

Prof. Dr. Verena Majuntke

# Grundlagen der Programmierung

## Datentypen

10.11.2020

# Grundlagen der Programmierung

Grundwissen, um Java-Programme aufzubauen

- Programmiersprache Java
- Java-Programme
- Datentypen
- Variablen: Initialisierung und Wertzuweisung
- Operatoren und Ausdrücke
- Literale

# Lernziele

- Welche Eigenschaften hat die Programmiersprache Java?
- Wie erstelle ich ein ausführbares Java-Programm?
- Wie sieht die Struktur einer Java-Datei aus?
- Was ist eine Variable?
- Welche Datentypen gibt es und welche Wertebereiche haben sie?
- Wie initialisiere ich Variablen und wie weise ich ihnen Werte zu?
- Wie bilde ich logische und arithmetische Ausdrücke?

# Programmiersprache Java

# Die Programmiersprache Java

## Einfach

- Überschaubarer Sprachumfang und klare Syntax

## Objektorientiert

- Java ist eine objektorientierte Sprache

## Dynamisch

- anpassbar zur Laufzeit

## Verteilt

- Client-/Server-Anwendungen

# Die Programmiersprache Java

## Robust

- Java-Code ist robust gegen Fehleranfälligkeit – umfangreiches Exception-Handling

## Sicher

- Byte-Code-Verifizierung und Sicherheitskonzept („Sandbox“-Prinzip)

## Performant (JIT-Compiler)

- Steigerung der Performance

# Die Programmiersprache Java

## Interpretiert

- Die JVM (Java Virtual Machine) interpretiert den Byte-Code

## Architekturneutral (Byte-Code)

- Der Java-Compiler erzeugt Byte-Code, der auf jeder JVM lauffähig ist

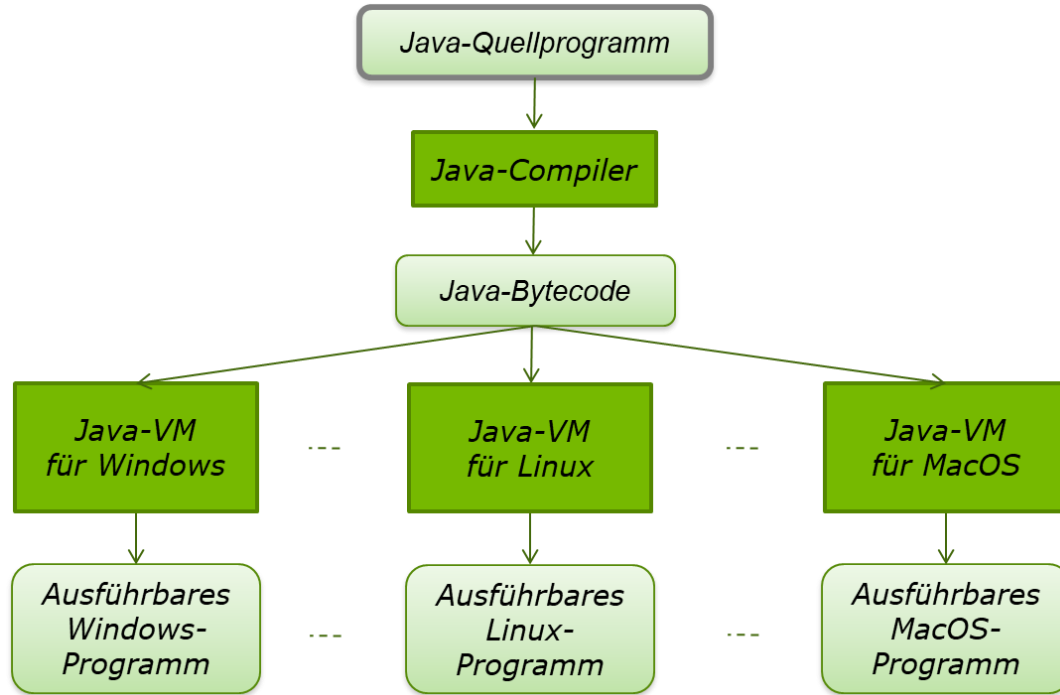
## Portabel (Quelltext)

