

CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2009
Doc 5 Iterator, Filters, Null Object, Object Recursion
Feb 9, 2009

Copyright ©, All rights reserved. 2009 SDSU & Roger Whitney, 5500
Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent ([http://
www.opencontent.org/opl.shtml](http://www.opencontent.org/opl.shtml)) license defines the copyright on this
document.

References

Design Patterns: Elements of Reusable Object-Oriented Software, Gamma, Helm, Johnson, Vlissides, 1995, pp. 257-271

“Null Object”, Woolf, in Pattern Languages of Program Design 3, Edited by Martin, Riehle, Buschmann, Addison-Wesley, 1998, pp. 5-18

Pipe & Filter References

Pattern-Oriented Software Architecture: Vol 1 A System of Patterns, Buschmann et al, Wiley, 1996, pp 53-70.

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

Detailed Discussion

<http://john.cs.olemiss.edu/~hcc/softArch/notes/pipes.html>

Reading

Feb 10 - Iterator pattern

Feb 12 - Visitor and Strategy patterns

Feb 17 - Decorator and Command patterns

Feb 19 - Chapter 1 of Design Patterns, Gamma, Helm, Johnson, Vlissides

Linked List Assignment

Print out the strings that contain an odd number of characters.

How to satisfy the requirements and still maintain LinkedList abstraction?

Iterator Pattern

Provide a way to access the elements of a collection sequentially without exposing its underlying representation

Iterator Solution

Java

```
LinkedList<Strings> strings =  
    new LinkedList<Strings>();
```

code to add strings

```
for (String element : strings) {  
    if (element.size % 2 == 1)  
        System.out.println(element);  
}
```

```
Iterator<String> list = strings.iterator();  
while (list.hasNext()){  
    String element = list.next();  
    if (element.size % 2 == 1)  
        System.out.println(element);  
}
```

Ruby Iterator Examples

a = [1, 2, 3, 4]

a.each { x puts x}	1 2 3 4
result = a.collect { x x + 10} puts result	11 12 13 14
result = a.find_all { x x > 2 } puts result	3 4
puts a.any? { x x > 2}	true
puts a.detect { x x > 2 }	3

Ruby Solution

```
strings = LinkedList.new
```

```
code to add strings
```

```
result = strings.find_all { |element| element.size % 2 = 1 }  
puts result
```

