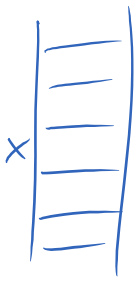
II.1.2 Einfache Datentypen

Samstag, 14. Oktober 2017 09:00

Für jede Variable muss bei Deklaration ihr Datentyp angegeben werden. Bei Var-Dekl. wird Speicherplatz für Werte dieses Typs reserviert.

```
int x;
```




```
x=10;
```

Variable ≙ symbolischer Name für Speicherplatz geeigneter Größe

Primitive Datentypen:

Typen für Grundwerte

String ist nicht primitiv.

Ganze Zahlen (byte, short, int, long)

- In Java können mit prim. Datentypen nur endlich viele ganze Zahlen dargestellt werden:

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

byte $n=1$ $\{-2^7, \dots, 2^7-1\} =$

byte $n=1$ $\{-2^7, \dots, 2^7-1\} =$
short $n=2$ $\{-128, \dots, 127\}$
int $n=4$
long $n=8$

Mit int kann man also
$\{-2^{31}, \dots, 2^{31}-1\}$ darstellen

Zur Darstellung von
byte benötigt man 8 bits
1 byte

short ..... 16 bits
2 byte

int 4 byte

Darstellung im Zweierkomplement

Mit m bits kann man die
Zahlen von $\{-2^{m-1}, \dots, 2^{m-1}-1\}$
darstellen.

Bsp: $m=3$

	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 $-z$
dar als $2^m - z$.

z.B.: -4 wird dargestellt
als $\underbrace{2^3 - 4}_{4}$

⇒ Vorderste Bit zeigt, ob die Zahl negativ.

-4 | 1 0 0

Zahl negativ.

Man kommt von z zu -z, indem Ziffern vertauscht und 1 addiert.

"Einer Komplement" ← Man verwendet Zweier- statt Einerkomplement, um eine eindeutige Darstellung der 0 zu haben.

Verständnis d. Zweierkomplements erklärt, was bei Operationen passiert, die die Grenzen des Datentyps überschreiten.

```
int x = 2147483647;
int y = 1
x = x+y;
```

```
 0 1 1 ....11
+0 0 0 ...01
------------
 1 0 0 0....00
```

Ergebnis: -2147483648

⇒ Gefahr für Prog.-Fehler

Überläufe sollte man bei guten Prog. vermeiden.

Jshell

- Möglichkeit, direkt Java-Ausdrücke auswerten zu lassen
- REPL = Read-Evaluate-Print-Loop

- Ganze Zahlen: +, -, *, /, %
  - / ↑ Ganzzahldivision
  - % ↑ Modulo

- long x = 5l;
  ↑ 5 als "long" Zahl

- +, -, ... auf byte oder short liefert als Ergebnis trotzdem ein int

Gleitkommazahlen (float, double)

float : 32 Bit
double : 64 Bit

Zahlen endhalten. (steht für das deutsche Komma)

1.5 .5

2.34 e-23 ← steht für $2{,}34 \cdot 10^{-23}$

Diese Zahlen haben den Typ double.

1.5f hat den Typ float.

Gleitkommazahl besteht aus

	Vorzeichen	Exponent	Mantisse
float	1 Bit	8 Bit	23 Bit
double	1 Bit	11 Bit	52 Bit

Da nur endlich viele rationale Zahlen darstellbar sind ⇒ Rundungsfehler möglich

Wahrheitswerte

Wahrheitswerte

Datentyp boolean hat Werte true und false.

Vordef. Operationen:

- >, <, >=, <=, ==, !=
  - = ist Zuweisung
  - == untersucht auf Gleichheit
  - != ungleich

Vergleichsoperationen

- &&, ||, !
  - && "und"
  - || "oder"
  - ! "nicht"
  - ...

