

3. Java - Sprachkonstrukte I

Namen und Bezeichner, Variablen, Zuweisungen, Konstanten, Datentypen, Operationen, Auswerten von Ausdrücken, Typkonversionen

54

Definition: Namen und Bezeichner

Namen bezeichnen Entitäten in einem Programm wie zum Beispiel Variablen, Konstanten, Typen, Methoden oder Klassen.

Buch auf Seite 21

56

Lernziele

- Sie kennen die grundlegendsten Bausteine der Programmiersprache Java
- Sie verstehen den Einsatz von **Variablen** in einem Programm und können diese korrekt einsetzen
- Sie wissen, wie **Werte** direkt im Code beschrieben werden (**Literale**)
- Sie können einfache **arithmetische Ausdrücke** in Java lesen und interpretieren
- Sie erkennen den Zweck des **Typsystems** und können den Typ eines Ausdruckes bestimmen

55

Namen und Bezeichner

Erlaubte Namen für Entitäten im Programm:

- Namen beginnen mit einem **Buchstaben** oder den Zeichen **_** oder **\$**
- Danach optional eine Folge von **Buchstaben, Zahlen** oder den Zeichen **_** oder **\$**

57

Namen - was ist erlaubt

Valid identifiers (green background):

- _myName
- TheCure
- __AN\$WE4_1S_42__
- \$bling\$

Invalid identifiers (red background):

- me@home
- strictfp ?!
- 49ers
- side-swipe
- Ph.D's

58

Schlüsselwörter

Folgende Wörter werden von der Sprache bereits benützt und können deshalb nicht als Namen benützt werden:

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

59

Definition: Variablen

Variablen sind benannte Behälter für Werte und haben einen bestimmten **Typ**. Variablen müssen vor ihrer ersten Verwendung **deklariert** werden.

Buch auf Seite 23

60

Variablen

- Variablen sind **Behälter** für einen Wert
- Haben einen **Datentyp** und einen **Namen**
- Der Datentyp bestimmt, welche Art von Werten in der Variable erlaubt sind

`int x` `int y` `float f` `char c`

23

42

0.0f

'a'

Deklaration in Java:

```
int x = 23, y = 42;  
float f;  
char c = 'a';
```

↑
Initialisierung

61

Definition: Konstanten

Konstanten sind Variablen die bei ihrer Deklaration initialisiert werden und anschliessend ihren Wert nicht mehr ändern dürfen.

Buch auf Seite 35

Definition: Typen

Ein Typ definiert eine Menge von Werten, die zu diesem Typ gehören sowie eine Menge von Operationen, die mit den Werten des Typs ausgeführt werden dürfen.

Buch auf Seite 24

Konstanten

- Schlüsselwort **final**
- Der Wert der Variable kann genau einmal gesetzt werden

```
final int maxSize = 100;
```

Tip: Benützen Sie **final** immer, es sei denn der Wert muss tatsächlich veränderlich sein.

62

63

Definition: Standardtypen

Java bietet verschiedene vordefinierte Typen für diverse Zahlenbereiche sowie Wahrheitswerte und Zeichenketten.

