Compiling Classes and Variables

Adding Classes

- Type representation for classes
- Class scopes in symbol table
- Dynamic allocation for an object
- References with “this” in expressions & parameters
- Definitions (assignments) and uses of class member variables

PA 5 Adds…

-Extend the symbol table for local and member variables
-Implement equality comparison for class references
-Perform semantic analysis to find ALL redeclared variables
-Implement new Type class to represent class references
-Extend type checking for member variables, locals, and assignment
-Code generation for local variable uses and assignments, "new" expression, equality comparison of class refs variables, object refs

PA5 Does NOT add

- Subclasses and Inheritance
- Polymorphic call sites

Symbol Table with Classes What are the contents?

- Program global scope:
- A given class scope:
- A given method scope:

While building symbol table:

- current scope
- manage scope stack: push, pop when?

Recall Using a Symbol Table

- To process a portion of the program that creates a scope (block statements, function declaration, classes, etc.)
  - Enter a new scope
  - Add all variable declarations to the symbol table
  - Process the body of the block/function/class
  - Exit the scope

Building the Symbol Table: BuildSymTable Visitor

The BuildSymTable constructor initializes the symbol table

```
mCurrentST = new symtable.SymTable();
```

inTopClass: create ClassSTE, insert it and pushScope

outTopClass: popScope

What happens inMethod ?

out Method: ?

Building the Symbol Table: BuildSymTable Visitor

The BuildSymTable constructor initializes the symbol table

```
mCurrentST = new symtable.SymTable();
```

inTopClass: create ClassSTE, insert it and pushScope

outTopClass: popScope

What happens inMethod ?

- create a list of formal types
- create a signature: formalTypes + node type (=return type)
- create a methodSTE with class&method name, node, signature
- insert it and pushScope
- create an entry for “this”

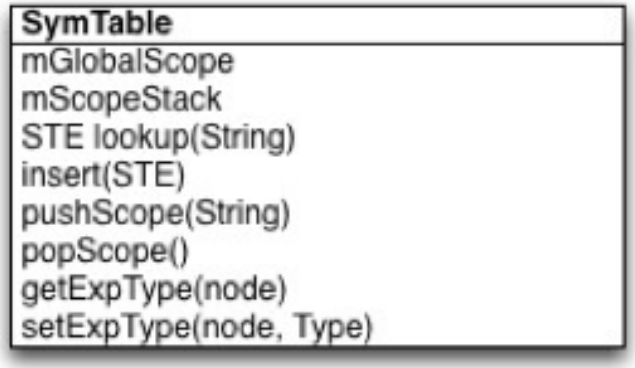
out Method: ? popScope

Building the Symbol Table: BuildSymTable Visitor

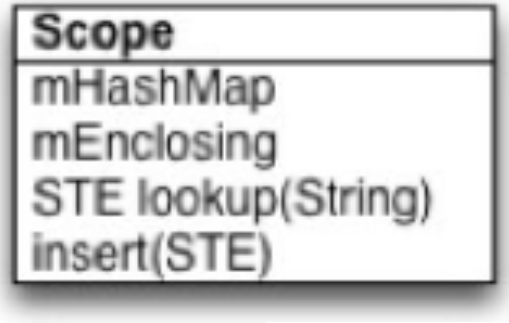
outVarDecl: ?

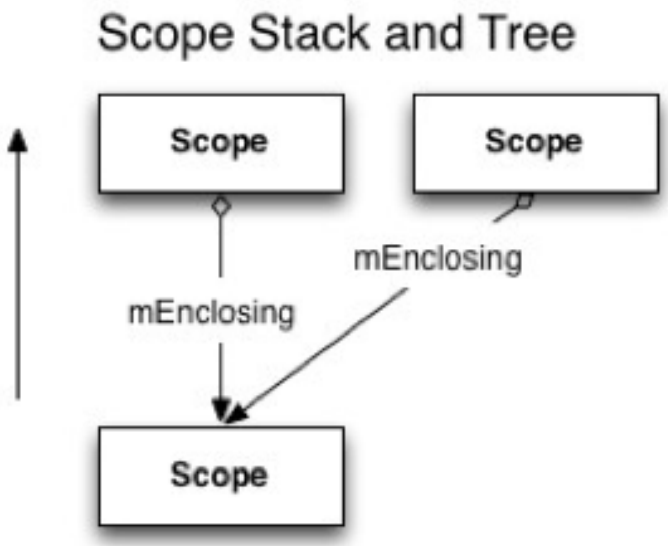
- determine whether a local or a formal (How?)
- create a VarSTE with type, base, and offset information
- insert VarSTE into the symbol table

