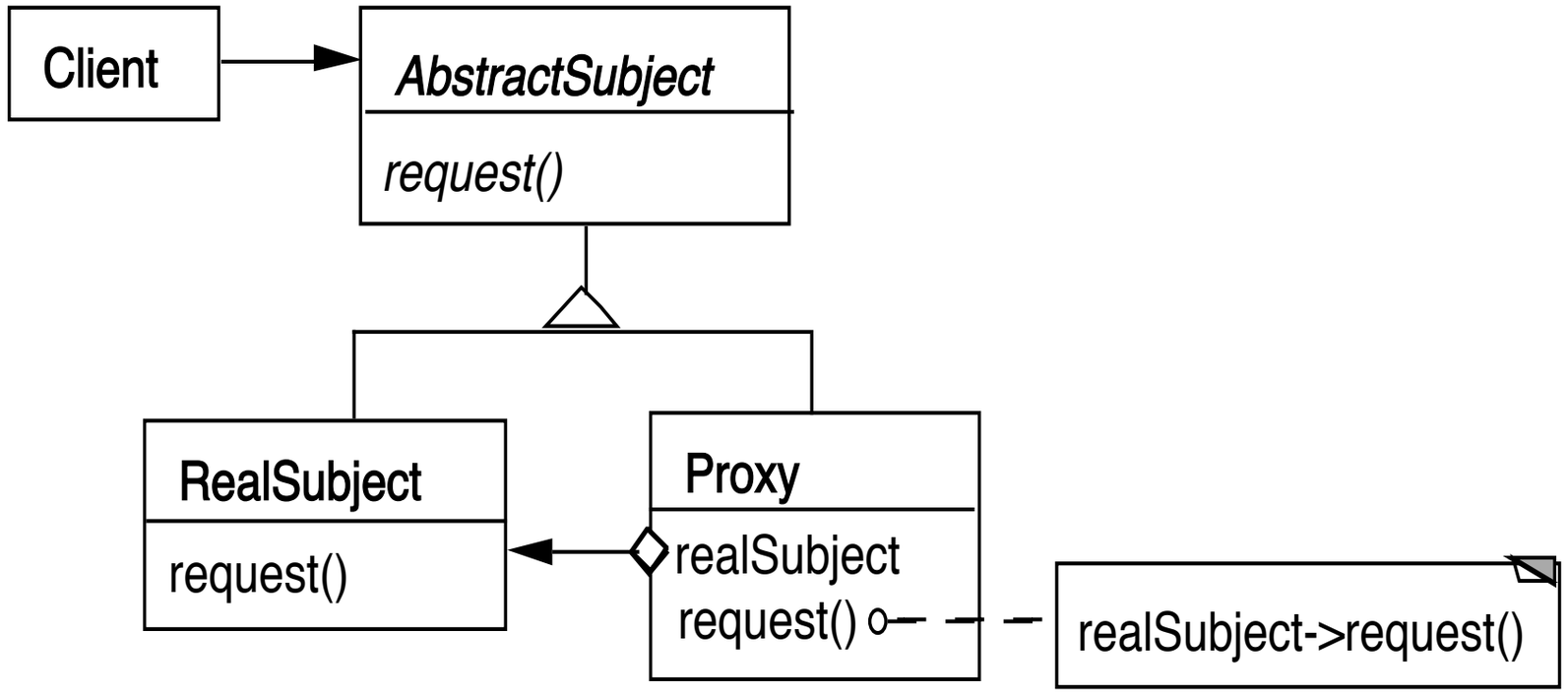
CS 635 Advanced Object-Oriented Design & Programming
Fall Semester, 2020

Doc 13 Proxy, Bridge, Chain, Pipes & Filters
Oct 8, 2020

Proxy

Proxy (Surrogate)

a person authorized to act on behalf of another

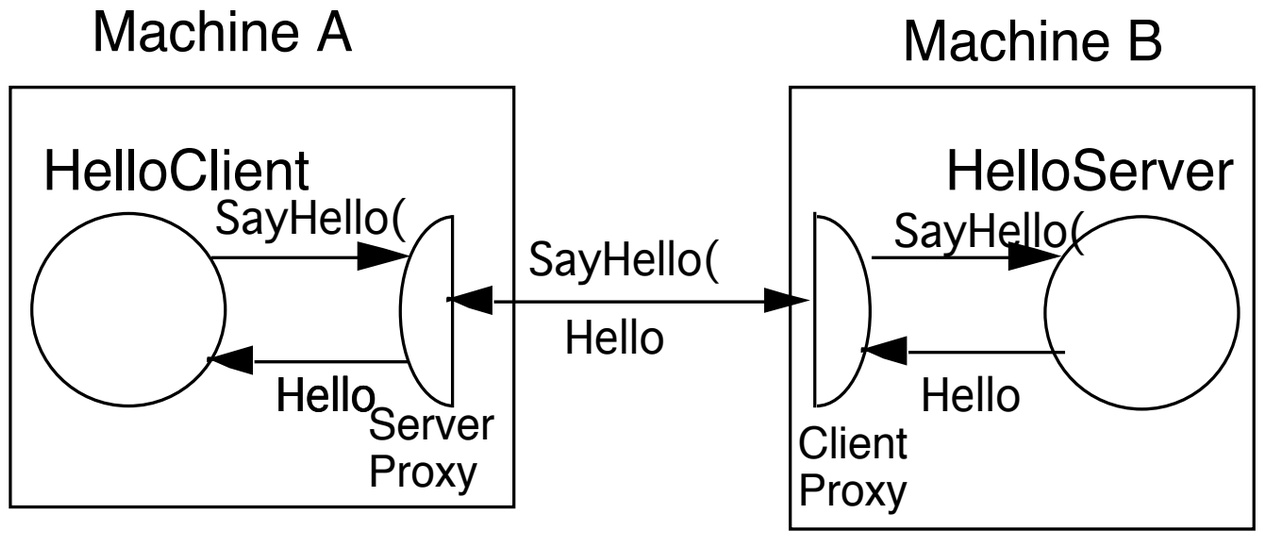



```
class Proxy {
   AbstractSubject realSubject;

   public Foo service(Bar x ) {
      return realSubject(x);
   }
```

Why do it?