- Java-Quellcode lässt sich auf jeder JVM kompilieren

## Nebenläufig (Threads)

- Java-Threads laufen (pseudo-)parallel zur gleichen Zeit ab

# Quellprogramm – ausführbares Programm





# Was ist ein Programm?

- Eine Menge von **Anweisungen** und **Ausdrücken** in einer Programmiersprache
- Die **Syntax** legt den korrekten Aufbau des Programms und die gültigen Zeichen fest
- **Syntaxfehler** werden vom Compiler erkannt und angezeigt

```
import util.MeineTabelle;

public class Start {

    /**
     * @param args
     */
    public static void main(String[] args) {

        MeineTabelle mtabelle = new MeineTabelle();
        mtabelle.gibausUntereAsciiTabelle();
        System.out.println(" ----- ");
        mtabelle.gibausObereAsciiTabelle();
    } // end of main
} // end of class
```

# Typische Anweisungen

- **Ausgabe** von Zeichen, Wörtern, Sätzen und Zahlen
- **Eingabe** von Zeichen, Wörtern, Sätzen und Zahlen
- Durchführung einer **Berechnung**
- **Konvertieren** einer Zahl in ein Zeichen
- ...

# Beispiel einer Java-Datei

```
// Die folgende Klasse gehört zum Packet ub01
package ub01;

// Importiert alle Klassen aus dem Paket java.io
import java.io.*;

// Deklariert die von Außen zugreifbare Klasse HalloWelt
public class HalloWelt {

    // Deklariert objektinstanzunabhängig die nicht von Außen zugreifbare das Attribut Alter
    private static int alter=46;

    // Deklariert objektinstanzunabhängig die nicht von Außen zugreifbare Methode main an
    public static void main(String[] args) {

        // Gibt über die Methode println der Klasse out "Hallo Welt" aus
        System.out.println("Hallo Welt");

        // Gibt das Alter aus
        System.out.println("Ich bin " + alter + " Jahre alt.");

        // Schliesst die Methode main ab
    }

    // Schliesst die Klasse HalloWelt ab
}
```

# Variablen, Datentypen und Operatoren

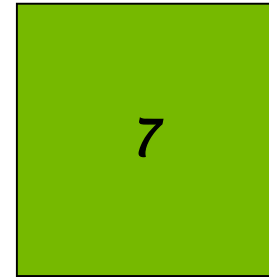
# Was gehört noch zu Java?

In einer Programmiersprache wie Java gibt es

- Variablen
- Kontrollstrukturen
- Methoden
- Parameter
- Operatoren
- und vieles mehr...

# Was ist eine Variable?

- Ein Behälter oder auch Platzhalter für einen Wert
- Eine Variable hat
  - einen **Namen** (Bezeichner, identifier)
  - einen **Datentyp** (type)
  - einen **Wert** (Inhalt, value)



*meinX*

# Elementare Datentypen / Referenzen

## Elementare Datentypen

- für **ganzzahlige Werte**, z.b.: *13 42 815*
- für **Fließkommawerte**, z.b.: *3.1415 99.5*
- für **Wahrheitswerte**: *true false*
- für **Zeichen**, z.b.: *A B \$ z @*

## Referenzdatentypen

- Verweis (*reference*) auf ein Objekt.

# Elementare Datentypen

Übersicht über die **elementaren Datentypen**:

<code>byte</code>	byte-length integer
<code>short</code>	short integer
<code>int</code>	integer
<code>long</code>	long integer
<code>float</code>	floating point single precision
<code>double</code>	floating point double precision
<code>char</code>	character
<code>boolean</code>	boolean value



# Wertebereiche

Typ	Wertebereich	Länge
byte	-128 bis 127	8 Bit
short	-32.768 bis 32.767	16 Bit
int	-2.147.483.648 bis 2.147.483.647	32 Bit
long	-9.223.372.036.854.775.808 bis 9.223.372.036.854.775.807	64 Bit
float	$-3,4 * 10^{38}$ bis $3,4 * 10^{38}$	32 Bit
double	$-1,7 * 10^{308}$ bis $1,7 * 10^{308}$	64 Bit
char	65.536 Unicode Zeichen	16 Bit
boolean	true/false	2 Bit

# Wertebereich: zeichenorientierter Datentyp

<i>Datentyp</i>	<i>Bits</i>	<i>Wertebereich</i>
char	16-bit Unicode	2 Byte UC: ‘ \u0000 ‘ bis ‘ \uffff ‘



Unicode,  
machen wir  
später...

