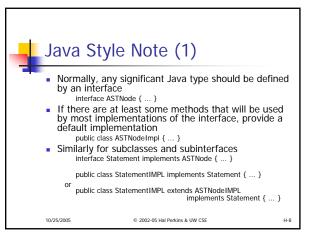


```
More Statement Nodes

// if (exp) stmt [else stmt]
public class IfNode extends StmtNode {
    public ExpNode exp;
    public StmtNode thenStmt, elseStmt;
    public IfNode(ExpNode exp,StmtNode thenStmt,StmtNode elseStmt) {
        this.exp=exp; this.thenStmt=thenStmt;this.elseStmt=elseStmt;
    }
    public IfNode(ExpNode exp, StmtNode thenStmt) {
        this(exp, thenStmt, null);
    }
    public String toString() { ... }
}
```

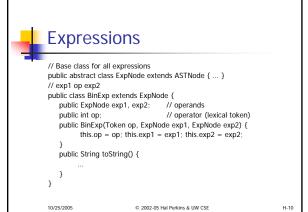


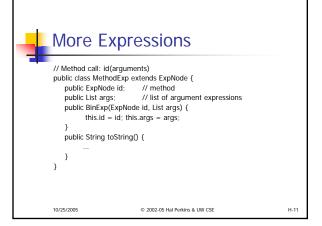


Java Style Note (2)

- Method parameters and variables should use the interface names as types for maximum flexibility wherever possible
- Implementations of nodes can either extend some other class or directly implement an interface as appropriate
- Specific kinds of nodes that will not be extended can be defined directly – no interface needed
- These slides use inheritance only (historical laziness and it's more compact)
 - Exercise: how would you rework the code in the previous examples?

10/25/2005 © 2002-05 Hal Perkins & UW CSE







&c

- These examples are meant to give you some ideas, not necessarily to be used literally
 - E.g., you might find it much better to have a specific AST node for "argument list" that encapsulates the generic java.util.List of arguments
- You'll also need nodes for class and method declarations, parameter lists, and so forth
 - Starter code in book and on web for MiniJava

0/25/2005 © 2002-05 Hal Perkins & UW CSE H-12



Position Information in Nodes

- To produce useful error messages, it's helpful to record the source program location corresponding to a node in that node
 - Most scanner/parser generators have a hook for this, usually storing source position information in tokens
 - Would be nice in our projects, but not required (i.e., get the parser/AST construction working first)

10/25/2005

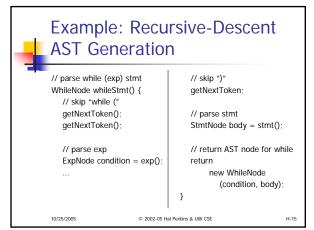
© 2002-05 Hal Perkins & UW CSE



AST Generation

- Idea: each time the parser recognizes a complete production, it produces as its result an AST node (with links any subtrees that are the components of the production in its instance variables)
- When we finish parsing, the result of the goal symbol is the complete AST for the program

/25/2005 © 2002-05 Hal Perkins & UW CSE





AST Generation in YACC/CUP

- A result type can be specified for each item in the grammar specification
- Each parser rule can be annotated with a semantic action, which is just a piece of Java code that returns a value of the result type
 - The semantic action is executed when the rule is reduced

10/25/2005 © 2002-05 Hal Perkins & UW CSE



YACC/CUP Parser Specification

Specification

non terminal StmtNode stmt, whileStmt;
non terminal ExpNode exp;
...
stmt ::= ...
| WHILE LPAREN exp:e RPAREN stmt:s
{: RESULT = new WhileNode(e,s); :}
;

10/25/2005

© 2002-05 Hal Perkins & UW CSE



SableCC/JavaCC/others

- Integrated tools like these provide tools to generate syntax trees automatically
 - Advantage: saves work, don't need to define AST classes and write semantic actions
 - Disadvantage: generated trees might not have the right level of abstraction for what we are trying to do

10/25/2005

© 2002-05 Hal Perkins & UW CSE



Operations on ASTs

- Once we have the AST, we may want to
 - Print a readable dump of the tree (pretty printing)
 - Do static semantic analysis
 - Type checking
 - Verify that things are declared and initialized properly
 - Etc. etc. etc. etc.
 - Perform optimizing transformations on the tree
 - Generate code from the tree, or
 - Generate another IR from the tree for further processing (maybe flatten to a linear IR)

© 2002-05 Hal Porkins & HW CSF

Where do the Operations Go?

