II.1.2 Einfache Datentypen

Mittwoch, 4. November 2020 12:12

Primitive Datentypen: Grund-Datentypen v. Java, aus denen komplexere Datentypen aufgebaut werden.

Ganze Zahlen (byte, short, int, long)

Dienen zur Darstellung der Zahlen von

$\{-2^{8n-1}, \dots, 2^{8n-1}-1\}$, wobei

$\{-2^7, \dots, 2^7-1\}$ $\longrightarrow$

-128 $\quad$ 127

		Speicherplatz
byte:	$n=1$	8 Bit = 1 Byte
short:	$n=2$	16 Bit = 2 Byte
int:	$n=4$	32 Bit = 4 Byte
long:	$n=8$	64 Bit = 8 Byte

Zahlen werden im Zweierkomplement gespeichert.

Bei n Bits kann man die Zahlen von -2^{n-1} bis $2^{n-1}-1$ darstellen.

Bsp: $n=3$: Zahlen von $\underbrace{-2^2}_{-4}$ bis $\underbrace{2^2-1}_{3}$

	2^2	2^1	2^0
3	0	1	1
2	0	1	0
1	0	0	1
0	0	0	0
-1	1	1	1
-2	1	1	0
-3	1	0	1
-4	1	0	0

Stelle negative Zahl $-z$ als $2^n - z$ dar.

z.B. -4 wird dargestellt als

$2^3 - 4 = 4$

-3 wird dargestellt als

$2^3 - 3 = 5$

- Vorderste Bit gist Vorzeichen an (1 $\hat{=}$ negativ

- Vorderste Bit gist Vorzeichen an (1 ≙ negativ
  0 ≙ nicht negativ)
- Von z zu -z: Invertiere Ziffern und addiere 1.

Arithmetische Operationen werden auf Binärzahlen im Zweierkomplement ausgeführt.

$2^{31}-1$

```
int x = 2_147_483_647;
```

↑ steht für 2.147.483.647
bzw 2147483647

```
int y = 1;
x = x + y;
```
← x hat den Wert -2_147_483_648

Was macht Java?

```
  01....11
+ 00...01
----------
  10....00
```
← entspr. der Zahl -2^{31}

Überlauf!
Quelle von Prog-Fehlern.
Schlechter Stil: Ausnutzen v. Überläufen für Programmiertricks

---

Operationen auf Ganzzahlen: +, -, *, /, %, ...

/ : Ganzzahldivision, Rest wird weggelassen, d.h. 2/3 ergibt 0

% : Modulo / Rest der Division

- Operationen auf byte, short, int erzeugen int:

```
byte a = 1;
```

byte a = 1;

a = a + 1; Fehler!

Typ int

- Werte vom Typ "long" lassen sich durch "L" erzeugen:

1L
1l ist die Zahl 1 als "long" Wert

- jshell mit Typen: /set feedback verbose

Gleitkommazahlen (float, double)

-1.5 100. .57

0,57

1.2 e-23

$1{,}2 \cdot 10^{-23}$

1.2f

1,2 als float-Zahl

1.2 oder 1.2d

1,2 als double-Zahl

	Vorzeichen	Exponent	Mantisse	
float	1 Bit	8 Bit	23 Bit	} 32 Bit
double	1 Bit	11 Bit	52 Bit	} 64 Bit

Operationen: +, -, *, /, ...

Vorsicht: Rundungsfehler

Wahrheitswerte (boolean)

Wahrheitswerte (boolean)

Datentyp mit den Werten true, false.

Vordef. Operationen mit Ergebnis v. Typ boolean:

==, !=, >, >=, <, <=, ...

↑ Gleichheit ↖ Ungleichheit

Bsp: boolean b = (2 == 3);

(2 == 3) → false

= → Zuweisung

== → Gleichheit

Operationen auf Werten v. Typ boolean:

&& , || , !

&& ↑ und (Konjunktion)

|| ↑ oder (Disjunktion)

! ↑ nicht (Negation)

Bsp: false && true

&& und || werden von links nach rechts ausgewertet.
Wenn Ergebnis nach Auswertung des linken Arguments feststeht, wird rechtes Arg. nicht ausgewertet.

false && exp
true || exp

} Hier wird der Ausdruck exp nicht ausgewertet.

exp ↑ "expression"

"expression"

Zeichen (char)

'a', 'A', '1', '$', ...

'\n', ...

↑ Steuerzeichen für "newline"

Vordef. Operationen:

==, !=, <, <=, >, >=, ...

Vergleich der Zeichen anhand der Nummer im Unicode

Bsp: 'a' hat die Nr. 97

'b' hat die Nr. 98

'a' < 'b' ergist true

⇒ Vergleich nach alphabetischer Sortierung

Zeichenketten (Strings)

Kein primitiver Datentyp

- Notation mit "...": "hallo"
- Konkatenation mit +: "hal" + "lo" ergist "hallo"

Typkonversion

Java ist eine statisch getypte Sprache,
d.h. jeder Ausdruck hat einen Typ
und Operationen können nur auf Argumente d. richtigen Typen

u.v. jeder Ausdruck "

und Operationen können nur auf Argumente d. richtigen Typen angewendet werden. Typkorrektheit wird vor Laufzeit vom Compiler überprüft ("statisch").

=> keine Typüberprüfung zur Laufzeit nötig.

Manchmal möchte man Werte v. einem Typ in einen anderen konvertieren.

2 + 3 ✓ + verknüpft zwei int-Zahlen

2.3 + 3.5 ✓ + verknüpft zwei double-Zahlen

2 + 3.5 auch erlaubt:

int double

Hierzu wird 2 in den double-Wert 2.0 konvertiert.

=> 5.5 (v. Typ double)

Implizite Datentypanpassung

- vom speziellen zum allgemeinen Typ
- passiert automatisch, wenn Werte des allgemeineren Datentyps benötigt werden
- Konversion zu String passiert nicht automatisch, sondern nur in bestimmten Funktionen wie System.out.print oder +

```
int x = 'a';
```

int char — setzt x auf 97

```
int y = 'a' + 1;
```

setzt y auf 98

Explizite Datentypanpassung (Type Cast)

- Umwandlung von einem Typ zum anderen wird explizit erzwungen.
- Geht auch vom allgemeinen zum speziellen Typ.

```
int x = (int) 2.7;
```
2.7 ist double

setzt x auf 2

```
(int) 'a'
```
ergist 97

```
(float) 1 / 2
```
(float) 1 ergist 1.0f

2 wird automatisch in 2.0f konvertiert

ergist 0.5f

```
(float) (1 / 2)
```
(1 / 2) ergist 0 v. Typ int

ergist 0.0f

```
(char) ('a' + 1)
```
'a' wird konvertiert in 97

ergist 'b'

Type Cast nur möglich, wenn eine entsprechende Umwandlung zwischen den jeweiligen Typen definiert.