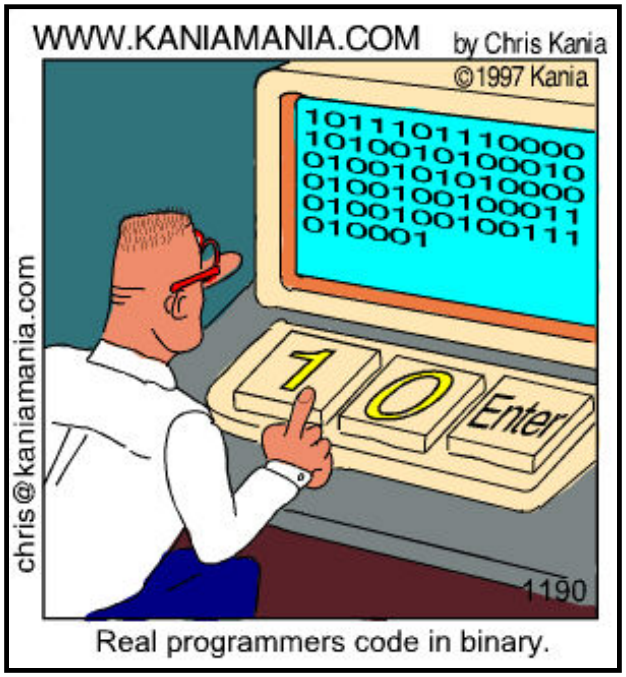
Grundlagen der Programmierung

VL 02: Erste Schritte

Prof. Dr. Samuel Kounev,
M.Sc. Norbert Schmitt



Real programmers code in binary.

Danksagung

- Vorlesungsmaterialien von Prof. Dr. Detlef Seese wurden als Basis verwendet
- Unterstützung bei der technischen und inhaltlichen Gestaltung des Vorlesungsmaterials leisteten:

  Jóakim v. Kistowski

  Dietmar Ratz, Joachim Melcher, Roland Küstermann, Jana Weiner, Hagen Buchwald, Matthes Elstermann, Oliver Schöll, Niklas Kühl, Tobias Diederich

Inhalt und Ziele

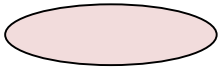
- Wie erstellt man einfache Java/C Programme?
- Was sind Grundbestandteile der Sprache?
  - Compiler, Grammatik, Grundelemente der Sprache

Erste Java-Programme

Java

```
public class Kehrwert {
  public static void main (String[] args) {
    // Meine Kehrwertberechnung
    double x, y;
    x = 5;
    y = 1 / x;
    System.out.print("Der Kehrwert von 5 ist ");
    System.out.println(y);
    System.out.println("Das war's!");
  }
}
```

Wortsymbole, **Schlüsselwörter**, reservierte Wörter



frei wählbare **Bezeichner** für Namen von Variablen, Methoden, Klassen, …
(teilweise in der Java-Klassen-Bibliothek vordefinierte Bedeutung)

Wichtig: Jede(s) ausführbare Klasse (Programm) muss eine Methode **`main`** haben

Erste Java-Programme

Java

```
public class Kehrwert {
  public static void main (String[] args) {
    // Kehrwertberechnung
    double x, y;
    x = 5;
    y = 1 / x;
    System.out.print("Der Kehrwert von 5 ist ");
    System.out.println(y);
    System.out.println("Das war's!");
  }
}
```

Klassen- bzw. Programm-Name

Kommentar
(vom Compiler ignoriert)

Vereinbarungen, Deklarationen

Wertzuweisung mit Konstante

Erste Java-Programme

Java

```
public class Kehrwert {
  public static void main (String[] args) {
    // Kehrwertberechnung
    double x, y;
    x = 5;
    y = 1 / x;
    System.out.print("Der Kehrwert von 5 ist ");
    System.out.println(y);
    System.out.println("Das war's!");
  }
}
```

Wertzuweisung mit Ausdruck

Jede Anweisung wird durch **;** abgeschlossen

Ausgabeanweisungen Methodenaufrufe

auszugebender Wert (**double**)

Argumentklammer

auszugebender Wert (Text)

Vorgehensweise am Rechner

Java

1. Editieren

Mit einem Texteditor den Programmtext erstellen und in einer Datei speichern: z.B.

```
notepad Kehrwert.java
```

2. Übersetzen

Programmtext mit dem Compiler in Java-Bytecode übersetzen:

