



# Grammatik, Bezeichner, Datentypen, Entwicklungszyklus

Thomas Röfer

Syntax/Semantik

Chomsky-Grammatiken

(Erweiterte) Backus-Naur-Form

Bezeichner

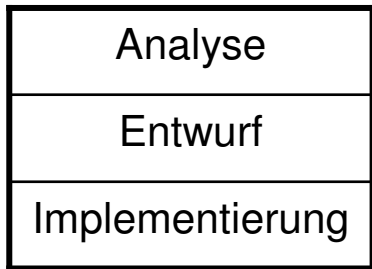
Datentypen, Literale

Entwicklungszyklus

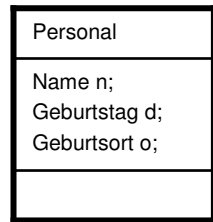


# Rückblick „Objektorientierte Software-Konzepte und UML“

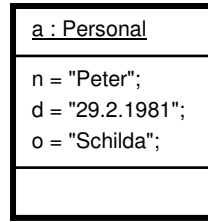
Objektorientierter Ansatz



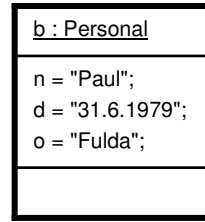
Klassen und Objekte/Instanzen



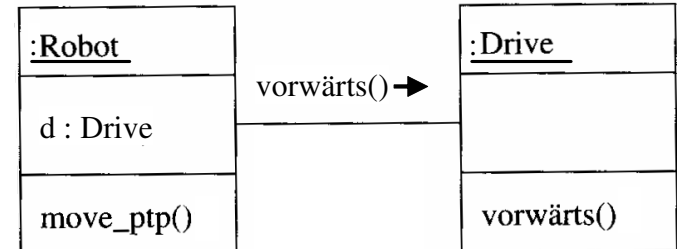
Klasse



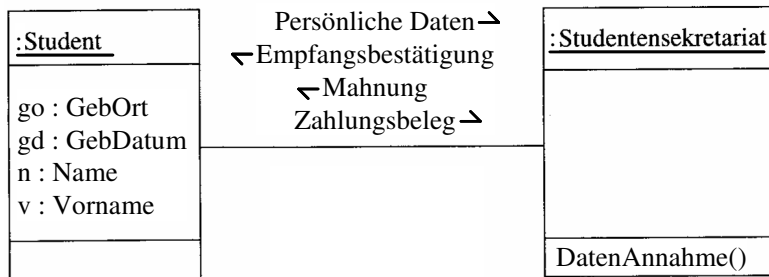
Instanzen



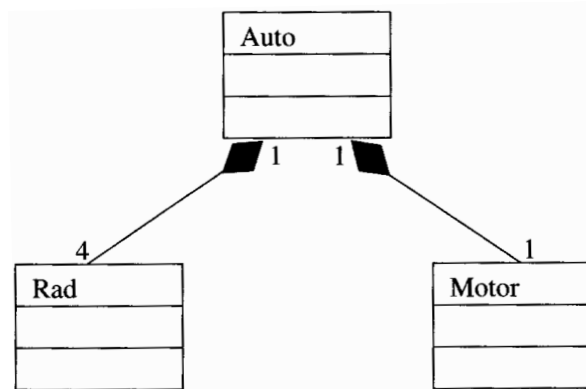
Client/Server



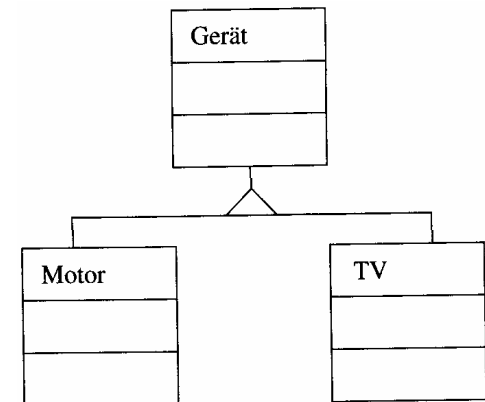
Informationsfluss/Nachrichten



has-a-Beziehungen



is-a-Beziehungen