- boolean x = 3 > 2;

  ⇒ x bekommt den Wert true

Zeichen (char)

- beliebige Unicode-Zeichen
- 'a', '3', 'ü', '\n'
  - '...' damit man das Zeichen 3 von der Zahl 3 unterscheiden kann
  - Steuer-Zeichen (newline)
- Vordef. Operationen:

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

  vergleicht Zeichen nach ihrem

Index im Unicode

- z.B. 'a' hat Index 97
  'b' — " — 98
  ⇒ 'a' < 'b' wertet zu true aus.

String

- Kein primitiver Datentyp
- vereinfachende Schreibweisen:
  "hallo"
  "hal" + "lo" (Konkatenation von Strings)
- Datenobjekte werden in Strings konvertiert, wenn sie Argument von System.out.print sind. (durch eine toString-Methode)

Typkonversion

- Java ist streng getypte Sprache, d.h. jeder Ausdruck hat einen Typ. Operationen dürfen nur auf Argumente des passenden Typs angewendet werden. Typkorrektheit wird beim Compilieren überprüft.
- Bsp: 2 + 3 = 5
  ↑ ↑ ↑

• Bsp: $2 + 3 = 5$

int int int

+ darf auf 2 int-Argumente angewendet werden u. ergibt dann ein int-Resultat.

$2.5 + 3.2 = 5.7$

double double double

$2 + 3.2$ erlaubt?

int double

+ verknüpft immer 2 Argumente des gleichen Typs u. liefert ein Resultat dieses Typs.

2 + 3.2 ist möglich, denn int-Werte können automatisch in double-Werte konvertiert werden:

2 wird in 2.0 konvertiert

$2 + 3.2$

$2.0 + 3.2 = 5.2$

double double double

```
2.7f + 3.2   = 5.9
 ↑     ↑      ↑
float  double double
```

wird zu

2.7 konvertiert

- automatische (implizite) Typkonvertierung vom speziellen zum allgemeinen Typ.
(Bei Konvertierung von long nach float können Rundungsfehler auftreten.)
Automat. Konvertierung nach String nur in speziellen Fällen (z.B System.out.print, +).
- Bsp: `int x = 'a';`
  `int` ← `char`

  ⇒ x hat den Wert 97

  `double y = 2;`
  `double` ← `int`

  ⇒ y hat den Wert 2.0
- Typkonvertierung vom allge-

meinen zum speziellen Typ ist ggf. auch möglich, aber nicht automatisch.

```
int x = 2.5;
    ↑     ↑
   int  double
```

Typfehler ↯

- Explizite Typkonversion:

```
(int) 2.5
```

Falls eine entsprechende Typumwandlg. bekannt ist, wandelt Java den Wert in den gewünschten Typ.

```
int x = (int) 2.5;
```

Danach hat x den Wert 2.

(Type Cast)

Bsp: (float) 1 / 2 — wird konvertiert in 2.0f — ergibt 0.5f

(float) (1 / 2) — ergibt 0.0f

(1 / 2): int / int → 0 (int)

Bsp: double x = 82.2, y = 8.5;

```
int   n = (int) (x/y);        n ist 9
                  9.67...
char  a = (char) (int) x;     a ist 'R'
                  82
      b = (char) (a + 1);     b ist 'S'
                  83
                  'R'
                  Typ char
                       int
```

Bsp:

```
true ? 1 : 2f
 ↑     ↑     ↑
boolean int  float
```

↑ erlaubt, denn 1 wird automatisch in 1.0f konvertiert. Ausdruck wird daher zu 1.0f ausgewertet.

Bei bedingten Ausdrücken

$Aus_1 \; ? \; Aus_2 : Aus_3$

muss Aus_1 Typ boolean haben u. Aus_2 u. Aus_3 müssen den gleichen Typ haben.

- Ein Operator f, der Argument von Typ t verlangt, darf auf Argumente v. Typ t' angewendet werden, falls es eine implizite Typanpassung von t' nach t gibt.