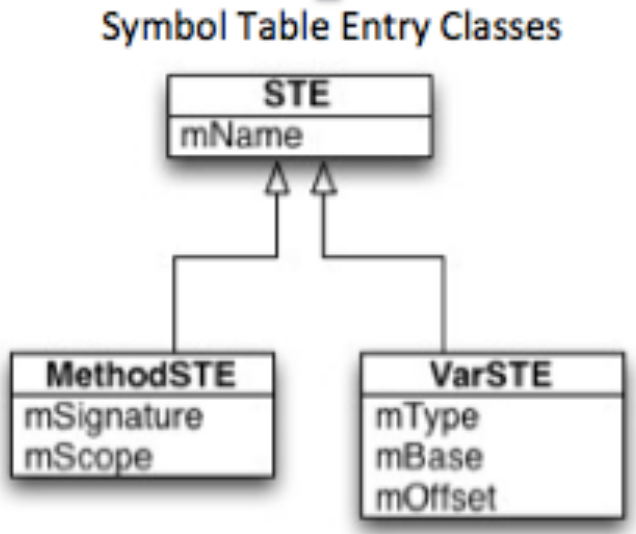
We need Class scope in Symbol Table



Recall the Symbol table data structure without Class scope:



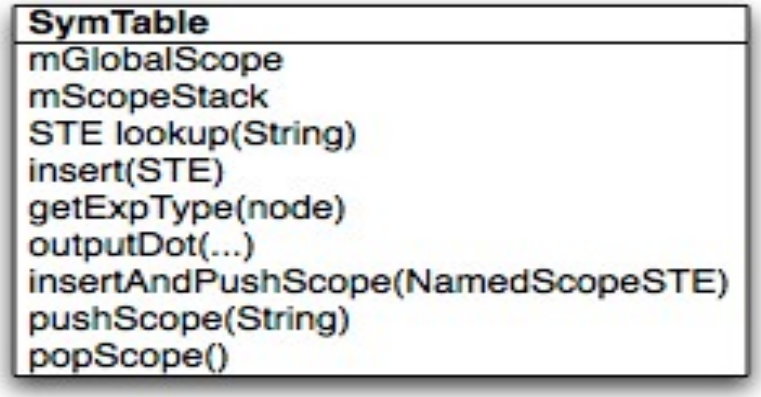


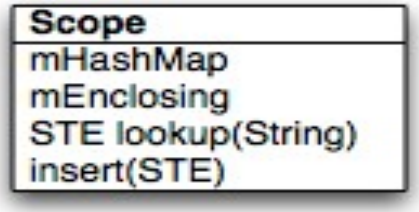


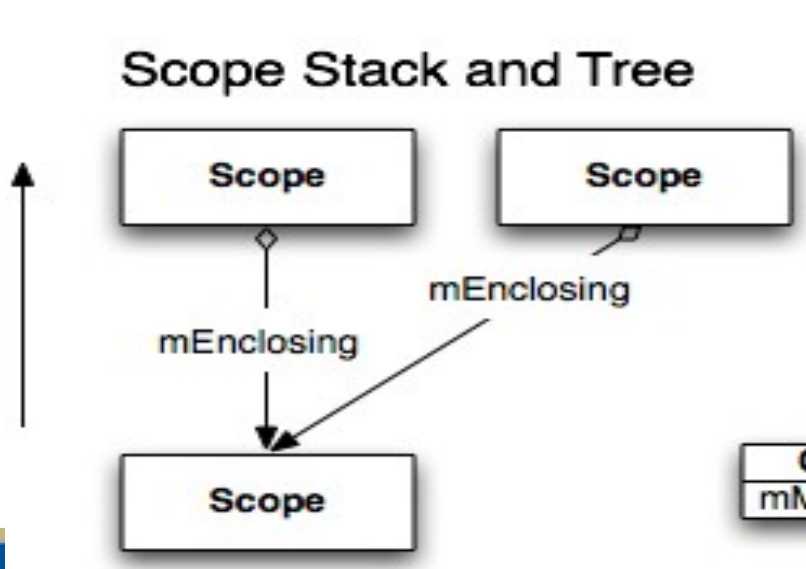
Introducing NamedScopeSTE

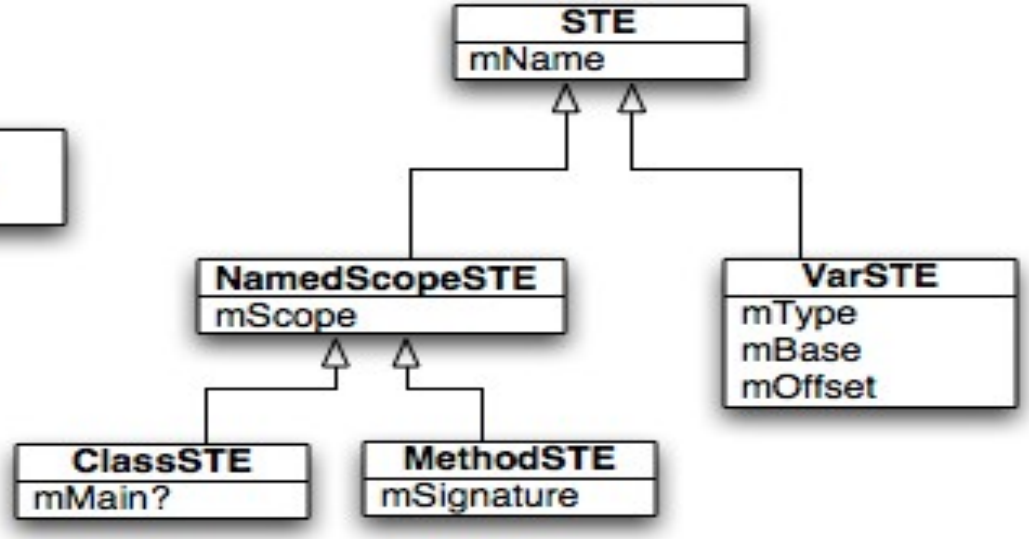
- Extend STE;
- Scope for named scope (class or method)