Buch auf Seite 24

64

65

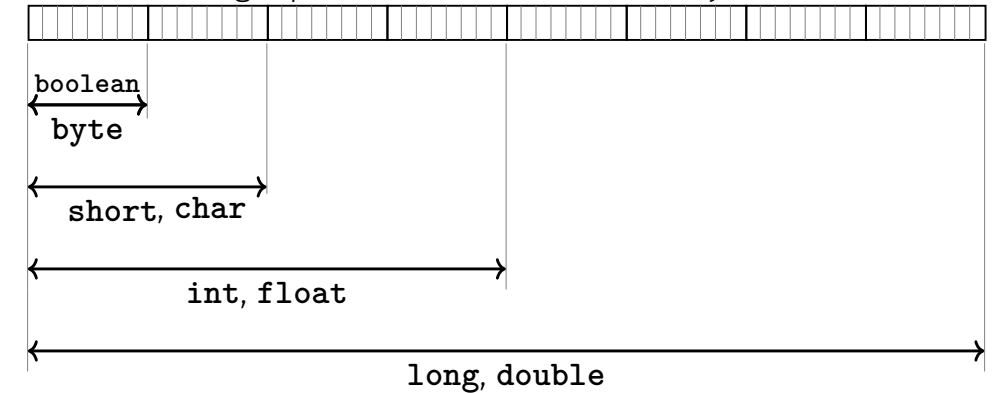
Standardtypen

Datentyp	Definition	Wertebereich	Initialwert
byte	8-bit Ganzzahl	$-128, \dots, 127$	0
short	16-bit Ganzzahl	$-32'768, \dots, 32'767$	0
int	32-bit Ganzzahl	$-2^{31}, \dots, 2^{31} - 1$	0
long	64-bit Ganzzahl	$-2^{63}, \dots, 2^{63} - 1$	0L
float	32-bit Fließkommazahl	$\pm 1.4E^{-45}, \dots, \pm 3.4E^{+38}$	0.0f
double	64-bit Fließkommazahl	$\pm 4.9E^{-324}, \dots, \pm 1.7E^{+308}$	0.0d
boolean	Wahrheitswert	true, false	false
char	Unicode-16 Zeichen	'\u0000', ..., 'a', 'b', ..., '\uFFFF'	'\u0000'
String	Zeichenkette	∞	null

66

Typen und Speicherbelegung

Zur Erinnerung: Speicherzellen enthalten 1 Byte = 8 bit



67

Definition: Literale

Repräsentation eines Wertes eines Standardtypen im Source Code.

Buch auf Seite 22 - 23

Literale: Ganzzahlen

- Typ **int** (oder **short, byte**)

12 : Wert 12
-3 : Wert -3

- Typ **long**

25_872_224L : Wert 25'872'224

Tip: Unterstriche zwischen Ziffern sind erlaubt!

68

69

Literale: Fließkommazahlen

unterscheiden sich von Ganzzahlliteralen durch Angabe von

- Dezimalkomma

1.0 : Typ **double**, Wert 1
 1.27f : Typ **float**, Wert 1.27

- und/oder Exponent.

1e3 : Typ **double**, Wert 1000
 1.23e-7 : Typ **double**, Wert $1.23 \cdot 10^{-7}$
 1.23e-7f : Typ **float**, Wert $1.23 \cdot 10^{-7}$

Literale: Zeichen und Zeichenketten

- Zeichen stehen in einzelnen Hochkommas

'a' : Typ **char**, Wert 97

- Zeichenketten stehen in doppelten Hochkommas:

"Hello There!" : Typ **String**
 "a" : Typ **String**

Achtung: Zeichen und Zeichenketten sind zwei unterschiedliche Dinge!

