
    public static int NOTICE = 5;
    public static int DEBUG = 7;
    protected int mask;

    protected Logger next;
    public Logger setNext(Logger l) {
        next = l;
        return this;
    }

    public void message(String msg, int priority) {
        if (priority <= mask) log(msg);
        if (next != null) next.message(msg, priority);
    }

    abstract void log(String message);
}

class StderrLogger extends Logger {
    public StderrLogger(int mask) { this.mask = mask; }

    void message(String msg, int priority) { send to err}
}

class EMailLogger extends Logger {
    public EMailLogger(int mask) { this.mask = mask; }

    void message(String msg, int priority) { email here}
}

class DebugLogger extends Logger {
    public DebugLogger(int mask) { this.mask = mask; }

    void message(String msg, int priority) { debug stuff}
}
Is this the Chain of Responsibility?
Object-Oriented Recursion

A method polymorphically sends its message to a different receiver

Eventually a method is called that performs the task

The recursion then unwinds back to the original message send
class HeadNode {
    public String toString() {
        return "(" + next.toString();
    }
}

class Node {
    public String toString() {
        return " " + element + next.toString();
    }
}

class TailNode {
    public String toString() {
        return " ");
    }
}
class HeadNode {
    public void add(int value) {
        next.add(value);
    }
}

class Node {
    public void add(int value) {
        if (element > value)
            prependNode(value);
        else
            next.add(value);
    }
}

class TailNode {
    public void add(int value) {
        prependNode(value);
    }
}
OO Recursion

Decorator

Chain of Responsibility
Architectural Patterns

Specify basic structure of an application
Pipes & Filters

`ls | grep -i b | wc -l`

**Context**

Processing data streams

**Problem**

Building a system that processes or transforms a stream of data

**Forces**

Small processing steps are easier to reuse than large components

Non-adjacent processing steps do not share information

System changes should be possible by exchanging or recombining processing steps, even by users

Final results should be presented or stored in different ways
Solution

Divide task into multiple sequential processing steps or filter components

Output of one filter is the input of the next filter

Filters process data incrementally

   Filter does not wait to get all the data before processing
Solution Continued

Data source – input to the system

Data sink – output of the system

Pipes - connect the data source, filters and data sink

Pipe implements the data flow between adjacent processes steps

Processing pipeline – sequence of filters and pipes

Pipeline can process batches of data
Python Interpreter

http://wiki.cs.uiuc.edu/cs427/Python+-+Batch+Sequential
Muffin - Client Side Http Filter

WebBrowser -> Server

Server

Handler

Filter

Filter

Filter

HttpRelay

Filter Manager

Web Server

Internet

http://muffin.doit.org/

http://mark.boyns.org/
Intercepting Filter - Problem

Preprocessing and post-processing of a client Web request and response

A Web request often must pass several tests prior to the main processing
  Has the client been authenticated?
  Does the client have a valid session?
  Is the client's IP address from a trusted network?
  Does the request path violate any constraints?
  What encoding does the client use to send the data?
  Do we support the browser type of the client?

Nested if statements lead to fragile code
Common processing, such as checking the data-encoding scheme or logging information about each request, completes per request.

Centralization of common logic is desired.

Services should be easy to add or remove unobtrusively without affecting existing components, so that they can be used in a variety of combinations, such as

Logging and authentication

Debugging and transformation of output for a specific client

Uncompressing and converting encoding scheme of input
Intercepting Filter - Solution

Create pluggable filters to process common services

http://java.sun.com/blueprints/corej2eepatterns/Patterns/InterceptingFilter.html