

## Wo sind wir?

- Java-Umgebung
- **Lexikale Konventionen**
- Datentypen
- Kontrollstrukturen
- Ausdrücke
- Klassen, Pakete, Schnittstellen
- JVM
- Exceptions
- Java Klassenbibliotheken
- Ein-/Ausgabe
- Collections
- Threads
- Applets, Sicherheit
- Grafik
- Beans
- Integrierte Entwicklungsumgebungen

## Übersicht lexikale Struktur von Java

- Zeichensatz
- Lexikale Analyse der Sprache
- Grundsymbole der Sprache
  - Bezeichner
  - Schlüsselworte
  - Literale
  - Separatoren
  - Operatoren

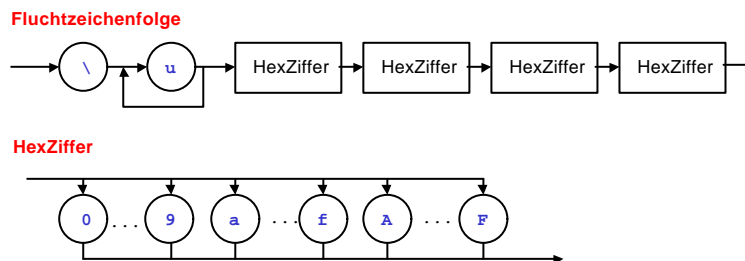
## Lexikale Struktur von Java-Programmen

- Alle Java-Programme werden im **Unicode-Zeichensatz** geschrieben.
- <http://www.unicode.org/>
- Mit wenigen Ausnahmen wird davon **nur der ASCII-Teil** verwendet, d.h. die ersten 128 Zeichen (d.h. keine deutschen Umlaute)
- **Ausnahmen (voller Unicode-Zeichensatz möglich):**
  - Kommentare
  - Bezeichner (z.B. von Variablen- und Methodennamen; gute Idee???)
  - Inhalt von Zeichenketten und Zeichenkettenliteralen
- **Beispiel:**

```
// Grüzi miteinand
"Fahre zur Hölle, erbärmlicher Schurke!"
```
- Über **Fluchtzeichenfolgen** können mit reinen ASCII-Zeichenfolgen beliebige Unicode-Zeichen angegeben werden

## Fluchtzeichenfolge

- Fluchtzeichenfolgen **bestehen nur aus ASCII-Zeichen**
- Mit Fluchtzeichenfolgen lässt sich **jedes Unicode-Zeichen** angeben
- Der Wert der hexadezimalen Zahl entspricht dem Code des Zeichens im Unicode-Zeichensatz
- Mehrere u hintereinander nur für bestimmte Werkzeuge interessant
- **Syntax:**



### Beispiele:

`\u005c` steht für Z  
`\uu005c` steht für Z

## Lexikale Übersetzung

- Zuerst wird (in 3 Teilschritten; s.u.) in jeder Übersetzung durch einen Java-Compiler die Folge von Unicode-Eingabezeichen **zerlegt in eine Folge von Eingabeelementen**
- **Eingabeelemente von Java sind:**
  - Wortzwischenräume
  - Kommentare
  - Token (dies sind die terminalen Symbole der Java-Grammatik):
    - Bezeichner
    - Schlüsselworte
    - Literale
    - Trennzeichen
    - Operatoren
- **Schritt 1:** Ersetzen von Fluchtzeichenfolgen in Unicode-Zeichen
- **Schritt 2:** Aufteilung des Unicode-Zeichenstroms aus Schritt 1 in Eingabezeichen und Zeilenbegrenzer
- **Schritt 3:** Aufteilung des Stroms aus Schritt 2 in eine Folge von Eingabeelementen

## Beispiel

```
// Dies ist ein Kommentar  
class Test {  
    int \u005c;
```



Fluchtzeichen  
nach Unicode

```
// Dies ist ein Kommentar  
class Test {  
    int Z;
```



Aufteilung in Zeichen  
und Zeilenbegrenzer

```
// Dies ist ein Kommentar ENDE  
class Test { ENDE  
    int Z; ENDE  
} ENDE
```

