4.2 Ausdrücke

- Ein Ausdruck setzt sich zusammen aus Konstanten, Variablen und Operationen/Operationsaufrufen.
  - Durch Auswertung eines Ausdrucks wird ein Wert berechnet.
  - Jeder Ausdruck ist einem Datentyp zugeordnet.
- Beispiel: sum + 1.0f / fakultaet(i)

Formale Definition (rekursiv):

- **Ein Ausdruck A vom Typ D entspricht genau einem der folgenden drei Fälle:**
  i. A ist eine Konstante vom Typ D,
  ii. A ist eine Variable vom Typ D,
  iii. A ist eine Operation $f(t_1, \ldots, t_m)$ mit $f: D_1 \times \ldots \times D_m \rightarrow D$ und $t_1, t_2, \ldots, t_m$ sind Ausdrücke der Typen $D_1, \ldots, D_m$.

Auswertung von Ausdrücken

- Ein Ausdruck lässt sich folgendermaßen **rekursiv** berechnen:
  i. Falls der Ausdruck aus genau einer Konstanten vom Typ D besteht, so entspricht der Wert des Ausdrucks dem der Konstanten zugeordneten Werts des Datentyps D.
  ii. Falls der Ausdruck aus genau einer Variablen besteht, so ist der Wert des Ausdrucks gleich dem aktuellen Wert der Variablen.
  iii. Falls der Ausdruck aus einer Operation $f(t_1,\ldots,t_m)$ besteht, so wird zunächst der Wert w_i des Ausdrucks t_i berechnet, i = 1,…,m. Der Wert des Ausdrucks $f(t_1,\ldots,t_m)$ ergibt sich dann aus $f(w_1,\ldots,w_m)$.
- Auswertungsreihenfolge:
  - Bei der Auswertung von Ausdrücken bestimmt die Priorität der einzelnen Operatoren die Auswertungsreihenfolge.
    - bekannte Regel: Punkt- vor Strichrechnung
    - Auswertungsreihenfolge kann durch Setzen von Klammern geändert werden.

Auswahl von Operatoren

(nach Prioritäten absteigend sortiert)

höchste Priorität

Priorität aufsteigend

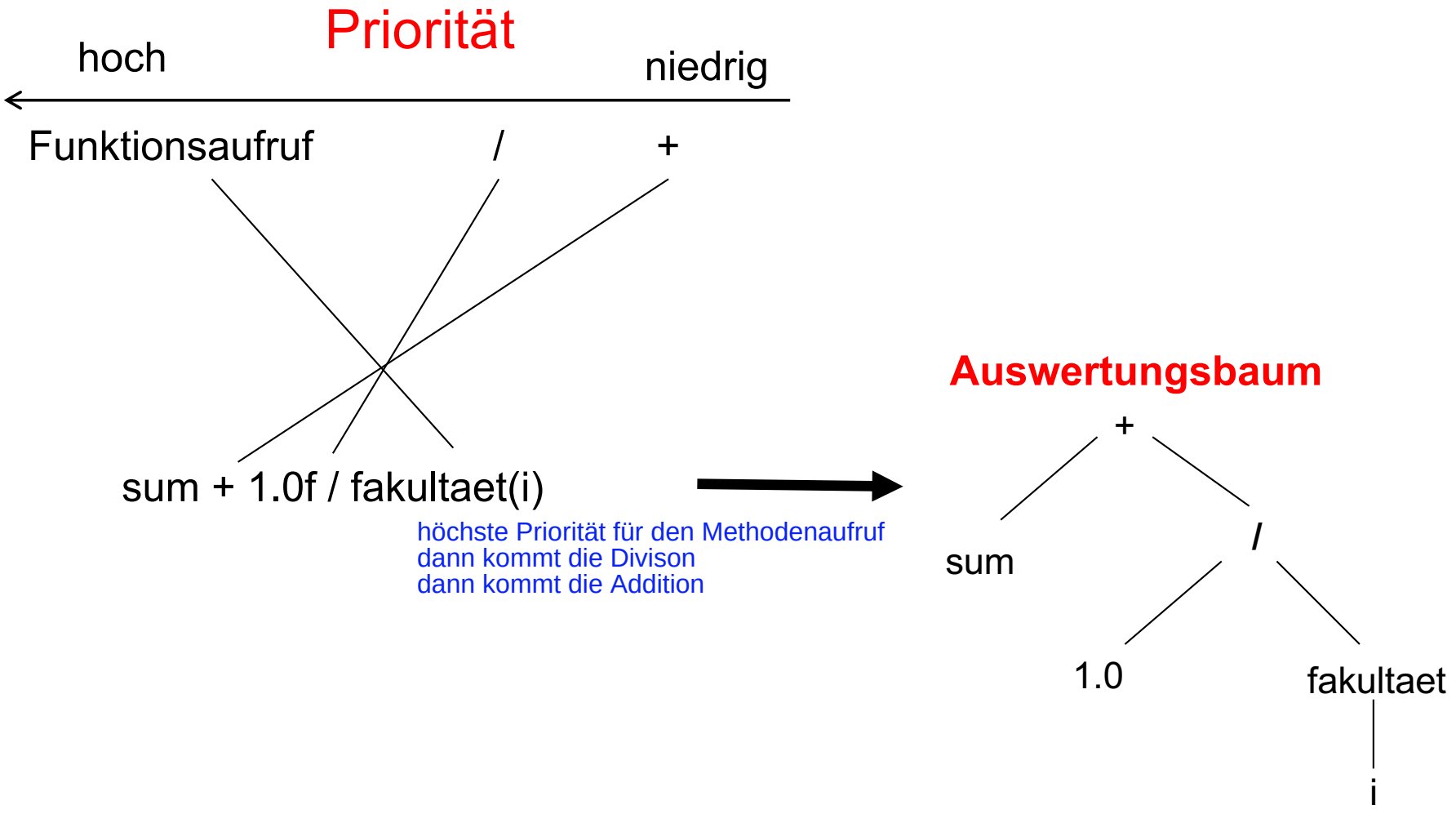
Methodenaufruf	f(Parameterliste)
Postfix-Operatoren	var++ var--
unäre Operatoren	++var --var +expr -expr !expr ~expr
Typumwandlung	(Typ) expr
Multiplikationsoperatoren	expr*expr expr/expr expr%expr
Additionsoperatoren	expr+expr expr-expr
Vergleichsoperatoren	expr<expr expr>expr expr>=expr expr<=expr
Gleichheitsoperatoren	expr==expr expr!=expr
Bitoperatoren	expr & expr
	expr ^ expr
	expr | expr
verkürztes logisches Und	expr && expr
verkürztes logisches Oder	expr || expr
bedingter Zuweisungsoperator	expr ? expr : expr
Zuweisungsoperatoren	var=expr, var+=expr, var*=expr, var/=expr

