

# 1. Einführung

GPS-1-1

Themen dieses Kapitels:

- 1.1. Zeitliche Einordnung, Klassifikation von Programmiersprachen
- 1.2. Implementierung von Programmiersprachen
- 1.3. Dokumente zu Programmiersprachen
- 1.4. Vier Ebenen der Spracheigenschaften

© 2009 bei Prof. Dr. Uwe Kastens

## Vorlesung Grundlagen der Programmiersprachen SS 2015 / Folie 101

**Ziele:**

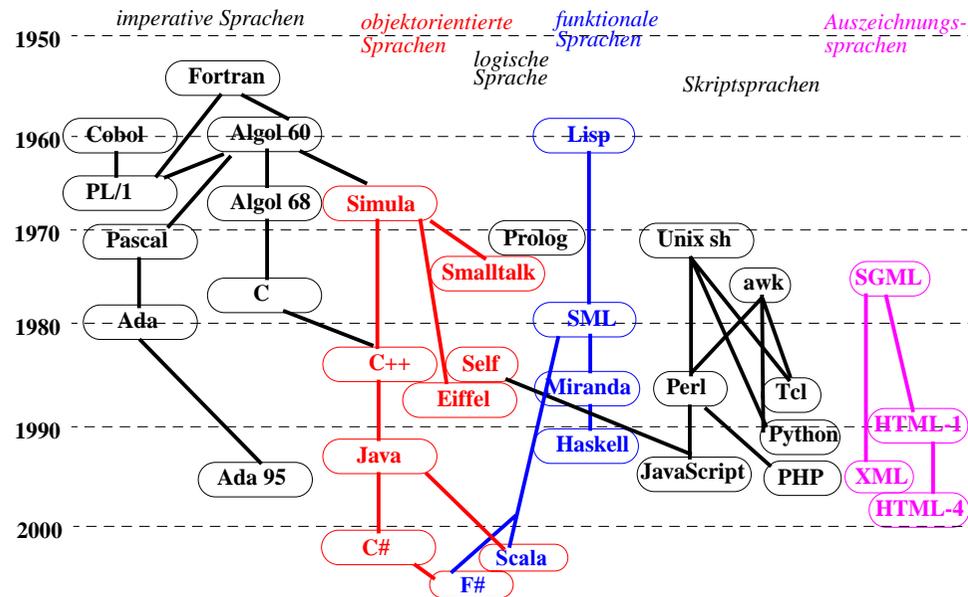
Übersicht zu diesem Kapitel

**in der Vorlesung:**

Erläuterungen dazu

## 1.1 Zeitliche Einordnung, Klassifikation von Programmiersprachen

GPS-1-2



nach [D. A. Watt: Programmiersprachen, Hanser, 1996; Seite 5] und [Computer Language History <http://www.levenez.com/lang>]

© 2015 bei Prof. Dr. Uwe Kastens

## Vorlesung Grundlagen der Programmiersprachen SS 2015 / Folie 102

**Ziele:**

Sprachen zeitlich einordnen und klassifizieren

**in der Vorlesung:**

Kommentare zur zeitlichen Entwicklung.

Verwandschaft zwischen Sprachen:

- Notation: C, C++, Java, C#, JavaScript, PHP;
- gleiche zentrale Konzepte, wie Datentypen, Objektorientierung;
- Teilsprache: Algol 60 ist Teilsprache von Simula, C von C++;
- gleiches Anwendungsgebiet: z. B. Fortran und Algol 60 für numerische Berechnungen in wissenschaftlich-technischen Anwendungen

**nachlesen:**

Text dazu im Buch von D. A. Watt

**Übungsaufgaben:**

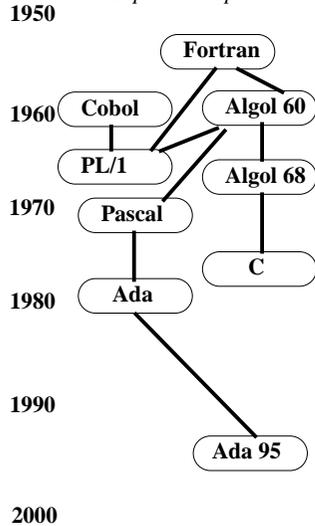
**Verständnisfragen:**

In welcher Weise können Programmiersprachen miteinander verwandt sein?

## Klassifikation: Imperative Programmiersprachen

GPS-1-3a

imperative Sprachen



### charakteristische Eigenschaften:

- Variable mit Zuweisungen, veränderbarer Programmzustand,
- Ablaufstrukturen (Schleifen, bedingte Anweisungen, Anweisungsfolgen)
- Funktionen, Prozeduren
- implementiert durch Übersetzer

### Ziele:

Charakteristika imperativer Sprachen kennenlernen

### Verständnisfragen:

Ordnen Sie die Beispiele von Folie 104 ein.

