CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2004
Doc 3 Iterators & Null Object
Contents

References

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

Refactoring: Improving the Design of Existing Code, Fowler, 1999, pp. 260-266

"Null Object", Woolf, in *Pattern Languages of Program Design 3*, Edited by Martin, Riehle, Buschmmann, Addison-Wesley, 1998, pp. 5-18

Reading

Design Patterns: pp. 257-271

Future Readings

Composite & Visitor patterns

Iterator

Provides a way to access elements of an aggregate object sequentially without exposing its underlying representation

Java Example

Enumeration, Iterator, and Streams in Java are iterators

```
Vector listOfStudents = new Vector();

// code to add students not shown

Iterator list = listOfStudents.iterator();
while ( list.hasNext() )
   {Student x = list.next();
   System.out.println( x );
   }
```

C# Example

```
Student[ ] listOfStudent;
// code to add students not shown
foreach (Student x in listOfStudent )
   {
   Console.WriteLine(x.ToString());
   }
```

Smalltalk Examples

Streams, do:, select:, reject:, collect:, detect:, inject:into: are iterators in Smalltalk

```
| sum  |
sum := 0.
#( 1 7 2 3 9 3 50) do: [:each | sum := sum + each squared].
^sum
```

```
#( 1 7 2 3 9 3 50) inject: 0 into:
   [:partialSum :number | partialSum + number squared]
```

```
'this is an example' select: [:each |  each isVowel ]
```

What's The Big Deal?

```
void print(ArrayList list)
   {
   for( int k = 0; k < list.size(); k++ )
      System.out.println( list.get(k) );
   }

void print(LinkedList list )
   {
   Node current = list.first();
   System.out.println( current );
   while (current.hasNext() )
      {
      current = current.next();
      System.out.println( current );
      }
   }

void print(Collection list )
   {
   Iterator items = list.iterator();
   while (items.hasNext() )
      {
      System.out.println( items.next() );
      }
   }

print: aCollection
   aCollection do:
      [:each |
      Transcript
         show: each;
         cr]
```

What's The Big Deal?

Iterator abstracts out underlying representation of collection

Programmer does not have to know implementation details of each type of collection

Can write code that works for wide range of collects

Do not have to change code if change the type of collection used

Design Principle 1

Program to an interface, not an implementation

Use abstract classes (and/or interfaces in Java) to define common interfaces for a set of classes

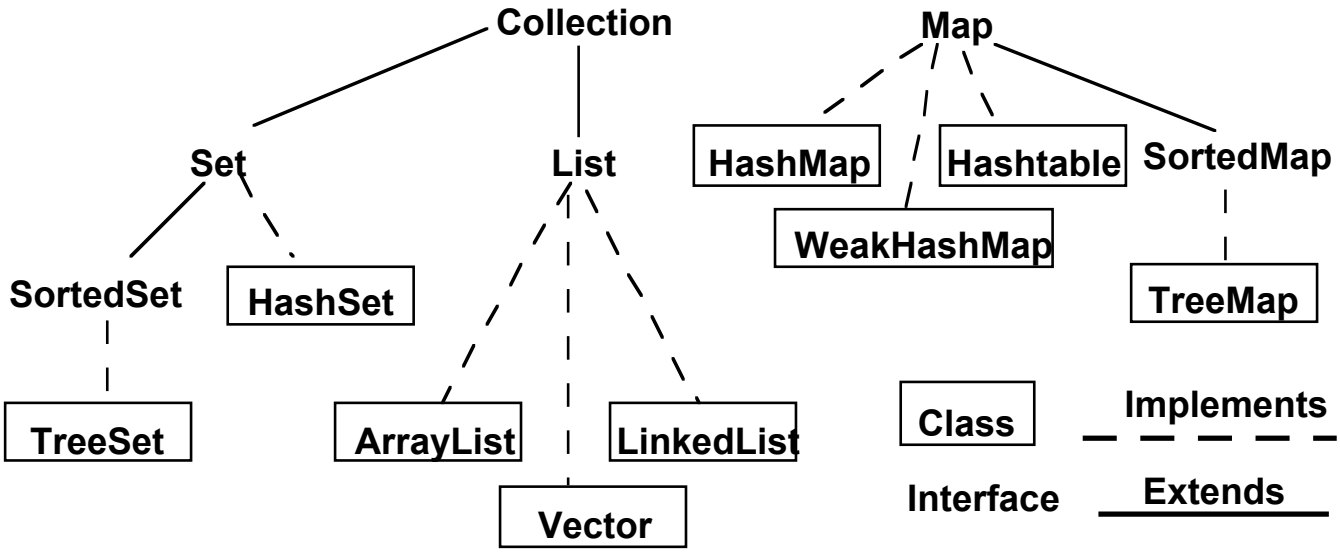
Declare variables to be instances of the abstract class not instances of particular classes

