

Praktische Informatik 1

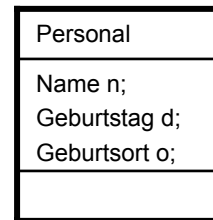
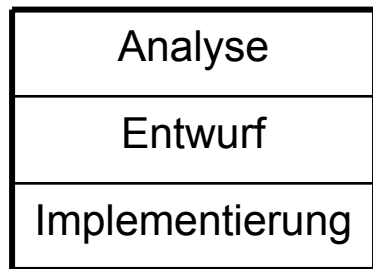
Grammatik, Java-Programme, Bezeichner, Datentypen

Thomas Röfer

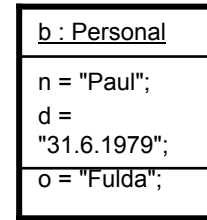
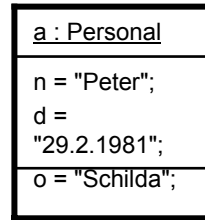
- Syntax/Semantik
- Chomsky-Grammatiken
- (Erweiterte) Backus-Naur-Form
- Eigenständige Java-Programme
- Bezeichner
- Datentypen, Literale
- Musterlösung, Übungsblatt

Rückblick „Objektorientierte Software-Konzepte und UML“

Objektorientierter Ansatz Klassen und Objekte/Instanzen

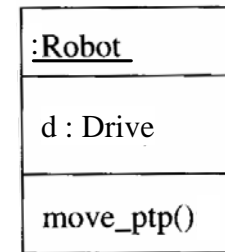


Klasse

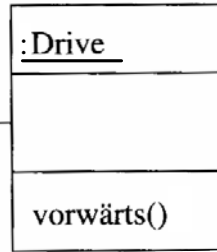


Instanzen

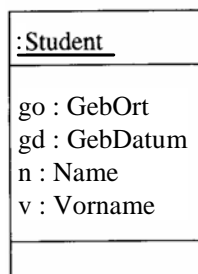
Client/Server



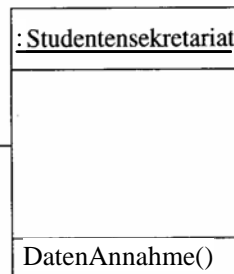
vorwärts() →



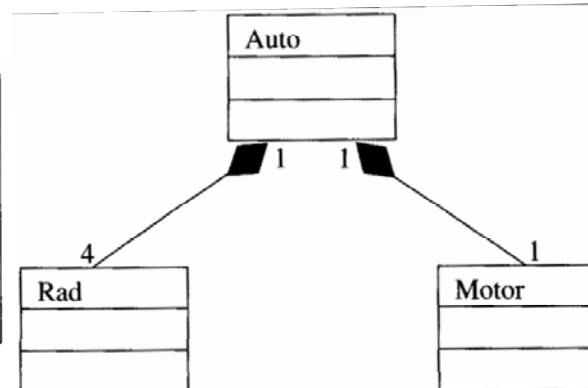
Informationsfluss/Nachrichten



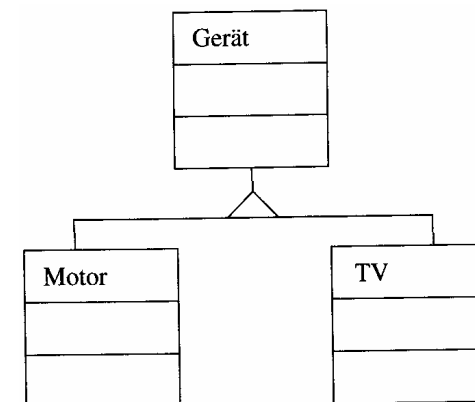
Persönliche Daten →
← Empfangsbestätigung
← Mahnung
Zahlungsbeleg →



