

# Praktische Informatik 1

## Operatoren und Ausdrücke

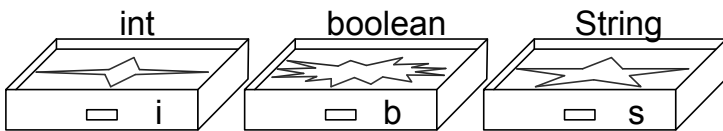
Thomas Röfer

- Operatoren
- Schreibweisen von Operatoren
- Arten von Operatoren
- Vorrang von Operatoren
- Typanalyse von Ausdrücken
- Musterlösung, Übungsblatt



# Rückblick „Variablen, Konstanten und Referenzen“

## Variablen/Konstanten



Name, Typ, Wert, Ort, Lebenszeit

## Arten von Variablen

Lokale Variablen
Funktionsparameter
Objektattribute
Klassenattribute

## Sichtbarkeit

```
class HideAndSeek {
    static int target;
    static void fun() {
        if (target > 10) {
            System.out.println(target);
            int target = 17;
            System.out.println(target);
        }
        System.out.println(target);
    }
}
```

## Stack

	M	
boolean	b	true
int	i	42
String	s	
BinTree	t	
	M	

## Heap

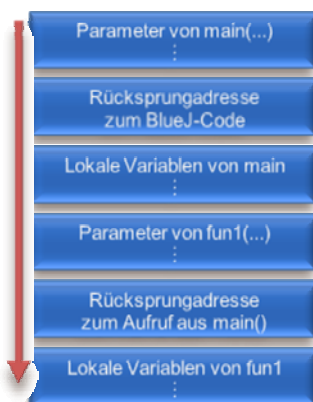
"Hallo"		
BinTree	left	null
int	val	0
BinTree	right	null
boolean	isEmpty	true
BinTree	left	
int	val	17
BinTree	right	null
boolean	isEmpty	false
	M	

## Call by Reference/Value

	M	
int	a	17
int	b	42
	M	

	M	
int	value	17
int	value	42
	M	

	M	
Value	a	
Value	b	
	M	



swap1

swap2

swap3

# Operatoren

- Operatoren sind Funktionen, die anders notiert werden
- In Java gibt es nur vordefinierte Operatoren mit festgelegter Bedeutung
  - Man kann weder neue Operatoren definieren, noch welche *überladen*
- Beispiel
  - Das  $+$  in  $a + b$  entspricht einer Funktion *int javaPlusInt(int, int)*, die zwei Ganzzahlen addiert und das Ergebnis zurückliefert
- Typen
  - Operatoren haben einen Typ, z.B.
  - $- : int \rightarrow int$
  - $+ : int \times int \rightarrow int$
  - $== : int \times int \rightarrow boolean$

# Operatorschreibweisen

- Präfix

- Operator steht vor dem Operanden
- Z.B. unäres Minus:  $-a$

- Postfix

- Operator steht hinter dem Operanden
- Z.B. Fakultät:  $a!$  (nicht in Java); Postinkrement:  
 $a++$

- Infix

- Operator steht zwischen zwei Operanden
- Z.B. Addition:  $a + b$



# Operatorschreibweisen

- Roundfix
  - Mehrteiliger Operator kreist seinen Operanden ein
  - Z.B. Klammern:  $(a + b)$
- Mixfix
  - Mehrteiliger Operator steht zwischen seinen Operaden
  - Z.B. Integral:  $\int f(x,y) dx$  (nicht in Java),  
Bedingungsoperator:  $a > b ? a : b$

# Überladungen von Operatoren

- Viele Operatoren gibt es in mehreren Überladungen
  - Jede Überladung hat einen eigenen Eingabebereich
  - Jede Überladung definiert eine eigene Funktion
- Beispiel: +
  - $+$  :  $\text{int} \times \text{int} \rightarrow \text{int}$
  - $+$  :  $\text{long} \times \text{long} \rightarrow \text{long}$
  - $+$  :  $\text{float} \times \text{float} \rightarrow \text{float}$
  - $+$  :  $\text{double} \times \text{double} \rightarrow \text{double}$
  - $+$  :  $\text{String} \times \text{String} \rightarrow \text{String}$
  - $+$  :  $\text{long} \times \text{String} \rightarrow \text{String}$
  - $+$  :  $\text{String} \times \text{long} \rightarrow \text{String}$
  - $+$  :  $\text{double} \times \text{String} \rightarrow \text{String}$
  - $+$  :  $\text{String} \times \text{double} \rightarrow \text{String}$

```
byte b = 1;  
b = b + b; // Fehler!  
short s = 1;  
s = s + s; // Fehler!  
int i = 1;  
i = i + i; // ok
```

