Ausdrücke (expressions)

Sven Eric Panitz

Wann immer in einem Programm Daten erzeugt werden, benötigt man einen **Ausdruck**.

Oder umgekehrt:

Ein Ausdruck wertet immer zu einem Wert aus.

Ein Ausdruck hat immer ein Rechenergebnis.

Kleine Faustregel

Alles, was man in der Jshell eingibt und zu einem Ergebnis ausgwertet wird, das in einer Variablen gespeichert wird, ist ein Ausdruck.

Kleine Faustregel 2

Alles, was man `System.out.println(...)` übergeben kann, ist ein Ausdruck.

Wo braucht man Ausdrücke?

- Zuweisung an Variablen:

```
int x = 42;
```

- Rückgabe von Methodenergebnis:

```
int fortytwo(){return 42;}
```

- Übergabe von Parametern:

```
f(42);
```

- Und viel mehr…. (Bedingungen, Schleifen)

Übersicht für Ausdrücke

- Literale: `42`
- Operatorausdrücke: `(17 + 4) * 2`
- Methodenaufrufe: `fac(5)`
- Variablen, Parameter, Feldzugriff: `x`
- Erzeugung neuer Objekte mit `new`:
  `new Date(10,5,1967)`
- Lambda Ausdrücke: `xs.map( x -> x*x )`

Literale

Literale für ganze Zahlen

- Ganze Zahlen (int): `42` `-17`
- Lange ganze Zahle (long): `3405691582L`
- Oktalzahlen (int): `077`
- Hexadezimalzahlen: `0xFF` `0x1` `0xCAFEBABEL`
- Binärzahlen: `0b101010`

```
jshell> var x1 = 42;
   ...> var x2 = -17;
   ...> var x3 = 3405691582L;
   ...> var x4 = 077;
   ...> var x5 = 0xFF;
   ...> var x6 = 0xCAFEBABEL;
   ...> var x7 = 0B101010;

x1 ==> 42
x2 ==> -17
x3 ==> 3405691582
x4 ==> 63
x5 ==> 255
x6 ==> 3405691582
x7 ==> 42
```

```
jshell> /vars
|     int x1 = 42
|     int x2 = -17
|     long x3 = 3405691582
|     int x4 = 63
|     int x5 = 255
|     long x6 = 3405691582
|     int x7 = 42
```

Literale für Kommazahlen

- `17.42`
- `8f`
- `8d`
- `314159E-5`

Zeichen und Textliterale

- Einzelne Zeichen (char):

```
'a'

'\u03A9'
```

- Stringobjekte:

```
"Hallo Welt"
"mit Zeilenende\nund Anführung\" dazu"
```

Fluchtsequenzen für Zeichen

- \t Tabulatorzeichen
- \b Backspace
- \n neue Zeile
- \r Rücklauf (carriage return)
- \f formfeed
- \' einfaches Anführungszeichen
- \" doppeltes Anführungszeichen
- \\ rückwärtiger Schrägstrich (backslash)

Bool`sche Literale

- `true`
- `false`

Achtung: bool`sche Werte haben nichts mit den Zahlen 1 und 0 zu tun.

Operatorausdrücke

Übersicht: Operatorausdrücke

- Arithmetische Operatoren: `+ -, *, /, %`
- Vergleichsoperatoren: `<, <=, >, >=, ==, !=`
- Logische Operatoren: `||, &&, !`
- Bedingungsoperator: `? :`

- Bitoperatoren: `|, &, ^, ~, <<, >>, >>>`

Infixschreibweise

- Alle binären Operatoren werden infix benutzt.

  D.h. der Operator steht zwischen den Operanden:

  ```
  17 + 4
  ```

  im Gegensatz:

  zu präfix z.B. in Lisp: `(+ 17 4)`

  oder postfix z.B. in Postscript oder Forth: `17 4 +`

Arithmethische Operatoren

- Addition: `17+4`
  funktioniert für alle Zahlentypen
  Zusätzlich: wird auch verwendet für Strings.
- Subtraktion: `17-4`
  Auch unär: `-17`
- Multiplikation: `1.42 * 5`

  (für alle Zahlentypen)
- Division: `17/4`

  Bei ganzen Zahlen nur der ganzzahlige Anteil
- Moduloperation, Rest bei Division: `17%4`

Es gilt:

Punktrechnung vor Strichrechnung:
`17 + 4 * 2` wertet zu `25` aus

Ansonsten Klammern verwenden:
`(17 + 4) * 2` wertet zu `42` aus

Vergleichsoperatoren

- Zwei Zahlen lassen sich bezüglich der Größe vergleichen. Ergebnis ist ein Wahrheitswert.
- Es gibt 6 Vergleichsoperatoren: `<, <=, >, >=, ==, !=`
- `==, !=` sind auch für Objekte verwendbar.

