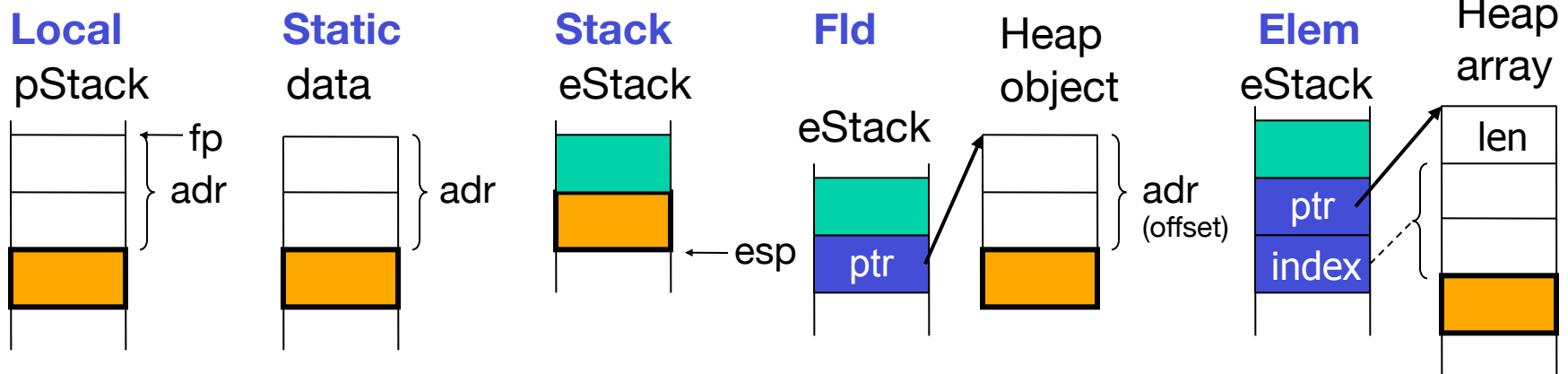


Klasse Operand

```

class Operand {
    public enum Kind {
        Con, Local, Static, Stack, Fld, Elem, Meth
    }
    public Kind kind;
    public Struct type;
    public int val;           // Con: value
    public int adr;         // Local, Static, Fld, Meth: address
    public Obj obj;        // Meth: method object from the symbol table
}

```



Beispiel

Assignment = Designator "=" Expr .

Expr = Term { "+" Term } .

Term = Factor { "*" Factor } .

Factor = number | Designator.

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

Designator erkennen



```
void Designator() {
    check(ident);

    if (sym == period) {
        scan();

        check(ident);

    } else if (sym == lbrack) {
        scan();

        Expr();

        check(rbrack);
    }
}
```

Operand erzeugen



```
Operand Designator() {
    check(ident);
    Operand x = new Operand(tab.find(t.str), this);
    if (sym == period) {
        scan();

        check(ident);
        Obj obj = tab.findField(t.str, x.type);
        x.kind = Item.Kind.Fld; x.type = obj.type; x.adr = obj.adr;
    } else if (sym == lbrack) {
        scan();

        Operand y = Expr();

        x.kind = Operand.Kind.Elem; x.type = x.type.elemType;
        check(rbrack);
    }
    return x;
}
```

Code erzeugen



```
Operand Designator() {
  check(ident);
  Operand x = new Operand(tab.find(t.str), this);
  if (sym == period) {
    scan();
    code.load(x);
    check(ident);
    Obj obj = tab.findField(t.str, x.type);
    x.kind = Item.Kind.Fld; x.type = obj.type; x.adr = obj.adr;
  } else if (sym == lbrack) {
    scan();
    code.load(x);
    Operand y = Expr(),
    code.load(y);
    x.kind = Operand.Kind.Elem; x.type = x.type.elemType;
    check(rbrack);
  }
  return x;
}
```

x.type.kind == Struct.Kind.Class

x.type.kind == Struct.Kind.Arr

y.type == Tab.intType



Konstruktor Operand(Obj)

```
Operand(Obj o, Parser parser) {  
    type = o.type;  
    val = o.val;  
    adr = o.adr;  
    switch (o.kind) {  
        case Con:  
            kind = Kind.Con;  
            break;  
        case Var:  
            if (o.level == 0) { kind = Kind.Static; } else { kind = Kind.Local; }  
            break;  
        case Meth:  
            kind = Kind.Meth; obj = o;  
            break;  
        default:  
            parser.error(NO_OPERAND);  
    }  
}
```

