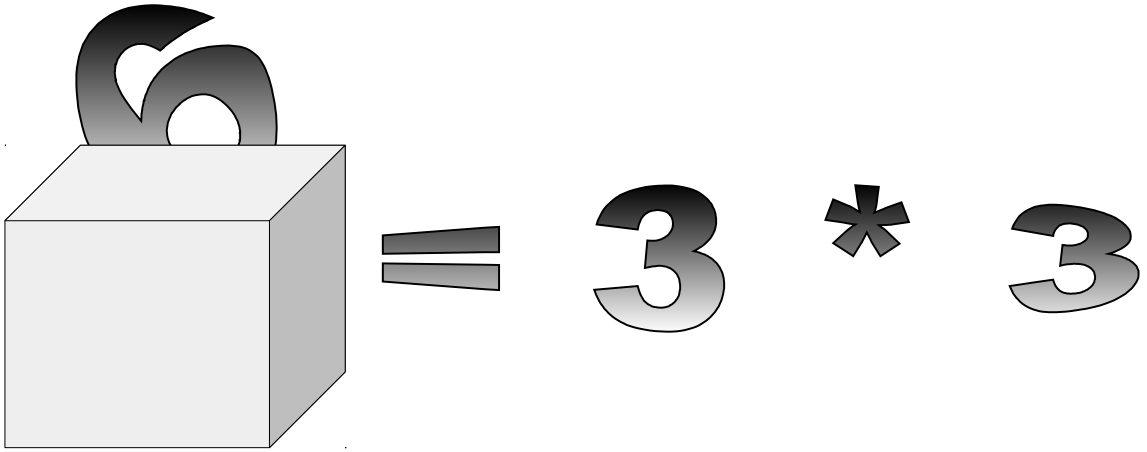
Java - Zahlen, Wahrheitswerte und Zeichen



Kommentare

- Hilfe für den Entwickler.
- Wer hat wann welche Änderung vorgenommen?
- Warum werden diese Anweisungen hier ausgeführt?
- Bei Codeänderungen müssen Kommentare angepasst werden.
- Der Compiler überliest Kommentare bei der Ausführung.

Beispiel

- Warum wird an dieser Position im Code diese Berechnung durchgeführt? Nicht: Wie werden die Werte berechnet?
- Warum wird der String an diesem Zeichen getrennt? Nicht: Wie wird der String getrennt?

Einzeiliger Kommentar

```
// Einzeiliger Kommentar
```

- Beginn: Zwei Schrägstriche ohne Leerzeichen.
- Der Kommentar endet automatisch mit der Zeile.

… nach einer Anweisung

```
final double umrechnung = 0.5707; // Braunschw. Elle
```

- Die Anweisung links vom Kommentar wird erläutert.
- Erläuterung von komplizierten Berechnungen.
- Hinweise auf die Nutzung der deklarierten Variablen und Konstanten.

… als Platzhalter in einer leeren Methode

```
public static void main(String[] args) {
        // TODO code application logic here
}
```

- Hinweis auf die, zu implementierende Aktivität.
- Hinweis auf fehlenden Code.

Mehrzeiliger Kommentar ...

```
/*
   Mehrzeiliger
   Kommentar
*/
```

- Beginn des Kommentars: `/*`.
- Ende des Kommentars: `*/`.
- Die Zeilen zwischen Beginn und Ende werden vom Compiler überlesen.

… am Anfang einer Methode

```
/*
  Umrechnung der Braunschweiger Elle in Meter
  Parameter: Länge der Elle (double)
  Return: Meterangabe (double)
*/
public static double elleToMeter(double elle)
```

- Welche Aktion ist in der Methode implementiert?
- Welche Parameter müssen der Methode übergeben werden?
- Was für ein Wert gibt die Methode zurück?

… am Anfang einer Klasse

```
/*
    Umrechnung von alten Maßeinheiten
    Autor: Benutzer
    Geändert am: 01.03.2017
*/
public class UmrechnungMasse {
```

- Welche Objekte symbolisiert die Klasse?
- Wer hat die Klasse implementiert?
- Wann wurde diese Klasse implementiert?
- Dokumentation der letzten Änderung.

Anweisungen

```
float summe;
summe = 7 / 2;
System.out.println(summe);
```

- Deklaration von Variablen und Konstanten.
- Berechnung von variablen Werten.
- Aufruf von Methoden.

Ende einer Anweisungen

```
float summe;
summe = 7 / 2;
System.out.println(summe);
```

- Jede Anweisung endet mit dem Semikolon.
- Pro Zeile können beliebig viele Anweisungen stehen. Aber es sollte nie mehr als eine Anweisung stehen.

