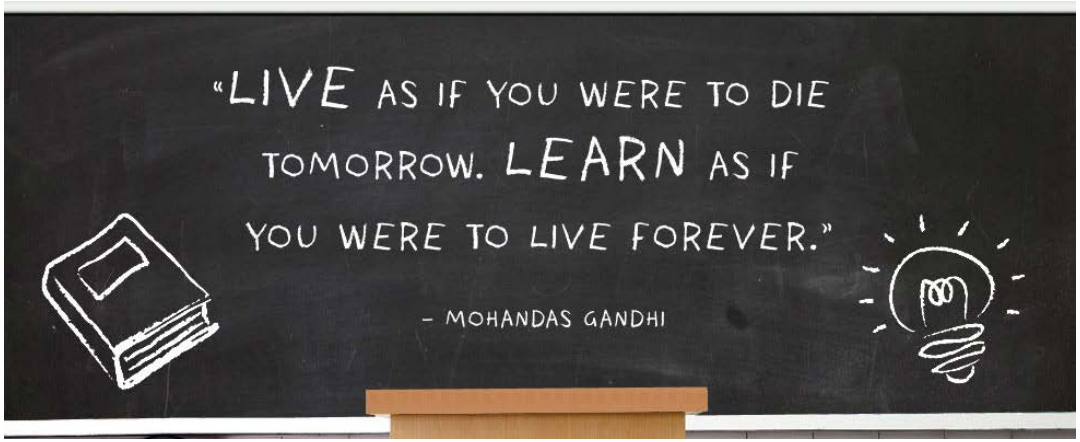
Grundlagen der Programmierung

VL04: Anweisungen und Ablaufsteuerung - Teil I

Prof. Dr. Samuel Kounev,
M.Sc. Norbert Schmitt



Danksagung

- Vorlesungsmaterialien von Prof. Dr. Detlef Seese wurden als Basis verwendet
- Unterstützung bei der technischen und inhaltlichen Gestaltung des Vorlesungsmaterials leisteten:

  Jóakim v. Kistowski

  Dietmar Ratz, Joachim Melcher, Roland Küstermann, Jana Weiner, Hagen Buchwald, Matthes Elstermann, Oliver Schöll, Niklas Kühl, Tobias Diederich

Inhalt und Ziele

- Warum benötigen wir eine Ablaufsteuerung und was versteht man darunter?
- Anweisungen
- Blöcke und ihre Struktur
- Entscheidungsanweisungen
  - Die **`if`**-Anweisung
  - Die **`switch`**-Anweisung

Anweisungen und Ablaufsteuerung

- Bisher bekannte Arten von Anweisungen
  - Deklaration z.B. `int a;`
  - Wertzuweisung z.B. `a = 5;`
  - Methodenaufruf z.B. `System.out.println(a);`
  - Leere Anweisung z.B. `;`
  - Hierdurch können wir nur rein lineare Programmstrukturen erzeugen
  - Dieses Vorlesungskapitel beschäftigt sich mit weiteren Strukturierungsmöglichkeiten

Strukturierung durch Blöcke

- Ein Block ist eine durch Klammern { und } zusammengefasste Folge von Anweisungen

```
{
   Anweisung1
   Anweisung2
   ...
   Anweisungk
}
```

- Blöcke können geschachtelt sein, d.h. jede Anweisung kann selbst wieder durch einen Block ersetzt werden
- Sie dienen der Strukturierung von Programmen

Strukturierung durch Blöcke (2)


```
{ int i = 1;
   {                          // Start des ersten Blocks
      int j = 2;              // j ist lokal im ersten Block
      ...                     // i ist hier auch bekannt
   }                          // Ende des ersten Blocks
   {                          // Start des zweiten Blocks
      long k;                 // k ist lokal im zweiten Block
      ...                     // i ist bekannt, j unbekannt
   }                          // Ende des zweiten Blocks
   ...                        // i ist bekannt, j und k nicht
}

                              // i, j und k sind hier unbekannt
```

Variablen sind nur bis zum Ende des Blocks, in dem sie definiert wurden, gültig!

