II.1.3 Anweisungen und Kontrollstrukturen

Mittwoch, 26. Oktober 2016 09:00

Anweisung: überführt Programmzustand in den nächsten —— " ——

Progr.-Zustand: Daten im Speicher + Befehlszähler

Kontrollfluss: Reihenfolge der ausgeführten Anweisungen (≙ Befehlszähler)

Datenfluss: Übergabe von Daten von einer Anweisung zur nächsten (≙ Daten im Speicher)

Java: ausführbare Programme ≙ Block eines Methodenrumpfs

Block: Folge von Anweisungen + Variablendeklarationen

Methodenaufruf ✓

Zuweisung

`double x = 10;` ← Var. bekommt Wert eines kompatiblen Typs zugewiesen

`x = x + 1;` ← x wird um 1 erhöht

Kurzschreibweise: `x += 1` `x *= 2` etc

$\hat{=}$ $x = x+1$ $\qquad$ $\hat{=}$ $x = x*2$

Weitere Kurzschreibweise

$x++$ $\quad$ $++x$ $\quad$ für $\quad$ $x = x+1$

$x--$ $\quad$ $--x$ $\quad$ für $\quad$ $x = x-1$

`x++;` ist eine Anweisung (`x = x+1;`)
Aber man kann manche Anweisung leider auch als Ausdruck benutzen

```
int a=2, b=3;
a *= b++;
```

Anweisung b++ wird gleichzeitig als Ausdruck verwendet.
"Wert" von b++ ist b
"Wert" von ++b ist b+1

Danach ist $a=6$, $b=4$

```
int a=2, b=3;
a *= ++b;
```

Danach ist $a = 8$, $b=4$.

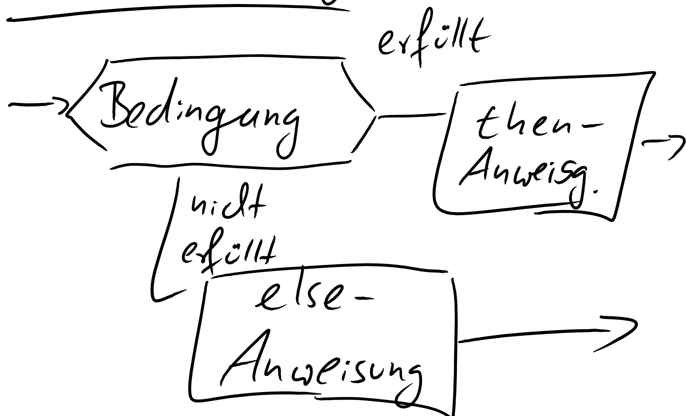
Die Verwendung von Anweisungen als Ausdrücke führt meist zu schlecht lesbaren Programmen ⇒ sollte man vermeiden!
Wir verwenden "a++" nur als Anweisung, nicht als Ausdruck.

Einzige Ausnahme: $a = b = 2;$

entspricht $a = (b=2);$

Anweisung, die auch als Ausdruck mit Wert 2 verwendet werden kann

if-Anweisung



Ein Block { ... }
zählt ebenfalls als Anweisung.

if-Anweisungen können geschachtelt werden.
Falls es einen "else"-Zweig gibt, so gehört er immer zum innersten möglichen "if".

switch-Anweisung

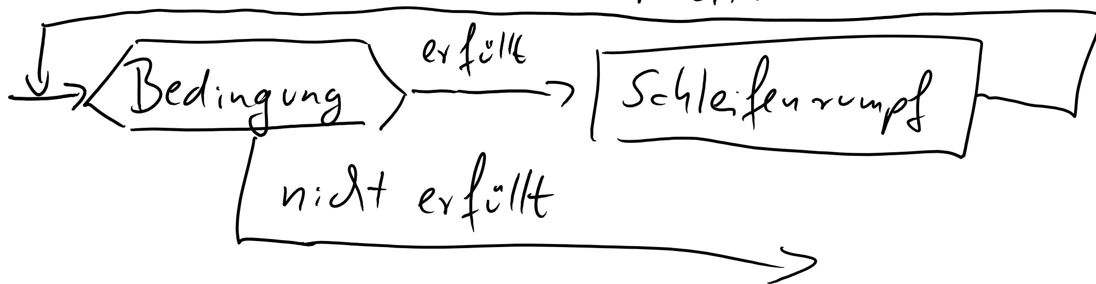
Für Fallunterscheidungen mit vielen Fällen, die alle vom Wert einer Variablen abhängen (anstelle von geschachtelten if-An-

weisungen).

break dient dazu, die switch-Anweisung zu verlassen, sobald die Anweisung des jeweiligen Falls ausgeführt wurde.