Remote Proxy




```
String server = getHelloHostAddress( args);
Hello proxy = (Hello) Naming.lookup( server );
String message = proxy.sayHello();
System.out.println( message );
```

More General Proxy

```
class Proxy {
    AbstractSubject realSubject;

    public Foo service(Bar x ) {
        some preprocessing
        result = realSubject(x);
        some postprocessing
    }
```

Virtual Proxy

Creates/accesses expensive objects on demand

O-R Mapping Layers

Java's Synchronized List

```
ArrayList notSafe = new ArrayList();
List threadSafe = Collections.synchronizedList(notSafe);
```

```
static class SynchronizedList {
    List list;
    public Object get(int index) {
synchronized(mutex) {return list.get(index);}
}
```

Java's Unmodifiable List

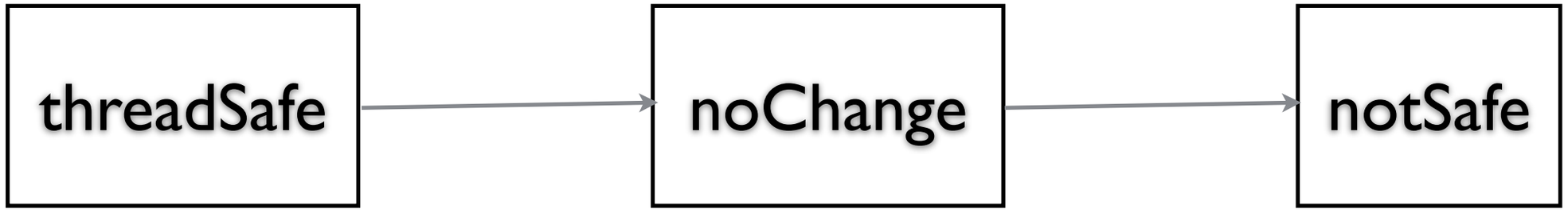
```
ArrayList notSafe = new ArrayList();
List noChange = Collections.unmodifiableList(notSafe);
```

```
static class UnmodifiableList {
    List list;
public Object get(int index) { return list.get(index);}

public Object set(int index, Object element) {
    throw new UnsupportedOperationException();
    }
```

Proxy or Decorator?

```
ArrayList notSafe = new ArrayList();
List noChange = Collections.unmodifiableList(notSafe);
List threadSafe = Collections.synchronizedList(noChange);
```



Proxy verses Decorator

"Decorators can have similar implementations as proxies"

