CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2013
Doc 17 Decorator, Chain of Responsibility
April 18, 2013

Decorator

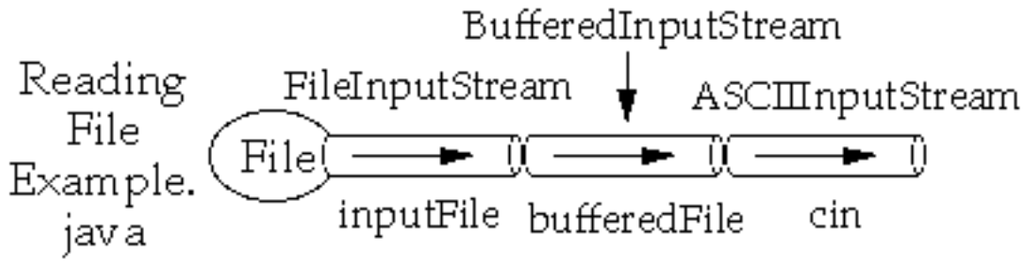
Prime Directive

Data + Operations

Decorator Pattern

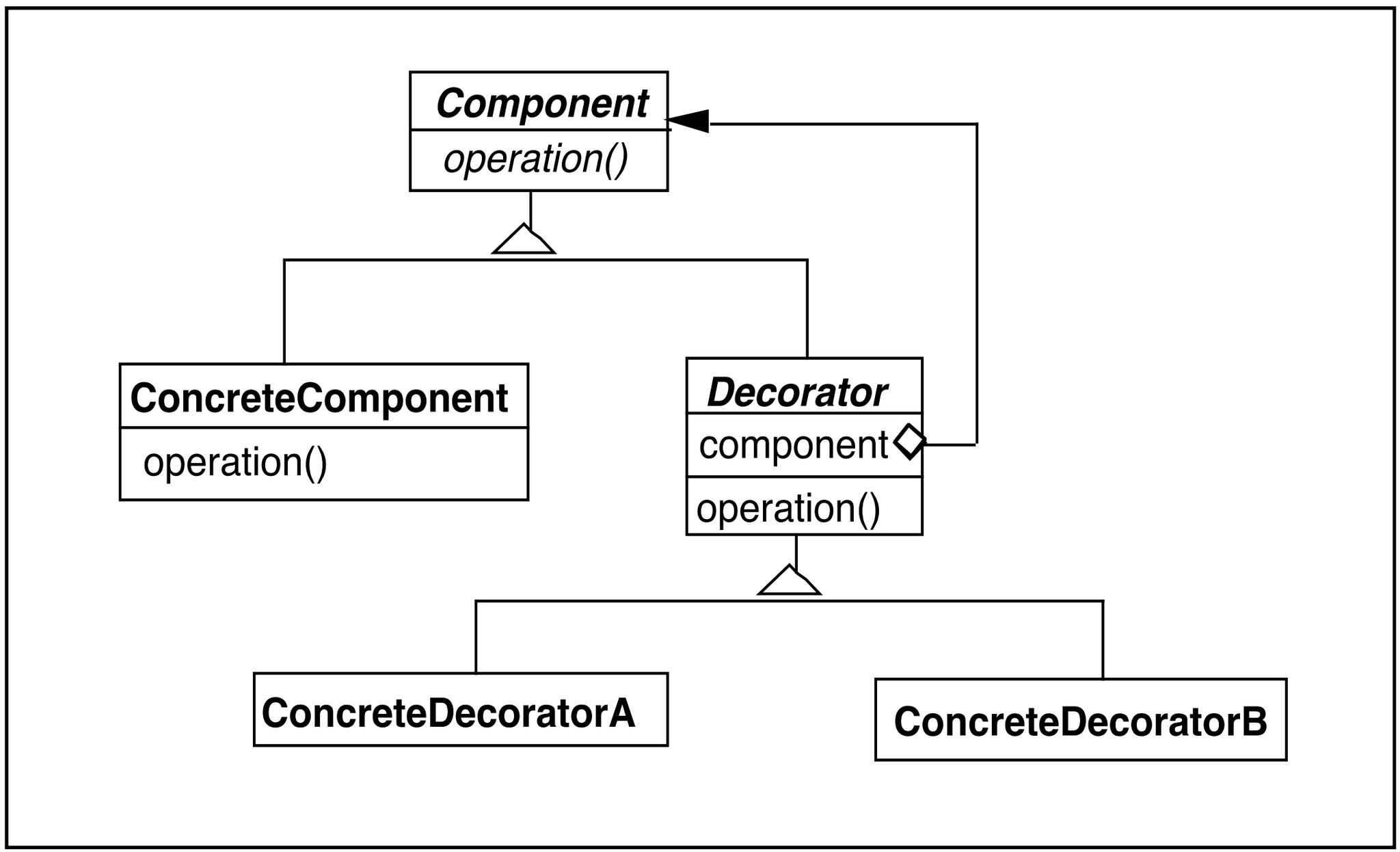
Adds responsibilities to individual objects