# Wertebereich: bitorientierter, boolescher Datentyp

<i>Datentyp</i>	<i>Bits</i>	<i>Wertebereich</i>
boolean	8 = 1 Byte	true, false



Eigentlich würde ja 1 Bit reichen...

# Attribute und Datentypen

Es gibt verschiedene Schreibweisen, Attribute zu deklarieren:

**Standard-UML** *<attribute name> : <attribute type>;*



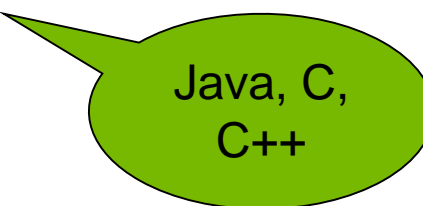
Pascal,  
Delphi

**Corba-IDL** *<attribute type> <attribute identifier>;*

`int myAttribute;`

`float umsatzsteuer;`

`boolean isMarried;`



Java, C,  
C++

# Initialisierung und Wertzuweisung

# Initialisierung

Attributen und Variablen kann bereits bei ihrer **Definition** ein **initialer Wert** zugewiesen (*assign*) werden:

```
int myAttribute = 42;
```

```
float umsatzsteuer = 19.0f;
```

```
boolean isMarried = true;
```



*Datentyp*

*Bezeichner*

*Initialwert*

# Wertzuweisung: linke/rechte Seite

*linke Seite*  
`meinAttribut` = 15;

Attribut auf der linken Seite;  
In dieses Attribut wird hinein  
**geschrieben**, sein Wert verändert sich.

einY = *rechte Seite*  
`meinAttribut`;

Attribut auf der rechten Seite;  
Sein Wert wird nur **gelesen**, **nicht**  
jedoch sein Inhalt verändert.

# Wertzuweisungen

Wurde ein Attribut definiert, kann ihm danach über eine **Zuweisung** beliebig oft ein neuer **Wert** zugewiesen werden:

```
myAttribute = 42;  
umsatzsteuer = 25.0f;  
isMarried = false;  
umsatzsteuer = 25.5f;
```

Der **Datentyp** wird bei einer Wertzuweisung nicht mehr angegeben. Er bleibt unverändert erhalten.



# Übung: Datentypen

## Welchen Datentypen nutzen Sie?

- 1) Postleitzahl von Berlin
- 2) Weltbevölkerung
- 3) Bevölkerung von Deutschland
- 4) Klausurnote
- 5) Konstante PI
- 6) Hausnummer (von kleinen Straßen)
- 7) Buchstabe
- 8) Klausur bestanden/nicht bestanden

