3.4 Ablaufsteuerung (Kontrollstrukturen)

Sequenz (Folge von Anweisungen)

Der Kaffee-Algorithmus beinhaltet einfache Schritte, die einer nach dem anderen auszuführen sind. Wir sagen, ein solcher Algorithmus ist eine **Folge (Sequenz)** von Schritten, was bedeutet:

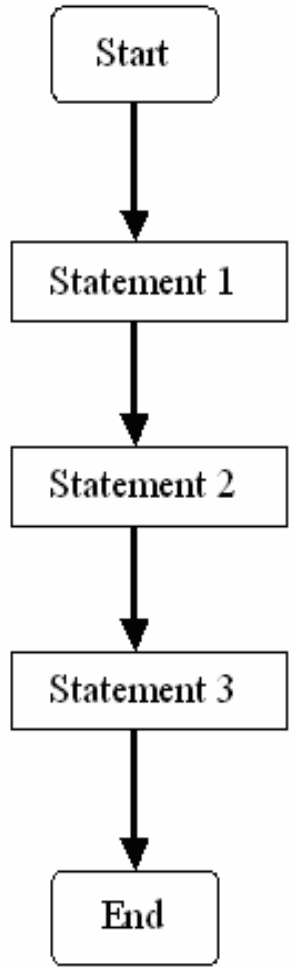
1. Zu einem Zeitpunkt wird nur ein Schritt ausgeführt. Schritte werden nie parallel ausgeführt.
2. Jeder Schritt wird genau einmal ausgeführt: keiner wird wiederholt, keiner wird ausgelassen.
3. Die Reihenfolge, in der die Schritte ausgeführt werden, ist die gleiche, in der sie niedergeschrieben sind.
4. Mit der Beendigung des letzten Schrittes endet der gesamte Algorithmus.

Sequenz (Folge von Anweisungen)

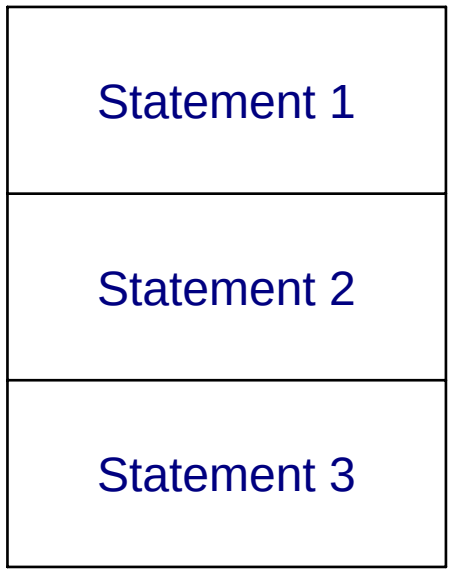
Die Ausführung eines solchen Algorithmus ist sehr starr. Im Beispiel des Algorithmus "Koche Kaffee":

- Was passiert, wenn manche Kaffeetrinker Milch und/oder Zucker mögen? (Selektion)
- Lassen sich Teile das Algorithmus auch zum Teekochen verwenden? (Wiederverwendung)
- Kann das Kaffeepulver in die Tasse gegeben werden, während das Wasser zum Kochen gebracht wird? (Parallelisierung)

Flussdiagramm der Sequenz



Struktogramm



Selektion

Bedingte Ausführung von Anweisungen

a) Einfache Form

Falls Bedingung

dann Anweisung

Englisch

if condition

then statement(s)

Bedingte Ausführung von Anweisungen

b) Bedingte Anweisung mit Alternative (allgemeine Form):

Falls Bedingung

dann Anweisung 1

sonst Anweisung 2

Englisch

if condition

then statement 1

else statement 2

Anmerkung:

Die einfache Form ist ein Spezialfall der allgemeinen Form, bei der Anweisung 2 die leere Anweisung ist ("tue nichts").

Mehrfachauswahl

Falls

Bedingung 1 **dann** Anweisung 1

Bedingung 2 **dann** Anweisung 2

.

.

.

andernfalls

Anweisung n + 1

Die Bedingungen 1 bis n müssen sich gegenseitig ausschließen; d. h. es dürfen nicht zwei Bedingungen gleichzeitig erfüllt sein.

Beispiele für bedingte Anweisungen

Beispiel 1: Einfache bedingte Anweisung

(2.1.1.) Nehme Kaffeeglas aus dem Fach

(2.1.2.) **Falls** Kaffeeglas leer ist

dann nehme neues Kaffeeglas aus dem Schrank

(2.1.3.) Entferne Deckel vom Kaffeeglas

Beispiel 2: Bedingte Anweisung mit Alternative

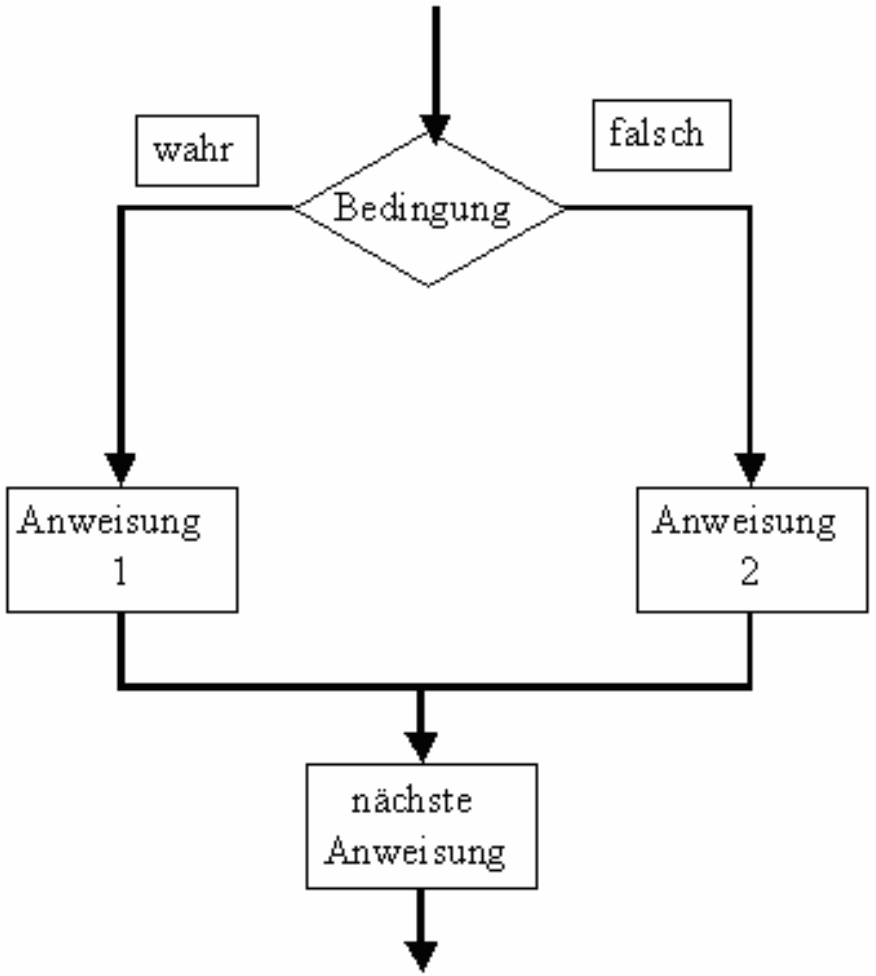
max (x,y)

