# Compiladores
(IF688)

**Leopoldo Teixeira**
lmt@cin.ufpe.br | @leopoldomt

# Checando Escopo

Um nome pode ter
diferentes significados
em um mesmo programa

# Código Java “válido”

*não quer dizer que seja recomendável…*

```
public class Name {

   int Name;

    Name Name(Name Name) {
         Name.Name = 137;
         return Name((Name) Name);
    }

}
```

**Adaptado de** Keith Schwartz

# Código Java “válido”

*não quer dizer que seja recomendável…*

```
public class Name {

   int Name;

   Name Name(Name Name) {
        Name.Name = 137;
        return Name((Name) Name);
    }

}
```

**Adaptado de** Keith Schwartz

# Código C++ “válido”

*não quer dizer que seja recomendável…*

```
int Awful() {
   int x = 137;
   {
      string x = “Scope!";
      if (float x = 0)
          double x = x;
   }
   if (x == 137) cout << "Y";
}
```

**Adaptado de** Keith Schwartz

# Código C++ “válido”

*não quer dizer que seja recomendável…*

```
int Awful() {
   int x = 137;
   {
      string x = “Scope!";
      if (float x = 0)
         double x = x;
   }
   if (x == 137) cout << "Y";
}
```

**Adaptado de** Keith Schwartz

# Escopo

- O escopo de uma entidade é o conjunto de pontos em um programa onde o nome da entidade se refere a si mesma
- A introdução de novas variáveis pode “esconder” variáveis existentes
- Como manter o registro do que é visível?

# Tabelas de Símbolos

- Mapeamento de nomes para a entidade a que o nome se refere
- Na medida que rodamos análise semântica, continuamente atualizamos a tabela com informações sobre o escopo
- O que utilizar na prática para implementar?
- Que operações precisam ser definidas?

```
0: int x = 137;
1: int z = 42;
2: int MyFunction(int x, int y) {
3:  printf("%d,%d,%d\n", x, y, z);
4:  {
5:   int x, z;
6:   z = y;
7:   x = z;
8:   {
9:    int y = x;
10:   {
11:    printf("%d,%d,%d\n", x, y, z);
12:   }
13:   printf("%d,%d,%d\n", x, y, z);
14:  }
15:  printf("%d,%d,%d\n", x, y, z);
16: }
17:}
```

**Adaptado de** Keith Schwartz

```
0: int x = 137;
1: int z = 42;
2: int MyFunction(int x, int y) {
3:  printf("%d,%d,%d\n", x, y, z);
4:  {
5:   int x, z;
6:   z = y;
7:   x = z;
8:   {
9:    int y = x;
10:   {
11:    printf("%d,%d,%d\n", x, y, z);
12:   }
13:   printf("%d,%d,%d\n", x, y, z);
14:  }
15:  printf("%d,%d,%d\n", x, y, z);
16: }
17:}
```

**Adaptado de** Keith Schwartz

```
0: int x = 137;
1: int z = 42;
2: int MyFunction(int x, int y) {
3:  printf("%d,%d,%d\n", x, y, z);
4:  {
5:   int x, z;
6:   z = y;
7:   x = z;
8:   {
9:    int y = x;
10:   {
11:    printf("%d,%d,%d\n", x, y, z);
12:   }
13:   printf("%d,%d,%d\n", x, y, z);
14:  }
15:  printf("%d,%d,%d\n", x, y, z);
16: }
17:}
```

**Adaptado de** Keith Schwartz

```
0: int x = 137;
1: int z = 42;
2: int MyFunction(int x, int y) {
3:  printf("%d,%d,%d\n", x, y, z);
4:  {
5:   int x, z;
6:   z = y;
7:   x = z;
8:   {
9:    int y = x;
10:   {
11:    printf("%d,%d,%d\n", x, y, z);
12:   }
13:   printf("%d,%d,%d\n", x, y, z);
14:  }
15:  printf("%d,%d,%d\n", x, y, z);
16: }
17:}
```

**Adaptado de** Keith Schwartz

```
0: int x = 137;
1: int z = 42;
2: int MyFunction(int x, int y) {
3:  printf("%d,%d,%d\n", x, y, z);
4:  {
5:   int x, z;
6:   z = y;
7:   x = z;
8:   {
9:    int y = x;
10:   {
11:    printf("%d,%d,%d\n", x, y, z);
12:   }
13:   printf("%d,%d,%d\n", x, y, z);
14:  }
15:  printf("%d,%d,%d\n", x, y, z);
16: }
17:}
```