Dynamically
Transparently

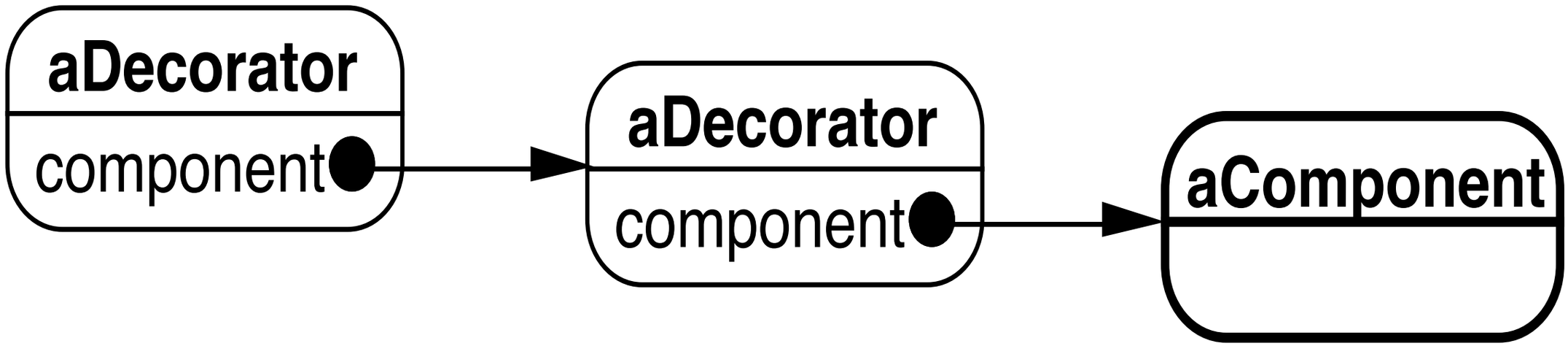



```
import java.io.*;
import sdsu.io.*;
class  ReadingFileExample
      {
      public  static  void  main( String  args[]  ) throws Exception
            {
            FileInputStream inputFile;
            BufferedInputStream bufferedFile;
            ASCIIInputStream  cin;

            inputFile = new FileInputStream( "ReadingFileExample.java" );
            bufferedFile = new BufferedInputStream( inputFile );
            cin = new ASCIIInputStream( bufferedFile );
```

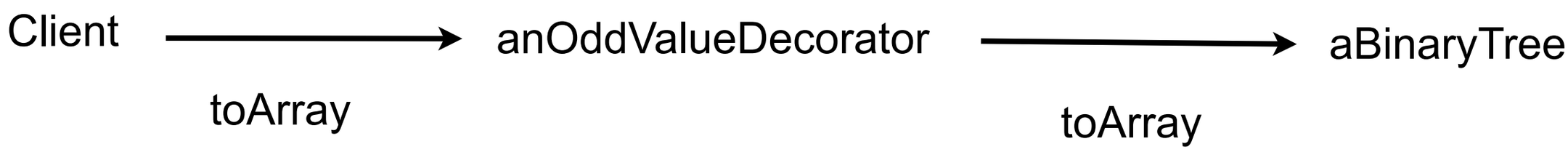

Component
operation()
ConcreteComponent
operation()
Decorator
component
operation()
ConcreteDecoratorA
ConcreteDecoratorB

