

CS 635 Advanced Object-Oriented Design &  
Programming  
Spring Semester, 2007  
Doc 4 Visitor  
Feb 8, 2007

Copyright ©, All rights reserved. 2007 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (<http://www.opencontent.org/opl.shtml>) license defines the copyright on this document.

## References

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

Magritte Meta-Described Web Application Development, Lukas Renggli, June 2006, Master Thesis Universität Bern, <http://www.iam.unibe.ch/~scg/Archive/Diploma/Reng06a.pdf>

## 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?

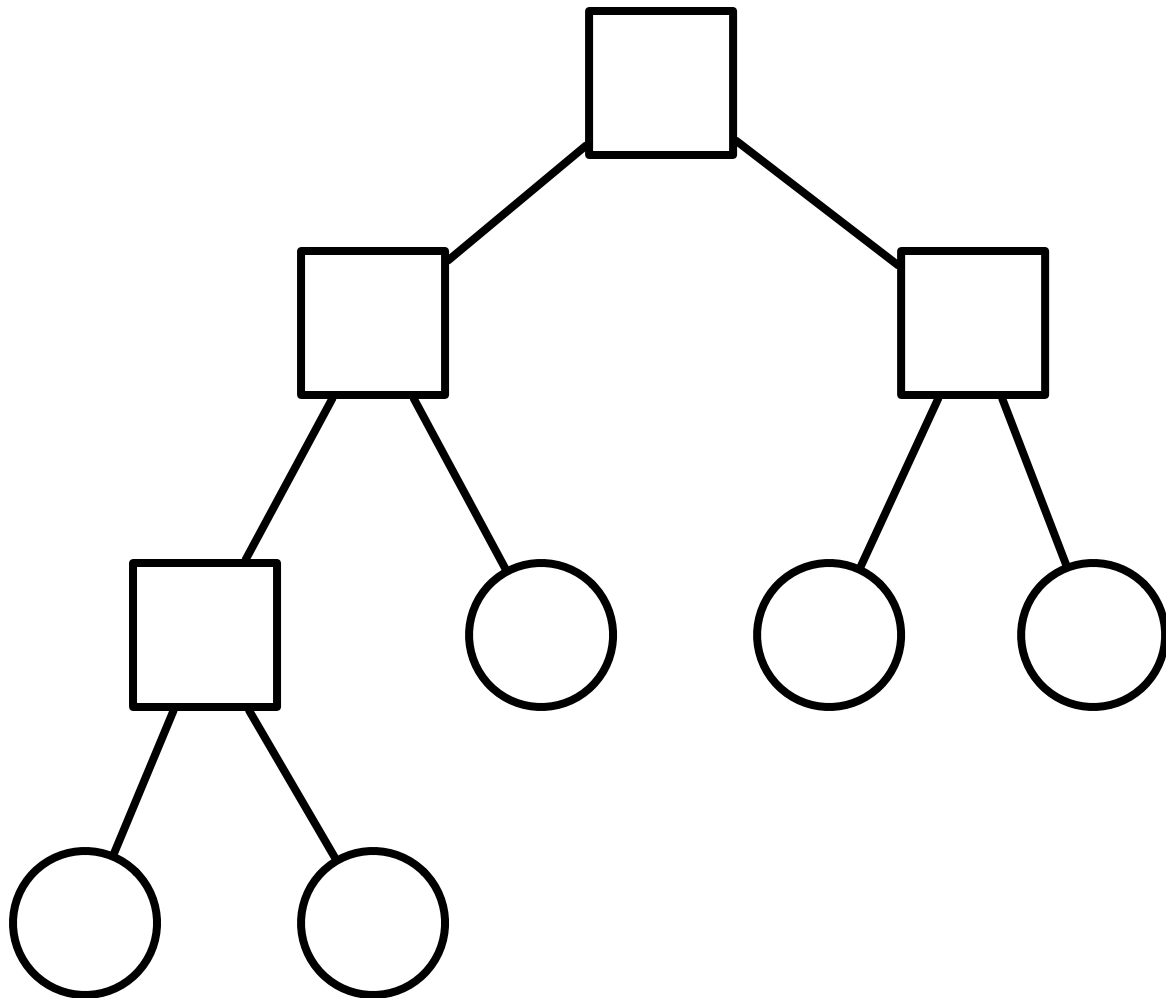
# Visitor

## 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

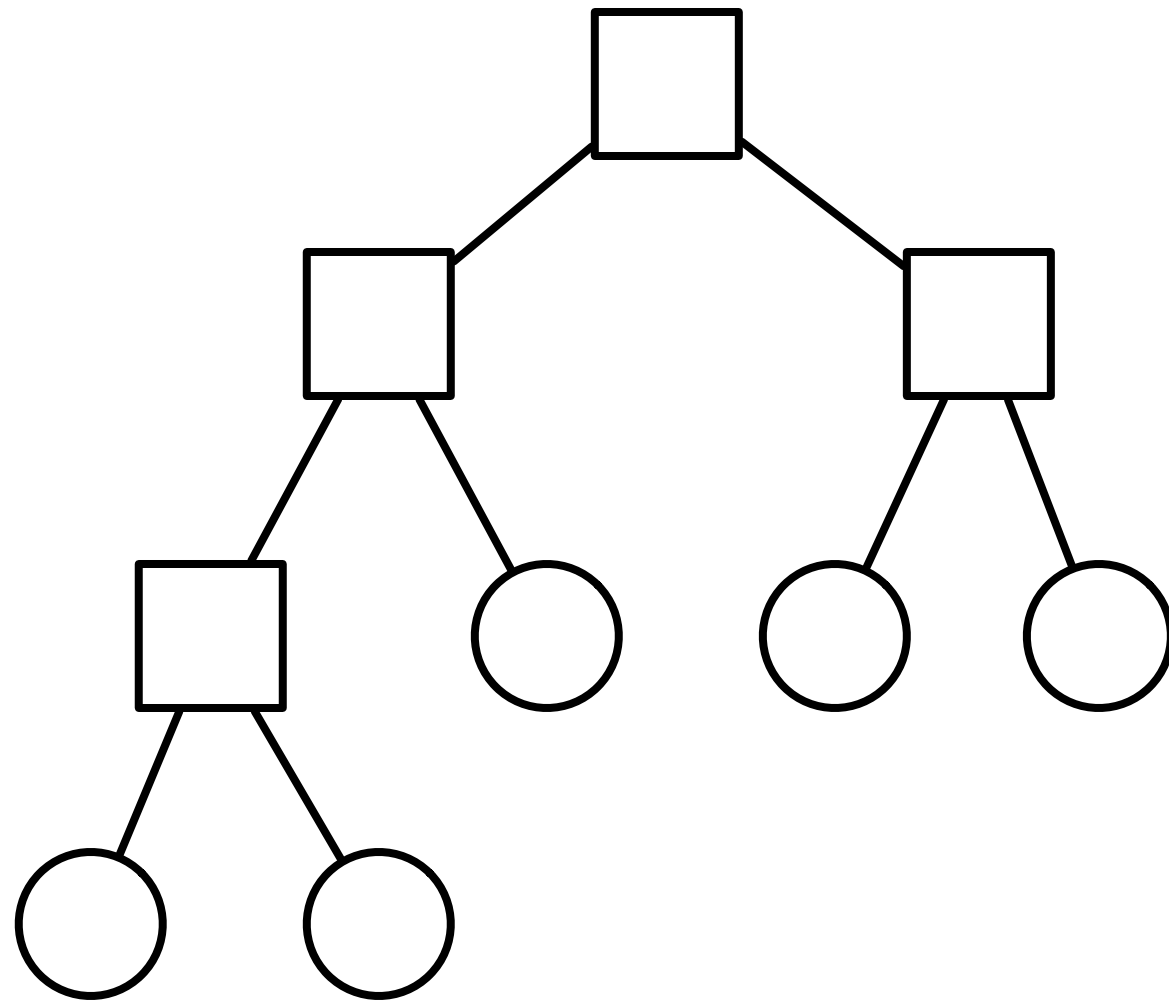
Pass the visitor to each element in the structure

The element then activates the visitor

Visitor performs its operation on the element

Each visitX method only deals with one type of element

# Tree Example



Visitor

# Double Dispatch

Note that a visit to one node requires two method calls

```
Node example = new BinaryTreeNode();  
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

Have 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 performed 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 eliminates the need for an accept method in aspect oriented Java

AspectS provides a similar process for Smalltalk

# Why not use one of this instead of the Visitor?

```
package example;
class BinaryTree {
