Fehlerbehandlung

- Panic Mode
  - Abbruch beim ersten Fehler
  - **Übung 3**
- Allgemeine Fangsymbole
  - Synchronisation der restlichen Eingabe mit der Grammatik
  - Parser kennt an jeder Stelle alle gültigen Nachfolge-Symbole
  - Aufwendig
- Spezielle Fangsymbole
  - Synchronisation nur an besonders "sicheren" Stellen.
  - Beispiele: Schlüsselwörter, Strichpunkte, …
  - **Übung 4**

Beispiel: Deklarationen


```
DeclPart    = { ForwardDecl } "{" Body "}" .
ForwardDecl = "void" ident "(" ")" ";" .
Body        = ... .
```

Welche Deklarationen kann man damit erzeugen?

```
void p1();
void p2();
void p3();
...
{
    ...
}
```

Beispiel: Deklarationen


```
DeclPart    = { ForwardDecl } "{" Body "}" .
ForwardDecl = "void" ident "(" ")" ";" .
Body        = ... .
```

```
private void DeclPart () {
  while (sym == void_) {
    ForwardDecl();
  }
  check(lbrace); Body(); check(rbrace);
}
```

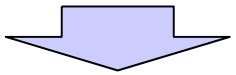
Bsp: Fehler in *ForwardDecl*


```
void p [];
{ … }
```

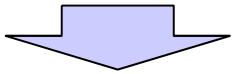
```
                        Erkenne DeclPart
next() → void_             Erkenne ForwardDecl
                              void_ erkannt
next() → ident                ident erkannt
next() → lbrack               ERROR: "( expected"
                                  ERROR: ") expected"
                                  ERROR: "; expected"
                              ERROR: "{ expected"
                              ...
                              ERROR: "} expected"
```

Bsp: First/Follow-Sets


```
DeclPart    = { ForwardDecl } "{" Body "}" .
ForwardDecl = "void" ident "(" ")" ";" .
Body        = ... .
```



First(ForwardDecl) = void_
Follow(ForwardDecl) = First(ForwardDecl) + lbrace = void_, lbrace



```
private EnumSet<Token.Kind> followFwdDecl =
    EnumSet.of(void_, lbrace, eof);
```

Beispiel: Deklarationen


```
DeclPart    = { ForwardDecl } "{" Body "}" .
ForwardDecl = "void" ident "(" ")" ";" .
Body        = ... .
```

```
private void DeclPart () {
  for (;;) {
    if (sym == void_) { ForwardDecl(); }
    else if (followFwdDecl.contains(sym)) { break; }
    else { recoverFwdDecl(); }
  }
  check(lbrace); Body(); check(rbrace);
}

private void recoverFwdDecl() {
  error("invalid forward declaration");
  do {
    scan();
  } while (!followFwdDecl.contains(sym));
}
```

Bsp: Fehler in *ForwardDecl* (2)


```
void p [];
{ … }
```

```
                        Erkenne DeclPart
next() → void_             Erkenne ForwardDecl
                              void_ erkannt
next() → ident                ident erkannt
next() → lbrack               ERROR: "( expected"
                                  ERROR: ") expected"
                                  ERROR: "; expected"
                              ERROR: "invalid forward decl."
next() → rpar
next() → semicolon
next() → lbrace            lbrace erkannt
next() → ...                        Erkenne Body
...                                    …
next() → rbrace            rbrace erkannt
```

LL(1)-Bedingung

- Alternativen haben verschiedene terminale Anfänge
- Linksrekursionen verboten!

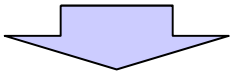
⇨ Bei Top-Down-Analyse
mit einem Vorgriffssymbol entscheiden,
welche Alternative ausgewählt werden muss.

- Abhilfen:
  - gleiche Anfänge ⇨ Faktorisieren
  - Linksrekursionen ⇨ Umwandlung in Iteration

Regel *Statement*


```
Statement
= Assignment
| ProcedureCall
| Increment | Decrement
| … .
```

gut lesbar, aber nicht LL(1), weil alle Alternativen mit **`ident`** beginnen



Abhilfe: Faktorisieren

```
Statement
= Designator
  ( AssignOp Expr          // Assignment
  | ActPars                // ProcedureCall
  | "++" | "--"            // Increment | Decrement
  ) ";"
| … .
```

Beispiel: LL(1)


```
S = a B B B | b C.      ( S = S1 | S2. , S1 = aBBB. , S2 = bC. )
B = b B | a C.          ( B = B1 | B2. , B1 = bB. , B2 = aC. )
C = S S | c.            ( C = C1 | C2. , C1 = SS. , C2 = c. )
```

$$first(S1) \cap first(S2) = \{a\} \cap \{b\} = \{\}$$
$$first(B1) \cap first(B2) = \{b\} \cap \{a\} = \{\}$$
$$first(C1) \cap first(C2) = first(S) \cap \{c\} = \{a, b\} \cap \{c\} = \{\}$$

Beispiel: LL(1)-Konflikt


```
S = a B B B | b C.   (S = S1 | S2. S1 = aBBB. S2 = bC.)
B = b B | a C d.     (B = B1 | B2. B1 = bB. B2 = aCd.)
C = [ S S | c ].     (C = C1 | C2 | C3. C1 = SS. C2 = c. C3= ε.)
```

```
FC1 = first(C1)      = first(S) = {a, b}
FC2 = first(C2)      = {c}
FC3 = first(C3)      = follow(C) =
                     = {d} ∪ follow(S) =
                     = {d} ∪ first(S) ∪ follow(C) =
                     = {d} ∪ {a, b} =
                     = {a, b, d}
```

```
FC1 ∩ FC2 = {}
FC2 ∩ FC3 = {}
FC1 ∩ FC3 = {a, b}
```

Beispiel: LL(1)-Konflikt


```
S = a B B B | b C.
B = b B | a C d.
C = [ S S | c ].
```

Beispielsatz: a a b b d a d a d

```
S = a B             B B
B =   a C         d
C =     S       S
S =     b C
C =       S …
```

UE 3: Syntaxanalyse (*Parser*)

- Abgabe
  - elektronisch bis Mi, 11.11.2008, 18:00
    - alle Java-Dateien des Compilers
  - auf Papier
    - nur Parser.java