# Arten von Operatoren

- Arithmetische Operatoren
  - z.B. '+' oder '-'
- Bit-Operatoren
  - z.B. '>>' oder '&'
- Vergleichsoperatoren
  - z.B. '>' oder '<'
- Logische Operatoren
  - z.B. '&&' oder '||'
- Zuweisungsoperatoren
  - z.B. '=' oder '+='
- Sonstige
  - z.B. '(type)' oder '? :'

# Arithmetische Operatoren

- Arithmetische Operatoren verrechnen ein oder zwei Werte zu einem neuen Wert
- Bei zwei Operanden müssen beide einen kompatiblen Typ haben
  - Eigentlich müssen beide Operanden denselben Typ haben, aber Java fügt automatisch Typumwandlungen in Richtung des „größeren“ Typs ein
- Der Ergebnistyp ist identisch mit dem „größeren“ Typ der beiden Operanden

# Arithmetische Operatoren

- Addition
  - $a + b$ ,  $17 + 4$ ,  $17 + 4.0$
  - `"a" + "b"` (`"ab"`),  $17 + "4"$  (`"174"`), `"17" + 4.` (`"174.0"`)
- Subtraktion
  - $a - b$ ,  $17 - 4$ ,  $17 - 4.0$
- Multiplikation
  - $a * b$ ,  $17 * 4$ ,  $17 * 4.0$
- Division
  - $a / b$ ,  $17 / 4$  (`== 4`),  $17 / 4.0$  (`== 4.25`)
- Modulo (Divisionsrest)
  - $a \% b$ ,  $17 \% 4$  (`== 1`),  $17.5 \% 4.5$  (`== 4.0`)

# Arithmetische Operatoren

- Ganzzahlarithmetik
  - Überlauf ist kein Fehler
  - Das Ergebnis wird auf die entsprechende Bit-Anzahl gekappt
  - Division durch 0 ist nicht erlaubt (erzeugt *ArithmeticException: / by zero*)
- Fließkommaarithmetik
  - Division durch 0 ist kein Fehler!
  - $+x / 0.0 == \text{Double.POSITIVE\_INFINITY}$
  - $-x / 0.0 == \text{Double.NEGATIVE\_INFINITY}$
  - $0.0 / 0.0 == \text{Double.NaN}$
  - Bei einem Überlauf ist das Ergebnis *Double.POSITIVE\_INFINITY* oder *Double.NEGATIVE\_INFINITY*

# Logische Operatoren

- Logische Operatoren verrechnen ein oder zwei Wahrheitswerte zu einem neuen Wahrheitswert
- Wahrheitstabellen

- Nicht

false	true
true	false

- Oder

	false	true
false	false	true
true	true	true

- Und

	false	true
false	false	false
true	false	true

- Exklusiv-  
Oder

	false	true
false	false	true
true	true	false

# Logische Operatoren

- Logisches Nicht
  - $!(age > 18) == (age \leq 18)$
- Logisches Und
  - $age > 18 \ \& \ age < 65$
- Logisches Oder
  - $age \leq 18 \ | \ age \geq 65$
- Logisches Exklusiv-Oder
  - $age \leq 18 \ \wedge \ age \geq 65$



