OO Programmierung in Java

Einführung

WS 2012/2013

Prof. Dr. Margarita Esponda

Freie Universität Berlin

Homepage

http://www.esponda.de/WS_12_13/JBK

Vorlesungsfolien
Literaturliste
Übungen
Zusätzliches Material

esponda@inf.fu-berlin.de

Sprechstunde: Fr. 8-10 und 12:30 - 13:30 Uhr

Raum 161

Java ist plattformunabhängig

Quellprogramm

```
public class …
   public read (a)[….
      b = readNum();
      if (a<b) then
          a = a*a;
      else
          a = a+b;
        …
```

Java-Compiler

javac

Bytecode

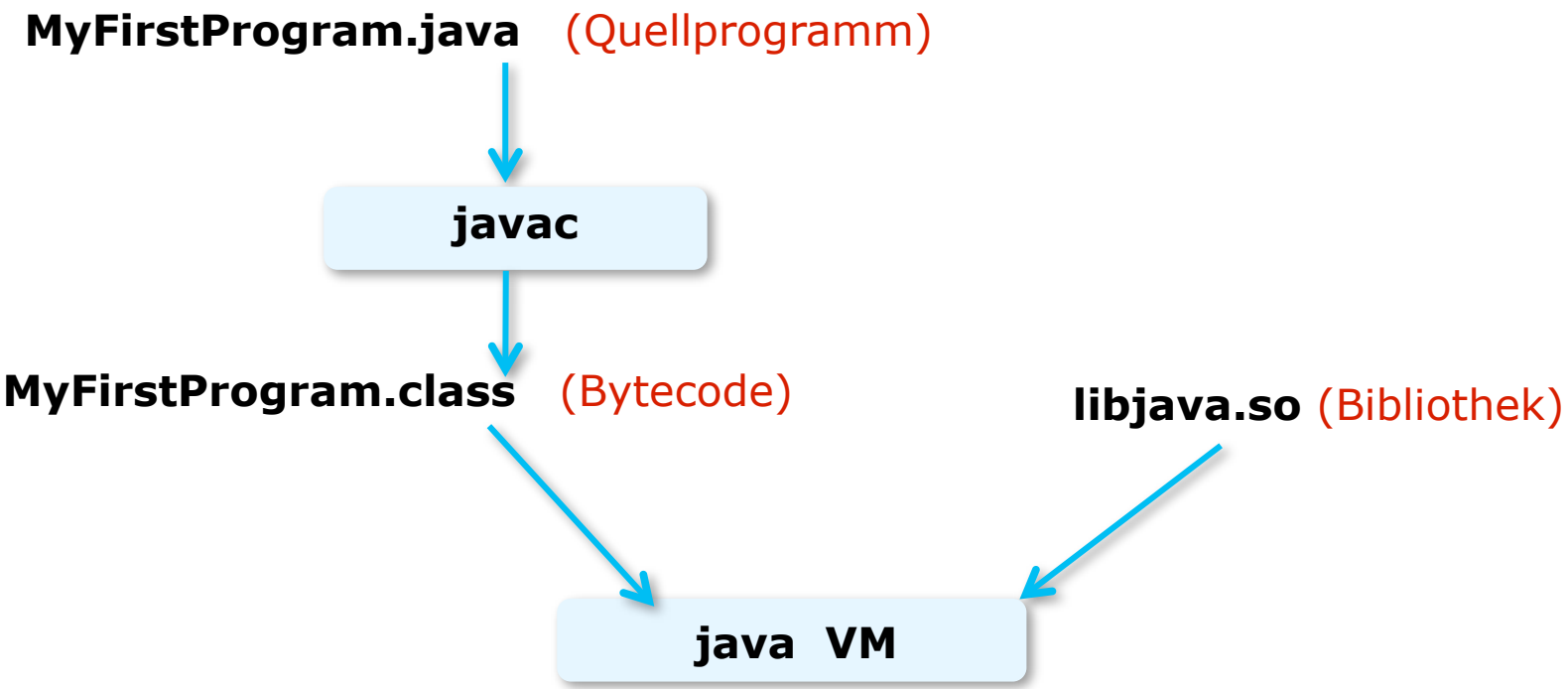
```
iLOAD   #1  C
iLOAD   #2  B
iMULT   #1  #2  #3
iLOAD   #4  A
iADD    #3  #4  #1
iSTORE  #1  C
iLOAD   #4  A
iADD    #3  #4  #1
```

Interpreter

JVM JVM JVM JVM

Der Interpreter (JVM) wird direkt von der Hardware ausgeführt.

Übersetzen/Ausführen




```
Übersetzen:     javac  MyFirstProgram.java
Ausführen:      java   MyFirstProgram
```

Java-Programme

- Java-Programme bestehen aus einer oder mehreren *Klassen*

- Eine Klasse ist in der Regel in einer eigenen, gleichnamigen Datei mit der Endung **.java** definiert, z. B.

MyFirstProgram.java

- Die Programmausführung beginnt immer mit der Methode **main** einer der Klassen.

Java-Programm

Beispiel:

```
/* Ein einfaches aber vollständiges  Java-Programm  */
public class  MyFirstProgram {

        public static void main ( String[] args ) {
                System.out.println( "Es läuft ! " );
        }

} // end of class MyFirstProgram
```

Jedes Java-Programm muss mindestens eine Methode namens **main** haben.
Hier fängt die Programmausführung an.

