Datentypen und Variablen in Java

Bei der ***Deklaration*** (Definition/ Festlegung) einer Variablen muss stets ein bestimmter Datentyp angegeben werden. Dieser ist abhängig von den Werten, die die Variable annehmen soll. Außerdem entscheidet er darüber, wie viel Platz die Variable im Speicher belegt.

Die wichtigsten **Datentypen** sind folgender Tabelle zu entnehmen:

Primitive Datentypen

Typname	Beschreibung	Länge in Byte	Wertebereich
`boolean`	Boole'scher Wert (wahr oder falsch, 1 Bit)	1	`true, false`
`char`	einzelnes Zeichen (16 Bit)	2	alle Unicode-Zeichen, z. B. `'a', ..., 'z', 'A', ..., 'Z', '3', '%' ...` (Zeichen werden in einfache Hochkommata gesetzt)
`byte`	ganze Zahl (8 Bit)	1	$-2^7, \ldots, 2^7 - 1$ (−128, …, 127)
`short`	ganze Zahl (16 Bit)	2	$-2^{15}, \ldots, 2^{15} - 1$ (−32 768, …, 32 767)
`int`	ganze Zahl (32 Bit)	4	$-2^{31}, \ldots, 2^{31} - 1$ (−2 147 483 648, …, 2 147 483 647)
`long`	ganze Zahl (64 Bit)	8	$-2^{63}, \ldots, 2^{63} - 1$ (−9 223 372 036 854 775 808, …, 9 223 372 036 854 775 807)
`float`	Fließkommazahl (32 Bit)	4	$-3{,}4028 \cdot 10^{38}, \ldots, 3{,}4028 \cdot 10^{38}$; $-45 < \text{Exponent} \leq 38$
`double`	Fließkommazahl (64 Bit)	8	$-1{,}7977 \cdot 10^{308}, \ldots, 1{,}7977 \cdot 10^{308}$; $-324 < \text{Exponent} \leq 308$

Zusatz: der Datentyp „String" steht für Zeichenketten (Wörter/ Texte, die in Gänsefüßchen gesetzt werden) und ist nicht in obiger Liste aufgeführt, weil er zu den so genannten Referenztypen gehört (Genaueres später).

Eine **Variable** besteht immer aus

- einem **Namen** („Bezeichner"), der nicht mit einer Nummer beginnen darf, stattdessen mit einem *Kleinbuchstaben* starten sollte (Rest in *camelCase*)
- einem **Wert** und
- einem Platz im **Arbeitsspeicher** des Computers.

Dem Speicherplatz können nacheinander verschiedene Werte zugewiesen werden. Die erste Wertzuweisung wird dabei auch als ***Initialisierung*** bezeichnet!

Eine **lokale Variable** ist nur innerhalb der Prozedur ‚sichtbar', in der sie deklariert wurde sowie in allen untergeordneten Blöcken – nicht aber in anderen Prozeduren auf der gleichen oder höheren Ebene. Im Gegensatz dazu werden **globale Variablen** außerhalb einer Prozedur deklariert und sind somit innerhalb der ganzen Klasse sichtbar.

Auch ist in Java die Möglichkeit der Deklaration von **Konstanten** gegeben – dies sind Speicherplätze, denen nur ein einziges Mal ein fester, unveränderlicher Wert zugewiesen werden soll.
Konstante werden im Prinzip wie Variablen behandelt, bekommen aber das Schlüsselwort ***final*** vorangestellt. Der Name einer Konstanten sollte in *Großbuchstaben* geschrieben werden.

Beispiel:

```
public class DatentypenUndVariablen{
 public static void main (String args[]) {
  //Variablendeklarationen
  //Name - Start mit Kleinbuchstabe!
  //teils bereits initialisiert!
     int zahl=7; //int a; auch erlaubt
     double b=7.342;
     char c='z';
     String s="Sagt er doch";
     boolean o;
  //Konstantendeklaration
      final int Z=3;
  //Programm
  //o wird initialisiert
     o=true;
 //Ausgabe der Werte
     System.out.println(zahl);
      System.out.println(b);
     System.out.println(c);
      System.out.println(s);
      System.out.println(Z);
     System.out.println(o);
  }
}
```

Übung: Entwickle eigenständig ein ähnliches Beispiel!

