CSE431 – Translation of Computer Languages

- Visitor Pattern

Doug Shook

Abstract Syntax Trees

- The AST is a bridge between the front end and back end of the compiler
  - Front end responsibilities?
  - Back end responsibilities?

- Later phases will visit nodes of our AST
  - What actions are performed depends on the type of the AST node and the phase of the compiler

Node Class Hierarchy

- Some features are common to all nodes
  - Children, parents, siblings
  - Can put all of these features into a base class
- Specific node types will inherit from base class
  - Will the hierarchy be flat or deep?
    - What are the implications of flat vs. deep?
    - What factors will determine this?

Visitor Pattern

- Consider:
  - Java has about 50 node types
  - GCC has about 200 phases
  - We must be able to handle every node type during each phase
- How should we structure this?
  - Phase code in the nodes?
  - New class for each phase?
  - Possible hierarchies?

Visitor Pattern

- Solution: one class for each phase
  - Visit method for each node type in each phase
- Problem: most languages are *single dispatch*
  - During AST traversal, we refer to all nodes by the base class, AbstractNode (why?)
  - Single dispatch means that the declared type determines which method to invoke
    - What's the problem here?

Double Dispatch

- We want to use the actual type of each node
- Each phase extends the base Visitor class
  - Inherits the visit(AbstractNode) method
- Each node class includes an *accept* method
  - The visitor is passed to this method
  - Calls *visit* on *this*

Double Dispatch

```
class Visitor
    /*    Generic visit                                          */
    procedure VISIT( AbstractNode n )                            (28)
        n.ACCEPT( this )                                         (29)
    end
end
class TypeChecking extends Visitor                               (30)
    procedure VISIT( IfNode i )
    end
    procedure VISIT( PlusNode p )
    end
    procedure VISIT( MinusNode m )
    end
end

class IfNode extends AbstractNode
    procedure ACCEPT( Visitor v )                                (31)
        v.VISIT( this )
    end
    ...
end
```

Reflective Visitor

- Double dispatch still has some disadvantages
  - What are they?
- These issues can be avoided with the use of reflection
  - Ability to examine and modify types, properties, functions, etc. at run time

Reflective Visitor

- Visitor.visit(n) calls Visitor.dispatch()
- Dispatch uses reflection to examine n's type and invoke the appropriate method
  - Exact type
  - Super class
  - Default
- This can be extended to handle interfaces and multiple inheritance

Exercises

```
class IfNode extends AbstractNode
    procedure typeCheck( )
        /*  Type-checking code for an if                    */
    end
    procedure codeGen( )
        /*  Generate code for an if                         */
    end
    ...
end

class PlusNode extends AbstractNode
    procedure typeCheck( )
        /*  Type-checking code for a plus                   */
    end
    procedure codeGen( )
        /*  Generate code for a plus                        */
    end
    ...
end
...
```

- How does this approach compare to double dispatch?
  - Is it better or worse? Explain your answer.