CS 635 Advanced Object-Oriented Design & Programming

Spring Semester, 2006

Doc 4 Visitor

Feb 2, 2006

Reference

Design Patterns: Elements of Resuable Object-Oriented Software, Gamma, Helm, Johnson, Vlissides, Addison-Wesley, 1995, pp. 331-344

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?

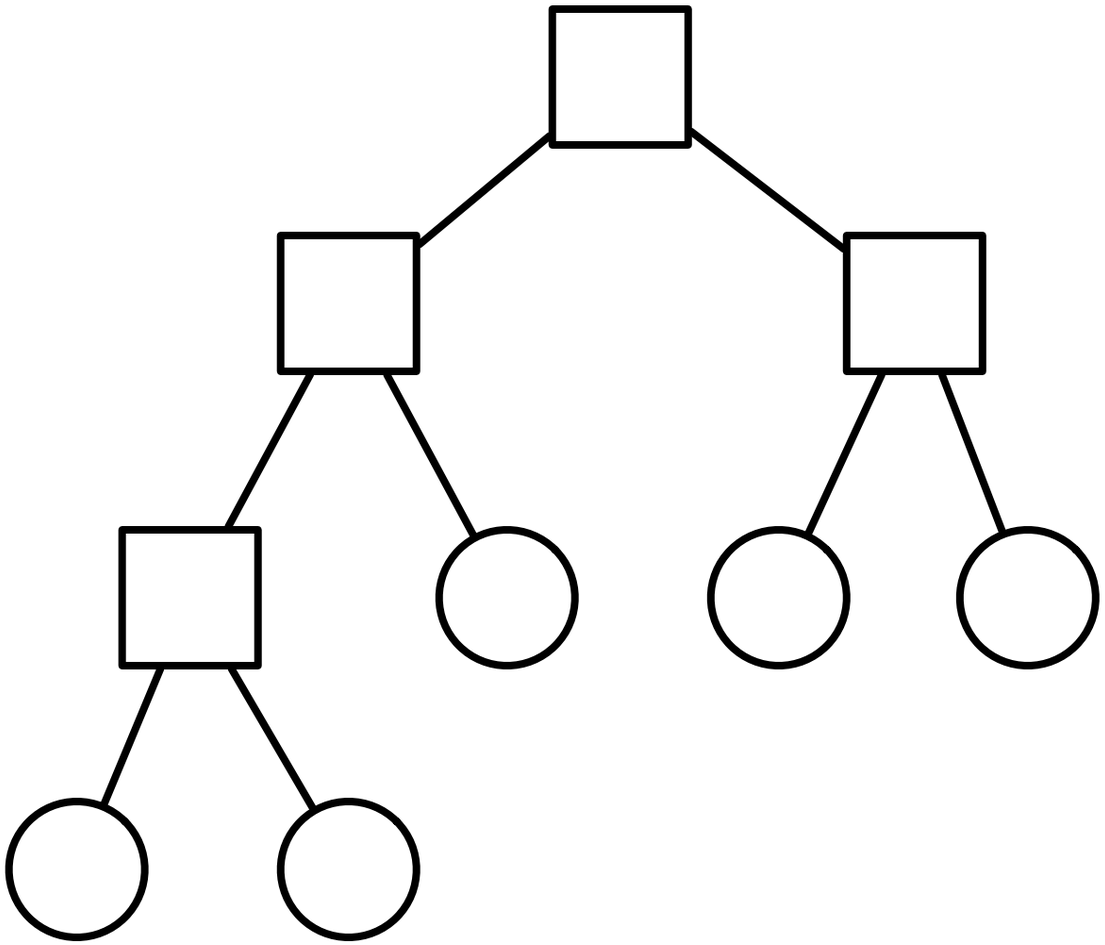
Visitor

Intent

Represent an operation to be performed on the elements of an object structure

Visitor lets you define a new operation without changing the classes of the elements on which it operates

Tree Example



```
class Node { ... }

class BinaryTreeNode extends Node {...}

class BinaryTreeLeaf extends Node {...}
```

Tree Example

```
class BinaryTreeNode extends Node {
        public void accept(Visitor aVisitor) {
                aVisitor.visitBinaryTreeNode( this );
        }
}

class BinaryTreeLeaf extends Node {
        public void accept(Visitor aVisitor) {
                aVisitor.visitBinaryTreeLeaf( this );
        }
}

abstract class Visitor {
        abstract void visitBinaryTreeNode( BinaryTreeNode );
        abstract void visitBinaryTreeLeaf( BinaryTreeLeaf );
}

class HTMLPrintVisitor extends Visitor {
        public void visitBinaryTreeNode( BinaryTreeNode x ) {
                HTML print code here
        }
        public void visitBinaryTreeLeaf( BinaryTreeLeaf x){ ...}
}
```

