visitor Paradigm

For Advanced Programming
Paul Cockshott

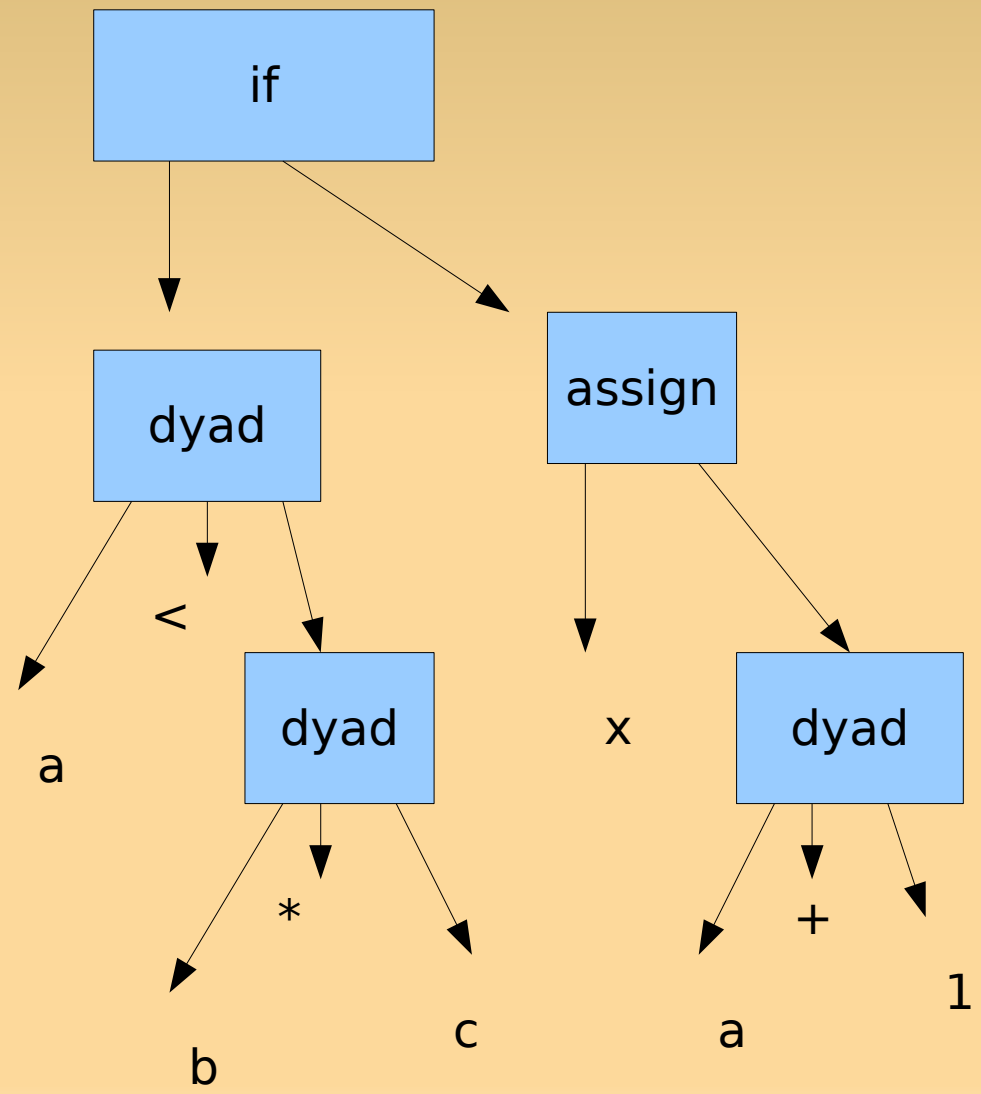
What is it for

- We may have trees which represent some linguistic or other structure
- We may want to Examine this tree
- We may want to Modify this tree

Example of a tree

- We can represent part of a program as a tree

if(a<(b*c))x= a+1;



Node abstract class

This has methods

- **void examine(TreeExaminer e)**

  A method that is used by an examiner to visit all locations.

- **Node modify(TreeModifier m)**

  A method that must be instantiated allowing a TreeModifier to substitute values into the tree.