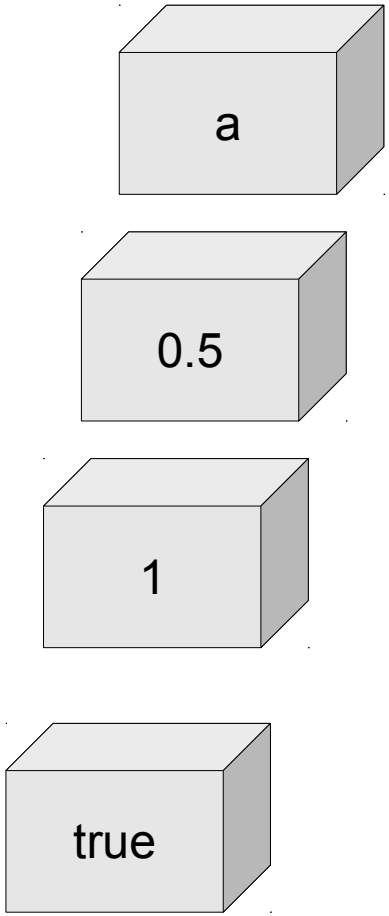
Block von Anweisungen

```
public static void main(String[] args) {
    float summe;
    summe = 7 / 2;
    System.out.println(summe);
}
```

- Zusammenfassung von Anweisung.
- Beginn und Ende mit den geschweiften Klammern.
- Zum Beispiel beschreibt die Methode mit Hilfe eines Blocks von Anweisungen eine bestimme Aktion.

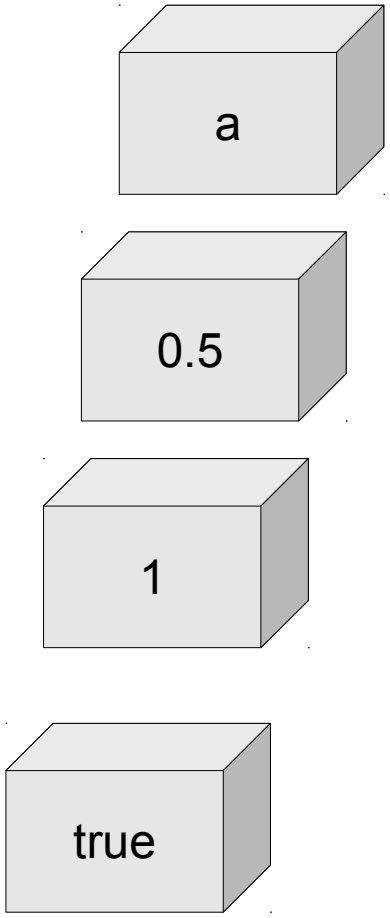
Primitive Werte in Java

- Ganz- oder Dezimalzahlen, die direkt in dem Code stehen oder berechnet werden.
- Einzelne Buchstaben oder Zahlen, die als Text interpretiert werden.
- True (Wahr) oder False (Falsch) als Ergebnis eines Vergleichs von zwei Werten.



Variablen für primitive Werte.

- Container für variable Werte.
- Speicherung eines einfachen Typs von Wert.
- Der zu speichernde Wert kann berechnet oder direkt im Code eingegeben werden.

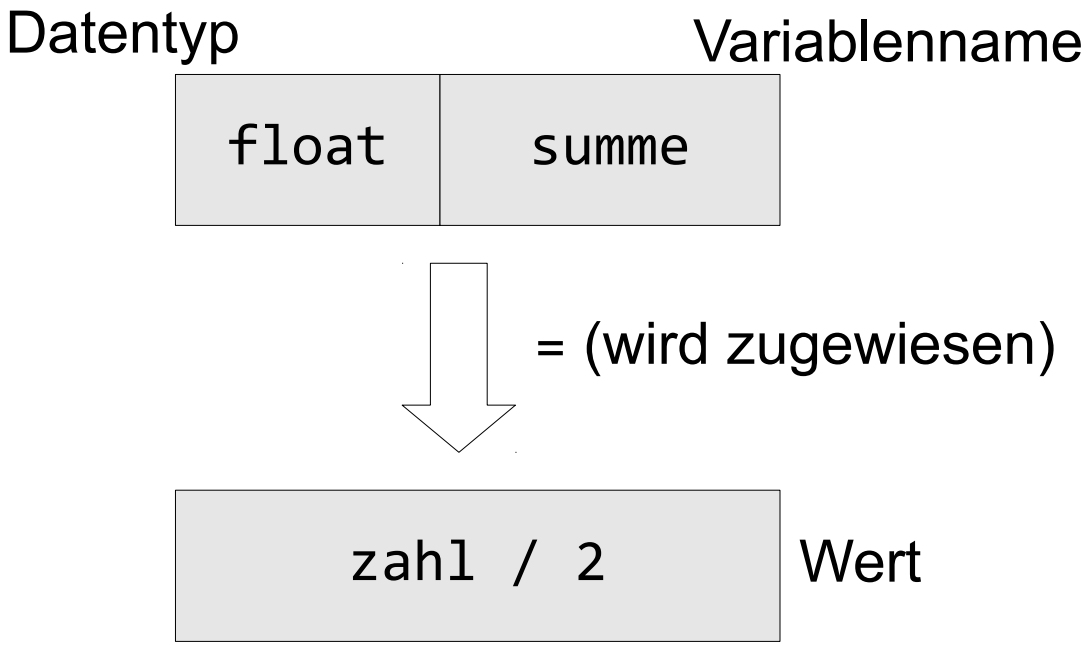


Beispiel

1	int x
0.5	float y
true	boolean wahr
a	char zeichen

```
{
    float summe;
    int zahl;

    Zahl = 7;
    summe = zahl / 2;
    System.out.println(summe);
 }
```

Merkmale einer Variablen



Variablennamen

- Label für ein Container.
- Kennzeichnung einer Position im Speicher.
- Jeder Name kommt nur einmal in einem Block vor. Der Block von Anweisungen beginnt und endet mit den geschweiften Klammern.
- Namen für Variablen beginnen mit einem Kleinbuchstaben.

