Kapitel 3: Anweisungen

- Bedingte Anweisungen (Verzweigungen)
- Wiederholte Anweisungen (Schleifen)
  - Abweisschleife
  - Durchlaufschleife
  - Zählschleife

Bedingte Anweisungen

- Bedingte Anweisung
  if (x != 0) x = 1/x; falls x != 0, dann x = 1/x
- Verzweigungen
  if (a > b) max = a; "then-Zweig"
  else max = b; "else-Zweig"
- Syntax (EBNF)
  IfStatement = "**if**" "**(**" *Expression* "**)**" *Statement* ["**else**" *Statement*].
- Semantik (Ablauf)
  Werte die Bedingung aus.
  Falls die Bedingung gilt, führe die then-Anweisung aus.
  Falls Bedingung nicht gilt, führe die else-Anweisung aus (falls vorhanden).

Komplexe Bedingungen

- Bedingungen
  - als Bedingung kann jeder Ausdruck vom Typ *boolean* eingesetzt werden.
  - zusammengesetzte Bedingungen sind möglich ("&&", "||", "!").
  - Bezeichner für Bedingungen sollten Eigenschaftswörter sein.
    boolean *equal*, *empty*;
- Regeln von DeMorgan (Negation komplexer Bedingungen)
  Für zwei Ausdrücke p und q vom Typ *boolean* gilt:
  !(p && q) ≡ !p || !q
  !(p || q) ≡ !p && !q

Beispiel:
!(x==0 && y==0) ≡ x!=0 || y!=0

Nachweis über Wahrheitstabelle

p	q	p&&q	!(p&&q)	!p	!q	!p || !q
true	true	true	false	false	false	false
true	false	false	true	false	true	true
false	true	false	true	true	false	true
false	false	false	true	true	true	true

Sequentielle Auswertung (Kurzschlussauswertung)

Beispiel: **if** (y != 0 && x / y > 0) …

mit strikter Auswertung

y != 0	x / y > 0	y != 0 && x / y > 0
true	true	true
true	false	false
false	*Fehlerabbruch*	

mit sequentieller Auswertung

y != 0	x / y > 0	y != 0 && x / y > 0
true	true	true
true	false	false
false	(wird nicht ausgewertet)	false

Sequentielle Auswertung basiert auf folgendem Prinzip

p && q wenn !p, dann false, sonst q
p || q wenn p, dann true, sonst q

Anweisungen und Blöcke

- Problem
  - Oft enthalten then-Zweig und/oder else-Zweig mehrere Anweisungen.
  - Syntaktisch ist nur eine einzelne Anweisung erlaubt.
- Lösung: Blöcke
  - Block fasst mehrere Anweisungen zu einer einzigen zusammen.
  - EBNF: *Block* = "**{**" {*Statement*} "**}**".
- Beispiel

```
if (x < 0) {
  negNumbers++;
  System.out.println (-x);
} else {  // x >= 0
  posNumbers++;
  System.out.println (x);
}
```


Zuordnung von **else** (*dangling else*)

- Beispiel:
  geschachtelte if-Anweisungen

```
if (a > b)
    if (a != 0) max = a;
else
    max = b;
```

- Zu welchem **if** gehört das **else**?
  Mehrdeutigkeit
- Regel:
  else gehört immer zum letzten **if**
- Abhilfe:
  Mit Klammern { } Block bilden

```
if (a > b) {
    if (a != 0) max = a;
} else  // a <= b
    max = b;
```


Einrückungen

- Grundregel
  - Einrückungen dienen der Lesbarkeit und sollen die (tatsächliche) Programmstruktur verdeutlichen.
  - Einrückungen werden vom Übersetzer ignoriert.
  - Als Einrückungen eignen sich 2-3 Leerzeichen oder 1 Tabulatorschritt.
- Stilregeln für if-else
  - Die nachgeordneten Anweisungen der Verzweigung werden eingerückt.
  - **else** steht unter dem zugehörigen **if**.
  - Abschließende Klammer "}" steht ebenfalls in einer Linie mit **if** (und **else**).
  - Bei komplexen Verzweigungen können mehrere Einrückungen nötig sein.
  - **else** sollte man mit Zusicherung versehen.