Falls x > y

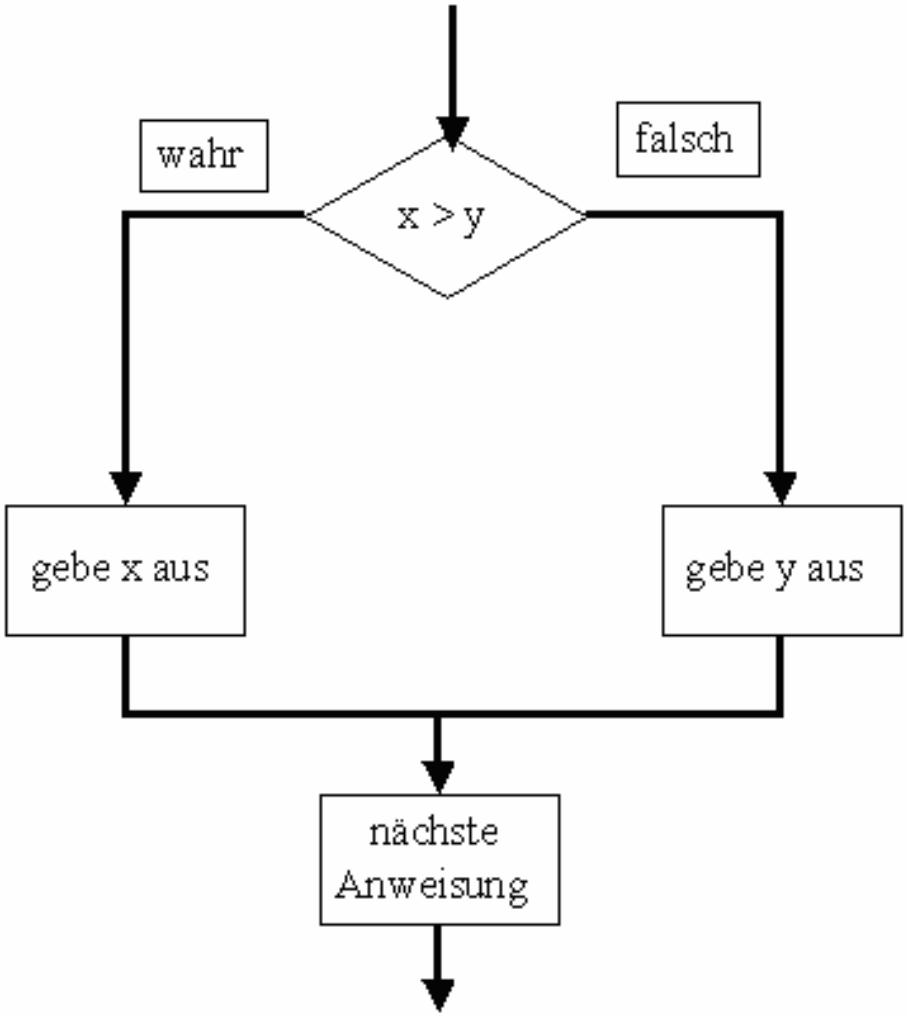
dann gib x aus

sonst gib y aus

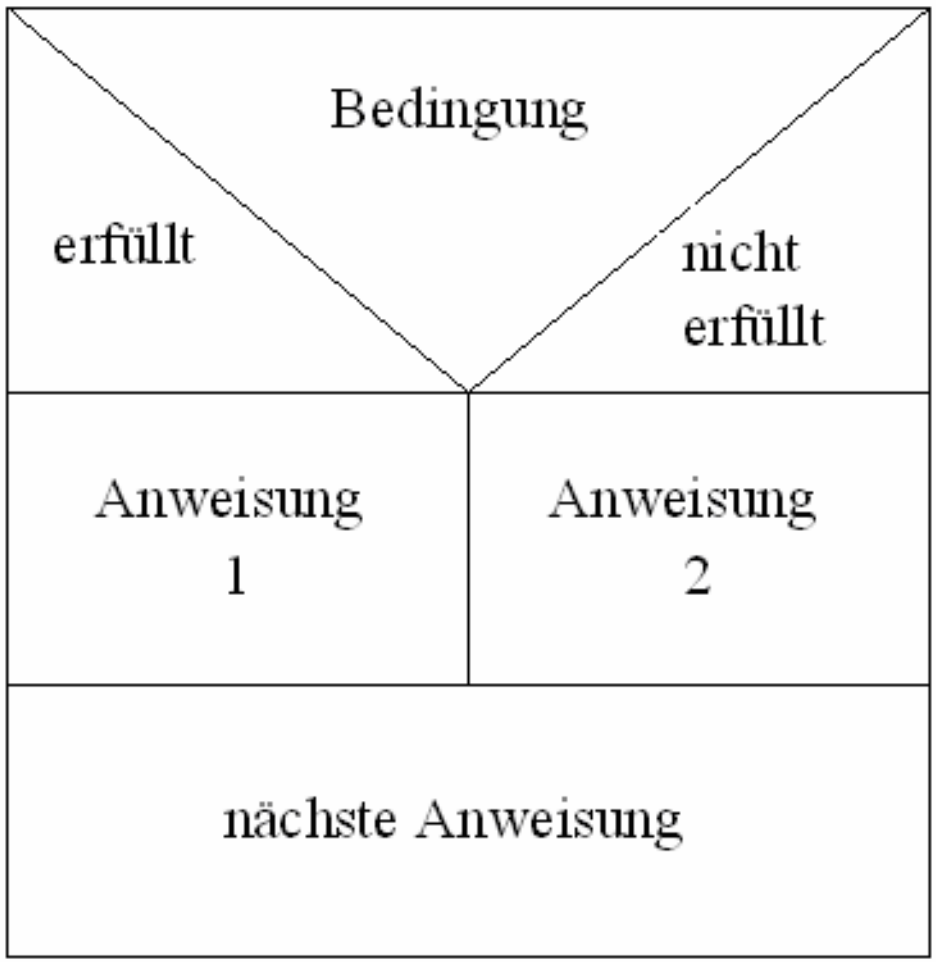
Flussdiagramm der bedingten Anweisung



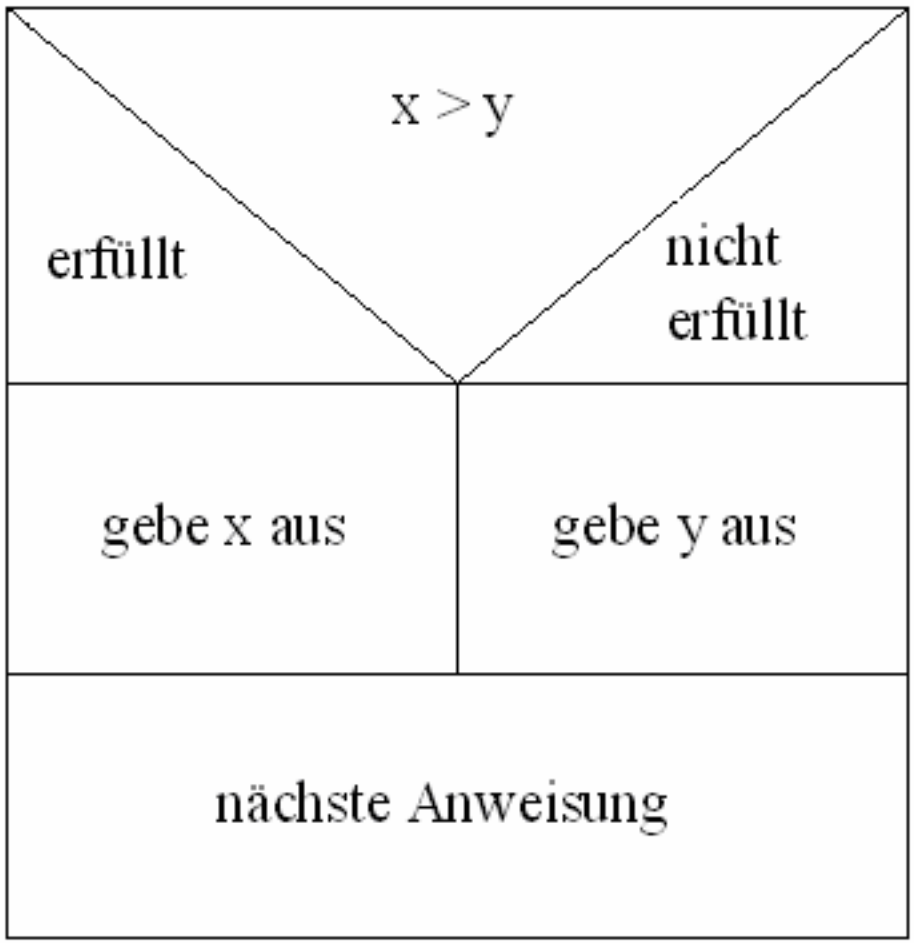
Flussdiagramm zur Berechnung des Maximums



Struktogramm der bedingten Anweisung



Struktogramm zur Berechnung des Maximums



Beispiel (1)

Pseudocode

falls die Note des Studenten größer oder gleich 60 ist
dann gebe „passed“ aus.

Java

```
if ( grade >= 60 ) {
    System.out.println( “passed“ );
}
```

Beispiel (2)

Pseudocode

falls die Note des Studenten ist größer oder gleich 60

dann

gebe „passed“ aus.

sonst

gebe „failed“ aus.

Java (erste Formulierung)

```
if ( grade >= 60 ) {
    System.out.println( “passed“ );
}  else {
  System.out.println( “failed“ );
}
```

Alternative Formulierung (1)

Java (zweite Formulierung)

```
System.out.println( grade >= 60 ? “passed“ :
   “failed“ );
```

Hinweis

Man benutzt die zweite Formulierung nur, wenn unbedingt notwendig, da sie nicht einfach zu lesen ist.

Alternative Formulierung (2)

Beispiel 3: geschachtelte bedingte Anweisung

max (x, y, z)

Falls x > y

dann falls x > z

dann wähle x

sonst wähle z

sonst falls y > z

dann wähle y

sonst wähle z

Die Einrückung im Text macht im Pseudocode deutlich, zu welcher Bedingung ein **sonst** gehört. Ohne Einrückung wäre die Zuordnung nicht immer eindeutig!

Geschachtelte falls/dann-Strukturen (1)

