CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2007
Doc 3 Iterator & Null Object
Feb 6, 2007

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, Buschmmann, Addison-Wesley, 1998, pp. 5-18

Binary Search Tree Assignment

Print out the elements that end in either an 'a' or an 'e' in alphabetic order.

How to satisfy the requirements and still maintain BST abstraction?

Iterator Solution

Java

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

code to add strings

for (String element : strings) {
        if (element.endsWith("e" ) )
                System.out.println(element);
}

Iterator<String> list = strings.iterator();
while (list.hasNext()){
        String element = list.next();
        if (element.endsWith("e"))
        System.out.println(element);
        }
}
```

Ruby

```
numbers  = BST.new

code to add numbers

numbers.each { |element|
    if element.endsWith("e")
        puts 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

Pattern Parts

Intent

Motivation

Applicability

Structure

Participants

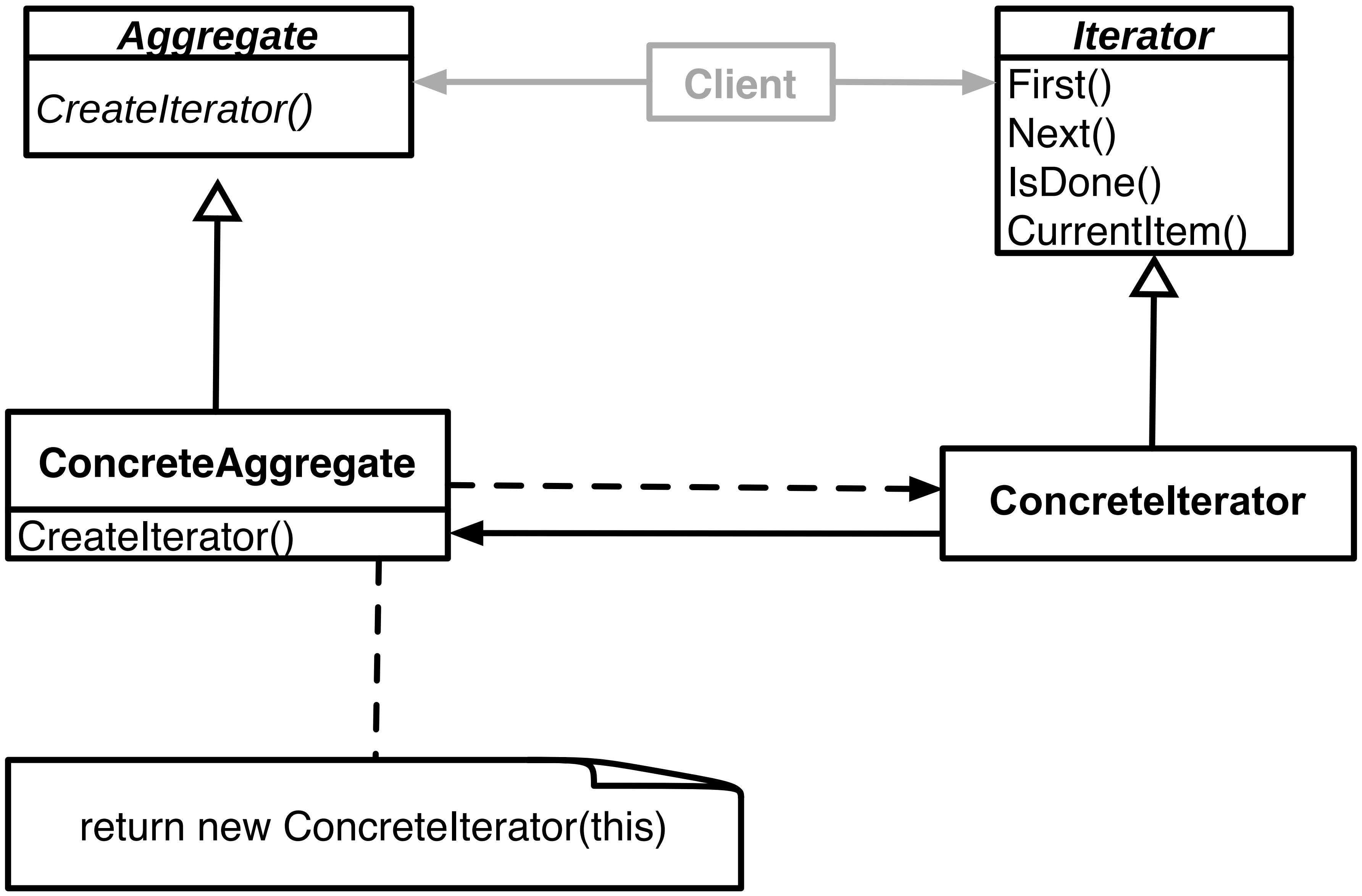
Collaborations

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();
}
```

```
var numbers = new LinkedList();

code to add numbers

for (int k =0; k < numbers.size(); k++ ) {
        Integer a = (Integer) numbers.get(k);
}
```

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)

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

code to add strings

Iterator<String> list = strings.iterator();
while (list.hasNext()){
        String element = list.next();
        if (element.endsWith("e"))
        System.out.println(element);
        }
}
```

Internal (Passive)

```
numbers  = BST.new

code to add numbers

numbers.each { |element|
    if element.endsWith(“e”)
                puts element
}
```

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?