has-a-Beziehungen



is-a-Beziehungen



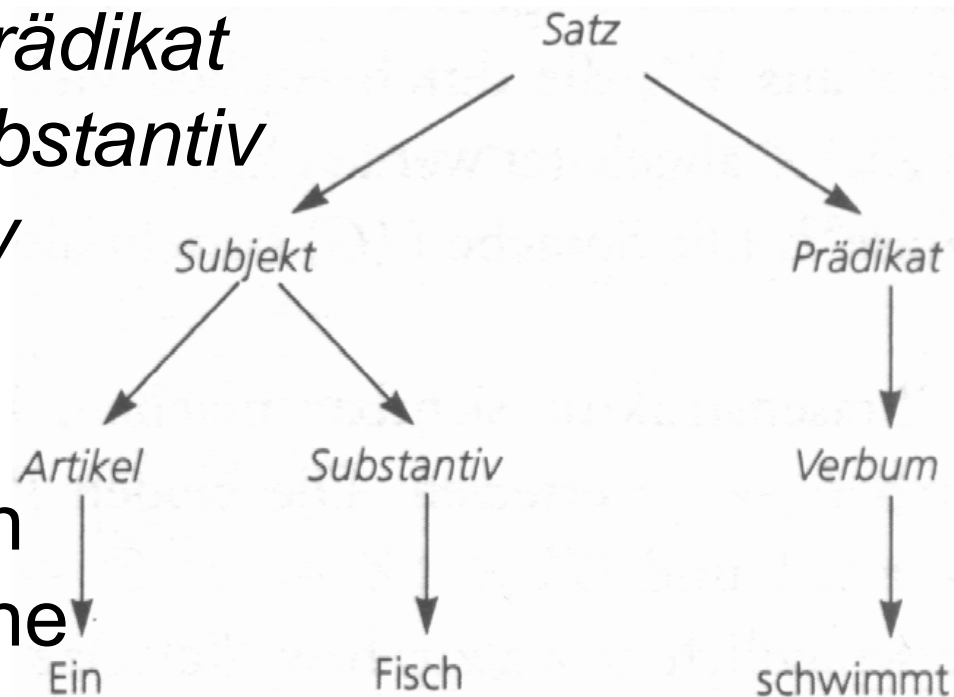
Syntax und Semantik

- 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

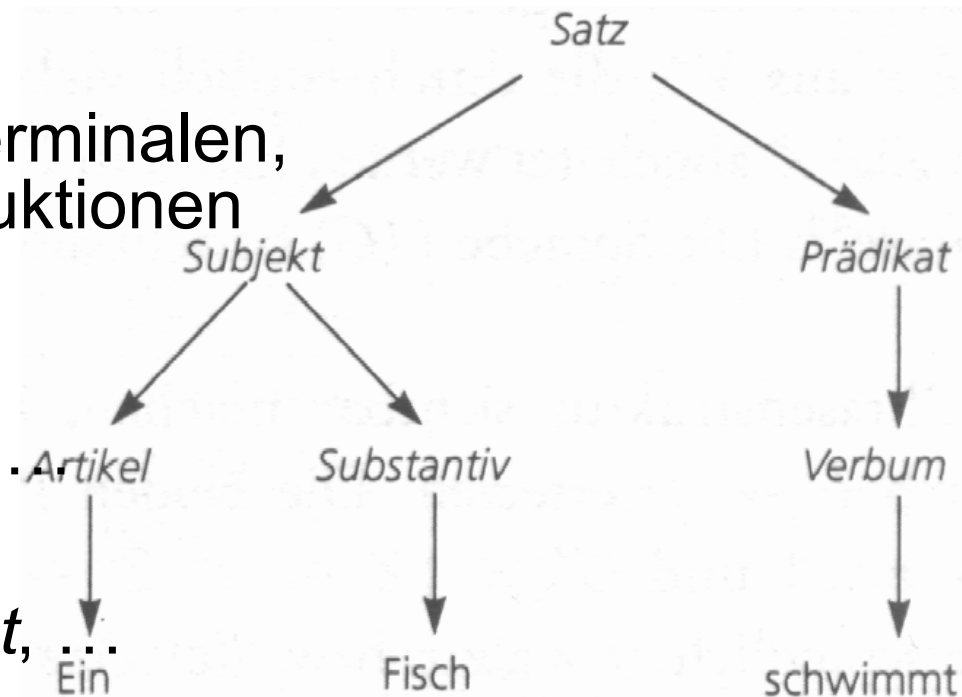
- *Satz* → *Subjekt Prädikat*
- *Subjekt* → *Artikel Substantiv*
- *Subjekt* → *Substantiv*
- *Prädikat* → *Verbum*
- *Artikel* → *Ein*
- *Substantiv* → *Fisch*
- *Substantiv* → *Fische*
- *Verbum* → *schwimmt*
- *Verbum* → *schwimmen*