Dyad

Extends Node has fields

- Node left

  left operand
- Op O

  operator
- Node right

  right operand

Assign extends Node

- Node dest

  Destination of assignment

- Node src

  Source data being assigned

If – extends Node

- Node a1

  action taken if cond is true
- Node a2

  action taken if cond is false
- Node cond

  the condition

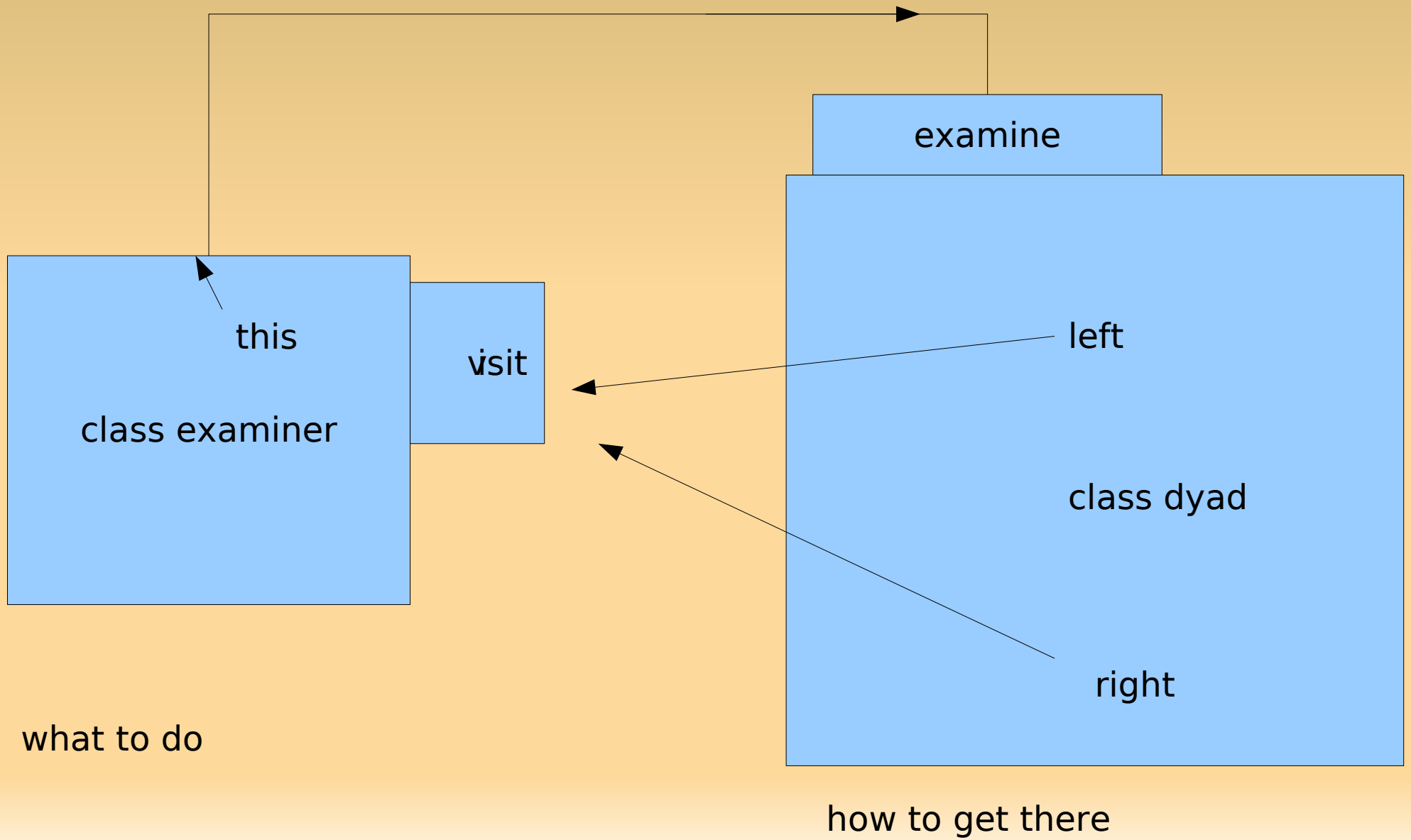
Examine

- Each class of node has to implement an Examine method which is used by TreeExaminer classes to examine the tree
- I will show you the Examine method of the class Dyad

Examine for Dyad

```
public void examine(TreeExaminer e) {
        if (e.visit(this)) {
            if (left != null) {
                left.examine(e);
            }
            if (right != null) {
                right.examine(e);
            }
            if (O != null) {
                O.examine(e);
            }
        }
        e.leave(this);
    }
```

mutual feedback



Examine for class If

```
public void examine(TreeExaminer e) {
        if (e.visit(this)) {
            if (a1 != null) {
                a1.examine(e);
            }
            if (a2 != null) {
                a2.examine(e);
            }
            if (cond != null) {
                cond.examine(e);
            }
        }
        e.leave(this);
    }
```

