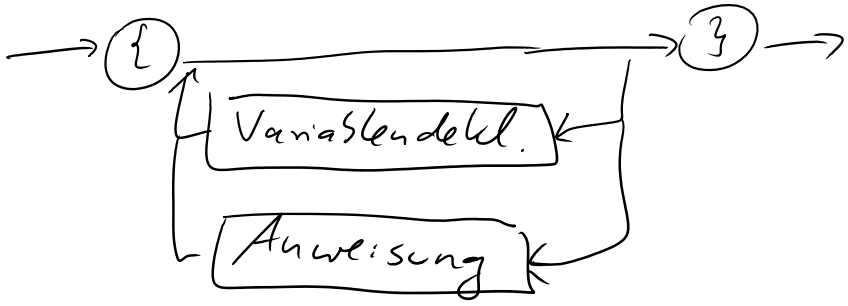
II.1.3 Anweisungen und Kontrollstrukturen

Dienstag, 17. Oktober 2017 09:00

- Methodenrumpf ist Block

Block



- Anweisungen enden mit ; oder mit }
- Anweisung: bewirkt Übergang v. einem Prog.-Zustand zum nächsten
- Prog-Zustand: Daten im Speicher (inkl. Werte der Variablen)
  + Befehlszähler (gibt an, welche Anweisung als nächstes ausgeführt werden muss)
- Kontrollfluss: Reihenfolge der auszuführenden Anweisungen

Datenfluss: Übergabe v. Daten v. einer Anweisung zur nächsten

Zuweisung

```
Var = Ausdruck;
```

Typ des Ausdrucks muss zum Typ der Variable passen (implizite Typkonversion möglich).

- Bsp: `X = x+10;`
  erhöht den Wert der Var. x um 10

analog: `x = x * 2;` etc.

Hierfür existieren Kurzschreibweisen:

`x += 10;` statt `x = x + 10;`

`x *= 2;` statt `x = x * 2;`

`x -= y;` statt `x = x - y;`

- Für `x = x + 1` bzw. `x = x - 1` existieren auch die Kurzschreibweisen `x++` `x--` `++x` `--x`

- Man kann in Java Anweisungen auch als Ausdruck benutzen.

```
x = 4;
y = 5;   <- Anweisung
x++;
:
```

Aber auch:

```
x = 4;
y = x++;
```

← Diese Anweisung lässt sich auch als Ausdruck verwenden. Liefert x als Erg. zurück.

Hinterher: x ist 5, y ist 4

```
x = 4;
y = ++x;
```

← Anweisung, die als Ausdruck den Wert x+1 zurückliefert

Hinterher: x ist 5, y ist 5

Verwendung von Anweisungen als Ausdruck führt meist zu schlechter Lesbarkeit.

=> x++ ++x nur als Anweisung benutzen, nicht als Ausdruck

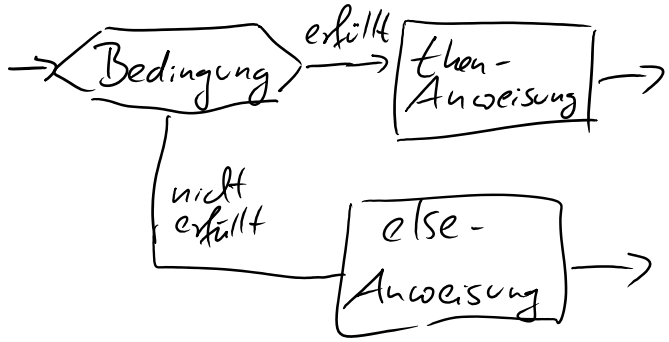
- Auch Zuweisung X = Ausdruck kann als Ausdruck benutzt werden: Anweisung, die auch als Ausdruck benutzt wird, mit Wert 5.

```
x = y = 5;
```

Solche Schreibweisen sind lesbar, aber keine anderen Anweisungen als Ausdrücke verwenden. (Methodenaufrufe dürfen natürlich als Ausdruck benutzt werden.)

if-Anweisung

Flussdiagramm zur Illustration der Arbeitsweise von "if":



- "else" kann fehlen
- "else" gehört zum innersten möglichen "if" bei geschachtel-

ten if-Anweisungen.

- Damit "else" zu äußerem "if" gehören kann: Gruppiere mehrere Anweisungen zu einem Block.
  { Anw$_1$; ...; VarDekl; ... Anw$_i$; }
  zählt als eine Anweisung.

