MeggyJr Java
PA4
Symbol Tables

Promoting Bytes to Ints

- Bytes need to be promoted to integers when …
  - –Adding or subtracting a byte to or from an integer
  - –Result of an addition or a subtraction should be promoted
  - –Equality comparison between a byte and an int
  - –Less than comparison between a byte and an int
  - –Passing a byte as an argument when the formal is an int
- Doing the promotion with sign extension (AVR code)

Possible Helper Routines

- promoteValue
- genExpLoad
- genVarLoad

Identifying the Scoping Rules of a PL

Using the full MeggyJava grammar,

1. List how new names are introduced:
   - Class names
   - Method names
   - Formal parameter names
   - Class member variable names
   - Method local variable names

2. Where are existing names used/accessed:
   - Expressions:
   - New object creations
   - Method calls
   - Actual parameters
   - Other expressions

3. List rules for visibility of names (Scoping) rules: static scoping.

What features does PA4 add?

- Meggy.toneStart
- < operator
- User-defined methods
- Parameters (formals and actuals)
- Method calls

Notice: still no local or class variables, assignments, or arrays, or objects (though syntax is included for object creation)

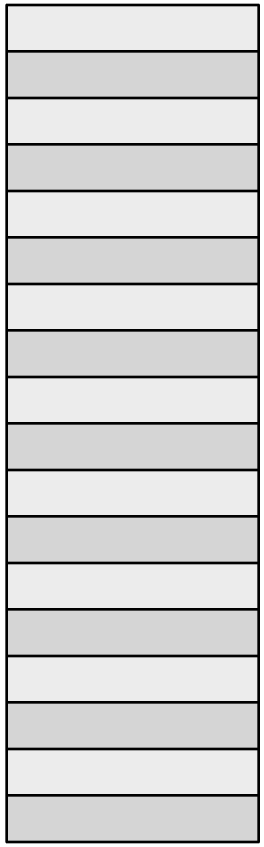
Symbol Table

- Information maintained in a symbol table - environment
  - For each identifier: type, scope (nesting level), string and run-time location
  - For named scopes, the set of identifiers it contains.
  - While processing the program, the current symbol table is used first

- Scoping in MeggyJava
  - Do we have unnamed scopes in MeggyJava? <look at grammar>
  - What kinds of scopes do we have? <look at PA4raindrop.java>

Runtime Stack in MeggyJava (with recursion)

```
class PA4raindrop {
public static void main(String[] whatever){
   while (true) {
     new Cloud().rain((byte)3,(byte)7); …}   }}
class Cloud {
   public void rain(byte x, byte y) {
  // light up x,y if is in bounds and continue recursion
   if (this.inBounds(x, y)) {
      Meggy.setPixel(x, y, Meggy.Color.BLUE)
      if (this.inBounds(x,(byte)(y+(byte)1))) {
         Meggy.setPixel(x, (byte)(y+(byte)1), Meggy.Color.DARK);
         } else {}
     Meggy.delay(100);
     this.rain(x, (byte)(y-(byte)1));
    } else {}  }
public boolean inBounds(byte x, byte y) {
   return ((byte)(0-1) < y) && (y < (byte)8);  }
}
```



Requirements for Symbol Table
Mapping an expression node in AST to type

What information does the symbol table already maintain?
[mapping of expression nodes in the AST to type]

Where is this information currently computed?
[CheckTypes visitor]

Where and what kinds of information do we need in PA4 for code generation

CallExp and CallStmt nodes: formal parameter types; return type for CallExp

MethodDecl node
Prologue: how many formals, types
Epilogue – return type

IdExp node – know where id is stored on run-time stack and its type

Strategy for PA4

- Visitor that builds a symbol table with …
  - Type information about expressions
  - Type and location information about parameters
  - Type signature information about methods: return type, number formals, types of formals
  - The symbol table should be able to keep track of what scope we are in.

- Visitor that does type checking

- Visitor that does code generation

Symbol Table Interface Needed for PA4

- STE lookup(methodID)
  - Returns the STE for the method. The STE for the method includes the type signature for the method (parameter types and return type).
  - <What would STEs for the methods in PA4raindrop.java have?>

- STE lookup(parameterID)
  - The STE for the parameter needs to know the type and location for that parameter.
  - The location is the base (e.g. “Y”) and the offset (e.g. +1)

What do you need to do to handle these new features?

1. Copy your working PA3 compiler to a separate folder.
2. Write test cases for new features of PA4, one feature per test case.
3. Add new PA4 grammar rules to the JavaCup file, and test the grammar additions incrementally.
4. Add the actions to build the corresponding AST parts for those rules and test tree building

Now, you have correct PA ASTs, Now what? Symbol Table Building!!