Aufteilung in  
Eingabelemente



```
// Dies ist ein Kommentar  
class Test {  
    int Z;  
}
```

## Trennung von Token

- Token werden durch Zwischenraumzeichen und Kommentare getrennt
- **Zwischenraumzeichen** sind:
  - Leerzeichen
  - Horizontaler Tabulatorzeichen
  - Seitenvorschub
  - Zeilenende (CR, LF oder CR+LF)
- Als Token wird die **längstmögliche Verlängerung** genommen
- **Beispiel:**
  - `+=` ist ein Token ("`+=`")
  - `+ =` sind zwei Token ("`+`" und "`=`")
  - `a -- b` sind drei Token ("`a`" und "`--`" und "`b`").  
(Längstmögliche Verlängerung ist grammatikalisch falsch, grammatikalisch wäre richtig: `a - -b`)

## Bezeichner

Ein Bezeichner ist eine **beliebig lange Folge von Buchstaben und/oder Ziffern**, wobei das **erste Zeichen ein Buchstabe** sein muss. Ausgenommen sind Schlüsselwörter und "quasi"-Schlüsselwörter. Es sind zwar prinzipiell beliebige Unicode-Zeichen erlaubt, aber ein Amerikaner kann mit dem Bezeichner *größterÜberläufer* evtl. wenig anfangen, evtl. noch nicht einmal in seinem Editor korrekt anzeigen lassen. Besser ist die Beschränkung auf den ASCII-Zeichensatz.

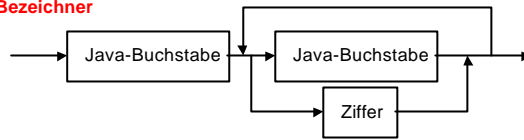
Bezeichner werden z.B. benötigt als Variablennamen, Methodennamen, Klassennamen usw.

**Beispiele:**

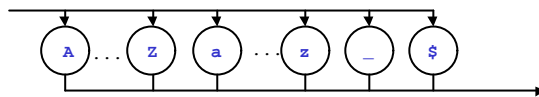
```
i           meineVariable   meineVariable2
```

## Syntaxdiagramm

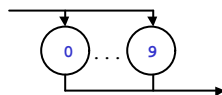
### Bezeichner



### Java-Buchstabe



### Ziffer



## Schlüsselwörter

abstract	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	
continue	goto	package	this	

### "quasi"-Schlüsselwörter:

    null      true      false

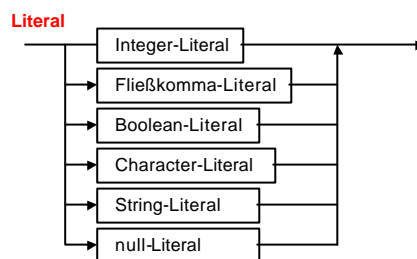
## Literale

Ein Literal ist die **Quelltext-Darstellung eines Wertes** eines einfachen Typs (int, float, char usw.) oder vom Typ String oder des null-Typs. Literale werden oft auch als **"Konstanten"** bezeichnet.

Beispiele:

5      3.14      3.14f      "Hallo"    true      null

Je nach Basistyp werden verschiedene Literale unterschieden:



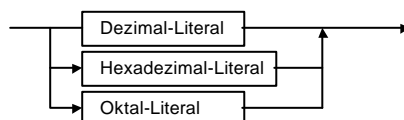
## Integer-Literal

Mit einem Integer-Literal werden **ganzzahlige Werte** dargestellt. Die Zahlkonstanten können zur:

- Basis 10 (dezimal)
- Basis 16 (hexadezimal)
- Basis 8 (oktal)

angegeben werden.

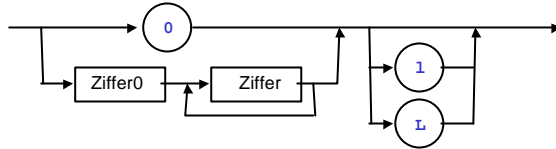
**Integer-Literal**



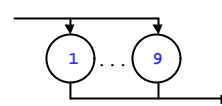
Durch Anfügen eines "l" oder "L" entsteht eine Konstante vom Typ **long**, sonst ist die Konstante vom Typ **int**.