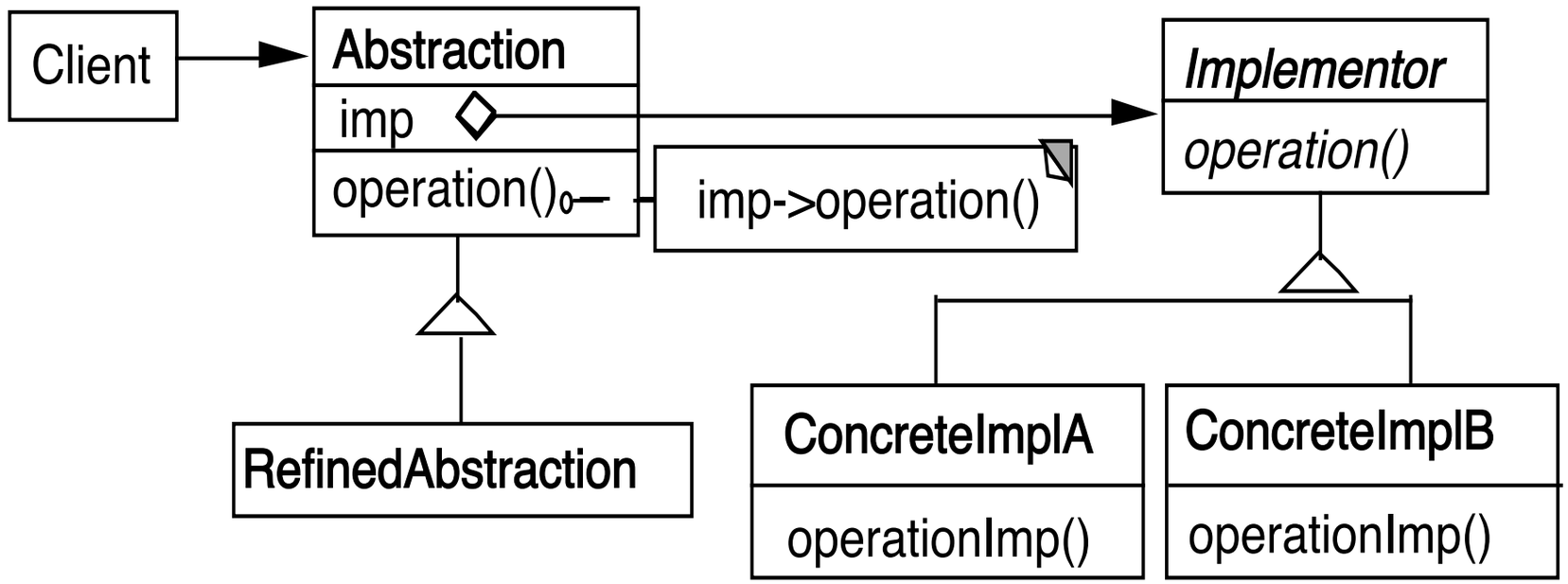
Proxy controls access to an object

Decorator adds one or more responsibilities to an object

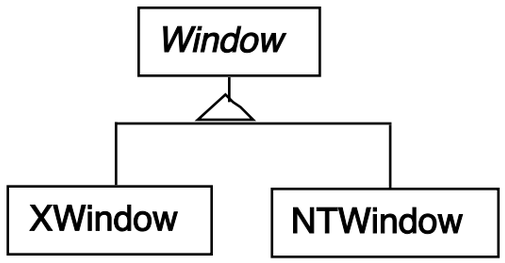
Bridge

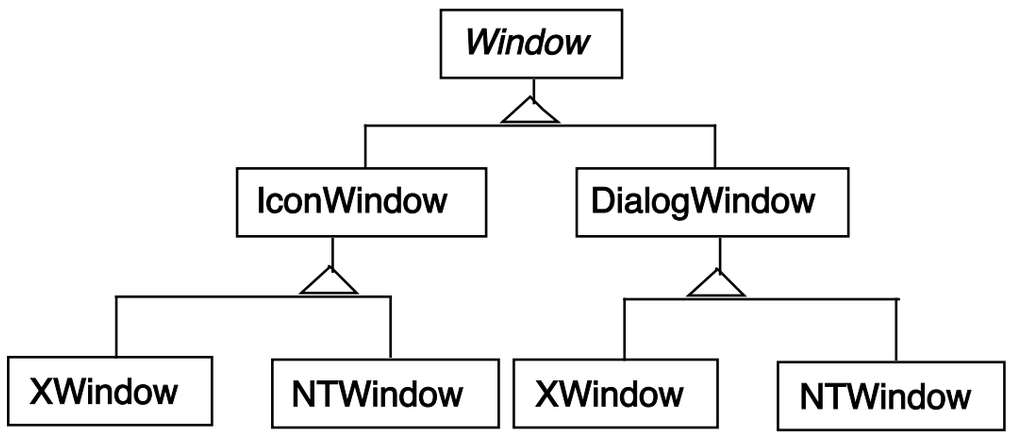
Bridge

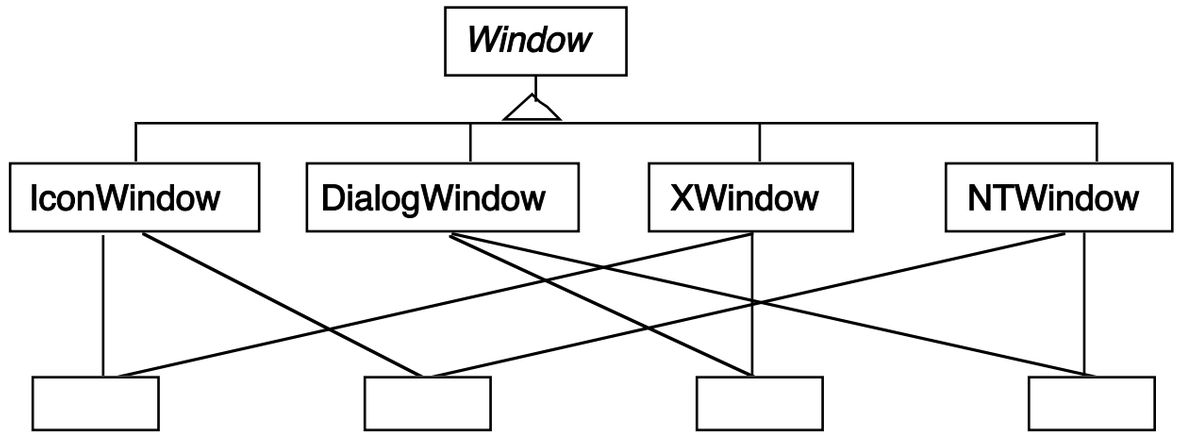
Decouple an abstraction from its implementation