- Extend SymTable package from PA3 so PA4 compiler keeps track of:
  - For each method, INSERT: type signature, formal parameters and their types

  (MethodDecl nodes)

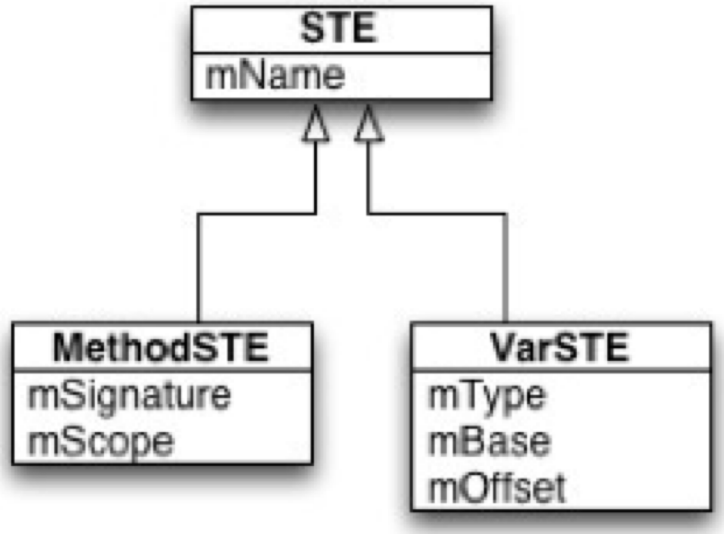
  -> At each call site, LOOK UP type information for methods, parameters, and expressions and storage location for parameters

  (CallExp and CallStmt nodes)

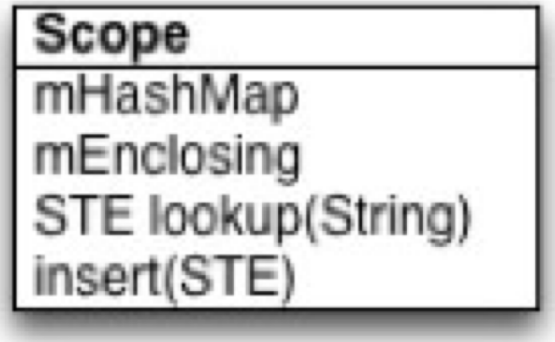
  -> At each access to actual parameter, LOOK UP type information and storage location (IdLiteral)

PA4 Symbol Table
Entries and Scopes

Symbol Table Entry Classes



Scope Class for a single scope, one for each Scope will be created and linked together to make a symbol table.