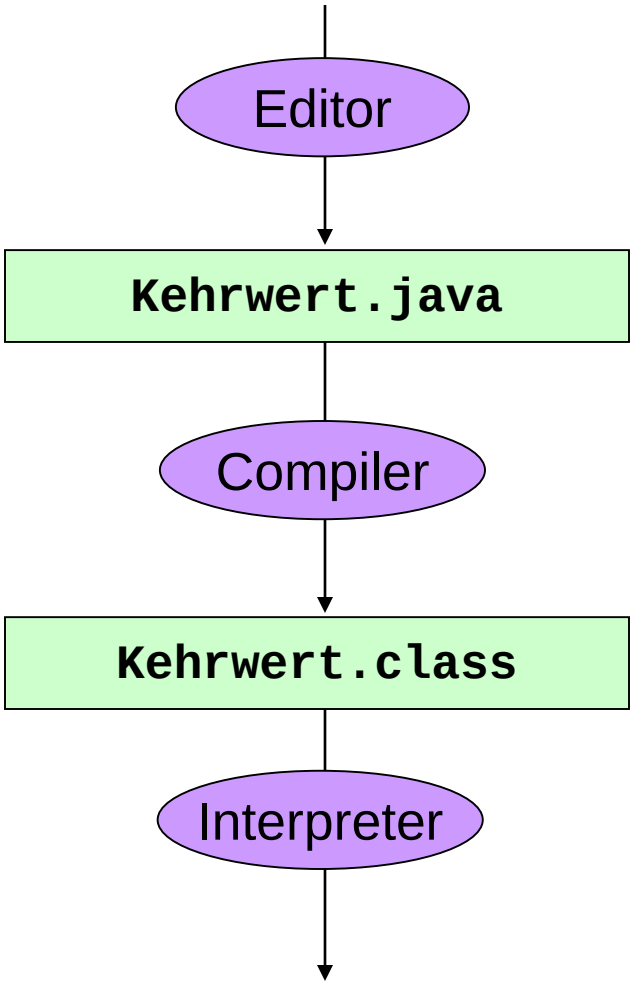
```
javac Kehrwert.java
```

Dabei entsteht die gleichnamige Datei mit Endung "`.class`"

3. Ausführen

Bytecode mit dem Interpreter ausführen:

```
java Kehrwert
```



Erste C-Programme

C/C++

```
#include <stdio.h>
int main()
{
        double x, y;
        x = 5;
        y = 1/x;
        printf("Der Kehrwert von 5 ist %f\n", y);
        printf("Das war's!\n");
        return 0;
}
```

Wortsymbole, **Schlüsselwörter**, reservierte Wörter

frei wählbare **Bezeichner** für Namen von Variablen, Methoden, Klassen, …
(teilweise in der C-Standardbibliothek vordefinierte Bedeutung)

Vorgehensweise am Rechner

C/C++

1. Editieren

Mit einem Texteditor den Programmtext erstellen und in einer Datei speichern: z.B.

```
nano kehrwert.c
```

2. Übersetzen

Programmtext mit dem Compiler in Maschinencode übersetzen:

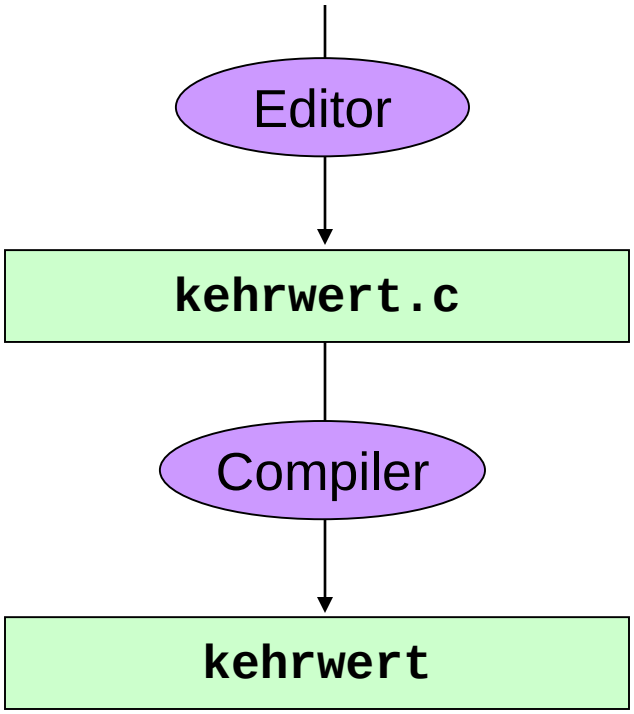
```
gcc –o kehrwert kehrwert.c
```

Dabei entsteht die ausführbaren Datei mit der Name "**kehrwert**"