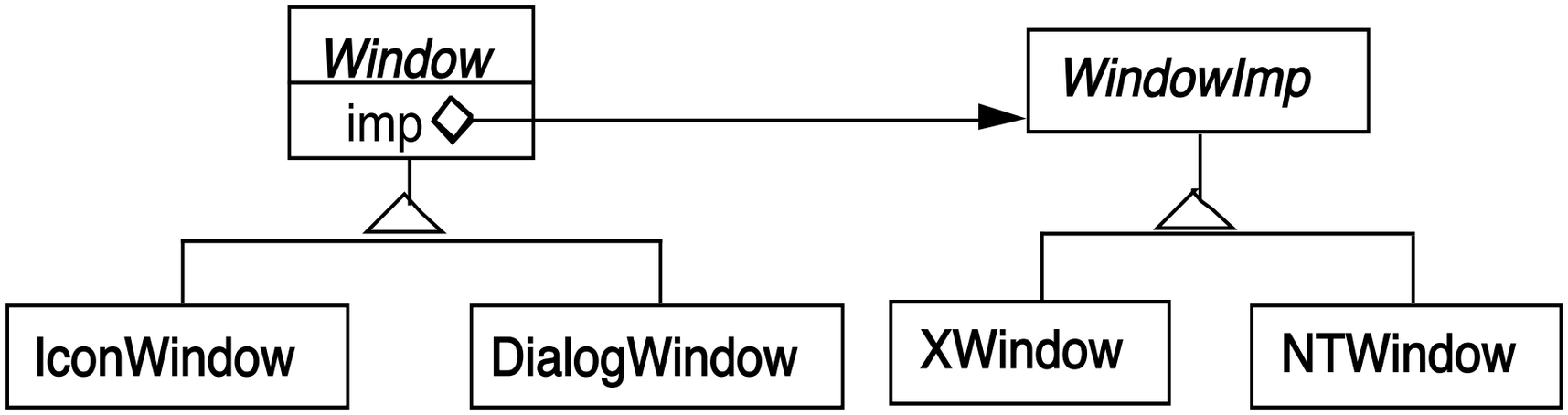
Windows


Window
XWindow
NTWindow


Window
IconWindow
DialogWindow
XWindow
NTWindow
XWindow
NTWindow


Window
IconWindow
DialogWindow
XWindow
NTWindow

Using the Bridge Pattern

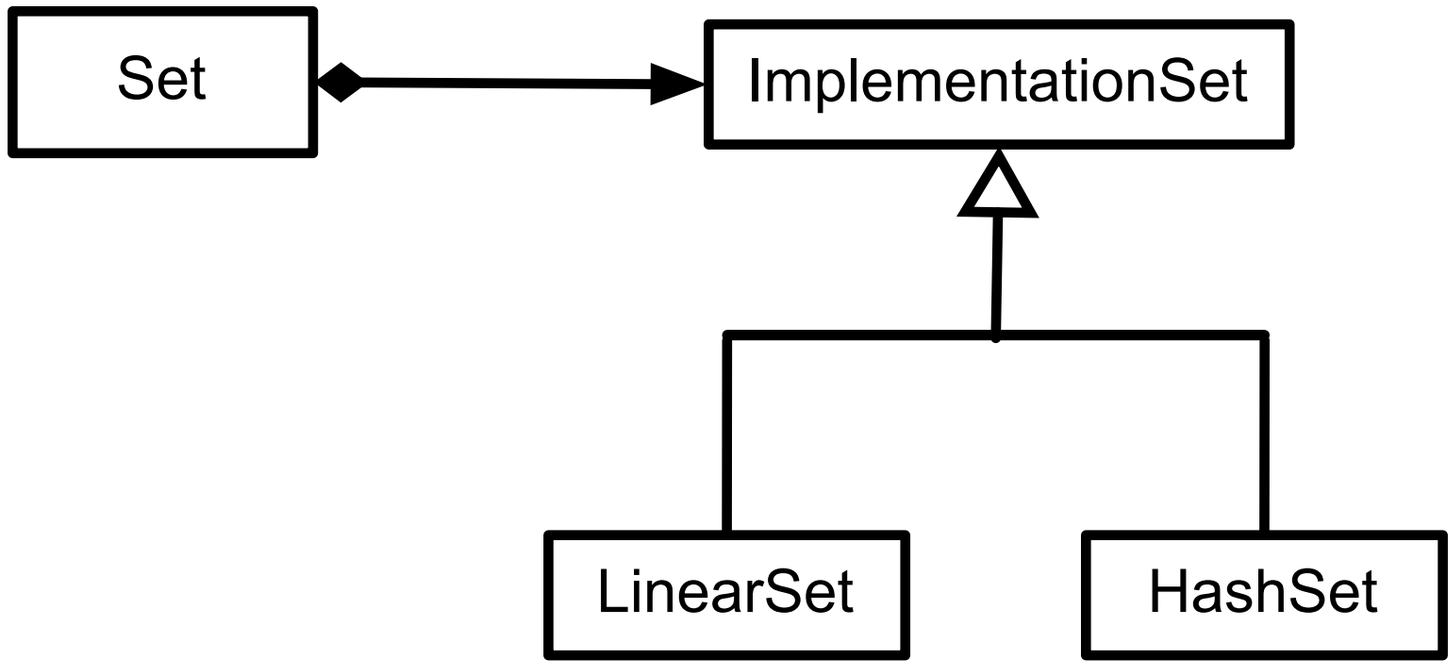


Peers in Java's AWT

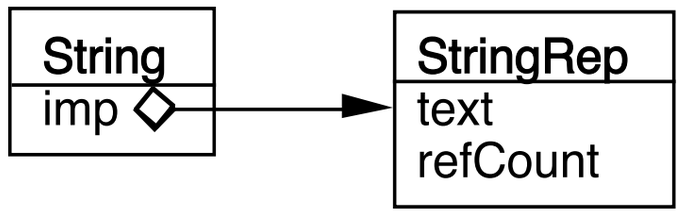
Peer = implementation

```
    public synchronized void setCursor(Cursor cursor) {
        this.cursor = cursor;
        ComponentPeer peer = this.peer;
        if (peer != null) {
            peer.setCursor(cursor);
        }
    }
```

IBM Smalltalk Collections



Smart Pointers in C++



Coplien's Implementation

```
class StringRep  {
    friend String;

    private:
        char *text;
        int refCount;

        StringRep(){ *(text = new char[1] = '\0'; }

        StringRep( const StringRep& s )    {
            ::strcpy( text = new char[::strlen(s.text) + 1, s.text);
        }

        StringRep( const char *s)      {
            ::strcpy( text = new char[::strlen(s) + 1, s);
        }

        StringRep( char** const *r)    {
            text = *r;
            *r = 0;
            refCount = 1;;
        }
        ~StringRep()       { delete[] text; }
        int length() const { return ::strlen( text ); }
        void print() const { ::printf("%s\n", text ); }
}
```

```
class String{
    friend StringRep
    public:
        String operator+(const String& add) const { return *imp + add; }
        StringRep* operator->() const         { return imp; }
        String()       { (imp = new StringRep()) -> refCount = 1;       }
        String(const char* charStr)    { (imp = new StringRep(charStr)) -> refCount = 1; }
        String operater=( const String& q)  {
            (imp->refCount)--;
            if (imp->refCount <= 0 &&
                 imp != q.imp )
                delete imp;
            imp = q.imp;
            (imp->refCount)++;
            return *this;
        }

        ~String()     {
            (imp->refCount)--;
            if (imp->refCount <= 0 ) delete imp;
        }

    private:
        String(char** r)  {imp = new StringRep(r);}
        StringRep *imp;
};
```