Pattern Parts

Intent

Motivation

Applicability

Structure

Participants

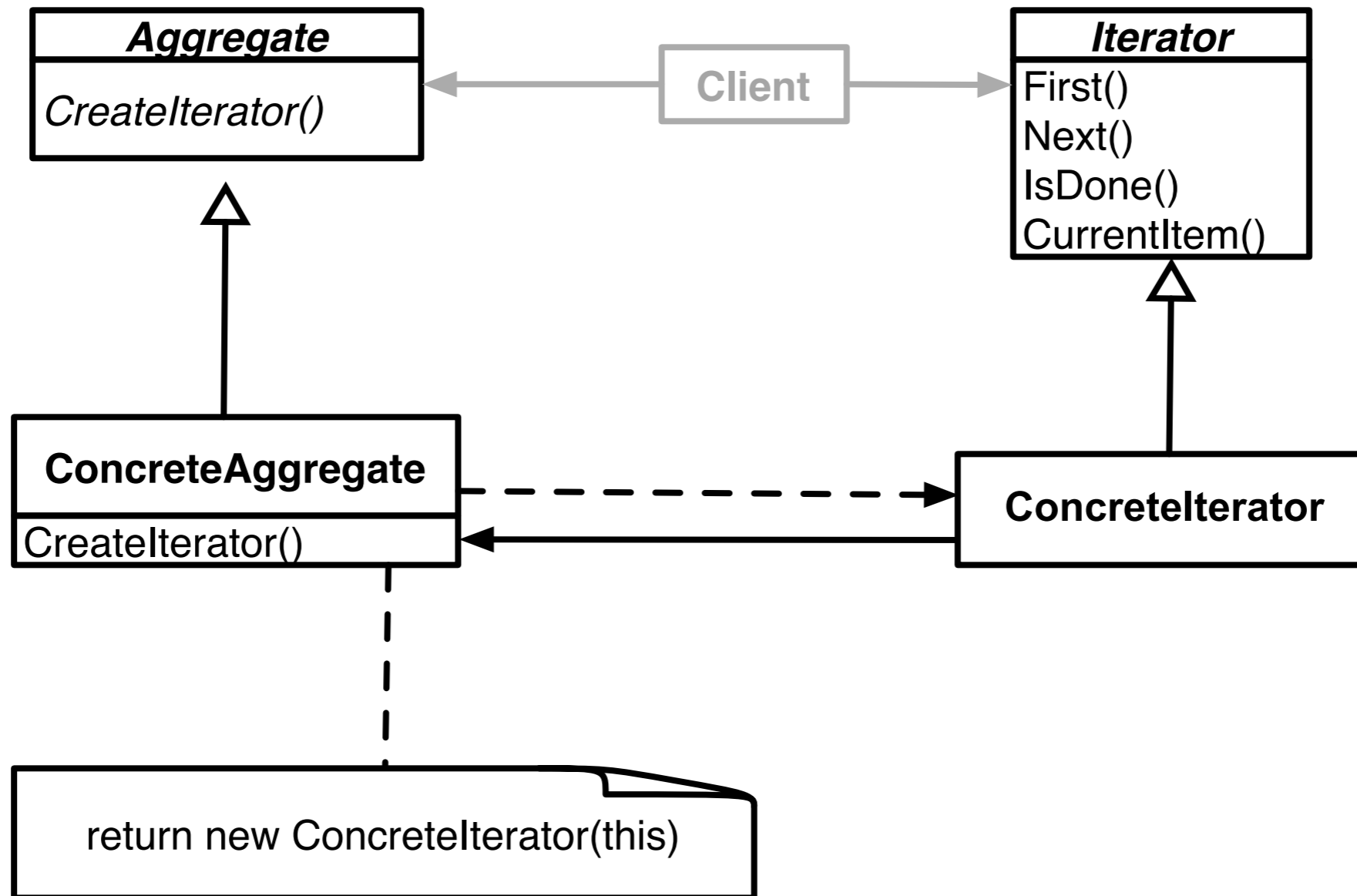
Collaborations

Consequences

Implementation

Sample Code

Iterator Structure



Issue - What is the big deal?

```
var numbers = new LinkedList();
```

code to add numbers

```
Iterator list = numbers.iterator();
while ( list.hasNext() ) {
    Integer a = (Integer) list.next();
    int    b = a.intValue();
    if ((b % 2) == 0)
        System.out.println( x );
}
```

```
var numbers = new LinkedList();
```

code to add numbers

```
for (int k =0; k < numbers.size(); k++ ) {
    Integer a = (Integer) numbers.get(k);
    int    b = a.intValue();
    if ((b % 2) == 0)
        System.out.println( x );
}
```

Issues - Concrete vs. Polymorphic Iterators

Concrete

```
Reader iterator = new StringReader( "cat");  
int c;  
while (-1 != (c = iterator.read() ))  
    System.out.println( (char) c);
```

Polymorphic

```
Vector listOfStudents = new Vector();  
  
// code to add students not shown  
  
Iterator list = listOfStudents.iterator();  
while ( list.hasNext() )  
    System.out.println( list.next() );
```

Memory leak issue in C++, Why?

Issue - Who Controls the Iteration?

External (Active)

```
var numbers = new LinkedList();
```

code to add numbers

```
Vector evens = new Vector();  
Iterator list = numbers.iterator();  
    while ( list.hasNext() ) {  
        Integer a = (Integer) list.next();  
        int    b = a.intValue();  
        if ((b % 2) == 0)  
            evens.add(a);  
    }
```

Internal (Passive)

```
numbers = LinkedList.new
```

code to add numbers

```
evens = numbers.find_all { |element| element.even? }
```

Issue - Who Defines the Traversal Algorithm

Object being iterated

Iterator

Issue - Robustness

What happens when items are added/removed from the iteratee while an iterator exists?

```
Vector listOfStudents = new Vector();
```

```
// code to add students not shown
```

```
Iterator list = listOfStudents.iterator();
```

```
listOfStudents.add( new Student( "Roger" ) );
```

```
list.hasNext();           //What happens here?
```

Are Java's Input Streams & Readers Iterators?