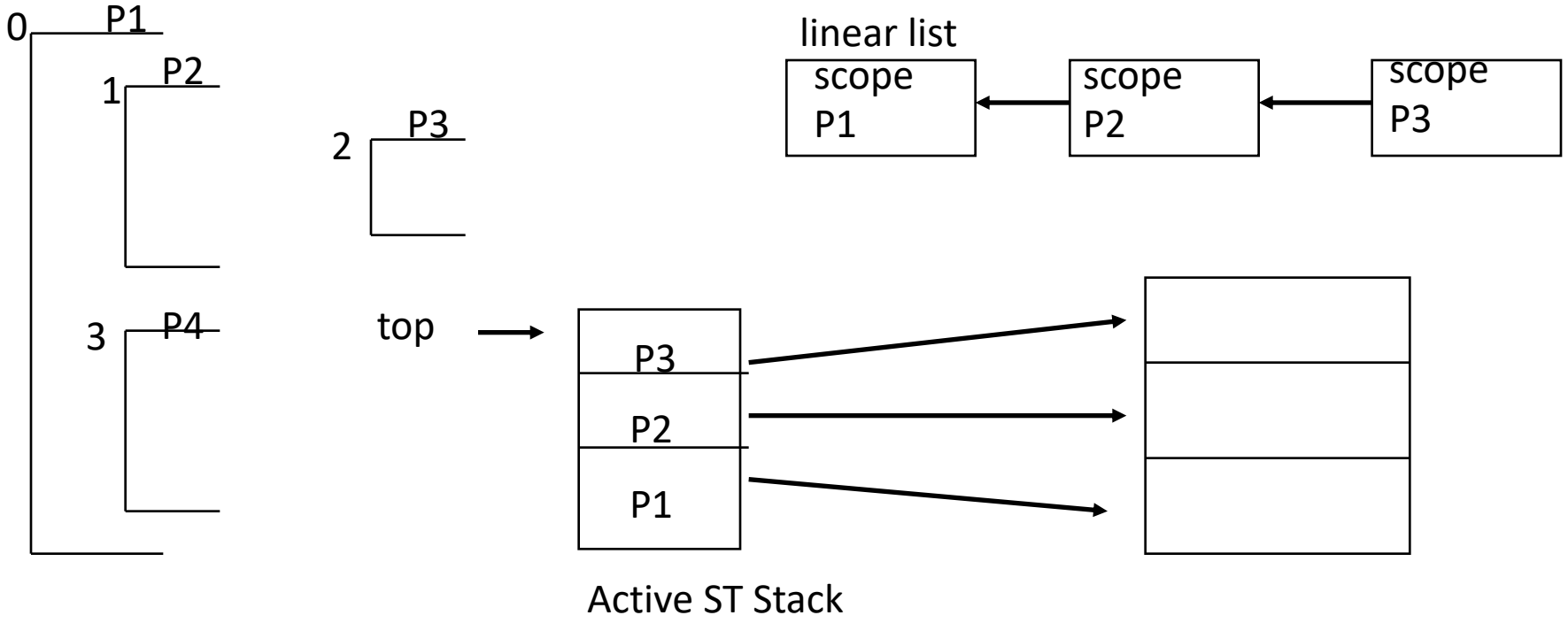


VarSTE, variable symbol table entry

- type
- base, string for base register "Y", later will need another one
- offset, number or string for offset from base register

MethodSTE

The method symbol table entry contains a reference to signature information and to the method's scope.



During AST Visitor, we have an Active Symbol Table Stack and current ST pointer

SymTable Operations

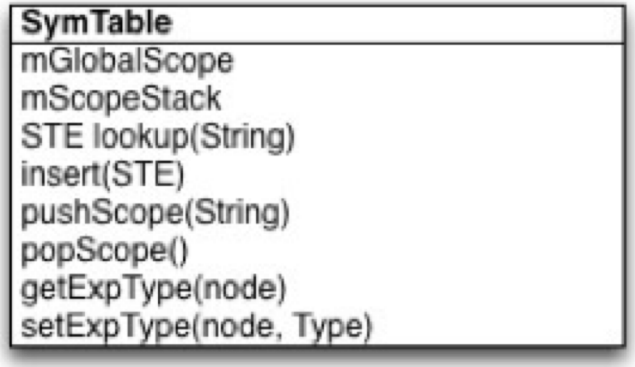
SymTable Class: A stack of scopes with current most deeply nested scope at top of stack
And a reference to the outermost (or global) scope.

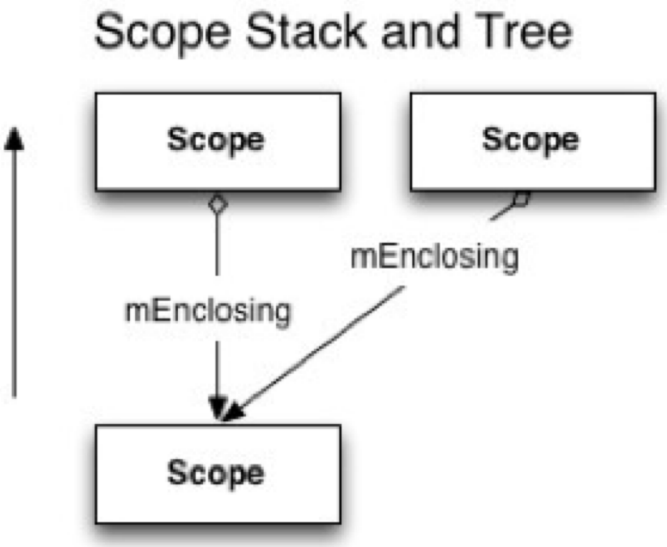
STE lookup(String) - lookup in most nested

void insert(STE) - Insert STE into most deepl Scope.

void pushScope(String)
- Look up a named scope like a method and then push its scope on stack.

void popScope()
- Pop top scope off stack.





What are the steps at inMethodDecl during BuildSymTable visitor?

inMethodDecl:

(1) Look up method name in current symbol table to see if there are any duplicates. Generate an error if needed.

