



Abstract Syntax Trees

- The parser's output is an abstract syntax tree (AST) representing the grammatical structure of the parsed input
- ASTs represent only semantically meaningful aspects of input program, unlike concrete syntax trees which record the complete textual form of the input
 - There's no need to record keywords or punctuation like (), ;, else
 - The rest of compiler only cares about the abstract structure

1/27/2009 © 2002-09 Hal Perkins & UW CSE E1-



MiniJava AST Node Classes

Each node in an AST is an instance of an AST class

IfStmt, AssignStmt, AddExpr, VarDecl, etc.

Each AST class declares its own instance variables holding its AST subtrees

- IfStmt has testExpr, thenStmt, and elseStmt
- AssignStmt has lhsVar and rhsExpr
- AddExpr has arg1Expr and arg2Expr
- VarDecl has typeExpr and varName

CSE401 Wi09



AST Class Hierarchy

- AST classes are organized into an inheritance hierarchy based on commonalities of meaning and structure
- Each "abstract non-terminal" that has multiple alternative concrete forms will have an abstract class that's the superclass of the various alternative forms
 - Stmt is abstract superclass of IfStmt, AssignStmt, etc.
 - Expr is abstract superclass of AddExpr, VarExpr, etc.
 - Type is abstract superclass of IntType, ClassType, etc.

CSE401 Wi09



AST Extensions For Project

- New variable declarations:
 - StaticVarDecl
- New types:
- DoubleTypeArrayType
- New/changed
- statements:

 IfStmt can omit else
- branch
- ForStmtBreakStmt
- ArrayAssignStmt

CSE401 Wi09



DoubleLiteralExpr

ArrayLookupExpr

ArrayLengthExpr

ArrayNewExpr

OrExpr

Automatic Parser Generation in MiniJava

- We use the CUP tool to automatically create a parser from a specification file, Parser/minijava.cup
- The MiniJava Makefile automatically rebuilds the parser whenever its specification file changes
- A CUP file has several sections:
 - introductory declarations included with the generated parser
 - declarations of the terminals and nonterminals with their types
 - The AST node or other value returned when finished parsing that nonterminal or terminal
 - precedence declarations
 - productions + actions

CSE401 Wi09

CSE 401 Wi09 E1-1



Terminal Declarations

Terminal declarations we saw before:





Java Generics and MiniJava

- CUP did not support Java generics when the MiniJava starter code was written, so there are some hacks
- An example: we'd like to write nonterminal List<Expr> Exprs;
 but instead the code has
- nonterminal List/*<Expr>*/ Exprs;
 There are other hacks. Deal with them as
- gracefully as you can

 Don't make pointless changes to the code save your energy for more interesting things

your chargy for more interesting things

1/27/2009 © 2002-09 Hal Perkins & UW CSE E1



Precedence Declarations

- Can specify precedence and associativity of operators
 - equal precedence in a single declaration
 lowest precedence textually first
 - specify left, right, or nonassoc with each declaration
- Examples:

CSEARL WING

CSE401 W/09



Productions

All of the form:

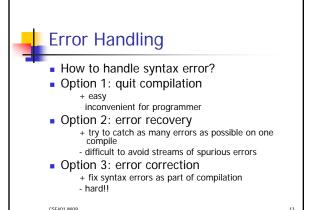
- Can label symbols in RHS with :var suffix to refer to its result value in Java code
 - varleft is set to line in input where var symbol was

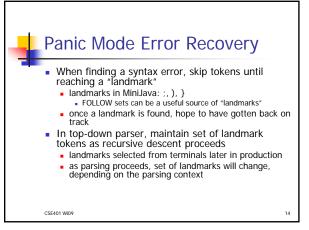
CSE401 Wi09

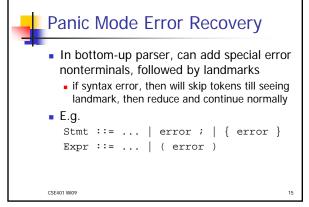
Productions (cont.)

Example

CSE 401 Wi09 E1-2







```
EBNF Syntax of initial MiniJava
Program
             ::= MainClassDecl { ClassDecl }
MainClassDecl ::= class ID {
                 public static void main
                 ( String [ ] ID ) { { Stmt } }
ClassDecl
             ::= class ID [ extends ID ] {
                 { ClassVarDecl } { MethodDecl } }
ClassVarDecl ::= Type ID ;
            ::= public Type ID
MethodDecl
                ( [ Formal { , Formal } ] )
                 { { Stmt } return Expr ; }
Formal
             ::= Type ID
Type
             ::= int |boolean | ID
  CSF401 Wi09
```

CSE 401 Wi09 E1-3