Übersicht

Inhalte

- ✓ Ausdrücke und Operatoren

Klasse Tools

Ausdrücke

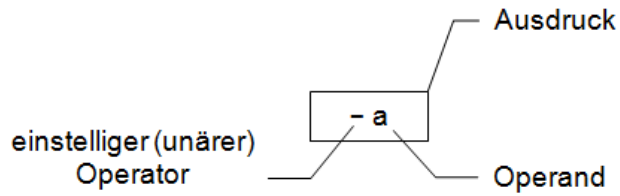
- Alles das, was einen Wert zurückliefert, stellt einen Ausdruck dar.
  - ✓ Im einfachsten Falle der Bezeichner einer Variablen oder Konstanten

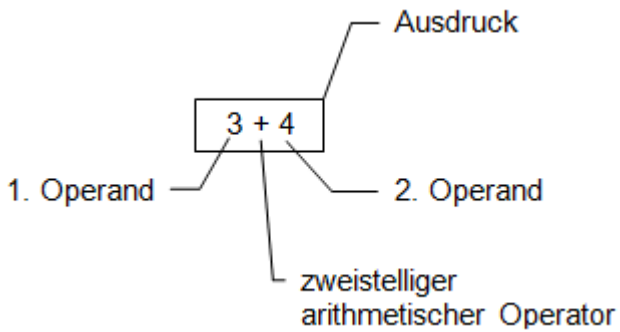
- Verknüpft man Ausdrücke durch sogenannte Operatoren entstehen komplexe Ausdrücke
  - ✓ Die Reihenfolge der Auswertung eines komplexen Ausdrucks kann durch runde Klammern beeinflusst werden.

- Ziel dieser Verknüpfungen ist die Berechnung neuer Werte

Operatoren

- In Java gibt es folgende Arten von Operatoren
  - einstellige (unäre, monadische)
  - zweistellige (binäre, dyadische)
  - einziger dreistelliger (ternären, tryadischen) nämlich Bedingungsoperator ? :





- unäre Operatoren:
  - Postfix-Operatoren hinter dem Operanten
  - Präfix-Operatoren vor dem Operanten

Ausdrücke und Anweisungen

- Ausdrücke haben einen Rückgabewert, Anweisungen nicht
- Wert eines Ausdrucks ist der Rückgabewert. Jeder Rückgabewert hat einen Typ.
- Verschiedene Anweisungen: Selektionsanweisungen, Iterationsanweisungen, Sprunganweisungen, die leere Anweisung, die *try*-Anweisung, die *throw*-Anweisung, die assert-Anweisung, die synchronized-Anweisung und Ausdrucksanweisungen

- Ausdrucksanweisungen:
  - Mit Anhängen eines Semikolons wird der Ausdruck zur Anweisung (Ausdrucksanweisung):
    - (kombinierte) Zuweisungen
    - Postfix- und Präfix-Inkrement- und Dekrementoperator angewandt auf Variablen
    - Methodenaufrufe
    - Ausdrücke, die mit ***new*** ein Objekt erzeugen

Nebeneffekte

- Auch Seiteneffekte oder Nebenwirkungen genannt

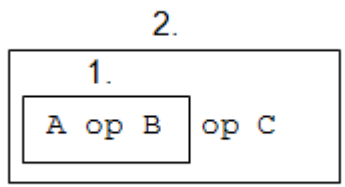
- Zwei Sorten:
  - Nebeneffekte von Operatoren
  - Nebeneffekte bei Methoden, die nicht nur lesend, sondern auch schreibend auf Variable zugreifen

- Beispiel:
  - *int u= 1*
  - *int v*
  - *v = u++; // v == 1, u == 2*

Auswertungsreihenfolge

- Reihenfolge:
  1. Teilausdrücke in Klammern
  2. Ausdrücke mit unären Operatoren von rechts nach links
  3. Teilausdrücke mit mehrstelligen Operatoren

- Abarbeitung erst nach Priorität, bei gleicher Priorität:
  - Assoziativität: Reihenfolge, wie Operatoren und Operanden verknüpft werden, bei derselben Priorität (Vorrangstufe) von Operatoren