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

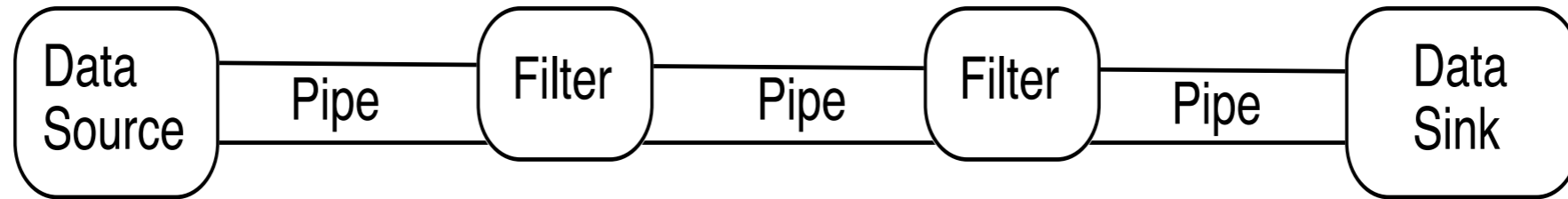
Divide task into multiple sequential processing steps or filter components

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

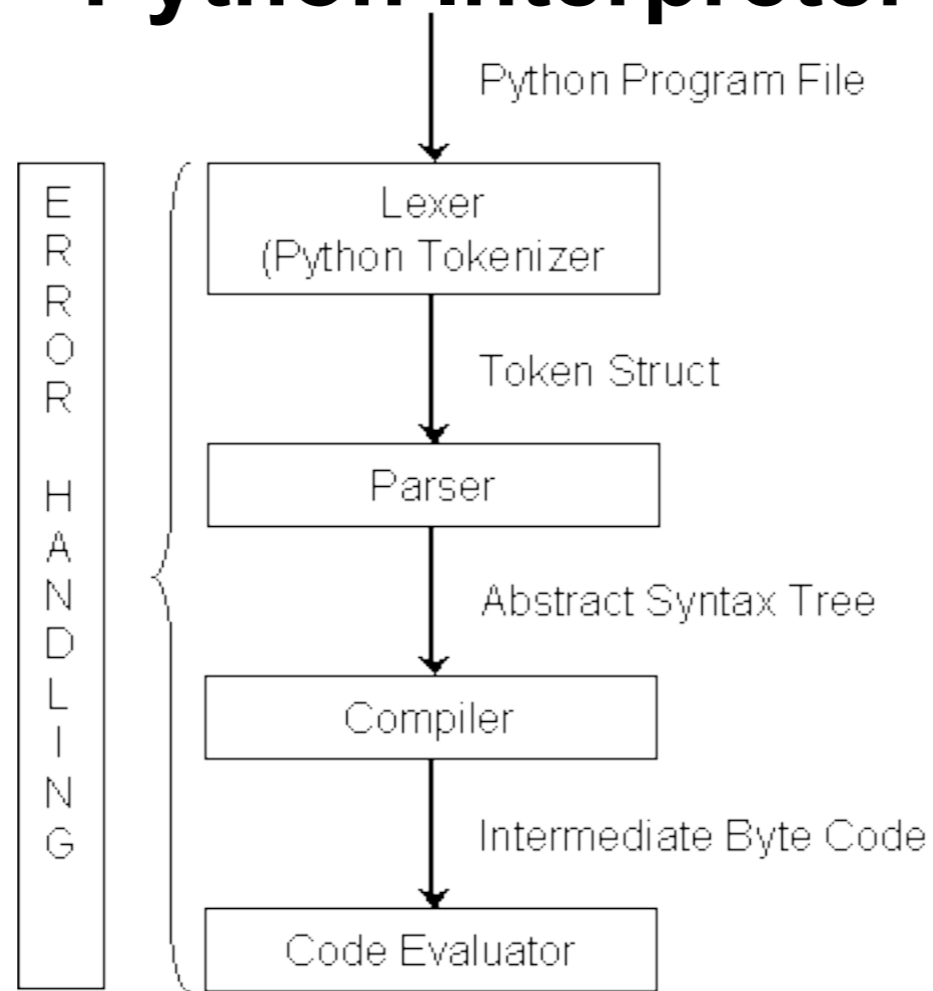
Pipes - connect the data source, filters and data sink

