II.1.3 Anweisungen und Kontrollstrukturen

Montag, 15. Oktober 2018 11:00

Methodenrumpf ist Block:

Anweisungen enden mit ; oder }

Anweisung: Übergang von einem Prog-Zustand zum nächsten

Prog-Zustand: Daten im Speicher (insbes. Werte der Variablen) + Befehlszähler (welches ist die nächste auszuführende Anweisung?)

Kontrollfluss: Reihenfolge, in der Anweisungen abgearbeitet werden.

Datenfluss: Übergabe der Daten v. einer Anweisung an die nächste.

Zuweisung

```
x = 5;

x = x + 1;
```

↑ alter Wert der Var. x wird um 1 erhöht

Kurzschreibweisen:

```
x += 5;   für  x = x+5;
x -= 5;   für  x = x-5;
x *= 5;
x /= 5;        etc.
```

```
x++;
++x;      } für  x = x+1;
```

```
x--;
--x;      } für  x = x-1;
```

```
--x; }
```

Man kann manche Anweisungen auch als Ausdruck verwenden:

```
x = 1;
y = x++;
```

Diese Anweisung kann man auch als Ausdruck benutzen, hat den Wert x vor der Erhöhung.

⇒ x ist 2
y ist 1

```
x = 1;
y = ++x;
```

⇒ x ist 2
y ist 2

Verwirrend ⇒

Anweisungen wie x++, ++x sollten nicht als Ausdruck

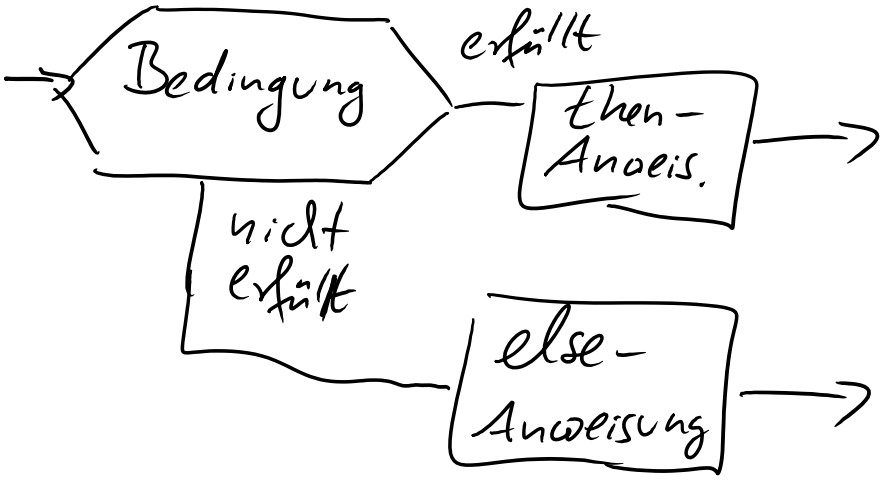
verwendet werden.

Ausnahme:

```
X = Y = 5;
```

↑ setzt sowohl x als auch y auf 5

Bedingte Anweisungen



- Falls man mehrere Anweisungen im then- oder else-Teil haben möchte: Block { ... }
- Bei geschachtelten if's:

"else" gehört immer zum

innersten möglichen if

- else-Anweisung darf fehlen

Switch-Anweisung

für große Fallunterscheidungen

switch (int-/char-/String-Ausdruck) {

case Aus_1 : Anw_1 break;

⋮

case Aus_n : Anw_n break;