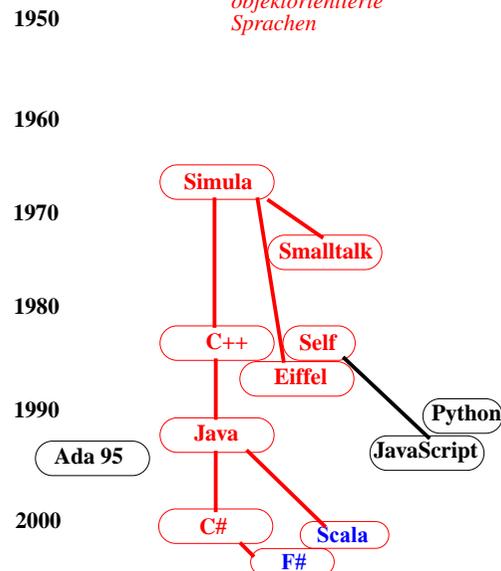
© 2015 bei Prof. Dr. Uwe Kastens

nach [D. A. Watt: Programmiersprachen, Hanser, 1996; Seite 5]  
[Computer Language History <http://www.levenez.com/lang>]

## Klassifikation: objektorientierte Programmiersprachen

GPS-1-3b

objektorientierte Sprachen



### charakteristische Eigenschaften:

- Klassen mit Methoden und Attributen, Objekte zu Klassen
- Vererbungsrelation zwischen Klassen
- Typen: objektorientierte Polymorphie: Objekt einer Unterklasse kann verwendet werden, wo ein Objekt der Oberklasse benötigt wird
- dynamische Methodenbindung
- Self und JavaScript haben keine Klassen; Vererbung zwischen Objekten
- Fast alle oo Sprachen haben auch Eigenschaften imperativer Sprachen
- implementiert: Übersetzer: Simula, C++, Eiffel, Ada Übersetzer + VM: Smalltalk, Java, C# Interpretierer: Self, Python, JavaScript

### Ziele:

Charakteristika objektorientierter Sprachen kennenlernen

### Verständnisfragen:

Ordnen Sie die Beispiele von Folie 104 ein.

© 2015 bei Prof. Dr. Uwe Kastens

nach [D. A. Watt: Programmiersprachen, Hanser, 1996; Seite 5]  
[Computer Language History <http://www.levenez.com/lang>]

## Klassifikation: logische Programmiersprachen

1950

*logische Sprache*

1960

1970

Prolog

1980

1990

2000

### charakteristische Eigenschaften:

#### Prädikatenlogik als Grundlage

Deklarative Programme ohne Ablaufstrukturen, bestehen aus Regeln, Fakten und Anfragen

Variable ohne Zuweisungen, erhalten Werte durch Termersetzung und Unifikation

keine Zustandsänderungen  
keine Seiteneffekte

keine Typen

implementiert durch Interpretierer

nach [D. A. Watt: Programmiersprachen, Hanser, 1996; Seite 5]  
[Computer Language History <http://www.levenez.com/lang>]

## Vorlesung Grundlagen der Programmiersprachen SS 2015 / Folie 103c

### Ziele:

Charakteristika logischer Sprachen kennenlernen

### Verständnisfragen:

Ordnen Sie die Beispiele von Folie 104 ein.

## Klassifikation: funktionale Programmiersprachen

1950

*funktionale Sprachen*

1960

Lisp

1970

1980

SML

Miranda

Haskell

1990

2000

Scala

F#

### charakteristische Eigenschaften:

rekursive Funktionen,  
Funktionen höherer Ordnung  
d.h. Funktionen als Parameter oder als Ergebnis

Deklarative Programme ohne Ablaufstrukturen;  
Funktionen und bedingte Ausdrücke

Variable ohne Zuweisungen,  
erhalten Werte durch Deklaration oder  
Parameterübergabe

keine Zustandsänderung,  
keine Seiten-Effekte

Typen:  
Lisp: keine  
SML, Haskell: parametrische Polymorphie

implementiert durch  
Lisp: Interpretierer  
sonst: Übersetzer und/oder Interpretierer

nach [D. A. Watt: Programmiersprachen, Hanser, 1996; Seite 5]  
[Computer Language History <http://www.levenez.com/lang>]

## Vorlesung Grundlagen der Programmiersprachen SS 2015 / Folie 103d

### Ziele:

Charakteristika funktionaler Sprachen kennenlernen

### Verständnisfragen:

Ordnen Sie die Beispiele von Folie 104 ein.

## Klassifikation: Skriptsprachen

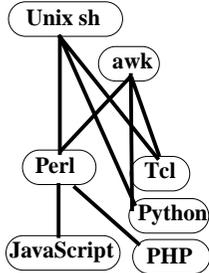
GPS-1-3e

1950

*Skriptsprachen*

1960

1970



1980

1990

2000

### charakteristische Eigenschaften:

Ziel: einfache Entwicklung einfacher Anwendungen (im Gegensatz zu allgemeiner Software-Entwicklung), insbes. Textverarbeitung und **Web-Anwendungen**

Ablaufstrukturen, Variable und **Zuweisungen wie in imperativen** Sprachen

Python, JavaScript und spätes PHP auch oo

Typen:

**dynamisch typisiert**, d.h. Typen werden bei Programmausführung bestimmt und geprüft

implementiert durch Interpretierer ggf integriert in Browser und/oder Web-Server

ggf Programme eingebettet in HTML-Texte

nach [D. A. Watt: Programmiersprachen, Hanser, 1996; Seite 5]  
[Computer Language History <http://www.levenez.com/lang>]

© 2015 bei Prof. Dr. Uwe Kastens

### Ziele:

Charakteristika von Skriptsprachen kennenlernen

### Verständnisfragen:

Ordnen Sie die Beispiele von Folie 104 ein.