# Logische Operatoren mit unvollständiger Auswertung

- Logisches Und

- `age > 18 && age < 65`

- ```
if (d != 0 && 100 / d > 1) {  
    :  
}
```

entsp

```
if (d != 0) {  
    if (100 / d > 1) {  
        :  
    }  
}
```

- Logisches Oder

- `age <= 18 || age >= 65`

- ```
if (d == 0 || 100 / d > 1) {  
    :  
}
```

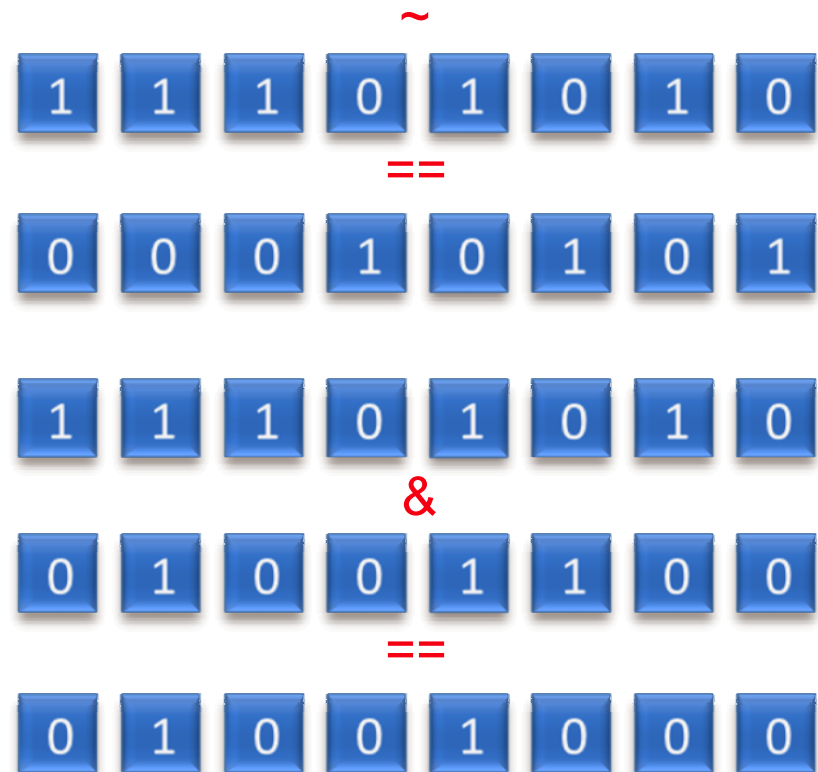
entsp

```
if (d == 0) {  
    :  
} else if (100 / d > 1) {  
    :  
}
```

# Bit-Operatoren

## Bit-Verknüpfungen

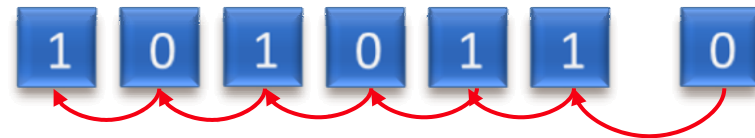
- Bit-Verknüpfungen können auf Ganzzahltypen angewendet werden
- Nicht
  - $\sim 0xea$
- Und
  - $0xea \& 0x4c$
- Oder
  - $0xea | 0x4c$
- Exklusiv-Oder
  - $0xea \wedge 0x4c$



# Bit-Operatoren

## Bit-Verschiebungen

- Links
  - $0x2b \ll 1$
- Rechts
  - $0x2b \gg 1$
- Vorzeichenlos rechts
  - $0x2b \ggg 1$



# Vergleichsoperatoren

- Vergleichsoperatoren liefern immer einen Wahrheitswert (*boolean*) zurück
- Vergleichsoperatoren gibt es für alle Basistypen und Referenzen
- Beide Operanden müssen einen kompatiblen Typ haben
  - Wie bei den arithmetischen Operatoren müssen beide Operanden denselben Typ haben, aber Java fügt automatisch Typumwandlungen in Richtung des „größeren“ Typs ein

# Vergleichsoperatoren

## Numerische Operatoren

- Gleichheit (auch für *boolean*)
  - `age == 18`
- Ungleichheit (auch für *boolean*)
  - `age != 18`
- Größer als
  - `age > 18`
- Größer oder gleich
  - `age >= 18`
- Kleiner als
  - `age < 18`
- Kleiner oder gleich
  - `age <= 18`

```
!(a == b) == (a != b)
(a >= b) == (a > b || a == b)
(a <= b) == (a < b || a == b)
!(a > b) == (a <= b)
!(a < b) == (a >= b)
```

# Vergleichsoperatoren

## Gleichheit von Referenzen/Objekten

- Identität
  - `o1 == o2`
  - Zeigt *o1* auf dasselbe Objekt wie *o2*?
- Gleichheit
  - `o1.equals(o2)`
  - Zeigen *o1* und *o2* auf gleiche Objekte?
  - `equals()` ist in Klasse *Object* definiert, muss aber in abgeleiteten Klassen überschrieben werden, damit diese die Gleichheit bestimmen können

```
Integer i = new Integer(1);  
Integer j = i;  
Integer k = new Integer(1);  
boolean a = i == j; // true  
boolean b = i == k; // false  
boolean c = i.equals(k); // true
```

# Zuweisungsoperatoren

- Zuweisungsoperatoren weisen einen *R-Wert* einem *L-Wert* zu und liefern den (veränderten) *L-Wert* nach der Zuweisung als Ergebnis
- Sie sind die einzigen Operatoren mit Seiteneffekt
  - Sie verändern den *L-Wert*
- Beispiele
  - $i = 7$
  - $s = \text{"Hallo"}$
  - $a[b[i]][2][17] = 42 + i$
  - $x = y = z = 0$
  - $a[x = 7] = 1$