Benefits of programming to an interface

Client classes/objects remain unaware of the classes of objects they use, as long as the objects adhere to the interface the client expects

Client classes/objects remain unaware of the classes that implement these objects. Clients only know about the abstract classes (or interfaces) that define the interface.

Programming to an Interface
Java Collections



Java Iterators & Arrays?

Arrays are common collections

How can one get an iterator on a Java array?

How would you pass an array to the following function?

```
void print(Collection list )
   {
   Iterator items = list.iterator();
   while (items.hasNext() )
      {
      System.out.println( items.next() );
      }
   }
```

Java Iterators in Practice

```
void printA(Collection list )
   {
   Iterator items = list.iterator();
   while (items.hasNext() )
      System.out.println( items.next() );
   }

void printB(String[] list)
   {
   for (int k = 0; k < list.length; k++)
      System.out.println( list[k]);
   }
```

Programmers are not used to programming to an interface

printA requires as much typing as printB

Smalltalk/C# Iterators in Practice

```
printA: aCollection
   1 to: aCollection size do: [:index |
   Transcript
      show: (aCollection at: index);
      cr.

printB: aCollection
   aCollection do: [:each |
      Transcript
         show: each;
         cr.

   Print(IEnumerable list )
      {
      foreach (Object x in list)
         {
         Console.WriteLine( x.ToString());
         }
      }
```

printB requires less typing than printB

Iterators are part of the C# language

Programmers use iterators just to avoid extra work

Sample Implementation of Java Enumerator

```
class VectorIterator implements Enumeration {
   Vector iteratee;
   int count;

   VectorIterator(Vector v) {
      iteratee = v;
      count = 0;
   }

   public boolean hasMoreElements() {
      return count < iteratee.elementCount;
   }

   public Object nextElement() {
      synchronized (iteratee) {
         if (count < iteratee.elementCount)
            return iteratee.elementData[count++];
      }
   throw new NoSuchElementException("VectorIterator");
   }
}
```

The iterator is using privileged access to Vectors fields

Issues
Concrete vs. Polymorphic Iterators
Concrete

Use Explicit Iterator Type

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

Polymorphic

Actual type of iterator is not known

```
Vector listOfStudents = new Vector();

// code to add students not shown

Iterator list = listOfStudents.iterator();
while ( list.hasNext() )
   Console.println( list.next() );
```

Polymorphic iterators can cause problems with memory leaks in C++ because they are on the heap!

Who Controls the iteration?
External (Active)

```
Vector listOfStudents = new Vector();

// code to add students not shown

Iterator list = listOfStudents.iterator();

while ( list.hasNext() )
   Console.println( list.next() );
```

Iteration control code is repeated for each use of the iterator

Who Controls the iteration?
Internal (Passive)

```
'this is an example' select: [:each |  each isVowel ]
```

Control code is inside the iterator

Programmer

- Does not repeat control code
- Can focus on what to do not how to do it

Who Defines the Traversal Algorithm?
Object being Iterated

Iterator can store where we are

In a Vector this could mean the index of the current item

In a tree structure it could mean a pointer to current node and stack of past nodes

```
BinaryTree searchTree = new BinaryTree();

// code to add items not shown

Iterator aSearch = searchTree.getIterator();
Iterator bSearch = searchTree.getIterator();
Object first = searchTree.nextElement( aSearch );
Object stillFirst = searchTree.nextElement( bSearch );
```

Iterator

Makes it easier to have multiple iterator algorithms on same type

On Vector class, why not have a reverseIterator which goes backwards?