## Klassifikation: Auszeichnungssprachen

GPS-1-3f

1950

*Auszeichnungssprachen*

1960

1970

SGML

1980

HTML-1

1990

XML

HTML-4

2000

### charakteristische Eigenschaften:

**Annotierung von Texten** zur Kennzeichnung der Struktur, Formatierung, Verknüpfung

Ziele: **Repräsentation strukturierter Daten** (XML), Darstellung von Texten, Hyper-Texten, Web-Seiten (HTML)

Sprachkonstrukte:

Baum-strukturierte Texte, Klammerung durch *Tags*, Attribute zu Textelementen

keine Ablaufstrukturen, Variable, Datentypen zur Programmierung

Ggf. werden Programmstücke in Skriptsprachen als spezielle Textelemente eingebettet und beim Verarbeiten des annotierten Textes ausgeführt.

nach [D. A. Watt: Programmiersprachen, Hanser, 1996; Seite 5]  
[Computer Language History <http://www.levenez.com/lang>]

© 2015 bei Prof. Dr. Uwe Kastens

### Ziele:

Charakteristika von Auszeichnungssprachen kennenlernen

### Verständnisfragen:

Ordnen Sie die Beispiele von Folie 104 ein.

## Eine Funktion in verschiedenen Sprachen

GPS-1-4

### Sprache A:

```
function Length (list: IntList): integer;
  var len: integer;
begin
  len := 0;
  while list <> nil do
  begin len := len + 1; list := list^.next end;
  Length := len
end;
```

### Sprache B:

```
int Length (Node list)
{ int len = 0;
  while (list != null)
  { len += 1; list = list.link; }
  return len;
}
```

### Sprache C:

```
fun Length list =
  if null list then 0
  else 1 + Length (tl list);
```

### Sprache D:

```
length([], 0).
length([Head | Tail], Len):-
  length(Tail, L), Len IS L + 1.
```

© 2005 bei Prof. Dr. Uwe Kastens

## Vorlesung Grundlagen der Programmiersprachen SS 2015 / Folie 104

### Ziele:

Vergleich von Spracheigenschaften

### in der Vorlesung:

- Identifikation der Sprachen (?);
  - unterschiedliche Notation für gleiche Konstrukte, z. B. while-Schleife in A und B;
  - unterschiedliche Konstrukte für gleiche Wirkung, z. B. Funktionsergebnis bestimmen: A Zuweisung, B return-Anweisung, C Ergebnis eines bedingten Ausdrucks;
  - gleiche abstrakte Struktur, z. B. A und B;
  - unterschiedliche Typisierung, z. B. A, B statisch explizit, C statisch implizit, D typlos;
  - unterschiedliche Klassifikation (?).
- (Auflösung der (?) in der Vorlesung)

### nachlesen:

### Übungsaufgaben:

### Verständnisfragen:

- Diskutieren Sie die Beispiele nach obigen Kriterien.
- Aus welchen Sprachen stammen die Beispiele?

## Hello World in vielen Sprachen

GPS-1-5a

### COBOL

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. HELLOWORLD.
000300 DATE-WRITTEN. 02/05/96 21:04.
000400* AUTHOR BRIAN COLLINS
000500 ENVIRONMENT DIVISION.
000600 CONFIGURATION SECTION.
000700 SOURCE-COMPUTER. RM-COBOL.
000800 OBJECT-COMPUTER. RM-COBOL.
000900
001000 DATA DIVISION.
001100 FILE SECTION.
001200
100000 PROCEDURE DIVISION.
100100
100200 MAIN-LOGIC SECTION.
100300 BEGIN.
100400 DISPLAY " " LINE 1 POSITION 1 ERASE EOS.
100500 DISPLAY "HELLO, WORLD." LINE 15 POSITION 10.
100600 STOP RUN.
100700 MAIN-LOGIC-EXIT.
100800 EXIT.
```

### FORTRAN IV

```
PROGRAM HELLO
DO 10, I=1,10
PRINT *,'Hello World'
10 CONTINUE
STOP
END
```

### Pascal

```
Program Hello (Input, Output);
Begin
  repeat
    writeln('Hello World!')
  until 1=2;
End.
```

### C

```
main()
{ for(;;)
  { printf ("Hello World!\n");
  }
}
```

### Perl

```
print "Hello, World!\n" while (1);
```

### Java

```
class HelloWorld {
  public static void main (String args[]) {
    for (;;) {
      System.out.print("HelloWorld");
    }
  }
}
```

© 2011 bei Prof. Dr. Uwe Kastens

## Vorlesung Grundlagen der Programmiersprachen SS 2015 / Folie 105a

### Ziele:

Eindruck von unterschiedlichen Sprachen

### in der Vorlesung:

Identität der Sprachen und Hinweise auf einige Eigenschaften

## Hello World in vielen Sprachen

GPS-1-5b

### Prolog

```
hello :-  
  printstring("HELLO WORLD!!!!").  
  printstring([]).  
  printstring([H|T]) :- put(H), printstring(T).
```

### Lisp

```
(DEFUN HELLO-WORLD ()  
  (PRINT (LIST ,HELLO ,WORLD)))
```

### SQL

```
CREATE TABLE HELLO (HELLO CHAR(12))  
UPDATE HELLO  
SET HELLO = 'HELLO WORLD!'  
SELECT * FROM HELLO
```

### HTML

```
<HTML>  
<HEAD>  
<TITLE>Hello, World Page!</TITLE>  
</HEAD>  
<BODY>  
Hello, World!  
</BODY>  
</HTML>
```

### Make

```
default:  
  echo "Hello, World!"  
  make
```

### Bourne Shell (Unix)

```
while (/bin/true)  
do  
  echo "Hello, World!"  
done
```

### LaTeX

```
\documentclass{article}  
\begin{document}  
\begin{center}  
\huge{HELLO WORLD}  
\end{center}  
\end{document}
```

### PostScript