Switch-Anweisung

- für große Fallunterscheidungen von int-Werten (oder char, String)
  anstelle von geschachtelten if-Anweisungen

int (oder char oder String)
switch (Ausdruck) { Es soll Fallunterscheidg. anhand des Werts dieses Ausdrucks durchgeführt werden

```
case Aus1 : Anw1
            break;
case Aus2 : Anw2
            break;
   :
default   : Anwdef }
```

Wenn Ausdruck den Wert Aus$_i$ hat, dann führe Anweisung Anw$_i$ aus.
Wenn keiner der vorigen Fälle zutrifft, dann führe Anw$_{def}$ aus.

- default-Fall darf fehlen

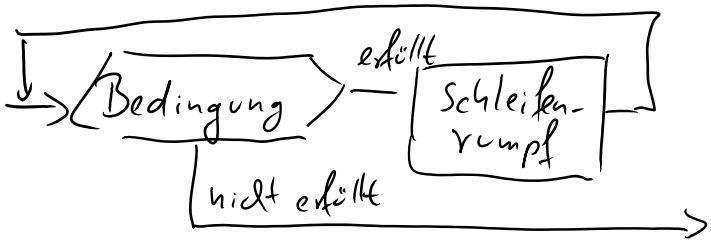
- default-Fall darf fehlen
- Man kann mehrere Fälle zusammenfassen: case 0: case 1: Anw ...
- "break" ist nötig, damit die Switch-Anweisung verlassen wird, sobald der richtige Fall gefunden wurde. Ansonsten würden alle weiteren Anweisungen im Switch-Rumpf ausgeführt.

⇒ Switch sollte immer mit break verwendet werden, sonst schlechter Stil.

Schleifen

Ziel: führe bestimmte Rechenschritte mehrfach aus

Flussdiagramm:



Gefahr der Nicht-Terminierung:

falls die Bedingung immer erfüllt ist, dann hält das Prog. nicht an.

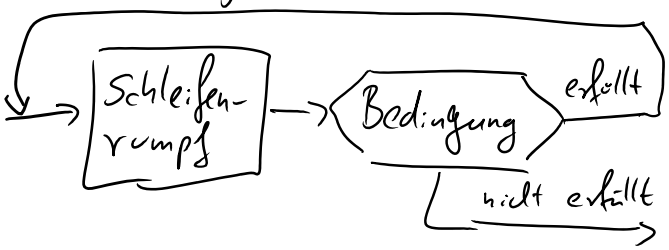
Prog. Prim: teiler durchläuft

Prog. Prim: teiler durchläuft alle Zahlen von 2 bis $\lceil \sqrt{n} \rceil$ und überprüft, ob n durch teiler teilbar ist, d.h. ob n % teiler == 0 ist.
(↑ modulo)

(Alg funktioniert bei Zahlen ≥ 2.)

do-Schleife

- Die 3 Schleifenarten in Java sind "äquivalent" (man kann jede while-Schleife auch als do- oder for-Schleife schreiben und umgekehrt). Aber: manchmal ist eine Schleife lesbarer als die andere.
- Flussdiagramm:



- Wurzel: berechne $\sqrt{x}$ für $x \geq 1$

Intervallschachtelung

[uG , oG]
↑ untere Grenze
↑ obere Grenze

Setze m auf die Mitte des

Setze m auf die Mitte des Intervalls: $m = \frac{uG + oG}{2}$.

Dann überprüfe, ob $\sqrt{x}$ in der oberen oder unteren Hälfte des Intervalls liegt.

$\sqrt{x} \in [uG, m]$, falls

$m > \sqrt{x}$ bzw.

$m^2 > x$

$\Rightarrow$ setze oG auf m

$\sqrt{x} \in [m, oG]$, falls

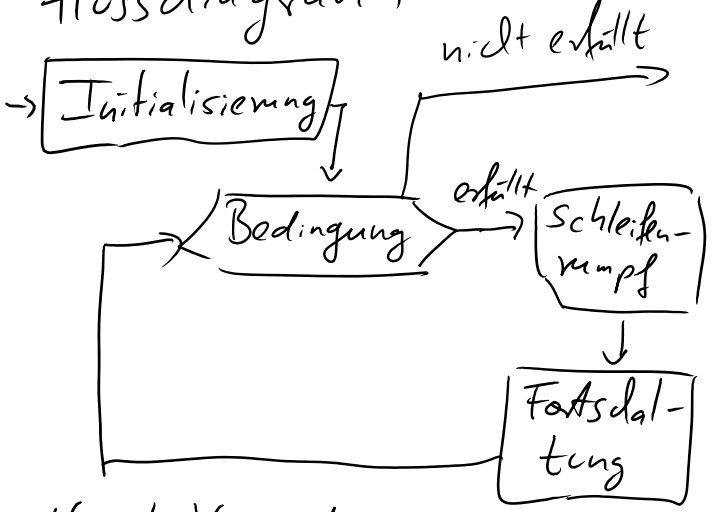
$m \leq \sqrt{x}$ bzw.