Einfache Anweisungen

Java

- Die bisher (z.B. ohne Verzweigungen und Schleifen) erzeugbaren Programme haben eine sehr einfache Gestalt:

```
public class Bezeichner {
   public static void main (String [] args) {
      Anweisung1
      Anweisung2
      ...
      Anweisungn-1
      Anweisungn
   }
}
```

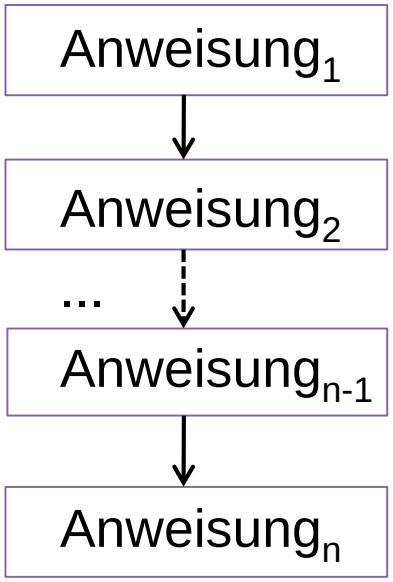
- Anweisungen können hier auch durch Blöcke oder geschachtelte Blöcke ersetzt werden

Einfache Anweisungen (2)

Struktogramm des obigen Programms

Einfache Anweisungen (3)