Pseudocode

falls die Note des Studenten ist größer oder gleich 90

dann drucke “A”

sonst

    falls die Note des Studenten ist größer oder gleich 80

    dann

        drucke “B“

Fortsetzung ---->

Geschachtelte falls/dann-Strukturen (2)

sonst

falls die Note des Studenten ist größer oder gleich 70

dann drucke “C“

sonst

falls die Note des Studenten ist größer oder gleich 60

dann

drucke “D”

sonst

drucke “F“

Geschachteltes `if/else` in Java

```
if ( grade >= 90 ) {
  System.out.println( “A“ );
} else {
  if ( grade >= 80 ) {
    System.out.println( “B“ );
  } else {
    if ( grade >= 70 ) {
      System.out.println( “C“ );
    } else {
      if ( grade >= 60 ) {
        System.out.println( “D“ );
      } else {
        System.out.println( “F“ );
      }}}}
```

Probleme mit geschachtelten Strukturen

```
if ( x > 5 )
  if ( y > 5 )
    System.out.println( “x and y are > 5” );
else
  System.out.println( “x is <= 5” );
```

FEHLER!!

Wie Sie bereits wissen, bringt Java ein **else** stets mit dem letzten vorherigen **if** in Verbindung. Daher heißt es richtig:

```
if ( x > 5 )
  if ( y > 5 )
    System.out.println( “x and y are > 5” );
  else
    System.out.println( “y is <= 5” );
```

Bemerkungen

- Geschachtelte bedingte Ausdrücke sind schwierig zu testen: Tiefe n bedeutet, dass 2n verschiedene Kombinationen zu testen sind!
- Solche Strukturen sind auch schwer zu verstehen.

Mehrfachauswahl (1)