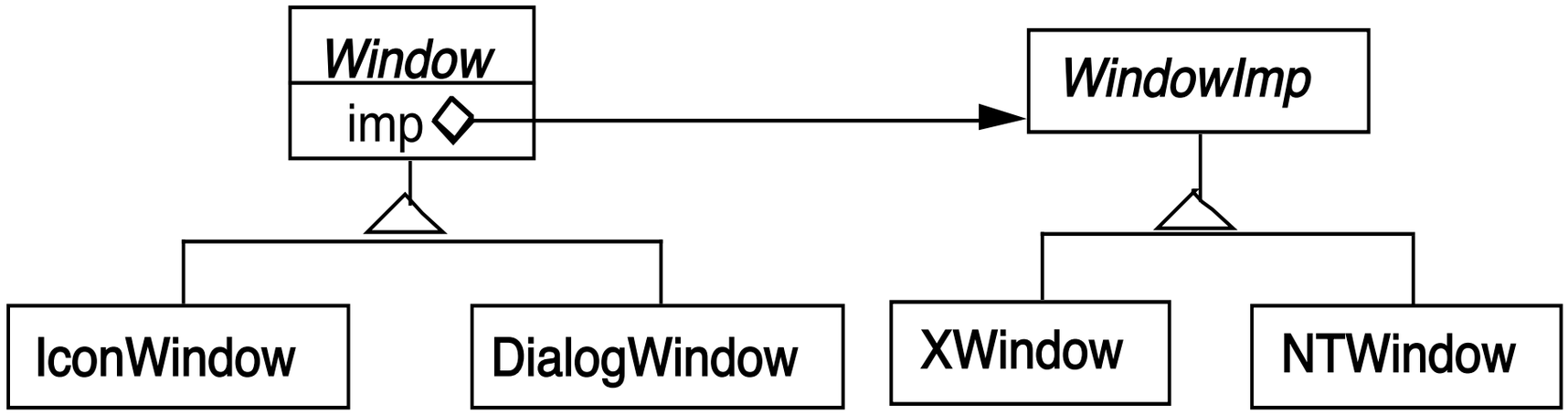
Why Use Bridge

Implementation selected at run-time

Implementation changed during run-time

Why Use Bridge

Abstraction & implementations are extensible by subclassing

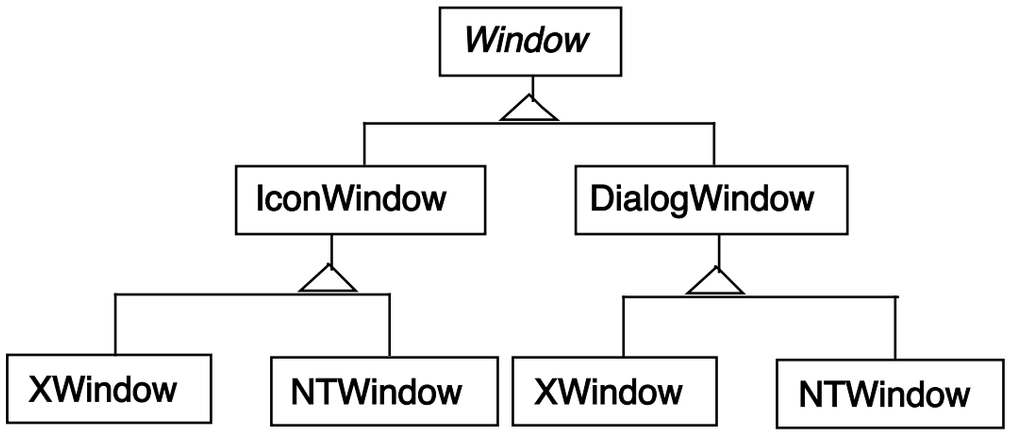


Why Use Bridge

When changes in the implementation should not require client code to be recompiled

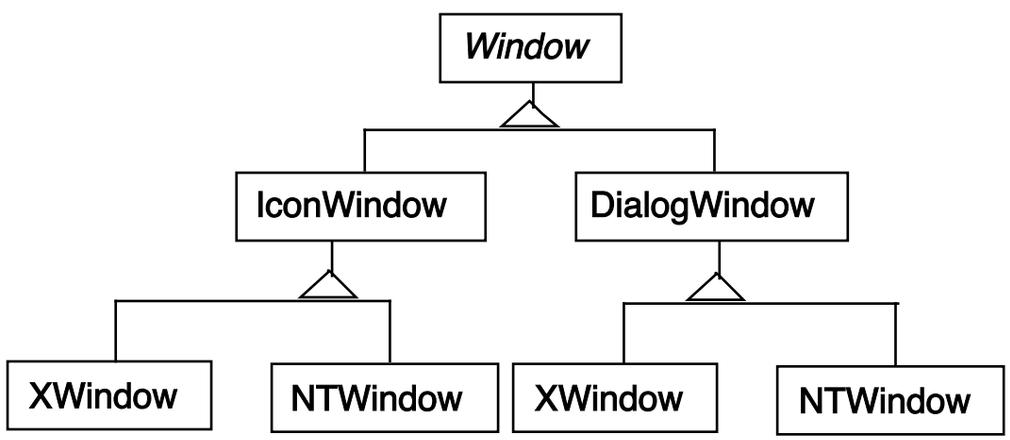
Why Use Bridge

Proliferation of classes



Why Use Bridge

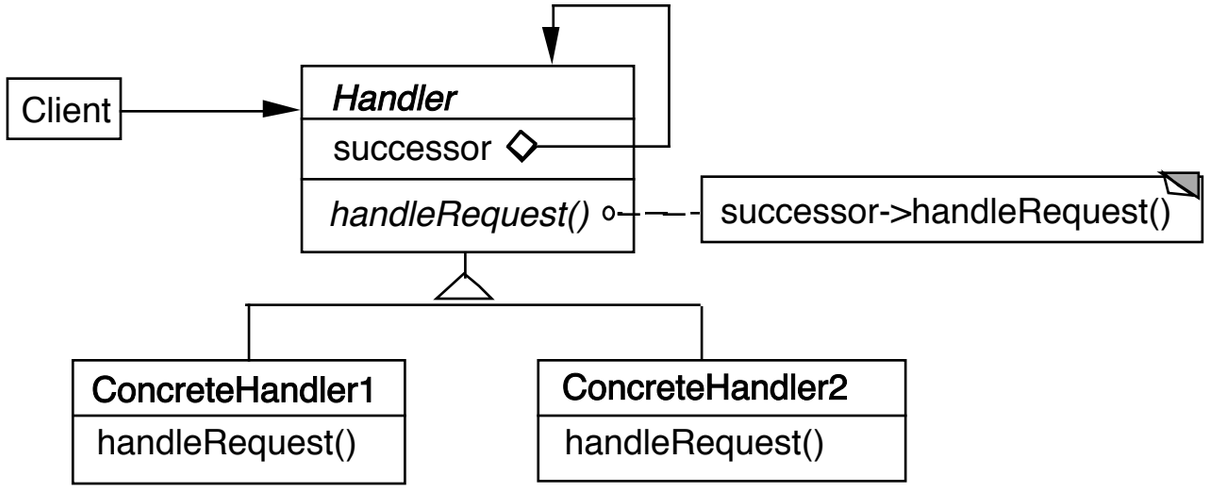
Share implementation among multiple objects