```
/Font /Helvetica-Bold findfont def  
/FontSize 12 def  
Font FontSize scalefont setfont  
{newpath 0 0 moveto (Hello, World!) show showpage} loop
```

© 2011 bei Prof. Dr. Uwe Kastens

## Vorlesung Grundlagen der Programmiersprachen SS 2015 / Folie 105b

### Ziele:

Eindruck von unterschiedlichen Sprachen

### in der Vorlesung:

Identität der Sprachen und Hinweise auf einige Eigenschaften

## Sprachen für spezielle Anwendungen

GPS-1-6

- **technisch/wissenschaftlich:** FORTRAN, Algol-60
- **kaufmännisch** RPG, COBOL
- **Datenbanken:** SQL
- **Vektor-, Matrixrechnungen:** APL, Lotus-1-2-3
- **Textsatz:** TeX, LaTeX, PostScript
- **Textverarbeitung, Pattern Matching:** SNOBOL, ICON, awk, Perl
- **Skriptsprachen:** DOS-, UNIX-Shell, TCL, Perl, PHP
- **Auszeichnung (Markup):** HTML, XML
- **Spezifikationssprachen:** SETL, Z  
VHDL  
UML  
EBNF  
Allgemeine Spezifikationen von Systemen  
Spezifikationen von Hardware  
Spezifikationen von Software  
Spezifikation von KFGn, Parsern

© 2011 bei Prof. Dr. Uwe Kastens

## Vorlesung Grundlagen der Programmiersprachen SS 2015 / Folie 106

### Ziele:

Es gibt nicht nur Programmiersprachen!

### in der Vorlesung:

Erläuterungen dazu; Beispiele zeigen

### nachlesen:

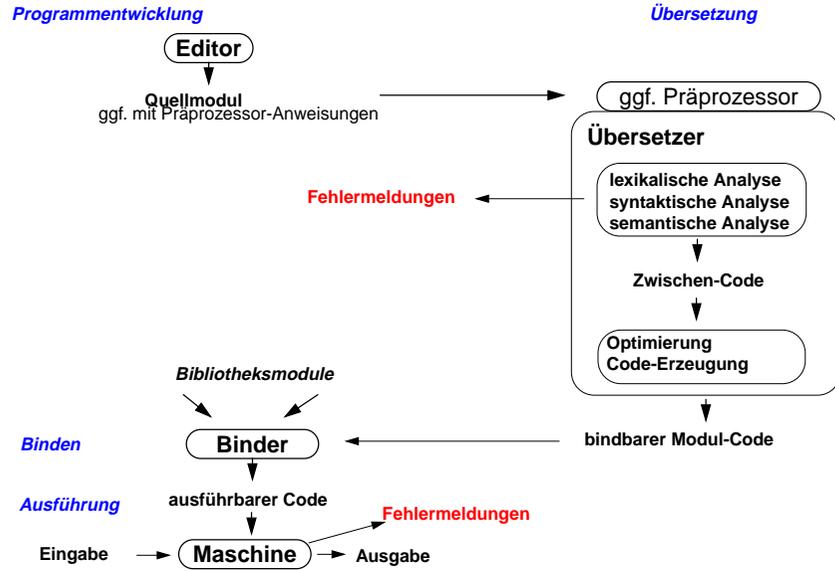
Interessante Web Site: [The Language List](#)

### Verständnisfragen:

Geben Sie mindestens 3 weitere Spezialsprachen an.

## 1.2 Implementierung von Programmiersprachen Übersetzung

GPS-1-8



© 2009 bei Prof. Dr. Uwe Kastens

**Ziele:**

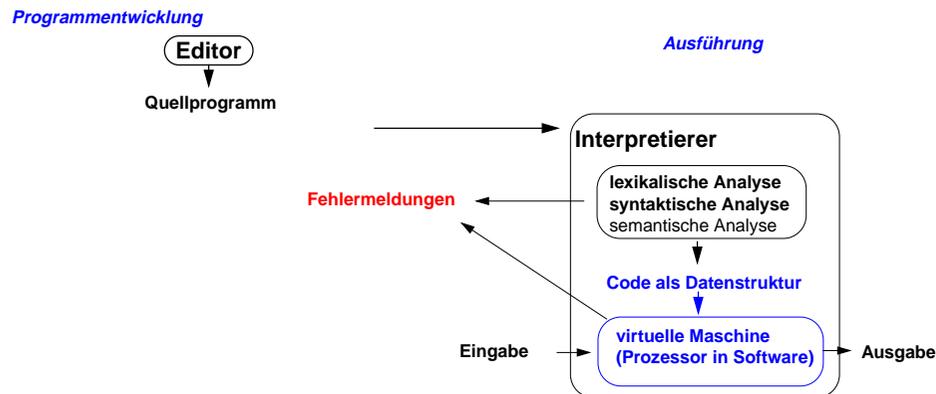
Übersetzer und seine Umgebung kennenlernen

**in der Vorlesung:**

- Erläuterung der Komponenten;
- getrennte Übersetzung von Moduln, Klassen, Modul-Code binden;
- integrierte Entwicklungsumgebungen (IDE);
- Präprozessoren auf Folie GPS 1.10

## Interpretation

GPS-1-8a



© 2009 bei Prof. Dr. Uwe Kastens

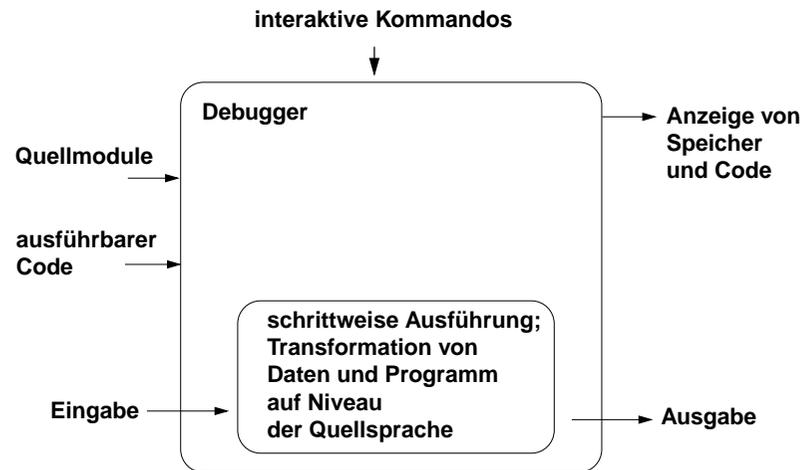
**Ziele:**

Interpretation statt Übersetzung

**in der Vorlesung:**

- Erläuterung des Interpretierers
- Konsequenzen für Sprachen und Benutzung

## Testhilfe: Debugger



## Vorlesung Grundlagen der Programmiersprachen SS 2015 / Folie 108b

### Ziele:

Prinzip von Debuggern kennenlernen

### in der Vorlesung:

- Erläuterung der Funktionsweise

## Präprozessor CPP

### Präprozessor:

- bearbeitet Programmtexte, bevor sie vom Übersetzer verarbeitet werden
- Kommandos zur Text-Substitution - ohne Rücksicht auf Programmstrukturen
- sprachunabhängig
- cpp gehört zu Implementierungen von C und C++, kann auch unabhängig benutzt werden