**Adaptado de** Keith Schwartz

```
0: int x = 137;
1: int z = 42;
2: int MyFunction(int x, int y) {
3:  printf("%d,%d,%d\n", x@2, y@2, z@1);
4:  {
5:   int x, z;
6:   z = y;
7:   x = z;
8:   {
9:    int y = x;
10:   {
11:    printf("%d,%d,%d\n", x, y, z);
12:   }
13:   printf("%d,%d,%d\n", x, y, z);
14:  }
15:  printf("%d,%d,%d\n", x, y, z);
16: }
17:}
```

**Adaptado de** Keith Schwartz

```
0: int x = 137;
1: int z = 42;
2: int MyFunction(int x, int y) {
3:  printf("%d,%d,%d\n", x@2, y@2, z@1);
4:   {
5:    int x, z;
6:    z = y;
7:    x = z;
8:    {
9:     int y = x;
10:    {
11:     printf("%d,%d,%d\n", x, y, z);
12:    }
13:    printf("%d,%d,%d\n", x, y, z);
14:  }
15:  printf("%d,%d,%d\n", x, y, z);
16: }
17:}
```

**Adaptado de** Keith Schwartz

```
0: int x = 137;
1: int z = 42;
2: int MyFunction(int x, int y) {
3:  printf("%d,%d,%d\n", x@2, y@2, z@1);
4:  {
5:   int x, z;
6:   z = y;
7:   x = z;
8:   {
9:    int y = x;
10:   {
11:    printf("%d,%d,%d\n", x, y, z);
12:   }
13:   printf("%d,%d,%d\n", x, y, z);
14:  }
15:  printf("%d,%d,%d\n", x, y, z);
16: }
17:}
```

**Adaptado de** Keith Schwartz

```
0: int x = 137;
1: int z = 42;
2: int MyFunction(int x, int y) {
3:  printf("%d,%d,%d\n", x@2, y@2, z@1);
4:  {
5:   int x, z;
6:   z = y;
7:   x = z;
8:   {
9:    int y = x;
10:   {
11:    printf("%d,%d,%d\n", x, y, z);
12:   }
13:   printf("%d,%d,%d\n", x, y, z);
14:  }
15:  printf("%d,%d,%d\n", x, y, z);
16: }
17:}
```

**Adaptado de** Keith Schwartz

```
0: int x = 137;
1: int z = 42;
2: int MyFunction(int x, int y) {
3:  printf("%d,%d,%d\n", x@2, y@2, z@1);
4:  {
5:   int x, z;
6:   z = y@2;
7:   x = z;
8:   {
9:    int y = x;
10:   {
11:    printf("%d,%d,%d\n", x, y, z);
12:   }
13:   printf("%d,%d,%d\n", x, y, z);
14:  }
15:  printf("%d,%d,%d\n", x, y, z);
16: }
17:}
```

**Adaptado de** Keith Schwartz

```
0: int x = 137;
1: int z = 42;
2: int MyFunction(int x, int y) {
3:  printf("%d,%d,%d\n", x@2, y@2, z@1);
4:  {
5:   int x, z;
6:   z = y@2;
7:   x = z;
8:   {
9:    int y = x;
10:   {
11:    printf("%d,%d,%d\n", x, y, z);
12:   }
13:   printf("%d,%d,%d\n", x, y, z);
14:  }
15:  printf("%d,%d,%d\n", x, y, z);
16: }
17:}
```

**Adaptado de** Keith Schwartz

```
0: int x = 137;
1: int z = 42;
2: int MyFunction(int x, int y) {
3:  printf("%d,%d,%d\n", x@2, y@2, z@1);
4:  {
5:   int x, z;
6:   z = y@2;
7:   x = z@5;
8:   {
9:    int y = x;
10:   {
11:    printf("%d,%d,%d\n", x, y, z);
12:   }
13:   printf("%d,%d,%d\n", x, y, z);
14:  }
15:  printf("%d,%d,%d\n", x, y, z);
16: }
17:}
```

**Adaptado de** Keith Schwartz

```
0: int x = 137;
1: int z = 42;
2: int MyFunction(int x, int y) {
3:  printf("%d,%d,%d\n", x@2, y@2, z@1);
4:  {
5:   int x, z;
6:   z = y@2;
7:   x = z@5;
8:   {
9:    int y = x;
10:   {
11:    printf("%d,%d,%d\n", x, y, z);
12:   }
13:   printf("%d,%d,%d\n", x, y, z);
14:  }
15:  printf("%d,%d,%d\n", x, y, z);
16: }
17:}
```