# Konstanten

**Konstanten** sind unveränderliche Attribute, deren Wert nach der Initialisierung nicht mehr verändert werden können

*Modifizier*

*für Konstanten Großbuchstaben verwenden*

***final*** *datentyp*

***KONSTANTE = Initialwert;***

***final*** double

**PI = 1,514;**

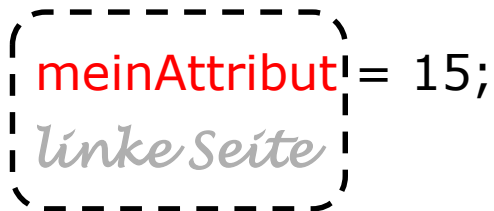
***final*** float

**UMSATZSTEUER = 19.0f;**

***final*** boolean

**JA = true;**

# Wertzuweisung: linke/rechte Seite

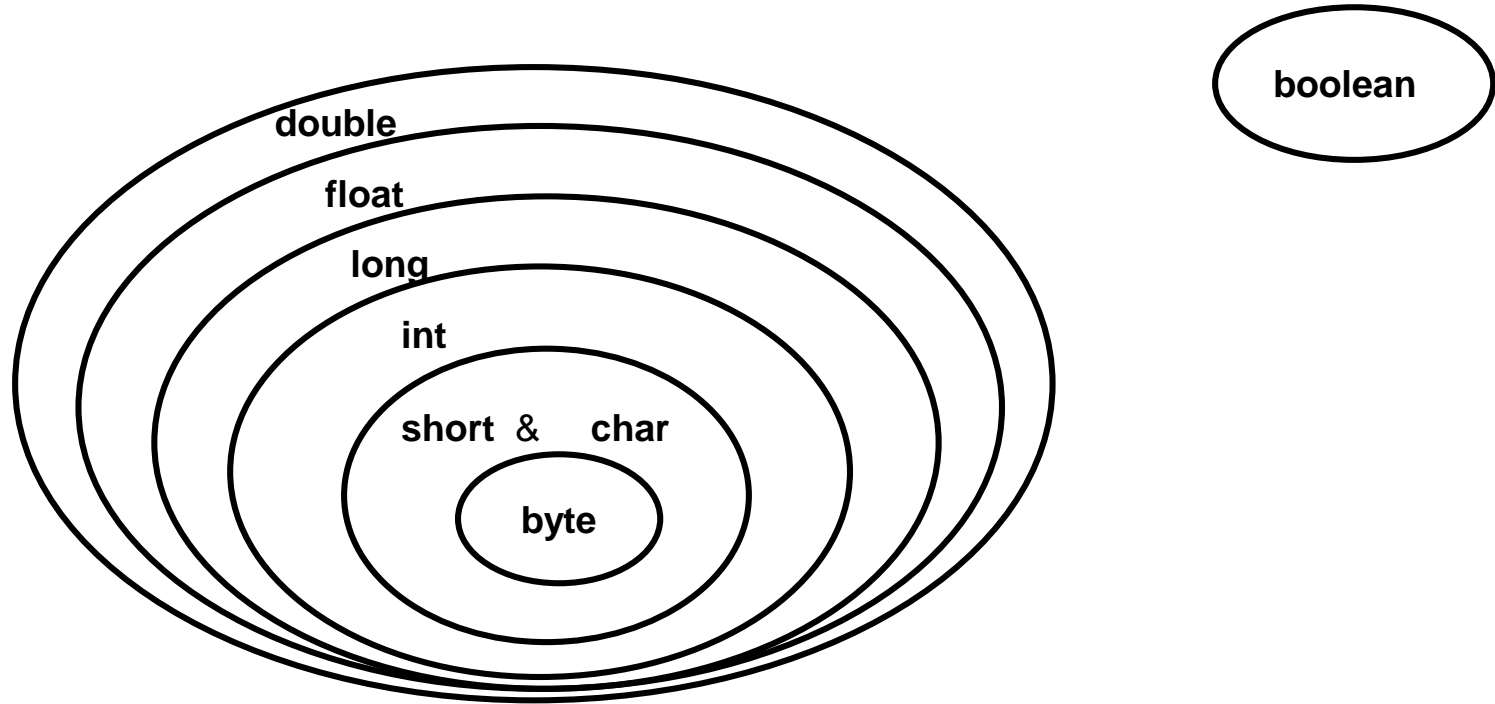
 `meinAttribut` = 15;  
*linke Seite*

Rechte und linke Seite einer Wertzuweisung müssen den gleichen Datentyp haben!

einY =  `meinAttribut`;  
*rechte Seite*

*Zuweisungen sind auch dann erlaubt, wenn der Datentyp der linken Seite den Typ der rechten Seite gemäß **Typ-Teilmengebeziehung** einschließt!*  
Aber was ist mit „einschließen“ gemeint?