Zeichen: In ASCII Tabelle

0	<NUL>	32	<SPC>	64	@	96	`	128	À	160	†	192	€	224	†
1	<SOH>	33	!	65	A	97	a	129	Á	161	°	193	í	225	+
2	<STX>	34	"	66	B	98	b	130	Â	162	±	194	¬	226	,
3	<ETX>	35	#	67	C	99	c	131	Ã	163	£	195	√	227	~
4	<EOT>	36	\$	68	D	100	d	132	Ä	164	§	196	ƒ	228	‰
5	<ENQ>	37	%	69	E	101	e	133	Å	165	•	197	≈	229	ˆ
6	<ACK>	38	&	70	F	102	f	134	Û	166	¶	198	Δ	230	É
7	<BEL>	39	'	71	G	103	g	135	ä	167	ß	199	«	231	À
8	<BS>	40	(72	H	104	h	136	å	168	®	200	»	232	Ê
9	<TAB>	41)	73	I	105	i	137	ä	169	©	201	…	233	Ë
10	<LF>	42	*	74	J	106	j	138	å	170	™	202		234	Ï
11	<VT>	43	+	75	K	107	k	139	ä	171	ˆ	203	À	235	İ
12	<FF>	44	,	76	L	108	l	140	ä	172	ˆ	204	À	236	ı
13	<CR>	45	-	77	M	109	m	141	ç	173	±	205	Ö	237	ı
14	<SO>	46	.	78	N	110	n	142	é	174	Æ	206	œ	238	Œ
15	<SI>	47	/	79	O	111	o	143	ê	175	Ø	207	œ	239	Œ
16	<DLE>	48	0	80	P	112	p	144	ë	176	∞	208	-	240	•
17	<DC1>	49	1	81	Q	113	q	145	è	177	±	209	-	241	Œ
18	<DC2>	50	2	82	R	114	r	146	í	178	≤	210	ˆ	242	Œ
19	<DC3>	51	3	83	S	115	s	147	î	179	≥	211	ˆ	243	Œ
20	<DC4>	52	4	84	T	116	t	148	ï	180	Υ	212	ˆ	244	Œ
21	<NAK>	53	5	85	U	117	u	149	ı	181	μ	213	ˆ	245	ı
22	<SYN>	54	6	86	V	118	v	150	ñ	182	ð	214	+	246	ˆ
23	<ETB>	55	7	87	W	119	w	151	ó	183	Σ	215	◊	247	ˆ
24	<CAN>	56	8	88	X	120	x	152	ô	184	Π	216	ϣ	248	ˆ
25	<EMS>	57	9	89	Y	121	y	153	õ	185	π	217	Υ	249	ˆ
26	<SUB>	58	:	90	Z	122	z	154	ö	186	∫	218	/	250	ˆ
27	<ESC>	59	;	91	[123	{	155	ø	187	ø	219	€	251	ˆ
28	<FS>	60	<	92	\	124		156	ú	188	ø	220	€	252	ˆ
29	<GS>	61	=	93]	125	}	157	û	189	Ω	221	ˆ	253	ˆ
30	<RS>	62	>	94	^	126	~	158	ü	190	æ	222	fi	254	ˆ
31	<US>	63	?	95	_	127		159	ü	191	ø	223	fi	255	ˆ

Definition: Zuweisungen

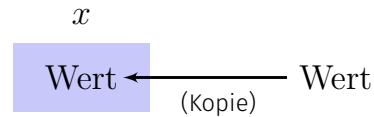
Eine Zuweisung wird dazu verwendet, einen (berechneten) Wert in einer Variablen zu speichern.

Buch auf Seite 27

Wertzuweisungen

Kopiert einen Wert in die Variable x

- In Pseudocode: $x \leftarrow$ Wert
- In Java: $x =$ Wert



"=" ist der Zuweisungsoperator **und nicht ein Vergleich!**

`int y = 42` ist also Deklaration + Wertzuweisung in einem.

Wertzuweisungen - Beispiel

```
int a = 3;
double b;
b = 3.141;
int c = a = 0;
String name = "Inf";
```

Eine **verschachtelte** Zuweisung:
Der Ausdruck `a = 0` speichert
den Wert 0 in Variable `a`. **und gibt
den Wert danach zurück**

Definition: Arithmetische Ausdrücke

Ein arithmetischer Ausdruck besteht aus Operanden und Operatoren und berechnet einen numerischen Wert eines bestimmten Typs.

Buch auf Seite 28

Arithmetische Binäre Operatoren

Infix Notation: $x \text{ op } y$ mit folgenden Operatoren

op: + - * / %
 ↑
 Modulo

Arithmetische Binäre Operatoren

- Division x / y : Ganzzahldivision falls x und y Ganzzahlen sind.
- Division x / y : Fließkommadivision falls x **oder** y eine Fließkommazahl ist.

Ganzzahldivision und Modulo

- $5 / 3$ ergibt 1 $-5 / 3$ ergibt -1
- $5 \% 3$ ergibt 2 $-5 \% 3$ ergibt -2

78

Arithmetische Zuweisung

$x = x + y$



$x += y$

```
x -= 3;           // x = x - 3
name += "x"      // name = name + "x"
num *= 2;        // num = num * 2
```

Analog für $-$, $*$, $/$, $\%$

79

Arithmetische Unäre Operatoren

Prefix Notation: $+x$ oder $-x$

Angenommen x ist 3

- $2 * -x$ ergibt -6
- $-x - +1$ ergibt -4

80

Inkrement/Dekrement Operatoren

Inkrement Operatoren $++x$ und $x++$ haben den gleichen **Effekt**: $x \leftarrow x + 1$. Aber unterschiedliche Rückgabewerte:

- **Präfixoperator** $++x$ gibt **neuen** Wert zurück:

```
a = ++x;     $\iff$     x = x + 1; a = x;
```

- **Postfixoperator** $x++$ gibt den **alten** Wert zurück:

```
a = x++;     $\iff$     temp = x; x = x + 1; a = temp;
```

Analog für $x--$ und $--x$.

