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
LinkedList Assignment

Print out the even elements in the list from the front to the back of the list

How to satisfy the requirements and still maintain LinkedList abstraction?
Java

```java
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 );
}
```

Ruby

```ruby
numbers = LinkedList.new

code to add numbers

numbers.each { |element|
    puts element if element.even?
}
```
Collecting all the evens

Java
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);
   }

Ruby
numbers = LinkedList.new

code to add numbers

evens = numbers.find_all { |element| element.even? }
## Ruby Iterator Examples

```ruby
a = [1, 2, 3, 4]

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a.each {</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td>result = a.collect {</td>
<td>x</td>
</tr>
<tr>
<td>puts result</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>14</td>
</tr>
<tr>
<td>result = a.find_all {</td>
<td>x</td>
</tr>
<tr>
<td>puts result</td>
<td>4</td>
</tr>
<tr>
<td>puts a.any? {</td>
<td>x</td>
</tr>
<tr>
<td>puts a.detect {</td>
<td>x</td>
</tr>
</tbody>
</table>
Pattern Parts

Intent
Motivation
Applicability
Structure
Participants
Collaborations
Consequences
Implementation
Sample Code
Iterator Structure

```
CreateIterator()

Aggregate
CreateIterator()
ConcreteAggregate
First()
Next()
IsDone()
CurrentItem()
Iterator
return new ConcreteIterator(this)
Client
ConcreteIterator

ConcreteAggregate
CreateIterator()

return new ConcreterIterator(this)
```
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 );
}

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)

```java
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)

```java
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?
Null Object

```
Client
  AbstractObject
    request()
    RealObject
      request()
    NullObject
      request() do nothing
```

NullObject implements all the operations of the real object,

These operations do nothing or the correct thing for nothing
Null Object & Binary Search Tree

Node

BinaryNode NullNode

BinaryTree:

10
   /   
  5   20
     /   
    8   
       /   
      Null Node Null Node
Comparing Normal Tree with Tree with Null Nodes

Normal BST

```java
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

```java
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.
}
```

```java
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

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uses polymorphic classes</td>
<td>Forces encapsulation</td>
</tr>
<tr>
<td>Simplifies client code</td>
<td>Makes it difficult to distribute or mix into the behavior of several collaborating objects</td>
</tr>
<tr>
<td>Encapsulates do nothing behavior</td>
<td>May cause class explosion</td>
</tr>
<tr>
<td>Makes do nothing behavior reusable</td>
<td>Forces uniformity</td>
</tr>
<tr>
<td></td>
<td>Is non-mutable</td>
</tr>
</tbody>
</table>
Implementation

Too Many classes

Multiple Do-nothing meanings

Try Adapter pattern

Transformation to RealObject

Try Proxy pattern