In a complex structure the iterator may need access to the iteratee's implementation

How Robust is the iterator?

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

```
Vector listOfStudents = new Vector();

// code to add students not shown

Enumeration list = listOfStudents.elements();
Iterator failFastList = listOfStudents.iterator();

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

list.hasMoreElements();
failFastList.hasNext();          //Exception thrown here
```

Additional Iterator Operations

Augmenting basic iteration operations may improve their usefulness

previous()
   back up one location

add(Object item)
   add item to the iteratee at current location

remove()
   remove the current item from the iteratee

skipTo(some location, item or condition)
   go to the location indicated

mark()
   mark current location for future return

Iterators and Privileged Access

An iterator may need privileged access to the aggregate structure for traversal

Iterators for Composites

Traversing a complex structure like a graph, tree, or composite can be difficult

An internal iterator can use recursion to keep track of where to go next

For example using a depth-first search algorithm on graph

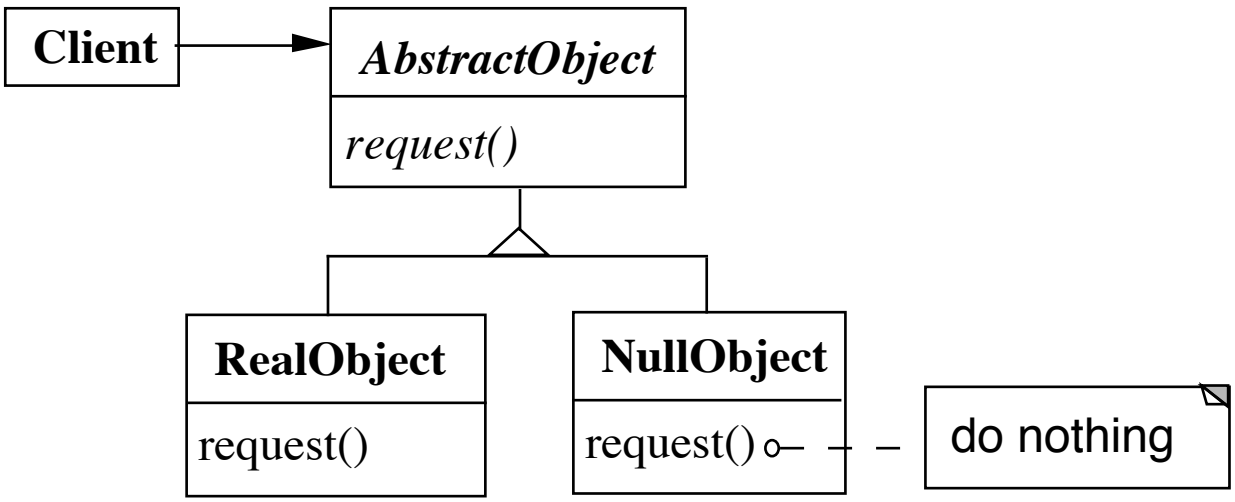
If each element in the aggregate “knows” how to traverse to the next element and previous element, than an external iterator can be used

Null Iterator

A Null iterator for the empty aggregates can be useful of each.

