6. Iteration
(Schleifenanweisungen)

Java-Beispiel:

TemperatureTable.java

Schwerpunkte

- While-Anweisung: "abweisende" Schleife
- Do-while-Anweisung: "nichtabweisende" Schleife
- For-Anweisung: zählergesteuerte Schleife
- Klassifikation von Anweisungen
- Strukturierte Programmierung

Syntax

While-Anweisung

EBNF:

`while` **`(`** Bedingung **`)`**
Anweisung

Unterschiede zu Pascal?

Solange die 'Bedingung' gilt, wiederhole man die 'Anweisung'

Anwendung:
Anweisung wird ggf. n i c h t bearbeitet

Pascal

`while` **`(`** Bedingung **`)`**
`do` Anweisung

Do-while-Anweisung

EBNF:

```
do
   Anweisung
while ( Bedingung ) ;
```

Syntax nicht konsequent: ';'

Solange die 'Bedingung' gilt, wiederhole man die 'Anweisung', wobei die Bedingung n a c h Ausführung der Anweisung getestet wird

Anwendung:
Anweisung wird mindestens e i n m a l bearbeitet

Do-while-Anweisung: mit while-Anweisung ausgedrückt

```
do
   Anweisung
while ( Bedingung ) ;
```

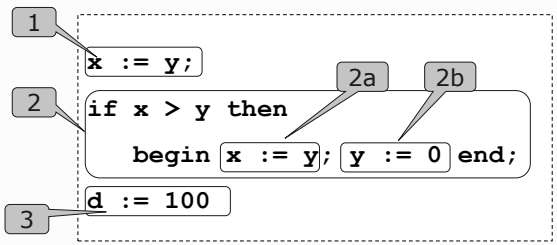
dasselbe wie:

```
Anweisung
while ( Bedingung )
   Anweisung
```

Do-while wird nicht benötigt - erspart aber Schreibaufwand

Rolle des ';` in der Syntax: Trennung oder Abschluss von Anweisungen ?

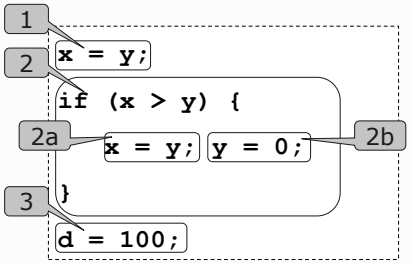
- Pascal, Modula-2, Ada, ... : Trennung von Anweisungen



Rolle des ';'

- C, C++, Java:
  - Abschluss von Anweisungen (syntaktisch Teil der Anweisung)
  - aber: mit vielen Ausnahmen (ohne ';' : while, if, ...)

→ Unsauberkeit in Sprachdefinition (kein einheitliches Prinzip: Fehlerquelle)



Beispiel: Fakultät

n! = 1 * 2 * . . . * n

Wichtig: Anfangswerte

Abbruchbedingung

```
int n, x = 1, fakultaet = 1;
   ... // n einlesen
while (x <= n) {
   fakultaet = fakultaet * x;
   x = x + 1;
}
```

z. B. für n = 4:

- ***x*** durchläuft die Werte:
  1, 2, 3, 4, 5
  (bei x = 5 gilt nicht mehr: x <= n)
- dabei durchläuft *fakultaet* die Werte:
  1, 1, 2, 6, 24

For-Schleife

Auswahl einer Wiederholanweisung

Wiederholung
(einer Anweisung)

Solange eine Bedingung gilt

Feste Anzahl von Wiederholungen

mindestens e i n Mal

auch: 0 Mal

Java

```
do A
while (B);
```

```
while (B)
   A
```

```
for (I;T;U)
   A
```

Pascal

```
REPEAT A
UNTIL NOT B;
```

```
WHILE B
DO A
```

```
FOR x := 1 TO n
DO A
```


For-Schleife

EBNF:

```
for (Initialisierung ; Test ; Update)
      Anweisung
```

Ausdrücke
(insb. Zuweisungen)

dasselbe wie:

```
Initialisierung ;
while ( Test )    {
   Anweisung ;
   Update ;
}
```

For-Anweisung wird nicht benötigt
- Bessere Lesbarkeit in vielen Fällen

Beispiel: Fakulät mit 'for'

```
int n, x, fakultaet = 1;
   ... // n einlesen
for (x = 1; x <= n ; x = x + 1)
   fakultaet = fakultaet * x;
```

Kurzform: x++
analog: x-- wie x = x - 1

dasselbe:

```
x = 1;
while (x <= n) {
   fakultaet = fakultaet * x;
   x = x + 1;
}
```

Anwendung: **A n z a h l der Wiederholungen ist bekannt (hier: n) !**

For-Anweisung in Java (C, C++): aus sprachtheoretischer Sicht ein Unglück

Was man will:

- Man kennt Anzahl der Wiederholungen: n
- Eine Laufvariable i läuft von einem Anfangswert (z.B. 1) bis zum bekannten Endwert (n)

Pascal:

```
fak := 1;
FOR x := 1 TO n DO
   fak := fak * x;
```

Java, C, C++:

```
fak = 1;
for (x = 1; x <= n ; x = x + 1)
   fak = fak * x;
```


For-Anweisung in Java (C, C++): unübersichtliche Programme möglich

```
for (a = 1 ; b * c == 0 ; k = k + 1)
      x = y;
```

So nicht!