Chomsky-Grammatiken

Begriffe

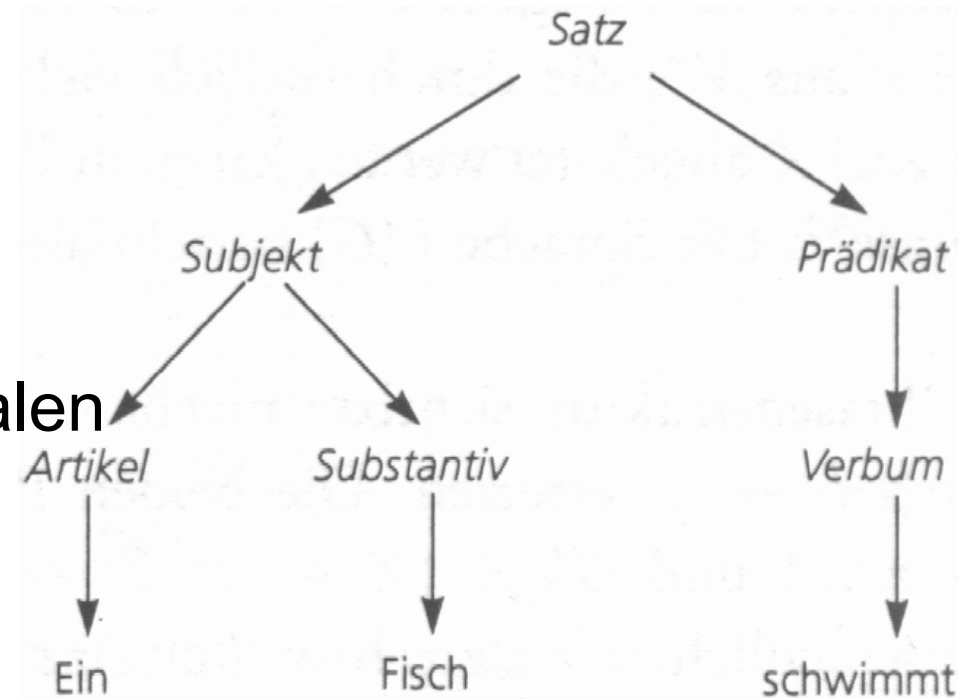
- Chomsky-Grammatik
 - Ein Grammatik aus Terminalen, Nichtterminalen, Produktionen und einem Ziel
- Terminale
 - Ein, Fisch, schwimmt,
- Nichtterminale
 - *Satz*, *Subjekt*, *Prädikat*,
- Startsymbol/Ziel
 - *Satz*