**Adaptado de** Keith Schwartz

```
0: int x = 137;
1: int z = 42;
2: int MyFunction(int x, int y) {
3:  printf("%d,%d,%d\n", x@2, y@2, z@1);
4:  {
5:   int x, z;
6:   z = y@2;
7:   x = z@5;
8:   {
9:    int y = x;
10:   {
11:    printf("%d,%d,%d\n", x, y, z);
12:   }
13:   printf("%d,%d,%d\n", x, y, z);
14:  }
15:  printf("%d,%d,%d\n", x, y, z);
16: }
17:}
```

**Adaptado de** Keith Schwartz

```
0: int x = 137;
1: int z = 42;
2: int MyFunction(int x, int y) {
3:  printf("%d,%d,%d\n", x@2, y@2, z@1);
4:  {
5:   int x, z;
6:   z = y@2;
7:   x = z@5;
8:   {
9:    int y = x@5;
10:   {
11:    printf("%d,%d,%d\n", x, y, z);
12:   }
13:   printf("%d,%d,%d\n", x, y, z);
14:  }
15:  printf("%d,%d,%d\n", x, y, z);
16: }
17:}
```

**Adaptado de** Keith Schwartz

```
0: int x = 137;
1: int z = 42;
2: int MyFunction(int x, int y) {
3:  printf("%d,%d,%d\n", x@2, y@2, z@1);
4:  {
5:   int x, z;
6:   z = y@2;
7:   x = z@5;
8:   {
9:    int y = x@5;
10:   {
11:    printf("%d,%d,%d\n", x, y, z);
12:   }
13:   printf("%d,%d,%d\n", x, y, z);
14:  }
15:  printf("%d,%d,%d\n", x, y, z);
16: }
17:}
```

**Adaptado de** Keith Schwartz

```
0: int x = 137;
1: int z = 42;
2: int MyFunction(int x, int y) {
3:  printf("%d,%d,%d\n", x@2, y@2, z@1);
4:  {
5:   int x, z;
6:   z = y@2;
7:   x = z@5;
8:   {
9:    int y = x@5;
10:   {
11:    printf("%d,%d,%d\n", x, y, z);
12:   }
13:   printf("%d,%d,%d\n", x, y, z);
14:  }
15:  printf("%d,%d,%d\n", x, y, z);
16: }
17:}
```

**Adaptado de** Keith Schwartz

```
0: int x = 137;
1: int z = 42;
2: int MyFunction(int x, int y) {
3:  printf("%d,%d,%d\n", x@2, y@2, z@1);
4:  {
5:   int x, z;
6:   z = y@2;
7:   x = z@5;
8:   {
9:    int y = x@5;
10:   {
11:    printf("%d,%d,%d\n", x@5, y@9, z@5);
12:   }
13:   printf("%d,%d,%d\n", x, y, z);
14:  }
15:  printf("%d,%d,%d\n", x, y, z);
16: }
17:}
```

**Adaptado de** Keith Schwartz

```
0: int x = 137;
1: int z = 42;
2: int MyFunction(int x, int y) {
3:  printf("%d,%d,%d\n", x@2, y@2, z@1);
4:  {
5:   int x, z;
6:   z = y@2;
7:   x = z@5;
8:   {
9:    int y = x@5;
10:   {
11:    printf("%d,%d,%d\n", x@5, y@9, z@5);
12:   }
13:   printf("%d,%d,%d\n", x, y, z);
14:  }
15:  printf("%d,%d,%d\n", x, y, z);
16: }
17:}
```

**Adaptado de** Keith Schwartz

```
0: int x = 137;
1: int z = 42;
2: int MyFunction(int x, int y) {
3:  printf("%d,%d,%d\n", x@2, y@2, z@1);
4:  {
5:   int x, z;
6:   z = y@2;
7:   x = z@5;
8:   {
9:    int y = x@5;
10:   {
11:    printf("%d,%d,%d\n", x@5, y@9, z@5);
12:   }
13:   printf("%d,%d,%d\n", x@5, y@9, z@5);
14:  }
15:  printf("%d,%d,%d\n", x, y, z);
16: }
17:}
```