Benutzerdefinierte Namen

- Namen kennzeichnen eindeutig eine Variable, Methode etc.
- Namen sind Platzhalter für Werte und Referenzen in einem Programmcode.
- Namen von Methoden und Klassen sind Platzhalter für Blöcke von Anweisungen. Mit Hilfe des Namens wird ein Block von Anweisungen gestartet.

Benutzerdefinierte Namen in Java

- Zusammensetzung einer Zeichenkette aus beliebigen Buchstaben, Ziffern, Unterstrich und dem Dollarzeichen.
- Nutzung des Unicode-Zeichensatzes.
- In einem benutzerdefinierten Namen sind keine Leerzeichen erlaubt.

Konventionen

- Die Klein- und Großbuchstaben A ... Z und die Zahlen 0 … 9 werden genutzt.
- Jeder Variablennamen beginnt mit einem Buchstaben.

Weitere Regeln

- Jeder Name ist in seinem Gültigkeitsbereich einmalig. In einem Block von Anweisungen ist jeder benutzerdefinierte Namen eindeutig.
- Schlüsselwörter aus der Programmiersprache dürfen nicht genutzt werden.
- Java beachtet die Groß- und Kleinschreibung.

Zusammengesetzte Namen

- Die Wörter in dem Namen kreisradius werden durch Unterstriche getrennt. Der Variablenname kreis_radius wird für die Angabe des Kreisradius genutzt.
- Die Wörter in dem Namen kreisradius werden mit Hilfe der KameI-Notation hervorgehoben. Der Variablenname kreisRadius wird für die Angabe des Kreisradius genutzt.

Geeignete Variablennamen

- Speicherung eines Kreisradius. Name der Variablen: radius.
- Angabe des minimalen Wertes. Name der Variablen: min_wert.
- Temperaturwert in Celcius. Name der Variablen: temperaturCelcius.
- Speicherung der Autofarbe: color, colorAuto. Ungeeignet sind Namen wie autoGelb, morriesFarbe.
- Zähler können mit nur einen Buchstaben benannt werden.

Deklaration von Variablen

`float`	`summe`	`;`
`Datentyp`	`variablenname`	`;`

- Der Variablenname ist frei wählbar.
- Entsprechend des angegebenen Datentyps wird Speicher bereitgestellt.
- Der Datentyp legt die Größe des Containers fest.

Hinweise

- Vor der Nutzung muss eine Variable in einer Methode deklariert werden.
- Nicht deklarierte Variablen lösen den Fehler `cannot find symbol` in der IDE NetBeans aus.

Initialisierung von Variablen

`summe`	`=`	`7`
`variablenname`	`=`	`wert`

- Mit Hilfe des Gleichheitszeichens wird einer Variablen ein Anfangswert zugewiesen.
- Der Variablen wird erstmalig entsprechend ihres Datentyps befüllt.

Hinweise

- Vor der Nutzung muss eine Variable in einer Methode initialisiert werden.
- Nicht deklarierte Variablen lösen den Fehler `variable zahl might not have been initialized` in der IDE NetBeans aus.

Zuweisungsoperator

`summe`	`=`	`7`	`;`
`variablenname`	`=`	`wert`	`;`

- Das Gleichheitszeichen ist ein Operator, der einen Container befüllt.
- Der Wert rechts vom Gleichheitszeichen wird der Variablen links vom Gleichheitszeichen zugewiesen.
- Die Variable links vom Gleichheitszeichen hat einen bestimmten Typ. Der zu zu weisende Wert muss entsprechend dieses Typs interpretierbar sein.

… deklarieren und initialisieren

`float`	`summe`	`=`	`7`	`;`
`Datentyp`	`variablenname`	`=`	`wert`	`;`

Nutzung

```
public class Java02_PrimitiveDatentypen {

    public static void main(String[] args) {
        float summe;
        int zahl;

        zahl = 7;
        summe = zahl / 2;
        System.out.println(summe);
    }
}
```

Die deklarierten Variablen können in dem Block, beginnend ab hier, genutzt werden.

Konstanten

```
final double Bs_Elle = 0.5707;
double meter;
double elle;

elle = 45.0;
meter = elle * Bs_Elle;
```

- Konstanten sind Platzhalter für Werte, die nicht durch den Code verändert werden.
- Symbole für Werte, die immer wieder im Code genutzt werden.

Konstanten

`final`	`double`	`Bs_Elle`	`=`	`0.5707`	`;`
`final`	`Datentyp`	`Konstanten-Name`	`=`	`wert`	`;`

- Deklarationen von Konstanten beginnen mit dem Schlüsselwort `final`.
- Konstanten müssen gleichzeitig deklariert und initialisiert werden.