Allgemeine Form

falls

Variable = Bedingung 1: Anweisung 1

Variable = Bedingung 2: Anweisung 2

…

andernfalls Anweisung n+1

oder

falls Variable **gleich**

Bedingung 1: Anweisung 1

Bedingung 2: Anweisung 2

…

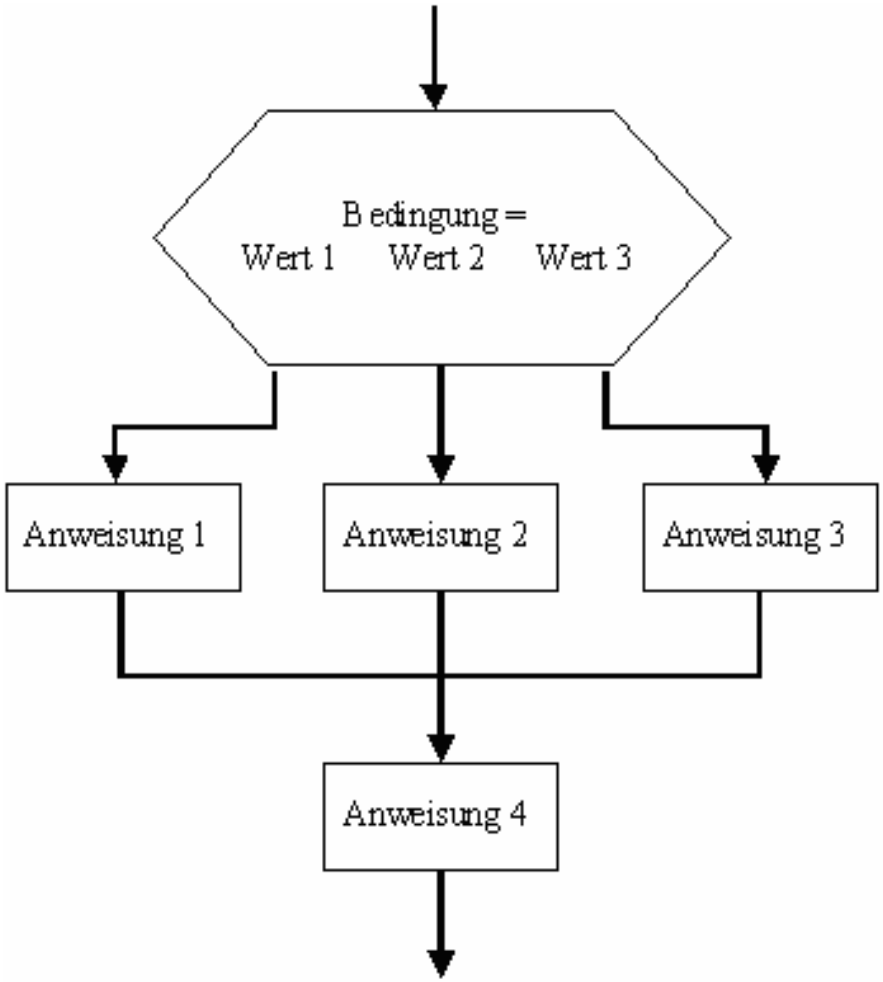
andernfalls Anweisung n+1

Mehrfachauswahl (2)

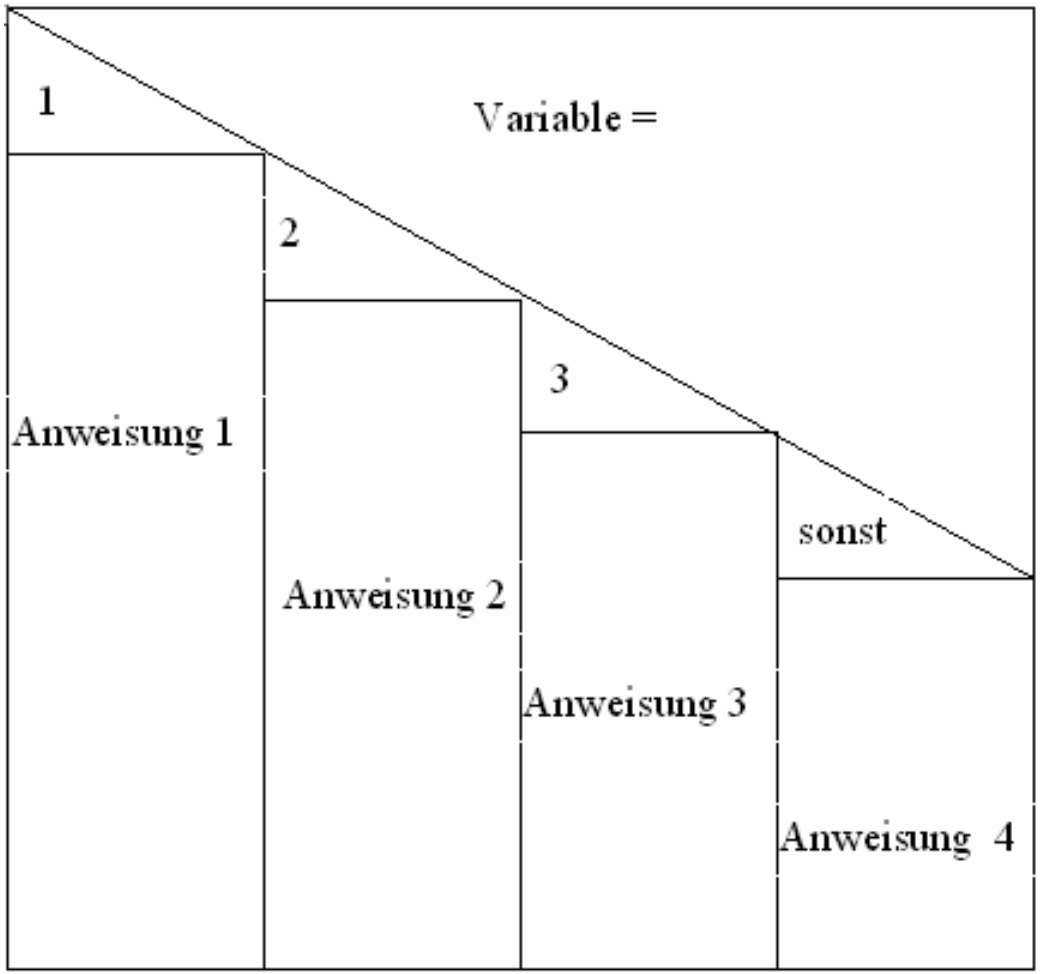
falls
    Münze = Fünfer: addiere 5 auf Summe
    Münze = Zehner: addiere 10 auf Summe
    Münze = Fünfziger: addiere 50 auf Summe
andernfalls:
    gebe Münze zurück

Oder in anderer Schreibweise:
falls Münze **gleich**
    Fünfer: addiere 5 auf Summe
    Zehner: addiere 10 auf Summe
    Fünfziger: addiere 50 auf Summe
andernfalls:
    gebe Münze zurück