Kommentare in Java

Zeilenend-Kommentare

```
// von hier aus bis zum Ende der Zeile wird dieser Text ignoriert
```

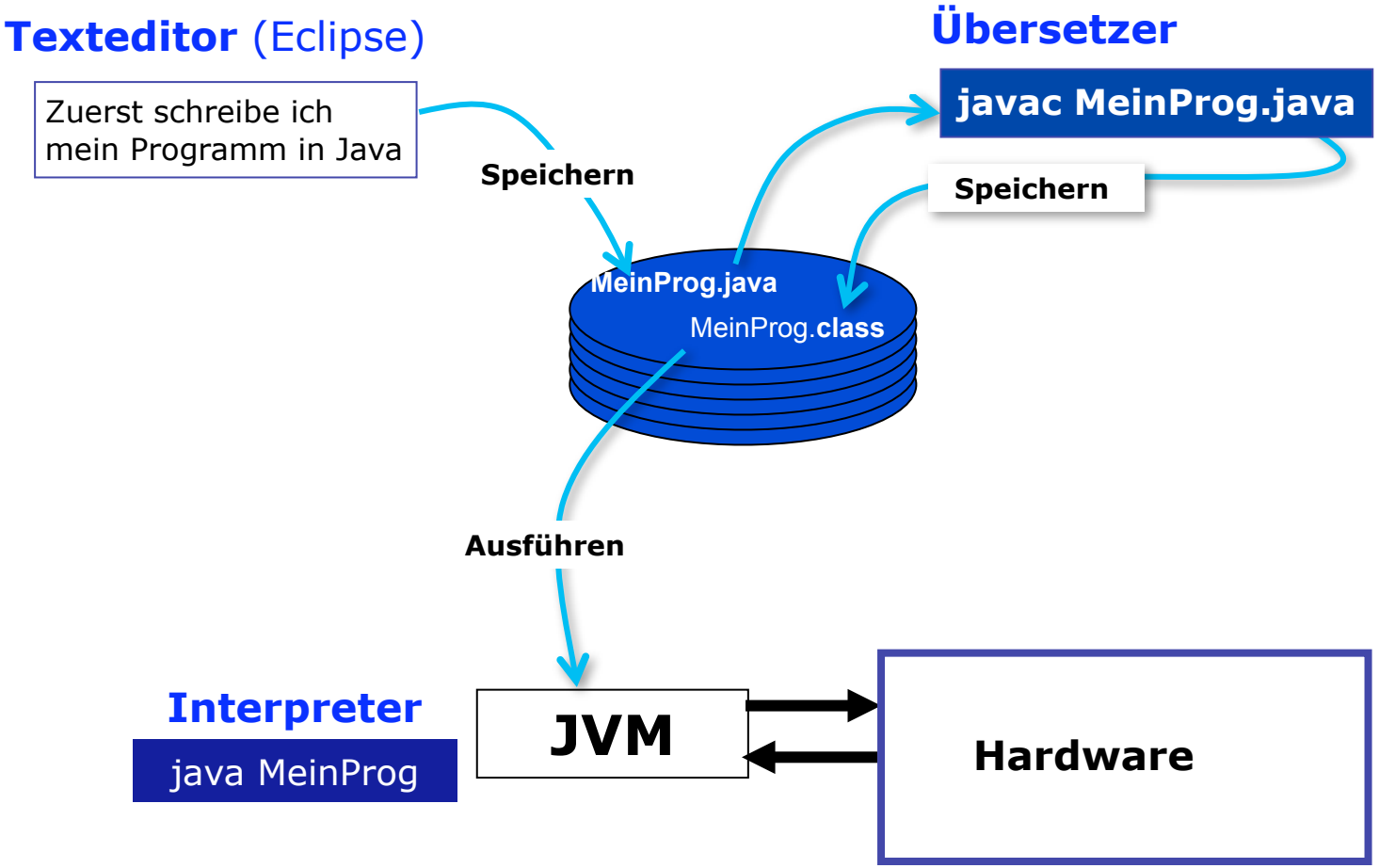
Block-Kommentare

```
/*  alle  diese  Zeilen  hier  werden  von
      dem  javac  völlig  ignoriert ........
*/
```

Javadoc-Kommentare

```
/** dieser Text wird von dem javadoc-Programm verwendet,
    um automatische Dokumentation in html-Format zu
    erzeugen
*/
```

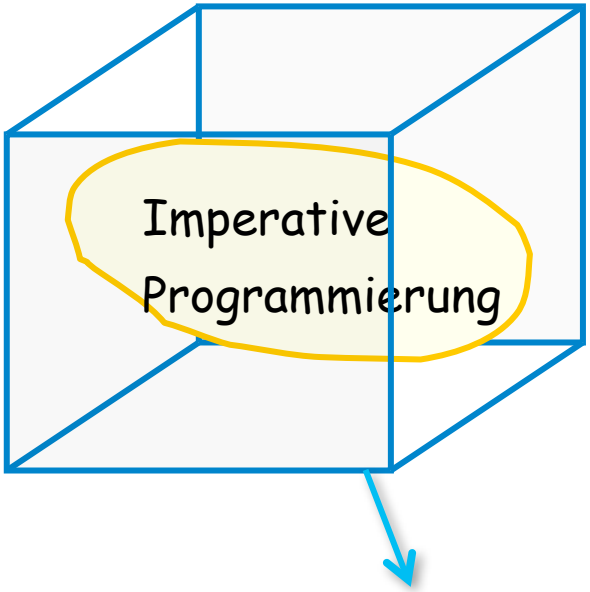
Vom Quellprogramm bis zur Ausführung



JDK (Java Development Kit)

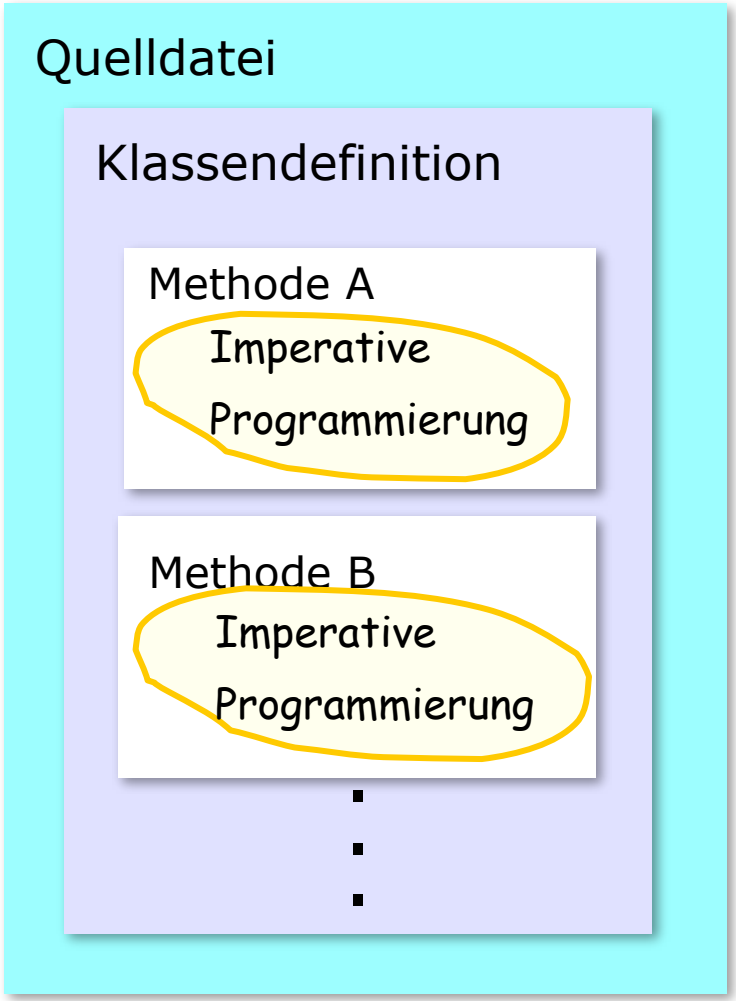
javac	Java-Compiler
java	Java-Interpreter
appletviewer	Interpreter für "Applets"
javadoc	Dokumentationsgenerator
jar	Archivierungsprogramm
jdb	Java-Debugger