Konstanten-Namen

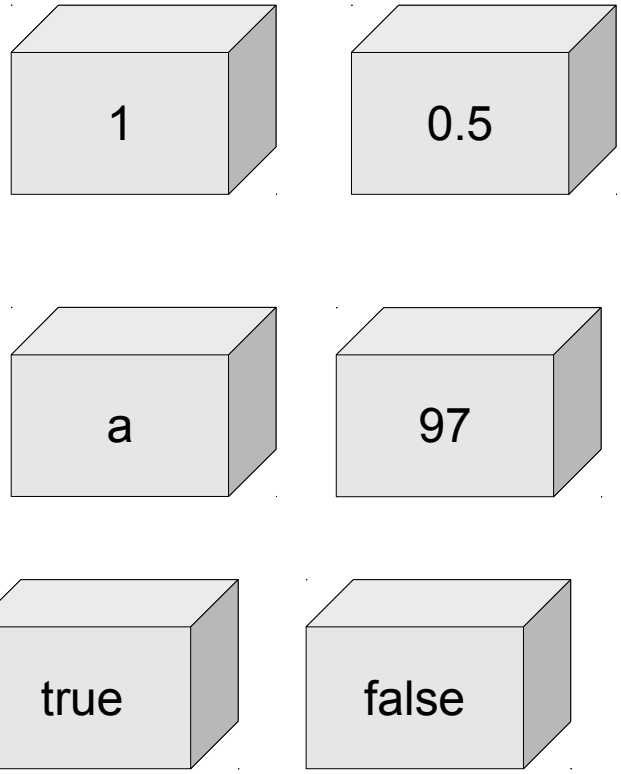
- Label für ein Container. Der Inhalt des Containers kann nicht verändert werden.
- Jeder Name kommt nur einmal in einem Block vor. Der Block von Anweisungen beginnt und endet mit den geschweiften Klammern.
- Konstanten-Namen nutzen die Großbuchstaben. Die Wörter in dem benutzerdefinierten Namen werden durch den Unterstrich getrennt.

Primitive Datentypen

- Beschreibung der Standard-Werttypen. Welche Art von Wert wird in der Variablen gespeichert?
- Die Größe des Containers (Speichers) für einen Wert wird festgelegt.
- Wie wird der Wert verarbeitet?
- Auflistung aller primitiven Datentypen: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html.

... in Java

- Abbildung von Ganz- oder Dezimalzahlen für Berechnungen und so weiter. Zahlen sind in Java immer vorzeichenbehaftet.
- Einzelne Buchstaben oder Zahlen, die in einem Zeichensatz kodiert sind. Java arbeitet mit dem Unicode-Zeichensatz.
- Abbildung von Fragen, die mit Ja oder Nein beantwortet werden können. In einer grafischen Oberfläche werden Kontrollkästchen zur Darstellung genutzt.



Boolean

```
boolean wahr = true;
boolean falsch = false;
boolean vergleich;

vergleich = (3 > 5);
```

- Zwei Zustände: wahr (true) oder falsch (false).
- Entspricht einem Lichtschalter, der das Licht ein- oder ausschaltet.
- Beantwortung von Ja / Nein-Fragen. Ist der Wert A kleiner als der Wert B?

Ganzzahlen

```
byte bZahl;
short shZahl;
int intZahl;
long lngZahl;

bZahl = 10; // Dezimale Schreibweise
```

- Zahlen ohne Nachkommastellen.
- Der Datentyp legt den Wertebereich der Ganzzahl fest.

Datentypen

Datentyp	Wertebereich	Speicherbedarf
`byte`	-128 - 127	8 Bit
`short`	-32768 - 32767	16 Bit
`int`	-2.147.483.648 – 2.147.483.647	32 Bit
`long`	-9.223.372.036.854.775.808 - 9.223.372.036.854.775.807	64 Bit

Überschreiten des Wertebereichs

```
byte bZahl;

bZahl = 0x000A * 0x000D;
```

- Syntaxfehler: `incompatible types: possible lossy conversion from int to byte.`

Schreibweise von Literalen

```
byte bZahl;
short shZahl;
int intZahl;
long lngZahl;

bZahl = 0x00A; // Hexadezimale Schreibweise
bZahl = 012;    // Oktale Schreibweise
bZahl = 0b1010; // binäre Schreibweise
bZahl = 10; // Dezimale Schreibweise
```

Zahlensysteme

- Dezimalsystem auf der Basis von 10.
- Hexadezimalsystem auf der Basis von 16. Die Zahlen von 0 bis 9 und die ersten Buchstaben des Alphabets (A...F).
- Binärzahlen 0 und 1 zur Darstellung von Zuständen.
- Oktalsystem auf der Basis von 8. Für die Darstellung von Zahlen werden die Ziffern 0 bis 7 genutzt.

Literale vom Datentyp „Long“

```
int intZahl;
long lngZahl;

intZahl = 2147483647;
lngZahl = 2147483647;
```

- Der Datentyp `long` erweitert den Datentyp `int`.

