8. Ausdrücke, Operatoren (einfache Typen)

Teil 1

Java-Beispiel:
Unicode.java

Schwerpunkte

- Vollständige und unvollständige Auswertung
- Seiteneffekte
- Overloading: Überladung von Operatoren
- Implizite und explizite (cast) Typumwandlung
- Prioritäten

Einführendes Beispiel: Grundprobleme

Beispiel: Schaltjahr–Test

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

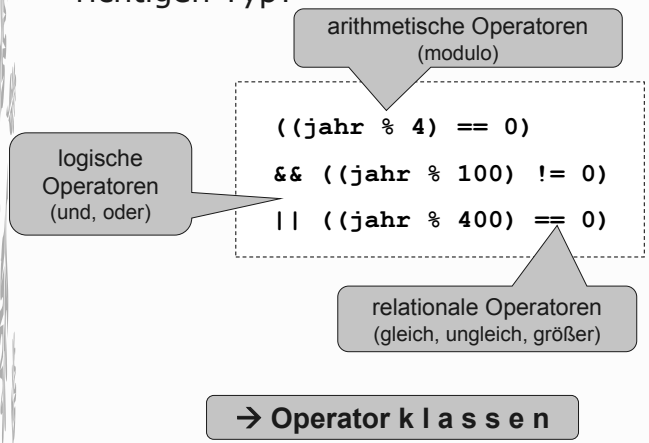
Welche Probleme sind zu klären ?

Syntaktische und semantische Probleme

Schaltjahr:
- alle durch 4 teilbaren Zahlen, die nicht auch durch 100 teilbar sind
- alle durch 400 teilbaren Zahlen

Probleme (1)

- Besitzen die Operanden (z. B. 'jahr') den richtigen Typ?



→ Operator k l a s s e n

Probleme (2)

- Korrekte Zusammenfassung der Teilausdrücke?

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

A
&& B
|| C

A
B
C

A && B || C

→ Welche Zusammenfassung beabsichtigt?
→ Operator-Prioritäten (Bindungsstärke)?
→ Mehr Klammern?
→ Weniger Klammern?

Klammern: mehr oder weniger?

Klammern einsparen?

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

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

Ohne Klammern korrekt, aber schlecht lesbar

Auf oberster Ebene: logisch zwei Teilausdrücke

Zusätzliche Klammern nicht nötig (&& bindet stärker), ggf. besser lesbar

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

Korrekt:

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


Probleme (3)

- Falls Teilausdrücke bereits den Gesamtwert bestimmen: Restlichen Ausdruck auch noch auswerten? (praktisch relevant: Effizienz, Definiertheit des Restes)

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

- ((jahr % 4) == 0) ist falsch
  → . . . && . . . ist falsch
- ((jahr % 4) == 0) && ((jahr % 100) != 0) ist wahr
  → . . . || . . . ist wahr

Java:	&&, ||	mit verkürzter Auswertung
	& |	mit vollständiger Auswertung

Syntax von Ausdrücken in Java: Seiteneffekte

Aufgabe von Ausdrücken: Wert bestimmen

Aufgabe von Anweisungen: Werte verändern

Java: Ausdrücke können beides (problematisch)

Ausdrücke in Java: großzügige Auslegung

- Literale (Zahlen, Zeichenketten, Wahrheitswerte, ...)
- Variablen (+ Parameter)
- Methodenaufrufe

elementar

- Zusammengesetzte Ausdrücke mit arithmetischen, relationalen, logischen und bitweisen Operatoren
- Zuweisungen (!) → Wert der Zuweisung „x = Ausdruck" ist Wert des Ausdrucks
- ...

Kritik (an Java, C):
Keine klare Trennung zwischen Ausdrücken und Anweisungen!

→ Fehlerquelle!