Programmablaufplan (PAP) des obigen Programms

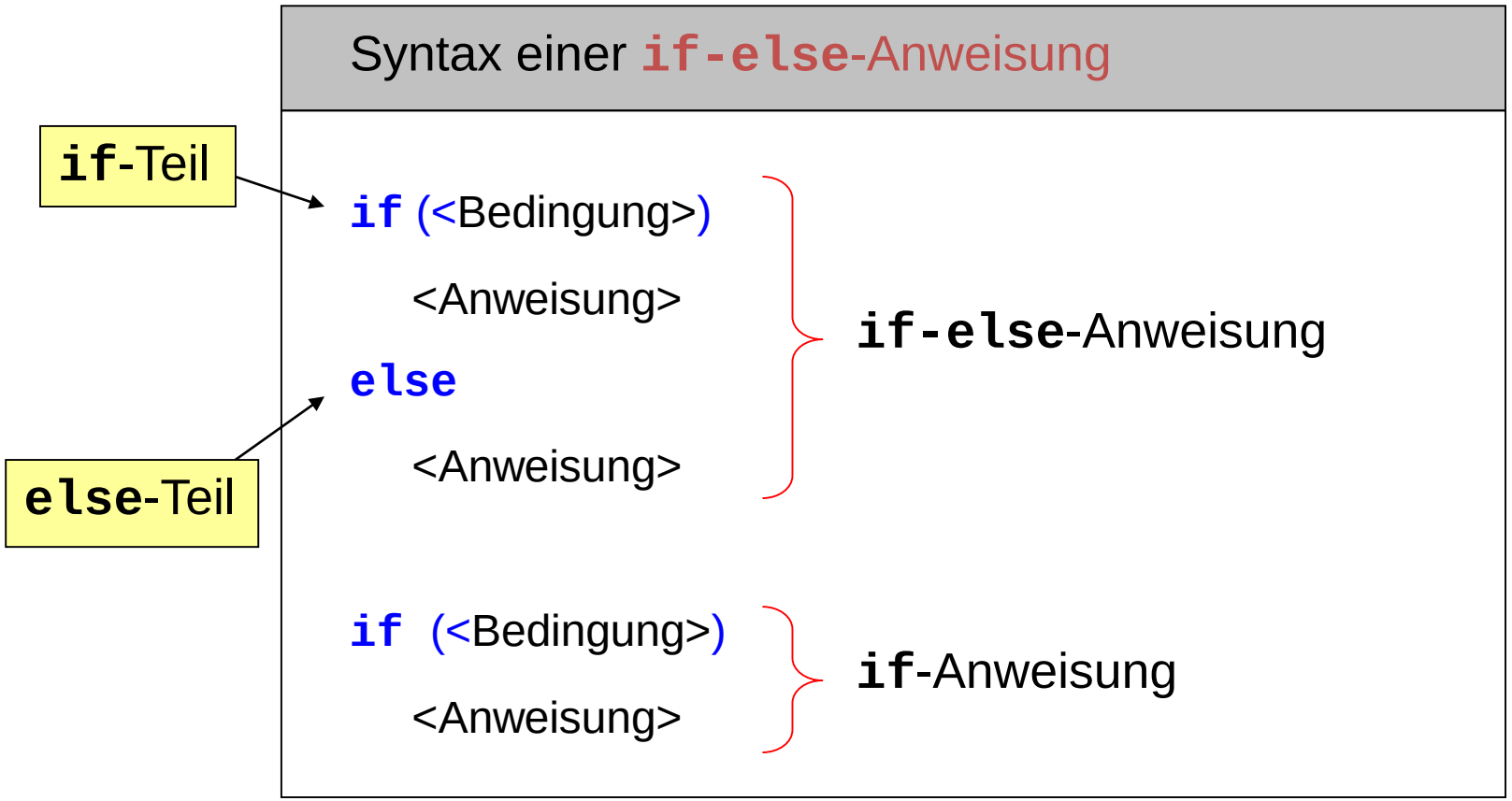


- Durch solche Programme kann weder die Suche von Elementen in einer Datei, noch das Sortieren einer Datei, noch das Zählen der Anzahl der Elemente einer Datei bewerkstelligt werden.
- Wie erzeugt man Verzweigungen, bzw. Schleifen?

`if`-Anweisung

Java

C/C++



<Bedingung> = Boolescher Ausdruck

`if`-Anweisung (2)

Java

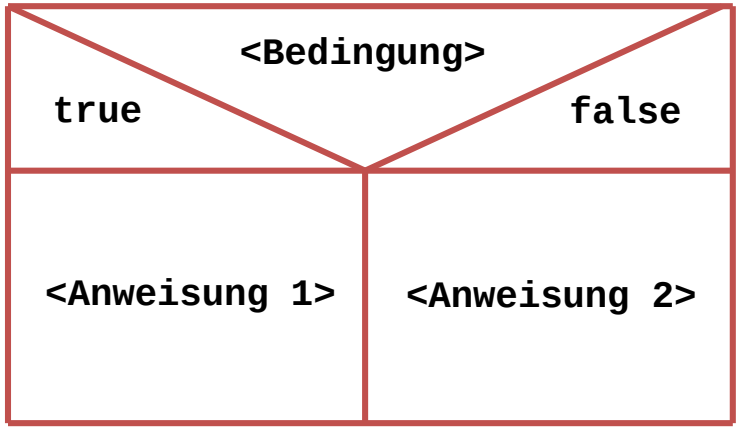
C/C++

Die Anweisungen können durch Blöcke ersetzt werden

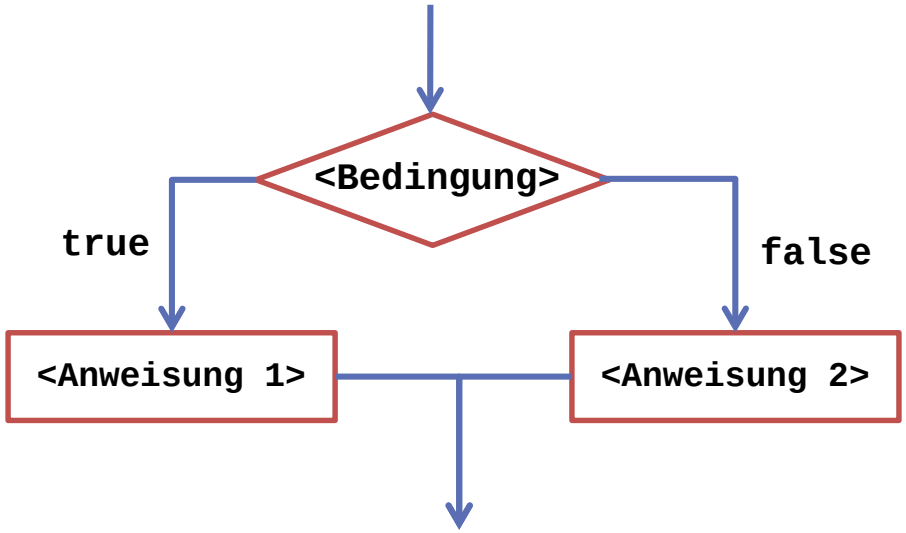
```
if(<Bedingung>){
    <Anweisung>
    ...
    <Anweisung>
}
else {
    <Anweisung>
    ...
    <Anweisung>
}
```

Semantik von `if`-Anweisung

- Zunächst wird die Bedingung ausgewertet
- Ist sie wahr (true), so führt man den **`if`**-Teil aus und überspringt den **`else`**-Teil
- Ist sie falsch (false), so überspringt man den **`if`**-Teil und führt nur den **`else`**-Teil aus



