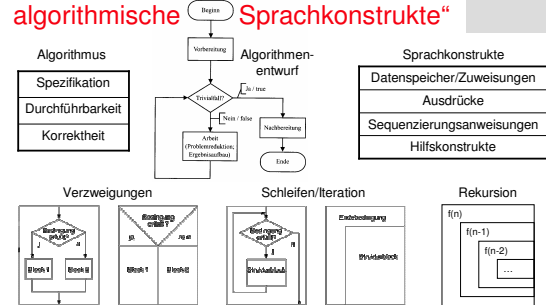


Grammatik, Bezeichner, Datentypen, Entwicklungszyklus

Thomas Röfer

Syntax/Semantik
Chomsky-Grammatiken
(Erweiterte) Backus-Naur-Form
Bezeichner
Datentypen, Literale
Entwicklungszyklus

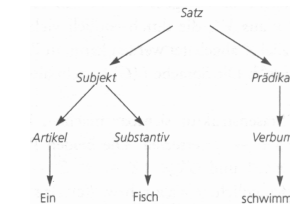
Rückblick „Algorithmen und algorithmische Sprachkonstrukte“



Syntax und Semantik

- › **Syntax**
 - › Korrekte Art und Weise, sprachliche Elemente zusammenzuführen und zu Sätzen zuzuordnen
 - › Ist durch formale *Grammatik* eindeutig beschrieben
 - › *Lexikalische Analyse*: Zerlegung in *Tokens*
 - › Tokens: Bezeichner, Literale, Schlüsselwörter, Operatoren...
- › **Semantik**
 - › legt die *Bedeutung* der Sprachkonstrukte fest
 - › Formale Beschreibung der Semantik sehr aufwendig (aber möglich)
 - › daher oft informelle Beschreibung der Semantik

Chomsky-Grammatiken

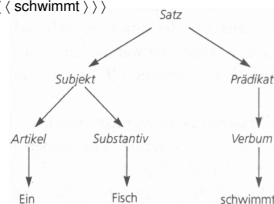


- › **Produktionen**
 - › Satz → Subjekt Prädikat
 - › Subjekt → Artikel Substantiv
 - › Subjekt → Substantiv
 - › Prädikat → Verbum
 - › Artikel → Ein
 - › Substantiv → Fisch
 - › Substantiv → Fische
 - › Verbum → schwimmt
 - › Verbum → schwimmen

- › **Begriffe**
 - › Vokabular (Nichtterminale + Terminale)
 - › Startsymbol/Ziel
 - › Sprachschatz (alle möglichen terminalen Zeichenreihen ohne Grammatik)
 - › Satzform/Phrase

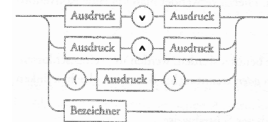
Chomsky-Grammatiken (2)

- › **Struktur**
 - › <<< Ein >> < Fisch >> << schwimmt >>>
 - › schwach äquivalent: << Ein >> << Fisch >> < schwimmt >>>
 - › strukturaquivalent: << Ein < Fisch >> << schwimmt >>>
- › **Parsen**
 - › Ist eine Zeichenreihe eine Phrase? → Zerteilung (*parsing*)
 - › Umkehr des Ableitungssystem → Reduktions-/Zerteilungssystem
- › **Chomsky-Grammatik**
 - › Ein Grammatik aus Terminalen, Nichtterminalen, Produktionen und einem Ziel



Backus-Naur-Form

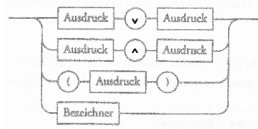
- › **BNF**
 - › Ausdruck = Ausdruck '∨' Ausdruck | Ausdruck '∧' Ausdruck | "(Ausdruck)" | Bezeichner
 - › Bezeichner = 'a' | 'b' | ... | 'z'
- › **Zerlegung 1**
 - › Ausdruck = (a ∨ b) ∧ c ∨ d
 - › Ausdruck = (a ∨ b) ∧ c Ausdruck = d
 - › Ausdruck = a ∨ b Bezeichner = c
 - › Ausdruck = a Ausdruck = b
 - › Bezeichner = a Bezeichner = b