Imperative Grundbestandteile von Java



Objektorientierte Verpackung

Der imperative Bestandteil eines Java-Programms befindet sich innerhalb der Methoden.



Imperative Grundbestandteile von Java

Wie programmieren wir den Inhalt einer Methode?

1. Primitive Datentypen in Java
2. Deklaration von Variablen (OOP)
3. Ausdrücke in Java
4. Die vielen Operatoren von Java
5. Einfache Anweisungen
6. Anweisungen zur Ablaufsteuerung

Was ist eine Variable?

Variablen sind Stellen im Speicher, in denen Werte abgelegt werden können.

Variablen haben:

- *Typ*
- *Namen*
- *Wert*

  → Im Programm sichtbar

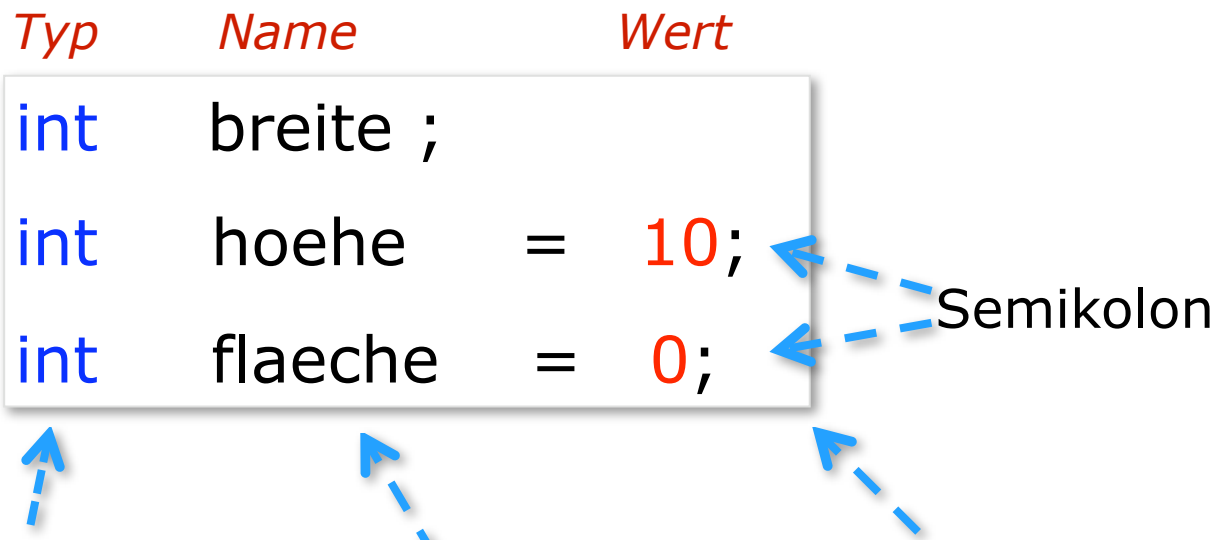
- *Adresse*

  → Stelle im Speicher, wo die Variable sich während der Ausführung des Programms befindet

Variablen müssen vor der erstmaligen Benutzung deklariert werden.

Die Änderung des Wertes einer Variablen geschieht durch *Wertzuweisung* (=).

Variablendeklarationen innerhalb von Methoden



In Java sagt der Datentyp vor allem, wie viel Speicherplatz benötigt wird, um die entsprechende Variable speichern zu können.

Symbolischer Name einer Speicheradresse

Der Wert 0 ist der Inhalt der Speicheradresse unmittelbar nach dieser Deklaration. Das kann sich aber später ändern.

Programmbeispiel mit Variablendeklarationen

```
public class SomeVarDeclarations {

    public static void main(String[] args) {
        // Variablen
            double y = 10.5;
            int a, b, sum, mult;
            String message;
        // Wertzuweisungen
            a = 10; b = 30;
            sum = a+b;
            mult = a*b;
            message = "Berechnungen:";
        // Ausgabe im Eingabeaufforderung-Fenster
            System.out.println( message );
            System.out.println( mult );
            System.out.println( " Die Summe ist gleich " + sum );
    }  // end of main

} // end of class
```