# Syntax und Semantik

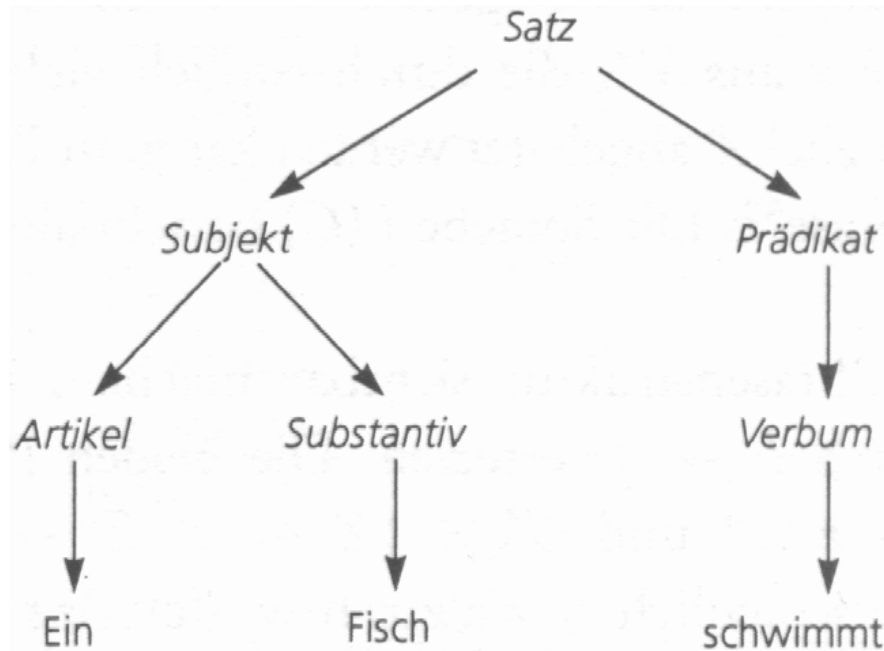
## ▶ **Syntax**

- ▶ 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



## ► Begriffe

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

## ► Produktionen

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

## Chomsky-Grammatiken (2)

### ▶ Struktur

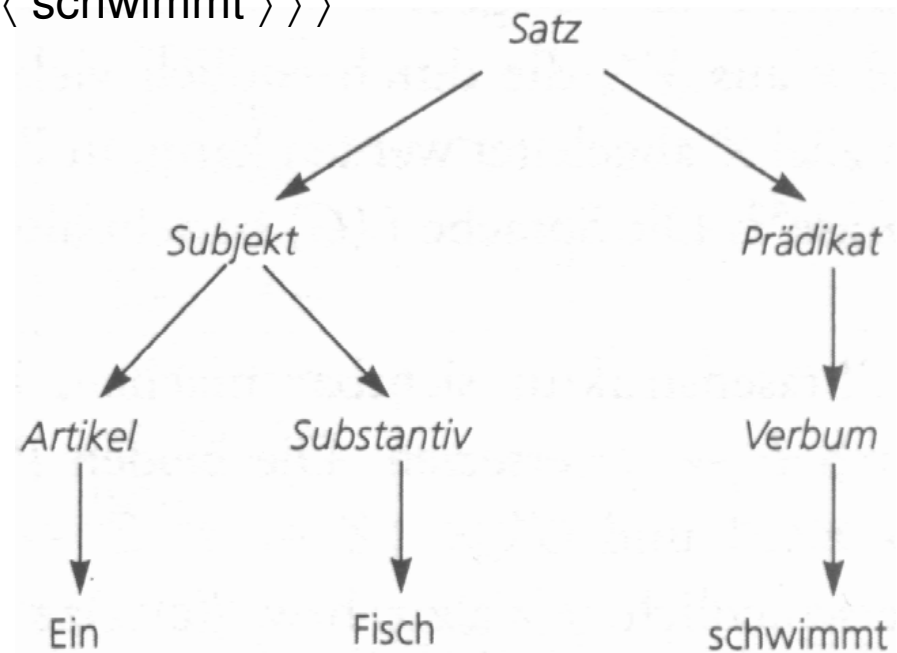
- ▶  $\langle \langle \langle \text{Ein} \rangle \langle \text{Fisch} \rangle \rangle \langle \langle \text{schwimmt} \rangle \rangle \rangle$
- ▶ schwach äquivalent:  $\langle \langle \text{Ein} \rangle \langle \langle \text{Fisch} \rangle \langle \text{schwimmt} \rangle \rangle \rangle$
- ▶ strukturäquivalent:  $\langle \langle \text{Ein} \langle \text{Fisch} \rangle \rangle \langle \langle \text{schwimmt} \rangle \rangle \rangle$