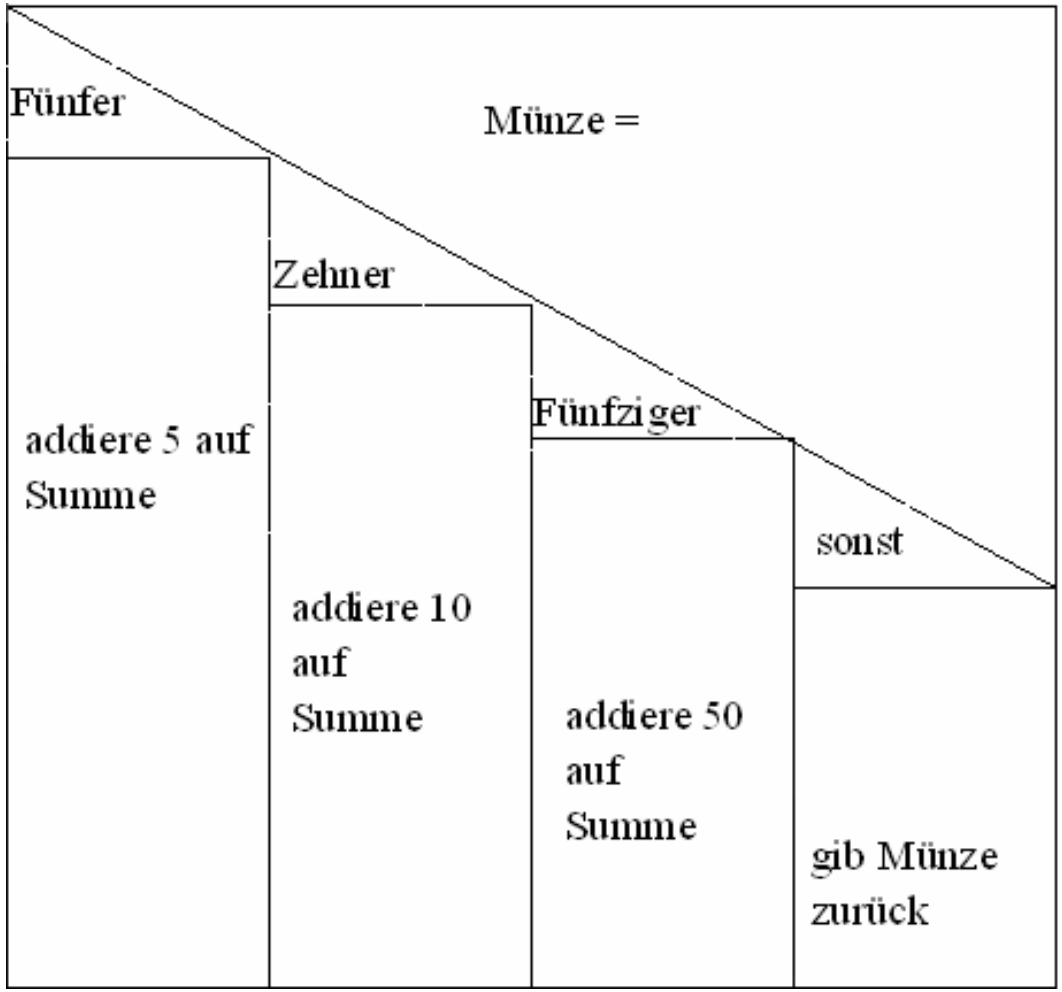
Flussdiagramm für die Mehrfachauswahl



Struktogramm für die Mehrfachauswahl



Struktogramm für das Münzbeispiel



Mehrfachauswahl in Java

```
switch (Ausdruck) {
  case Wert1: Anweisung 1; break;
  case Wert2: Anweisung 2; break;
      ...
  default: Anweisung n+1;  break;
}
```

Java-Code für das Münzbeispiel

```
switch (coin) {
  case 5: sum += 5; break;
  case 10: sum += 10; break;
  case 50: sum += 50; break;
  default: giveBack(coin);
}
  ...
```

Beispiel: Java-Code für die Notenzuweisung (1)

```
switch ( grade ) {
  case ‘A’: case ‘a’:
    ++aCount;
    break;
  case ‘B’: case ‘b’:
    ++bCount;
    break;
  case ‘C’: case ‘c’:
    ++cCount;
    break;
```

Beispiel: Java-Code für die Notenzuweisung (2)

```
  case 'D': case 'd':
    ++dCount;
    break;
  case 'F': case 'f':
    ++fCount;
    break;
  default: showStatus( "Incorrrect grade. Try again." );
    break;
}
```

Wiederholungsanweisung (Iteration, Schleife)

Wiederholte Ausführung einer Anweisung (oder einer Folge von Anweisungen), bis eine Endbedingung erfüllt ist.

wiederhole

    Anweisung(en)

bis Bedingung

Beispiel für eine Schleife (1)

Beispiel 1

Lies den gesuchten Namen ein.

Hole den ersten Namen aus der Liste.

wiederhole

falls Name der Gesuchte ist **dann**

gib die Adresse aus

sonst

hole den nächsten Namen aus der Liste

bis der gesuchte Name gefunden oder die Namensliste erschöpft ist

Beispiel für eine Schleife (2)