Mögliche Fehlerquelle

```
int intZahl;
long lngZahl;

intZahl = 2147483647;
lngZahl = 2147483650;
lngZahl = 2147483650L;
```

- Fehler: `integer to large`, wenn das Literal den Datentyp `int` überschreitet, obwohl es in einer Variablen vom Datentyp `long` gespeichert werden soll.
- Literale vom Datentyp `long` müssen mit dem Buchstaben L gekennzeichnet werden.

Gleitkommazahlen

- Zahlen mit Nachkommastellen.
- Als Dezimaltrennzeichen wird der Punkt genutzt.
- Der Datentyp gibt die Genauigkeit an. Gleitkommazahlen nähren sich in Abhängigkeit der Genauigkeit einem Wert.

Datentypen

Datentyp	Wertebereich	Speicherbedarf
`float`	-3,4 * 10^38 bis 3,4 * 10^38	32 Bit
`double`	-1,7 * 10^308 bis 1,7 * 10^308	64 Bit

Datentyp	Genauigkeit
`float`	Einfache Genauigkeit. Bis auf 7 Stellen genau.
`double`	Doppelte Genauigkeit. Bis auf 16 Stellen genau.

Dezimal- oder Expontialschreibweise

```
double dblZahl;

dblZahl = 730.0;
dblZahl = 700.0;

dblZahl = 7.3e2;
dblZahl = 7.0E2;
```

Dezimaltrennzeichen „Komma“

```
double dblZahl;

dblZahl = 0,6;
```

- Fehler: `; excepted`.
- Ein Komma als Dezimaltrennzeichen wird als Syntaxfehler angezeigt.

Überlauf bei Gleitkommazahlen

```
double dblZahl;

dblZahl = 4.7E308;
```

- Fehler: `floating point number to large.`

Literale vom Datentyp „Float“

```
float fltZahl;

fltZahl = 0.6F;
fltZahl = 0.1234567F;
```

- Fehler: `possible lossy conversion from double to float`.
- Werte vom Datentyp float werden standardmäßig als double interpretiert.
- Literale vom Datentyp `float` müssen mit dem Buchstaben F gekennzeichnet werden.

Ein einzelnes Zeichen

- Speicherung eines einzeln Zeichens in Abhängigkeit der Zeichenkodierung.
- Alphanumerische Zeichen wie zum Beispiel „a“.
- Zahlen von 0 bis 65535.
- Escape-Sequenzen wie '\n' für einen Zeilenumbruch.

Character (ein Zeichen)

```
char buchstabe = 'A';
char ziffer = 65;
```

- Alphanumerische Zeichen aus dem ASCII-Zeichensatz beginnen und enden mit einem Apostroph.
- Die Ziffer 65 spiegelt die Dezimalkodierung des Buchstaben „A“ in dem ASCII-Zeichensatz wieder.

Zeichenkodierung

- ASCII (American Standard Code for Information Interchange). Definition von 128 Zeichen.
- ANSI oder Latin-1-Zeichensatz. Darstellung von 256 Zeichen. Die ersten 128 Zeichen entsprechen den ASCII-Zeichensatz.
- Siehe http://cs.stanford.edu/people/miles/iso8859.html.

Hinweise

- Das erste Zeichen im Zeichensatz wird mit Hilfe von `'\u0000'` angegeben.
- Die ersten 256 Zeichen des UTF-8-Zeichensatzes entsprechen dem ISO 8859-1 (Latin 1)-Zeichensatz.
- Siehe https://unicode-table.com/de/.

Unicode-Zeichen

```
char buchstabe = 'A';
char ziffer = 65;
char unicode = '\u0041';
```

- Unicode-Zeichen beginnen und enden mit einem Apostroph.
- \u maskiert die nachfolgende Zeichenfolge als Kodierung in einem Unicode-Zeichensatz.

Hinweise

- Das erste Zeichen im Zeichensatz wird mit Hilfe von `'\u0000'` angegeben.
- Die ersten 256 Zeichen des UTF-8-Zeichensatzes entsprechen dem ISO 8859-1 (Latin 1)-Zeichensatz.

Zeichencodes in NetBeans

- Rechtsklick auf das Projekt.
- Auswahl von *Properties* im Kontextmenü.
- Die Kategorie *Sources* ist aktiv.
- Das Kombinationsfeld *Encoding* auf der Seite rechts, unten gibt Auskunft über den genutzten Zeichensatz.

Ausdruck

- Nach bestimmten Regeln werden Operanden und Operatoren zusammengesetzt. Leerzeichen zwischen Operanden und Operatoren erhöhen die Lesbarkeit.
- Berechnung eines Wertes mit Hilfe von Operanden und Operatoren.
- Ausdrücke können geklammert werden. Die runden Klammern erhöhen die Lesbarkeit bei komplexen Ausdrücken und verändern die Rangfolge der Operatoren.

Operanden

- Variablen, die einen dynamischen Wert speichern.
- Konstanten, die einen festen Wert speichern.
- Literale. Statische Werte, die direkt in einer Anweisung stehen. Direkte Darstellung von Zahlen, Zeichenfolgen etc. im Ausdruck.