# Typ-Teilmengebeziehung



# Wertzuweisung: Typverträglichkeit

```
double myAttribute;  
int otherAttr;  
...
```

Ist die Wertzuweisung  
zulässig?

```
myAttribute = otherAttr;
```

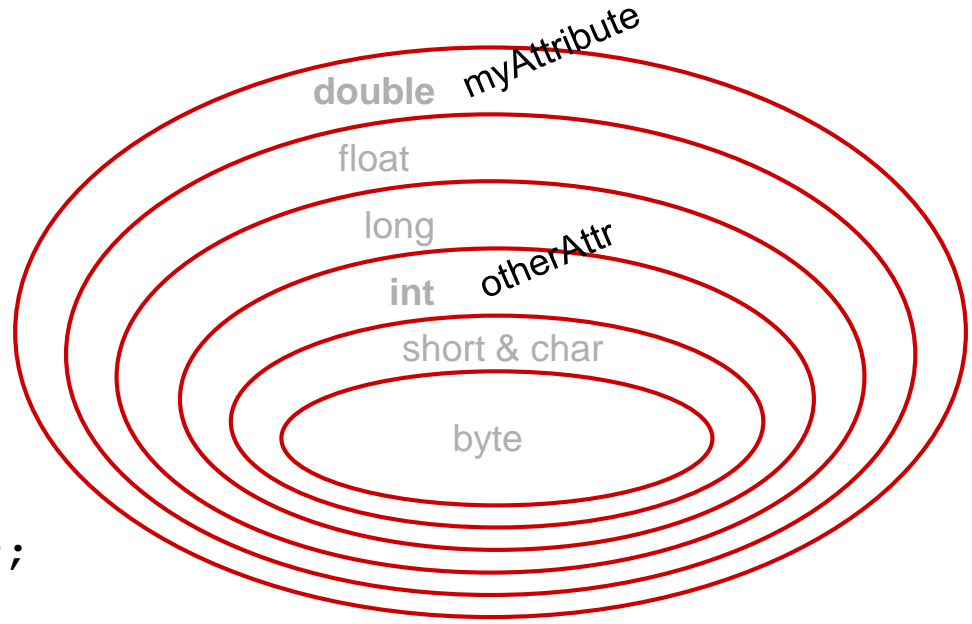
# Wertzuweisung: Typverträglichkeit

```
double myAttribute;  
int otherAttr;  
...
```

Ist die Wertzuweisung  
zulässig?

```
myAttribute = otherAttr;
```

✓ ok



# Wertzuweisung: Typverträglichkeit

```
short myAttribute;  
long otherAttr;  
...
```

Ist die Wertzuweisung  
zulässig?

```
myAttribute = otherAttr;
```

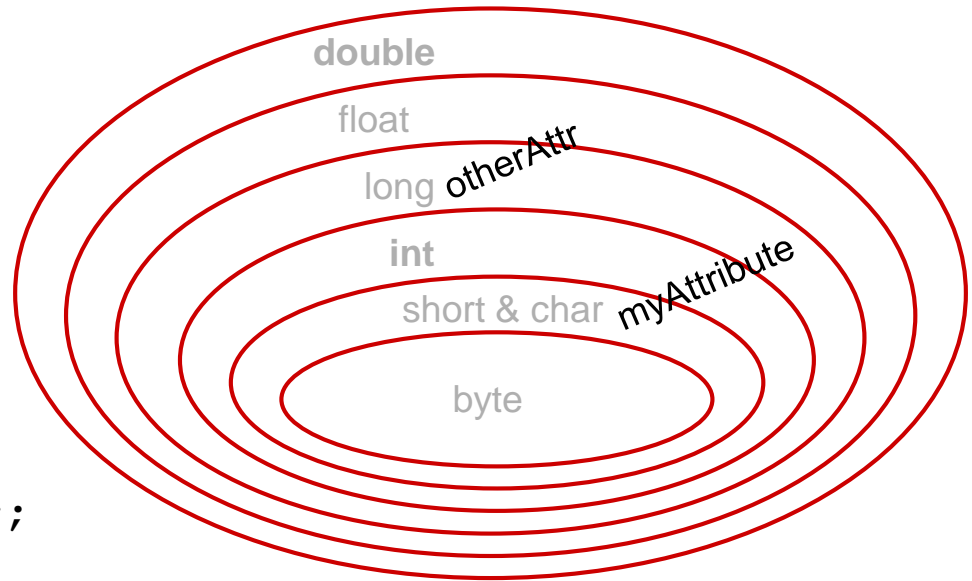
# Wertzuweisung: Typverträglichkeit

```
short myAttribute;  
long otherAttr;  
...
```