Beispiel 2

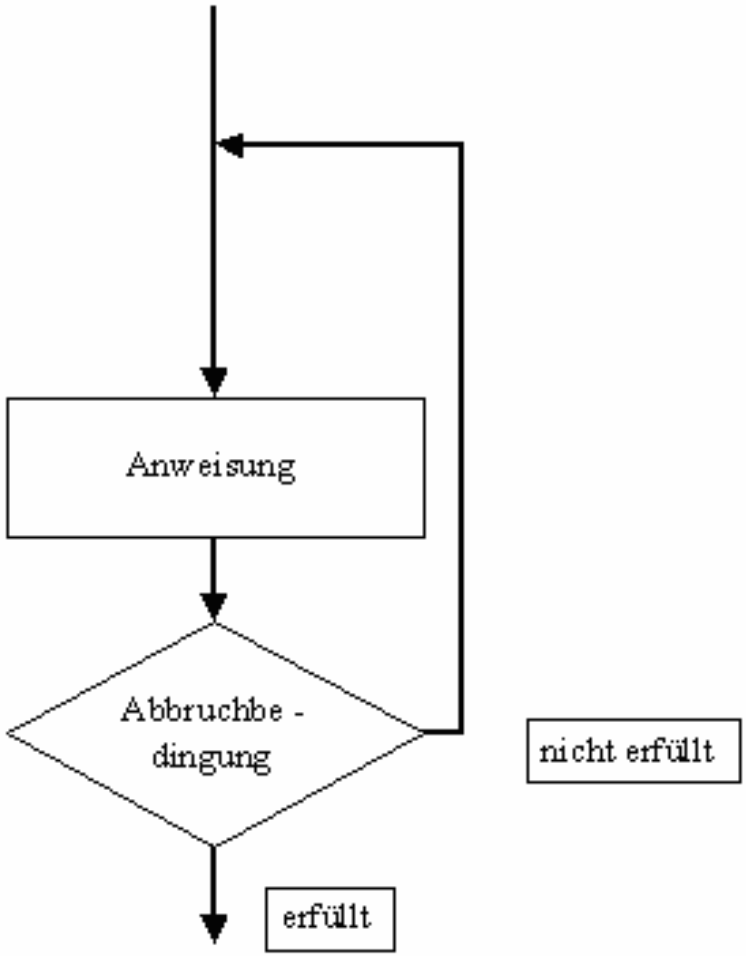
Summe = 0

wiederhole

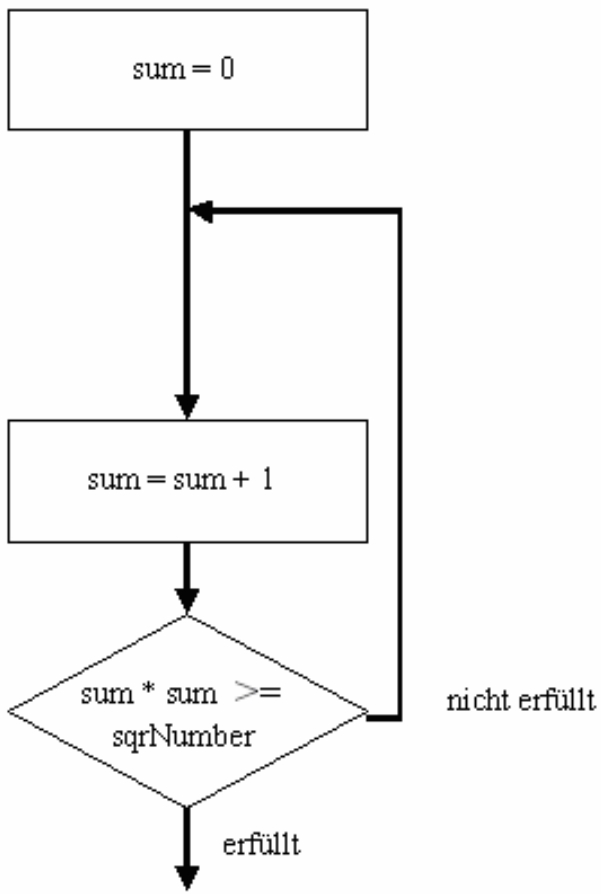
    addiere 1 auf Summe

bis Summe*Summe < sqrNumber

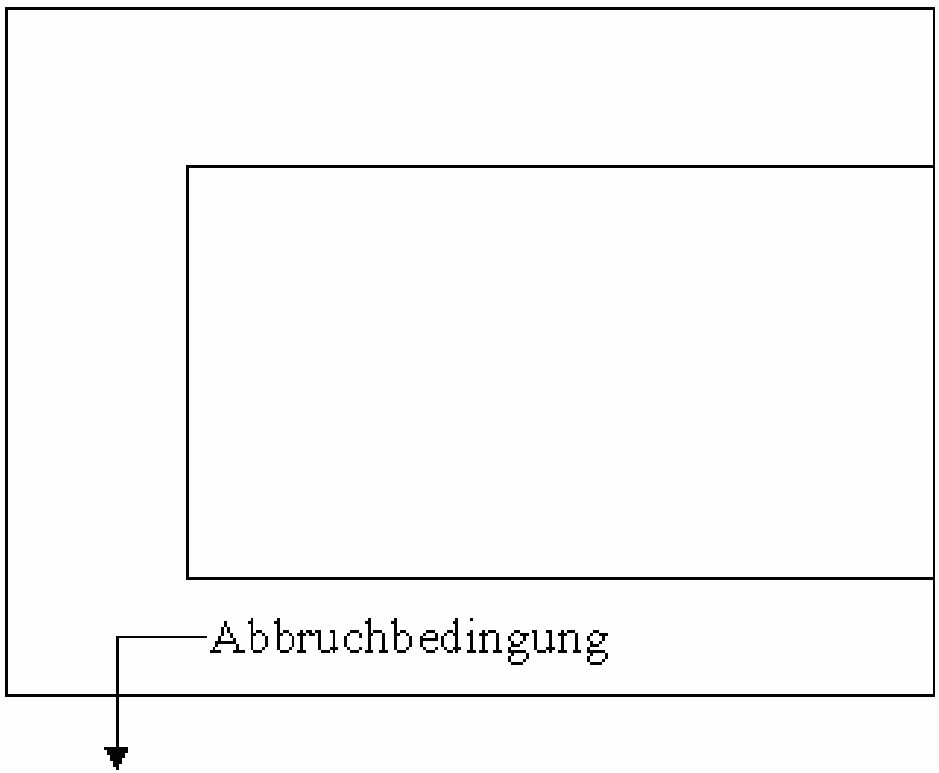
Flussdiagramm



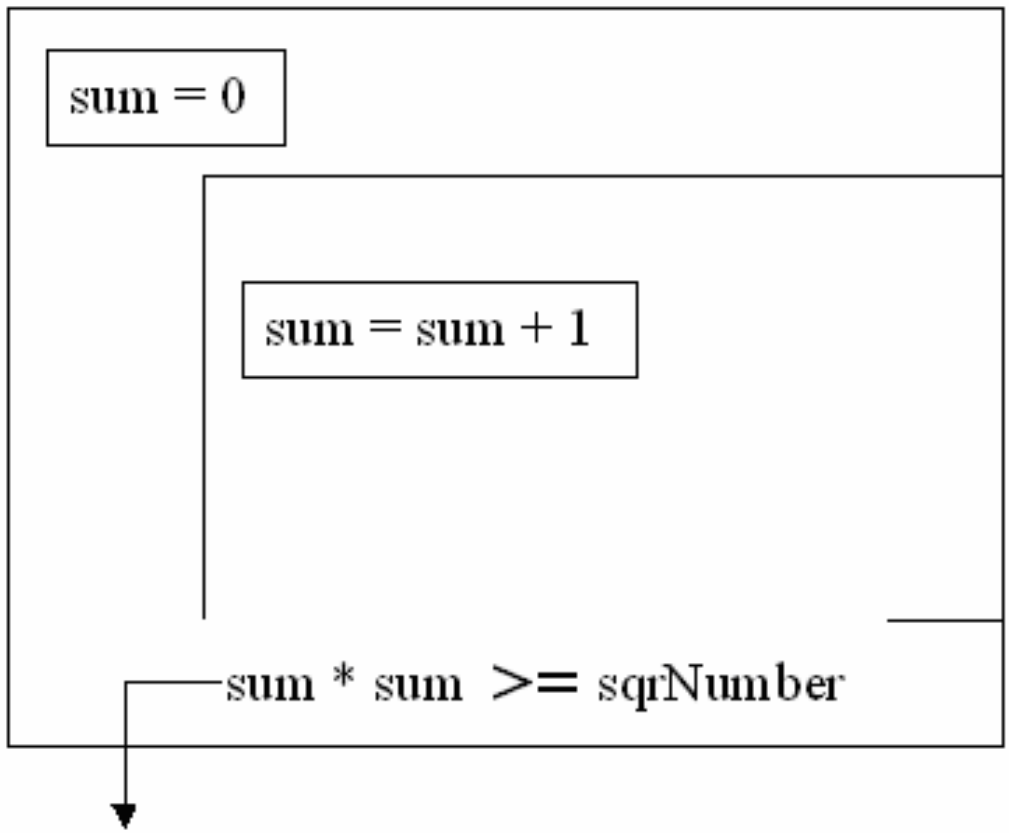
Flussdiagramm für die Summation



Struktogramm für die Schleife



Struktogramm für die Summation



Pseudocode für Quadratzahlen

{Berechne die Quadratzahlen von 1-10}

setze Wert auf 0

wiederhole

    addiere 1 auf Wert

    drucke Wert und Wert*Wert

bis Wert = 10

Bemerkung:

Die Abbruchbedingung wird **nach** der Anweisung geprüft.