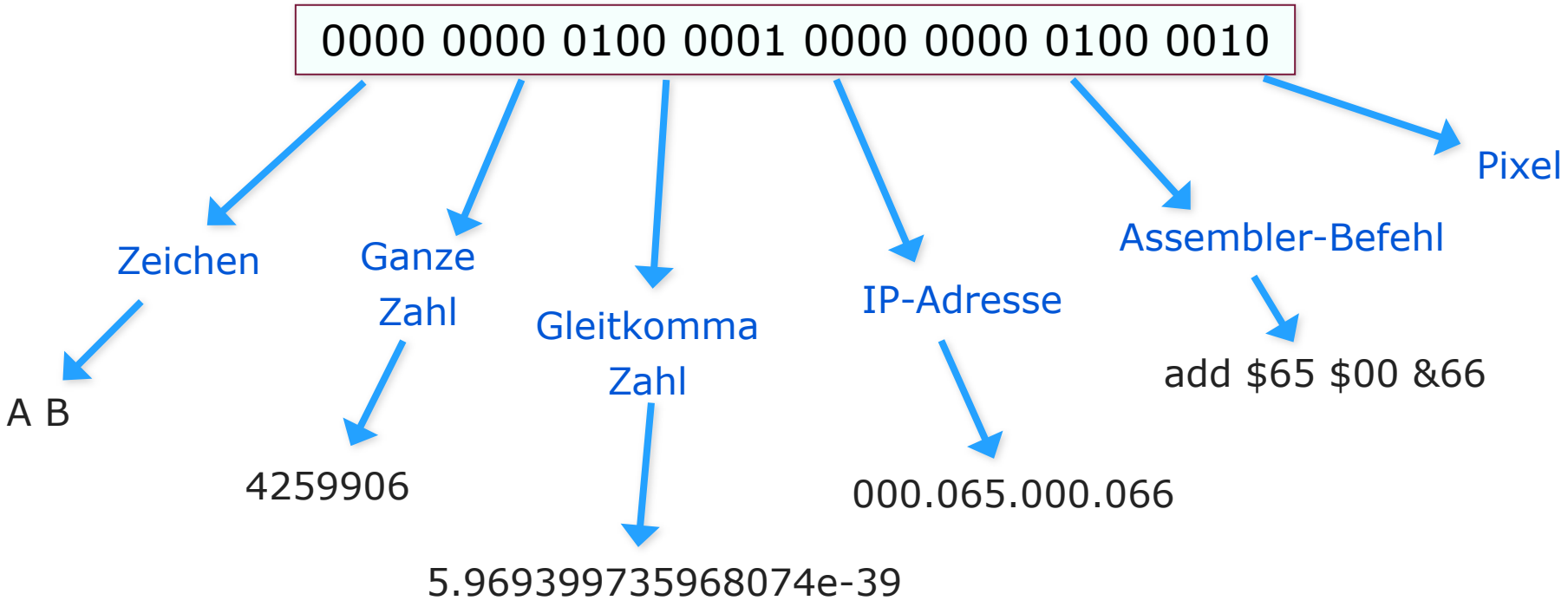
```
public  static void main ( String[] args ) {
    double y = 10.5;
    int a, b, sum, mult;
    a = 10;
    b = 30;
    sum = a+b;
    mult = a*b;
    System.out.println( "Es läuft! " );
    System.out.println( y );
    System.out.println( "a = " +a);
    System.out.println( sum * mult) ;
}
```

Speicher

y	10.5
a	10
b	30
sum	40
mult	300

```
Es läuft!
10.5
a = 10
12000
```

Eine Speicheradresse mit 32 Bits kann sehr unterschiedlich interpretiert werden.



Binärsystem → Zehnersystem

(mit Zahlen ohne Vorzeichen)

Das Binärsystem benötigt nur 2 verschiedene Symbole, um alle Zahlen zu kodieren. (**0** oder **1**)

Binärdarstellung **Dezimaldarstellung**

$$\underset{2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0}{\boxed{1\,0\,1\,0\,1\,0_{\mathbf{2}}}} = 1\cdot 2^5 + 0\cdot 2^4 + 1\cdot 2^3 + 0\cdot 2^2 + 1\cdot 2^1 + 0\cdot 2^0$$

$$= 32 + 8 + 2$$

$$= \boxed{42_{\mathbf{10}}}$$

Freie Universität Berlin

Zahlen in einem Stellenwertsystem

$$z_n z_{n-1} z_{n-1} \cdots z_2 z_1 z_0 = \sum_{i=0}^{n} z_i \cdot b^i$$

In dem Binärsystem sind die Ziffern $z_i \in \{0;1\}$ *und* $b = 2$

Im Dezimalsystem sind die Ziffern $z_i \in \{0;1;2;3;4;5;6;7;8;9\}$ *und* $b = 10$

Freie Universität Berlin

Zehnersystem → Binärsystem

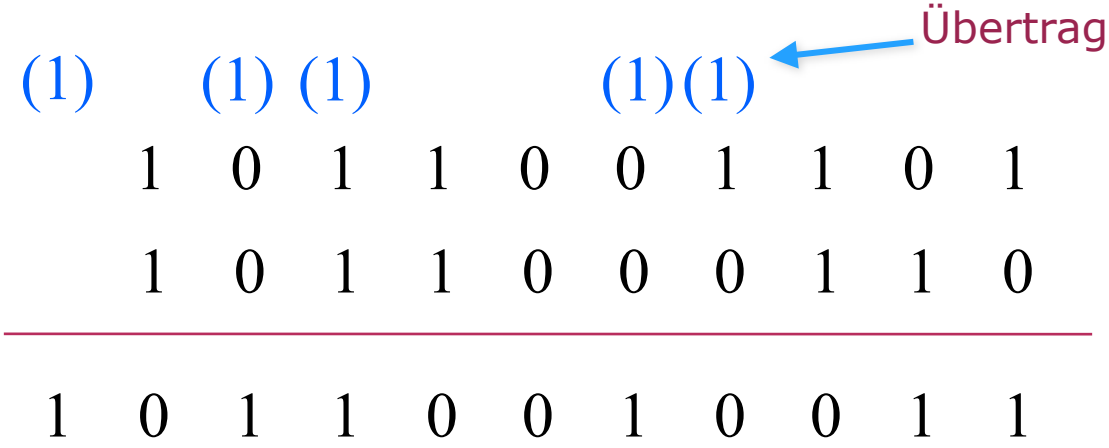
(Zahlen ohne Vorzeichen)

$42_{10} = 101010_{2}$

	Rest
42	0
21	1
10	0
5	1
2	0
1	1
0	

Freie Universität Berlin

Addition mit Binärzahlen



Freie Universität Berlin

Multiplikation mit Binärzahlen

	1	0	1	1
x		1	0	1
	1	0	1	1
0	0	0	0	
1	0	1	1	
1	1	0	1	1	1

11
x 5
55

= 32 + 16 + 4 + 2 + 1 = **55**

Freie Universität Berlin

Positive und Negative Zahlen

Mit einem **32**-Bit-Wort können $\mathbf{2^{32}}$ verschiedene Zahlen dargestellt werden.

Das erlaubt einen Wertebereich von **0** bis $\mathbf{2^{32}-1}$, wenn wir nur positive Zahlen damit darstellen wollen.