Beispiel: zur Abschreckung !

```
class Zuweisung {
   public static void
            main (String[] args) {
      int x = 2, y = 5, z = 1;
      x = (z++ - (y = y + x));
   }
}
```

Endwerte: x = -6, y = 7, z = 2

Vermeiden: **Seiteneffekte** in Ausdrücken!
(neben der Berechnung eines Wertes: Veränderung von Variablenwerten - *Fehlerquelle*)

Seiteneffekte: z++ und ++z

```
int x = 2,
y = 5, z = 1;
```

```
x = ( z++ - (y = y + x));
```

3 Zuweisungen

Wert = z
danach: z = z + 1
Resultat: x = -6, y=7, z=2

1-7

```
x = ( ++z - (y = y + x));
```

erst: z = z + 1
Wert = z + 1
Resultat: x = -5, y=7, z=2

2-7

Seiteneffekte: grundsätzlich vermeiden!

(aber: es gibt einige wenige sinnvolle Anwendungsfälle)

Aufgabe eines Ausdrucks:
- Werte berechnen
- nicht: Werte verändern

3 Zuweisungen

```
x = (z++ - (y = y + x));
```

```
y = y + x;
x = z - y;
z++;
```

explizite Angabe der zu ändernden Variablenwerte (Reihenfolge!)

```
x = (++z - (y = y + x));
```

```
++z;
y = y + x;
x = z - y;
```


Operatorenübersicht:

- **Typbindung**
- **Overloading**
- **Position der Operanden**

Operatoren: typgebunden

```
a + b / c
```

numerisch: byte, short, ... double

```
ende && anfang
```

boolean

```
a >> b
```

int, long – nicht: short

Zusammenfassung: Primitive Datentypen

Typ	Länge (Byte)	Wertebereich
boolean	1	true, false
char	2	Alle Unicode-Zeichen
byte	1	$-2^7 \ldots 2^7 - 1$
short	2	$-2^{15} \ldots 2^{15} - 1$
int	4	$-2^{31} \ldots 2^{31} - 1$
long	8	$-2^{63} \ldots 2^{63} - 1$
float	4	$+ / - 3.4028234738 * 10^{38}$
double	8	$+ / - 1.79769313486231570 * 10^{308}$

Arithmetische Operatoren

Numerische Operanden, Ergebnistyp: numerisch

Konvertierung in umfassenderen Typ bei unterschiedlichen Operanden-Typen, z.B. 10 + 3.14

+	Positives Vorzeichen	n
-	Negatives Vorzeichen	–n
+	Summe	a + b
-	Differenz	a – b
*	Produkt	a * b
/	Quotient	a / b
%	Restwert	a modulo b
++	Präinkrement	++a ergibt a+1, erhöht a um 1
++	Postinkrement	a++ ergibt a, erhöht a um 1
--	Prädekrement	--a ergibt a-1, verringert a um 1
--	Postdekrement	a-- ergibt a, verringert a um 1

Operatoren: überladen (Overloading)

```
x = a + b
```

```
+:  int    x  int    →  int
+:  float  x  float  →  float
+:  double x  ...
+:  . . .
```

'+': unterschiedliche Operationen

mathematisch — Hardwareoperationen

Effizienz, Speicherplatz

"Overloading": ein Name für verschiedene Operationen

Operatoren: fordern bestimmte Stellung der Operanden

Infix–Operator: `a + b` `a << b`

Postfix–Operator: `a++`

Prefix–Operator: `++a` `!true` `~a` `^a`

Sonstige (3-stellig): `a > b ? a : b`

3 – stelliger Infix-Operator

Relationale Operatoren

Für numerische Operanden (auch gemischt)

(Un-) Gleichheit auch für Objekttypen (Vergleich der Adressen) *!*

Ergebnistyp: boolean

==	Gleich	a == b
!=	Ungleich	a != b
<	Kleiner	a < b
<=	Kleiner gleich	a ≤ b
>	Größer	a > b
>=	Größer gleich	a ≥ b

Typ 'boolean'