Schleifen

- dienen dazu, bestimmte Rechenschritte immer wieder zu wiederholen.
- 3 Schleifenarten (while, do, for): sind prinzipiell äquivalent, aber versch. Schleifen aus Gründen der Lesbarkeit.



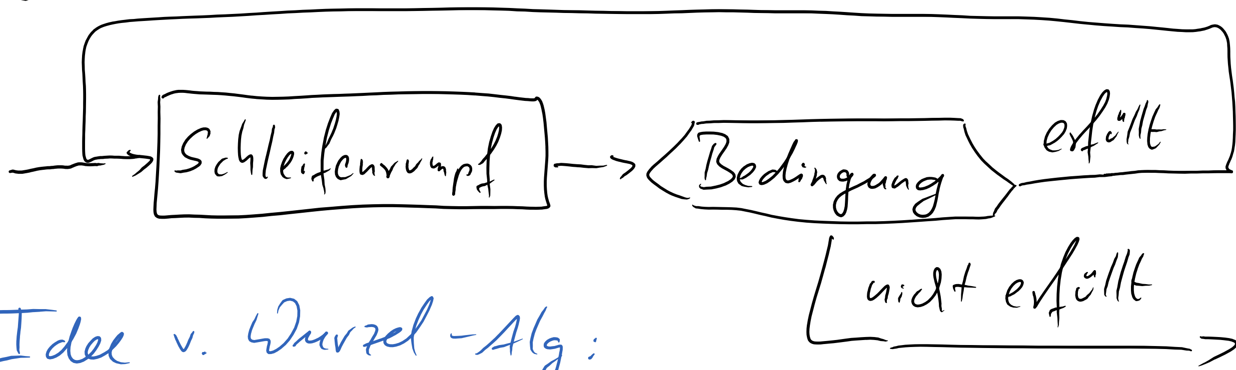
Bsp: Algorithmus zur Überprüfung, ob n eine Primzahl ist

Idee: Verwende eine Variable "teiler", die von 2 bis $\sqrt{n}$ läuft.
Überprüfe, ob diese die Zahl n teilt.

Klasse Math stammt aus Paket java.lang, das automatisch importiert wird.

do-Schleife: verwendet, wenn der Schleifenrumpf

auf jeden Fall mindestens 1 mal ausgeführt werden soll.



Idee v. Wurzel-Alg:

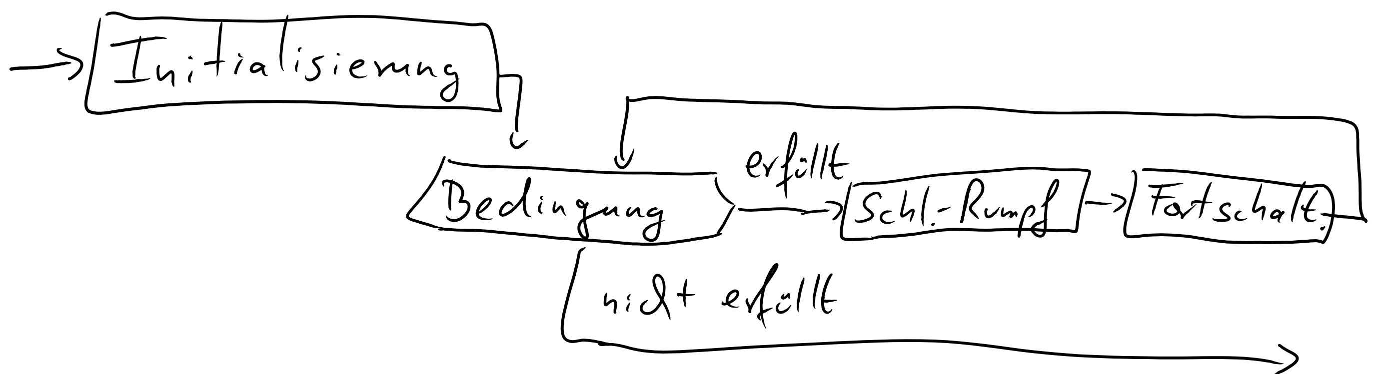
Zur Berechng. von $\sqrt{x}$ verwende Intervallschachtelung $[uG, oG]$, $uG = 0$, $oG = x$ am Anfang