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = 0_{10}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_2 = 1_{10}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_2 = 2_{10}$
.....
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_2 = 4\ 294\ 967\ 294_{10}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2 = \mathbf{4\ 294\ 967\ 295_{10}}$

Wie kann dieser Wertebereich aufgeteilt werden, so dass positive und negative Zahlen dargestellt werden können?

Freie Universität Berlin

Positive und Negative Zahlen

1. Lösung **Balancierte Teilung:**

Vorzeichen **0 = positive** **1 = negative**

0000 0000 0000 0000 0000 0000 0000 0000

Probleme:

0000 0000 0000 0000 0000 0000 0000 0000 = **+0**

1000 0000 0000 0000 0000 0000 0000 0000 = **-0**

Zwei verschiedene Kodierungen für Null!

Sehr schlecht für eine Hardware-Implementierung!

Freie Universität Berlin

Positive und Negative Zahlen

2. Lösung **Unbalancierte Teilung:**

0000 0000 0000 0000 0000 0000 0000 0000 = 0
0000 0000 0000 0000 0000 0000 0000 0001 = 1
0000 0000 0000 0000 0000 0000 0000 0010 = 2
. . . . .
0111 1111 1111 1111 1111 1111 1111 1110 = 2 147 483 646
0111 1111 1111 1111 1111 1111 1111 1111 = 2 147 483 647
1000 0000 0000 0000 0000 0000 0000 0000 = **-2 147 483 648**
1000 0000 0000 0000 0000 0000 0000 0001 = -2 147 483 647
1000 0000 0000 0000 0000 0000 0000 0010 = -2 147 483 646
. . . . .
1111 1111 1111 1111 1111 1111 1111 1101 = -3
1111 1111 1111 1111 1111 1111 1111 1110 = -2
1111 1111 1111 1111 1111 1111 1111 1111 = -1

Es gibt eine negative Zahl mehr!

Freie Universität Berlin

Zweierkomplementdarstellung

Übertrag

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

0000 0000 0000 0000 0000 0000 0000 0001 = 1

1111 1111 1111 1111 1111 1111 1111 1111 = -1

1 0000 0000 0000 0000 0000 0000 0000 0000 = 0

32 Bits

Arithmetischer Überlauf (Overflow)

Positive und negative Zahlen können einfach addiert werden und das Ergebnis ist richtig!

Freie Universität Berlin

Zweierkomplementdarstellung

Wie kann ich aus n die Zahl –n kodieren?

Beispiel: $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_2 = \mathbf{7_{10}}$

Schritt 1: Alle Bits werden umgekippt

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1000_2$

Schritt 2: Eine 1 wird addiert

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1001_2 = \mathbf{-7_{10}}$

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_2 = 7_{10}$

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1001_2 = -7_{10}$

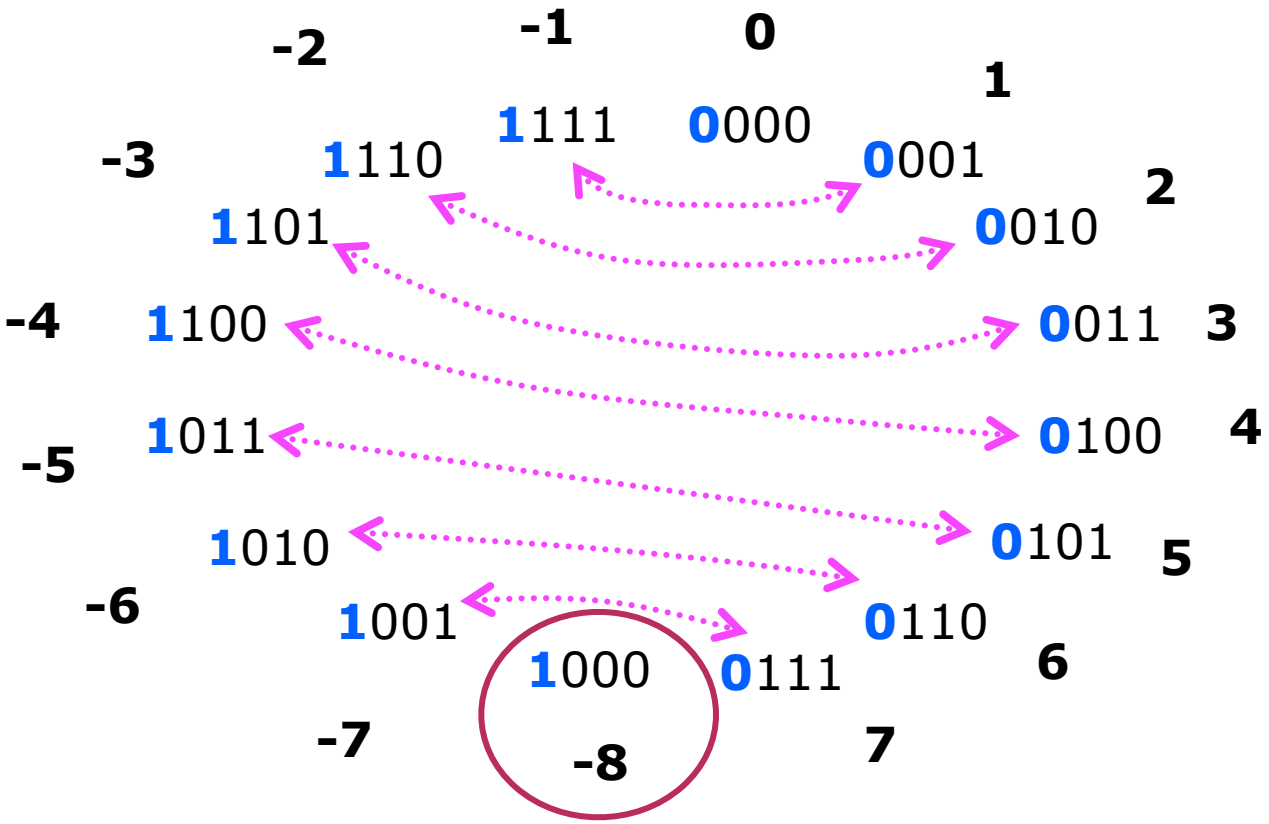
1 0000 0000 0000 0000 0000 0000 0000 0000

overflow

Freie Universität Berlin

Zweierkomplementdarstellung

mit nur 4 Bits



Freie Universität Berlin

Zweierkomplementdarstellung

-1_{10}		1_{10}		
1111	+	0001	→	0000
11111111	+	00000001	→	00000000
1111111111111111	+	0000000000000001	→	0000000000000000

```
  01111111111111111111111111111111   → 2 147 483 647
+ 00000000000000000000000000000001
----------------------------------
  10000000000000000000000000000000   → -2 147 483 648
```