Symbol Table Data Structures



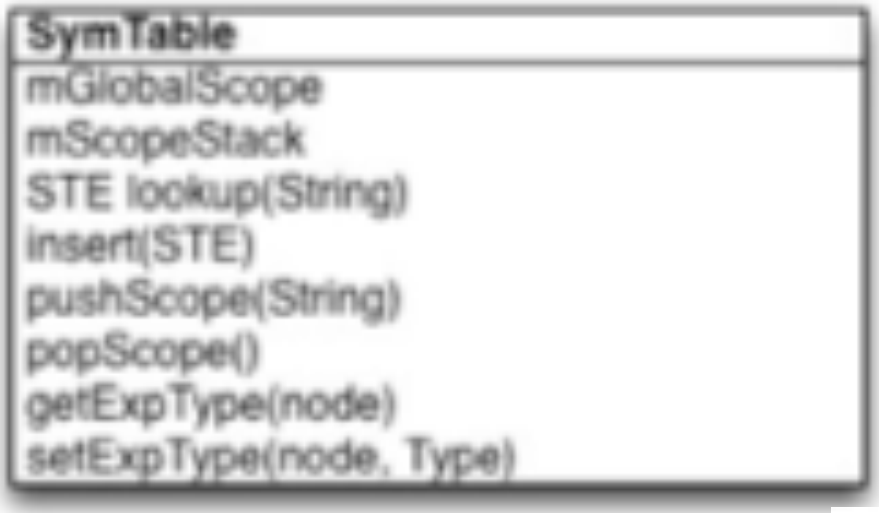


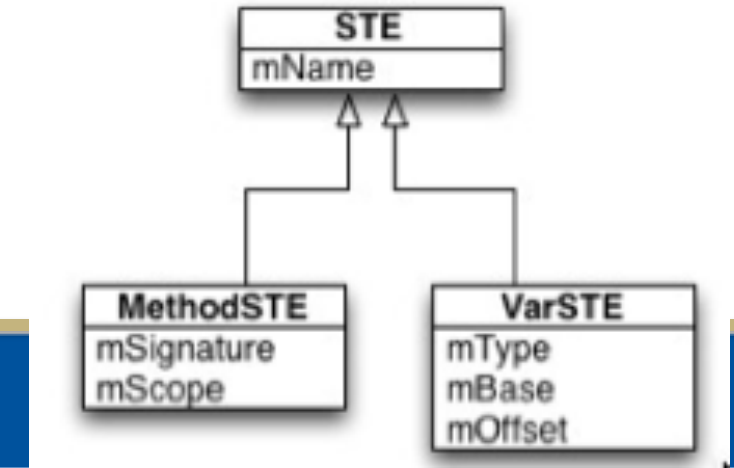




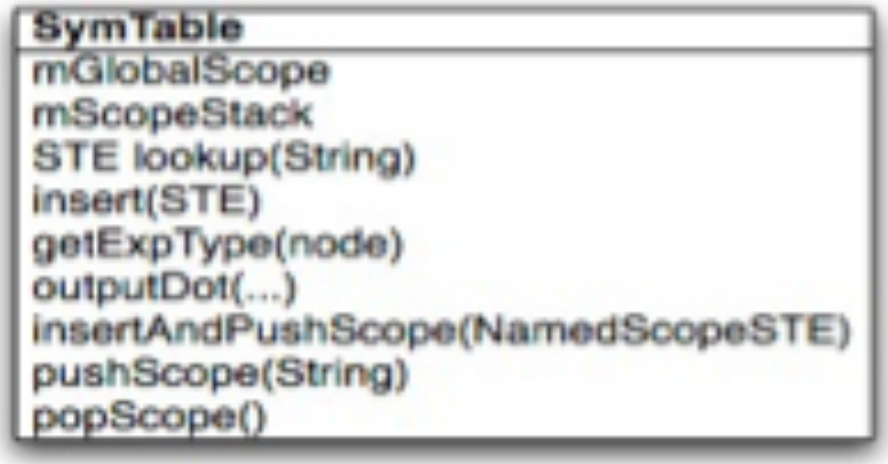
SymTable Changes

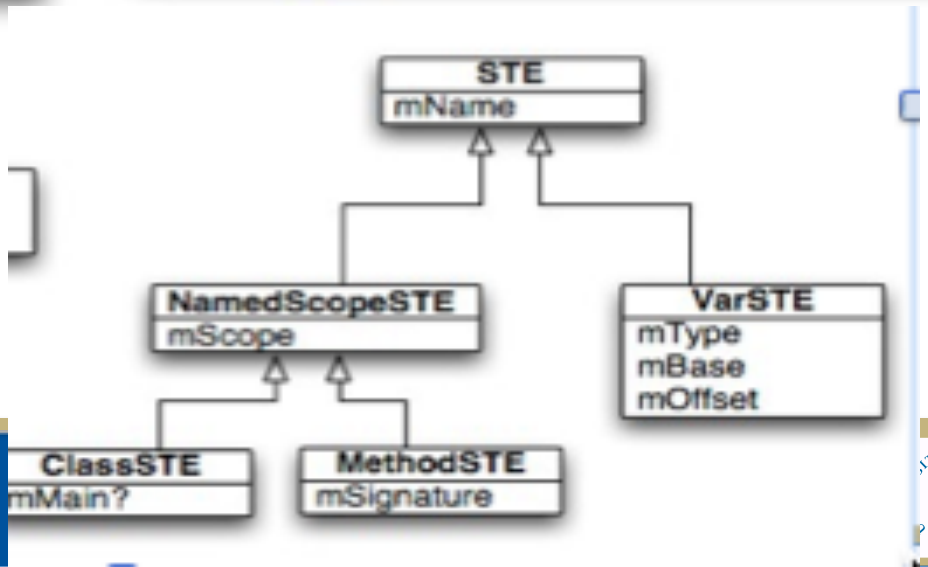
Before





After





Class ClassSTE

Boolean **mMain** - is this the main class?

String **mSuperClass** - super class, null if none

Int **mOffset** – keep track of offset where next field would be put

Int **mNumMembers** - # of member variables

Exercise: Draw the Symbol Table

```
class C {
    int i; C c;

 public int foo( boolean p ) { B b;
  // type correctness of below?
  p = true;
  c = this;
  b = new B();
  c = new B(); i = x;
 }
}
class B { }
```

Type Checking with Classes

What did you do for InMethodDecl? OutMethodDecl?

inMethodDecl(MethodDecl node):

push a new scope: mCurrentST.pushScope(node.getName());

outMethodDecl(MethodDecl node): look up

method in class scope: MethodSTE

methodSTE =

(MethodSTE)mCurrentST.lookup(node.getName());

check that it is not defined already in this class scope (no overloading)

check that the return type in the signature methodSTE conforms with the type of the return expression

Type Checking with Classes

So, what do we need to do inTopClassDecl?

inTopClassDecl:

Create an instance variable of type classSTE
Set the name of the class
Use new classSTE for setting type of "this"
Push new scope:

```
mCurrentST.Class pushScope (node.getName);
```