### ▶ Parsen

- ▶ Ist eine Zeichenreihe eine Phrase?  
→ Zerteilung (*parsing*)
- ▶ Umkehr des Ableitungssystems  
→ Reduktions-/Zerteilungssystem

### ▶ Chomsky-Grammatik

- ▶ Ein Grammatik aus Terminalen, Nichtterminalen, Produktionen und einem Ziel









# 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

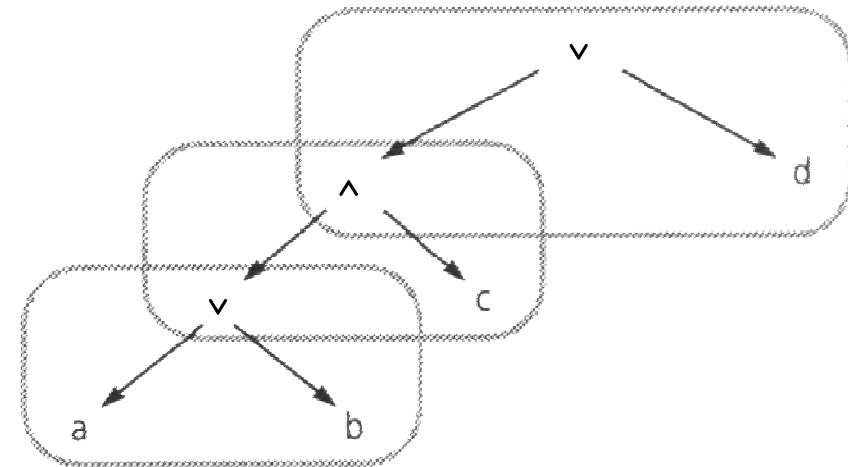
## ▶ EBNF in EBNF

- ▶ 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'



Kantorowitsch-Baum

## ▶ Zerlegung

- |   |            |           |                            |     |            |     |
|---|------------|-----------|----------------------------|-----|------------|-----|
| ▶ |            |           | Ausdruck = (a ∨ b) ∧ c ∨ d |     |            |     |
| ▶ |            |           | Term = (a ∨ b) ∧ c         |     | Term       | = d |
| ▶ | Faktor     | = (a ∨ b) | Faktor                     | = c | Faktor     | = d |
| ▶ | Ausdruck   | = a ∨ 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**

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

## ▶ **Schlüsselwörtern (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:
- ▶ Mehrere Worte werden durch Großbuchstaben aneinander gefügt:
- ▶ '\_' und '\$' werden nicht verwendet
- ▶ Für Laufvariablen in Schleifen:

stream, name

thisIsAVeryLongName

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'



# 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

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

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

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

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

## ▶ Literale

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

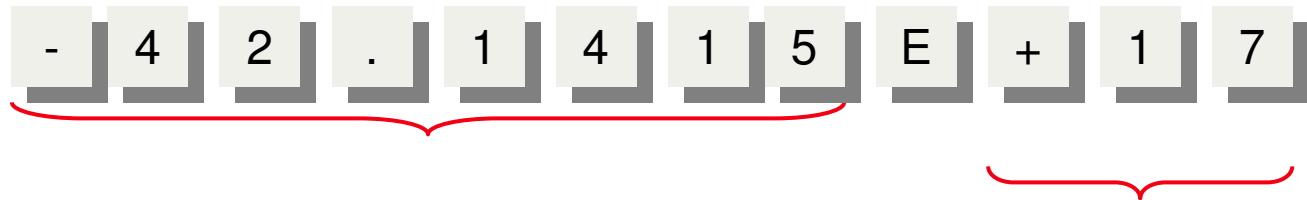
Ein L am Ende bedeutet *long*



# Fließkommazahlen

## ▶ Format

- ▶ Mantisse
- ▶ Exponent
- ▶ Wert = Mantisse  $\times 10^{\text{Exponent}}$



## ▶ Literale

- ▶ Float: 0.0f, .382f, 3.1415F, -3.1415f, 17E7f, 17e-7f, 17E+7F
- ▶ Double: 0.0, .0392039029303, -3.141592653589d, 1E+100

Ein *F* am Ende bedeutet *float*, ein *D* *double*. Wird nichts angegeben, ist der Typ *double*