Freie Universität Berlin

Subtraktion

8 Bits

77_{10} ⋯▸ 0 1 0 0 1 1 0 1

70_{10} ⋯▸ 0 1 0 0 0 1 1 0

Zweierkomplement

(1) (1) (1) (1)

77_{10} ⋯▸ 0 1 0 0 1 1 0 1

\+

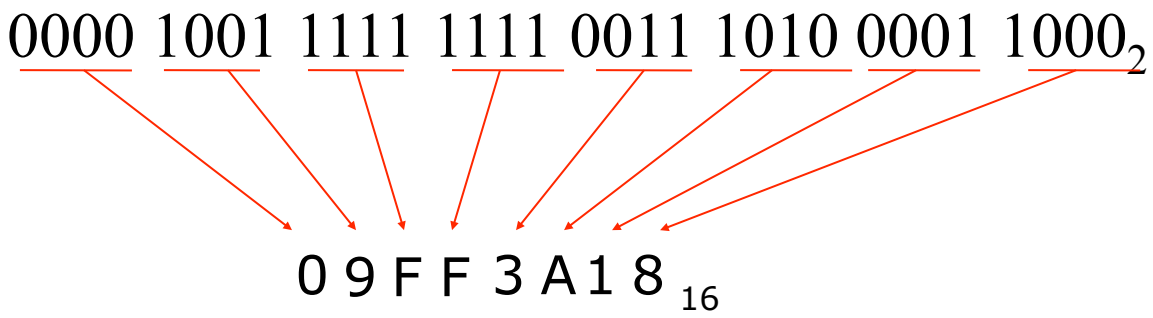
$-\ 70_{10}$ ⋯▸ 1 0 1 1 1 0 1 0

(1) 0 0 0 0 0 1 1 1 ⋯▸ 7_{10}

Freie Universität Berlin

Hexadezimal-Darstellung

0000	**0**
0001	**1**
0010	**2**
0011	**3**
0100	**4**
0101	**5**
0110	**6**
0111	**7**
1000	**8**
1001	**9**
1010	**A**
1011	**B**
1100	**C**
1101	**D**
1110	**E**
1111	**F**



Beispiel: Farben

Hexadezimal: **FF 00 AD** HTML

Dezimal: **255, 000, 173** Java

Freie Universität Berlin

Hexadezimal-Kodierung

Basis = 16

Ziffern = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F }

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Hexadezimal → Dezimal

Beispiel:

$$1\,A\,4\,F\,B_{16} = 1\cdot16^4 + A\cdot16^3 + 4\cdot16^2 + F\cdot16^1 + B$$

16^4 16^3 16^2 16^1 16^0

$$= 107771_{10}$$

Dezimal → Hexadezimal

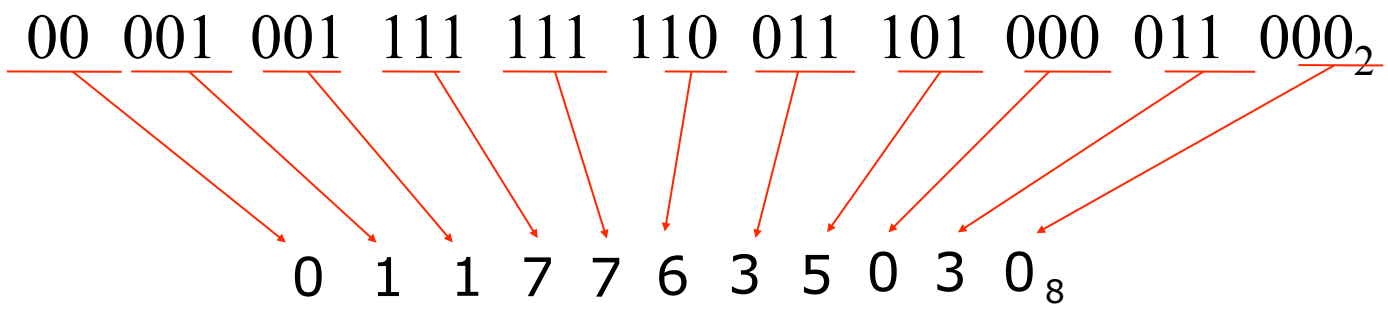
	Rest
107771	11
6735	15
420	4
26	10
1	1
0	

$1\,A\,4\,F\,B_{16}$

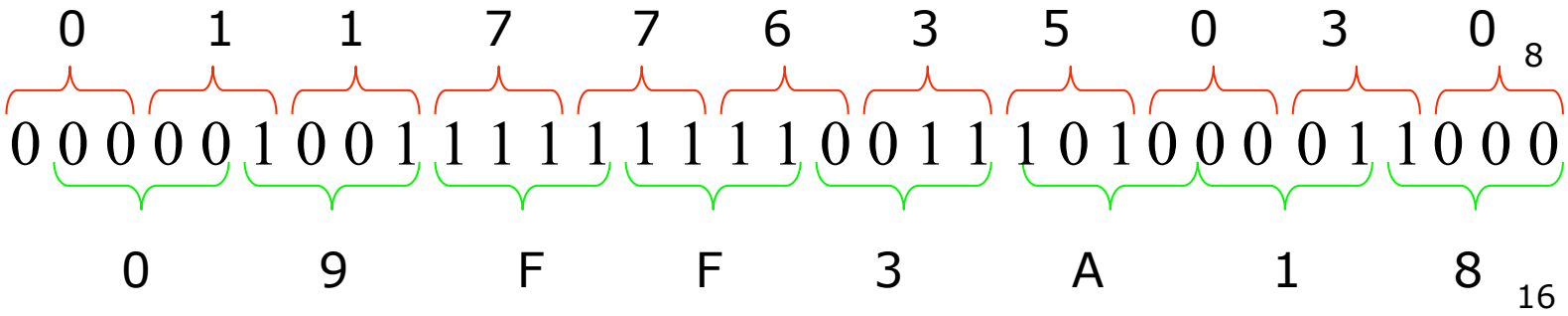
Freie Universität Berlin

Oktaldarstellung

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7



Oktal → **Hexadezimal**



Primitive Datentypen in Java

Java hat **8** elementare Datentypen und legt für jeden primitiven Datentyp eine **feste Speichergröße** und damit einen **festen Zahlenbereich** fest.