Struktogramm



Programmablaufplan (PAP)

Beispiele: `if`-Anweisung

Java

```
// (c)
    double x = InputHelper.readDouble("x = ");
    double y = InputHelper.readDouble("y = ");
    if (x > 0)
      if (y > 0)
        System.out.println("beide positiv");
      else
        System.out.println("nur x positiv");
    else
      if (y > 0)
        System.out.println("nur y positiv");
      else
        System.out.println("beide nicht positiv");
```

Regel: Ein **`else`** wird dem nächstliegenden vorangehenden **`if`** zugeordnet, dem noch kein **`else`** zugeordnet ist und das in keinem separaten Block liegt. Besser ist, zu klammern!

Beispiele: `if`-Anweisung

Java

Übung: Zu welchem **`if`** gehört das **`else`**?

```
int a;
int b;
...                          // Anweisungen
if (a > 0)
   if (b > a)
       System.out.println("b > 1");
else
   System.out.println( "Fehler: a ist <= Null");
```

zum zweiten **if**

Klammern können Semantik präzisieren

Beispiele: `if`-Anweisung

Java

```
// (d) Mehrfachverzweigung
    Scanner input = new Scanner(System.in);
    System.out.println("Punktzahl: ");
    int punkte = input.nextInt();
    String note;
    if (punkte > 50)
      note = "sehr gut";
    else if (punkte > 41)
      note = "gut";
    else if (punkte > 33)
      note = "befriedigend";
    else if (punkte > 25)
      note = "ausreichend";
    else
      note = "nicht bestanden";
    System.out.println("Note: " + note);
```

Beispiele: `if`-Anweisung

Java

```
// (d) Mehrfachverzweigung
    Scanner input = new Scanner(System.in);
    System.out.println("Punktzahl: ");
    int punkte = input.nextInt();
    String note;
    if (punkte > 50)
      note = "sehr gut";
    else if (punkte > 41)
            note = "gut";
         else if (punkte > 33)
                 note = "befriedigend";
              else if (punkte > 25)
                      note = "ausreichend";
                   else
                      note = "nicht bestanden";
    System.out.println("Note: " + note);
```

Beispiele: `if`-Anweisung

Java

```
// (e) Nicht compilierbar!
    int u = 5, v = 2, w = 4;
    int z;
    if (u = v + w)
      z = 27;
    else
      z = z + 27;

// (f) Compilierbar, aber logisch problematisch!
    int z = 100;
    boolean bu = true, bv = false, bw = false;
    if (bu = bv && bw)
      z = 42;
    else
      z = z + 42;
    System.out.println("z = " + z);
    System.out.println("bu = " + bu);
```

Zuweisung anstelle von Vergleich!
Dadurch **`int`**-Wert in der Klammer.

Die Variable **`z`** ist nicht initialisiert!

Zuweisung anstelle von Vergleich!
Dadurch ist der Wert in der Klammer
durch den neuen Wert von **`bu`** gegeben.

Beispiele: `if`-Anweisung

Java

Beispiel: Wo endet diese if-else-Anweisung?

Welches Problem taucht hier auf?

```
int x = (new Scanner(System.in)).nextInt();
if (x == 0) {
    System.out.println("x ist gleich 0");
}
else
    System.out.println("x ist ungleich 0");
    System.out.println("1/x liefert " + 1/x);
    System.out.println("Division durchgefuehrt");
```

Problem: Division durch 0. Es fehlen Klammern!

`switch`-Anweisung

Java

C/C++

Im Gegensatz zur **`if`**-Anweisung erlaubt die **`switch`**-Anweisung eine Auswahl unter mehreren Alternativen