Chomsky-Grammatiken

Begriffe

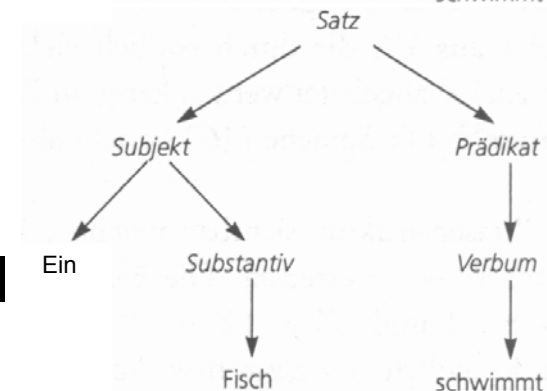
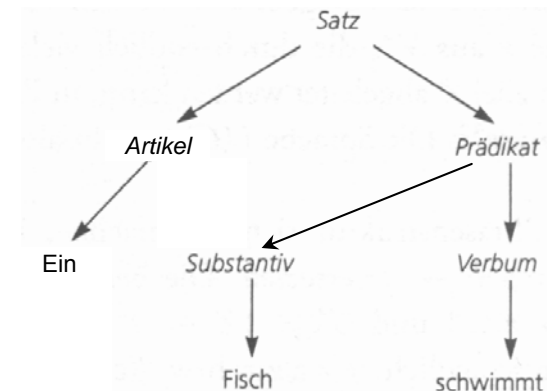
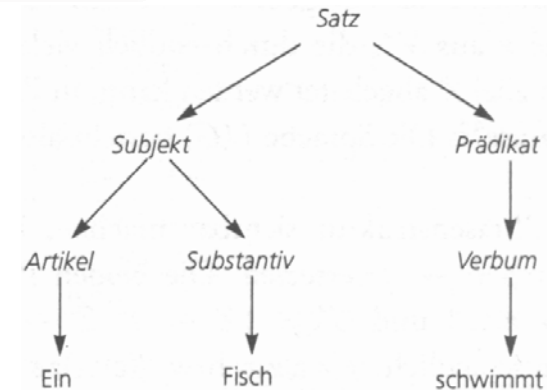
- Vokabular
 - Nichtterminale + Terminale
- Sprachschatz
 - Alle möglichen terminalen Zeichenreihen ohne Grammatik
- Satzform/Phrase
 - Ein Fisch schwimmt, Ein Fische schwimmt, ...



Chomsky-Grammatiken

Struktur

- Beispiel
 - $\langle \langle \langle \text{Ein} \rangle \langle \text{Fisch} \rangle \rangle \langle \langle \text{schwimmt} \rangle \rangle \rangle$
- schwach äquivalent
 - Gleicher Sprachschatz
 - Unterschiedliche Struktur
 - $\langle \langle \text{Ein} \rangle \langle \langle \text{Fisch} \rangle \langle \text{schwimmt} \rangle \rangle \rangle$
- Strukturäquivalent
 - Gleicher Sprachschatz
 - Einfügen/Weglassen von Nichtterminal
 - $\langle \langle \text{Ein} \langle \text{Fisch} \rangle \rangle \langle \langle \text{schwimmt} \rangle \rangle \rangle$



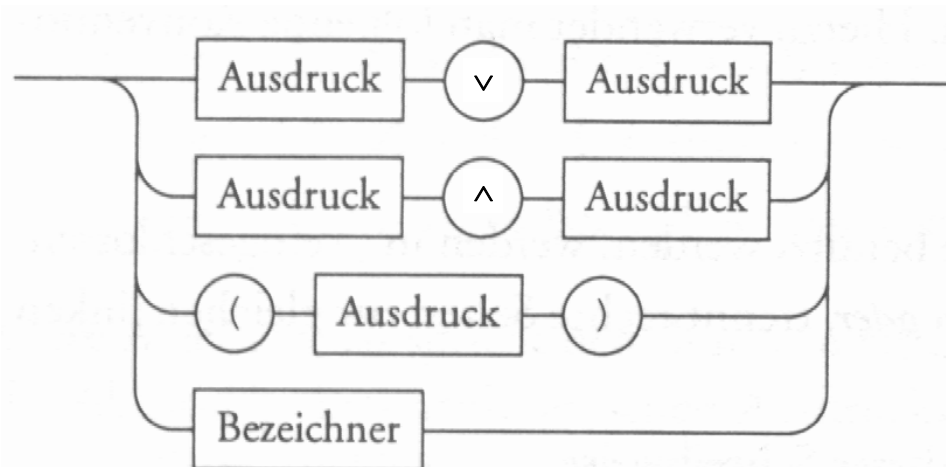
Chomsky-Grammatiken

Parsen

- Ist eine Zeichenreihe eine Phrase?
 - Zerteilung (parsing)
- Umkehr des Ableitungssystems
 - Reduktions-/Zerteilungssystem
- Wie parst man einen Text in linearer Zeit?
 - d.h. ohne Ausprobieren?