- Operanden strikt von links nach rechts auswerten
- Jeder Operand eines Operators erst ausgewertet, bevor mit Operation begonnen wird
  - Ausnahmen: *&&*, || und *? :*

L-Werte und R-Werte

- Ausdruck, der Variable im Speicher bezeichnet, nennt man L-Wert (lvalue oder left value)
- L-Wert besitzt einen Speicherplatz im Arbeitsspeicher
- R-Wert(rvalue oder right value) kann nur rechts stehen, besitzt keine feste Speicherstelle, also kein Wert wird zugewiesen

- Unterschied zwischen modifizierbarem und nicht modifizierbarem L-Wert

Zusammenstellung der Operatoren I

- Einstellige arithmetische Operatoren
  - Positiver Vorzeichenoperator +*a*

    +a → *+a hat denselben Rückgabewert wie a.*

  - Negativer Vorzeichenoperator *-a*

    -a → *-a hat vom Betrag her denselben Rückgabewert wie a, aber mit umgekehrtem Vorzeichen.*

  - Postfix-Inkrementoperator: A++

    a = 1; b = a++; → Erg.: b = 1, Nebeneffekt: a = 2

  - Präfix-Inkrementoperator: ++A

    a = 1; b = ++a; → *Erg.: b = 2, Nebeneffekt: a = 2*

  - Postfix-Dekementoperator: A--

    a = 1; b = a--; → *Erg.: b = 1, Nebeneffekt: a = 0*

  - Präfix-Dekrementoperator: --A

    a = 1; b = --a; → *Erg.: b = 0, Nebeneffekt: a = 0*

Zusammenstellung der Operatoren II

- Zweistellige arithmetische Operatoren
  - Additionsoperator A + B
  - Subtraktionsoperator A – B
  - Multiplikationsoperator A * B

```
// Punkt vor Strichrechnung
3 * 5 + 3     // Erg.: 18
3 * (5 + 3)   // Erg.: 24
```

  - Divisionsoperator A / B
  - Ist mindestens ein Operand eine `double`- oder `float`-Zahl, so ist das Ergebnis eine Gleitpunktzahl

```
5 / 5         // Erg.: 1
5 / 3         // Erg.: 1
11.0 / 5      // Erg.: 2.2
```

→ Division durch 0 führt in Java nicht zum Absturz, Ausnahme wird ausgelöst, bei Ganzzahldivision *ArithmeticException,* bei Gleichpunktdivision Ergebnis *Infinity*

Zusammenstellung der Operatoren III

- Zweistellige arithmetische Operatoren
  - Restwertoperator (Moduloperator) A % B (Gibt den ganzzahligen Rest bei Division von A durch B)

```
  5  %   3     =  2
  8  %   4     =  0
  3  %   7     =  3
(-7) %   2     = -1   // denn (-7) / 2 ergibt -3 (2 * -3 = -6 | Rest -1)
(-7) % (-2)    = -1   // denn (-7) / (-2) ergibt 3
  7  % (-2)    =  1   // denn 7 / (-2) ergibt -3
  7  %   2     =  1   // denn 7 / 2 ergibt 3
```

Zusammenstellung der Operatoren IV

- Zuweisungsoperatoren:
  - Einfache: A = B
  - Kombinierte:
    - Zum Beispiel: Additions-Zuweisungsoperator: A += B

```
a += 1          // hat den gleichen Effekt wie ++a
```

- Ausdruck rechts des Zuweisungsoperators wird implizit in den Typ der Variablen links des Zuweisungsoperators gewandelt

- Sonstige kombinierte Zuweisungsoperatoren

```
a -= 1          // a = a - 1
b *= 2          // b = b * 2
c /= 5          // c = c / 5
d %=5           // d = d % 5
```

Zusammenstellung der Operatoren V

- Relationale Operatoren (auch Vergleichsoperatoren genannt)
- Gleichheitsoperator A==B
  - stimmt der Wert links mit dem Wert rechts überein, wird ein true zurückgeliefert

```
3 == 3              // Erg.: true

2 == 3              // Erg.: false
```