Ist die Wertzuweisung  
zulässig?

```
myAttribute = otherAttr;
```

**nicht ok**





# Wertzuweisung: Typverträglichkeit

```
int myAttribute;  
float otherAttr;  
...
```

Ist die Wertzuweisung  
zulässig?

```
myAttribute = otherAttr;
```

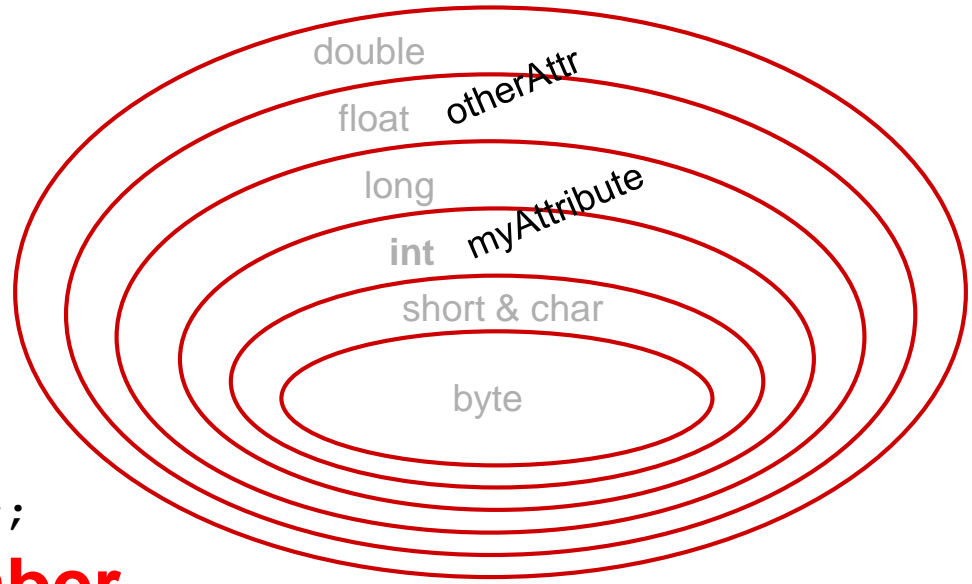
# Wertzuweisung: Typverträglichkeit

```
int myAttribute;  
float otherAttr;  
...
```

Ist die Wertzuweisung  
zulässig?

```
myAttribute = otherAttr;
```

**nicht ok, aber**



# Casting = Datentypen passend gemacht

- **Casting** = explizite Typumwandlung
- Wert des Ausdrucks auf der rechten Seite wird umgewandelt
- Datentyp des Attributs bleibt unverändert
- Inhalt des Attributs auf der linken Seite ist evtl. nicht mit dem des Attributs auf der rechten Seite identisch

```
int myAttribute;
```

```
float otherAttr;
```

```
...
```

```
myAttribute = (int) otherAttr;
```

Cast-Operator

✓ ok

# Übung: Knobelaufgabe

```
int alpha = 225;
```

```
int beta = 7;
```

```
alpha/beta } 32
```

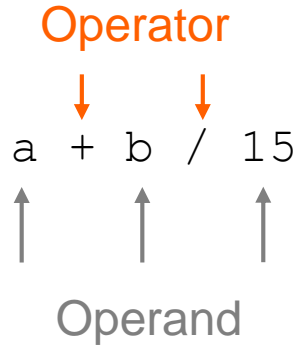
Passen Sie den Code so an, dass nur **ganzzahlige** Variablendefinitionen vorkommen (also *alpha* und *beta* sollen den Datentyp **int** haben) und versuchen Sie, auf das korrekte **reelle** Ergebnis (mit Nachkommastellen) zu kommen.  
Beispiel-Ausgabe: ( 225 / 7 = 32,14)

# Ausdrücke und Operatoren

# Was ist ein Ausdruck?

Unter einem **Ausdruck (expression)** versteht man eine Rechenvorschrift mit der ein **Wert** ermittelt wird. Ein Ausdruck besteht aus **Operanden** und **Operatoren**.