Syntax `switch`-Anweisung

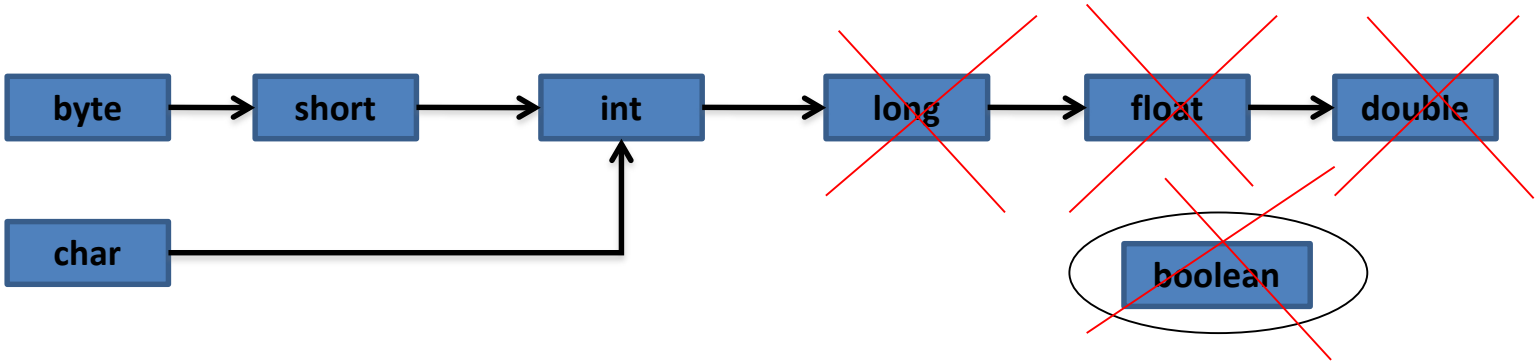
```
switch (<switch-Ausdruck>) {
        case <Konstante>: <Anweisungsfolge>; break;
        case <Konstante>: <Anweisungsfolge>; break;
        ...
        case <Konstante> : <Anweisungsfolge>; break;
        default : <Anweisungsfolge>
}
```

Semantik: Berechne Wert des **`switch`**-Ausdrucks und prüfe der Reihe nach die Konstanten bis eine Übereinstimmung gefunden wird. Stimmt der Wert erstmals mit der i-ten Konstante überein, so gelangt i-te Anweisungsfolge zur Ausführung.

`switch`-Anweisung

Java

- Der <`switch`-Ausdruck> muss einen der folgenden Datentypen zurückliefern: **`byte`**, **`short`**, **`int`** oder **`char`**



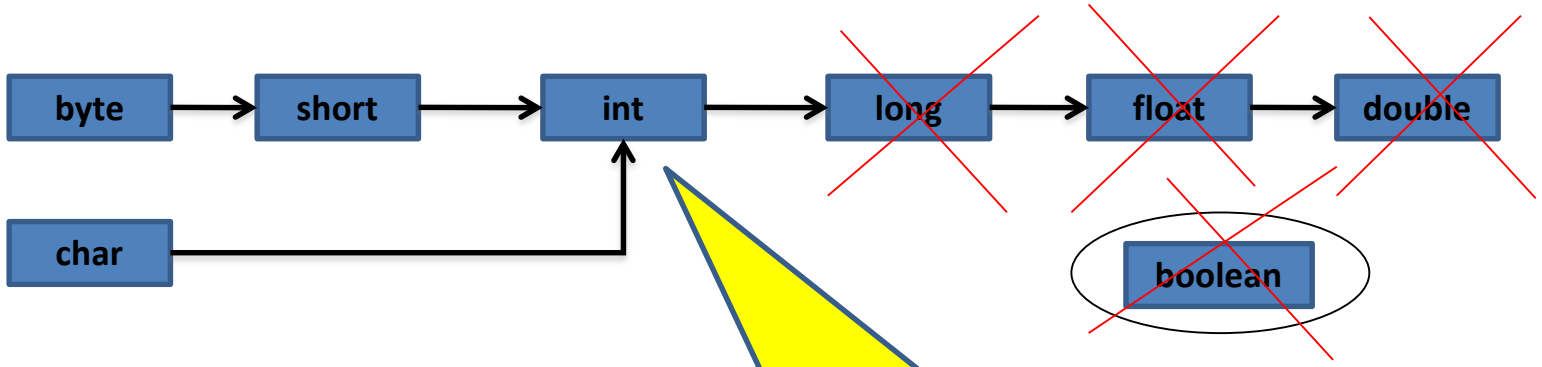
- Die Werte hinter dem **`case`** müssen vom Typ des `switch`-Ausdrucks sein, bzw. diesem zuweisbar sein (siehe später Zuweisungskompatibilität).
- Die Werte sind konstante Ausdrücke.
- Kein Wert der Konstanten dieser `case`-Zweige darf doppelt auftreten. Ihre Ordnung ist jedoch egal.
- Eine **Anweisungsfolge** ist eine Folge von Anweisungen:

  <Anweisung> ... <Anweisung>

`switch`-Anweisung (Java 7)

Java

- Der <`switch`-Ausdruck> muss einen der folgenden Datentypen zurückliefern: **`byte`**, **`short`**, **`int`** oder **`char`**



- Die Werte hinter dem **`case`** müssen v… …witch-Ausdrucks sein, bzw. diesem zuweisbar sein (siehe …
- Die Werte sind konstante A…
- Kein Wert der Konstant… Ihre Ordnung ist jedoch eg…
- Eine **Anweisungsfolge** ist ein…

  <Anweisung> ... <Anweisung>

Java 7 erlaubt hier auch **`Strings`**, **`Character`**, **`Byte`**, **`Short`**, **`Integer`** und **`enum`**-Typen - Siehe später nach der Einführung in Objektorientierung.

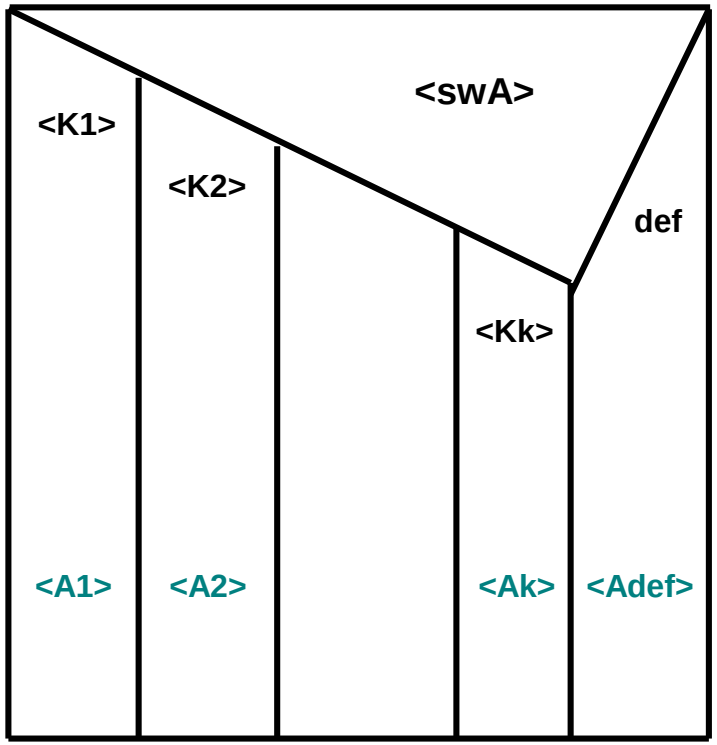
`switch`-Anweisung

Java

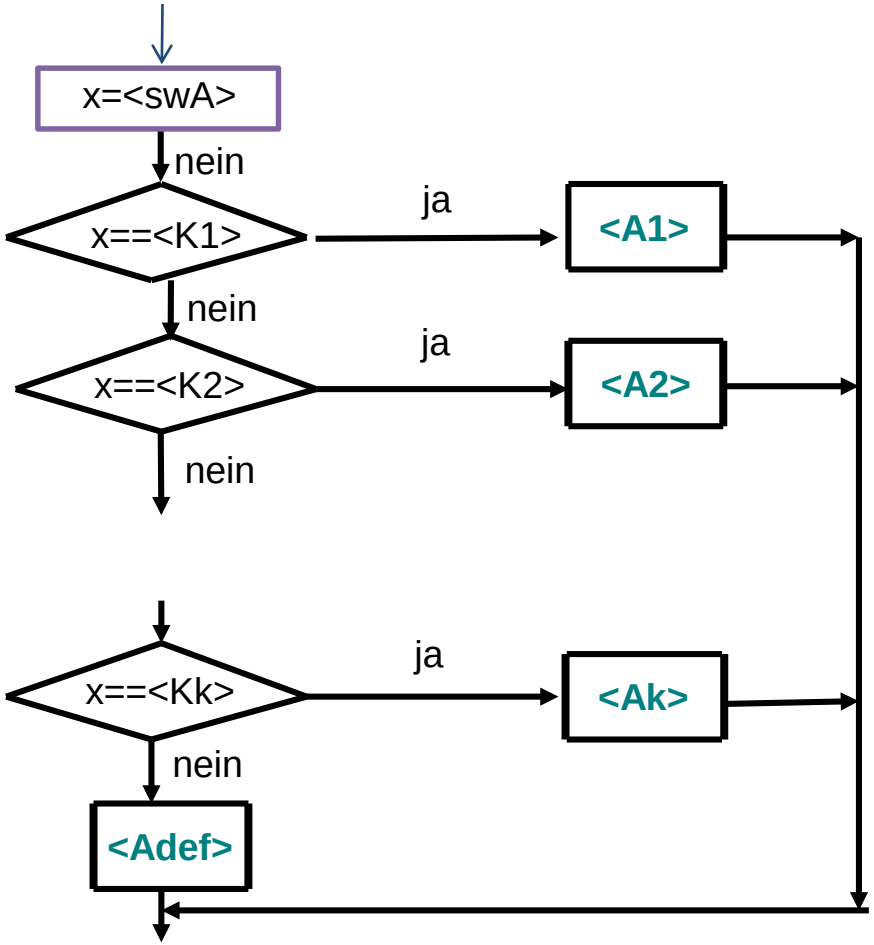
C/C++

```
switch (<swA>) {
        case <K1>: <A1> break;
        case <K2> : <A2> break;
        …
        case <Kk> : <Ak> break;
        default : <Adef>
    }
```