- Ungleichheitsoperator: A != B
  - stimmt der Wert links mit dem Wert rechts nicht überein, wird ein true zurückgeliefert

```
5 != 5              // Erg.: false
3 != 5              // Erg.: true
```

- Größer- und Kleineroperator: A > B und A < B

```
5 > 3               // Erg.: true
3 > 3               // Erg.: false
5 < 5               // Erg.: false

2 >= 1              // Erg.: true
1 <= 1              // Erg.: true
```

Zusammenstellung der Operatoren VI

- Logische Operatoren
- Können nur auf Operanden vom Typ *Boolean* und *boolean* angewendet werden

  - Logische UND: *A && B* und *A & B*

A	B	A && B
false	false	false
false	true	false
true	false	false
true	true	true

  - Logische ODER: *A || B* *A | B*

A	B	A || B
false	false	false
false	true	true
true	false	true
true	true	true

  - Verknüpfungsreihenfolge: Wenn *&&* oder || -Operatoren gemischt sind, dann gilt nicht prinzipiell links nach rechts, da *&&* höhere Priorität hat.

Zusammenstellung der Operatoren VI

- Logische Operatoren

  - Exklusiv – ODER Operator: *A ^ B*
    (Liefert nur dann *true*, wenn ***genau einer*** der Operanden den Wert true hat und der andere nicht)

A	B	A ^ B
false	false	false
false	true	true
true	false	true
true	true	false

  - Negationsoperator: *!A*
    (Wahrheitswerte werden negiert)

A	!A
true	false
false	true

Logische Bit Operatoren I

- UND-Operator für Bits: A & B
  - Finden auf allen Bits der Operanden statt, bei Bit-Operationen werden jeweils die Bits der entsprechenden Position miteinander verknüpft.

Bit n von A	Bit n von B	Bit n von A & B
0	0	0
0	1	0
1	0	0
1	1	1

```
byte b1 = 14;     //  1110
byte b2 = 3;      //& 0011
                  //  0010 → dezimal 2

System.out.println("b1 & b2 = " + (b1 & b2));
```

```
b1 & b2 = 2
```

Logische Bit Operatoren II

- ODER-Operator für Bits: A | B
  - Finden auf allen Bits der Operanden statt, bei Bit-Operationen werden jeweils die Bits der entsprechenden Position miteinander verknüpft.

Bit n von A	Bit n von B	Bit n von A | B
0	0	0
0	1	1
1	0	1
1	1	1

```
byte b1 = 14;     //  1110
byte b2 = 3;      //| 0011
                  //  1111 → dezimal 15

System.out.println("b1 | b2 = " + (b1 | b2));
```

```
b1 | b2 = 15
```

Logische Bit Operatoren III

- Exclusiv-ODER-Operator für Bits: A ^ B
  - die Operation bitweises Exklusiv-ODER findet auf allen Bits der Operanden statt

Bit n von A	Bit n von B	Bit n von A ^ B
0	0	0
0	1	1
1	0	1
1	1	0

```
byte b1 = 14;     //  1110
byte b2 = 3;      //^ 0011
                  //  1101 → dezimal 13

System.out.println("b1 ^ b2 = " + (b1 ^ b2));
```

```
b1 ^ b2 = 13
```