    public Iterator iterator() {...}
    ...
}

class DoFoo {
    Iterator elements;
    public DoFoo(BinaryTree tree) {
        elements = tree.iterator();
    }

    public void doIt() {
        while (elements.hasNext() ) {
            Integer next = (Integer) elements.next
();
            do foo here with next
        }
    }
}
```

# Magritte

Web applications have data (domain models)

We need to

- Display the data

- Enter the data

- Validate data

- Store Data

# Magritte

For each field in a domain model (class) provide a description

Description contains

Data type

Display string

Field name

Constraints

descriptionFirstName

```
^ (MAStringDescription auto: 'firstName' label: 'First Name' priority: 20)
  beRequired;
  yourself.
```

descriptionBirthday

```
^ (MADateDescription auto: 'birthday' label: 'Birthday' priority: 70)
  between:(Date year: 1900) and:Datetoday;
  yourself
```



# Magritte

Each domain model has a collection of descriptions

Different visitors are used to

- Generate html to display data

- Generate form to enter the data

- Validate data from form

- Save data in database

# Sample Page

```
editor := (Person new asComponent)
        addValidatedSwitch;
        yourself.
result := self call: editor.
```

## Edit Person

**Title:**

**First Name:**

**Last Name:**

**Home Address:**

**Office Address:**

**Picture:**  no file selected

**Birthday:**

**Age:**

[Kind](#) [Number](#)

**Phone Numbers:** The report is empty.