

3. Java - Sprachkonstrukte I

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

Namen und Bezeichner

Ein Programm (also Klasse) braucht einen Namen

```
public class SudokuSolver { ...
```

- Konvention für Klassennamen: *CamelCase* benutzen → Wörter sind zusammengeschrieben mit jeweils einem Grossbuchstaben

Erlaubte Namen für "Entitäten" im Programm:

- Namen beginnen mit einem *Buchstaben* oder `_` oder `$`
- Danach Folge von *Buchstaben*, *Zahlen* oder `_` oder `$`

Namen - was ist erlaubt

Allowed identifiers (green background):

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

Disallowed identifiers (red background):

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

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

Variablen und Konstanten

- 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

89

Konstanten

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

Beispiel

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

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

90

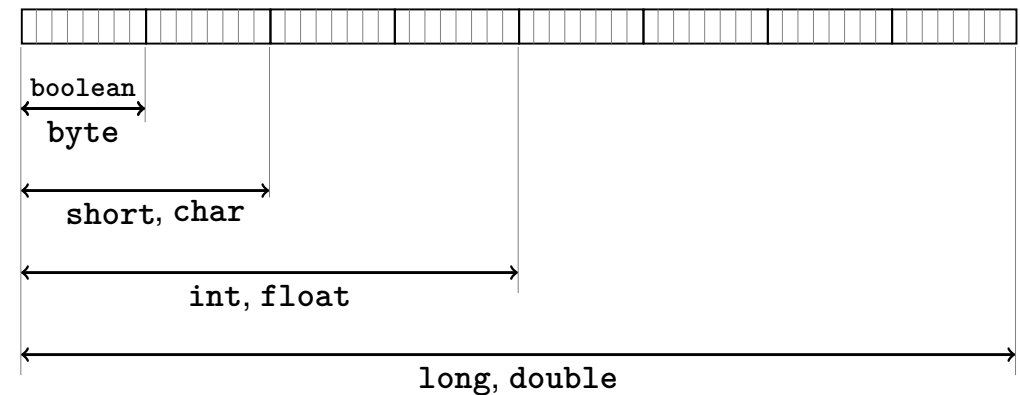
Standardtypen

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

91

Typen und Speicherbelegung

Zur Erinnerung: Speicherzellen enthalten 1 Byte = 8 bit



92

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 Zahlen sind erlaubt!

93

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 $1.23 \cdot 10^{-7}$

1.23e-7f

ganzzahliger Teil

Exponent

fraktionaler Teil

94

Literale: Zeichen und Zeichenketten

- Einzelne Zeichen:

'a' : Typ `char`, Wert 97

- Zeichenketten:

"Hello There!" : Typ `String`

"a" : Typ `String`

Achtung: Zeichen und Zeichenketten sind zwei unterschiedliche Dinge!

95

Zeichen: In ASCII Tabelle

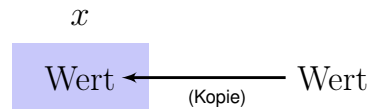
0	<NUL>	32	<SPC>	64	@	96	`	128	A	160	+	192	ç	224	#
1	<SOH>	33	!	65	A	97	a	129	À	161	°	193	ì	225	·
2	<STX>	34	"	66	B	98	b	130	Ç	162	¢	194	í	226	¸
3	<ETX>	35	#	67	C	99	c	131	È	163	£	195	√	227	¸
4	<EOT>	36	\$	68	D	100	d	132	Ë	164	§	196	f	228	‰
5	<ENQ>	37	%	69	E	101	e	133	Ö	165	•	197	≈	229	À
6	<ACK>	38	&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