- Pure "object-oriented" style
 - Really smart AST nodes
 - Each node knows how to perform every operation on itself

```
public class WhileNode extends StmtNode {
 public WhileNode(...);
  public typeCheck(...):
 public StrengthReductionOptimize(...);
 public generateCode(...);
 public prettyPrint(...);
```

10/25/2005 © 2002-05 Hal Perkins & LIW CSF



Critique

- This is nicely encapsulated all details about a WhileNode are hidden in that class
- But it is poor modularity
- What happens if we want to add a new Optimize operation?
 - Have to open up every node class
- Furthermore, it means that the details of any particular operation (optimization, type checking) are scattered across the node classes

10/25/2005

© 2002-05 Hal Perkins & UW CSE



Modularity Issues

- Smart nodes make sense if the set of operations is relatively fixed, but we expect to need flexibility to add new kinds of nodes
- Example: graphics system
 - Operations: draw, move, iconify, highlight
 - Objects: textbox, scrollbar, canvas, menu. dialog box, plus new objects defined as the system evolves

10/25/2005 © 2002-05 Hal Perkins & UW CSE



Modularity in a Compiler

- Abstract syntax does not change frequently over time
 - ∴ Kinds of nodes are relatively fixed
- As a compiler evolves, it is common to modify or add operations on the AST nodes
 - Want to modularize each operation (type check, optimize, code gen) so its components are together
 - Want to avoid having to change node classes to modify or add an operation on the tree

© 2002-05 Hal Perkins & UW CSE



H-21

Two Views of Modularity

 Binop	if	while	ехр	IDENT	
Х	Х	Х	Χ	Х	Type check
Х	Х	Х	х	Х	Optimize
Х	Х	Х	Х	Х	Generate x86
Х	Х	Х	Х	Х	Flatten
Х	Х	Х	Χ	Х	Print

circle text canvas scroll dialog H-22

10/25/2005

© 2002-05 Hal Perkins & UW CSE



Visitor Pattern

- Idea: Package each operation in a separate class
 - One method for each AST node kind
- Create one instance of this visitor class
 - Sometimes called a "function object"
- Include a generic "accept visitor" method in every node class
- To perform the operation, pass the visitor object around the AST during a traversal

© 2002-05 Hal Porkins & HW CSF



Avoiding instanceof

- Next issue: we'd like to avoid huge if-elseif nests to check the node type in the visitor void checkTypes(ASTNode p) { if (p instanceof WhileNode) { ... }
 - else if (p instanceof IfNode) { ... } else if (p instanceof BinExp) { ... } ...
- Solution: Include an overloaded "visit" method for each node type and get the node to call back to the correct operation for that node(!)
 - "Double dispatch"

© 2002-05 Hal Perkins & LIW CSF



One More Issue

- We want to be able to add new operations easily, so the nodes shouldn't know anything specific about the actual visitor class
- Solution: an abstract Visitor interface
 - AST nodes include "accept visitor" method for the interface
 - Specific operations (type check, code gen) are implementations of this interface

10/25/2005

© 2002-05 Hal Perkins & UW CSE



Visitor Interface

```
interface Visitor {
  // overload visit for each node type
  public void visit(WhileNode s);
  public void visit(IfNode s);
  public void visit(BinExp e);
}
```

Aside: The result type can be whatever is convenient, not necessarily void

10/25/2005

© 2002-05 Hal Perkins & UW CSE

H-28



Specific class TypeCheckVisitor

```
// Perform type checks on the AST
public class TypeCheckVisitor implements Visitor {
  // override operations for each node type
  public void visit(WhileNode s) { ... }
  public void visit(IfNode s) { ... }
  public void visit(BinExp e) {
     e.exp1.accept(this); e.exp2.accept(this);
```

© 2002-05 Hal Perkins & UW CSE

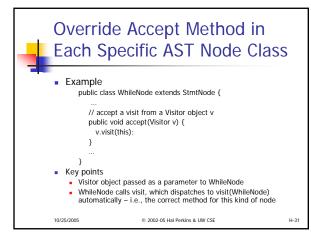


Add New Visitor Method to **AST Nodes**

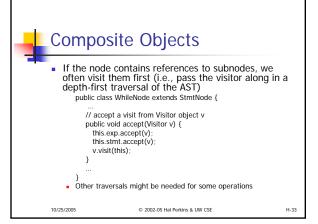
Add a new method to class ASTNode (base class or interface describing all AST nodes)

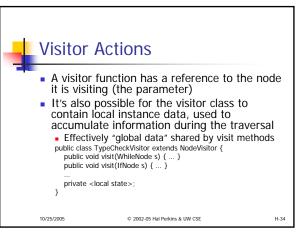
```
public abstract class ASTNode {
   // accept a visit from a Visitor object v
  public abstract void accept(Visitor v);
```

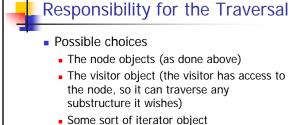
10/25/2005 © 2002-05 Hal Perkins & UW CSE











 In a compiler, the first choice will handle many common cases

10/25/2005 © 2002-05 Hal Perkins & UW CSE H-3



References

- For Visitor pattern (and many others)
 Design Patterns: Elements of Reusable
 Object-Oriented Software
 Gamma, Helm, Johnson, and Vlissides
 Addison-Wesley, 1995
- Specific information for MiniJava AST and visitors in the textbook

10/25/2005 © 2002-05 Hal Perkins & UW CSE H-36



Coming Attractions

- Static Analysis
 - Type checking & representation of types
 - Non-context-free rules (variables and types must be declared, etc.)
- Symbol Tables
- & more

10/25/2005

© 2002-05 Hal Perkins & UW CSE