•

Abstract Class TreeExaminer

- public boolean visit(Node n)

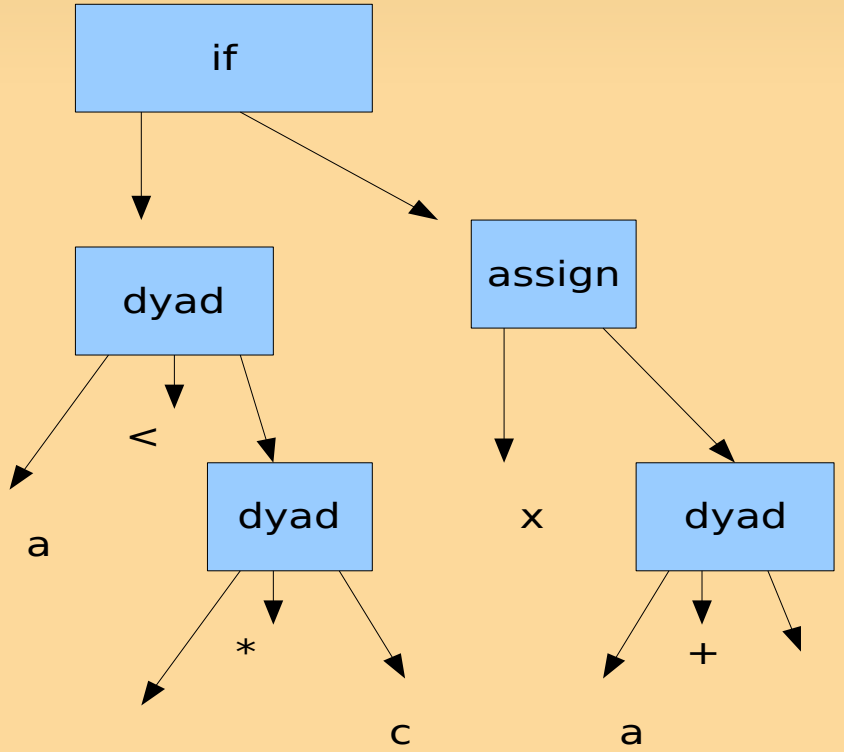
  This is called each time a node is visited, but before any subtrees are visited. If it returns false the subtree below the node is not visited

- public void leave(Node n)

  This is called after all subtrees have been visited

CommonExpressionFinder

- This is a tree visitor that will find all expressions that occur more than once in a tree
- In this tree 'a' occurs twice



leave method

```
public void leave(Node n) {
      String s = n.toString();
       Object o = allexp.get(s);
       if (o == null) {
           int[]count = new int[1];
           count[0] = 1;
           allexp.put(s, count);
       } else {
           int[] freq = (int[]) o;
           freq[0]++;
       }
  }
```

Assume that we have a hashtable called allexp within the ExpressionFinder

we are using a one element array of integers to hold the count

Modify method of Dyad

```
public Node modify(TreeModifier m) {
    if (m.visit(this)) {
        try {
            return new Dyad(m.modified(left), m.modified(right),
                    ((Op) m.modified(O)));
        } catch (Exception ex) {
            System.out.println(ex);
        }
    }
    return this;
 }
```

Abstract class TreeModifier

- Node modified(Node n)

  This returns the rewritten node n

- boolean visit(Node n)

  This is called each time a node is visited, but before any subtrees are visited.

ExpressionSubstituter

- Field Summary
- Node[] A

  array of targets
- Node[] B

  array of replacement values

Modified method

```
public Node modified(Node n ) {
    String oldpad=pad;
    boolean found=false;
    int pos=0;
    Node res=n;
    try{
        for (int i=0;i<Astr.length;i++)
         if(!found)if (eq(n,i)) {found=true;pos=i;}
        if (found) res=B[pos]; else res=n.modify(this);}
    catch(Exception ex)
    { System.out.println("Error in modifying ");}
    return res;
}
```

Visitor Paradigm

- Allow examiners and modifiers to be written that do not know how the tree is structured
- The examiners and modifiers can concentrate on one small task, the visit, leave and modified methods encapsulate the work.

Compare to Iterator/Enumerator

- Collections classes use Iterator/Enumerator classes to return the elements of a collection.
- These are less general than the Visitor Paradigm and are suitable only for collections rather than irregular complex trees.