Auswahl aus Alternativen (**switch**)

Beispiel: Berechnung der Tage pro Monat

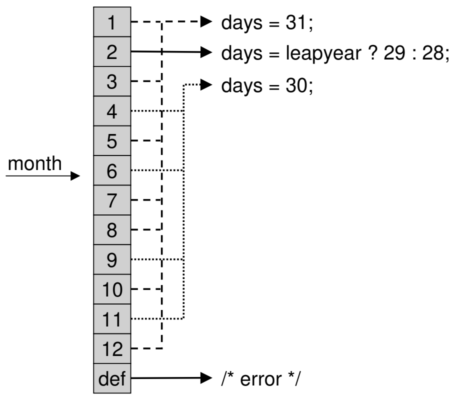
Dazu: **switch**-Anweisung

```
switch (month) {
  case 1: case 3: case 5: case 7: case 8: case 10: case 12:
    days = 31; break;
  case 4: case 6: case 9: case 11:
    days = 30; break;
  case 2:
    days = leapyear ? 29 : 28; break;
  default:
    /* Fehlerausgabe */
}
```

Ablauf der Mehrfachverzweigung

- Ablauf des **switch**
  - werte **switch**-Ausdruck aus
  - bestimme passende Marke (**case** oder **default**, falls vorhanden)
  - führe ggf. Anweisungen aus
- **break** nach jedem **case**
  - beendet **switch**-Anweisung
  - ohne **break**: Anweisungen des nächsten **case** *Fehlerquelle!*
- Regeln für **case**-Marken
  - nur Literale mit passendem Typ (z.B. auch Zeichen möglich)
  - müssen verschieden sein

Sprungtabelle für **switch**



Vergleich: **switch** vs. **if**-Kaskade

- Kaskadiertes **if**-**else**

  if (m=1 || m=3 || m=5 || m=7 || m=8 || m=10 || m=12) days = 31;
  else if (m=4 || m=6 || m=9 || m=11) days = 30;
  else if (m=2) days = leapyear ? 29 : 28;
  else /* Fehlerausgabe */

- Vergleich
  - **switch** ist schneller, da mit Sprungtabelle gearbeitet wird.
  - bei weit auseinanderliegen Marken ist Tabelle sehr groß und enthält viele Lücken (Platzbedarf!)

```
switch (n) {
  case 1: …; break;
  case 10000: …; break;
}
```


Abweisschleife (**while**)

- Struktur
  - Aufbau: **while** (*Bedingung*) *Anweisung*;
  - EBNF: *WhileStatement* = "**while**" "**(**" *Expr* "**)**" *Statement*.
  - Ablauf: Schleife wird durchlaufen, solange die Bedingung gilt.
  - Wichtig: Die Bedingung wird vor dem Schleifendurchlauf geprüft.
- Beispiel: Summe der Zahlen von 1 bis *n*

```
int sum = 0;
int i = 1;               Initialisierung
while (i <= n) {         while (noch nicht fertig) {
  sum = sum + i;           Verarbeitung …
  i = i + 1;               Weiterschalten
}                        }
```


Zusicherungen bei Schleifen

- Zusicherungen

  Aus Schleifenbedingung kann man bestimmte Aussagen ableiten.

```
int i = 0;
while (i <= n) {
  /* i <= n */
  …
  i++;
} /* i > n */  bzw. genauer: /* i == n+1 */
```

- Invarianten
  - Eigenschaften, die zu Beginn und am Ende jeden Durchlaufs gelten.
  - Mittel, um die Korrektheit von Schleifen/Programmen formal zu beweisen.
  - Beispiel: /* sum == Summe(1..i-1) */
  - Vollständige Beweise sind mehrstufig, d.h. zu zeigen ist:
    - Die Invariante gilt zu Beginn der Schleife
    - Die Invariante drückt am Ende das gewünschte Ergebnis aus
    - Die Invariante gilt am Ende jedes beliebigen Durchlaufs
    - Terminierung: Die Schleife wird nur endlich oft durchlaufen

Durchlaufschleife (**do-while**)

- Struktur
  - Aufbau: **do** *Anweisung* **while** (*Bedingung*);
  - EBNF: *DoStatement* = "**do**" *Statement* "**while**" "**(**" *Expr* "**)**" "**;**".
  - Ablauf: Schleifenrumpf wird durchlaufen, bevor Bedingung geprüft wird.

- Beispiel: "Umkehren" einer Dezimalzahl *n*

```
do {
  System.out.print (n%10);
  n = n / 10;
} while (n > 0);
```

n	Ausgabe
123	3
12	2
1	1
0	

Abweisschleife würde für Beispiel *n* = 0 nicht funktionieren.

