# Deklarierte Namen in MicroJava

| | | |
|---|---|---|
| • Programm | `Program()` | |
| – Konstanten | `ConstDecl()` | |
| – Globale Variablen | `VarDecl()` | `level = 0` |
| – 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 {
  enum Kind {
    Con, Var, Type, Meth, Prog
  }

  Kind   kind;
  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 {
  enum Kind {
    None, Int, Char, Arr, Class
  }

  Kind   kind;
  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 {
  public static final Struct
                  noType, intType, charType, nullType;
  public Obj noObj, chrObj, ordObj, lenObj;

  public Parser parser;   // target for errors
  public Scope topScope;  // current scope
  private int level;      // nesting level of current scope

  public Tab(Parser parser);
  public void openScope();
  public void closeScope();
  public Obj insert(Obj.Kind kind, String name, Struct type);
  public Obj find(String name);
  public Obj findField(String name, Struct type);
}
```

# Füllen der Symbolliste

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

# Symbolliste verwenden

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

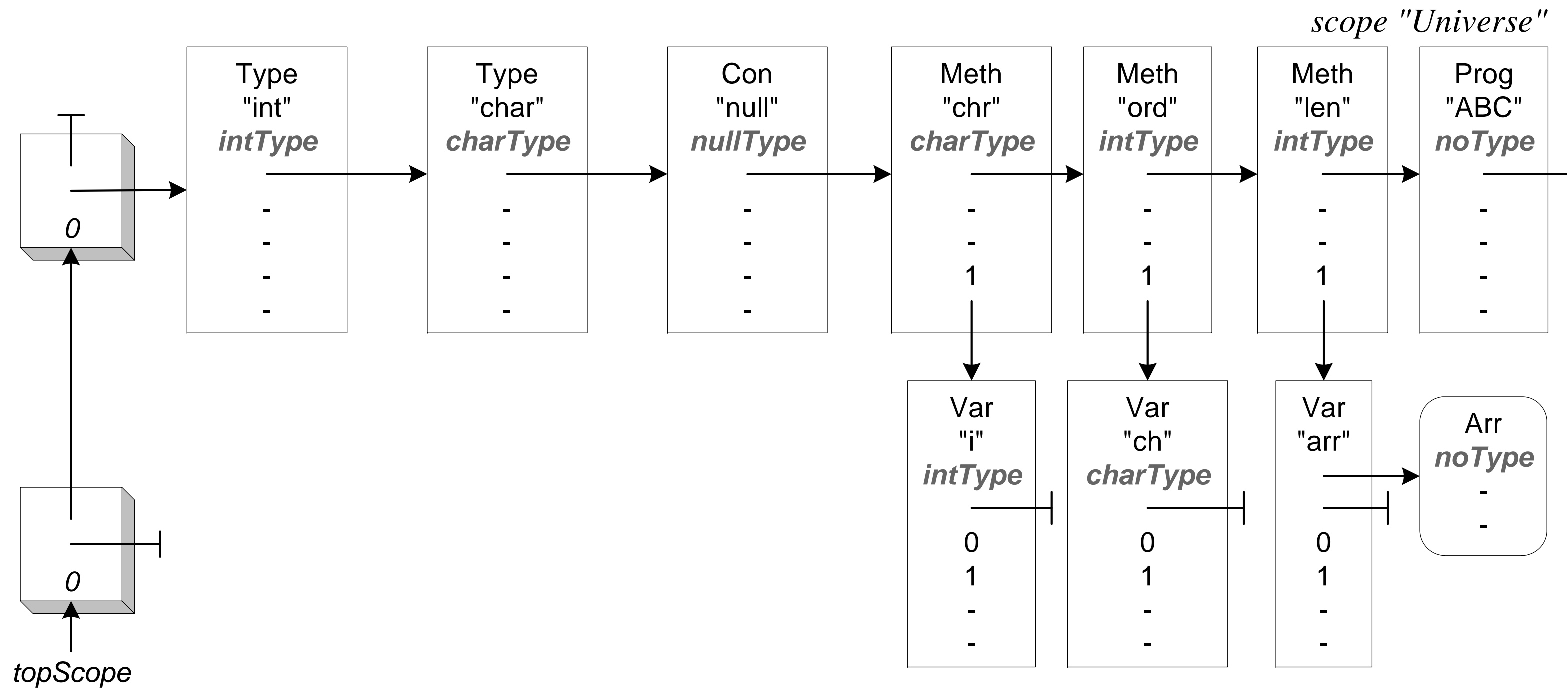
# Klassen

```
/** ClassDecl = "class" ident "{" {VarDecl} "}" . */
private void ClassDecl() {
  check(class_);
  check(ident);
  Obj clazz = tab.insert(Obj.Kind.Type,
      t.str, new Struct(Struct.Kind.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();
}
```

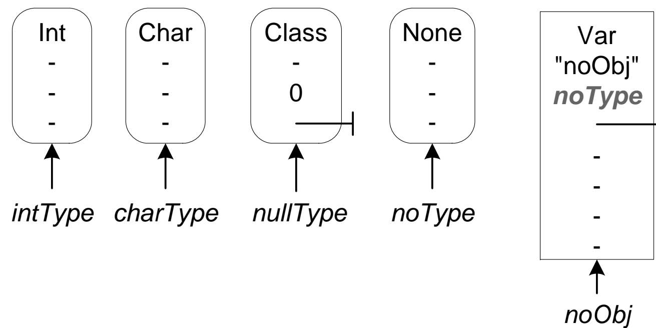
# Beispiel: Symbollistenaufbau

```
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
```

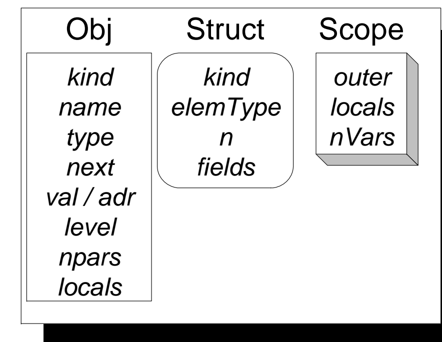
# Beispiel: Bei Punkt ①




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* - -

Arr *charType* - -

*scope "ABC"*

Var "c" 0 0 - -

Var "max" *intType* 1 0 - -

Var "npp" *charType* 2 0 - -

Meth "put" *intType* - - -

*0*

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

*0*

*topScope*

# Beispiel: Bei Punkt ③

# Beispiel: Bei Punkt ④

# Beispiel: Bei Punkt ⑤

scope "Universe"

0
topScope

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* - -

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, 26.11.2008, 20:15
    - alle Java-Dateien des Compilers
    - keine class-Dateien
    - keine Testfälle
    - auf Verzeichnisstruktur achten
  - auf Papier
    - Parser.java, Tab.java