Decorator forwards all component operations



Favor Composition over Inheritance

Refactoring: Move Embellishment to Decorator



Benefits & Liabilities

Benefits

Simplifies a class

Distinguishes a classes core responsibilities from embellishments

Liabilities

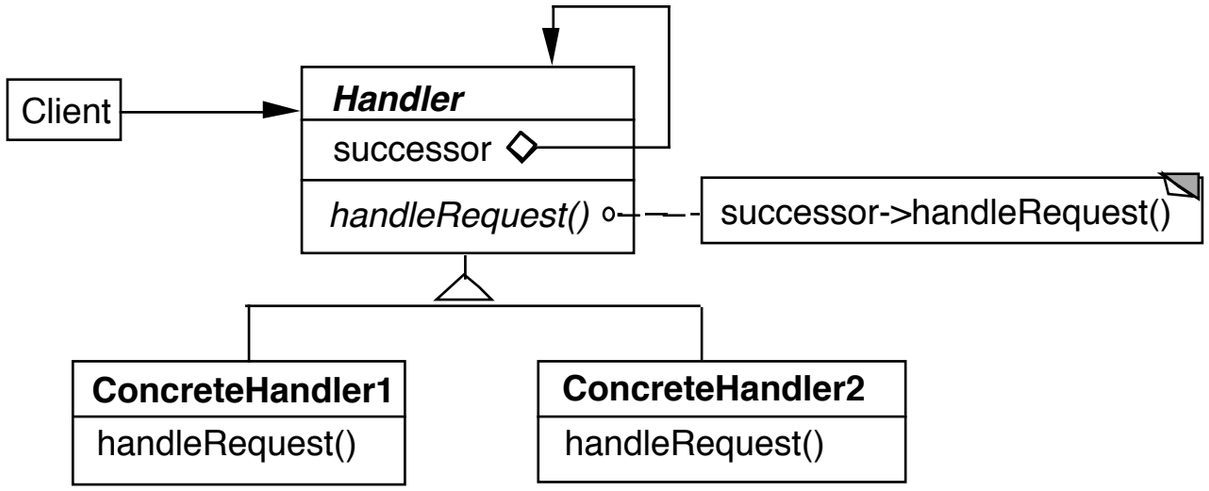
Changes the object identity of a decorated object