Operatoren

- Regeln zur Berechnung von Werten.
- Verknüpfung von Operanden.
- Vorschriften zur Bildung von Ausdrücken aus mehreren Operanden.

Operatoren in Java

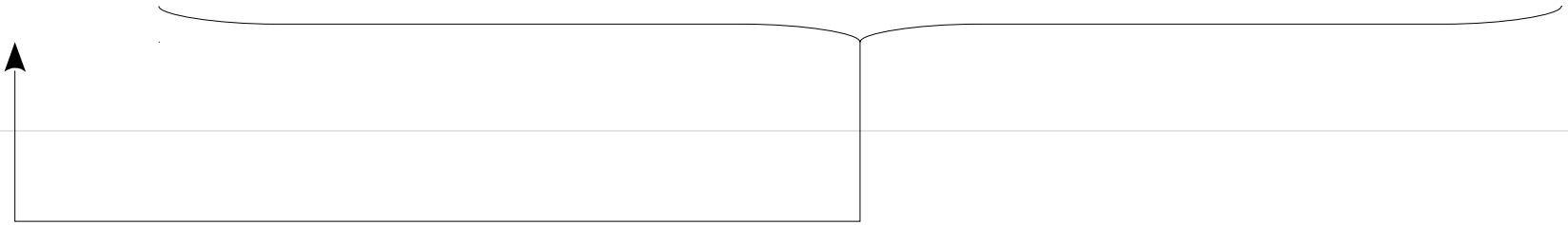
- Arithmetische Operatoren berechnen mit Hilfe eines Ausdrucks einen Wert. Der Wert wird einer Variablen zugewiesen, in einer Tabelle gespeichert etc.
- Operatoren vergleichen zwei Werte und geben einen boolschen Wert zurück. Falls der Vergleich stimmt, wird true (wahr) zurückgegeben. Andernfalls wird false (falsch) zurückgegeben. Bedingte Anweisungen und Schleifen in Java nutzen Vergleichsoperatoren.
- Logische Operatoren verknüpfen Ausdrücke, die einen boolschen Wert zurückgeben. Logische Operatoren wie AND, OR und NOT verknüpfen Bedingungen in Anweisungen.

Arithmetische Operatoren

Operator	Berechnung	Beispiel
+	Addition	3 + 4 = 7
-	Subtraktion	3 – 4 = -1
*	Multiplikation	3 * 4 = 12
/	Division	3 / 4 = 0.75
%	Division mit Rest	3 % 4 = 3

Zuweisungsoperator

```
double dblZahl = 0.3;
int intZahl = 4;
double ergebnis = 0;

ergebnis = dblZahl * dblZahl + 2 * dblZahl * intZahl + intZahl * intZahl;
```

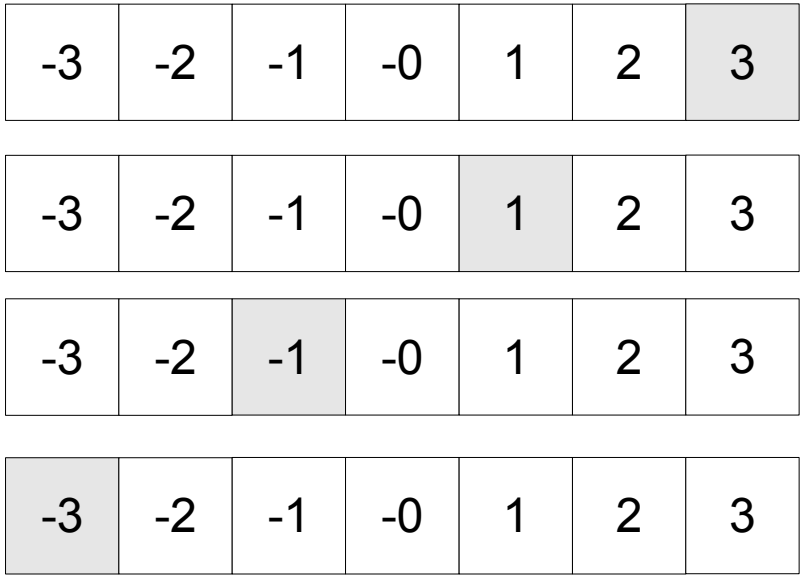


Hinweise

- Zuerst wird der Ausdruck berechnet. Falls in dem Ausdruck eine Gleitkommazahl vorkommt, wird das Ergebnis der Berechnung als Gleitkommazahl interpretiert. Fall in dem Ausdruck nur Ganzzahlen genutzt werden, ist das Ergebnis auch vom Typ „Ganzzahl“.
- Das Ergebnis des Ausdrucks wird der Variablen links vom Gleichheitszeichen zugewiesen. Das Ergebnis muss entsprechend des Datentyps der Variablen interpretiert werden können.