`switch`-Anweisung



Struktogramm



Programmablaufplan (PAP)

Beispiel: `switch`-Anweisung

Java

```
System.out.println("Zeichen eingeben: ");
char t = (new Scanner(System.in)).nextChar();
    int s = 7;
    switch (t) {
      case 'X':
        s = 25;
        break;
      case 'Y':
        s = 40;
        break;
    }
    System.out.println("s = " + s);
```

Eingabe:	Ausgabe:
X	s = 25
Y	s = 40
x	s = 7
r	s = 7

`switch`-Anweisung

Java

C/C++

```
switch (<switch-Ausdruck>) {
        case <Konstante>: <Anweisungsfolge> break;
        case <Konstante> : <Anweisungsfolge> break;
        ...
        case <Konstante> : <Anweisungsfolge> break;
        default : <Anweisungsfolge>
    }
```

Der **`default`**-Zweig ist optional. Es darf höchstens ein **`default`**-Zweig auftreten.

Die Anweisungsfolgen können auch Blöcke sein.

`break` ist optional.

`switch`-Anweisung

- Befindet sich ein **`break`** hinter der betreffenden Anweisung, so wird die **`Switch`**-Anweisung beendet, *andernfalls* werden auch die weiteren Anweisungsfolgen ausgeführt (*bis zum nächsten break*).
- Stimmt der Wert des **`Switch`**-Ausdrucks mit keinem der Werte hinter dem Schlüsselwort **`case`** überein, so wird die Anweisung hinter dem **`default`** ausgeführt.
- Danach wird die **`switch`**-Anweisung beendet.

Beachte:
Vergessene **`break`**-Anweisungen führen oft zu schwer auffindbaren Fehlern!

`switch`-Anweisung

Java

```
Scanner input = new Scanner(System.in);
System.out.print("a = ");
int a = input.nextInt();
int b;
switch (a) {
  case 1:
    b = 10;
    a = 100;
  case 2:
  case 3:
    b = 25;
    a = 1500;
    break;
  case 4:
    b = 40;
    a = 1000000;
  default:
    b = 0;
    a = 0;
}
System.out.println("a = " + a + "   b = " + b);
```

Eingabe:	Ausgabe:
a = 3	a = 1500 b = 25
a = 2	a = 1500 b = 25
a = 1	a = 1500 b = 25
a = 4	a = 0 b = 0
a = 7	a = 0 b = 0

Beispiel Monatslänge

Java

- **Aufgabe:** Schreiben Sie ein Programm, welches durch eine **switch**-Anweisung aus der Nummer eines Monats die Länge des betreffenden Monats ermittelt. Lassen Sie dabei Schaltjahre unberücksichtigt.

```
System.out.println("Monat = ");
int monat = (new Scanner(System.in)).nextInt();
    int tage = 0;
    switch ( monat ) {
      case  4:
      case  6:
      case  9:
      case 11:  tage=30; break;
      case  2:  tage=28; break;
      default:  tage=31; break;
    }
    System.out.println("Tage = " + tage);
```