Code harder to understand and debug

Combinations of decorators may not work correctly together

Chain of Responsibility

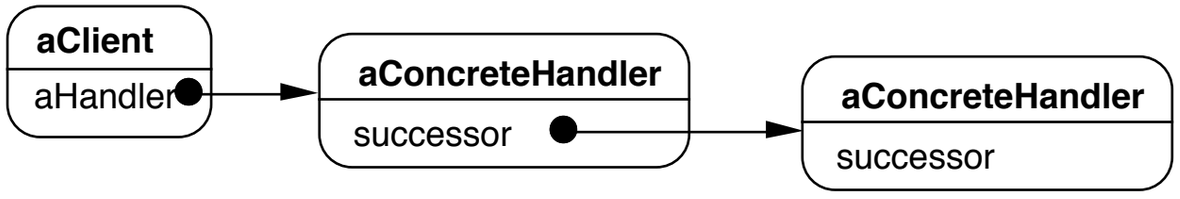
Chain of Responsibility



Dynamically create chain of handlers

Multiple handlers may be able to handle a request

Only one handler actually handles the request



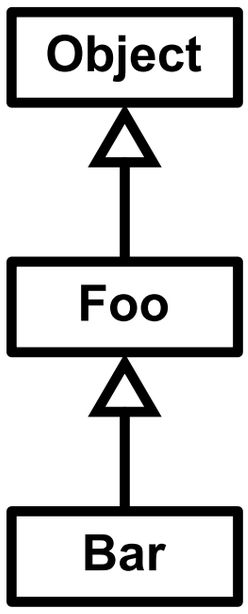
Consequences

Reduced coupling

Added flexibility in assigning responsibilities to objects

Not guaranteed that request will be handled

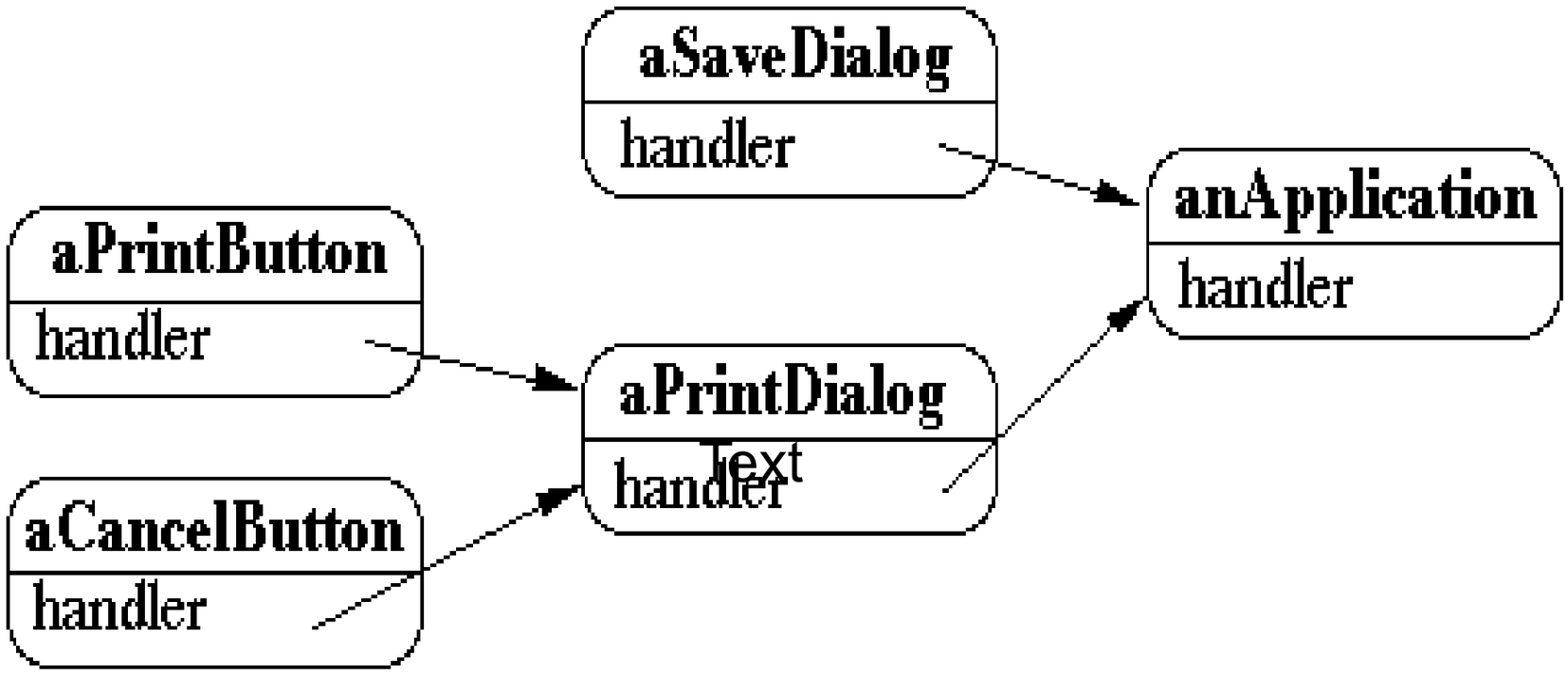
Finding Methods




```
test = new Bar();
test.toString();
```