Bridge verses Adapter

Chain of Responsibility

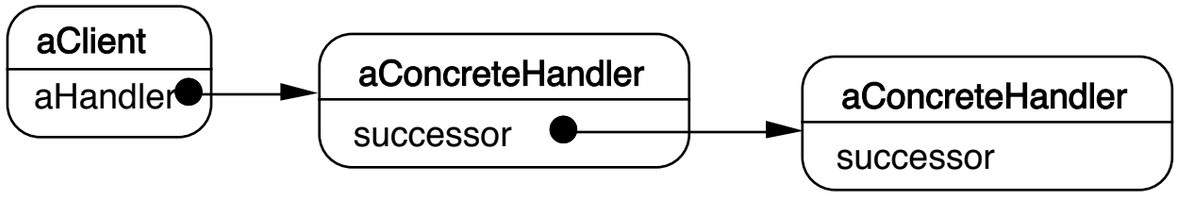
Chain of Responsibility



Dynamically create chain of handlers

Multiple handlers may be able to handle a request

Only one handler actually handles the request



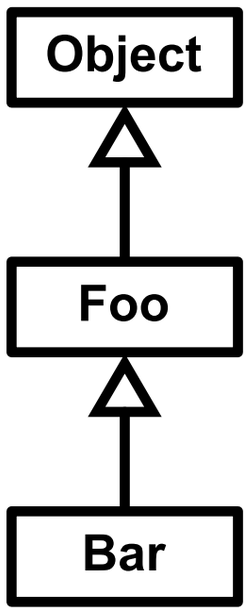
Consequences

Reduced coupling

Added flexibility in assigning responsibilities to objects

Not guaranteed that request will be handled

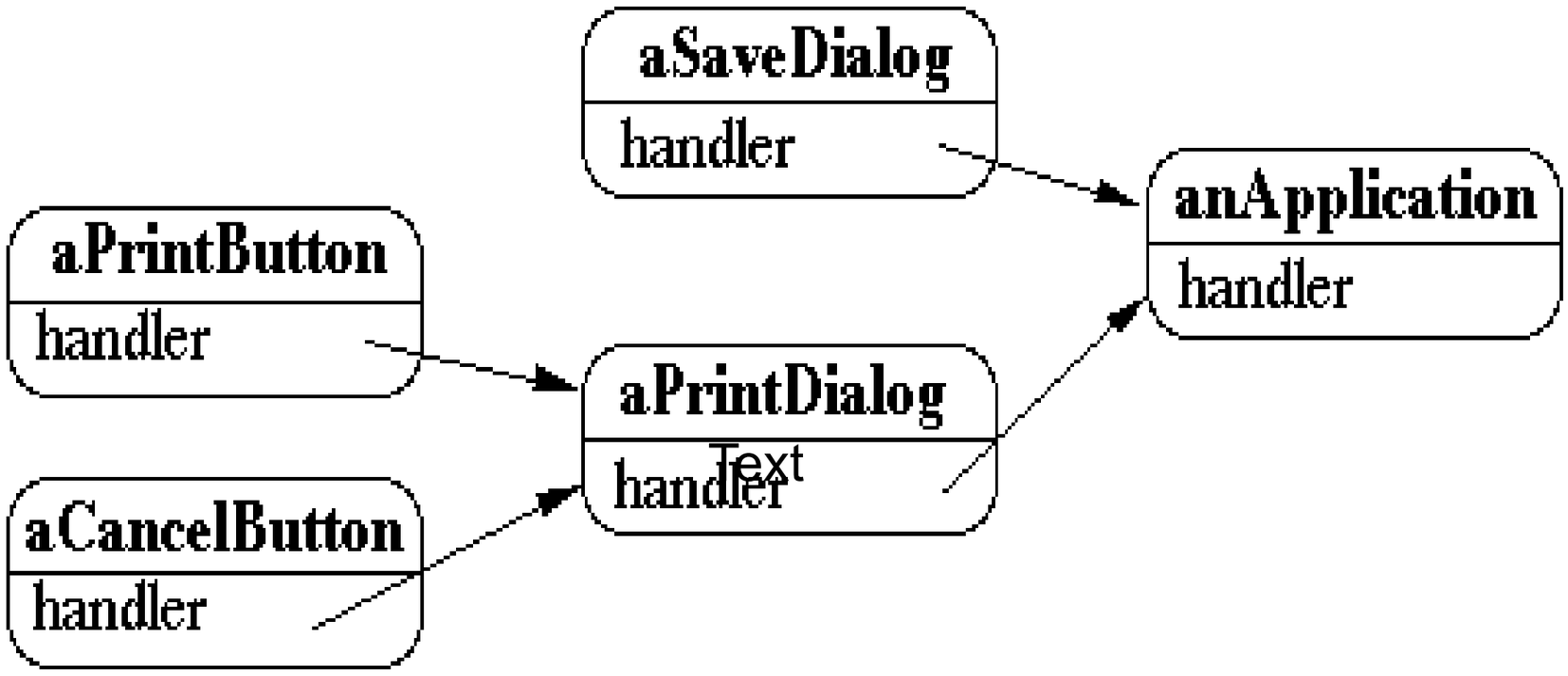