Logische Bit Operatoren IV

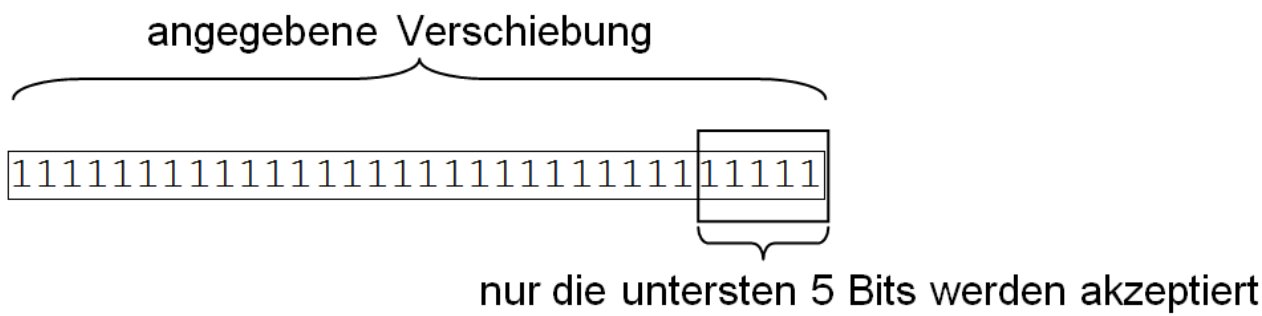
- Negationsoperator für Bits: ~A
  - die Operation einer bitweisen Negation finden auf allen Bits der Operanden statt

Bit n von `A`	Bit n von `~A`
0	1
1	0

```
int a = 9;          //  a = 00000000 00000000 00000000 00001001
int b = ~a;         //  b = 11111111 11111111 11111111 11110110
                    //  b → dezimal -10
```

Shift Operatoren für Bits I

- Shift-Operatoren sind Verschiebeoperatoren
- der linke Operand eines Shift-Operators ist der zu verschiebende Wert
- der rechte Operand gibt die Anzahl Stellen an, um die verschoben wird
  - *ist der linke Operand ein int-Wert, werden nur die 5 niederwertigsten Bits des rechten Operanden akzeptiert → daher kann nur um 0 bis 31 Stellen verschoben werden*
  - *bei einem long-Typ werden nur die 6 niederwertigsten Bits akzeptiert*



Shift Operatoren für Bits II

- Vorzeichenbehafteter Rechtsshift-Operator: A >> B
  - es werden B Bitstellen von A nach rechts verschoben

```
int a;

                                                          verloren
a = 8;              // 00000000 00000000 00000000 00001000
                       ---------------------------------------->
a = a >> 3;         // 000 00000 00000000 00000000 00000001
                    aufgefüllt
                                                          verloren
a = -7;             // 11111111 11111111 11111111 11111001
                       ---------------------------------------->
a = a >> 3;         // 111 11111 11111111 11111111 11111111
                     aufgefüllt
```

Shift Operatoren für Bits III

- Vorzeichenloser Rechtsshift-Operator: A >>> B
  - es werden B Bitstellen von A nach rechts verschoben
  - dabei werden stets Nullen von links nachgeschoben

```
int a;

                                                        verloren
a = 8;              // 00000000 00000000 00000000 00001000
                       ---------------------------------------->
a = a >>> 3;        // 000 00000 00000000 00000000 00000001
                    aufgefüllt
                                                        verloren
a = -7;             // 11111111 11111111 11111111 11111001
                       ---------------------------------------->
a = a >>> 3;        // 000 11111 11111111 11111111 11111111
                     aufgefüllt
```

Shift Operatoren für Bits IV

- Linksshift-Operator: A << B
  - es werden B Bitstellen von A nach links verschoben und mit Nullen aufgefüllt

```
int a;

                    verloren
a = 8;              // 00000000 00000000 00000000 00001000
                       <----------------------------------
a = a << 3;         // 00000000 00000000 00000000 01000000
                                                   aufgefüllt
```

Bedingungsoperator

- Bedingungsoperator
- A ? B : C
  - einziger Operator mit drei Operanden
  - zunächst wird der boolesche Ausdruck A ausgewertet
  - ist A *true*, wird B ausgewertet, ist A *false*, dann wird C ausgewertet

Wir haben das if/else-Konstrukt zwar noch nicht kennengelernt, aber

```
return A ? B : C;
```

heißt nichts anderes

```
if(A)
    return B;
else
    return C;
```

CAST Operator

- CAST Operator
- Typkonvertierungsoperator – zur expliziten Typumwandlung
  - Erfolgt durch (Typname) Ausdruck

```
int a = 1;          // a hat den Wert 1
double b = 3.5;     // b hat den Wert 3.5
a = (int) b;        // Explizite Typkonvertierung in den Typ int

a = (int) 4.1       // a bekommt den Wert 4 zugewiesen
a = (int) 4.9       // a bekommt ebenfalls den Wert 4 zugewiesen
```