(2) create a function signature object of some kind

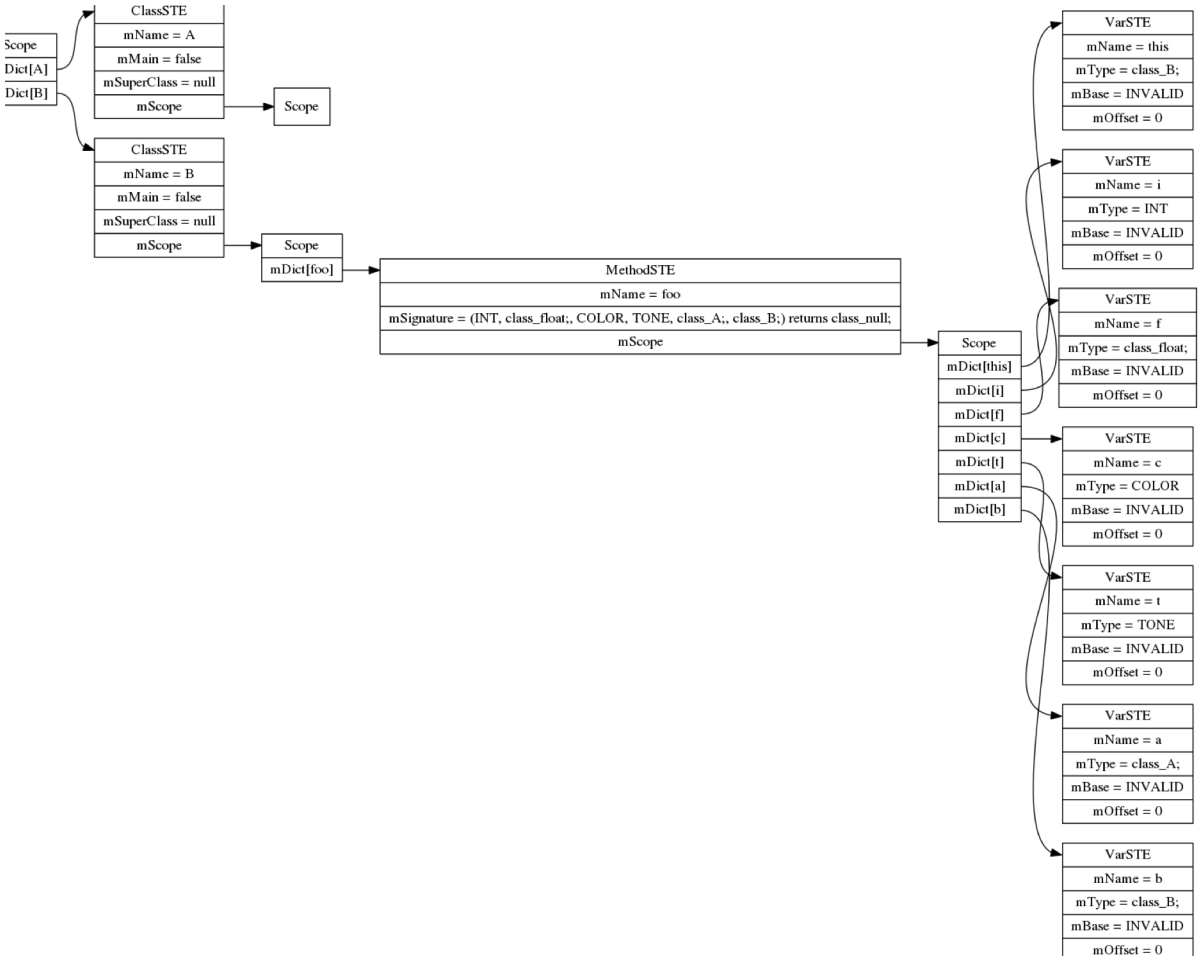
(3) create a MethodSTE

(4) insert the MethodSTE into the symbol table with SymTable.insert

Example

- **Function with parameters**
- import meggy.Meggy;
-
- class problem2 {
- public static void main(String[] whatever){
- }
- }
-
- class A {
- }
-
- class B {
- public void foo(int i, float f, Meggy.Color c, Meggy.Tone t, A a, B b) {
- }

Symbol table



How about parameters?

Insert of formal parameter into SymTable:

inMethodDecl

(1) after creating MethodSTE and inserting it (see above), then call pushScope(methodname) on the symbol table being built

(2) set current offset in visitor to 1

How about parameters?

outFormal:

(1) check if var name has already been inserted in SymTable using st.lookup(name). Error if there is a duplicate.

(2) create VarSTE with current method offset and type of formal

(3) increment visitor-maintained offset based on the type of the formal variable

(4) call st.insert

outMethodDecl:

(1) Store the number of bytes needed for parameters as size of the method.

We will see more about code generation and stack frame allocation next time.