Beispiel:



# Arithmetische und logische Ausdrücke

Wir wollen mit Werten rechnen, bilden also **arithmetische Ausdrücke**.

```
ergebnis = a + b / 15
```

Und wir wollen Werte vergleichen, bilden also **logische Ausdrücke**.

```
( 67 < 110 )
```

Oder beides:

```
67 < (110 * y)
```

# Arithmetische und logische Ausdrücke

## Arithmetischer Ausdruck

```
a + b;  
(a - b ) % c;  
meinX / 13;
```

Arithmetische Ausdrücke liefern als **Ergebnis** keinen booleschen Wert, sondern einen **numerischen Wert**.

## Logischer Ausdruck

```
(a <= b) ;  
(a && b )  
nicht != weiter
```

Logische Ausdrücke liefern als **Ergebnis** immer einen **booleschen Wert** (true oder false).



# Arithmetische Operatoren

## Grundrechenarten

- + plus oder String-Konkatenation
- minus
- \* mal
- / geteilt - Division

Der **Modulo-Operator** für ganze Zahlen

- % mod

# Ganzzahlige Division

Bei der **ganzzahligen Division** werden Nachkommastellen **abgeschnitten**.

```
int a = 8;  
int b = 5;  
int ergebnis = a / b;  // Ergebnis: 1
```

Bei **Division mit Fließkomma-Typ** bleiben **Nachkommastellen** erhalten.

```
float x = 8.0f;  
float y = 5.0f;  
float ergebnis = x / y;  // Ergebnis: 1,6
```

# Der Modulo-Operator

Der Modulo-Operator liefert für Integer-Werte den ganzzahligen Rest einer Division.

```
int a = 8;
```

```
int b = 5;
```

```
int ergebnis = a % b; // Ergebnis: 3, da  $8/5 \Rightarrow 1$  Rest 3
```

Der Operator **mod**  $n$  liefert als ganzzahligen Rest einen Wert der Menge

$\{ 0, 1, 2, \dots n-1 \}$ .

# Exkurs: Erklärung modulo n

## Beispiele:

$$17 \bmod 3 = 2, \text{ da } 17 = 5 * 3 + 2$$

(„3 passt fünfmal in 17 und es bleiben 2 übrig“ – der Rest ist also 2)

$$2 \bmod 3 = 2, \text{ da } 2 = 0 * 3 + 2$$

$$3 \bmod 3 = 0, \text{ da } 3 = 1 * 3 + 0$$

$$-8 \bmod 6 = -2, \text{ da } -8 = -1 * 6 - 2$$

# Logische Ausdrücke

Relationale Operatoren liefern das „**Verhältnis**“, in dem Operanden zueinander stehen. Sie werden verglichen.

**Relationale Operatoren liefern `true` oder `false`**

`<` kleiner, z.B. *1 < 3 ist `true`*

`<=` kleiner oder gleich

`>` größer

`>=` größer oder gleich

`==` gleich (*equals*)

`!=` ungleich

# Logische Operatoren

Für Verknüpfungen von Ausdrücken

- ! Negation
- & Bitweise *Und*-Verknüpfung (*nicht behandelt*)
- | Bitweise *Oder*-Verknüpfung (*nicht behandelt*)
- ^ Entweder-Oder-Auswertung (XOR)
- && Konditional (*bedingt*) auswertendes Und
- || Konditional (*bedingt*) auswertendes Oder

# Unäre arithmetische Operatoren

für **Vorzeichen**

**+x**          positive Zahl

**-x**          negative Zahl

für **Inkrement** (Erhöhung des Wertes um 1)