Typ	Bits	Zahlenbereich	Standardwert
byte	8	-128 . . . 127	0
short	16	-32.768 . . 32.767	0
int	32	-2^{31} . . . $2^{31}-1$	0
long	64	-2^{63} . . . $2^{63}-1$	0
float	32	ca. $3.402*10^{38}$ . . $1.402*10^{-45}$	0.0
double	64	ca. $1.798*10^{308}$ . . $4.94*10^{-324}$	0.0
boolean	1	true, false	false
char	16	Unicode	\u0000

Literale (explizite Wertangabe)

Konstanten sind explizit angegebene Objekte im Programm.

Sie besitzen einen Typ, der sich aus der Schreibweise der Konstanten ergibt.

Ganzzahlige Konstanten: **345** **0** **10**

Gleitpunktkonstanten: **3.45** **0.0** **0.5e+2** **0.5E-3**

Zeichenkonstanten:
- Sichtbar: **'A'** **'a'** **'1'** **'+'**
- Unsichtbar: **'\n'** **'\a'** **'\b'** **'\t'**

Zeichenkettenkonstanten: **"Zeichenkette"**

Namen von Variablen

beginnen immer mit

- einem Buchstaben — **area a2345**
- oder '_' — **_green _100**
- keine Sonderzeichen dazwischen — **wie +, =, & usw.**
- kein reserviertes Java-Wort — **int double main**

Konventionen

Variablennamen werden klein geschrieben — **umfang**

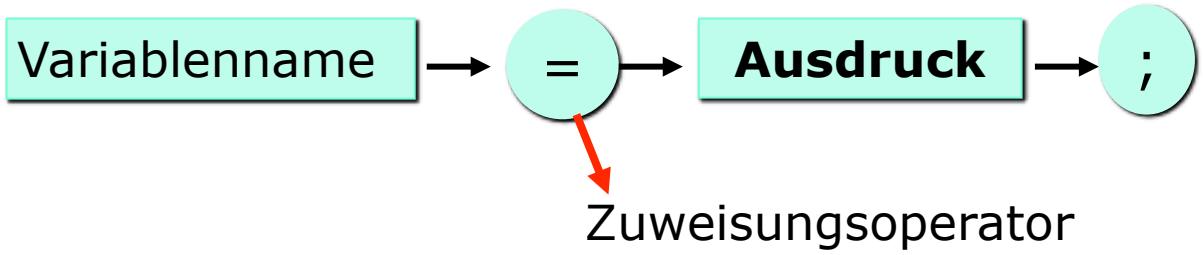
mit großen Buchstaben dazwischen — **penColor**

oder "underlines" dazwischen — **pen_color**

Einfache Anweisungen in Java

Die einfachsten Befehle in jeder Programmiersprache sind die Zuweisungen.

Zuweisung in Java



Beispiele:

```
b = 3 * b ;
c = a / b ;
```

Semikolon

Anweisungen zur Ablaufsteuerung

Einzelne Anweisungen (*statements*) werden mit einem Semikolon abgeschlossen:

```
a = b * c;
```

Anweisungen werden durch geschweifte Klammern zu *Blöcken* zusammengefasst:

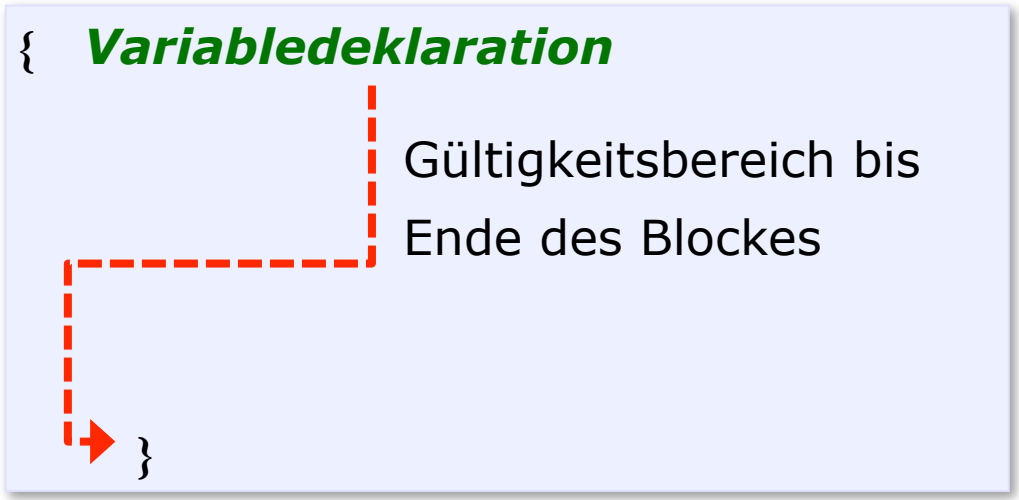
```
{
  a = b * r;
  x = a + z/a;
}
```

Die Abarbeitungsreihenfolge der Anweisungen verläuft normalerweise von oben nach unten und von links nach rechts.

Variablen in Java

Variablen können überall deklariert werden, aber nicht außerhalb von Klassendefinitionen.

Ihr Gültigkeitsbereich erstreckt sich von der Stelle ihrer Deklaration bis zum Ende des *Blocks*, in dem sie deklariert wurden.