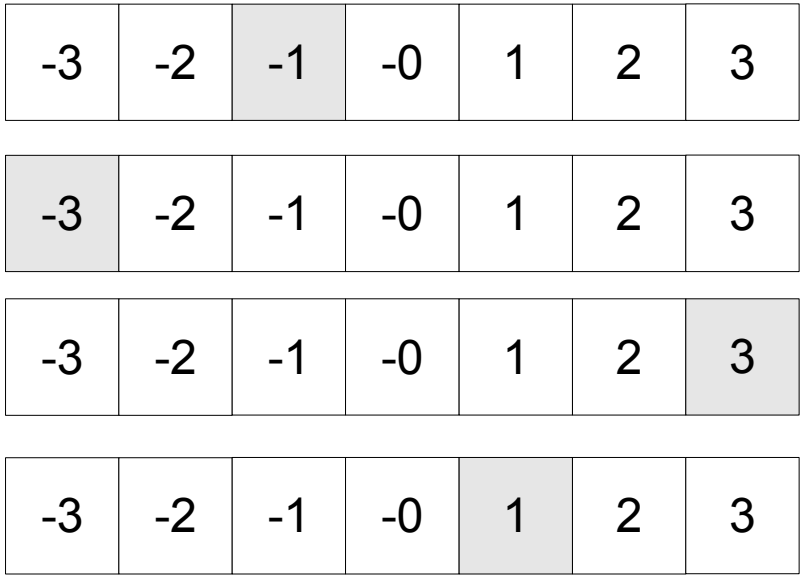
Addition von negativen und positiven Zahlen

```
int intZahl;

intZahl = +1 + +2;

intZahl = -1 + +2;

intZahl = 1 + -2;

intZahl = -1 + -2;
```



Subtraktion von negativen und positiven Zahlen

```
int intZahl;

intZahl = +1 - +2;

intZahl = -1 - +2;

intZahl = 1 - -2;

intZahl = -1 - -2;
```



Vorzeichenbehaftete Division und Multiplikation

- Haben beide Zahlen im Ausdruck ein positives Vorzeichen, so ist auch das Ergebnis des Ausdrucks positiv.
- Hat einer der beiden Zahlen im Ausdruck ein negatives Vorzeichen, so ist das Ergebnis des Ausdrucks negativ.
- Haben beide Zahlen im Ausdruck ein negatives Vorzeichen, so ist auch das Ergebnis des Ausdrucks positiv.

Ganzzahlen: Überlauf des Wertebereich

```
short sZahl;

sZahl = 5000 * 300;
```

- Es wird ein Syntaxfehler `incompatible types` angezeigt.

Gleitkommazahlen: Überlauf des Wertebereich

```
double dblZahl;

dblZahl = 4.7 * 1.0E308;
```

- Es wird kein Fehler angezeigt.
- Die Variable hat den Wert `Infinity` (unendlich).

Ganzzahlen: Division durch Null

```
int intZahl;

intZahl = 4 / 0;
```

- Eine Division durch Null ist nicht erlaubt.
- Ausnahmefehler: `java.lang. ArithmeticException: / by zero`.

Gleitkommazahl: Division durch Null

```
double dblZahl;

dblZahl = 4.0 / 0;
```

- Eine Division durch Null ist nicht erlaubt.
- Es wird kein Fehler angezeigt. Die Variable hat den Wert `Infinity`. Die Zahl ist unendlich.

Division mit Rest

```
double dblZahl;
int intZahl;

dblZahl = 23.5 % 7;
intZahl = 23 % 7;
intZahl = -23 % 7;
```

- Anwendung auf Gleitkomma- und Ganzzahlen.
- Der Rest ist negativ, wenn der Operand links vom Modulo-Operator negativ ist.

Rangfolge von Operatoren

```
double dblZahl = 0.3;
int intZahl = 4;
double ergebnis = 0;

ergebnis = dblZahl * dblZahl + 2 * dblZahl * intZahl +
           intZahl * intZahl;
ergebnis = (dblZahl * dblZahl) + (2 * dblZahl * intZahl) +
           (intZahl * intZahl);

ergebnis = (dblZahl * (dblZahl + 2) * dblZahl * intZahl) +
           (intZahl * intZahl);
```

Hinweise

- Punktrechnung geht vor Strichrechnung.

*	Multiplikation
/	Division
%	Division mit Rest
+	Addition
-	Subtraktion

Explizite Typ-Umwandlung

```
int intZahl;
double dblErgebnis;

intZahl = 7;
dblErgebnis = intZahl / 2;

intZahl = 7;
dblErgebnis = intZahl / 2.0;
```

- Zuerst wird der Ausdruck berechnet.
- Anschließend wird der Wert des Ausdrucks in den entsprechenden Datentyp automatisch konvertiert.