3. Ausführen

Auf Konsole ausführen:

```
./kehrwert
```



Compiler, Grammatik, Grundelemente

- Der Compiler übersetzt nur "korrekte" Java-Programme
- Die Korrektheit wird anhand der Grammatik überprüft
- **Grammatik**
  - Regeln für die Bildung von "Sätzen" der Sprache, also Programmen
  - festgelegt durch das Alphabet, die Syntax und die Semantik der Sprache
    - **Alphabet:** definiert den Zeichenvorrat der zur Darstellung von Programmen verwendet werden darf
    - **Syntax:** regelt, welche Zeichenfolgen des Alphabets zulässige Sätze (Programme) der Sprache bilden.
    - **Semantik:** beschreibt die Bedeutung der einzelnen Sprachelemente und die Beziehungen zwischen ihnen, wodurch die Bedeutung des Programms festgelegt wird.

Compiler, Grammatik, Grundelemente (2)

Satz	lexikalisch korrekt?	syntaktisch korrekt?	semantisch korrekt?
"hd$:r%."			
"Mein ist Ring das."			
"Rote Angst reiten gezaubert."			
"Das Pferd reitet auf dem Hobbit."			
"Das ist mein Ring."			

Compiler, Grammatik, Grundelemente (3)

Java

C/C++

- **Java/C-Programm:** prinzipiell lediglich eine Folge von Zeichen aus dem Zeichenvorrat (sogenannte Unicode-Zeichen) von Java/C.

- **Zeichenvorrat** (character set)
  - **Buchstaben** (letters)
    - Lateinische Groß- und Klein-Buchstaben (wie im ASCII- und ISO-Latin-1-Code)
    - der Unterstrich (underscore) : _
    - das Dollarzeichen: **$**
    - weitere Buchstaben aus dem Unicode (z.B. griechische Buchstaben)
  - **Ziffern** (digits)
    - Lateinische Ziffern 0 bis 9 (wie im ASCII- und ISO-Latin-1-Code)
    - weitere Ziffern aus dem Unicode (z. B. thailändische Ziffern)

Compiler, Grammatik, Grundelemente (4)

Java

C/C++

- **Whitespace-Zeichen** (unsichtbare Zeichen)
  - das Leerzeichen (Blank) (space character)
  - das Zeilenendezeichen (end-of-line character)
  - das Tabulatorzeichen (tab character)
  - das Seitenvorschubzeichen (form feed / page break / new page character)
- **Sonderzeichen für Satzzeichen** (Interpunktionszeichen) (punctuation marks)

  ( ) { } [ ] ; , .
- **Sonderzeichen für Operatoren**

  = > < ! ~ ? : & | + - * / ^ %

- **Sonderzeichen für Ersatzdarstellungen**

  **** (backslash)

- **Sonderzeichen für Anführungszeichen**

  ‚ ‘ „ “ (single resp. double quotes / quotation marks)

Compiler, Grammatik, Grundelemente (5)

- ISO-Latin-1-Code
  (Tabelle nur als Beispiel,
  ohne Einzelheiten)

ISO/IEC 8859-1 [Bearbeiten]

Code	…0	…1	…2	…3	…4	…5	…6	…7	…8	…9	…A	…B	…C	…D	…E	…F
0…	*nicht belegt*															
1…																
2…	*SP*	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3…	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4…	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5…	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6…	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7…	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
8…	*nicht belegt*															
9…																
A…	*NBSP*	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	*SHY*	®	¯
B…	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C…	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D…	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E…	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F…	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Quelle: http://de.wikipedia.org/wiki/ISO_8859-1

Compiler, Grammatik, Grundelemente (6)

Java

C/C++

- **Der Compiler**
  - muss zunächst die Folge von Zeichen "scannen"
  - sucht darin Gruppen von Zeichen
  - indem nach Trennern gesucht wird
  - weitere Zeichen zwischen zwei Trennern bilden ein sogenanntes lexikalisches Element (Token), ein Wort der Sprache.
- **Mögliche Trenner** (deliminators) **in Java:**
  - Whitespace-Zeichen (siehe oben)
  - Kommentare
  - Satzzeichen
  - Operatoren
- **Kommentare:** jeweils alle Zeichen zwischen und einschließlich der Zeichen