# Zeichen

## ▶ Literale

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

\ = Escape-Zeichen

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

||

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

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



# Zeichenketten

- ▶ **String ist eine Klasse, kein Basistyp**
  - ▶ Daher kann man Ausdrücke schreiben, wie z.B. `s.equals("Hallo")`
- ▶ **Literale**
  - ▶ "Hallo", "wie geht's?"
  - ▶ "Ich sage \"Mir geht's gut\"" → *Ich sage "Mir geht's gut"*
- ▶ **Beispiele**
  - ▶ `System.out.println("\"DM\"\\t\"Euro\"\\n1\\t0,51");`
  - ▶ "DM"            "Euro"  
1                0,51

In String-Literalen können alle Escape-Sequenzen aus char-Literalen verwendet werden



# Typkonvertierung

## ▶ Automatisch

- ▶ byte → short → int → long → float → double
- ▶ char → int

## ▶ Manuell

- ▶ byte b; short s; int i; long l; float f = 1.5; double d = -1.5; char c;
- ▶ b = (byte) f; (b == 1)
- ▶ f = (float) 178.2
- ▶ l = (int) d; (l == -2)
- ▶ c = (char) 32

Bei der Typumwandlung von double/float in einen Ganzzahltyp wird immer abgerundet

## ▶ Durch Funktionen

- ▶ String s = Integer.toString(i);
- ▶ i = Integer.parseInt(s);
- ▶ d = Double.parseDouble(s);



# Wrapper-Klassen

▶ Datentyp	Wrapper
▶ byte	Byte
▶ short	Short
▶ int	Integer
▶ long	Long
▶ float	Float
▶ double	Double
▶ boolean	Boolean
▶ char	Character

## ▶ Generell

- ▶ *Wrapper.MAX\_VALUE*
  - ▶ z.B. *Byte.MAX\_VALUE*
- ▶ *Wrapper.MIN\_VALUE*
- ▶ *Wrapper.parseWrap(String s)*
  - ▶ z.B. *Integer.parseInt("1234");*
- ▶ *Wrapper.toString(type t)*
  - ▶ z.B. *Integer.toString(1234);*

## ▶ Float und Double

- ▶ *Wrapper.NEGATIVE\_INFINITY*
  - ▶ z.B. *Double.NEGATIVE\_INFINITY*
- ▶ *Wrapper.POSITIVE\_INFINITY*
- ▶ *Wrapper.NaN*