96

Wertzuweisungen

Kopiert einen Wert in die Variable x

- In Pseudocode: $x \leftarrow \text{Wert}$
- In Java: $x = \text{Wert}$



“=” ist der Zuweisungsoperator *und nicht ein Vergleich!*

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

97

Wertzuweisungen

Beispiele

```
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*

98

Arithmetische Binäre Operatoren

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

op: + - * / %
 ↑
 Modulo

- **Präzedenz:** Punkt (und Modulo) vor Strich
- **Assoziativität:** Auswertung von Links nach Rechts

99

Arithmetische Binäre Operatoren

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

Beispiele

Ganzzahldivision und Modulo

- `5 / 3` ergibt 1 `-5 / 3` ergibt -1
- `5 % 3` ergibt 2 `-5 % 3` ergibt -2

100

Arithmetische Zuweisung

$x = x + y$



$x += y$

Beispiele:

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

Analog für $-$, $*$, $/$, $\%$

101

Arithmetische Unäre Operatoren

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

Präzedenz: Unäre Operatoren binden stärker als binäre.

Beispiele

Angenommen x ist 3

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

102

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:

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

Präzedenz: Inkrement und Dekrement binden stärker als unäre Operatoren.

Analog für $x--$ und $--x$.

103

Inkrement/Dekrement Operatoren

Beispiele

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

104

Ausdrücke (Expressions)

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

Analogie: Baukasten

Ausdrücke (Expressions)

Beispiele

primär: “-4.1d” oder “x” oder “Hi”

zusammengesetzt: “x + y” oder “f * 2.1f”

Der Typ von “12 * 2.1f” ist `float`

105

106

Celsius zu Fahrenheit

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

Beispiel: 15° Celsius sind 59° Fahrenheit

Celsius zu Fahrenheit - Analyse

`9 * celsius / 5 + 32`

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

Wie ist der Ausdruck geklammert?

107

108

Regel 1: Präzedenz

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

Beispiel

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

bedeutet

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

109

Regel 2: Assoziativität

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

Beispiel

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

bedeutet

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

110

Regel 3: Stelligkeit

Unäre Operatoren +, - vor binären +, -.

Beispiel

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

bedeutet

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

111

Klammerung

Jeder Ausdruck kann mit Hilfe der

- Assoziativitäten
- Präzedenzen
- Stelligkeiten (Anzahl Operanden)

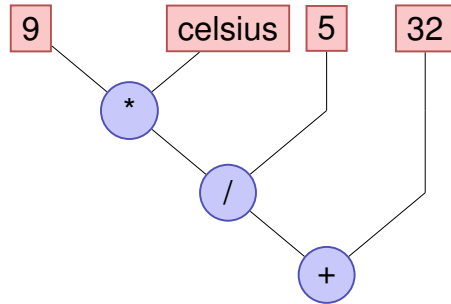
der beteiligten Operatoren eindeutig geklammert werden.

112

Ausdrucksbäume

Klammerung ergibt Ausdrucksbaum

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

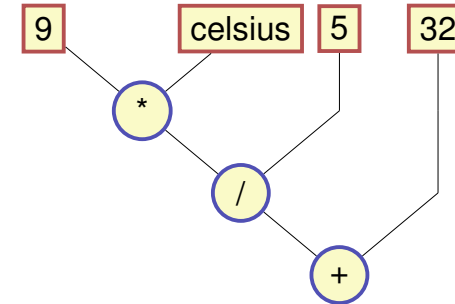


113

Auswertungsreihenfolge

"Von den Blättern zur Wurzel" im Ausdrucksbaum

`9 * celsius / 5 + 32`

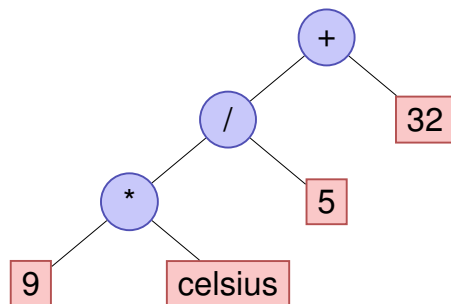


114

Ausdrucksbäume – Notation

Üblichere Notation: Wurzel oben

`9 * celsius / 5 + 32`



115

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

116

Typfehler

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;  
                ^
```

Explizite Typkonvertierung

Beispiel

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

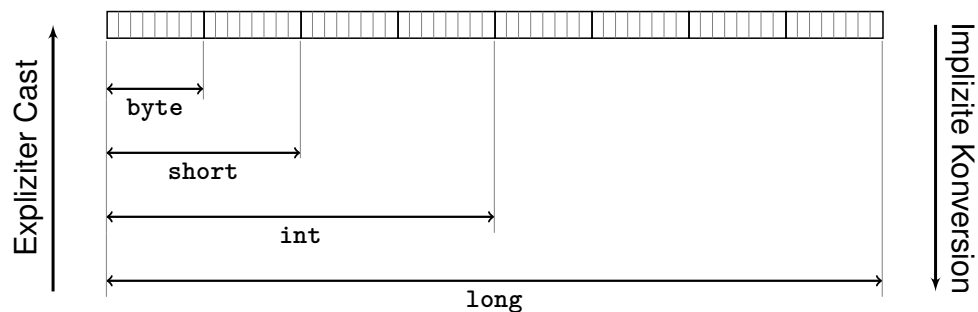
Explizite Typkonvertierung mit Typcasting: (typ)

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

117

118

Typ Konvertierung - Anschaulich für Ganzzahlen



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

Gemischte Ausdrücke, Konversion

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

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

119

120

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