Beginn des Kommentars	Ende des Kommentars
//	Zeilenende
/*	*/
/**	*/

Kommentare

Java

C/C++

- Traditionelle Kommentare
  - beginnen mit /* und enden mit */ können sich über mehrere Zeilen erstrecken (siehe Beispiel)
- einzeilige Kommentare (Zeilenkommentare)
  - beginnen mit // und enden am Ende der aktuellen Zeile
  - Bei C erst ab C99 unterstützt

Kommentare (2)

Java

- Dokumentationskommentare
  - werden von /** und */ eingeschlossen
  - können sich über mehrere Zeilen erstrecken
  - dienen zur Dokumentation von Programmen im Quelltext.
  - werden durch das im JDK enthaltene Programm javadoc aus dem Quelltext extrahiert und in ein HTML-Dokument verwandelt.
- Hinweis: Beginnen Sie jede Zeile des Textes dieser Kommentare mit einem zusätzlichen *.
  - ➢ Hierdurch wird der Kommentar optisch vom Quellcode abgesetzt

Kommentare (3)

Java

Beispiel JavaDoc Kommentar:

```
/**
* Dieses Programm berechnet den Wert von 7+11
*/
public class Kommentare {
   /**hier kommt unsere main-Methode*/
   public static void main ( String[] args ){
   int summe; //Deklaration
   summe = 7 + 11; //Addition
   System.out.println(summe);
   /* an dieser Stelle endet der Inhalt
   der Methode main */
   }
}
```

Kommentare (4)

Java

```
/** Dokumentationskommentare */
```

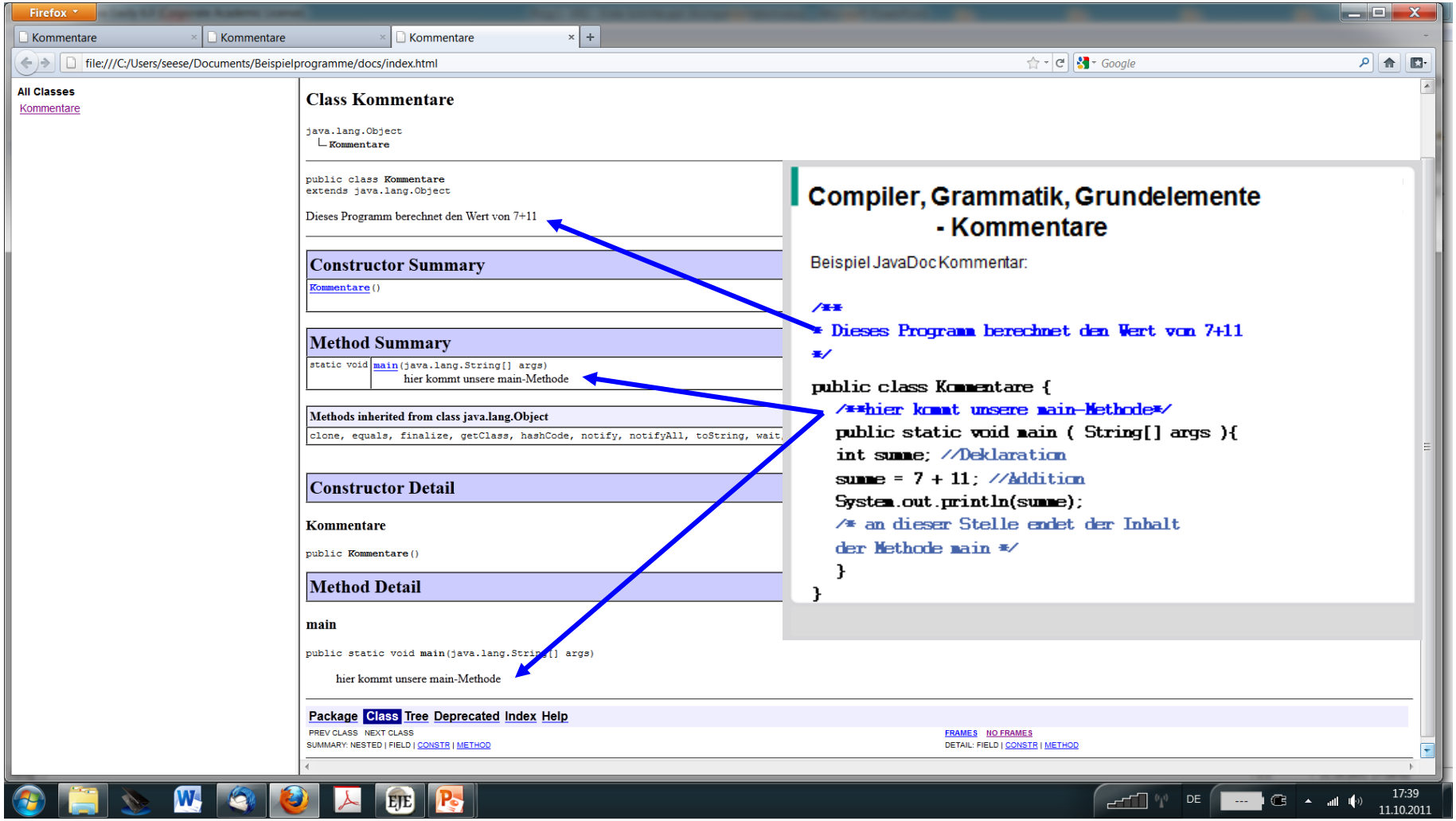
Beispiel Klasse `classname`:

