Funktionen

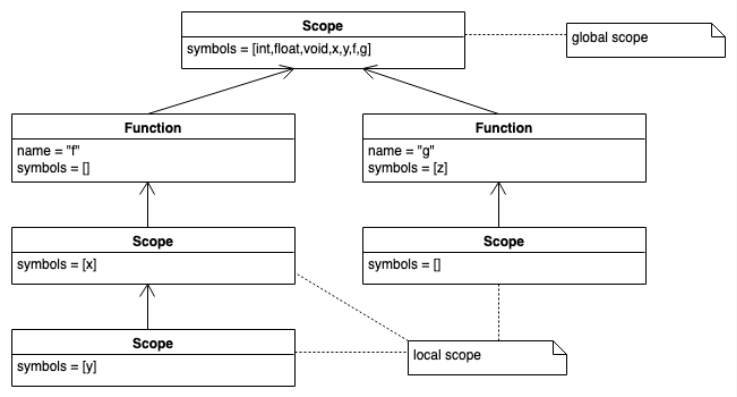
Carsten Gips (FH Bielefeld)

Funktionen und Scopes

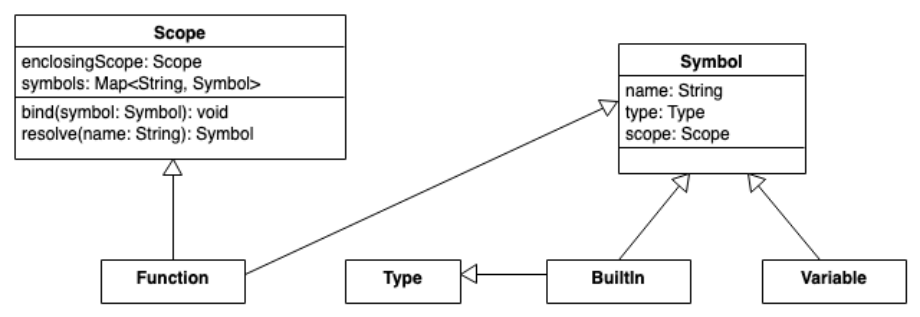
```
int x = 42;
int y;
void f() {
    int x;
    x = 1;
    y = 2;
    { int y = x; }
}
void g(int z){}
```

Funktionen und Scopes

```
int x = 42;
int y;
void f() {
    int x;
    x = 1;
    y = 2;
    { int y = x; }
}
void g(int z){}
```



Erweiterung des Klassendiagramms für Funktions-Scopes



Quelle: Eigene Modellierung nach einer Idee in (Parr 2010, 147)

Funktionen sind Symbole *und* Scopes

```
class Function(Scope, Symbol):
    def __init__(name, retType, enclScope):
        Symbol.__init__(name, retType)      # we are "Symbol" ...
        enclosingScope = enclScope          # ... and "Scope"
```

Funktionen: Listener

```
funcDecl : type ID '(' params? ')' block ;
params   : param (',' param)* ;
param    : type ID ;

call     : ID '(' exprList? ')' ;
exprList : expr (',' expr)* ;
```

```
int f(int x) {
    int y = 9;
}

int x = f(x);
```

```
def enterFuncDecl(Parser.FuncDeclContext ctx):
    name = ctx.ID().getText()
    type = scope.resolve(ctx.type().getText())
    func = Function(name, type, scope)
    scope.bind(func)
    # change current scope to function scope
    scope = func

def exitFuncDecl(Parser.FuncDeclContext ctx):
    scope = scope.enclosingScope
def exitParam(Parser.ParamContext ctx):
    t = scope.resolve(ctx.type().getText())
    var = Variable(ctx.ID().getText(), t)
    scope.bind(var)

def exitCall(Parser.CallContext ctx):
    name = ctx.ID().getText()
    func = scope.resolve(name)
    if func == None:
        error("no such function: " + name)
    if func.type == Variable:
        error(name + " is not a function")
```

Wrap-Up

- Symboltabellen: Verwaltung von Symbolen und Typen (Informationen über Bezeichner)
- Funktionen: Nested Scopes $=>$ hierarchische Organisation
- Umgang mit dem Funktionsnamen, den Parametern und dem Funktionskörper

LICENSE