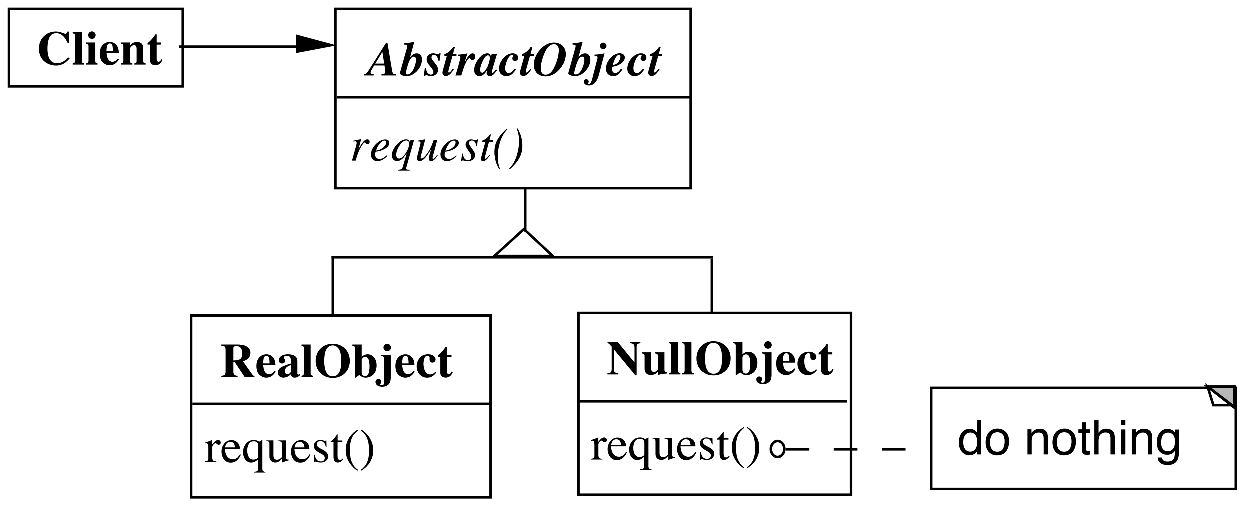
```
Vector listOfStudents = new Vector();

// code to add students not shown

Iterator list = listOfStudents.iterator();
listOfStudents.add( new Student( "Roger") );

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

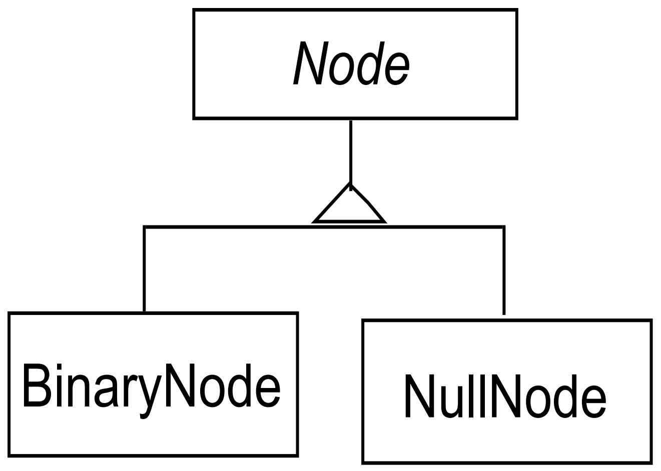
Null Object

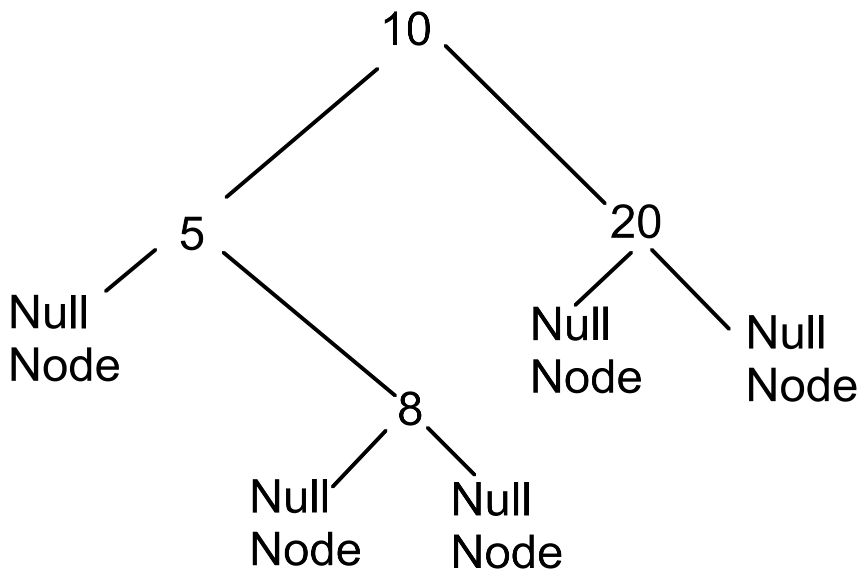


NullObject implements all the operations of the real object,

These operations do nothing or the correct thing for nothing

Null Object & Binary Search Tree





Normal Tree vs Tree with Null Nodes

Normal BST

```
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.
}
```

With Null Nodes

```
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

When not to use Null Objects

Very little code actually uses the variable directly

The code that does use the variable is well encapsulated - at least in one class

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

Consequences

Advantages

Uses polymorphic classes

Simplifies client code

Encapsulates do nothing behavior

Makes do nothing behavior reusable

Disadvantages

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