unter `classname.java` speichern

`javac classname.java` erzeugt Bytecode `classname.class`

`javadoc classname.java` erzeugt `classname.html`

Kommentare (5)



Kommentare (6)

Java

C/C++

- Kommentare können nicht geschachtelt werden
- /* und */ haben keine besondere Bedeutung in Zeilenkommentaren
- // hat keine besondere Bedeutung in Kommentaren, die mit /** oder /* beginnen
- Die Bedeutung eines Kommentarzeichens ist abhängig vom Kontext

Übungen

- Warum sind ordentliche Kommentare wichtig?
- Kann innerhalb eines mehrzeiligen mit /* beginnenden Kommentars */ vorkommen?
- Wo endet der folgende Kommentar?

```
/* Dieser Kommentar /* // /** endet hier */
```

Lexikalische Elemente (Tokens)

Java

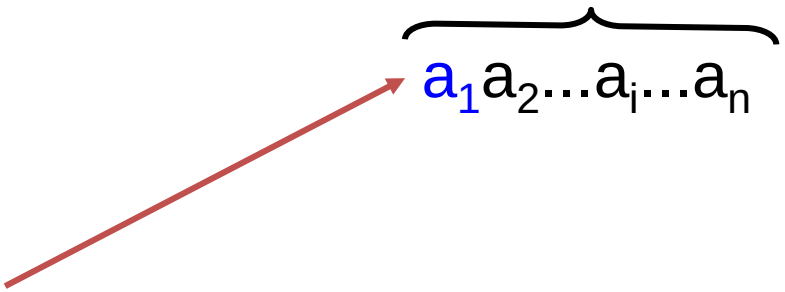
C/C++

- **Bezeichner (identifiers):** beliebig lange Zeichenfolge aus Buchstaben und Ziffern, beginnend mit einem Buchstaben (Unterscheidung Groß- und Kleinschreibung)
- **Schlüsselwörter / Wortsymbole (keywords):** reservierte Wörter mit vordefinierter Bedeutung. Können **nicht** als Bezeichner verwendet werden.
- **Literale (literals):** repräsentieren konstante Werte von Datentypen
  - Ganzzahl- oder Gleitkomma-Werte **(integer, floating point number)**
  - logische Werte (**`true`** und **`false`**)
  - Zeichen-Werte **(char)** (in zwei `'`-Zeichen eingeschlossen)
  - Zeichenkettenwerte **(string)** (in zwei `"`-Zeichen eingeschlossen)
  - die Nullreferenz (**`null`**, später in der Vorlesung)
- **Satzzeichen / Interpunktionszeichen** `. , ; ) ( } { ] [`

  (punctuation marks)
- **Operatoren**

Bezeichner (identifier)

Beliebige Folge von Buchstaben (z.B. A,...,Z,a,...,z,_,$) und Ziffern (0,...,9)



muss mit einem Buchstaben beginnen (z.B., A, ... , Z, a, ... , z, _ , $)

dürfen weder Schlüsselwörter, noch Literale (`true, false, …`) sein

Methoden in `java.lang.Character`:

Java

- `Character.isJavaIdentifierStart(char)`
- `Character.isJavaIdentifierPart(char)`
- `Character.isWhitespace(char)`

Compiler, Grammatik, Grundelemente (7)

Java

C/C++

- **Operatoren** (operators)
  - werden durch Operatorzeichen dargestellt
  - können auch durch zwei, drei oder vier Zeichen angegeben sein.
  - z. B.: + && >>> >>>=
- Operatoren sind sowohl Trenner als auch Tokens *(Token = lexikalisches Element / Bitfolge definierter Struktur)*
- Die Zeichensequenz

