Deklarierte Namen in MicroJava

• Programm	`Program()`	
– Konstanten	`ConstDecl()`	
– Globale Variablen	`VarDecl()`	`level = 0`
– Innere Klassen	`ClassDecl()`	
• Felder	`VarDecl()`	`level = 1`
– Methoden	`MethodDecl()`	
• Formale Parameter	`FormPars()`	
• Lokale Variablen	`VarDecl()`	`level = 1`

- Wo werden die Namen deklariert
  = wo werden sie in die Symbolliste eingefügt

Objektknoten

```
class Obj {
  static final int
    Con = 0, Var = 1, Type = 2, Meth = 3, Prog = 4;

  int    kind;     // Con, Var, Type, Meth, Prog
  String name;
  Struct type;
  Obj    next;
  int    val;      // Con: value
  int    adr;      // Var, Meth: address
  int    level;    // Var: 0 = global, 1 = local
  int    nPars;    // Meth: number of parameters
  Obj    locals;   // Meth: parameters and local objects
                   // Prog: global variables, constants,
                   //       classes and methods
}
```

Strukturknoten und Scope-Knoten

```
class Struct {
  static final int
    None = 0, Int = 1, Char = 2, Arr = 3, Class = 4;

  int    kind;      // None, Int, Char, Arr, Class
  Struct elemType;  // Arr: element type
  int    n;         // Class: number of fields
  Obj    fields;    // Class: list of fields
}


class Scope {
  Scope outer;   // next outer scope
  Obj   locals;  // list of objects in this scope
  int   nVars;   // number of variables in this scope
}
```

Symboltabelle

```
class Tab {
  static Struct noType, intType, charType, nullType;
  static Obj noObj, chrObj, ordObj, lenObj;

  static Scope topScope; // current scope
  static int level;      // nesting level of current scope

  static void init();
  static void openScope();
  static void closeScope();
  static Obj insert(int kind, String name, Struct type);
  static Obj find(String name);
  static Obj findField(String name, Struct type);
}
```

Füllen der Symbolliste

```
/** VarDecl = Type ident { "," ident } ";" . */
static void VarDecl() {
  Struct type = Type();
  check(ident);
  Tab.insert(Obj.Var, t.str, type);
  while (sym == comma) {
    scan();
    check(ident);
    Tab.insert(Obj.Var, t.str, type);
  }
  check(semicolon);
}
```

Symbolliste verwenden

```
/** Type = ident [ "[" "]" ] . */
static Struct Type() {
  check(ident);
  Obj o = Tab.find(t.str);
  if (o.kind != Obj.Type) error("NO_TYPE");
  Struct type = o.type;
  if (sym == lbrack) {
    scan();
    check(rbrack);
    type = new Struct(Struct.Arr, type);
  }
  return type;
}
```

Innere Klassen

```
/** ClassDecl = "class" ident "{" {VarDecl} "}" . */
static void ClassDecl() {
  check(class_);
  check(ident);
  Obj clazz =
    Tab.insert(Obj.Type, t.str, new Struct(Struct.Class));
  check(lbrace);
  Tab.openScope();
  while (sym == ident) VarDecl();
  check(rbrace);
  clazz.type.fields = Tab.topScope.locals;
  clazz.type.n = Tab.topScope.nVars;
  Tab.closeScope();
}
```

Beispiel: Symbollistenaufbau

```
program ABC ①
  char[] c;
  int max;
  char npp;
{
  int put ② (int x) { ③
    x++;
    print(x, 5);
    npp = 'C';
    return x;
  } ④
} ⑤
```

Beispiel: Bei Punkt ①

scope "Universe"

Type "int" *intType* → Type "char" *charType* → Con "null" *nullType* → Meth "chr" *charType* 1 → Meth "ord" *intType* 1 → Meth "len" *intType* 1 → Prog "ABC" *noType*