# Zuweisungsoperatoren

## Rechnen und Zuweisung

- Java erlaubt das Verbinden einer Rechenoperation mit der Zuweisung
  - $i += 17$
  - $a[a[1]] <<= 4$
  - $x *= x$
- $L\text{-Wert } op = R\text{-Wert}$  entspricht  $L\text{-Wert} = L\text{-Wert } op R\text{-Wert}$ 
  - aber der  $L\text{-Wert}$  wird nur einmal ausgewertet
  - $a += 7$  entspricht  $a = a + 7$
  - $b[a += 1] += 7$  entspricht nicht  $b[a += 1] = b[a += 1] + 7$



# Zuweisungsoperatoren

- Pre-Inkrement/Dekrement
  - Erhöht/verringert den Wert der Variablen und liefert diesen als Ergebnis zurück
  - `++i; ++a[5]; a[--i] = a[i];`
- Post-Inkrement/Dekrement
  - Erhöht/verringert den Wert der Variablen, liefert aber den alten Wert als Ergebnis zurück
  - `i++; a[5]++; a[i--] = a[i];`

```
int i = 0;
a[++i] = 17; // a[1] = 17
a[++i] = 18; // a[2] = 18
```

```
int i = 0;
a[i++] = 17; // a[0] = 17
a[i++] = 18; // a[1] = 18
```

```
class Counter {
    int counter;
    int preInkrement() {
        counter = counter + 1;
        return counter;
    }
    int postInkrement() {
        int old = counter;
        counter = counter + 1;
        return old;
    }
}
```

## Sonstige Operatoren

- Bedingungsoperator
  - Bedingung ? wenn-wahr-Ausdruck : wenn-falsch-Ausdruck
  - $a = b == 0 ? 0 : 100 / b$
  - $r += r >= \text{Math.PI} ? -2 * \text{Math.PI} : r < -\text{Math.PI} ? 2 * \text{PI} : 0$
- Erzeugen neuer Reihungen oder Objekte
  - `int[] a = new int[10]`
  - `Integer i = new Integer(1)`
- Typumwandlung
  - `int a = (int) 3.1415`
- Test auf Klassentyp
  - `v instanceof Value`

```
class Value {  
    int value;  
    public boolean equals(Object other) {  
        return other instanceof Value &&  
            value == ((Value) other).value;  
    }  
}
```

# Vorrang von Operatoren

- Bindungskraft (Präzedenz)
  - Was wird zuerst ausgewertet, wenn verschiedene Operatoren in einem Ausdruck verwendet werden?
  - z.B. Punkt- vor Strichrechnung:  $1 + 2 * 3 = 1 + (2 * 3) = 7$

- Assoziativität

- Was wird zuerst ausgewertet, wenn Operatoren gleicher Bindungsstärke in einem Ausdruck verwendet werden?

- z.B. links bei Minus:  $3 - 2 - 1 = (3 - 2) - 1 = 0$

- Beispiel

- $a = a \gg 4 \& 0x0f \mid a \ll 4 \& 0xf0$
  - $a = (((a \gg 4) \& 0x0f) \mid ((a \ll 4) \& 0xf0))$



# Vorrang von Operatoren

Operatoren	Asso.	Beschreibung
.,(),[]	links	Objektzugriff, Funktionsaufruf und Array-Index
-,+,!,~,++,-- Dekrement	rechts	Vorzeichen, logisches Nicht, arithm. Nicht, In-,
new,(typ)	rechts	Objekterzeugung und Typumwandlung
*,/,%	links	Multiplikation, Division, Modulo
+, -	links	Addition und Subtraktion
<<,>>,>>>	links	Links- und Rechtsverschiebung
<,<=,>,>=, instanceof Gleich, Typtest		Kleiner, Kleiner-Gleich, Größer, Größer-
==,!=	links	Gleichheit und Ungleichheit
&	links	arithmetisches Und
^	links	arithmetisches Exklusiv Oder
	links	arithmetisches Oder
&&	links	logisches Und
	links	logisches Oder
? :	links	Bedingungsoperator
=,+=,-=,*=,/=,%=,^=	rechts	Zuweisung und Rechenzuweisung
&=, =,<<=,>>=,>>>=	rechts	Rechenzuweisung

Bindungsstärke ↑

# Typерweiterung

- Automatische Typkonvertierung (Typерweiterung)
  - byte → short → int → long → float → double
  - char →
- Entspricht „unsichtbaren“ Operatoren
  - : byte → short
  - : short → int
  - : char → int
  - : int → long
  - : long → float
  - : float → double
- Wahl der Überladung eines Operators
  - Es wird immer die Überladung eines Operators gewählt, die die wenigsten Typерweiterungen erfordert
  - Ausnahme: konstante Ausdrücke werden nach der Berechnung wieder auf den kleinsten Typ reduziert, in dem das Ergebnis noch darstellbar ist