# Autoboxing und -unboxing

## ▶ Autoboxing

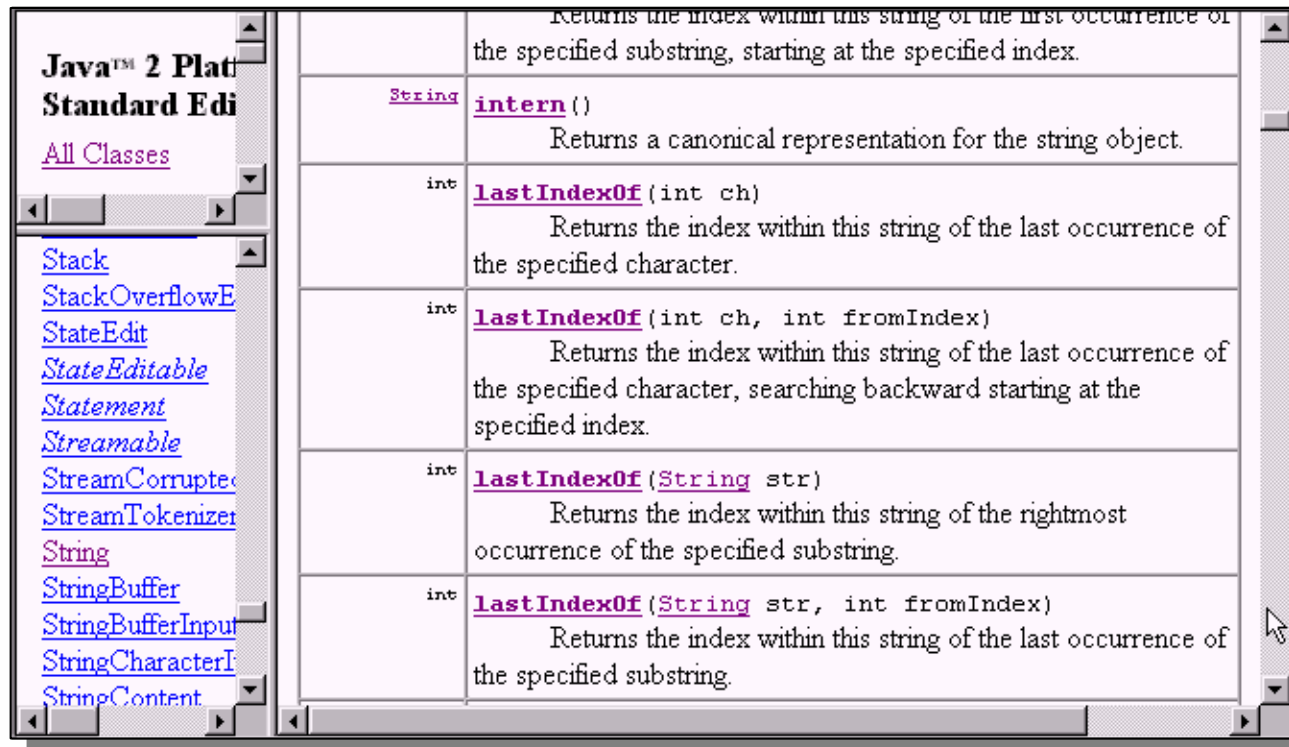
- ▶ Java wandelt automatisch einen Wert eines Basisdatentypen in ein Objekt der entsprechenden Wrapper-Klasse um
- ▶ Dadurch lassen sich Werte von Basisdatentypen wie Objekte behandeln (z.B. in vordefinierte sog. *Collections* einfügen)
  - ▶ `java.util.Vector<Integer> v = new java.util.Vector<Integer>();`
  - ▶ `v.add(17); // entspricht v.add(new Integer(17));`
- ▶ Funktioniert nur für Methodenparameter
  - ▶ `5.toString(); // Geht nicht!`

## ▶ Autounboxing

- ▶ Java wandelt automatisch ein Objekt einer Wrapper-Klasse in einen Wert des entsprechenden Basisdatentypen um
  - ▶ `int i = v.get(0) + 4; // Signatur von get ist: Integer get(int index)`
  - ▶ `// entspricht int i = v.get(0).intValue() + 4;`

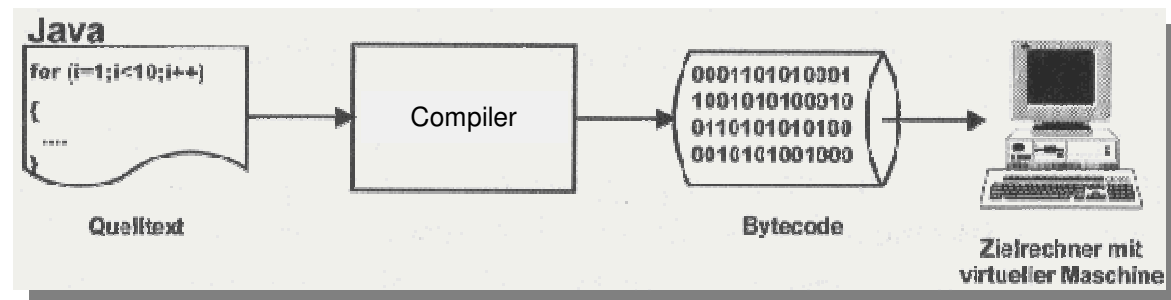
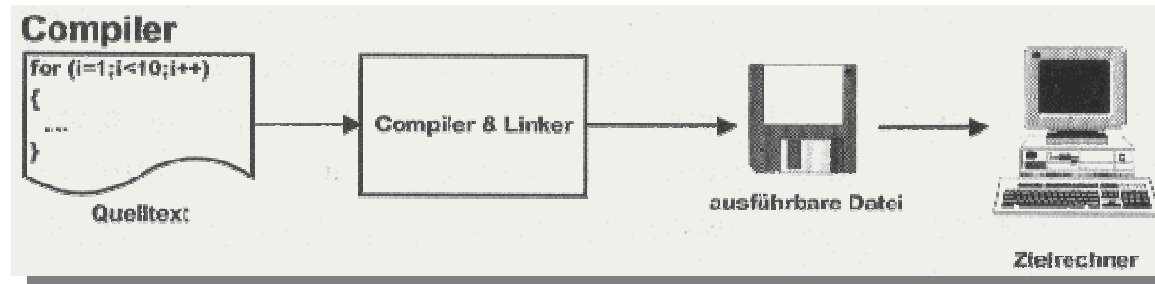
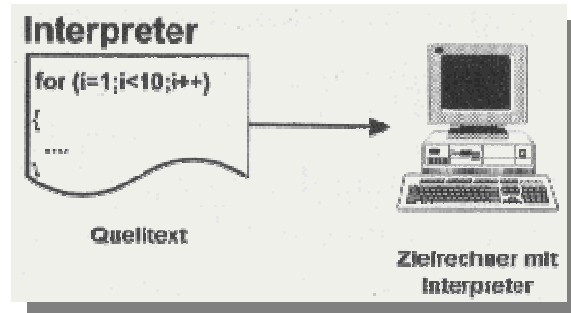


