Beispiel: Codeerzeugung

Assignment = Designator "=" Expr .

Expr = Term { "+" Term } .

Term = Factor { "*" Factor } .

Factor = number | Designator.

Designator = ident ["." ident | "[" Expr "]"].

Bsp 1: **i = 10;**

Deklaration:

```
class A
    final int max = 12;            // Konstante
    char c; int i;                 // globale Variablen
    class B { int x, y; }          // innere Klasse mit Feldern
{   void foo ()   int[] iarr; B b; int n;   {...}    }
```

```
const   10
putstatic   1
```

= **8** byte

Bsp 2: **n = 3;**

Deklaration:
```
class A
    final int max = 12;        // Konstante
    char c; int i;             // globale Variablen
    class B { int x, y; }      // innere Klasse mit Feldern
{   void foo ()   int[] iarr; B b; int n;   {...}    }
```

```
const_3
store_2
```
= **2** byte

Bsp 3: **n = 3 + i;**

Deklaration:

```
class A
    final int max = 12;          // Konstante
    char c; int i;               // globale Variablen
    class B { int x, y; }        // innere Klasse mit Feldern
{   void foo ()   int[] iarr; B b; int n;   {...}   }
```

```
const_3                                        = 6 byte
getstatic   1
add
store_2
```

Bsp 4: n = 3 + i * max - n;

Deklaration:

```
class A
    final int max = 12;        // Konstante
    char c; int i;             // globale Variablen
    class B { int x, y; }      // innere Klasse mit Feldern
{   void foo ()   int[] iarr; B b; int n;   {...}   }
```

```
const_3                                   = 13 byte
getstatic   1
const   12
mul
add
load_2
sub
store_2
```

Bsp 5: iarr[5] = 10;

Deklaration:

```
class A
     final int max = 12;        // Konstante
     char c; int i;             // globale Variablen
     class B { int x, y; }      // innere Klasse mit Feldern
{    void foo ()   int[] iarr; B b; int n;   {...}    }
```

```
load_0                                         = 8 byte
const_5
const   10
astore
```

Bsp 6: **b.y = iarr[5] * 3;**

Deklaration:

```
class A
    final int max = 12;         // Konstante
    char c; int i;              // globale Variablen
    class B { int x, y; }       // innere Klasse mit Feldern
  {  void foo ()   int[] iarr; B b; int n;   {...}   }
```

```
load_1
load_0
const_5
aload
const_3
mul
putfield  1
```

= **9** byte

Bsp 7: **n--;**

Deklaration:

```
class A
    final int max = 12;           // Konstante
    char c; int i;                // globale Variablen
    class B { int x, y; }         // innere Klasse mit Feldern
{   void foo ()   int[] iarr; B b; int n;   {...}   }
```

inc   2   -1   = **3** byte

Bsp 8: **i++;**

Deklaration:
```
class A
    final int max = 12;        // Konstante
    char c; int i;             // globale Variablen
    class B { int x, y; }      // innere Klasse mit Feldern
{   void foo ()   int[] iarr; B b; int n;   {...}    }
```

```
getstatic   1                                   = 8 byte
const_1
add
putstatic   1
```

Bsp 9: **iarr[0]--;**

Deklaration:

```
class A
    final int max = 12;         // Konstante
    char c; int i;              // globale Variablen
    class B { int x, y; }       // innere Klasse mit Feldern
{   void foo ()   int[] iarr; B b; int n;   {...}    }
```

```
load_0                                        = 7 byte
const_0
dup2
aload
const_m1
add
astore
```

Sem. Aktionen für Codeerzeugung (1)

```
Designator↑item =       (. Item item, x; Obj o; .)
   ident↑t              (. o = Tab.find(t.string); item = new Item(o); .)
   [ "." ident↑t        (. if (item.type.kind == Struct.Class) {
                              Code.load(item);
                              o = Tab.findField(t.string, item.type);
                              item.kind = Item.Fld;
                              item.type = o.type; item.adr = o.adr;
                              } else semError(...);
                         .)
   |  "["               (. Code.load(item); .)
      Expr↑x            (. if (item.type.kind == Struct.Arr) {
                              if (x.type != Tab.intType) semError(...);
                              Code.load(x);
                              item.kind = Item.Elem;
                              item.type = item.type.elemType;
                           } else semError(...);
                         .)
      "]"
   ] .
```

Klasse *Item* – Konstruktor *Item(Obj)*

```
public Item (Obj o) {
    type = o.type; adr = o.adr;
    switch (o.kind) {
        case Obj.Con:
            kind = Con; break;
        case Obj.Var:
            if (o.level == 0) kind = Static; else kind = Local; break;
        case Obj.Meth:
            kind = Meth; obj = o; break;
      default:
         Parser.Errors.semError(CREATE_ITEM);
         throw new Error();   // don't: System.exit(0);
    }
}
```

Sem. Aktionen für Codeerzeugung (2)

```
Factor↑item =               (.  Item item; .)
        number↑t            (.  item = new Item(Item.Con, t.val, Tab.intType);  .)
    |   Designator↑item .


Term↑x =
    Factor↑x
    {   "*" Factor↑y        (.  Code.load(x); Code.load(y);
                                if (x.type==Tab.intType && y.type==Tab.intType) {
                                    Code.put(Code.mul);
                                    x.kind = Item.Stack;
                                } else semError(...);
                             .)
    } .
```

Sem. Aktionen für Codeerzeugung (3)

```
Expr↑x =
    Term↑x
    {   "+" Term ↑y        (.  Code.load(x); Code.load(y);
                               if (x.type==Tab.intType && y.type==Tab.intType) {
                                   Code.put(Code.add);
                                   x.kind =  Item.Stack;
                               } else semError(...);
                           .)
    } .


Assignment =
    Designator↑x
    "=" Expr↑y             (.  if (y.type.assignableTo(x.type))
                                   Code.assign(x, y);
                               else semError(...);
                           .)
    .
```

Klasse *Code* – Hilfsmethode *load*

```
public static void load (Item x) {
    switch (x.kind) {
        case Item.Con:
            if (x.type == Tab.nullType) put(const_n + 0);
            else loadConst(x.adr); break;
        case Item.Local:
            if (0 <= x.adr && x.adr <= 3) put(load_n + x.adr);
            else { put(load); put(x.adr); } break;
        case Item.Static:
            put(getstatic); put2(x.adr); break;
        case Item.Stack:
            break;  // nothing to do (already loaded)
        case Item.Fld:
            put(getfield); put2(x.adr); break;
        case Item.Elem:
            if (x.type.kind == Struct.Char) put(baload);
            else put(aload); break;
        default:    Parser.Errors.semError("compiler error in Code.load");
                    throw new Error();  // don't: System.exit(0);
    }
    x.kind = Item.Stack;
}
```

Klasse *Code* – Hilfsmethode *assign*

```
public static void assign (Item x, Item y) {
    load(y);
    switch (x.kind) {
        case Item.Local:
            if (0 <= x.adr && x.adr <= 3) put(store_n + x.adr);
            else { put(store); put(x.adr); } break;
        case Item.Static:
            put(putstatic); put2(x.adr); break;
        case Item.Fld:
            put(putfield); put2(x.adr); break;
        case Item.Elem:
            if (x.type.kind == Struct.Char) put(bastore);
            else put(astore); break;
        default:
            Parser.Errors.semError("LHS of assignment must be a variable");
    }
}
```