81

Inkrement Operator - Beispiel

Angenommen `x` ist initial 2

■ `y = ++x * 3` ergibt: `x` ist 3 und `y` ist 9

■ `y = x++ * 3` ergibt: `x` ist 3 und `y` ist 6

82

Ausdrücke (Expressions)

- repräsentieren **Berechnungen**
 - sind entweder **primär**
 - oder **zusammengesetzt** ...
 - ...aus anderen Ausdrücken, mit Hilfe von **Operatoren**
 - sind statisch typisiert
- Analogie: Baukasten

83

Ausdrücke (Expressions) - Beispiel

primär: `"-4.1d"` oder `"x"` oder `"Hi"`

zusammengesetzt: `"x + y"` oder `"f * 2.1f"`

Der Typ von `"12 * 2.1f"` ist `float`

84

Celsius zu Fahrenheit

```
public class Main {  
    public static void main(String[] args) {  
        Out.print("Celsius: ");  
        float celsius = In.readFloat();  
        float fahrenheit = 9 * celsius / 5 + 32;  
        Out.println("Fahrenheit: " + fahrenheit);  
    }  
}
```

15° Celsius sind 59° Fahrenheit

85

Celsius zu Fahrenheit - Analyse

9 * celsius / 5 + 32

- Arithmetischer Ausdruck,
- enthält drei Literale, eine Variable, drei Operatorsymbole

Wie ist der Ausdruck geklammert?

Regel 2: Assoziativität

Arithmetische Operatoren (*, /, %, +, -) sind linksassoziativ: bei gleicher Präzedenz erfolgt Auswertung von links nach rechts.

```
9 * celsius / 5 + 32
```

bedeutet

```
((9 * celsius) / 5) + 32
```

Regel 1: Präzedenz

Multiplikative Operatoren (*, /, %) haben höhere Präzedenz ("binden stärker") als additive Operatoren (+, -).

```
9 * celsius / 5 + 32
```

bedeutet

```
(9 * celsius / 5) + 32
```

Regel 3: Stelligkeit

Unäre Operatoren +, - vor binären +, -.

```
9 * celsius / + 5 + 32
```

bedeutet

```
9 * celsius / (+5) + 32
```

Klammerung

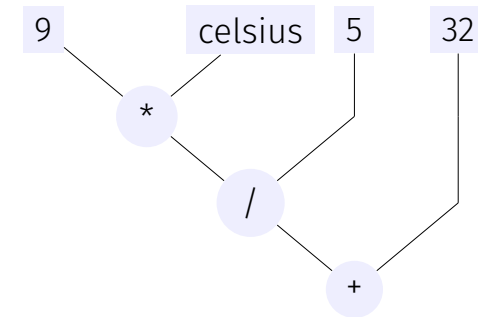
Jeder Ausdruck kann mit Hilfe der

- Assoziativitäten
 - Präzedenzen
 - Stelligkeiten (Anzahl Operanden)
- der beteiligten Operatoren eindeutig geklammert werden.