# Nachschlagen in der Dokumentation



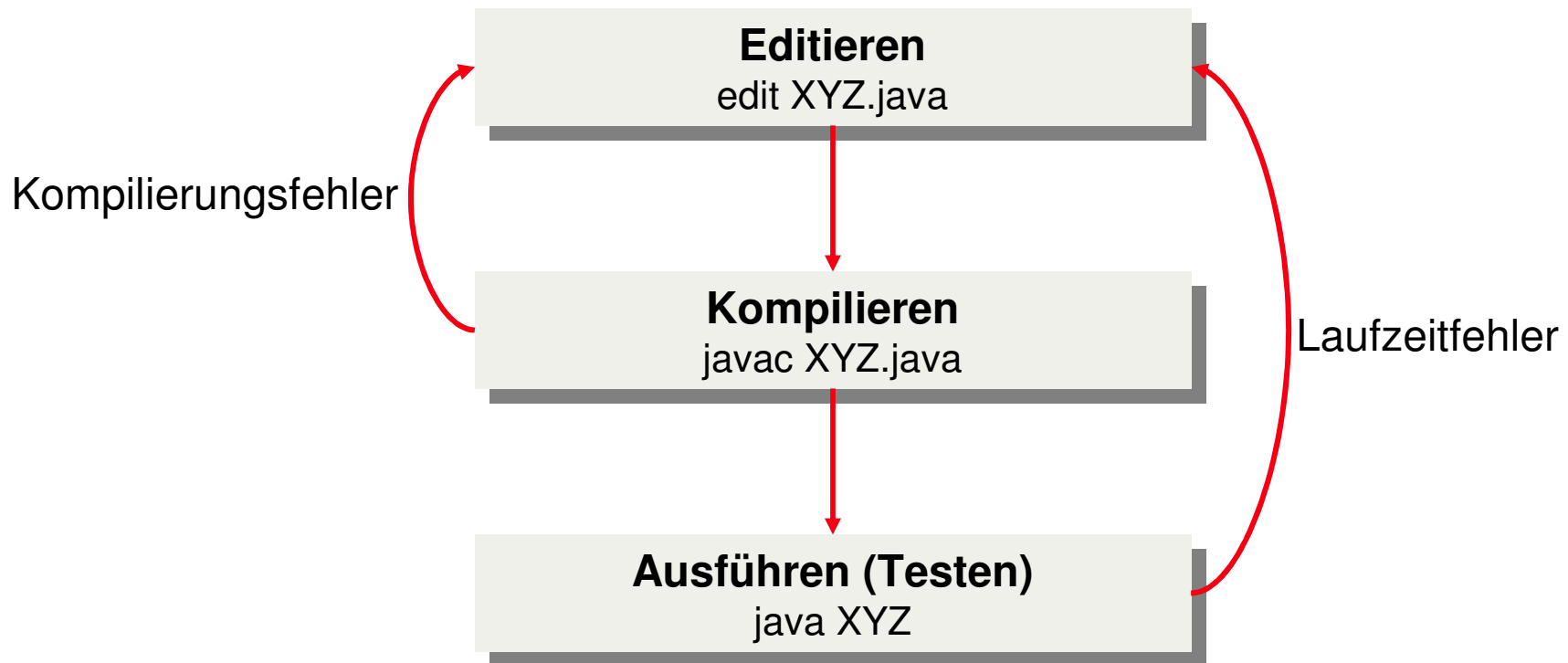


# Vom Quelltext zur Ausführung





# 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; ++i)
    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!**