**++x;**      (Präfix) äquivalent zu      **x = x + 1;**

**x++;**      (Postfix) äquivalent zu      **x = x + 1;**

für **Dekrement** (Verminderung des Wertes um 1)

**--x;**      (Präfix) äquivalent zu      **x = x - 1;**

**x-- ;**      (Postfix) äquivalent zu      **x = x - 1;**

# Zuweisungsoperatoren

**x += y;** entspricht **x = x + y;**

**x -= y;** entspricht **x = x - y;**

**x \*= y;** entspricht **x = x \* y;**

**x /= y;** entspricht **x = x / y;**

**x %= y;**

...

**x <<= y;**    **x >>= y;** verschieben ohne Vorzeichenerhalt

**x >>>= y;** verschieben mit Vorzeichenerhalt



# Tenärer arithmetischer Operator

Tenär = drei Operanden

Fragezeichen-Operator (Konditional-Operator)

**true**

bedingung ? wert1 : wert2

**false**

variable = bedingung ? wert1 : wert2

# Tenärer Operator

Beispiele:

```
maximal = (x >= y) ? x : y;
```

```
divide = (y != 0) ? (x / y) : 0;
```

```
pos = (a > 0) ? (a * a) : 0 ;
```

```
b = (weiter == true ) ? true : false ;
```

# Auswertungsreihenfolge Operatoren 1

Operatoren	Beschreibung	Rang
++, --, +, -, !, ...	Unäre Operatoren	1
(type)	Typumwandlung(Casting)	1
*, /, %	mal, geteilt, modulo	2
+, -	plus, minus, String-Verkettung	3
<<, >>, >>>	Schiebeoperatoren	4
<, <=, >, >=	Vergleichsoperatoren	5
==, !=	Vergleichsop. gleich/ ungleich	6
&	logisches od. bitweises UND	7

# Auswertungsreihenfolge Operatoren 2

Operatoren	Beschreibung	Rang
&	logisches od. bitweises UND	7
^	logisches od. bitweises Entweder-Oder	8
	logisches od. bitweises ODER	9
&&	Kond. auswert. <i>Und</i>	10
	Kond. auswert. <i>Oder</i>	11
?:	Tenä. Bedingungsoperator	12
=, +=, -=, *=, ...	Zuweisungs- und Verbundop.	13

# Literale

# Numerische Literale

**Literale** sind Werte in einem Programm, die man ohne Erklärung des Datentyps verwenden kann.

int	178
long	988L oder 8l
float	34.98f -7.9F
double	8.0 45.9d oder 45.9D 77.8e3

**Beachte:**

L oder l bei long

F oder f bei float

# Alphanumerische Literale

String	<code>"Heinz"</code> <code>"8769-8"</code> <code>"c"</code>
char	<code>' c '</code> <code>' \u0056 '</code>

## **Beachte:**

*Gänsefüßchen* für String  
doppelte Anführungsstriche für  
Zeichenketten

*Hochkomma* für char-Literal  
Einfache Anführungsstriche  
werden für einzelne Zeichen  
verwendet

# Variablen im Programm

```
public class ZahlenBerechnung {  
    public static void main(String[] args) {  
        int a = 5;  
        int b = 2;  
        int ergebnis;  
        ergebnis = (a % b); // Berechnung  
        System.out.print("Ergebnis von " + a + " modulo " + b);  
        System.out.println(" = " + ergebnis);  
    }  
}
```



# Lernziele

- Welche Eigenschaften hat die Programmiersprache Java?
- Wie erstelle ich ein ausführbares Java-Programm?
- Wie sieht die Struktur einer Java-Datei aus?
- Was ist eine Variable?
- Welche Datentypen gibt es und welche Wertebereiche haben sie?
- Wie initialisiere ich Variablen und wie weise ich ihnen Werte zu?
- Wie bilde ich logische und arithmetische Ausdrücke?



**Hochschule für Technik  
und Wirtschaft Berlin**

University of Applied Sciences

[www.htw-berlin.de](http://www.htw-berlin.de)