Zählschleife (**for**)

- Beispiel: Summe der Zahlen von 1 bis n

```
int sum = 0;
for (int i = 1; i <= n; i++) {
   sum = sum + i;
}
```

- Charakterisierung
  - Meist verwendet, wenn Anzahl der Durchläufe schon vorher bekannt ist.
  - Verwendung einer „Laufvariablen“, im Beispiel: i
  - Deklaration der Laufvariablen im Initialisierungsteil

Allgemeine Schleife (**for**)

- Allgemeine Form

  for (*init; cond; incr*) *S*;

init:	Initialisierungsanweisung
cond:	Schleifenbedingung
incr:	Inkrementierungsanweisung
S:	Schleifenrumpf

- Ablauf
  - *init* wird vor Betreten der Schleife ausgeführt.
  - *init* darf Deklarationen enthalten (z.B. für Laufvariable).
  - *cond* wird vor jedem Durchlauf der Schleife geprüft.
  - *incr* wird am Ende eines jeden Durchlaufs ausgeführt.
  - Bei jedem Durchlauf wird der Schleifenrumpf (*S*) ausgeführt.
  - für komplizierte Fälle besser die **while**-Schleife verwenden.

Beispiele

Gegeben sei eine positive ganze Zahl n. Gesucht ist eine Tabelle mit n Zeilen und n Spalten, in der das Element in Zeile i und Spalte j den Wert i * j enthält.

- Geschachtelte Schleifen: Multiplikationstabelle

```
public static void main (String[] arg) {
   int n = Integer.parseInt (arg[0]);
   for (int row = 1; row <= n; row++) {
     for (int col = 1; col <= n; col++)
       System.out.print (row*col + '\t');
     System.out.println();
   }
}
```

row	col	Ausgabezeilen
1	1 2 3	1 2 3
2	1 2 3	2 4 6
3	1 2 3	3 6 9

- Absteigen: Gerade Zahlen von 10 bis 2

```
for (int i = 10; i > 0; i = i - 2)
   System.out.println (i);
```

Abbruch von Schleifen

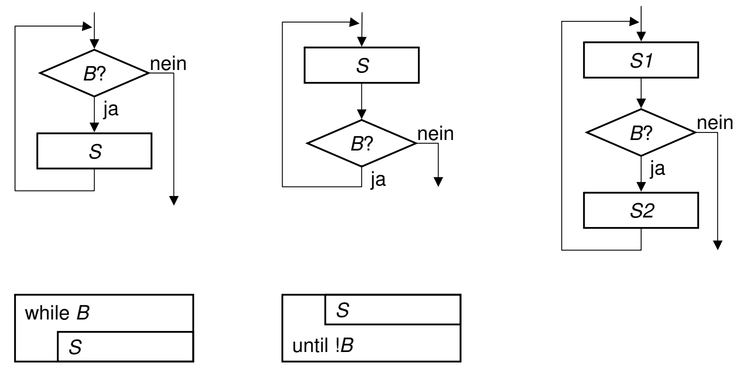
- Abbruch durch **break**
  - **break** bricht die Schleife ab und springt an das Ende.
  - **break** möglichst vermeiden, da Programmlogik verkompliziert wird.
  - besser ist die Kontrolle in der Schleifenbedingung.

```
                                  boolean finished = false;
while (B) {                       while (B && !finished) {
  S1;                               S1;
  if (...) break;                   if (...) finished = true;
  S2;                               else S2;        /* hier aufpassen! */
}                                 }
```

- Wann ist Schleifenabbruch sinnvoll?
  - Abbruch wegen Fehlern
  - bei Aussprüngen an verschiedenen Stellen
  - bei echten Endlosschleifen (z.B. Ampelschaltung) keine „Algorithmen"!
    for (;;) *S*;

Vergleich der Schleifenarten

Abweisschleife (while, for) | Durchlaufschleife (do) | allgemeine Schleife (for (;;) mit break)



Transformation der Schleifenarten

- Abweisende Schleife

```
                          if (B)
while (B)                   do
  S;                          S;
                            while (B);
```

- Durchlaufschleife

```
do                        S;
  S;                      while (B)
while (B);                  S;
```

- Zählschleife

```
                          init;
for (init; B; incr)       while (B) {
    S;                      S;
                            incr;
                          }
```

- Allgemeine Schleife

```
for (;;) {                S1;
  S1;                     while (!B) {
  if (B) break;             S2;
  S2;                       S1;
}                         }
```