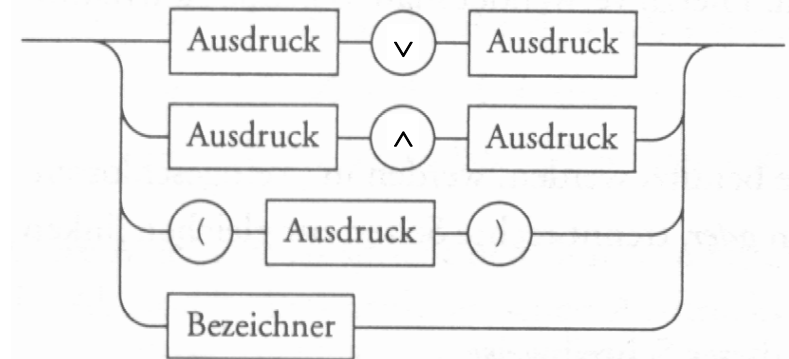
Backus-Naur-Form

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



Backus-Naur-Form

1. Zerlegung



• Ausdruck = (a \vee b) \wedge c \vee d

$$A = \underline{(a \vee b)}$$

$$A = a \vee b$$

$$A = a$$

|

$$B = a$$

$$A = b$$

$$B = b$$

$$A = c \vee \underline{d}$$

$$A = c$$

|

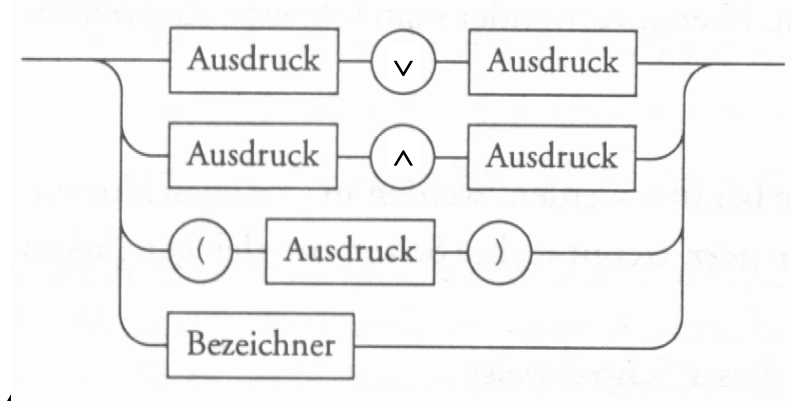
$$B = c$$

$$A = d$$

$$B = d$$

Backus-Naur-Form

2. Zerlegung



- Ausdruck = (a v b) ^ c v ~