niedrigste Priorität

var steht für Variable

expr steht für Ausdruck

Beispiel


Priorität
hoch
niedrig
Funktionsaufruf
/
+
sum + 1.0f / fakultaet(i)
höchste Priorität für den Methodenaufruf
dann kommt die Divison
dann kommt die Addition
Auswertungsbaum
+
sum
/
1.0
fakultaet
i

Seiteneffekte in Java

- Bisher haben wir bei der Auswertung eines Ausdrucks angenommen, dass die Reihenfolge der Auswertung der Operanden keinen Einfluss auf den Wert hat.
- Bei einer imperativen Programmiersprache kann der Wert aber von der Reihenfolge der Auswertung abhängen **(Seiteneffekt)**.
- In Java ist daher festgelegt, dass die Auswertung der Ausdrücke $f(t_1, \ldots t_m)$ bzw. (s op t) von links nach rechts erfolgt
  - d.h., t_i wird vor t_{i+1} für i = 1,…,m-1 bzw. s vor t ausgewertet

Hier bedeutet dies, dass zuerst geprüft wird, ob y ungleich 0 ist. Wenn y gleich 0 ist, liefert der Ausdruck sofort false zurück, und der zweite Teil (x/y > 1) wird nicht ausgewertet. Das verhindert eine Division durch Null, was einen Fehler verursachen würde.

Beispiel:

- Der Ausdruck ((y != 0) && (x/y > 1)) ist eine Kurzform für die Schreibweise

```
if (y == 0) return false; else return x/y > 1;
```

- Seiteneffekte können z.B. auftreten, wenn lesend und schreibend auf eine gemeinsame Variable zugegriffen wird.

Ausdrücke vs. Anweisungen

In Java wird zwischen Ausdrücken und Anweisungen nicht sauber unterschieden.

```
v = A
```

ist keine Anweisung, sondern ein ***Zuweisungsausdruck***. D.h. er hat einen Wert ***und*** bewirkt einen ***Seiteneffekt***. Die Auswertung eines solchen Ausdrucks erfolgt (im Gegensatz zum sonstigen Vorgehen) ***von rechts nach links***.

Wenn `v` eine Variable und `A` ein Ausdruck ist, dann bewirkt die ***Auswertung*** von `v = A`

1. Ergebnis des Zuweisungsausdrucks ergibt sich aus dem Ergebnis von `A`.
2. Das Ergebnis wird `v` zugewiesen.

Dies lässt sich zur folgenden ***Mehrfach-Zuweisung*** verallgemeinern:

```
vn = .. = v1 = A
```

Dieser Ausdruck wird ***rechts-assoziativ*** ausgewertet, d.h. alle Variablen `v1, ..., vn` bekommen den Wert von `A`.

1. `A` wird ausgewertet
2. Das Ergebnis wird `v1` zugewiesen.

   ...

(n+1). Das Ergebnis wird `vn` zugewiesen.

Weitere Zuweisungsoperatoren

Eine verkürzte Schreibweise für bestimmte Zuweisungsausdrücke lautet

`v op= A` und steht für `v = v op A`,

wobei `op` ein (arithmetischer oder Schiebe-) Operator ist.

