Nested Scopes

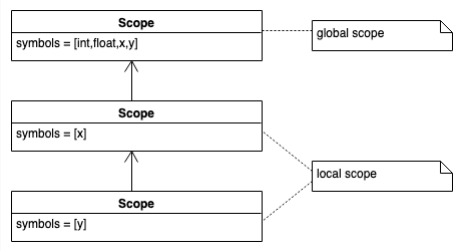
Carsten Gips (HSBI)

Symbole und (nested) Scopes

```
int x = 42;
float y;
{
    int x;
    x = 1;
    y = 2;
    { int y = x; }
}
```

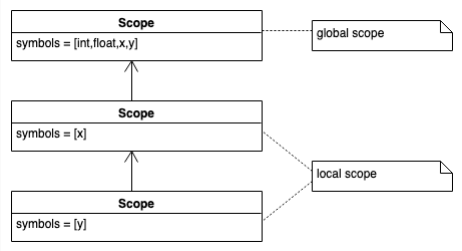
Symbole und (nested) Scopes

```
int x = 42;
float y;
{
    int x;
    x = 1;
    y = 2;
    { int y = x; }
}
```



Symbole und (nested) Scopes

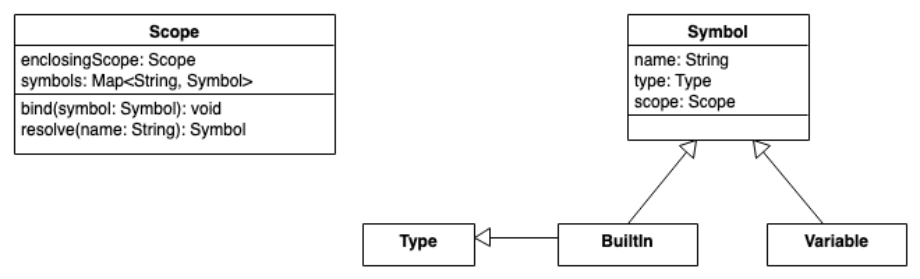
```
int x = 42;
float y;
{
    int x;
    x = 1;
    y = 2;
    { int y = x; }
}
```



Aufgaben:

- `bind()`: Symbole definieren
- `resolve()`: Symbole abrufen

Nested Scopes: Symbole und Scopes



Quelle: Eigene Modellierung nach einer Idee in (Parr 2010, p. 142)

Nested Scopes: Definieren und Auflösen von Namen

```
class Scope:
    Scope enclosingScope    # None if global (outermost) scope
    Symbol<String, Symbol> symbols

    def resolve(name):
        # do we know "name" here?
        if symbols[name]: return symbols[name]
        # if not here, check any enclosing scope
        try: return enclosingScope.resolve(name)
        except: return None     # not found

    def bind(symbol):
        symbols[symbol.name] = symbol
        symbol.scope = self     # track the scope in each symbol
```

Quelle: Eigene Implementierung nach einer Idee in (Parr 2010, p. 169)

Nested Scopes: Listener

```
start   :   stat+ ;

stat    : block | varDecl | expr ';' ;
block   : '{' stat* '}' ;

varDecl : type ID ('=' expr)? ';' ;
expr    : var '=' INT ;

var     : ID ;
type    : 'float' | 'int' ;
```

```
int x = 42;

{ int y = 9; x = 7; }
```

```
class MyListener(BaseListener):
    Scope scope

    def enterStart(Parser.FileContext ctx):
        globals = Scope()
        globals.bind(BuiltIn("int"))
        globals.bind(BuiltIn("float"))
        scope = globals

    def enterBlock(Parser.BlockContext ctx):
        scope = Scope(scope)
    def exitBlock(Parser.BlockContext ctx):
        scope = scope.enclosingScope

    def exitVarDecl(Parser.VarDeclContext ctx):
        t = scope.resolve(ctx.type().getText())
        var = Variable(ctx.ID().getText(), t)
        scope.bind(var)
    def exitVar(Parser.VarContext ctx):
        name = ctx.ID().getText()
        var = scope.resolve(name)
        if var == None: error("no such var: " + name)
```

Wrap-Up

- Symboltabellen: Verwaltung von Symbolen und Typen (Informationen über Bezeichner)
- Blöcke: Nested Scopes => hierarchische Organisation
- Binden von Bezeichner gleichen Namens an ihren jeweiligen Scope => `bind()`
- Abrufen von Bezeichnern aus dem aktuellen Scope oder den Elternscopes => `resolve()`

LICENSE