$$A = (a \vee b) \wedge c$$

$$A = (a \vee b)$$

$$A = a \vee b$$

$$A = a$$

$$B \mid = a$$

$$A = b$$

$$B \mid = b$$

$$A = c$$

$$B = c$$

$$A = d$$

$$\mid$$

$$B = d$$

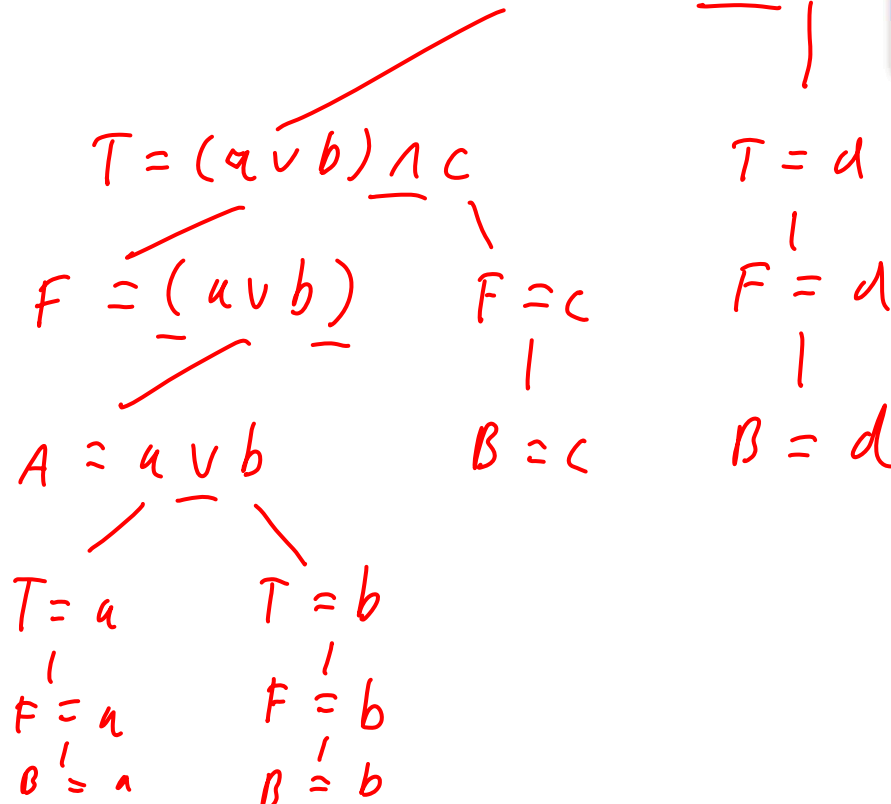
Erweiterte Backus-Naur-Form

- 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

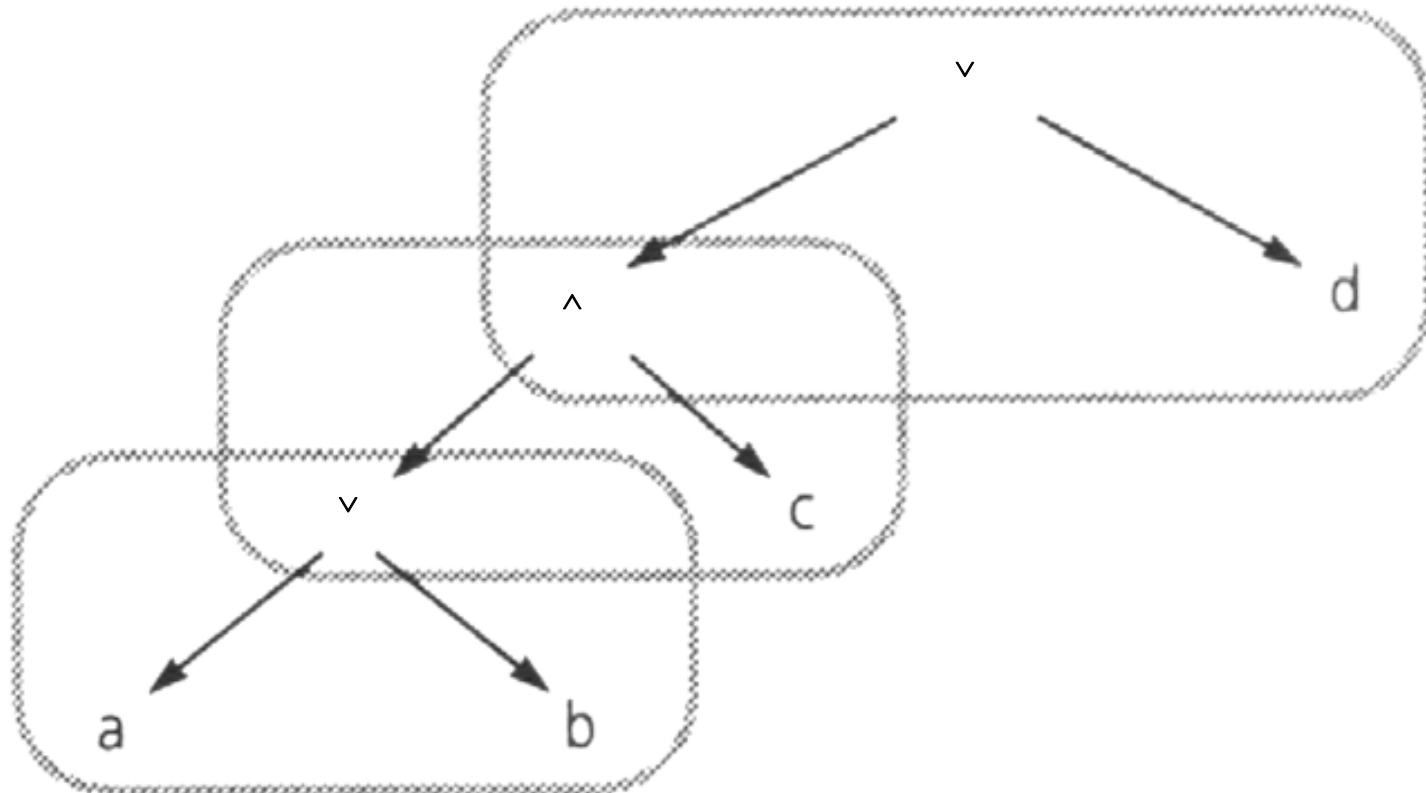
Ausdruck = Term { 'v' Term }
 Term = Faktor { '^' Faktor }
 Faktor = '(' Ausdruck ')' | Bezeichner
 Bezeichner = 'a' | 'b' | ... | 'z'