Übung

Worin unterscheiden sich die beiden Anweisungen?

```
System.out.println("Punkte = ");
int punkte = (new Scanner(System.in)).nextInt();
char note;
if      (punkte >= 80)                  note = '1';
else if (punkte >= 70)                  note = '2';
else if (punkte >= 60)                  note = '3';
else if (punkte >= 50)                  note = '4';
else                                    note = '5';

switch ((punkte / 10)) {
   case 10:
   case 9:
   case 8: note = '1'; break;
   case 7: note = '2'; break;
   case 6: note = '3'; break;
   case 5: note = '4'; break;
   default: note = '5'; break;
}
```

`switch`-Anweisung

- Ist der Wert des **`switch`**-Ausdrucks während des Ablaufs einer **`switch`**-Anweisung unveränderlich?

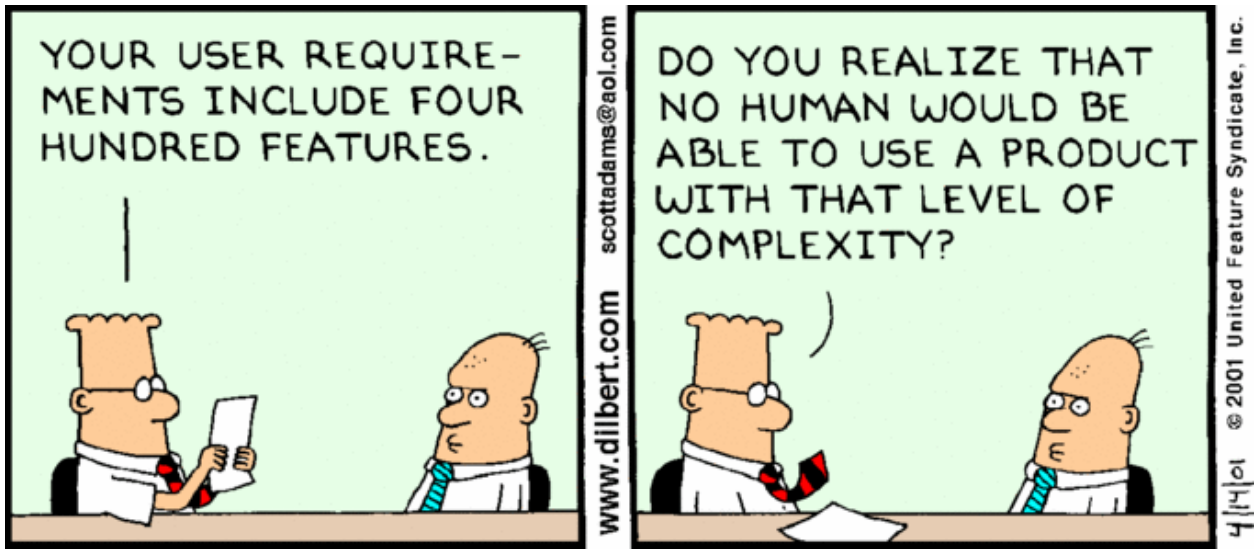
```
int x = 1;
switch(x) {
  case 1: x = 3;
  case 4: x = 5; break;
  case 3: x = 22;
}
System.out.println(x); // Welchen Wert hat x?
```

Zuweisungskompatibilität

```
short x = 3;
switch(x) {
  case (byte)1: x = 11;
  case 'a': x = 12;
  case (char)3: x = 13;
}
System.out.println(x);
```

Die betreffenden Ausdrücke müssen nur **zuweisungskompatibel** sein, d.h. beide Typen müssen gleich sein oder es muss eine automatische Typkonvertierung des Typs der im **`case`**-Zweig angegebenen Konstanten in den Typ des switch-Ausdrucks durchführbar sein (mehr später).

Fragen?


YOUR USER REQUIREMENTS INCLUDE FOUR HUNDRED FEATURES.
DO YOU REALIZE THAT NO HUMAN WOULD BE ABLE TO USE A PRODUCT WITH THAT LEVEL OF COMPLEXITY?
scottadams@aol.com
www.dilbert.com
© 2001 United Feature Syndicate, Inc.
4/4/01


GOOD POINT. I'D BETTER ADD "EASY TO USE" TO THE LIST.