## Syntaxdiagramme

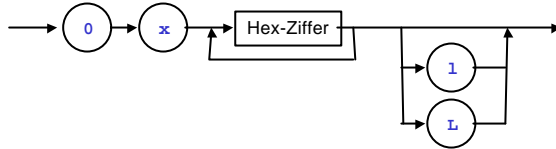
### Dezimal-Literal



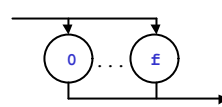
### Ziffer0



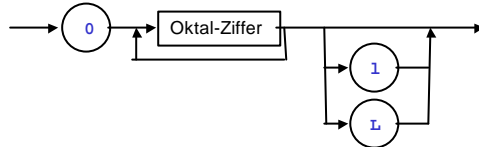
### Hexadezimal-Literal



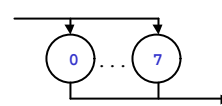
### Hex-Ziffer



### Oktal-Literal



### Oktal-Ziffer



Rudolf Berrendorf  
FH Bonn-Rhein-Sieg

Programmiersprache Java

37

## Beispiele

dezimal	hex.	Oktal	dezimal
0	0x0	00	= 12
17	0x11	021	= 17
33	0x21	041	= 33
12L	0xcL	014L	= 12

Rudolf Berrendorf  
FH Bonn-Rhein-Sieg

Programmiersprache Java

38

## Ausgesuchte Werte

	dezimal	hexadezimal	oktal
Minimaler Wert (int)	-2147483648 = $-2^{31}$	0x80000000 = $-2^{31}$	02000000000 = $-2^{31}$
Maximaler Wert (int)	2147483648 = $2^{31}$	0x7fffffff = $2^{31}-1$	01777777777 = $2^{31}-1$
-1 (int)	-1	0xffffffff	03777777777
Minimaler Wert (long)	-9223372036854775808L = $-2^{63}$	0x8000000000000000L = $-2^{63}$	01000000000000000000L = $-2^{63}$
Maximaler Wert (long)	9223372036854775808L = $2^{63}$	0x7fffffffffffffffL = $2^{63}-1$	07777777777777777777L = $2^{63}-1$
-1(long)	-1L	0xffffffffffffffffL	01777777777777777777L

Die dezimalen Werte  $2147483648=2^{31}$  und  $9223372036854775809=2^{63}$  (mit 32 bzw. 64 Bits nicht mehr im Zweierkomplement darstellbar) dürfen nur im Zusammenhang mit der Negation verwendet werden.

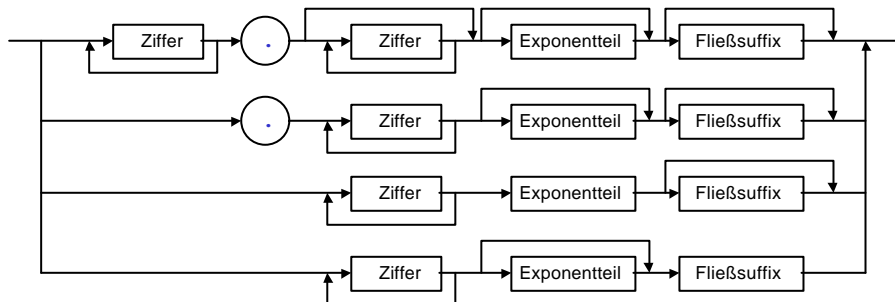
Rudolf Berrendorf  
FH Bonn-Rhein-Sieg

Programmiersprache Java

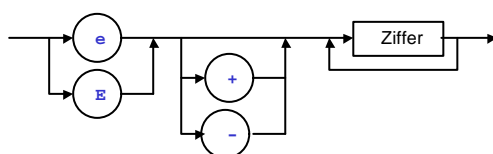
39

## Fließkomma-Literal

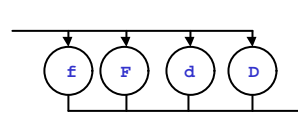
### Fließkomma-Literal



### Exponententeil



### Fließsuffix



Rudolf Berrendorf  
FH Bonn-Rhein-Sieg

Programmiersprache Java

40



## Beispiele

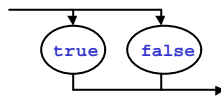
3.14	3.14f	3.14d	= 3.14
0.1	.1	1.e-1	= 0.1
10.0	10.	1e1	= 10.0