- Ausdruck = (a v b) ^ c v d



Zerlegung mit EBNF

Kantorowitsch-Baum



Die Sprache Java im Detail

- Eigenständige Java-Programme, Schlüsselwörter, Literale, Namen, Datentypen (Kap. 6.1-4)
- Variablen, Referenzen und Zuweisungen (Kap. 6.5)
- Operatoren und Ausdrücke (Kap. 6.6-7)
- Anweisungen, Verzweigungen, Schleifen und Ausnahmen (Kap. 6.8)
- Prozeduren, Funktionen, JavaDoc (Kap. 6.9)
- Vererbung (Kap. 8)
- Generisches Programmieren (Kap. 8)

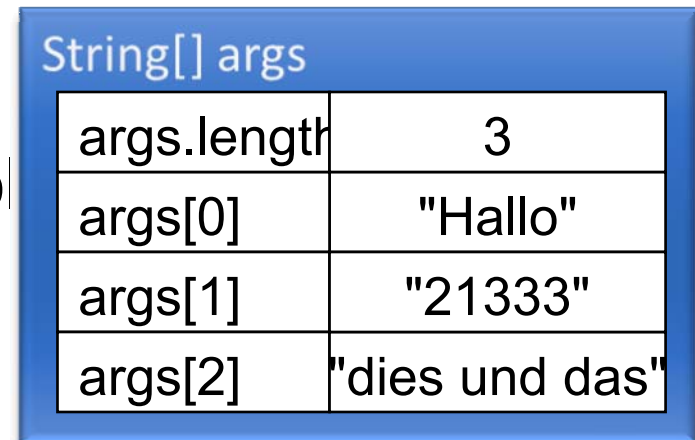
Java ohne BlueJ

- Enthält eine Klasse eine Funktion mit der Signatur

```
public static void main(String[] args)
```

- so kann diese von der Konsole aufgerufen werden
- Aufruf
 - Virtuelle Maschine java
 - Name der Klasse, die main() enthält
 - Eventuelle Parameter
 - werden als String-Array an main() übergeben

>java KlasseHallo 21333 "dies und da"



| String[] args | |
|---------------|----------------|
| args.length | 3 |
| args[0] | "Hallo" |
| args[1] | "21333" |
| args[2] | "dies und das" |

public static void main(String[] args) ...

Beispiel

```
class Greetings {  
    public static void main(String[ ] args) {  
        int i = 0;  
        while (i < args.length) {  
            if (args[i].equals("Walter")) {  
                System.out.println("Mein Gott " + args[i]);  
            } else {  
                System.out.println("Hallo " + args[i]);  
            }  
            i = i + 1;  
        }  
    }  
}
```

Java-Archiv (.jar)

- Enthält gleich mehrere Java-Klassen
 - Kann also komplettes Programm in einer einzelnen Datei enthalten
 - Ist eigentlich zip-Archiv
 - In BlueJ: Projekt|Als jar-Archiv speichern...
- Starten
 - `java -jar Archiv.jar Klasse`
- Archiv mit Manifest-Datei
 - META-INF/MANIFEST.MF
 - `java -jar Archiv.jar`