Spezialfall: Endlosschleife (1)

Wiederhole

Anweisung(en)

immer

Beispiele:

Wiederhole

falls Reaktortemperatur zu hoch

dann fahre Bremsstäbe ein

immer

Wiederhole

Schalte Ampeln fort

immer

Spezialfall: Endlosschleife (2)

Anmerkung 1

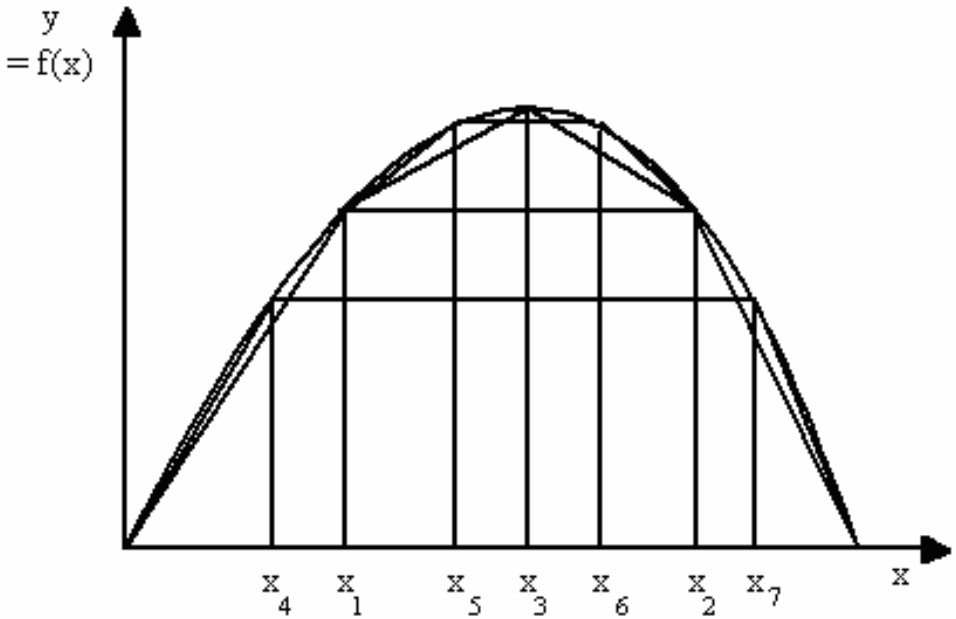
Endlosschleifen sind häufig unbeabsichtigt, weil die Abbruchbedingung nicht korrekt formuliert wurde.

Anmerkung 2

Endlose Algorithmen können korrekt sein, wenn sie ein über die Zeit stets wiederkehrendes Problem stets auf's Neue lösen. Sie sind nicht korrekt, wenn sie unendlich lange brauchen, um ein Problem zu lösen.

Beispiel: Newton-Iteration zur Integralberechnung ohne Abbruchschranke

Spezialfall: Endlosschleife (3)



Approximation eines Integrals

Abbruchbedingung: $\left| \Sigma_{neu} - \Sigma_{alt} \right| < \varepsilon$

Zweite Form der Iteration

```
Solange          Bedingung führe aus
                 Anweisung(en)      // Rumpf der Schleife
Anweisung n + 1                     // nicht mehr in der Schleife
```

Die Anweisungen in der Schleife werden ausgeführt, solange die Bedingung erfüllt ist.

Unterschied zur ersten Form: Die Bedingung wird **vor** der Ausführung des Schleifenrumpfes geprüft. Deshalb ist diese Form vorzuziehen, wenn damit gerechnet werden muss, dass in manchen Fällen bereits beim erstmaligen Eintritt in die Schleife die Abbruchbedingung erfüllt ist, der Schleifenrumpf also nicht ausgeführt werden soll.

Die beiden Formen der Iteration sind äquivalent, sie lassen sich (unter Verwendung der bedingten Anweisung) ineinander überführen ($\Rightarrow$ Übungsaufgabe).

Erste Form vs. zweite Form der Iteration (1)

Beispiel: Suche größte Zahl aus einer Liste

Algorithmus 1

Setze die erste Zahl der Liste als bislang größte Zahl

Wiederhole

Lies die nächste Zahl der Liste

Falls diese Zahl > bislang größte Zahl

dann setze diese Zahl als bislang größte Zahl

bis Liste erschöpft ist

Schreibe die bislang größte Zahl nieder

Dieser Algorithmus ist inkorrekt, falls eine gegebene Liste nur eine Zahl enthält! Daher:

Erste Form vs. zweite Form der Iteration (2)

Algorithmus 2

Setze die erste Zahl der Liste als bislang größte Zahl

Solange die Liste nicht erschöpft ist **führe aus**

    Lies die nächste Zahl der Liste

    Falls diese Zahl > bislang größte Zahl

        dann setze diese Zahl als bislang größte Zahl

Schreibe die bislang größte Zahl nieder

Java-Code für die Summation

```
sum = 0;
do {
  sum++;
  System.out.println(sum);
} while (sum * sum < 10);
```

```
sum = 0;
while (sum * sum < 10) {
  sum++;
  System.out.println(sum);
}
```

Äquivalenz von `for`- und `while`-Schleifen

```
for ( Ausdruck1; Ausdruck2; Ausdruck3 )
        Anweisung
```

ist äquivalent zu

=> Übungsaufgabe!

Java-Code für Grafikbeispiel (1)

```
import java.awt.Graphics;
import java.applet.Applet;
public class ContinueTest extends Applet {
  public void paint(Graphics g)
  {
    int xPos = 25;
    for ( int count = 1; count <= 10; count++ ) {
      if ( count == 5 ) {
        continue;
      }
```

Fortsetzung ----→

Java-Code für Grafikbeispiel (2)

```
      g.drawString(Integer.toString(count),xPos,25);
      xPos += 10;
      }
   }
}
```

Resultat

1 2 3 4 6 7 8 9 10

Die Anweisung **`continue`** dient dazu, das Ausdrucken der 5 zu unterbinden.

Zusammenfassung der Ablaufsteuerung

a) Sequenz (Folge)

b) Selektion

c) Iteration

Diese drei Konstrukte genügen, um jeden erdenklichen Algorithmus auszudrücken!

(Beweis: Theoretische Informatik)

3.5 Modularität

Das Konzept der Modularität

Ist es möglich, Teile aus einem Algorithmus an anderer Stelle, in demselben oder in einem anderen Algorithmus, wieder zu verwenden?

Ist es möglich, Teile aus dem Algorithmus so voneinander abzugrenzen, dass sie unabhängig voneinander entwickelt (verfeinert) werden können?

$\Rightarrow$ **Modularität**

Ein **Modul** ist ein Teilalgorithmus, der ein Teilproblem selbständig löst, ohne auf außerhalb liegende Schritte des Hauptalgorithmus Bezug zu nehmen. Module sind die **Bausteine von Algorithmen**.

Die Wirkung eines Moduls wird über **Parameter** gesteuert.

Beispiele für Module sind **Unterprogramme** (Prozeduren, Funktionen) in traditionellen Programmiersprachen und **Klassen** in Java.