$m^2 \leq x$

$\Rightarrow$ setze uG auf m

for-Schleife

Flussdiagramm



- Haupt-Verwendung: Zählschleifen, in denen eine Variable bis zu einer bestb. Grenze erhöht oder erniedrigt wird.

z.B.

```
for (int i=0; i<10; i++) {
        ⋮
}
```

- Initialisierung kann entweder eine Variablen deklaration (z.B. int i=0, j=5) oder bel. viele Zuweisungen (z.B. i=0, j=5) enthalten.
  - Komma statt Semikolon
- Faltschaltung: bel. viele Zuweisungen durch Komma getrennt.

so sind i und j in der for-Schleife zugreifbar, aber nicht außerhalb

- Guter Stil: Initialisierung setzt Schleifen-Laufvar. auf Anfangswerte und Faltschaltung erhöht/erniedrigt sie. Alle anderen Anweisungen gehören in den Schleifenrumpf.
- Schleifenrümpfe können selbst wieder Schleifen enthalten (geschachtelte Schleifen).
- Bsp-Prog gist Folgendes aus:

i	j

i	j
1	1
2	1
2	2
3	1
3	2
3	3

Sprunganweisungen

≙ unbedingte Verzweigungsanweisung

Weitere unbed. Verzw.-Anw. ist

```
System.exit(n)
```

Dies beendet das Prog. Jede Fall n ≠ 0 signalisiert einen Fehler.

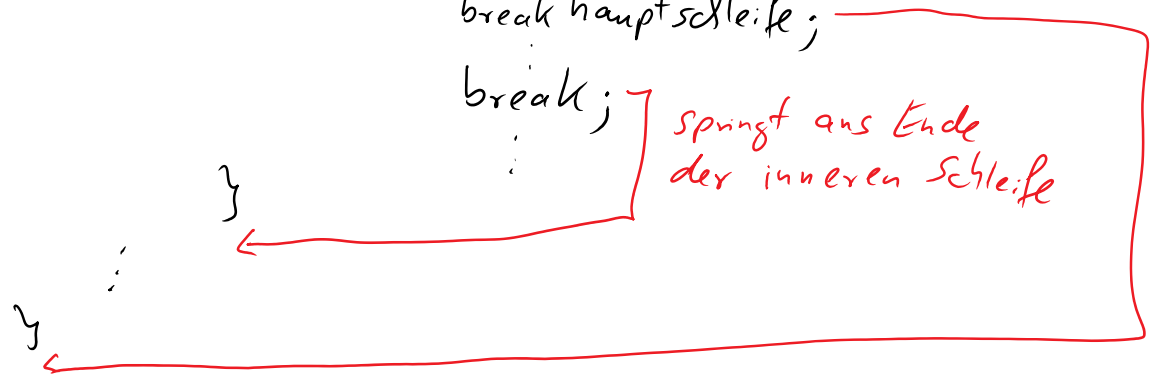
- Jede Anweisung kann mit einem Namen (Label) versehen werden.

meine Schleife: `while (...) {...}`

- break bricht die momentane Anweisung ab und springt zur nächsten Anweisung (für Schleifen und switch)

- Bei geschachtelten Schleifen wird die innerste Schleife abgebrochen:

```
hauptschleife: while (...) {
                   ⋮
               while (...) {
                   break hauptschleife;
                   break;
               }
                   ⋮
               }
```



- continue springt ans Ende des momentanen Schleifenrumpfs

```
hauptschl: while (...) {
               ⋮
           while (...) {
               continue hauptschl;
               continue;
               ⋮
           }
               ⋮
           }
```

- Bsp: Berechne alle "Freitage, der 13." eines Jahres. Eingabe sagt, welcher Wochentag

eines Jahres.
Eingabe sagt, welcher Wochentag
am 31.12. des Vorjahres war:

MO DI MI DO FR SA SO
1 6 7

- Sprunganweisungen sollten sehr zurückhaltend eingesetzt werden
  (können zu sehr unleserlichen Programmen führen).
- "goto considered harmful" (Dijkstra)