Type Check a call: receiver.funcName(args)

typeCheck(receiver, funcName, args):

- check that receiver is of type Class

How should we get the receiverType?

receiverInfo = lookupClass(receiverType.getClassName())

invocationInfo = receiverInfo.getScope().lookup(funcName);

- if it isn't there, throw exception

Type Check receiver.funcName(args)

Were already doing the following:

- get the Signature of the invoked method
- check argument count
- for each actual argument,
  type check that it is equal to the formal type
  (no widening when argument passing)
- the type of the call is the return type from the signature

Code Generation for Classes

- Object Layout and generating code for New
- Inheritance:
  - Object layouts with inheritance
  - Dynamic versus static type
  - Dynamic dispatch
    - Dispatch/virtual/V tables

Dynamically Allocating Objects

outNewExp:

1) push size of class instance onto stack

```
ldi    r24, lo8(OBJSIZEINBYTES)
ldi    r25, hi8(OBJSIZEINBYTES)
```

2) allocate that space on the heap

```
call    malloc
```

3) push return from malloc on stack

```
push    r25
push    r24
```

Let's look at the Stack + Heap

```
class PA5length {
    public static void main(String[] whatever){
            new Lengthy().run();
    }}
class Lengthy {
    int field;
    byte another;
    public void run() {
        another = (byte)2;
        Meggy.setPixel((byte)2, (byte)this.createA
                    this.createArray()[2]);
        Meggy.setPixel((byte)3, (byte)this.createArray().length,
                    this.createArray()[1]);
    }
    public Meggy.Color [] createArray() {
        Meggy.Color [] retval;
    …}
```

```
# Load constant int 3
    ldi   r24,lo8(3)
    ldi   r25,hi8(3)
#allocate object of size 3
    call malloc
# push two byte expression
    onto stack
      push  r25
      push  r24
```

```
# loading the implicit "this”
  # load a two byte variable from base+offset
  ldd    r31, Y + 2
  ldd    r30, Y + 1
  # push two byte expression onto stack
  push   r31
  push   r30
  #### function call
  # put parameter values into appropriate registers
  # receiver will be passed as first param
  # load a two byte expression off stack
  pop    r24
  pop    r25
  call   Lengthy_createArray
  # handle return value
  # push two byte expression onto stack
  push   r25
  push   r24
```

Code Generation with Inheritance

- Object layouts
- Dynamic versus static type
- Dynamic dispatch
  - Dispatch/virtual/V tables

Inheritance Graphs

```
Class A {
  a: Int <- 0;
  d: Int <- 1;
  f(): Int {a <- a+d};
}
Class B extends A {
  b: Int <- 2;
  f(): Int { a } ;
  g(): Int {a <- a – b};
};
```

```
Class C extends A {
   c: Int <- 3;
   h(): Int {a <- a * c};
 }
```

Object Layout

```
Class A {
  a: Int <- 0;
  d: Int <- 1;
  f(): Int (a <- a+d};
}
Class B extends A {
  b: Int <- 2;
  f(): Int { a } ;
  g(): Int {a <- a – b};
};
```

```
Class C extends A {
   c: Int <- 3;
   h(): Int {a <- a * c};
 }
```

Class Tag: ids the class	0
Object size	4
Dispatch table pointer	8
Attribute 1	12
Attribute 2	16
…	…

Object Layout

- Each attribute member is offset from base of allocated object.
- Why is object laid out like this?

Exercise:

Create an example set of classes with inheritance and some attributes in each class.

Trade with another group.

Draw the inheritance graph and show object layouts.

Static vs Dynamic Type of Object

```
public class TestPoly2 {
public static void main(String [] args) {
    Student [] all= new Student[3];
    all[0]= new Student("kate");
    all[1]= new MScStudent("mike");
    all[2]= new Student("Jane");
    for (int i=0;i<3;i++)
     System.out.println(all[i].toString());
} }
```

Array all -
Static type:
Dynamic Type:

Polymorphic call Site?

Dynamic versus Static Types

```
Class A {
  a: Int <- 0;
  d: Int <- 1;
  f(): Int (a <- a+d};
}
Class B extends A {
  b: Int <- 2;
  f(): Int { a } ;
  g(): Int {a <- a – b};
};
```

```
Class C extends A {
   c: Int <- 3;
   h(): Int {a <- a * c};
 }
```

Which methods are called?

e.g()

y.h()

x.f()