```

#include <stdio.h>           Datei an dieser Stelle einfügen
#include "induce.h"

#define MAXATTRS 256        benannte Konstante

#define ODD(x) ((x)%2 == 1)  parametrisiertes Text-Makro
#define EVEN(x) ((x)%2 == 0)

static void early (int sid)  Konstante wird substituiert
{ int attrs[MAXATTRS];
  ...
  if (ODD (currpartno)) currpartno--;  Makro wird substituiert
}                                       bedingter Textblock
#ifdef GORTO
  printf ("early for %d currpartno: %d\n",
         sid, currpartno);
#endif
  
```

## Vorlesung Grundlagen der Programmiersprachen SS 2015 / Folie 109

### Ziele:

Präprozessoren kennenlernen

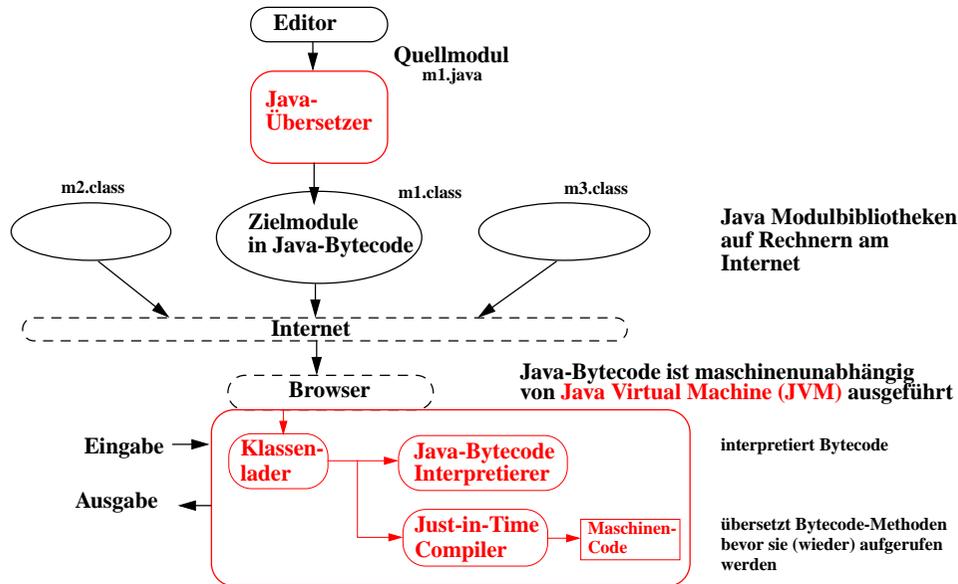
### in der Vorlesung:

Erläuterungen

- der Eigenschaften,
- der Kommandos,
- von Einsatzmöglichkeiten.

## Ausführung von Java-Programmen

GPS-1-10



© 2007 bei Prof. Dr. Uwe Kastens

### Ziele:

Maschinenunabhängigkeit durch Interpretation

### in der Vorlesung:

Java Bytecode, abstrakte Maschine, Rolle des Internet erläutern

### Verständnisfragen:

Siehe [SWE-16](#)

## 1.3 Dokumente zu Programmiersprachen

GPS1-11

### Reference Manual:

verbindliche Sprachdefinition, beschreibt alle Konstrukte und Eigenschaften vollständig und präzise

### Standard Dokument:

Reference Manual, erstellt von einer anerkannten Institution, z.B. ANSI, ISO, DIN, BSI

### formale Definition:

für Implementierer und Sprachforscher, verwendet formale Kalküle, z.B. KFG, AG, vWG, VDL, denotationale Semantik

### Benutzerhandbuch (Rationale):

Erläuterung typischer Anwendungen der Sprachkonstrukte

### Lehrbuch:

didaktische Einführung in den Gebrauch der Sprache

### Implementierungsbeschreibung:

Besonderheiten der Implementierung, Abweichungen vom Standard, Grenzen, Sprachwerkzeuge

© 2009 bei Prof. Dr. Uwe Kastens

### Ziele:

Unterschiedliche Zwecke von Sprachdokumenten erkennen

### in der Vorlesung:

- Auszüge aus Dokumenten
- typische Formulierungen in Reference Manuals

### nachlesen:

..., Abschnitt

### nachlesen:

nur ansehen:

- Java Reference Manual,
- Java Benutzerhandbuch,
- ein Java Lehrbuch

### Übungsaufgaben:

- Aussagen zu einer Spracheigenschaft in o.g. Dokumenten vergleichen

### Verständnisfragen:

**Ziele:**

Eindruck eines Standard-Dokumentes vermitteln.

**in der Vorlesung:**

Erläuterungen dazu

**Verständnisfragen:**

Wie würden Sie diesen Absatz ins Deutsche übersetzen?

**Beispiel für ein Standard-Dokument**

GPS-1-12

**6.1 Labeled statement****[stmt.label]**

A statement can be labeled.