Beispiele für Module (1)

Beispiel

Algorithmus "koche Kaffee" und Algorithmus "koche Tee" benutzen beide das Modul "koche Wasser" mit den Schritten

// **koche_Wasser**

(1.1.1.) Stelle Kessel unter Wasserhahn

(1.1.2.) Drehe Wasserhahn auf

(1.1.3.) Warte, bis Kessel voll ist

(1.1.4.) Drehe Wasserhahn zu

(1.2.) Schalte Kessel an

(1.3.) Warte, bis Kessel pfeift

(1.4.) Schalte Kessel aus

Beispiele für Module (2)

// Algorithmus zur Zubereitung einer Tasse Kaffee

(1.) rufe Modul **koche_Wasser** auf

// Gib Kaffeepulver in die Tasse

(2.1.1.) Nehme Kaffeeglas aus dem Fach

(2.1.2.) Entferne Deckel vom Kaffeeglas

(2.2.) Entnehme einen Löffel Kaffee

(2.3.) Kippe Löffel in die Tasse

(2.4.1.) Setze Deckel auf das Kaffeeglas

(2.4.2.) Stelle Kaffeeglas in das Fach zurück

// Fülle Wasser in die Tasse

(3.1) Gieße Wasser aus dem Kessel in die Tasse, bis die Tasse voll ist

Beispiele für Module (3)

// Algorithmus zur Zubereitung einer Tasse Tee

(1.) rufe Modul **koche_Wasser** auf

// Gib den Teebeutel in die Tasse

(2.1) Öffne die Packung mit den Teebeuteln

(2.2) Entnimm einen Teebeutel

(2.3) Lege den Teebeutel in die Tasse

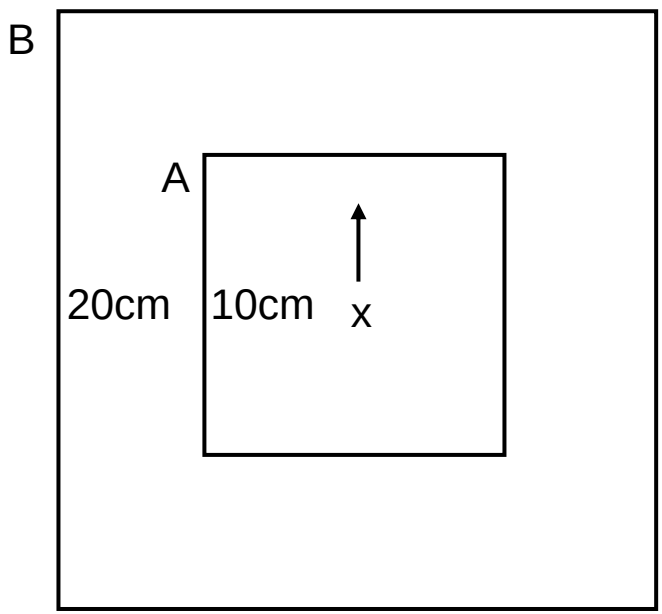
// Fülle Wasser in die Tasse

(3.1.) Gieße Wasser aus dem Kessel in die Tasse, bis die Tasse voll ist

Grafik-Beispiel (1)

Algorithmus "Zeichne zwei Quadrate"

- Positioniere auf Punkt A
- Zeichne Quadrat der Seitenlänge 10 cm
- Positioniere auf Punkt B
- Zeichne Quadrat der Seitenlänge 20 cm



Grafik-Beispiel (2)

Definiere das Zeichnen eines Quadrates als Modul mit dem Parameter "Seitenlänge".

Annahme: Der Plotter kann die folgenden Befehle unmittelbar ausführen:

bewege (x)	bewege Stift x cm vorwärts
links (x)	drehe Stift um x Grad nach links
rechts (x)	drehe Stift um x Grad nach rechts
Stift heben	hebe den Stift vom Papier
Stift senken	senke den Stift auf das Papier.

Grafik-Beispiel (3)

Dann kann das Modul "Quadratzeichnen" wie folgt definiert werden:

Modul Quadratzeichnen(Größe)

/* Zeichnet ein Quadrat mit der Seitenlänge `Größe` cm. Das Quadrat wird gegen den Uhrzeigersinn gezeichnet, beginnend mit der aktuellen Position des Zeichenstiftes. Die erste Kante wird entsprechend der aktuellen Ausrichtung des Zeichenstiftes gezeichnet. Der Zeichenstift wird auf den Ausgangspunkt zurückgesetzt und abgehoben. */

Stift senken

Wiederhole *4-mal*

    Bewege (Größe)

    Links (90)

Stift heben

Grafik-Beispiel (4)

Bei der Definition eines Moduls: **formale Parameter**

Beim Aufruf eines Moduls: **aktuelle Parameter**

Anzahl, Reihenfolge und Datentyp müssen übereinstimmen.

Damit kann der Algorithmus "Zeichne zwei Quadrate" geschrieben werden als:

(1) Positioniere Stift auf Punkt A

(2) Quadratzeichnen(10)

(3) Positioniere Stift auf Punkt B

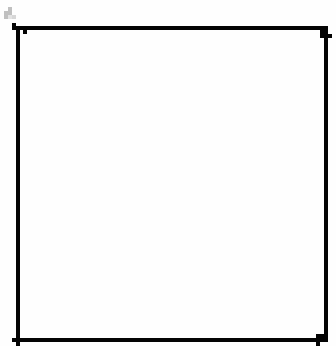
(4) Quadratzeichnen(20)

Grafik-Beispiel (5)

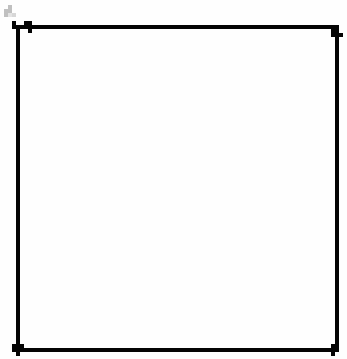
Programmablauf

nach (1)

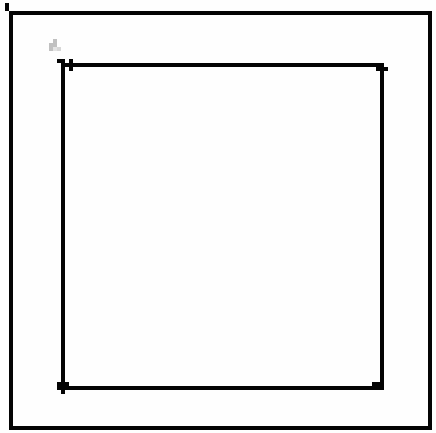
nach (2)



nach (3)



nach (4)



Dokumentieren von Modulen

Neben den explizit angegebenen Parametern sollten die impliziten Annahmen, die ein Modul macht, sorgfältig dokumentiert sein!

Beispiel: Zeichenrichtung der ersten Kante. Am besten als Kommentar im Kopf (Header) des Moduls.

Ebenso müssen Nebeneffekte nach Möglichkeit vermieden oder sorgfältig dokumentiert sein.

Die Schritte (1) und (3) können im Zuge der schrittweisen Verfeinerung ebenfalls als Module beschrieben oder direkt in Befehle an den Stift des Plotters aufgelöst werden.

Modularität in Java: Klassen und Objekte

Mit einer **Klassendeklaration** definiert man in Java neue Referenztypen und legt gleichzeitig deren Implementation fest. Jede Klasse ist implizit Subklasse der Java-Klasse `Object`; es gibt also eine gemeinsame Wurzel der Java-Vererbungshierachie.