Wird nicht explizit ein Fließsuffix angehängt, so ist das Literal vom Typ `double`.

## Boolean- und null-Literal

Die beiden `booleschen Literale` stehen für die Wahrheitswerte wahr und falsch.

### Boolean-Literal



Das `null-Literal` kennzeichnet den leeren Zeiger im Zusammenhang mit Referenztypen.

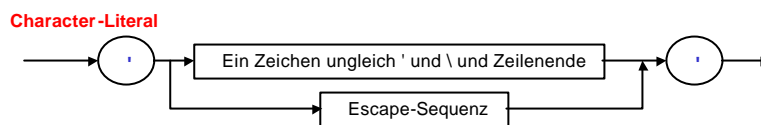
### null-Literal



## Character-Literal

Ein Character-Literal stellt **genau ein Zeichen des Unicode-Zeichensatzes** dar. Dabei sind auch Fluchtzeichenfolgen als ein Zeichen erlaubt, da diese schon in der ersten Phase des Übersetzens in Unicode-Zeichen umgewandelt werden (Schritt 1: Ersetzen von Fluchtzeichenfolgen in Unicode-Zeichen).

Weiterhin gibt es **Escape-Sequenzen** für besondere Kontrollzeichen.



### Beispiele:

'a' 'ä' '\u005c' (= 'Z') '\n'

## Escape-Sequenzen

<code>\b</code>	Backspace (BS)
<code>\t</code>	Horizontaler Tabulator (TAB)
<code>\n</code>	Zeilenvorschub (Line Feed, LF)
<code>\r</code>	Wagenrücklauf (Carriage Return, CR)
<code>\f</code>	Seitenvorschub (Form Feed, FF)
<code>\"</code>	"
<code>\'</code>	'
<code>\\</code>	\
<code>\0 bis \377</code>	Oktal-Code (wg. Kompatibilität zu C; geht auch mit Unicode)

### Beispiel:

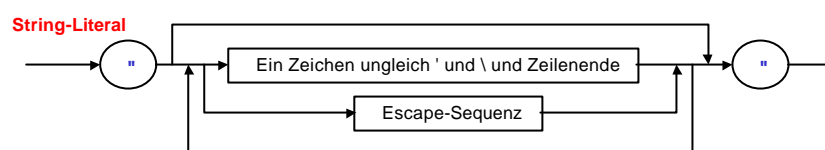
'\n' '\1'

## String-Literal

String- oder Zeichenketten-Literale sind **0 oder mehr Zeichen in Anführungszeichen eingeschlossen**. Längere Strings können in mehrere Teilstrings aufgebrochen werden, die durch + verkettet werden.

Der Typ solch eines Literals ist String, demzufolge ist jedes Zeichenketten-Literal eine **Referenz auf eine Instanz** der Klasse String!

**Wichtig:** 'a' ist ein Character, "a" ist ein String!



### Beispiele:

" " "Hallo" "Hallo\n" "Ätsch\n\tBätsch\n"

## Separatoren

Java kennt die folgenden **Separatoren** (Interpunktionszeichen):

(	)	
{	}	
[	]	
;		(Semikolon)
,		(Komma)
.		(Punkt)

## Operatoren

Java kennt die folgenden Operatoren:

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

### Wiederholung:

Zwischen den Zeichen dieser Operatoren dürfen keine Trennzeichen (Leerzeichen, Kommentare usw.) sein, sonst wären es zwei (oder mehr) Token!