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

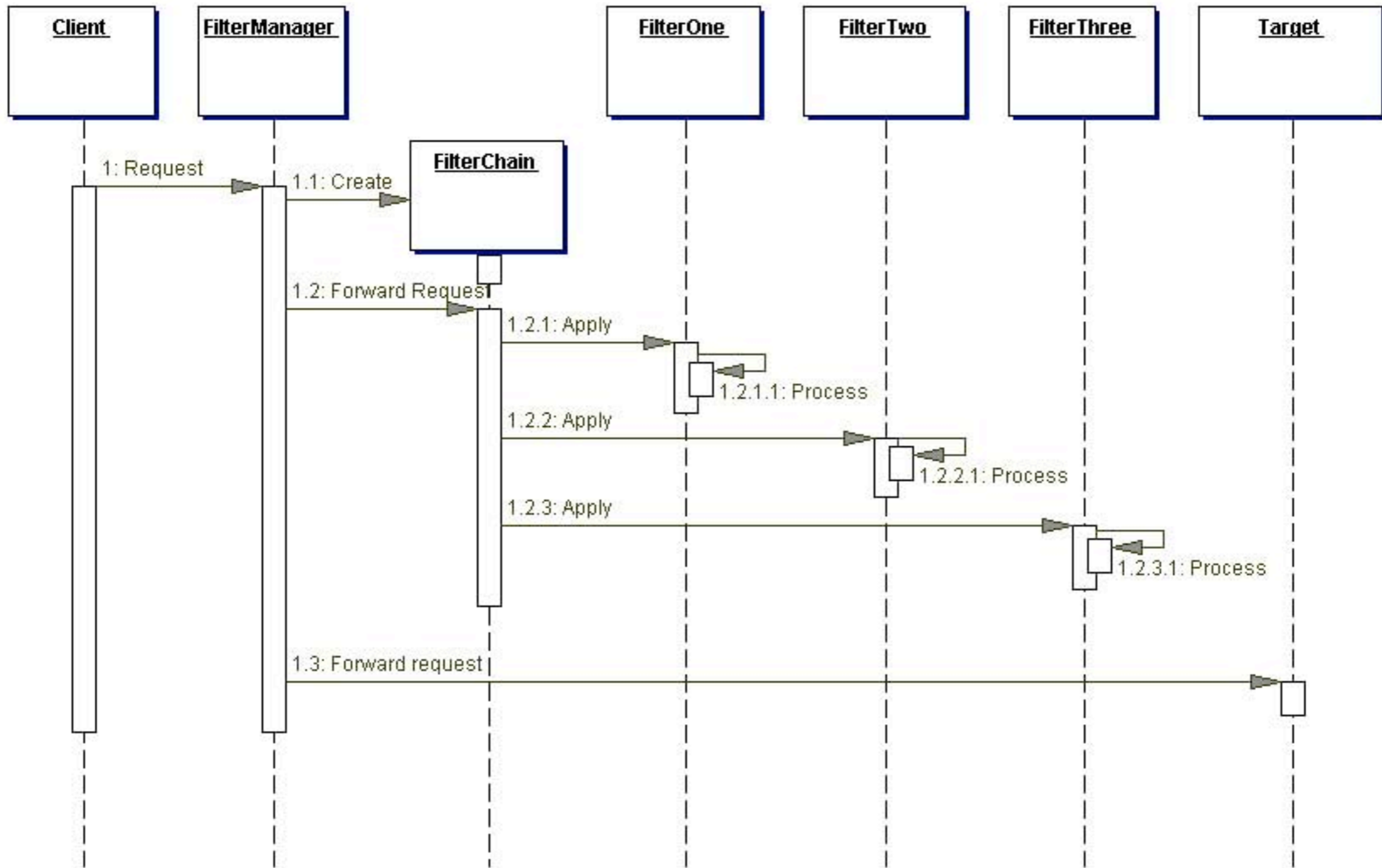
Logging and authentication

Debugging and transformation of output for a specific client

Uncompressing and converting encoding scheme of input

Intercepting Filter - Solution

Create pluggable filters to process common services



Linked List Problem

Use a filter for odd sized Strings

Java

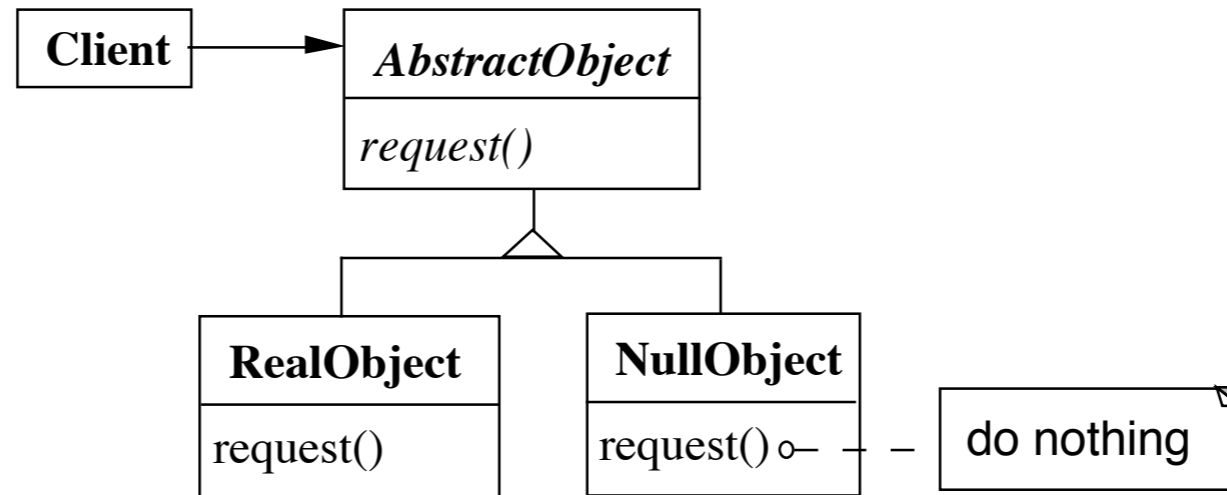