  `a+b,c-d`

  besteht für den Compiler aus den lexikalischen Elementen **a** (ein Bezeichner), **+** (ein Operator), **b** (ein Bezeichner), **,** (ein Satzzeichen), **c** (ein Bezeichner), **-** (ein Operator) und **d** (ein Bezeichner).
- Sinnvoll für Lesbarkeit von Programmen
  - zusätzliche Trennzeichen
  - im Beispiel **`a + b, c - d`**

Compiler, Grammatik, Grundelemente (8)

Java

C/C++

- **Regeln:**
  - Innerhalb von Bezeichnern, Literalen oder Schlüsselwörtern sowie zusammengesetzten Operator-Symbolen **darf kein** Trennzeichen stehen.
  - Unmittelbar aufeinanderfolgende Bezeichner, Zahlen oder Schlüsselwörter **müssen** durch **mindestens ein** Trennzeichen getrennt werden.

- Beispiele für zulässige Bezeichner:

```
Zahl_Maximum                IstBuchstabeOderZahl
AlterInJahren               GehaltInDM
WahlergebnisInProzent       _j23
$85                         αεκ
```

- Beispiele für unzulässige Bezeichner:

```
14i   class     int     gamma{14}   true
Character.isJavaLetter()
Character.isJavaLetterOrDigit()
```

Compiler, Grammatik, Grundelemente (9)

Java

- Schlüsselwörter (Wortsymbole)

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

(Hinweis: **true**, **false** und **null** werden zwar vom Compiler ähnlich wie die Schlüsselwörter behandelt, sind aber technisch gesehen eigentlich Literale)

- Operatoren

=	**<**	**>**	**!**	**~**	**?:**		
==	**!=**	**<=**	**>=**	**&&**	**||**		
++	**--**	**+**	**-**	*****	**/**	**%**	
&	**|**	**^**					
<<	**>>**	**>>>**	**<<=**	**>>=**	**>>>=**		
+=	**-=**	***=**	**/=**	**&=**	**|=**	**^=**	**%=**

Programmierstil - Beispiel

Java

```
public class MitStil {
  public static void main (String[] args){
    int a, b;
    a = 2;
    b = 3;
    a = a * b;
    b = a % b;
    System.out.println(a);
    System.out.println(b);
  }
}
```

```
public class ohnestil {public static
void main (String[  ]args){int A,b;A
=2;b=3;A=A*b;b=A%b;System.out.println
(A);System.out.println(b);}}
```

Die Goldenen Regeln der Code-Formatierung

Regel 1: Pro Zeile eine Anweisung!

Regel 2: Rücke zusammenhängende Teile ein!

Regel 3: Die zusammengehörigen geschweiften Klammern eines Blocks stehen alleine in ihrer Zeile und in der gleichen Spalte! (programmierfreundliche Variante – im Buch steht platzsparende Variante)

Regel 4: Die Befehlsblöcke in switch-Blöcken werden gemäß obigen Regeln formatiert!

Regel 5: Klassennamen beginnen immer mit einem Großbuchstaben! Setzt sich ein Klassennamen aus mehr als einem Wort zusammen, beginnt jedes dieser Worte mit einem Großbuchstaben.

Regel 6: Variablen- und Methodennamen beginnen immer mit einem Kleinbuchstaben! Setzen sich Namen aus mehr als einem Wort zusammen, beginnt jedes Wort mit einem Großbuchstaben.

Regel 7: Konstanten werden ausschließlich in Großbuchstaben geschrieben. Setzt sich eine Konstante aus mehreren Worten zusammen, so werden diese durch Unterstriche getrennt.

Regel 8: Eine Methode mit einem Rückgabewert heißt Funktion. Eine Funktion besitzt stets nur eine einzige return-Anweisung. Diese steht in der letzten Zeile des Methodenblocks und lautet return result;

Fragen?


CHINA
TAIWAN
BURMA
LAOS
THAILAND
CAMBODIA
VIETNAM
PHILIPPINE SEA
PHILIPPINES
MANILA
SOUTH CHINA SEA
SULU SEA
PACIFIC OCEAN
MALAYSIA
BRUNEI
SABAH
MALAYA
KUALA LUMPUR
SINGAPORE
SARAWAK
SUMATRA
KALIMANTAN
SULAWESI
IRIAN JAYA
JAKARTA
JAVA
INDONESIA
INDIAN OCEAN
AUSTRALIA


Java