default : Anw_{def} }

↑ default-Fall kann fehlen

Überprüft, ob Ausdruck den

Wert Aus_1 hat → führe Anw_1 aus

⋮

Aus_n hat → führe Anw_n aus

Sonst führe Anw_{def} aus,

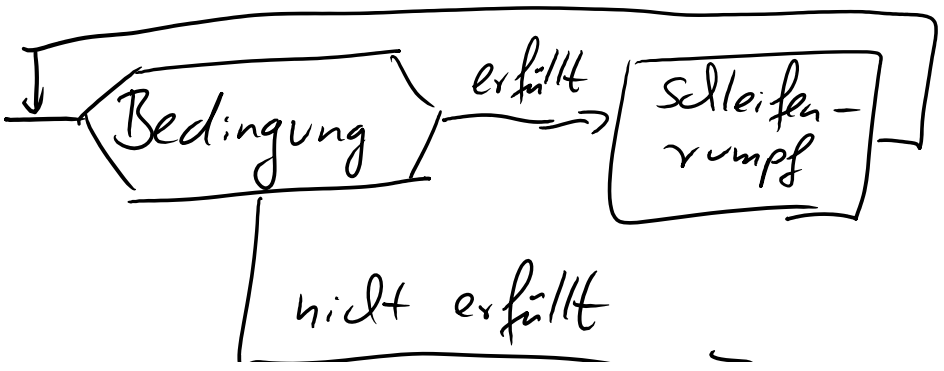
sofern vorhanden.

Mehrere cases können zusammengefasst werden.

Das "break" ist nötig, damit die switch-Anweisung nach dem jeweiligen Fall verlassen wird. (switch ohne break ist schlechter Stil.)

while-Schleife

Java hat 3 Arten von Schleifen: while, do, for. Sind alle "gleichmächtig", aber je nach Anwendung ist mal die eine, mal die andere lesbarer.



nicht erfüllt →

Gefahr der Nicht-Terminierung, wenn die Bedingung immer wieder zutrifft.

Bsp - Programm untersucht, ob n eine Primzahl ist, wobei: $n \geq 2$:

Durchlaufe alle Zahlen von

2 bis $\lfloor \sqrt{n} \rfloor$, $\lfloor 2.7 \rfloor = 2$

2 ↑ teiler

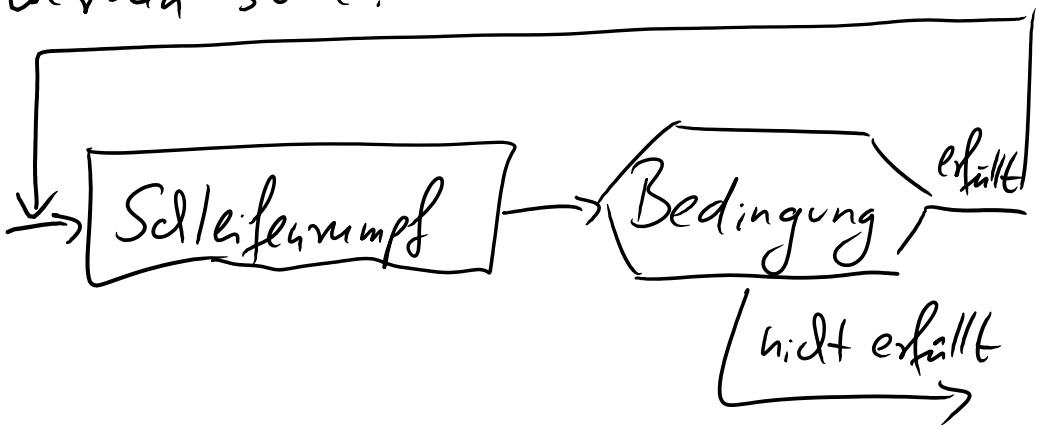
$\sqrt{n}$ ↑ wurzel

und überprüfe, ob n dadurch teilbar ist. (x % y == 0

genau dann, wenn → gdw. x ist durch y teilbar)

do-Schleife

Für Schleifen, bei denen der Schleifenrumpf auf jeden Fall mindestens einmal ausgeführt werden soll.



Bsp-Programm:

Intervallschachtelung zur Berechnung von $\sqrt{x}$ für $x \geq 1$.

Intervall $[ub, ob]$ mit

$\sqrt{x} \in [ub, ob]$

Am Anfang $[0, x]$.

ub ↗ 0, x ↖ ob

m wird auf die Mitte des Intervalls gesetzt, d.h. $m = \frac{ub+ob}{2}$.