```
LinkedList<Strings> strings =  
new LinkedList<Strings>();
```

code to add strings

```
Iterator<String> list = strings.iterator();  
Iterator<Strings> odd = new OddSizeFilter( list);  
while (odd.hasNext()){  
    String element = list.next();  
    System.out.println(element);  
}  
}
```

Null Object

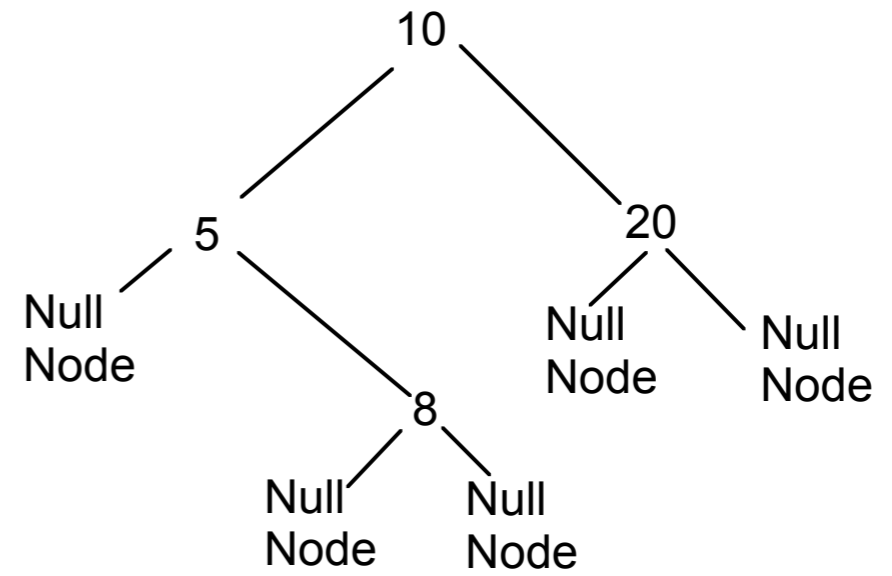
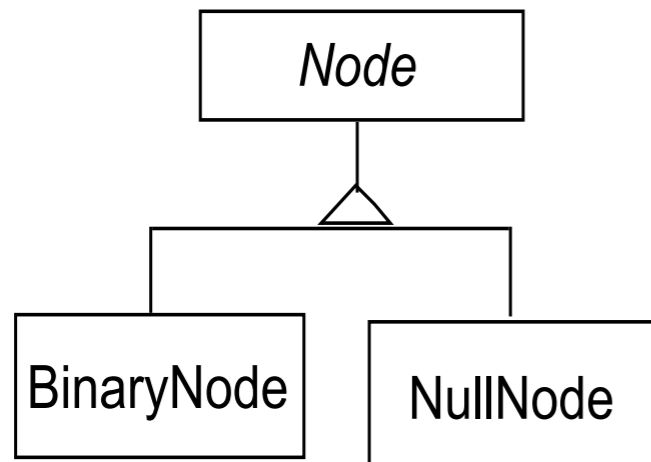
Null Object