Finding Methods




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

Context Help System

User clicks on component for help



Tree of handlers

From specific to general

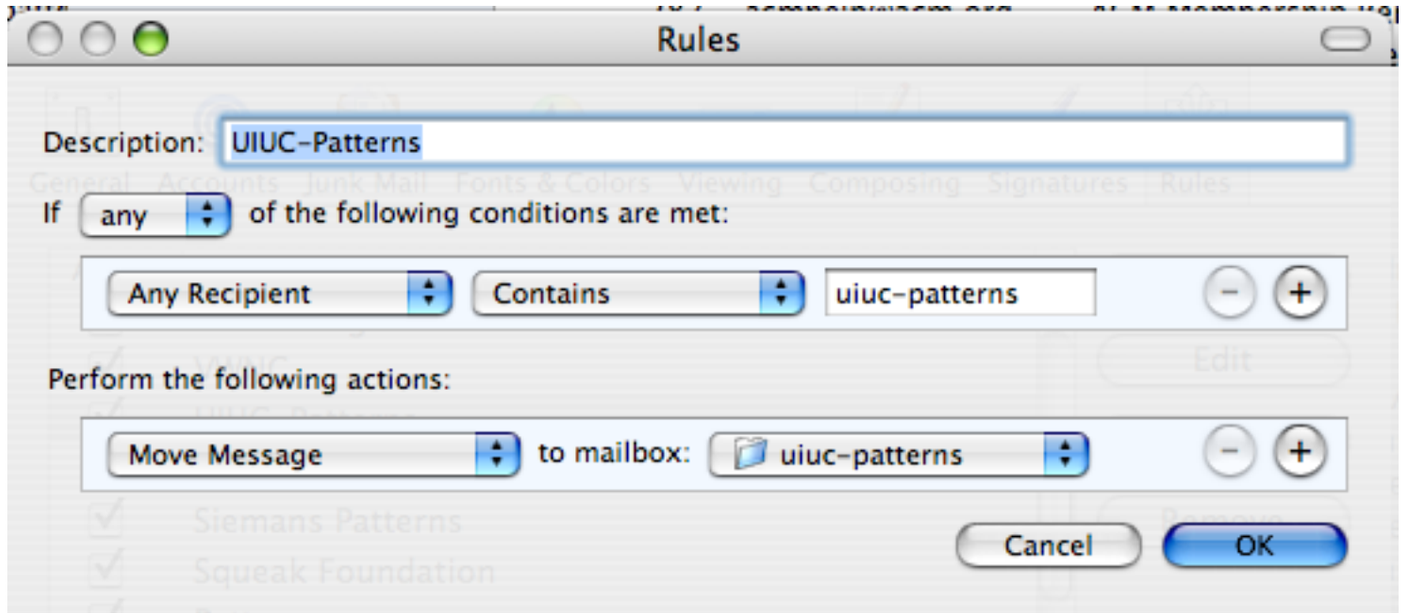
Email Filters in Mail Client

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

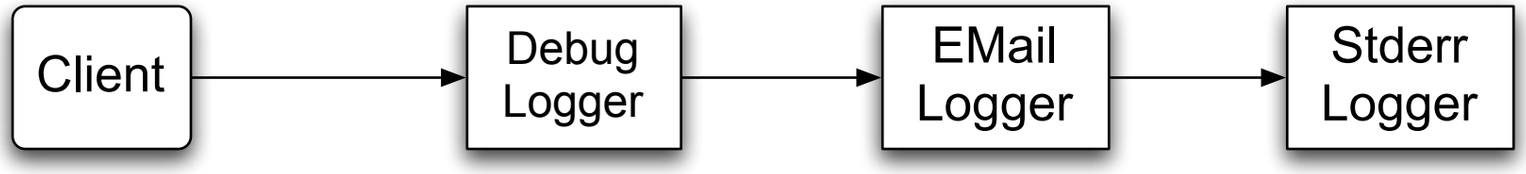
Microsoft Windows global keyboard events

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?