**Adaptado de** Keith Schwartz

```
0: int x = 137;
1: int z = 42;
2: int MyFunction(int x, int y) {
3:  printf("%d,%d,%d\n", x@2, y@2, z@1);
4:  {
5:   int x, z;
6:   z = y@2;
7:   x = z@5;
8:   {
9:    int y = x@5;
10:   {
11:    printf("%d,%d,%d\n", x@5, y@9, z@5);
12:   }
13:   printf("%d,%d,%d\n", x@5, y@9, z@5);
14:  }
15:  printf("%d,%d,%d\n", x, y, z);
16: }
17:}
```

**Adaptado de** Keith Schwartz

```
0: int x = 137;
1: int z = 42;
2: int MyFunction(int x, int y) {
3:  printf("%d,%d,%d\n", x@2, y@2, z@1);
4:  {
5:   int x, z;
6:   z = y@2;
7:   x = z@5;
8:   {
9:    int y = x@5;
10:   {
11:    printf("%d,%d,%d\n", x@5, y@9, z@5);
12:   }
13:   printf("%d,%d,%d\n", x@5, y@9, z@5);
14:  }
15:  printf("%d,%d,%d\n", x, y, z);
16: }
17:}
```

**Adaptado de** Keith Schwartz

```
0: int x = 137;
1: int z = 42;
2: int MyFunction(int x, int y) {
3:  printf("%d,%d,%d\n", x@2, y@2, z@1);
4:  {
5:   int x, z;
6:   z = y@2;
7:   x = z@5;
8:   {
9:    int y = x@5;
10:   {
11:    printf("%d,%d,%d\n", x@5, y@9, z@5);
12:   }
13:   printf("%d,%d,%d\n", x@5, y@9, z@5);
14:  }
15:  printf("%d,%d,%d\n", x@5, y@2, z@5);
16: }
17:}
```

**Adaptado de** Keith Schwartz

```
0: int x = 137;
1: int z = 42;
2: int MyFunction(int x, int y) {
3:  printf("%d,%d,%d\n", x@2, y@2, z@1);
4:  {
5:   int x, z;
6:   z = y@2;
7:   x = z@5;
8:   {
9:    int y = x@5;
10:   {
11:    printf("%d,%d,%d\n", x@5, y@9, z@5);
12:   }
13:   printf("%d,%d,%d\n", x@5, y@9, z@5);
14:  }
15:  printf("%d,%d,%d\n", x@5, y@2, z@5);
16: }
17:}
```

**Adaptado de** Keith Schwartz

```
0: int x = 137;
1: int z = 42;
2: int MyFunction(int x, int y) {
3:  printf("%d,%d,%d\n", x@2, y@2, z@1);
4:  {
5:   int x, z;
6:   z = y@2;
7:   x = z@5;
8:   {
9:    int y = x@5;
10:   {
11:    printf("%d,%d,%d\n", x@5, y@9, z@5);
12:   }
13:   printf("%d,%d,%d\n", x@5, y@9, z@5);
14:  }
15:  printf("%d,%d,%d\n", x@5, y@2, z@5);
16: }
17:}
```

**Adaptado de** Keith Schwartz

# Operações

- Tipicamente implementadas como pilha de tabelas
- Cada tabela corresponde a um escopo em particular
- Pilha facilita operações de ‘entrada’ e ‘saída’ de escopo
  - push/pop scope
  - insert/lookup symbol

# Usando Tabela de Símbolos

- Para processar uma porção do programa que delimita novo escopo
  - ‘Entre’ em um novo escopo
    - Adicione todas as declarações de variáveis à tabela de símbolos
    - Processe o corpo do bloco/função/classe
    - ‘Saia’ do escopo
- Muitas das análises semânticas são definidas em termos de travessias recursivas como a descrita acima

```
0:   int x;
1:   int y;
2:   int MyFunction(int x, int y)
3:   {
4:    int w, z;
5:    {
6:     int y;
7:    }
8:    {
9:     int w;
10:   }
11: }
```

**Adaptado de** Keith Schwartz

```
0:   int x;
1:   int y;
2:   int MyFunction(int x, int y)
3:   {
4:    int w, z;
5:    {
6:     int y;
7:    }
8:    {
9:     int w;
10:   }
11: }
```

**Adaptado de** Keith Schwartz

```
0:   int x;
1:   int y;
2:   int MyFunction(int x, int y)
3:   {
4:    int w, z;
5:    {
6:     int y;
7:    }
8:    {
9:     int w;
10:   }
11: }
```