Ausdrucksbäume

Klammerung ergibt Ausdrucksbaum

$((9 * \text{celsius}) / 5) + 32$



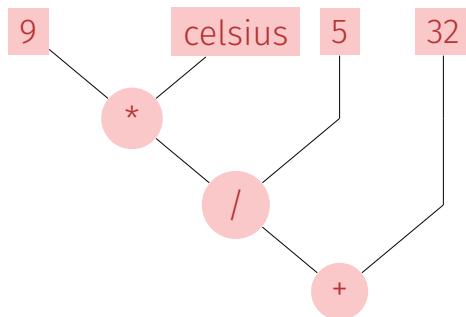
90

91

Auswertungsreihenfolge

"Von den Blättern zur Wurzel" im Ausdrucksbaum

$9 * \text{celsius} / 5 + 32$

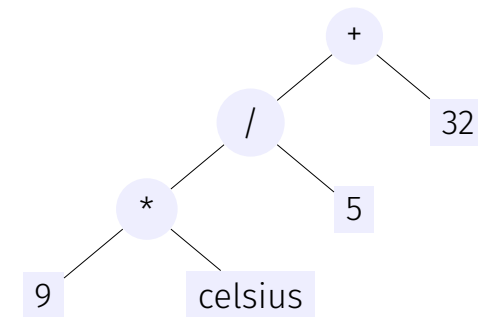


92

Ausdrucksbäume – Notation

Üblichere Notation: Wurzel oben

$9 * \text{celsius} / 5 + 32$



93

Definition: Typsystem

Ein Typsystem ist eine Menge Regeln, welche auf den diversen Konstrukten der Sprache angewandt wird.

Buch auf Seite 24

94

Typfehler - am Beispiel

```
int pi_ish;
float pi = 3.14f;

pi_ish = pi;
```

Compiler Fehler:

```
./Root/Main.java:12: error: incompatible types: possible lossy conversion
                    from float to int
    pi_ish = pi;
              ^
```

96

Typsystem

Java hat ein **statisches** Typsystem:

- Alle Typen müssen deklariert werden
- Wenn möglich wird die Typisierung vom Compiler geprüft ...
- ...ansonsten zur Laufzeit

Vorteil eines statischen Typsystems

- **Fail-fast** Fehler im Programm werden oft schon vom Compiler gefunden
- Verständlicher Code

95

Explizite Typkonvertierung

```
int pi_ish;
float pi = 3.14f;

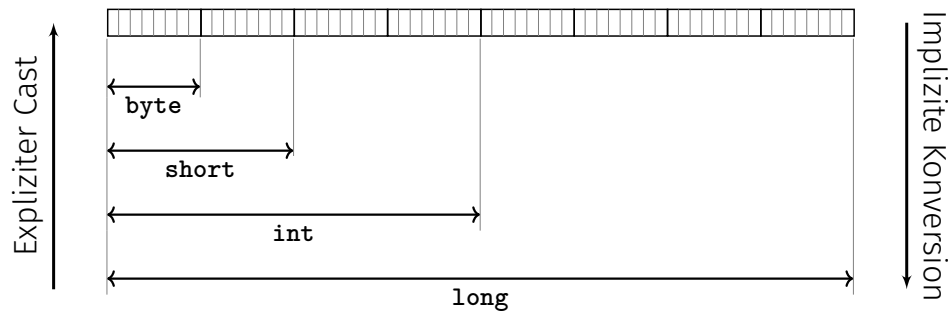
pi_ish = (int) pi;
```

Explizite Typkonvertierung mit Typecasting: (typ)

- Statisch typkorrekt, Compiler happy
- Laufzeitverhalten: Je nach Situation
 - Hier: **Genauigkeitsverlust**: $3.14 \Rightarrow 3$
- Kann das Programm zur Laufzeit zum Absturz bringen!

97

Typ Konvertierung - Anschaulich



Potentieller Informationsverlust bei explizitem Cast, da weniger Speicher zur Verfügung.

Definition: Gemischte Ausdrücke

Ein gemischter Ausdruck besteht aus Operanden mit unterschiedlichen Typen.

Buch auf Seite 70

98

99

Gemischte Ausdrücke, Konversion

- Fließkommazahlen sind allgemeiner als ganzzahlige Typen.
- In gemischten Ausdrücken werden ganze Zahlen zu Fließkommazahlen konvertiert.

```
9 * celsius / 5 + 32
```

100

Typkonversion bei binären Operationen

Bei einer binären Operation mit numerischen Operanden werden die Operanden nach folgenden Regeln konvertiert:

- Haben beide Operanden denselben Typ, findet keine Konversion statt
- Ist einer der Operanden **double**, so wird der andere nach **double** konvertiert
- Ist einer der Operanden **float**, so wird der andere nach **float** konvertiert
- Ist einer der Operanden **long**, so wird der andere nach **long** konvertiert
- Ansonsten: Beide Operanden werden nach **int** konvertiert

101