Backus-Naur-Form

BNF

- Ausdruck = Ausdruck \vee Ausdruck | Ausdruck \wedge Ausdruck | (' Ausdruck ') | Bezeichner
- Bezeichner = 'a' | 'b' | ... | 'z'



Zerlegung 2

- Ausdruck = (a \vee b) \wedge c \vee d
- Ausdruck = (a \vee b) Ausdruck = c \vee d
- Ausdruck = a \vee b Ausdruck = c Ausdruck = d
- Ausdruck = a Ausdruck = b Bezeichner = c Bezeichner = d
- Bezeichner = a Bezeichner = b

Erweiterte Backus-Naur-Form

Bedeutung

- [...] bezeichnet einen optionalen Teil auf der rechten Seite
- (...) umschließt eine Gruppe von Zeichen
- { ... } Inhalt der Klammer wird beliebig oft wiederholt (auch 0-mal)
- | trennt Alternativen

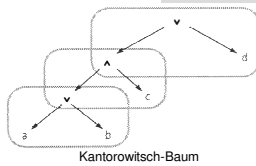
EBNF in EBNF

- Produktion = Bezeichner '=' Ausdruck
- Ausdruck = Term { '|' Term }
- Term = Faktor { '|' Faktor }
- Faktor = Bezeichner | '^' Literal '"' | '(' Ausdruck ')' | '[' Ausdruck ']' | '{' Ausdruck '}'

Zerlegung mit EBNF

EBNF

- Ausdruck = Term { '|' Term }
- Term = Faktor { '|' Faktor }
- Faktor = '(' Ausdruck ')' | Bezeichner
- Bezeichner = 'a' | 'b' | ... | 'z'



Zerlegung

- Ausdruck = (a \vee b) \wedge c \vee d
- Term = (a \vee b) \wedge c Term = d
- Faktor = (a \vee b) Faktor = c Faktor = d
- Ausdruck = a \vee b Bezeichner = c Bezeichner = d
- Term = a Term = b
- Faktor = a Faktor = b
- Bezeichner = a Bezeichner = b

Bezeichner

Schreibung

- Erstes Zeichen muss ein Buchstabe, '_' oder '\$' sein
- Alle weiteren Zeichen können Buchstaben, Ziffern, '_' oder '\$' sein
- Groß- und Kleinschreibung wird unterschieden

EBNF

- Identifizier = FirstChar { FurtherChar }
- FirstChar = '_' | '\$' | 'A' | 'B' | ... | 'Z' | 'a' | 'b' | ... | 'z' | ...
- FurtherChar = FirstChar | '0' | '1' | ... | '9'

Schlüsselwörtern (Nicht für Bezeichner verwenden)

- | | | | | |
|----------|----------|------------|--------------|-----------|
| abstract | continue | goto | package | this |
| assert | default | if | private | throw |
| boolean | do | implements | protected | throws |
| break | double | import | public | transient |
| byte | else | instanceof | return | try |
| case | extends | int | short | void |
| catch | final | interface | static | volatile |
| char | finally | long | super | while |
| class | float | native | switch | |
| const | for | new | synchronized | |

Konventionen für Bezeichner

Variablen

- Variablen beginnen mit Kleinbuchstaben: stream, name
- Mehrere Worte werden durch Großbuchstaben aneinander gefügt: thisIsAVeryLongName
- '_' und '\$' werden nicht verwendet
- Für Laufvariablen in Schleifen: i, j, k, ...