Hinweise

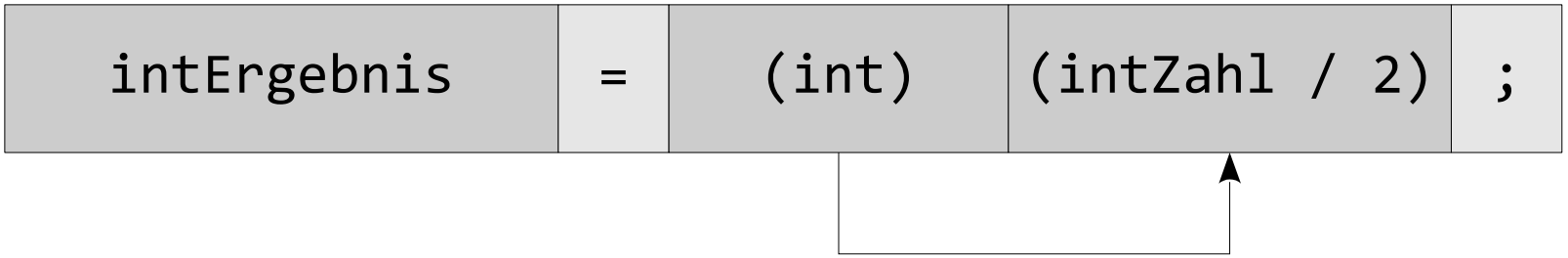
- Eine Umwandlung von byte → short → int → long oder float → double ist möglich.
- Eine Umwandlung zum Beispiel von double → float erzeugt einen Fehler. Eine Umwandlung von einem Datentyp mit einem größeren Wertebereich / Genauigkeit erzeugt einen Fehler.

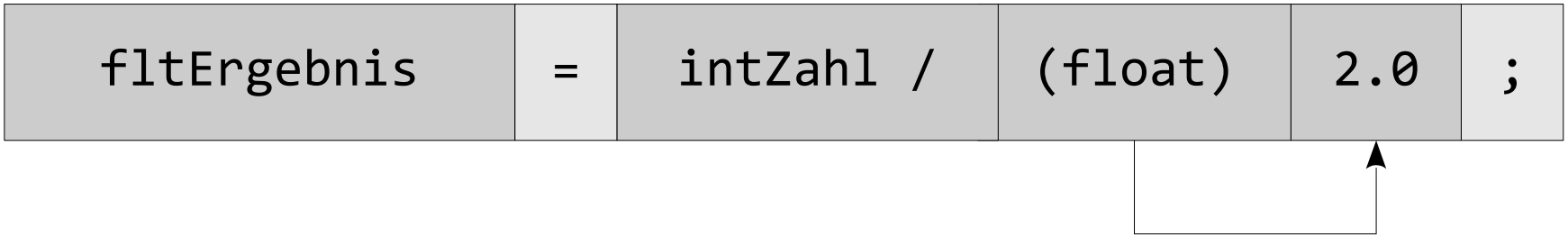
Implizite Typ-Umwandlung

```
int intZahl;
int intErgebnis;

intZahl = 7;

intErgebnis = (int) (intZahl / 2);
```

- In runden Klammern wird der Datentyp angegeben, in dem der Wert konvertiert werden soll.
- Es können alle primitiven Datentypen genutzt werden.

Beispiele





„Rundungsfehler“ bei Gleitkommazahlen

```
double dblXWert;
double dblYWert;
double dblErgebnisX;
double dblErgebnisY;

dblXWert = 10.007;
dblYWert = 10.009;

dblErgebnisX = dblXWert + 0.001;
dblErgebnisY = dblYWert - 0.001;
```

Arbeiten mit Dezimalzahlen

```
package java02_primitivedatentypen;
import java.math.BigDecimal;
public class Java02_PrimitiveDatentypen {
    public static void main(String[] args) {
        BigDecimal differenz = new BigDecimal( "0.001" );
        BigDecimal a = new BigDecimal( "10.007" );
        BigDecimal b = new BigDecimal( "10.009" );
        BigDecimal ergebnisA = new BigDecimal("0");
        BigDecimal ergebnisB = new BigDecimal("0");

        ergebnisA = a.add(differenz);
        ergebnisB = b.subtract(differenz);
    }
}
```

Import der Bibliothek

```
import java.math.BigDecimal;
```

- Die Bibliothek `java.math.BigDecimal` wird dem Programm bekannt gemacht.
- Die passende Bibliothek für Dezimalzahlen wird im Kopf des Programms eingebunden.

Deklaration der Variablen

```
BigDecimal differenz = new BigDecimal( "0.001" );
```

- Die Variable hat den Namen `differenz`.
- Die Variable nutzt keinen primitiven Datentyp. Die Variable basiert auf der Klasse `BigDecimal`. Die zu speichernden Werte werden entsprechend des Bauplans `BigDecimal` abgelegt.
- Mit Hilfe von `new` wird eine Instanz von der Klasse erstellt. In Abhängigkeit des Bauplans wird ein Objekt erstellt.
- Als Anfangswert wird der Instanz der Wert 0.0001 als Zeichenkette übergeben. Zeichenketten beginnen und enden immer mit den Anführungszeichen.

Deklaration der Variablen

```
ergebnisA = a.add(differenz);
```

- Eine Instanz einer Klasse kann Methoden nutzen, um die Attribute zu verändern.
- Der Punktoperator verbindet die Instanz mit der Methode.
- In diesem Beispiel wird die Methode `.add` genutzt, um den Wert in dem Objekt `a` mit dem Wert des Objekts `differenz` zu addieren.