Code.load()

```
void load(Operand x) {
    switch (x.kind) {
        case Con: loadConst(x.val); break;
        case Local:
            switch (x.adr) {
                case 0: put(OpCodes.load_0); break;
                case 1: put(OpCodes.load_1); break;
                case 2: put(OpCodes.load_2); break;
                case 3: put(OpCodes.load_3); break;
                default: put(OpCodes.load); put(x.adr); break;
            }
            break;
        case Static: put(OpCodes.getstatic); put2(x.adr); break;
        case Stack: break; // nothing to do (already loaded)
        case Fld: put(OpCodes.getfield); put2(x.adr); break;
        case Elem:
            if (x.type == Tab.charType) { put(OpCodes.baload); }
            else { put(OpCodes.aload); }
            break;
        default: parser.error(NO_VAL);
    }
    x.kind = Operand.Kind.Stack;
}
```

Factor und Term

```

Operand Factor() {
  Operand x;
  if (sym == number) {
    x = new Operand(t.val);
    scan();
  } else {
    x = Designator();
  }
  return x;
}

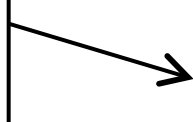
```

```

Operand Term() {
  Operand x = Factor();
  while (sym == times) {
    scan();
    code.load(x);
    Operand y = Factor();
    code.load(y);
    code.put(OpCode.mul);
  }
  return x;
}

```

x.type == Tab.intType &&
y.type == Tab.intType



Expr und Assignment

```
Operand Expr() {  
    Operand x = Term();  
    while (sym == plus) {  
        scan();  
        code.load(x);  
        Operand y = Term();  
        code.load(y);  
        code.put(OpCode.add);  
    }  
    return x;  
}
```

x.type == Tab.intType &&
y.type == Tab.intType

```
void Assignment() {  
    Operand x = Designator();  
    check(assign);  
    Operand y = Expr();  
    code.assign(x, y);  
}
```

y.type.assignableTo(x.type)

Klasse Code – Hilfsmethode assign

```
void assign(Operand x, Operand y) {  
    load(y);  
    switch (x.kind) {  
    case Local:  
        switch (x.adr) {  
        case 0: put(OpCode.store_0); break;  
        case 1: put(OpCode.store_1); break;  
        case 2: put(OpCode.store_2); break;  
        case 3: put(OpCode.store_3); break;  
        default: put(OpCode.store); put(x.adr); break;  
        } break;  
    case Static: put(OpCode.putstatic); put2(x.adr); break;  
    case Fld: put(OpCode.putfield); put2(x.adr); break;  
    case Elem:  
        if (x.type == Tab.charType) { put(OpCode.bastore); }  
        else { put(OpCode.astore); }  
        break;  
    default: parser.error(NO_VAR);  
    }  
}
```

Klasse Struct – Hilfsmethoden



```
boolean isRefType() {  
    return kind == Kind.Class || kind == Kind.Arr;  
}
```

```
boolean equals(Struct other) {  
    if (kind == Kind.Arr) {  
        return other.kind == Kind.Arr && elemType.equals(other.elemType);  
    } else {  
        return this == other; // must be same type node  
    }  
}
```

Klasse Struct – Hilfsmethoden



```
boolean compatibleWith(Struct other) {  
    return this.equals(other) ||  
           (this == Tab.nullType && other.isRefType()) ||  
           (other == Tab.nullType && this.isRefType());  
}
```

```
boolean assignableTo(Struct dest) {  
    return this.equals(dest) ||  
           (this == Tab.nullType && dest.isRefType()) ||  
           (this.kind == Kind.Arr && dest.kind == Kind.Arr &&  
            dest.elemType == Tab.noType); // for function len()  
}
```

Beispiel: $b.x = \text{iarr}[5] + i * n$

Deklaration: program 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; {...} }
```