```

labeled-statement :
    identifier : statement
  case constant-expression : statement
  default : statement

```

An identifier label **declares the identifier**. The only use of an identifier label is as the target of a goto. The **scope of a label** is the function in which it appears. Labels **shall not be redeclared within a function**. A label can be used in a goto statement before its definition. Labels have their **own name space** and do not interfere with other identifiers.

**[Aus einem C++-Normentwurf, 1996]**

Begriffe zu Gültigkeitsregeln, statische Semantik (siehe Kapitel 3).

**Ziele:**

Eindruck einer formalen Sprachdefinition vermitteln.

**in der Vorlesung:**

Erläuterungen dazu

**Verständnisfragen:**

Für welche der Symbole in der Prolog-Grammatik fehlen die Regeln?

**Beispiel für eine formale Sprachdefinition**

GPS-1-13

```

Prologprogramm ::= ( Klausel | Direktive )+ .
Klausel       ::= Fakt | Regel .
Fakt          ::= Atom | Struktur .
Regel         ::= Kopf ":" Rumpf "." .
Direktive     ::= ":" Rumpf
                | "?" Rumpf
                | "-" CompilerAnweisung
                | "?" CompilerAnweisung .

```

**[Spezifikation einer Syntax für Prolog]**

## Beispiel für ein Benutzerhandbuch

GPS-1-14

### R.5. Ausdrücke

Die [Auswertungsreihenfolge](#) von Unterausdrücken wird von den Präzedenz-Regeln und der Gruppierung bestimmt. Die üblichen mathematischen Regeln bezüglich der Assoziativität und Kommutativität können nur vorausgesetzt werden, wenn die Operatoren tatsächlich assoziativ und kommutativ sind. Wenn nicht anders angegeben, ist die [Reihenfolge der Auswertung der Operanden undefiniert](#). Insbesondere ist das [Ergebnis eines Ausdruckes undefiniert](#), wenn eine Variable in einem Ausdruck mehrfach verändert wird und für die beteiligten Operatoren keine Auswertungsreihenfolge garantiert wird.

#### Beispiel:

```
i = v[i++];          // der Wert von i ist undefiniert
i = 7, i++, i++;    // i hat nach der Anweisung den Wert 9
```

[Aus dem C++-Referenz-Handbuch, Stroustrup, 1992]

[Eigenschaften der dynamischen Semantik](#)

© 2005 bei Prof. Dr. Uwe Kastens

## Vorlesung Grundlagen der Programmiersprachen SS 2015 / Folie 114

### Ziele:

Eindruck eines Benutzerhandbuches

### in der Vorlesung:

Kurze Erläuterung des Reihenfolgeproblems der Ausdrucksauswertung

### Verständnisfragen:

Was bedeuten die Operatoren "++" und ", "?

## Beispiel für ein Lehrbuch

GPS-1-15

### Chapter 1, The Message Box

This is a very simple script. It opens up an alert message box which displays whatever is typed in the form box above. Type something in the box. Then click „Show Me“

#### HOW IT'S DONE

Here's the entire page, minus my comments. Take a few minutes to learn as much as you can from this, then I'll break it down into smaller pieces.