- Implizite Typumwandlung
  - z.B. bei Verknüpfung von String-Objekten mit Operanden anderer Datentypen

```
int a = 8;
System.out.println("Der Baum ist " + a + " Meter hoch.");
```

Prioritätentabelle Operatoren

Priorität	Operatoren	Bedeutung	Assoziativität
Priorität 1	`[]`	Array-Index	links
	`()`	Methodenaufruf	links
	`.`	Komponentenzugriff	links
	`++`	Postinkrement	links
	`--`	Postdekrement	links
Priorität 2	`++`	Präinkrement	rechts
	`--`	Prädekrement	rechts
	`+ -`	Vorzeichen (unär)	rechts
	`~`	bitweises Komplement	rechts
	`!`	logischer Negationsoperator	rechts
Priorität 3	`(type)`	Typ-Umwandlung	rechts
	`new`	Erzeugung	rechts
Priorität 4	`* / %`	Multiplikation, Division, Rest	links
Priorität 5	`+ -`	Addition, Subtraktion	links
	`+`	Stringverkettung	links
Priorität 6	`<<`	Linksshift	links
	`>>`	Vorzeichenbehafteter Rechtsshift	links
	`>>>`	Vorzeichenloser Rechtsshift	links
Priorität 7	`< <=`	Vergleich kleiner, kleiner gleich	links
	`> >=`	Vergleich größer, größer gleich	links
	`instanceof`	Typüberprüfung eines Objektes	links
Priorität 8	`==`	Gleichheit	links
	`!=`	Ungleichheit	links
Priorität 9	`&`	bitweises/logisches UND	links
Priorität 10	`^`	bitweises/logisches Exclusiv-ODER	links
Priorität 11	`|`	bitweises/logisches ODER	links
Priorität 12	`&&`	logisches UND	links
Priorität 13	`||`	logisches ODER	links
Priorität 14	`? :`	Bedingungsoperator	rechts
Priorität 15	`=`	Wertzuweisung	rechts
	`*= /= %= += -= <<= >>= >>>= &= ^= |=`	kombinierter Zuweisungsoperator	rechts

- Rechtsassoziativ sind Zuweisungsoperatoren, der Bedingungsoperator und unäre Operatoren. Alle anderen sind linksassoziativ.

Aufgabe 04.01

- Verständnisfragen

a. Welche Arten von Operatoren gibt es in Java
   - nach Anzahl der Operatoren?
   - nach Art der Wirkungsweise?

b. Was sind Ausdrücke?

c. Was sind Anweisungen?

d. Was sind Nebeneffekte?

Aufgabe 04.02

- Finden Sie ohne Java-Compiler (also händisch) heraus, welchen Wert die Variablen a und b nach den einzelnen Anweisungen haben. Vor jeder Anweisung seien folgende Werte gegeben:

```
int a = 2;
int b = 1;
```

a) `a = b = 2;`

b) `a = b * 3 + 2;`

c) `a = b * (3 + 2);`

d) `a *= b + 5;`

e) `b %= 2 * a;`

f) `a = --b;`

g) `b = ~a;`

h) `b = b++ * a;`

i) `a = - 5 - 5;`

j) `b = b << 2;`

k) `b = (a == b) ? 5 : 7;`

l) `a = --b * b++;`

m) `a = a ^ b;`

Aufgabe 04.03

- Implizite Konvertierung bei numerischen Datentypen

  Welche Ausgabe erhalten Sie von folgendem Programm? Begründen Sie Ihre Antwort! Hier das Programm:

```
// Datei: ImpliziteKonvertierung.java

public class ImpliziteKonvertierung{

    public static void main (String[] args){

        System.out.println ("Division von 10 durch 12: " + (10/12));
        System.out.println ("Division von 10. durch 12: " + (10./12));
        System.out.println ("Division von 10 durch 12.: " + (10/12.));
    }
}
```