NullObject

Structure



NullObject implements all the operations of the real object,

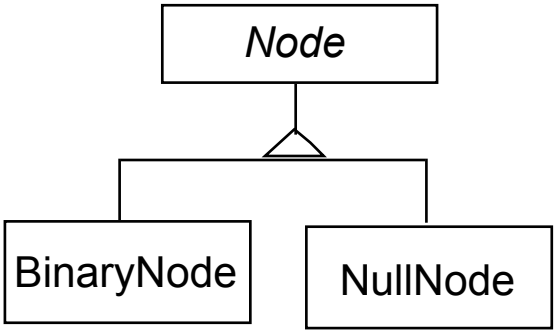
These operations do nothing or the correct thing for nothing

Binary Search Tree Example
Without Null Object

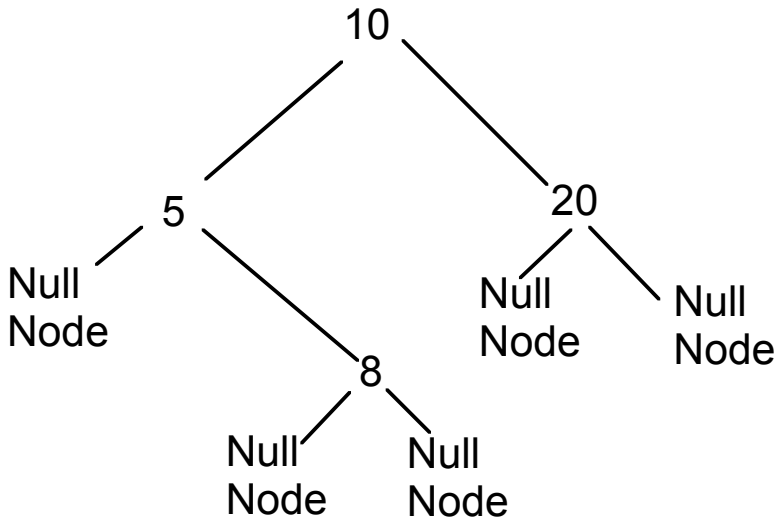
```
public class BinaryNode {
   Node left = new NullNode();
   Node right = new NullNode();
   int key;

   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.
}
```

Binary Search Tree Example
Class Structure



Object Structure



Searching for a Key

```
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.
}
```

Comments on Example

- BinaryNode always has two subtrees

  No need check if left, right are null

- Since NullNode has no state just need one instance

  Use singleton pattern for the one instance

- Access to NullNode is usually restricted to BinaryNode

  Forces indicate that one may not want to use the Null Object pattern

  However, familiarity with trees makes it easy to explain the pattern

- Implementing an add method in NullNode

  Requires reference to parent or

  Use proxy

Refactoring
Introduce Null Object[1]

You have repeated checks for a null value

Replace the null value with a null object

Example

```
customer isNil
   ifTrue: [plan := BillingPlan basic]
   ifFalse: [plan := customer plan]
```

becomes:

- Create NullCustomer subclass of Customer with:

```
NullCustomer>>plan
   ^BillingPlan basic
```

- Make sure that each customer variable has either a real customer or a NullCustomer

Now the code is:

```
plan := customer plan
```

- Often one makes a Null Object a singleton

[1] Refactoring Text, pp. 260-266

Applicability

Use the Null Object pattern when:

- 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

Use a variable containing null or some other special value instead of the Null Object pattern when:

- 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

  Different clients may have different idea of what “do nothing” means

- Is non-mutable

  NullObject objects cannot transform themselves into a RealObject

  become: message in Smalltalk allows null objects to “transform” themselves into real objects

Implementation

- Too Many classes

  Eliminate one class by making NullObject a subclass of RealObject

- Multiple Do-nothing meanings

  If different clients expect do nothing to mean different things use Adapter pattern to provide different do-nothing behavior to NullObject

- Transformation to RealObject

  In some cases a message to NullObject should transform it to a real object

  Use the proxy pattern

Generalized Null Object Pattern

A generalized Null Object pattern based on Objective-C with an implementation in Smalltalk can be found at:

http://www.smalltalkchronicles.net/edition2-1/null_object_pattern.htm