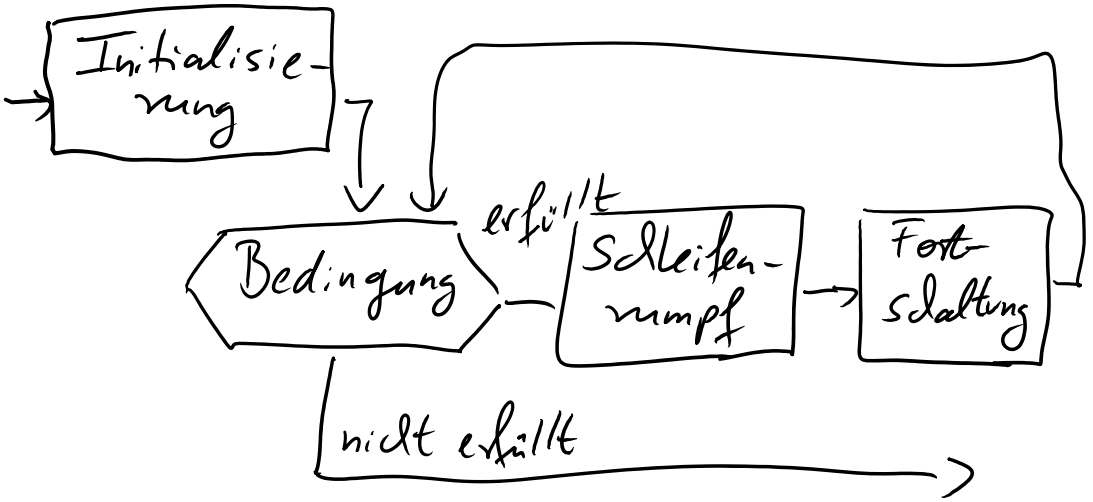
Anschließend wird überprüft, ob

$\sqrt{x} \in [ub, m)$ oder $\sqrt{x} \in [m, ob]$

d.h.: $\sqrt{x} < m$	d.h.: $\sqrt{x} \geq m$
bzw. $x < m^2$	bzw. $x \geq m^2$
$\Rightarrow$ setze $ob = m$	$\Rightarrow$ setze $ub = m$

Abbruch, wenn Intervallgröße $\leq \varepsilon$

for-Schleife



Beabsichtigte Verwendung v. for-Schleifen: Zählschleifen, bei denen eine Variable am Anfang auf einen bestimmten Wert gesetzt wird (Initialisierung) und in jedem Schleifen-

Steuerung) und in jedem Schleifendurchlauf um best. Wert erhöht oder erniedrigt wird (Fortschaltung), bis Bedingung nicht mehr zutrifft.

```
for (x=5; x>0; x--){...}
```

```
for (x=0; x<=10; x=x+2){...}
```

Für andere Arten von Schleifen sollte man while oder do verwenden.

Initialisierung:

- mehrere Zuweisungen, mit Komma getrennt

  $x=0, y=5$ oder

- eine Variablendeklaration

  int $x=0, y=5$

↑

Diese Variablen können dann nur in der for-Schleife benutzt werden.

Fortsetzung:
mehrere Zuweisungen möglich, mit Komma getrennt

Bsp mit geschachtelten for-Schleifen gibt Folgendes aus:

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

Sprunganweisungen

≙ unbedingte Verzweigungsan-

weisung.