Werte: true, false

```
boolean fertig, ...
fertig = false;
fertig = jahr > 1999;
```

Operatoren:	entsprechen Aussagenlogik
!	Negation
&	Konjunktion ('und')
|	Disjunktion ('oder')

Logische Operatoren

Für logische Typen (boolean)

Ergebnistyp: boolean

!	Negation	~a
&&	UND mit teilweiser Auswertung	$a \wedge b$
||	ODER mit teilweiser Auswertung	$a \vee b$
&	UND mit vollständiger Auswertung	$a \wedge b$
|	ODER mit vollständiger Auswertung	$a \vee b$
^	EXKLUSIV-ODER (entweder … oder)	$a \otimes b$

- „Teilweise Auswertung": weiter rechts stehende Teilausdrücke nicht mehr ausgewertet, falls Wert bereits feststeht

- z.B. a && b → false, falls bereits a falsch ist
  → b wird dann nicht mehr ausgewertet

Bitweise Operatoren

Bitmanipulation für `int` bzw. `long`

~	Einerkomplement	~a: Bits von a invertieren
|	Bitweises ODER	a | b: bitweise $a_i \vee b_i$
&	Bitweises UND	a & b: bitweise $a_i \wedge b_i$
^	Bitweises XOR	a^b: bitweise $a_i \otimes b_i$
>>	Rechtsschieben mit Vorzeichen	a >> b: Bits von a um b Positionen nach rechts, Vorzeichen wie bei a
>>>	Rechtsschieben ohne Vorzeichen	a >>> b: Bits von a um b Positionen nach rechts, mit 0 auffüllen, Vorzeichen überschreiben (0)
<<	Linksschieben	a << b: alle Bits von a um b Positionen nach links, mit 0 auffüllen, Vorzeichen wie bei a

a << 1	Multiplikation mit 2
a << 2	Multiplikation mit 4
a << n	Multiplikation mit 2^n

Effizienz

Zuweisungsoperatoren (1)

Zuweisungen sind Ausdrücke der Form

LinkeSeite Zuweisungsoperator Ausdruck

- z.B.: `x = x + y`

LinkeSeite bezeichnet einen Speicherplatz (i.a. einer Variablen)

Unterschied für "x" in `x = x + y ;`

Adresse Wert

- Typ einer Zuweisung = Typ von LinkeSeite
- Wert einer Zuweisung = Wert von Ausdruck

Zuweisungsoperatoren (2)

EBNF:

Zuweisungsoperator ::= = | += | *= | –= | /=
| %= | &= | ^=
| <<= | >>= | >>>=.

Kombination von "=" mit arithmetischen und bitweisen Operatoren in der Form "Operator=" für die Operatoren

\+ | * | - | / | % | & | ^ | << | >> | >>>

Wirkungsweise für x Operator= y jeweils wie x = x Operator y

x += 100	wie	x = x + 100
x <<= 2	wie	x = x << 2

Weitere Operatoren (1)

Fragezeichenoperator:

LogischerAusdruck ? Ausdruck1 : Ausdruck2

ergibt:

Ausdruck1, falls LogischerAusdruck==true
Ausdruck2, falls LogischerAusdruck==false

z. B. x > y ? max = x : max = y

String-Verkettung:

für Zeichenketten (Strings): string1 + string2

```
x = 1; y = 2;
System.out.println(x + y);
System.out.println(x + "" + y);
```

→ **Ausgabe: "3"**
→ **Ausgabe: "12"**

Weitere Operatoren (2)

new–Operator:

Erzeugung von Instanzen: new Typ ([Argumentliste])

mit dem Konstruktor *Typ([Argumentliste])* für die Initialisierung einer Instanz

instanceof–Operator :

InstanzName instanceof KlassenName

Ergebnistyp boolean:

true, falls *InstanzName* eine Instanz der Klasse *KlassenName* bzw. einer Subklasse von *KlassenName* bezeichnet