```
<HTML> <HEAD>
<SCRIPT LANGUAGE="JavaScript">
  function MsgBox (textstring) {alert (textstring)}
</SCRIPT>
</HEAD> <BODY>
<FORM> <INPUT NAME="text1" TYPE=Text>
  <INPUT NAME="submit" TYPE=Button VALUE="Show Me"
    onClick="MsgBox(form.text1.value)">
</FORM>
</BODY> </HTML>
```

[Aus einem JavaScript-Tutorial]

© 2005 bei Prof. Dr. Uwe Kastens

## Vorlesung Grundlagen der Programmiersprachen SS 2015 / Folie 115

### Ziele:

Eindruck einer Programmiersprach-Lehrbuches oder -Kurses vermitteln.

### in der Vorlesung:

Erläuterungen dazu

### Übungsaufgaben:

### Verständnisfragen:

Funktioniert der abgebildete HTML/JavaScript-Code in Ihrem Lieblingsbrowser?

## 1.4 Vier Ebenen der Spracheigenschaften

Die Eigenschaften von Programmiersprachen werden in 4 Ebenen eingeteilt:

Von a über b nach c werden immer größere Zusammenhänge im Programm betrachtet. In d kommt die Ausführung des Programmes hinzu.

Ebene	definierte Eigenschaften
a. Grundsymbole	Notation
b. Syntax (konkret und abstrakt)	Struktur
c. Statische Semantik	statische Zusammenhänge
d. Dynamische Semantik	Wirkung, Bedeutung

## Vorlesung Grundlagen der Programmiersprachen SS 2015 / Folie 116

### Ziele:

Einordnung von Spracheigenschaften

### in der Vorlesung:

- Ebenen gegeneinander abgrenzen;
- statische und dynamische Semantik definieren die Bedeutung von Konstrukten - nicht nur ihre Korrektheit;
- Beispiele anhand der Folien GPS-1-17 bis GPS-1-19 und aus Reference Manuals

### Übungsaufgaben:

- Geben Sie je 2 Verletzungen von Regeln zu a bis d in Java an.
- Schreiben Sie ein kurzes, fehlerhaftes Java-Programm, das zu a bis d je mindestens eine Fehlermeldung provoziert.

### Verständnisfragen:

Können Sie sich Sprachen vorstellen, die keine statische Semantik haben? Welche Aufgaben würde ein Übersetzer für solche Sprachen erledigen?

## Beispiel für die Ebene der Grundsymbole

Ebene	definierte Eigenschaften
a. Grundsymbole	Notation

typische **Klassen von Grundsymbolen**:

Bezeichner,  
 Literale (Zahlen, Zeichenreihen),  
 Wortsymbole,  
 Spezialsymbole

formal definiert z. B. durch **reguläre Ausdrücke**

Folge von Grundsymbolen:

```
int dupl ( int a ) { return 2 * a ; }
```

## Vorlesung Grundlagen der Programmiersprachen SS 2015 / Folie 117a

### Ziele:

Ebene der Grundsymbole verstehen

### in der Vorlesung:

Erläuterung des Beispiels

### Verständnisfragen:

Warum sollte man beim Programmieren alle Wortsymbole der Sprache kennen?

## Beispiel für die Ebene der Syntax

Ebene

definierte Eigenschaften

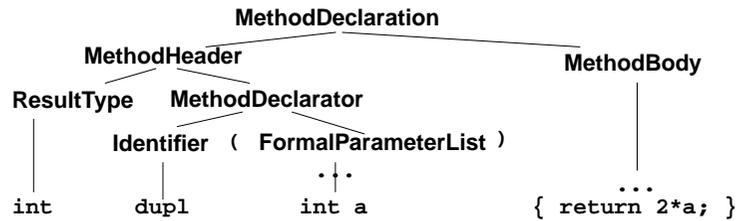
### b. Syntax (konkrete und abstrakte Syntax)

Struktur von Sprachkonstrukten

syntaktische Struktur

formal definiert durch **kontext-freie Grammatiken**

Ausschnitt aus einem Ableitungs- bzw. Strukturbaum:



## Vorlesung Grundlagen der Programmiersprachen SS 2015 / Folie 117b

**Ziele:**

Syntaktische Ebene verstehen

**in der Vorlesung:**

Erläuterung des Beispiels

## Beispiel für die Ebene der statischen Semantik

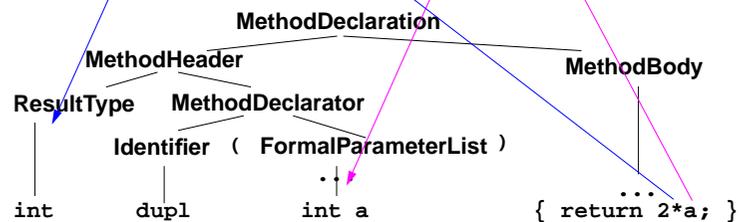
Ebene

definierte Eigenschaften

### c. statische Semantik

statische Zusammenhänge, z. B.

meist verbal definiert;

formal definiert z. B. durch **attributierte Grammatiken****a ist an die Definition des formalen Parameters gebunden. Bindung von Namen****Der return-Ausdruck hat den gleichen Typ wie der ResultType. Typregeln**

## Vorlesung Grundlagen der Programmiersprachen SS 2015 / Folie 117c

**Ziele:**

Ebene der statischen Semantik verstehen

**in der Vorlesung:**

Erläuterung des Beispiels

**Verständnisfragen:**

Kann es sein, dass das Einfügen der Beispielzeile

`int dupl ( int a ) { return 2 * a; }`

in ein korrektes Java-Programm zu einem nicht mehr übersetzbaren Java-Programm führt?.

## Beispiel für die Ebene der dynamischen Semantik

Ebene

definierte Eigenschaften

### d. dynamische Semantik

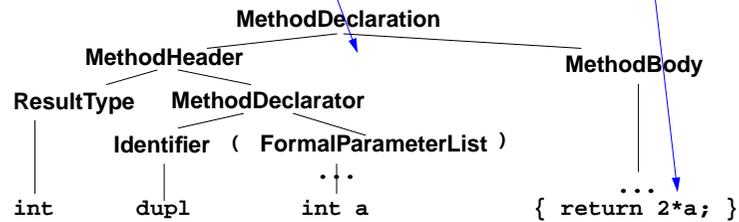
Bedeutung, Wirkung der Ausführung

von Sprachkonstrukten, Ausführungsbedingungen

meist verbal definiert;

formal definiert z. B. durch **denotationale Semantik**

Ein Aufruf der Methode `dup1` liefert das Ergebnis  
der Auswertung des `return`-Ausdruckes



## Vorlesung Grundlagen der Programmiersprachen SS 2015 / Folie 117d

**Ziele:**

Ebene der dynamischen Semantik verstehen

**in der Vorlesung:**

Erläuterung des Beispiels.

**Verständnisfragen:**

Beschreiben Sie einen Fehler, der erst bei der Ausführung eines Java-Programms erkannt werden kann.

## Statische und dynamische Eigenschaften

**Statische** Eigenschaften: aus dem Programm bestimmbar, ohne es auszuführen

**statische** Spracheigenschaften:

Ebenen a, b, c: Notation, Syntax, statische Semantik

**statische** Eigenschaften eines Programmes:

Anwendung der Definitionen zu a, b, c auf das Programm

Ein Programm ist **übersetzbar**, falls es die Regeln zu (a, b, c) erfüllt.

**Dynamische** Eigenschaften: beziehen sich auf die Ausführung eines Programms

**dynamische** Spracheigenschaften:

Ebene d: dynamische Semantik

**dynamische** Eigenschaften eines Programmes:

Wirkung der Ausführung des Programmes mit bestimmter Eingabe

Ein Programm ist **ausführbar**, falls es die Regeln zu (a, b, c) und **(d)** erfüllt.

## Vorlesung Grundlagen der Programmiersprachen SS 2015 / Folie 118

**Ziele:**

Begriffe statisch und dynamisch verstehen

**in der Vorlesung:**

Begriffe in Bezug zu den Ebenen erläutern

## Beispiel: Dynamische Methodenbindung in Java

Für den Aufruf einer Methode kann im Allgemeinen erst **beim Ausführen** des Programms bestimmt werden, **welche Methode** aufgerufen wird.