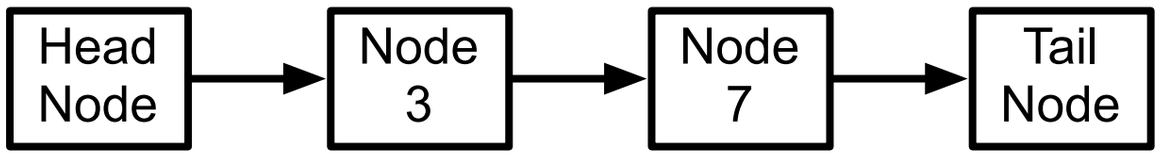
Object-Oriented Recursion

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



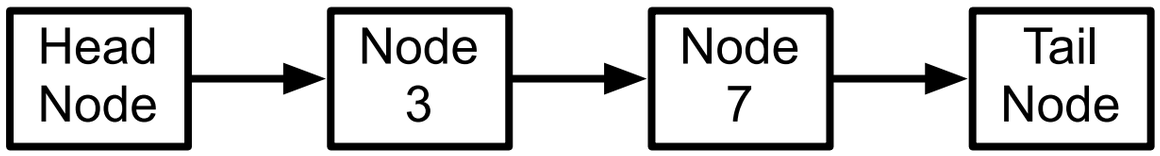
(3 7)

```
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

Pipes and Filters

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

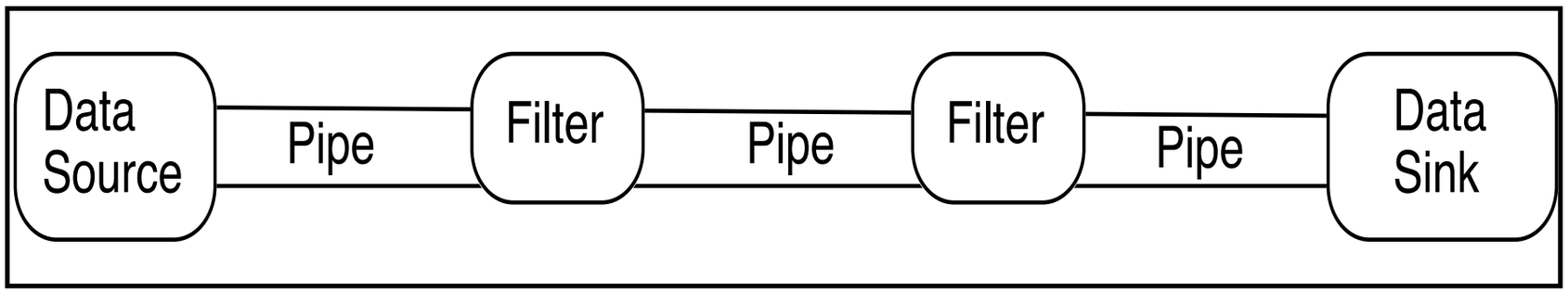
Divide task into multiple sequential processing steps or filter components

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

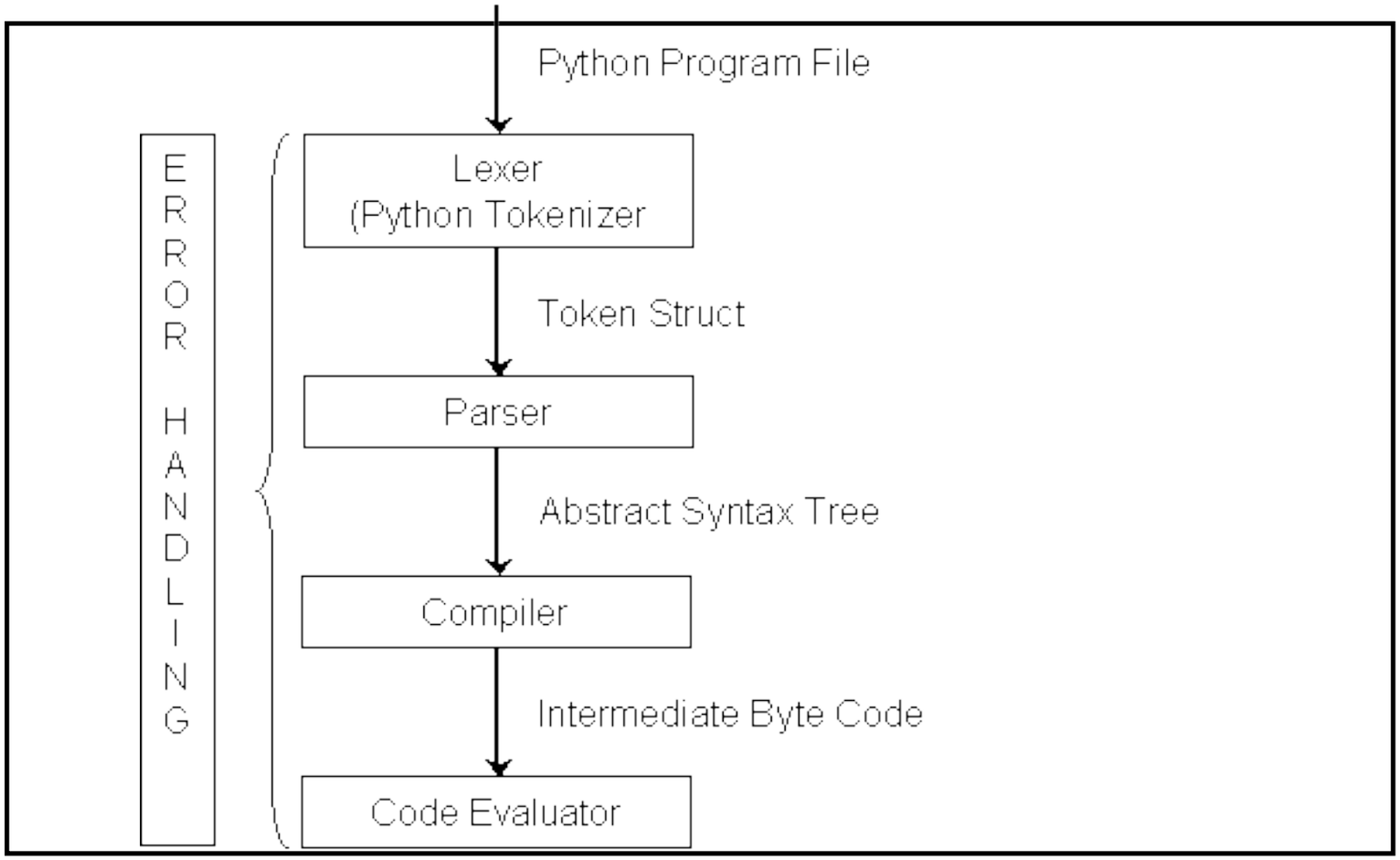
Pipes - connect the data source, filters and data sink

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

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

Intercepting Filter - Forces

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

Functional Programming

```
(defn sprinkles-milk2
    [amount]
    (-> amount
        base-price
        with-milk
        with-sprinkles
        compute-tax))
```

Collection Pipelines

Lays out a sequence of operations that pass a collection of items between them.

Special case of the Pipes and Filters pattern

"one of the most common, and pleasing, patterns in software"
Martin Fowler

```
String[] words = {"a", "ab", "abc", "abcd", "bat"};
List<String> wordList = Arrays.asList(words);
List<String> longWords
longWords = wordList.stream()
                        .filter( s -> s.length() > 2)
                        .filter( s -> s.charAt(0) == 'a')
                        .map( s -> s.toUpperCase())
                        .collect( Collectors.toList());
System.out.println(longWords);
```

https://martinfowler.com/articles/collection-pipeline/