**Adaptado de** Keith Schwartz

```
0:   int x;
1:   int y;
2:   int MyFunction(int x, int y)
3:   {
4:    int w, z;
5:    {
6:     int y;
7:    }
8:    {
9:     int w;
10:   }
11: }
```

**Adaptado de** Keith Schwartz

```
0:   int x;
1:   int y;
2:   int MyFunction(int x, int y)
3:   {
4:    int w, z;
5:    {
6:     int y;
7:    }
8:    {
9:     int w;
10:   }
11: }
```

**Adaptado de** Keith Schwartz

```
0:   int x;
1:   int y;
2:   int MyFunction(int x, int y)
3:   {
4:    int w, z;
5:    {
6:     int y;
7:    }
8:    {
9:     int w;
10:   }
11: }
```

**Adaptado de** Keith Schwartz

```
0:   int x;
1:   int y;
2:   int MyFunction(int x, int y)
3:   {
4:    int w, z;
5:    {
6:     int y;
7:    }
8:    {
9:     int w;
10:   }
11: }
```

**Adaptado de** Keith Schwartz

```
0:   int x;
1:   int y;
2:   int MyFunction(int x, int y)
3:   {
4:    int w, z;
5:    {
6:     int y;
7:    }
8:    {
9:     int w;
10:   }
11: }
```

**Adaptado de** Keith Schwartz

```
0:   int x;
1:   int y;
2:   int MyFunction(int x, int y)
3:   {
4:    int w, z;
5:    {
6:     int y;
7:    }
8:    {
9:     int w;
10:   }
11: }
```

**Adaptado de** Keith Schwartz

```
0:   int x;
1:   int y;
2:   int MyFunction(int x, int y)
3:   {
4:    int w, z;
5:    {
6:     int y;
7:    }
8:    {
9:     int w;
10:   }
11: }
```

**Adaptado de** Keith Schwartz

```
0:   int x;
1:   int y;
2:   int MyFunction(int x, int y)
3:   {
4:    int w, z;
5:    {
6:     int y;
7:    }
8:    {
9:     int w;
10:   }
11: }
```

**Adaptado de** Keith Schwartz

```
0:   int x;
1:   int y;
2:   int MyFunction(int x, int y)
3:   {
4:    int w, z;
5:    {
6:     int y;
7:    }
8:    {
9:     int w;
10:   }
11: }
```

**Adaptado de** Keith Schwartz

```
0:   int x;
1:   int y;
2:   int MyFunction(int x, int y)
3:   {
4:    int w, z;
5:    {
6:     int y;
7:    }
8:    {
9:     int w;
10:  }
11: }
```

**Adaptado de** Keith Schwartz

# Spaghetti Stacks

- Trata a tabela de símbolos como uma lista encadeada de escopos
- Cada escopo armazena um ponteiro para o seu pai mas a recíproca não é verdadeira
- Em qualquer ponto do programa a tabela pode ser vista como uma pilha

# Duas visões

- A estrutura de escopo é melhor representada pela spaghetti stack
- Spaghetti - Estrutura estática
- Pilha explícita - Estrutura dinâmica
- A pilha explícita pode ser vista como uma otimização da spaghetti

# Escopo em OO

- Herança - o escopo da classe filha tem ponteiro para o escopo da classe pai
- Acessar um atributo significa subir na cadeia de escopos até encontrar o atributo ou erro
- Podemos desambiguar o escopo explicitamente

```
public class Base {
    public int value = 1;
}
public class Derived extends Base {
    public int value = 2;
    public void doSomething() {
        int value = 3;
        System.out.println(value);
        System.out.println(this.value);
        System.out.println(super.value);
    }
}
```

**Adaptado de** Keith Schwartz

# Escopo na prática

```
public class A {
    private B myB;
}
```

**Adaptado de** Keith Schwartz

# Escopo na prática

```
public class A {
    private B myB;
}
class B {
    private A myA;
}
```

**Adaptado de** Keith Schwartz

# Passadas

- Conseguimos resolver análise léxica e sintática com uma única passada sobre a entrada
- Alguns compiladores também combinam análise semântica e geração de código
  - chamados de *single-pass compilers*
- Outros passam novamente pela entrada
  - chamados de *multi-pass compilers*

# Escopo em *multi-pass*

- Ler a entrada e construir a AST (primeira passada)
- Caminhar pela AST coletando informações sobre as classes (segunda passada)
- Caminhar pela AST checando outras propriedades (terceira passada)
- …
- Pode combinar algumas destas passadas, embora sejam logicamente distintas