Manifest-Version: 1.0
Class-Path:
Main-Class: Greetings

Bezeichner

- Schreibung
 - Erstes Zeichen muss ein Buchstabe, '_' oder '\$' sein
 - Alle weiteren Zeichen können Buchstaben, Ziffern, '_' oder '\$' sein
 - Groß- und Kleinschreibung wird unterschieden
- EBNF
 - Identifier = FirstChar { FurtherChar }
 - FirstChar = '_' | '\$' | 'A' | 'B' | ... | 'Z' | 'a' | 'b' | ... | 'z' | ...
 - FurtherChar = FirstChar | '0' | '1' | ... | '9'

Schlüsselwörter

- Nicht für Bezeichner verwenden

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

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, ...

Konventionen für Bezeichner

- Konstanten
 - Nur Großbuchstaben, Ziffern und '_'
 - Beispiele: MAX_WORDS, Math.PI
- Klassen
 - Genau wie Variablen, beginnen aber mit einem Großbuchstaben
 - Beispiele: System, Factorial

Elementare Datentypen in Java

| Datentyp | Standard | 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) | $\pm 1.40239846\text{E}-45$ bis $\pm 3.40282347\text{E}+38$ |
| • double | 0.0 | 8 Bytes (64 Bits) | $\pm 4.94065645841246544\text{E}-324$ bis $\pm 1.79769313486231570\text{E}+308$ |
| • boolean | | false | ? (min. 1 Bit) false, true |
| • char | '\u0000' | 2 Bytes (16 Bits) | '\u0000' bis '\uFFFF' |

Ganzzahlige Datentypen

• Zahlssysteme

- Dezimal, es gibt die Ziffern 0 bis 9
- Binär, nur die Ziffern 0 und 1
- Oktal, nur die Ziffern 0 bis 7
- Hexadezimal, die Ziffern 0 bis 9 und A bis F

$$\begin{array}{|c|c|} \hline 4 & 2 \\ \hline \end{array} = 4 \cdot 10 + 2$$

$$\begin{array}{|c|c|c|c|c|c|} \hline 1 & 0 & 1 & 0 & 1 & 0 \\ \hline \end{array} = 32 + 8 + 2$$

$$\begin{array}{|c|c|} \hline 5 & 2 \\ \hline \end{array} = 5 \cdot 8 + 2$$

$$\begin{array}{|c|c|} \hline 2 & A \\ \hline \end{array} = 2 \cdot 16 + 10$$

• Literale

- Dezimal: 42, +42, -42, 42L, 42l
- Oktal: 052, +052, -052, 052L, 052l
- Hexadezimal: 0x2A, +0x2a, -0x2A, 0x2aL, 0x2Al

Ein L am Ende
bedeutet *long*

Fließkommazahlen

- Format

- Mantisse

- Exponent

- Wert = Mantisse \times
 10^{Exponent}



- Literale

- float: 0f, .382f, 3.1415F, -3.1415f, 17E7f, 17e-7f, 17E+7F
 - double: 0.0, .0392039029303, -3.141592653589d,

Ein *F* am Ende bedeutet *float*, ein *D* *double*. Wird nichts angegeben, ist der Typ *double*, wenn ein Dezimalpunkt oder *E* enthalten ist, sonst *int*

Zeichen Literale

- Normal: 'a', 'b', 'c'
- Wagenrücklauf: '\r'
- Zeilenvorschub: '\n'
- Seitenvorschub: '\f'
- Tabulatorsprung: '\t'
- Backspace: '\b'
- Hochkomma: '\"'
- Backslash: '\\'
- Unicode Zeichen: '\u12ab'

\ = Escape-Zeichen

Java-Quelltexte werden im Unicode verarbeitet. An jeder Stelle kann \uXXXX stehen.

```
f\u006fr (int i = 0; i < 10; ++i)
```

||

```
for (int i = 0; i < 10; ++i)
```