Für die besonders häufigen Zuweisungen

```
v += 1
v -= 1
```

bzw.

```
v = v + 1
v = v - 1
```

gibt es die weitergehenden Abkürzungen

```
v++
v--
```

und

```
++v
--v
```

Diese sog. „***Autoinkrement-Operatoren***“ sind ebenfalls ***Ausdrücke***, es gibt sie in ***Präfix***- und in ***Postfix***-Form (mit unterschiedlicher Bedeutung):

```
++v  // erhöht v um 1 und liefert den erhöhten Wert
v++  // erhöht v um 1 und liefert den ursprünglichen Wert
```

Live Vote

PIN: QKWU

https://ilias.uni-marburg.de/vote/QKWU

Seiteneffekt und Inkrementierung

- Wir betrachten die Methode printInts mit zwei Parametern.

```
void printInts(int n, int m){
        System.out.println(n + "  " + m);
}
```

- *Was passiert beim Aufruf von printInts?*

```
int i = 42;

printInts(i++, i);
```

```
int i = 42;

printInts(++i, i);
```

```
int i = 42;

printInts(i, i++);
```

Empfehlungen

- Seiteneffekte sollten möglichst vermieden werden.
- Mehrere Inkrement- und Dekrementoperationen sollten in einem Ausdruck generell vermieden werden.
  - Verwenden Sie stattdessen den Zuweisungsoperator += (z. B. i += 1;), um Seiteneffekte zu vermeiden und die Lesbarkeit zu verbessern.
- Wenn mehrere Inkrement- und Dekrementoperationen in einem Ausdruck verwendet werden, dann sollten diese sich auf unterschiedliche Variablen beziehen.

Typumwandlung

- Manchmal ist es vorteilhaft, die strengen Typkonventionen bei der Auswertung von Ausdrücken zu umgehen.
- Java und andere Sprachen bieten deshalb Möglichkeiten, Werte eines Typs in einen Wert eines anderen Typs umzuwandeln (engl: casting)
  - Dies ist aber nur für eine sehr eingeschränkte Menge von Paaren von Typen möglich. So ist z.B. eine Typumwandlung von **boolean** nach **int** (und umgekehrt) nicht möglich.
    - Dieses Codefragment funktioniert z.B. **nicht** in Java:

```
int n = 10;
while (n) System.out.println(n);
```

  - bei jeder Typumwandlung muss das Ergebnis (d.h. der Wert) klar sein.

Explizite und implizite Typumwandlung

- Wir betrachten folgende Variablen

```
int i = 1;
long l = 12345678901;
float f = 3.14f;
```

- Unterscheidung zwischen
  - **expliziter** Typumwandlung (z.B. von **float→int**)
    - Angabe des Typs in Klammern vor dem Ausdruck. Z. B. :

```
i = (int) f;
```

  - **impliziter** Typumwandlung (z.B. von **int→float** oder **long→float**)
    - Dies ist nur möglich, wenn der Zieltyp einen größeren Wertebereich besitzt, z.B.

```
f = i;
f = l;
```

Typausweitung ist immer erlaubt!

Explizite und implizite Typumwandlung

```
int i=1;
long l=12345678901;
float f = 3.14f;

// explizite Typumwandlung (z.B. von float→int)
i = (int) f;
// implizite Typumwandlung
// (z.B. von int→float oder long→float)
f = i;
f = l;
```

- Eine implizite Typumwandlung ist immer dann möglich, wenn der Zieltyp einen größeren Wertebereich besitzt.

Typanpassung bei Java

- Typausweitung ist immer erlaubt!

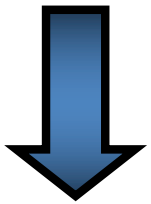
```
byte  vb = 127;
short vs = 255;
int   vi = 600000;
long  vl = 123456789;

vs = vb; vi = vs; // usw.
```

- Umgekehrt geht es nicht!

```
vb = vs; vi = vl; // usw.
```

- Was oft geht, ist eine explizite Anpassung mit einem sog. type cast.
  - Diese führt aber ggf. zu *Datenverlust*.
  - Zu jedem Datentyp T gibt es einen *type cast-Operator* (T).
  - Ob man ihn auch anwenden darf, hängt vom Kontext ab.
    - Generell gehen Anpassungen: *Zahlen → Zahlen*
    - Keinen Sinn macht z.B.: *String → Zahlen*

```
int   vix = 600000;
short vsx = (short) vix;
System.out.print(vsx);
```



```
10176
```

Live Vote

PIN: QKWU

https://ilias.uni-marburg.de/vote/QKWU

Beispiele für Java-Ausdrücke

- Seien `x`, `y`, `z` Variablen vom Typ int

Welchen Typ haben die folgenden Ausdrücke?

`x+2*(y+z)`	int
`x + 3.25 * (y + 0.1)`	double
`x + 5 == z - 28`	boolean