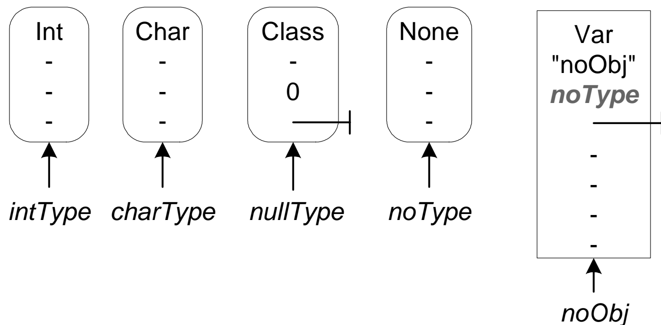
Var "i" *intType* 0 1

Var "ch" *charType* 0 1

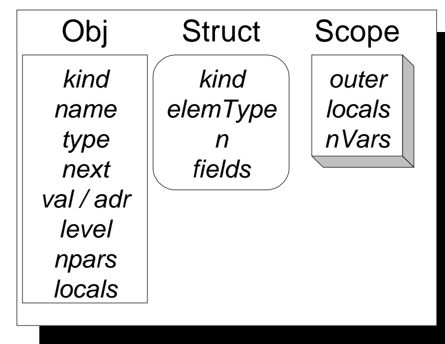
Var "arr" 0 1 → Arr *noType*

topScope

Vordefinierte Typen und Objekte:



Struktur der 3 Knotenarten:



Beispiel: Bei Punkt ②

scope "Universe"

Type "int" *intType* - - - - -

Type "char" *charType* - - - - -

Con "null" *nullType* - - - - -

Meth "chr" *charType* - - 1

Meth "ord" *intType* - - 1

Meth "len" *intType* - - 1

Prog "ABC" *noType* - - - - -

Var "i" *intType* 0 1 - -

Var "ch" *charType* 0 1 - -

Var "arr" 0 1 - -

Arr *noType* - -

0

Arr *charType* - -

scope "ABC"

Var "c" 0 0 - -

Var "max" *intType* 1 0 - -

Var "npp" *charType* 2 0 - -

Meth "put" *intType* - - -

~~0~~ ~~1~~ ~~2~~ 3

0

topScope

Beispiel: Bei Punkt ③

Beispiel: Bei Punkt ④

scope "Universe"

0

Type "int" *intType* - - - -

Type "char" *charType* - - - -

Con "null" *nullType* - - - -

Meth "chr" *charType* - - 1

Var "i" *intType* 0 1 - -

Meth "ord" *intType* - - 1

Var "ch" *charType* 0 1 - -

Meth "len" *intType* - - 1

Var "arr" 0 1 - -

Arr *noType* - -

Prog "ABC" *noType* - - - -

scope "ABC"

3

topScope

Var "c" 0 0 - -

Arr *charType* - -

Var "max" *intType* 1 0 - -

Var "npp" *charType* 2 0 - -

Meth "put" *intType* - - 1

Var "x" *intType* 0 1 - -

Beispiel: Bei Punkt ⑤

scope "Universe"

Type "int" *intType*

Type "char" *charType*

Con "null" *nullType*

Meth "chr" *charType* 1

Meth "ord" *intType* 1

Meth "len" *intType* 1

Prog "ABC" *noType*

0

topScope

Var "i" *intType* 0 1

Var "ch" *charType* 0 1

Var "arr" 0 1

Arr *noType*

Arr *charType*

scope "ABC"

Var "c" 0 0

Var "max" *intType* 1 0

Var "npp" *charType* 2 0

Meth "put" *intType* 1

Var "x" *intType* 0 1

UE 4: Symbolliste und Fehlerbehandlung

- Keine neuen Angabe- und Test-Klassen

- Abgabe
  - siehe Abgabeanleitung auf Homepage!
  - elektronisch bis Mi, 23.11.2005, 20:15
    - alle zum Ausführen benötigten Dateien
  - auf Papier bis Mi, 23.11.2005, 20:15
    - Parser.java, Errors.java, Tab.java, Struct.java