Danach wird das Intervall halbiert.

Mitte des Intervalls ist $m = \frac{uG + oG}{2}$.

Überprüfe, ob $\sqrt{x}$ in $[uG, m]$ liegt $\leftarrow x < m^2$
oder in $[m, oG]$. $\leftarrow x \geq m^2$

for-Schleife



for-Schleifen eignen sich besonders zur Implementierung von "Zählschleifen", in denen ein Parameter initialisiert

wird und in jedem Schleifendurchlauf erhöht oder erniedrigt wird, bis ein best. Wert erreicht ist.

```
for (  i=1, j=7 ; i <= 5 ;  i++, j--){ move(); }
```

In der Initialisierg. deklarierte Variablen "gelten" in der gesamten Schleife (auch in geschachtelten inneren Schleifen).

Ausgabe im Bsp:

1 1,
2 1,
2 2,
3 1,
3 2,
3 3

Initialisierung + Fortschaltg.
von for-Schleifen sollten nur den Startwert u. Erhöhen/ Erniedrigen der Zählvar. betreffen. Alles andere sollte in den Schleifenrumpf.

Sprunganweisungen:

- break + continue : unbedingte Sprunganweisungen
- System.exit(n) : — " ________________, die das Prog. beendet

jede Zahl
$n \neq 0$ deutet Fehler an

Jede Anweisung kann einen Namen bekommen (ein sogenanntes <u>Label</u>):

```
Hauptschleife: while (....) { .... }
```

break: bricht die aktuelle Anweisung ab, fahre mit der nächsten Anweisung fort.

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

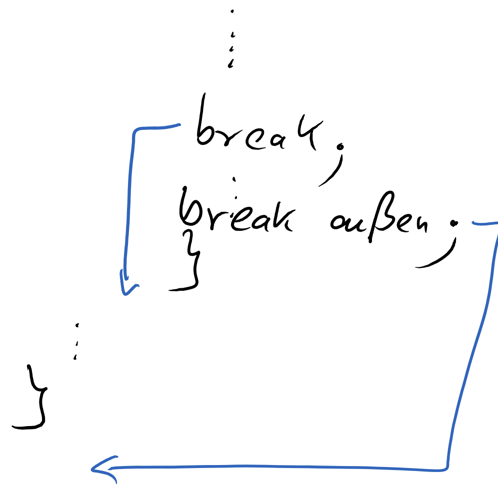
springt hinter die while-Schleife

break funktioniert nicht nur in Schleifen, sondern auch in switch-Anweisungen.

Bei geschachtelten Schleifen bricht break immer die innerste Schleife ab:

```
außen: while ( ) {
    :
    innen: while ( ) {
        :
```

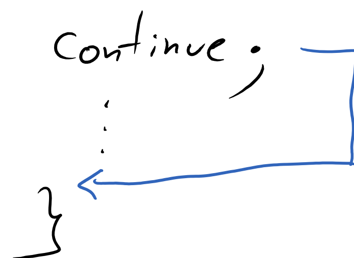
innen : while ( ) {



Mit Hilfe von labels kann man auch die äußere Schleife abbrechen.

Continue in einer Schleife bricht die Schleife nicht ab, sondern springt ans Ende des Schleifenrumpfs:

while (...) {



Bsp-Prog berechnet alle "Freitage, der 13." in einem Jahr.

Wochentage sind mit Zahlen codiert:

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

---

break und continue sollten sparsam benutzt werden, um Lesbarkeit des Codes zu erhalten.

Glücklicherweise enthält Java kein Konstrukt für beliebige Sprünge ("goto considered harmful")