Übersicht

Inhalte

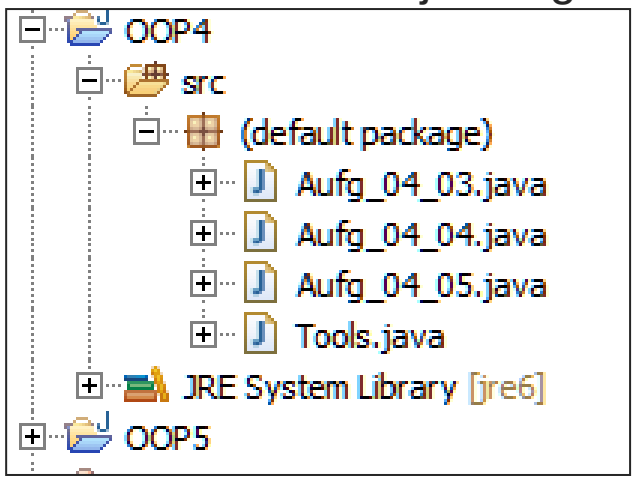
- ✓ Ausdrücke und Operatoren
- ✓ Klasse Tools

Die Klasse Tools

- die Klasse Tools enthält statische Methoden zum Einlesen von Eingaben über die Systemeingabe (int, double, boolean, String)
  - public static int intEingabe()
  - public static String stringEingabe()
  - public static double doubleEingabe()
  - public static boolean booleanEingabe()
- Sie müssen diese Klasse noch nicht verstehen, verwenden Sie sie einfach zum Einlesen von Eingaben!
- Wie geht das?
  - siehe nächste Folie

Die Klasse Tools

- die Klasse Tools (Tools.java) aus dem Online-Campus herunterladen und in das jeweilige Eclipse-Projekt hineinkopieren
- **ODER**
- in Eclipse eine neue Klasse namens Tools anlegen und den Quelltext aus der Datei Tools.java eingeben bzw. hineinkopieren



- die Klasse Tools.java sollte nun auf der gleichen Ebene (Package) sein, wie die anderen Klassen

- Über den Befehl *Tools.Funktionsname()* können Sie nun die jeweiligen Werte von der Eingabe einlesen

```
public static void main(String[] args) {

    int punkte = Tools.

    System.out.println(

}
```

intEingabe() : int - Tools
class : Class<Tools>
booleanEingabe() : boolean - Tools
doubleEingabe() : double - Tools
stringEingabe() : String - Tools

Aufgabe 04.05

- Arithmetische Operatoren

  Vervollständigen Sie die folgende Klasse unter Verwendung von arithmetischen Operatoren. Die einzulesenden Tage sollen umgerechnet werden. Es soll angezeigt werden, wie viele Stunden, Minuten und Sekunden das jeweils insgesamt sind.

```
public class Aufg_04_05 {

    public static void main(String[] args) {

        int sekunden=0, minuten=0, stunden=0, tage=0;
        tage = Tools.intEingabe();

        // Berechnungen mit arithmetischen Operatoren
        // ...

       System.out.println("Das sind");
       System.out.println(stunden + " Stunden oder");
       System.out.println(minuten + " Minuten oder");
       System.out.println(sekunden + " Sekunden");
    }
}
```

Aufgabe 04.04

- Arithmetische Operatoren

  Vervollständigen Sie die folgende Klasse unter Verwendung von arithmetischen Operatoren. Die einzulesenden Sekunden sollen umgerechnet werden. Es soll angezeigt werden, wie viele Tage und (Rest-)Stunden, (Rest-)Minuten und (Rest-)Sekunden das sind.

```
public class Aufg_04_04 {

    public static void main(String[] args) {

        int sekunden=0, minuten=0, stunden=0, tage=0;
        sekunden = Tools.intEingabe();

        // Berechnungen mit arithmetischen Operatoren
        // ...

       System.out.println("Das sind");
       System.out.println(tage + " Tage und");
       System.out.println(stunden + " Stunden und");
       System.out.println(minuten + " Minuten und");
       System.out.println(sekunden + " Sekunden");
    }
}
```