Put operations into separate object - a visitor

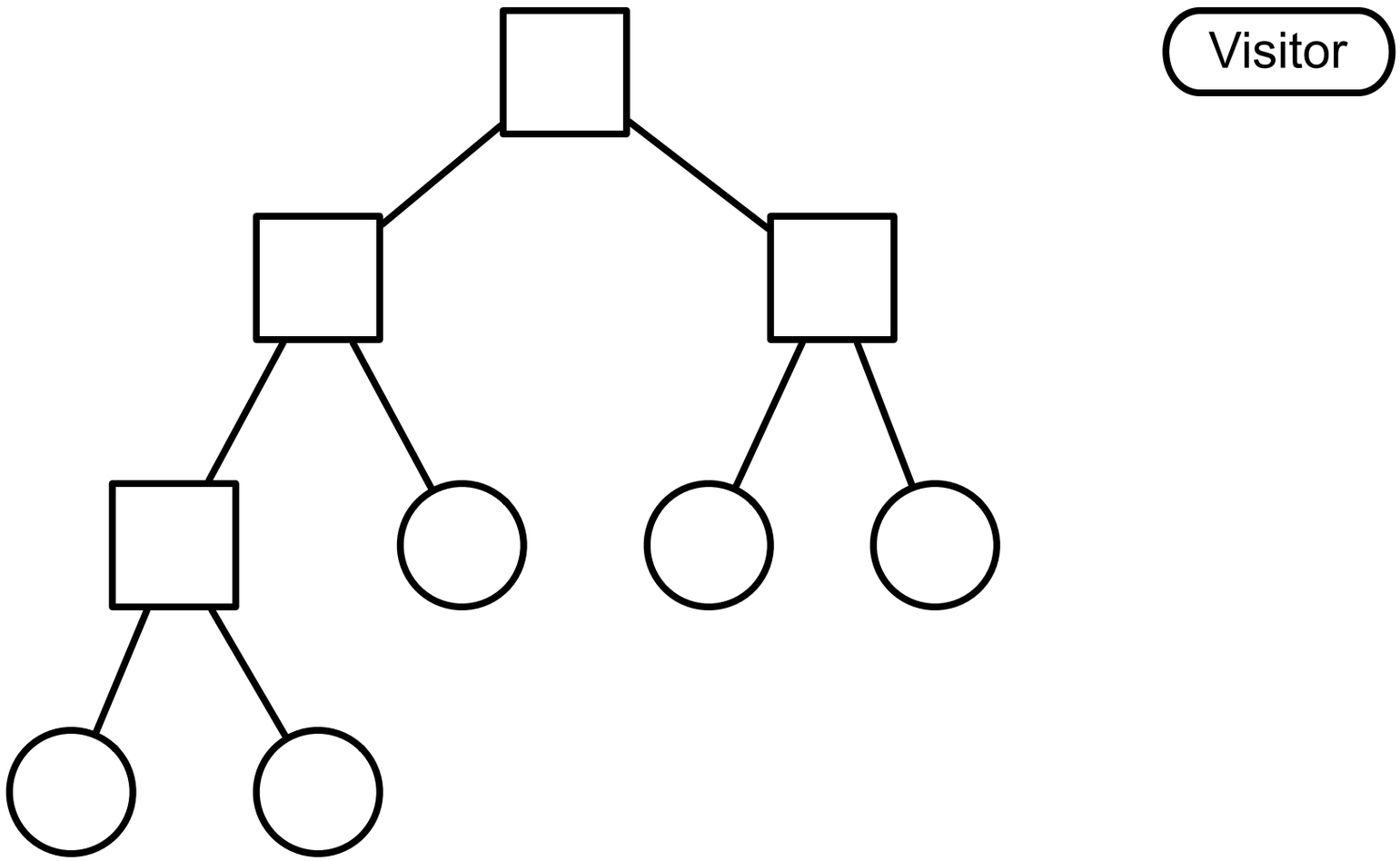
Pass the visitor to each element in the structure

The element then activates the visitor

Visitor performs its operation on the element

Each visitX method only deals with on type of element

Tree Example



Double Dispatch

Note that a visit to one node requires two method calls

```
Node example = new BinaryTreeLeaf();
Visitor traveler = new HTMLPrintVisitor();
example.accept( traveler );
```

example.accept() calls aVisitor.visitBinaryTreeNode(this);

The first method selects the correct method in the Visitor class

The second method selects the correct Visitor class

Issue - Who does the traversal?

Visitor

Elements in the Structure

Iterator

What is Wrong with This?

```
class Node {
	public void accept(Visitor aVisitor) {
		aVisitor.visit( this );
	}
}

abstract class Visitor {
	abstract void visit( Node );
}

class HTMLPrintVisitor extends Visitor {
	public void visit( Node x ) {
		if x is BinaryTreeNode {
			blah
		}
		else if x is BinaryTreeLeaf {
			more blah
		}
	}
}
```

When to Use the Visitor

When an object structure contains many classes of objects with differing interfaces, and you want to perform operations on these objects that depend on their concrete classes

When many distinct and unrelated operations need to be preformed on objects in an object structure and you want to avoid cluttering the classes with these operations

When the classes defining the structure rarely change, but you often want to define new operations over the structure

Consequences

Visitors makes adding new operations easier

Visitors gathers related operations, separates unrelated ones

Adding new ConcreteElement classes is hard

Visiting across class hierarchies

Accumulating state

Breaking encapsulation

Avoiding the accept() method

Visitor pattern requires elements to have an accept method

Sometimes this is not possible

- You don’t have the source for the elements

Aspect Oriented Programming

AspectJ eleminates the need for an accept method in aspect oriented Java

AspectS provides a similar process for Smalltalk