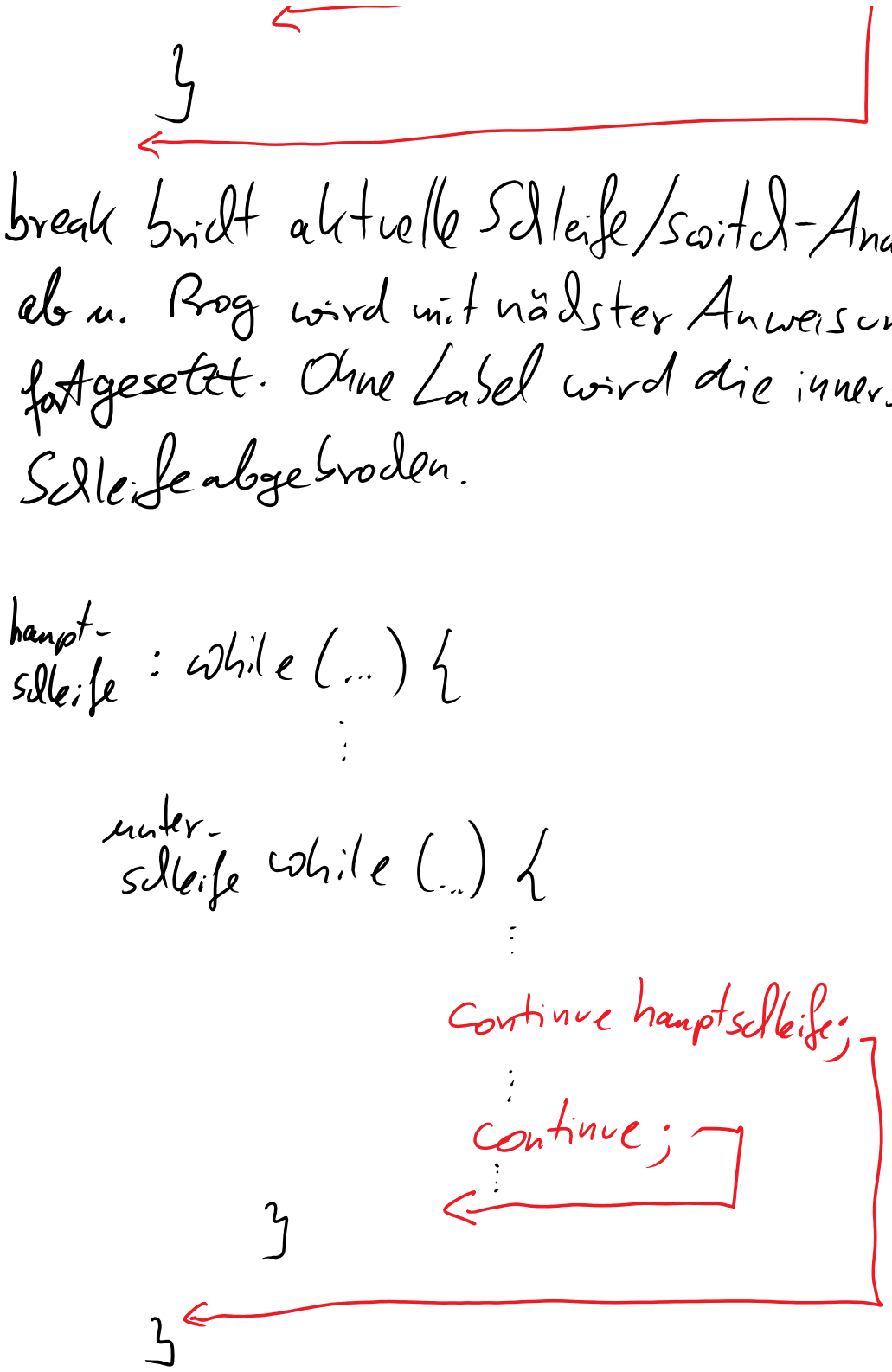
Weitere unsed. Verzweigungsanw:

System.exit(n)

bricht das Prog. ab, jedes n≠0 zeigt Fehler an.

Anweisungen können mit Labels versehen werden. Dadurch kann man bei geschachtelten Schleifen angeben, welche Schleife mit break bzw. continue verlassen werden soll.

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



continue springt ans Ende des Schleifenrumpfs der aktuellen Schleife.

Vorsicht: Sprünge können zu unlesbaren Programmen führen

⇒ break / continue zurückhaltend einsetzen.

"goto considered harmful"

Bsp: Bestimmt alle "Freitage, der 13." eines Jahres.

Eingabe: Letzter Wochentag des Vorjahres.

MO	DI	MI	DO	FR	SA	SO
1	2	3	4	5	6	7

31.12.2017 war ein Sonntag