```
class A {
    void draw (int i){...};
    void draw () {...}
}

class B extends A {
    void draw () {...}
}

class X {
    void m () {
        A a;
        if (...)
            a = new A ();
        else a = new B ();

        a.draw ();
    }
}
```

**statisch** wird am Programmtext bestimmt:

- der Methodenname: **draw**
- die Typen der aktuellen Parameter: keine
- der statische Typ von **a**: **A**
- ist eine Methode **draw** ohne Parameter in A oder einer Oberklasse definiert? ja
- **draw()** in **B** überschreibt **draw()** in **A**

**dynamisch** wird bei der Ausführung bestimmt:

- der Wert von **a**:  
z. B. Referenz auf ein **B**-Objekt
- der Typ des Wertes von **a**: **B**
- die aufzurufende Methode: **draw** aus **B**

## Vorlesung Grundlagen der Programmiersprachen SS 2015 / Folie 118a

### Ziele:

Begriffe statisch und dynamisch verstehen

### in der Vorlesung:

- Es wird erläutert, welche Aspekte der Methodenbindung in Java zur statischen und welche zur dynamischen Semantik gehören.
- Die Entscheidung zwischen überladenen Methoden wird vollständig statisch getroffen.

## Fehler im Java-Programm

Fehler klassifizieren: lexikalisch, syntaktisch, statisch oder dynamisch semantisch:

```
1 class Error
2 { private static final int x = 1..;
3   public static void main (String [] arg)
4   { int[] a = new int[10];
5     int i
6     boolean b;
7     x = 1; y = 0; i = 10;
8     a[10] = 1;
9     b = false;
10    if (b) a[i] = 5;
11  }
12 }
```

## Vorlesung Grundlagen der Programmiersprachen SS 2015 / Folie 119

### Ziele:

Ebenen **a** bis **d** anhand von Fehlern erkennen

### in der Vorlesung:

- Auflösung und Diskussion von nicht eindeutigen Klassifizierungen

### Übungsaufgaben:

siehe [Folie 116](#)

### Verständnisfragen:

- Warum kann man lexikalische und syntaktische Fehler nicht sicher unterscheiden?
- Regelt die Sprachdefinition immer eindeutig, in welchen Fällen ein Übersetzer einen Fehler melden muss?

## Fehlermeldungen eines Java-Übersetzers

```

Error.java:2: <identifier> expected
  { private static final int x = 1..;
                                ^

Error.java:5: ';' expected
  int i
    ^

Error.java:2: double cannot be dereferenced
  { private static final int x = 1..;
                                ^

Error.java:7: cannot assign a value to final variable x
  x = 1; y = 0; i = 10;
    ^

Error.java:7: cannot resolve symbol
symbol  : variable y
location: class Error
  x = 1; y = 0; i = 10;
            ^

Error.java:9: cannot resolve symbol
symbol  : variable b
location: class Error
  b = false;
    ^

Error.java:10: cannot resolve symbol
symbol  : variable b
location: class Error
  if (b) a[i] = 5;
      ^

7 errors

```

## Vorlesung Grundlagen der Programmiersprachen SS 2015 / Folie 120

### Ziele:

Fehlermeldung klassifizieren

### in der Vorlesung:

Erläuterungen dazu

### Verständnisfragen:

- Warum gibt es zu Zeile 2 zwei Meldungen?
- Was bedeuten die Meldungen zu den Zeilen 9 und 10?

## Zusammenfassung zu Kapitel 1

Mit den Vorlesungen und Übungen zu Kapitel 1 sollen Sie nun Folgendes können:

- Wichtige Programmiersprachen zeitlich einordnen
- Programmiersprachen klassifizieren
- Sprachdokumente zweckentsprechend anwenden
- Sprachbezogene Werkzeuge kennen
- Spracheigenschaften und Programmeigenschaften in die 4 Ebenen einordnen

## Vorlesung Grundlagen der Programmiersprachen SS 2015 / Folie 121

### Ziele:

Ziele des Kapitels erkennen

### in der Vorlesung:

Erläuterungen dazu