NullObject implements all the operations of the real object,

These operations do nothing or the correct thing for nothing

Null Object & Binary Search Tree



Normal BST **Comparing Normal Tree with Tree with Null** With Null Nodes

```
public class BinaryNode {  
    Node left  
    Node right;  
    int key;  
  
    public boolean includes( int value ) {  
        if (key == value)  
            return true;  
        else if ((value < key) & left == null) )  
            return false;  
        else if (value < key)  
            return left.includes( value );  
        else if (right == null)  
            return false;  
        else  
            return right.includes(value);  
    }  
    etc.  
}
```

```
public class BinaryNode extends Node {  
    Node left = new NullNode();  
    Node right = new NullNode();  
    int key;  
  
    public boolean includes( int value ) {  
        if (key == value)  
            return true;  
        else if (value < key )  
            return left.includes( value );  
        else  
            return right.includes(value);  
    }  
    etc.  
}  
  
public class NullNode extends Node {  
    public boolean includes( int value ) {  
        return false;  
    }  
    etc.  
}
```

Applicability

When to use Null Objects

Some collaborator instances should do nothing

You want clients to ignore the difference between a collaborator that does something and one that does nothing

Client does not have to explicitly check for null or some other special value

You want to be able to reuse the do-nothing behavior so that various clients that need this behavior will consistently work in the same way

Applicability

When not to use Null Objects

Very little code actually uses the variable directly

The code that does use the variable is well encapsulated

The code that uses the variable can easily decide how to handle the null case and will always handle it the same way

Consequences

Disadvantages

Advantages

Uses polymorphic classes

Simplifies client code

Encapsulates do nothing behavior

Makes do nothing behavior reusable

Forces encapsulation

Makes it difficult to distribute or mix into
the
behavior of several collaborating objects

May cause class explosion

Forces uniformity

Is non-mutable

Implementation

Too Many classes

Multiple Do-nothing meanings

Try Adapter pattern

Transformation to RealObject

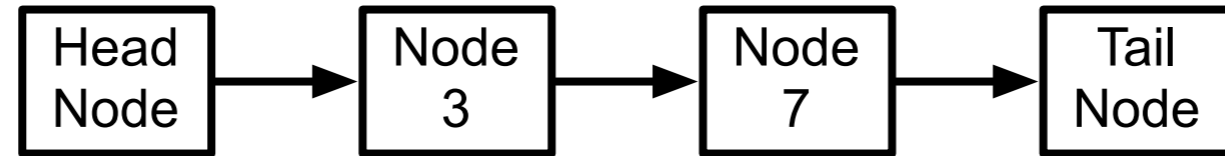
Try Proxy pattern

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



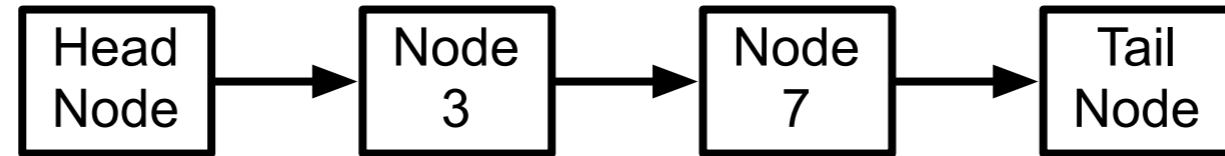
(3 7)

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

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

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

Without tail recursion doing this on a long linked list could cause a stack overflow. So while it may not be a good idea to do this on a linked list it does provide a simple example to explain the idea.



```
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);  
    }  
}
```