Konstanten

- Nur Großbuchstaben, Ziffern und '_' : MAX_WORDS, Math.PI

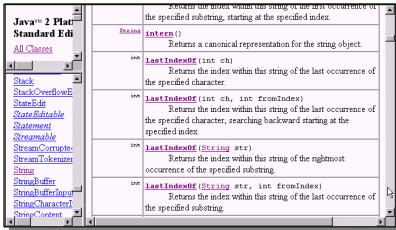
Klassen

- Genau wie Variablen, beginnen aber mit einem Großbuchstaben: System, Factorial

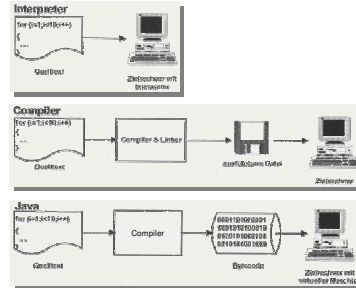
Elementare Datentypen in Java

Datentyp	Default	Speicherplatz	Wertebereich
byte	0	1 Byte (8 Bits)	-128 bis 127
short	0	2 Bytes (16 Bits)	-32768 bis 32767
int	0	4 Bytes (32 Bits)	-2147483648 bis 2147483647
long	0	8 Bytes (64 Bits)	-9223372036854775808 bis 9223372036854775807
float	0.0	4 Bytes (32 Bits)	$\pm 1.40239846E-45$ bis $\pm 3.40282347E+38$
double	0.0	8 Bytes (64 Bits)	$\pm 4.94065645841246544E-324$ bis $\pm 1.79769313486231570E+308$
boolean	false	? (min. 1 Bit)	false, true
char	'\u0000'	2 Bytes (16 Bits)	'\u0000' bis '\uFFFF'

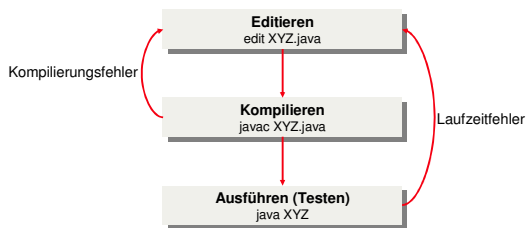
Nachschlagen in der Dokumentation



Vom Quelltext zur Ausführung



Der Entwicklungszyklus



Fehler beim Programmieren

- ▶ **Tippfehler (auch Copy and Paste)**
 - ▶ Semikolon vergessen, Klammern passen nicht...
 - ▶ Können vom Compiler entdeckt werden, müssen aber nicht
- ▶ **Strukturfehler**
 - ▶ Compiler verwendet andere Zuordnung als man denkt
- ▶ **Typfehler**
 - ▶ Werden bei Zuweisungen erkannt
 - ▶ Bleiben in Ausdrücken teilweise unerkannt
- ▶ **Fehler im Algorithmus**
 - ▶ Kommt bei Informatikern nicht vor ;-)

```
for(int i = 0; i < 10; ++i)
for(int j = 0; j < 10; ++j)
    System.out.println(i+j);

if(a == 1)
if(b == 1)
    System.out.println("ok");
else
    System.out.println("a != 1");

int i = "1";
float f = 3 / 2;
```

Fehlerbehebung

- ▶ **Kompilierungsfehler**
 - ▶ Nur den ersten Fehler beachten, die anderen könnten Folgefehler sein!
 - ▶ Fehlermeldung lesen, Quelltext bei der angegebenen Zeilennummer ansehen
 - ▶ Dokumentation zu Fehlermeldung und falschem Konstrukt lesen
- ▶ **Fehler während der Programmausführung**
 - ▶ Falls eine Fehlermeldung angezeigt wird, diese lesen und den Quelltext bei der angegebenen Zeilennummer ansehen
 - ▶ Testausgaben einbauen oder Debugger verwenden
 - ▶ Aber keine Fehler durch solche Testausgaben einbauen!
 - ▶ Verschiedene Testfälle durchspielen: Wann tritt der Fehler auf, wann nicht?

Nur ein verstandener Fehler ist ein beseitigter Fehler!

Nach Änderung des Programms kompilieren nicht vergessen!