Operationen auf Datentypen

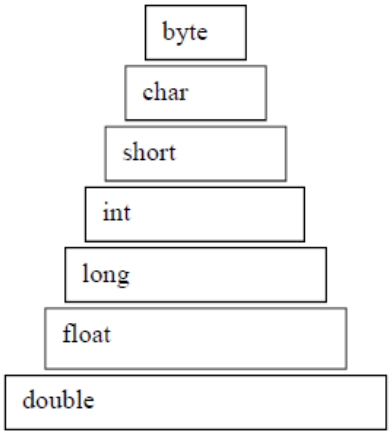
Operator	Erläuterung	Bemerkung/Beispiel
+	positives Vorzeichen	`+i` ist gleichbedeutend mit `i`.
–	negatives Vorzeichen	`-i` dreht das Vorzeichen von `i` um.
+	Addition	`x + y` ergibt die Summe von x und y.
–	Subtraktion	`x - y` ergibt die Differenz von x und y.
*	Multiplikation	`x * y` ergibt das Produkt von x und y.
/	Division	`x / y` ergibt den Quotienten von x und y. Achtung: Sind x und y beide ganzzahlig, so ist auch x / y ganzzahlig, d.h., `x / y` liefert die ganzzahlige Division ohne Rest (Beispiel: `7 / 3` liefert 2). Ist x oder y jedoch ein Fließkommawert, so ist auch x / y ein Fließkommawert (Beispiel: `7.0 / 3` liefert 2.3333333333333333).
%	modulo	`x % y` ergibt den ganzzahligen Rest bei Division von x durch y. (`7 % 3` liefert 1.)
++	Inkrement	`i++` entspricht `i = i + 1` und erhöht den Wert von i um 1.
--	Dekrement	`i--` entspricht `i = i - 1` und erniedrigt den Wert von i um 1.
=	Zuweisung	`x = y` weist x den Wert von y zu.
==	Vergleich	`x == y` ergibt wahr, wenn x gleich y ist bzw. wenn bei Referenztypen beide Werte auf dasselbe Objekt zeigen.
<	kleiner	`x < y` ergibt wahr, wenn x kleiner ist als y.
<=	kleiner gleich	`x <= y` ergibt wahr, wenn x kleiner oder gleich y ist.
>	größer	`x > y` ergibt wahr, wenn x größer ist als y.
>=	größer gleich	`x >= y` ergibt wahr, wenn x größer oder gleich y ist.
!= <>	ungleich	`x != y` (oder `x<>y`) ergibt wahr, wenn x ungleich y ist bzw. wenn bei Referenztypen beide Werte auf verschiedene Objekte zeigen.
!	logisches NICHT	`!x` ergibt wahr, wenn x falsch ist und umgekehrt.
&&	logisches UND	`x && y` ergibt wahr, wenn sowohl x als auch y wahr sind.
||	logisches ODER	`x || y` ergibt wahr, wenn mindestens einer der beiden Ausdrücke x oder y wahr ist.
^	exklusives ODER	`x^y` ergibt wahr, wenn x wahr ist und zugleich y falsch oder umgekehrt.
new	`new`-Operator	zur Erzeugung von Objekten
instanceof	`instanceof`-Operator	`x instanceof y` liefert wahr, wenn x eine Instanz der Klasse y oder einer ihrer Unterklassen ist. So lässt sich herausfinden, zu welcher Klasse ein bestimmtes Objekt gehört.

Verwendung unterschiedlicher Datentypen

Bei einer Zuweisung (z.B. x = a + 2) wird der Wert des Ausdrucks rechts vom Gleichheitszeichen der links stehenden Variablen zugewiesen. Diese Zuweisung erfolgt allerdings nur dann korrekt, wenn die Datentypen auf beiden Seiten typkompatibel sind.

Wenn wir uns die Datentypen double, float, long, int, short, char und byte als Behälter vorstellen, so haben diese Behälter eine unterschiedliche Größe – siehe Abbildung rechts.

Es gilt dabei, dass ein kleinerer Behälter in einen größeren gepackt werden kann, aber nicht umgekehrt.



Alle Zuweisungen eines kleineren in einen größeren Datentyp werden akzeptiert und bewirken automatisch eine Typumwandlung des Ergebnisses in den größeren Datentyp. Außerdem ist bei Berechnungen auf den Wertebereich des Datentyps zu achten.

Zuweisungen von einem größeren in einen kleineren Datentyp werden entweder vom Compiler nicht akzeptiert oder es werden keine korrekten Ergebnisse erzeugt.

Beispiele:

int x = 1/2

bewirkt, dass der Wert von x gleich 0 wird, da der Nachkommateil des eigentlichen Wertes 1/2 = 0,5 abgeschnitten wird, weil x nach Deklaration vom Typ Integer sein soll!

double x = 7

bewirkt, dass der byte-Wert 7 als double-Wert gespeichert wird, d.h. x = 7.0

Explizite Typumwandlung durch Casts

Durch sog. Type casts kann man einen Wert in einen anderen Datentyp umwandeln. Diesen Vorgang nennt man ***Casten***. Das Casten erfolgt durch den **Castoperator,** der dem zu castenden Wert den neuen Datentyp in Klammern voranstellt.

Beispiele:

```
int n;
short k;
double d = 3.1415;
    n = (int) d;          –   wandelt double nach int um – n = 3
    k = (short) 6.4567    –   wandelt float nach short um – k = 6
```

```
int b = 65;
System.out.println((char)(b + 1));
```

b + 1 ergibt 66. Anschließend wird wegen des Operators (char) das Zeichen mit der Nummer 66 auf dem Bildschirm ausgegeben, nämlich das Zeichen B