Context Help System

User clicks on component for help



Tree of handlers

From specific to general

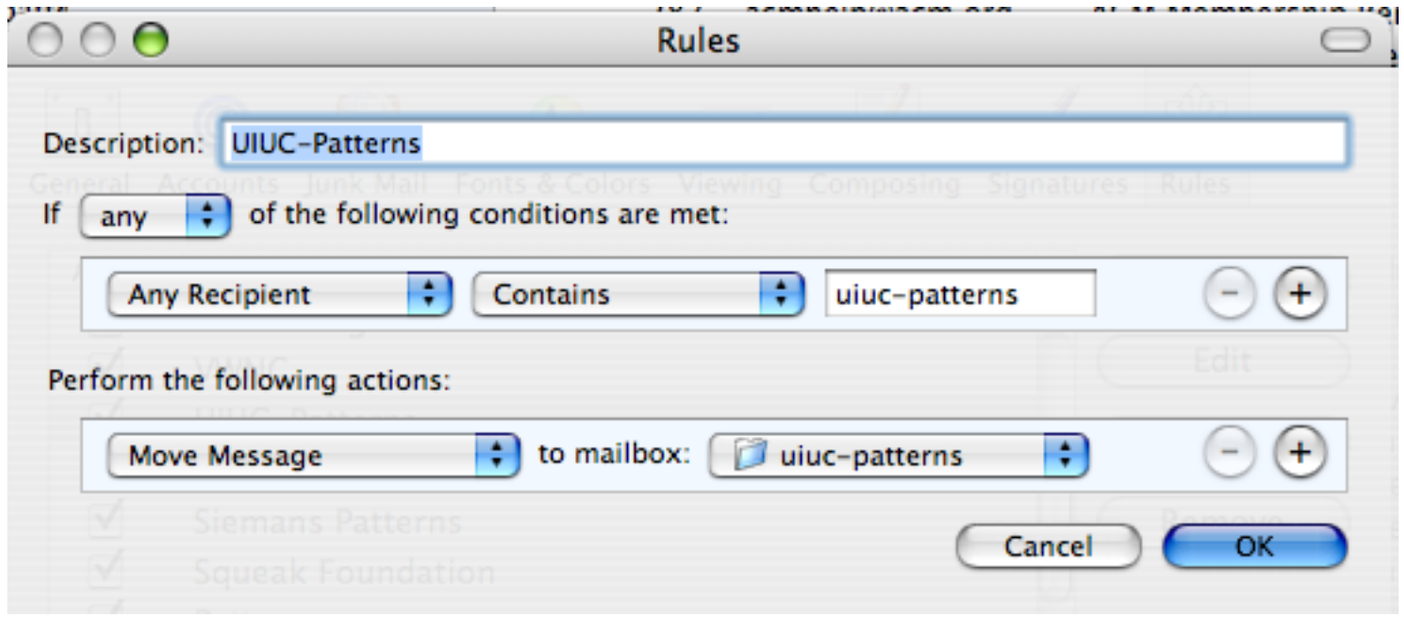
Email Filters in Mail Client

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

http://tomcat.apache.org/tomcat-4.1-doc/servletapi/javax/servlet/Filter.html

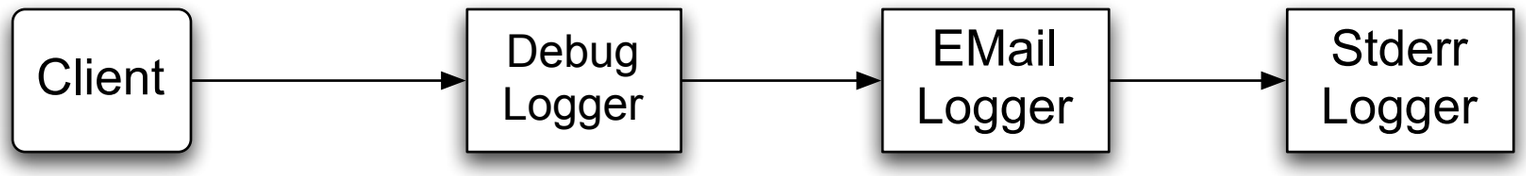
Microsoft Windows global keyboard events

http://www.javaworld.com/javaworld/jw-08-2004/jw-0816-chain.html

Apache Commons Chain

http://commons.apache.org/chain/

Logger Example




```
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
   }
}
```

First Attempt

```
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);
   }
```

Improved 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; }

   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?

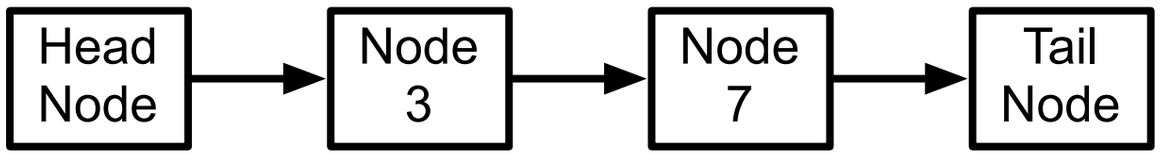
Object-Oriented Recursion

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

Linked List toString



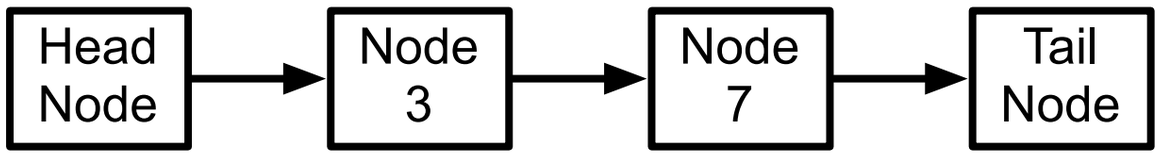
(3 7)

```
class HeadNode {
        public String toString() {
                return "(" + next.toString();
        }
}
```

```
class TailNode {
        public String toString() {
                return " )";
        }
}
```

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

Linked List add




```
class HeadNode {
        public void add(int value) {
                next.add(value);
        }
}
```

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

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

OO Recursion

Decorator

Chain of Responsibility