→ kein Bezug der Laufvariablen a zur Abbruchbedingung und Update

```
for ( ; ; ) {readNumber(); ... }
```

entspricht:

```
while (true){
   readNumber();...
}
```


Zyklen: weiteres Beispiel

- Tabelle:
  Umrechnung Celsius → Fahrenheit

Grad C	Grad F
-10.0	14.0
-9.0	15.8
-8.0	17.6
...	
10.0	50.0

→ Beispielprogramm mit while-Anweisung

TemperaturTable.java

```
class TemperatureTable {
 // Tabelle mit C/F Temperaturen
   public static void main (String[] args) {
       final double
          LOW_TEMP = -10.0,
          HIGH_TEMP = 10.0;

       double
          cent, // Grad Celsius
          fahr; // Grad Fahrenheit

       System.out.println("\tGrad C\t\tGrad F");
       cent = LOW_TEMP;
       while (cent <= HIGH_TEMP) {
          fahr = (9.0/5.0) * cent + 32.0; // C -> F
          System.out.println("\t" + cent + "\t\t" + fahr);
          cent = cent + 1.0;
       }
   }
}
```

Tabulator

cent++

Statt „while“ besser „for“ – Wie?


```
class TemperatureTable {
 // Tabelle mit C/F Temperaturen
   public static void main (String[] args) {
       final double
          LOW_TEMP = -10.0,
          HIGH_TEMP = 10.0;

       double
          cent, // Grad Celsius
          fahr; // Grad Fahrenheit

       System.out.println("\tGrad C\t\tGrad F");
       for(cent = LOW_TEMP; cent <= HIGH_TEMP; cent++) {
          fahr = (9.0/5.0) * cent + 32.0; // C -> F
          System.out.println("\t" + cent + "\t\t" + fahr);
       }
   }
}
```


Das 1 ½-Zyklus-Problem

Break-Anweisung

Break-Anweisung: Austritt aus Zyklen mitten im Zykluskörper

```
// z.B. Compiler: Symbole ermitteln und verarbeiten …
while (true) {
   ... Einlesen // u.a. führende Leerzeichen überlesen
   if (Keyboard.eof())
      break;
   ... verarbeite Eingabe ...
}
```

Strukturierte Anweisung wird verlassen: switch, while, if …

Ungünstig:
Abbruchbedingung nicht direkt in while-Bedingung sichtbar
(wenn möglich: vermeiden)

Aber – oft sinnvolle Lösung

Break-Anweisung: das " 1 ½ - Zyklus-Problem"

```
while (true) {
   ... einlesen ...
   if (Keyboard.eof())
      break;
   ... verarbeite Eingabe ...
}
```

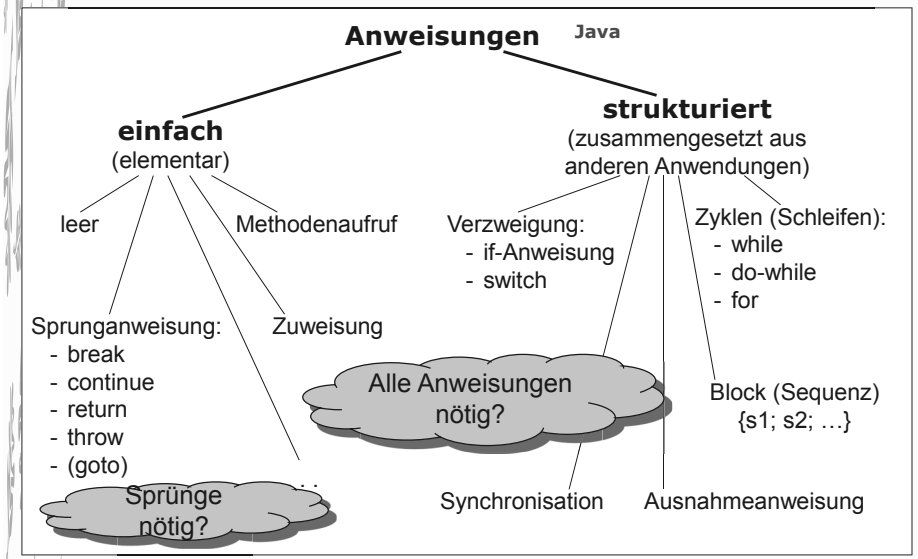
➔ auch ohne break: aber doppelter Code

```
... einlesen ...
while (! Keyboard.eof())
   ... verarbeite Eingabe ...
   ... einlesen ...
}
```


Strukturierte Programmierung

Bekannter Begriff in der Programmierung seit ca. 1975

Klassifikation von Anweisungen



Strukturierte Programmierung

früher:
(Fortran)

```
1     IF (a+b) 10, 20, 30
10    ...
      ...
      GO TO 40
20    ...
      GO TO 40
30    ...
40
```

(a+b)<0

Marke (markierte Anweisung)
(Fälle: a+b < 0 ... =0 ... >0)

Strukturierte Programmierung:

- Man kommt ohne Sprünge aus
- Ausreichend zur Algorithmenbildung:
  - Zuweisung
  - sequentielle Ausführung (a1; a2; ...)
  - Verzweigung (Langform von 'if')
  - Wiederholung (while)