  Machen dort allerdings meist nicht das, was man erwartet. (mehr dazu später)

Logische Operatoren

- Für die logischen Operationen ¬, ∧, ∨ stehen in Java folgende Operatoren zur Verfügung: `!`, `&&`, `||`
- Die beiden binären Operatoren werten eventuell den zweiten Operanden nicht aus, wenn das Ergebnis durch den ersten Operanden bereits fest steht:

```
true || somethingRotten
false && somethingRotten
```

Hier wird somethingRotten nicht ausgewertet.

Der Bedingungsoperator

- Es gibt einen terziären Operator, also einen Operator mit drei Operanden.
- Der erste Operand muss zu einem Wahrheitswert auswerten.
- Zweiter und dritter Operand müssen vom gleichen Typ sein.

```
(17+4 < x) ? "hello" : "Hallo"
```

- Wenn der erste Operand zu true auswertet, dann wird der zweite Operand zum Ergebnis ausgewertet, sonst der dritte Operand.

Bitoperationen

- & bitweise Und
- | bitweise Oder
- ^ bitweise exklusives Oder
- ~ bitweise Negat
- << Shift nach links
- >> Shift nach rechts
- >>> Shift nach rechts. Vorzeichen bleibt erhakten

Zuweisungsoperatoren

- Es gibt noch Bastarde, die Operatoren sind, also einen Wert berechnen, aber zusätzlich Variablen verändern:

  `+=, -=, *=, /=, %=, ++, --, >>>=, ...`

- Hier nicht weiter erklärt.

Funktionsaufrufe

- Der Aufruf einer Methode, die nicht als void deklariert ist, ist ein Ausdruck.

```
jshell> long f(long x){
   ...>     return x*x;
   ...> }
|  created method f(long)

jshell> f((17+4)*2)
$9 ==> 1764
```

Variablen, Felder und Parameter

- Variablen, Parameter und Felder können als Ausdruck verwendet werden.
- Der Wert ist dann der in diesen Variablen gespeicherte Wert.

```
jshell> var x = 42
x ==> 42

jshell> x
x ==> 42
```

Objekterzeugung mit new

- Auch der Aufruf eines Konstruktors zur Erzeugung eines neuen Objektes, ist ein Ausdruck.
- Das neu erzeugte Objekt ist der Wert des Ausdrucks.

```
jshell> record Person(String name){}
|  created record Person

jshell> new Person("Jupp Egon")
$13 ==> Person[name=Jupp Egon]

jshell>
```

Fallunterscheidung mit switch

```
int tageImMonat(int monat){
      return switch(monat){
        case 2 -> 28;
        case 4 -> 30;
        case 6 -> 30;
        case 9 -> 30;
        case 11 -> 30;
        default -> 31;
      };
  }
```

```
jshell> tageImMonat(4)
$15 ==> 30

jshell> tageImMonat(8)
$16 ==> 31

jshell>
```

λ-Ausdrücke (Lambda-Ausdrücke)

- Mit λ-Ausdrücken lassen sich eine bestimmte Art von Objekten erzeugen. (statt mit new)
- Beispiel:
```
Function<Integer,Integer> f = (x) -> x * x;
```
- Bei λ-Ausdrücken versagt manchmal die `System.out.println()`-Regel.
- Mehr zu λ-Ausdrücken später im Semester.

Alles lässt sich beliebig kombinieren

```
x = (y,z) ->
   f(y>=1*(z-g())
     ?new Bla(14^2)+"\""
     :str.toUpperCase()
    ,314159E-5)
```

Fazit:

Wann immer Daten erzeugt werden sollen, benötigt man Ausdrücke, die zu einem Ergebnis auswerten.