Arithmetische Operatoren

unär

Operator	Zeichen	Rtg.	Beispiel	Priorität
negatives Vorzeichen	-	←	-x	13

binär

Operator	Zeichen	Rtg.	Beispiel	Priorität
Multiplikation	*	→	a * b	12
Division	/	→	3 / 5	12
Rest	%	→	6 % 5	12
Addition	+	→	10 + 3	11
Subtraktion	-	→	10 - 3	11

Ausdrücke		Wert	Typ
7 / 2	→	3	int
1 / 2	→	0	int
1 / 2.0	→	0.5	double
4 % 2	→	0	int
7 % 2	→	1	int
1 % 2	→	1	int

Einfache Befehle in Java

```
  …
// Deklaration von Variablen
   int  a, b, c ;
// Zuweisungen
   a  =  7 % 2 ;
   b  =  a * a ;
   c  =  a / b + 1 ;
   …
```

Inkrement- und Dekrement-Operatoren

Operator	Benutzung	Beschreibung	Priorität
++	++ op	Inkrementiert op um 1	13
++	op ++	Inkrementiert op um 1	13
--	-- op	dekrememtiert op um 1	13
--	op --	dekrememtiert op um 1	13

Beispiele:

```
int i = 10, j = 0;
j = ++i;
j = i++;
```

j:	11	i:	11
j:	11	i:	12

```
int a = 5;
int b = 4;
int c = 0;
int d = 0;
c = --a + b++;
c = c + ++a;
d = c++ - ++a + b--;
```

a: 4	b: 5	c: 8
a: 5	b: 5	c: 13

Was ist der Inhalt der Variablen **a?** **b?** **c?** und **d?**

a?	b?	c?	d?
6	4	14	12

Vergleichsoperatoren

binär

Kleiner	**<**	→	op1 < op2	9
Größer	**>**	→	op1 > op2	9
Kleiner oder gleich	**<=**	→	op1 <= op2	9
Größer oder gleich	**>=**	→	op1 >= op2	9
Gleichheit	**==**	→	op1 == op2	8
Ungleichheit	**!=**	→	op1 != op2	8

```
public class SimpleStatements {

    public static void main( String args[]) {

        boolean w1, w2, w3;
        int a = 3;
        int b = 3;

        w1 = a<b;
        System.out.println("w1 = " + w1 );
        w2 = a<=b;
        System.out.println("w2 = " + w2);
        w3 = a!=b;
        System.out.println("w3 = " + w3);
    }
}
```

```
w1 = false
w2 = true
w3 = false
```

Logische Operatoren

unär

Operator	Zeichen	Rtg.	Beispiel	Priorität
logische Negation	**!**	←	!a	13

binär

UND	**&&**	→	a && b	4
ODER	**||**	→	a || b	3

ternär

bedingter Ausdruck	B **?** A_1 **:** A_2	←	!true ? 4 : 9	2

Beispiele:

```
boolean  w1 =  false;
boolean w2, w3;
w2  =   !w1;
w1  =   6%3 ==  0;
w2  =   w1 || w2;
w1  =  (6<7) && (7<6);
w3  =   w1 != w2;
```

w2	true
w1	true
w2	true
w1	false
w3	true

Die Vergleichsoperatoren können nur mit den elementaren Datentypen von **Java** operieren und produzieren immer einen Wahrheitswert.

Die logischen Operatoren operieren nur mit Wahrheitswerten.

Beispiele:

```
6 < 7   ==   3 > 0
```

```
jahr%4 == 0 && (jahr%100 != 0 || jahr%400 == 0)
```

Schaltjahr

Bedingung

```
a<0 ? -a : a
```

Ausdruck$_1$ **Ausdruck$_2$**

Logische Bitoperatoren

binär

bitweise UND	&	→	a & b	7
bitweise ex-ODER	^	→	a ^ b	6
bitweise ODER	|	→	a | b	5

ex-ODER

Aussage$_1$ ex-ODER ***Aussage$_2$*** ist wahr, falls nur eine der beiden Aussagen wahr ist, sonst falsch.

bitweise ex-ODER

```
0 ^ 0  →  0
0 ^ 1  →  1
1 ^ 0  →  1
1 ^ 1  →  0
```

Beispiel:

6 & 17 →

6 = **00000110**
17 = **00010001**
00000000 → **0**

6 | 7 →

6 = **00000110**
7 = **00000111**
00000111 → **7**

-6 ^ 7 →

-6 = **11111010**
7 = **00000111**
11111101 → **-3**

-6 ^ -6 →

-6 = **11111010**
-6 = **11111010**
00000000 → **0**

Bit-Operatoren

unär

Operator	Zeichen	Rtg.	Beispiel	Priorität
Bitkomplement	~	←	~ a	13

binär

Linksschieben	<<	⟶	op1 << op2	10
Rechtsschieben	>>	⟶	op1 >> op2	10

Bit-Operatoren

Bitkomplement

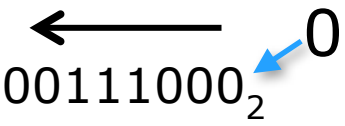
~14

~ 00001110_2 11110001_2 -15

Linksschieben

14<<2

00001110_2<<2

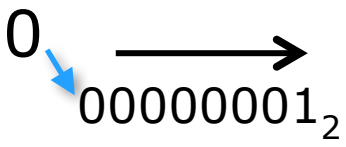


56

Rechtsschieben

14>>3

00001110_2>>3

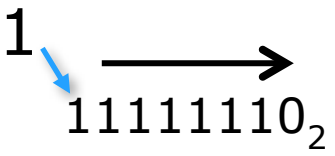


1

Rechtsschieben

-14>>3

11110010_2>>3



-2

Beispiele:

15 << 3 → 120 equiv. 15 * 2 * 2 * 2

~ 10 → -11

15 >> 3 → 1 equiv. 15 / 2 / 2 / 2

Wichtig !

Die Ausführungsreihenfolge aller Operatoren hängt von der Bindungskraft (*Präzedenz*) und Richtung (*Assoziativität*) der Operatoren ab und kann durch runde Klammern ( ) verändert werden.

Zuweisungsoperatoren

Zuweisungen	Benutzung	gleichbedeutend mit	Priorität
=	op1 = exp1	←	1
+=	op1 += op2	op1 = op1 + op2	1
-=	op1 -= op2	op1 = op1 - op2	1
*=	op1 *= op2	op1 = op1 * op2	1
/=	op1 /= op2	op1 = op1 / op2	1
%=	op1 %= op2	op1 = op1 % op2	1
&=	op1 &= op2	op1 = op1 & op2	1
|=	op1 |= op2	op1 = op1 | op2	1
^=	op1 ^ = op2	op1 = op1 ^ op2	1
<<=	op1 <<= op2	op1 = op1 << op2	1
>>=	op1 >>= op2	op1 = op1 >> op2	1