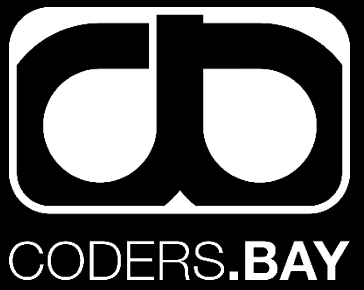
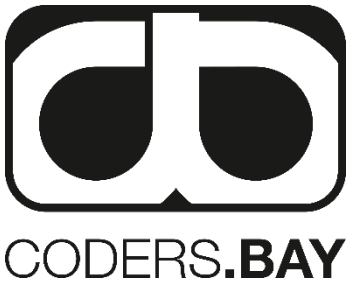


BEDINGTE CODE AUSFÜHRUNG (IF-ELSE, SWITCH)

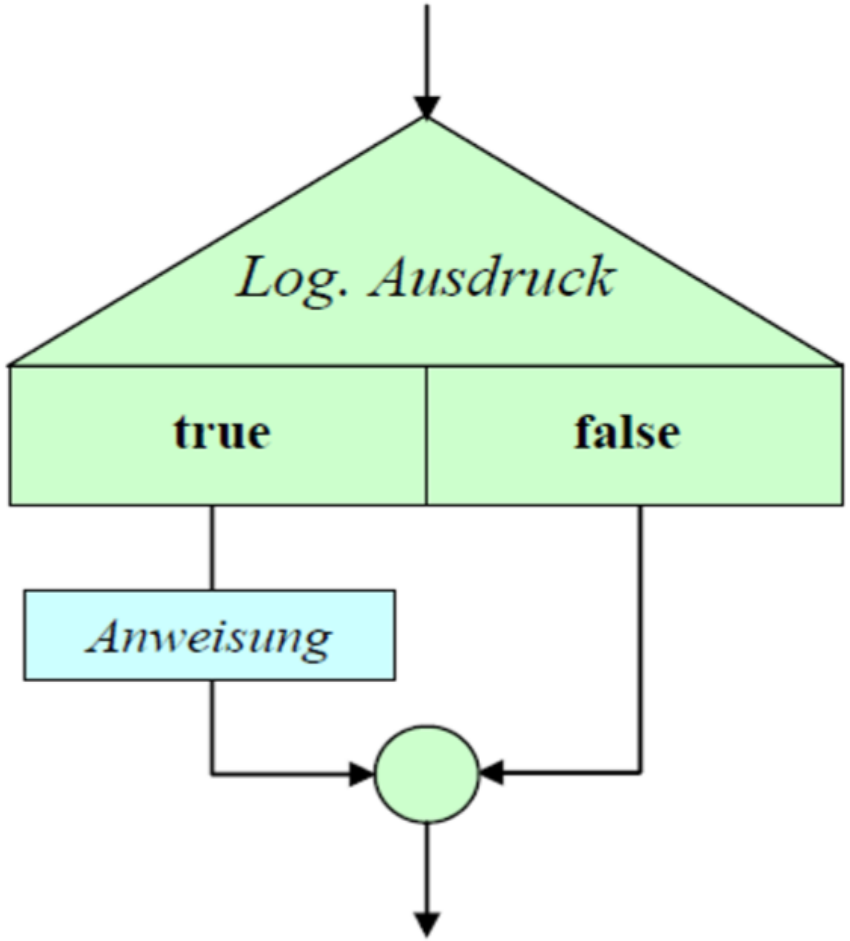
BEDINGTE ANWEISUNG UND FALLUNTERSCHEIDUNG



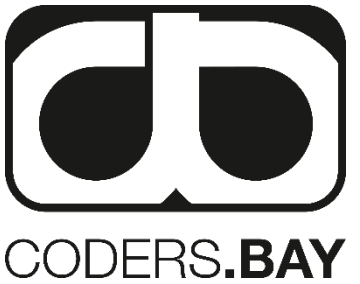
Oft ist es erforderlich, dass eine Anweisung nur unter einer bestimmten Bedingung ausgeführt wird. Etwas allgemeiner formuliert geht es darum, dass viele Algorithmen Fallunterscheidungen benötigen, also an bestimmten Stellen in Abhängigkeit vom Wert eines steuernden Ausdrucks in unterschiedliche Pfade verzweigen müssen.

IF-ANWEISUNG PROGRAMMABLAUFPLAN

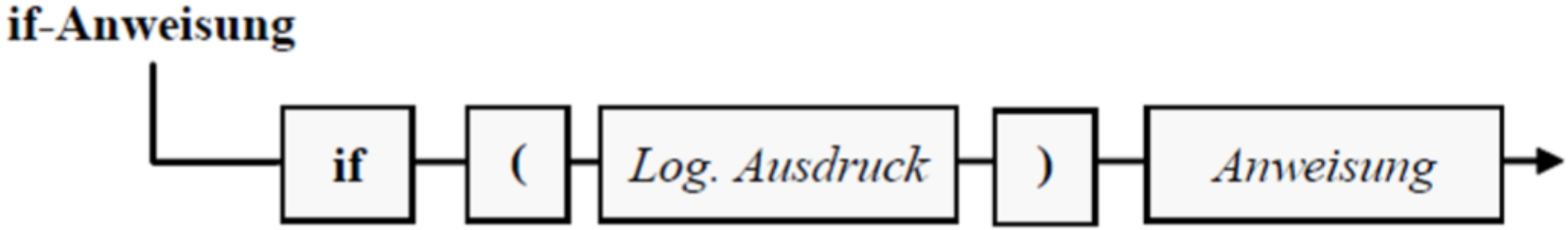
Nach dem folgenden Programmablaufplan (PAP) bzw. Flussdiagramm soll eine (Block-)Anweisung nur dann ausgeführt werden, wenn ein logischer Ausdruck den Wert true besitzt:



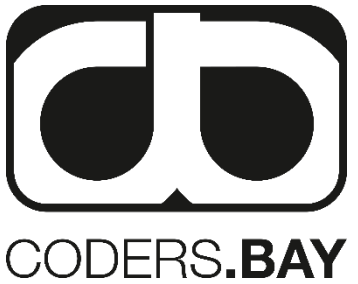
IF-ANWEISUNG SYNTAXDIAGRAMM



Während der Programmablaufplan den Zweck (die Semantik) eines Sprachbestandteils erläutert, beschreibt das Syntaxdiagramm recht präzise, wie zulässige Exemplare des Sprachbestandteils zu bilden sind.
Das folgende Syntaxdiagramm beschreibt die zur Realisation einer bedingten Ausführung geeignete **if-Anweisung**:

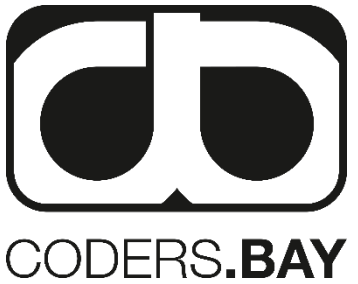


IF-ANWEISUNG SYNTAXDIAGRAMM



Um genau zu sein, muss zu diesem Syntaxdiagramm noch angemerkt werden, dass als bedingt aus- zuführende Anweisung keine Variablendeklaration erlaubt ist. Es ist übrigens nicht vergessen worden, ein Semikolon ans Ende des **if-Syntaxdiagramms** zu setzen. Dort wird eine Anweisung verlangt, wobei konkrete Beispiele oft mit einem Semikolon enden.

IF-ANWEISUNG BEISPIEL



Im folgenden Beispiel wird eine Meldung ausgegeben, wenn die Variable *anz* den Wert Null besitzt:

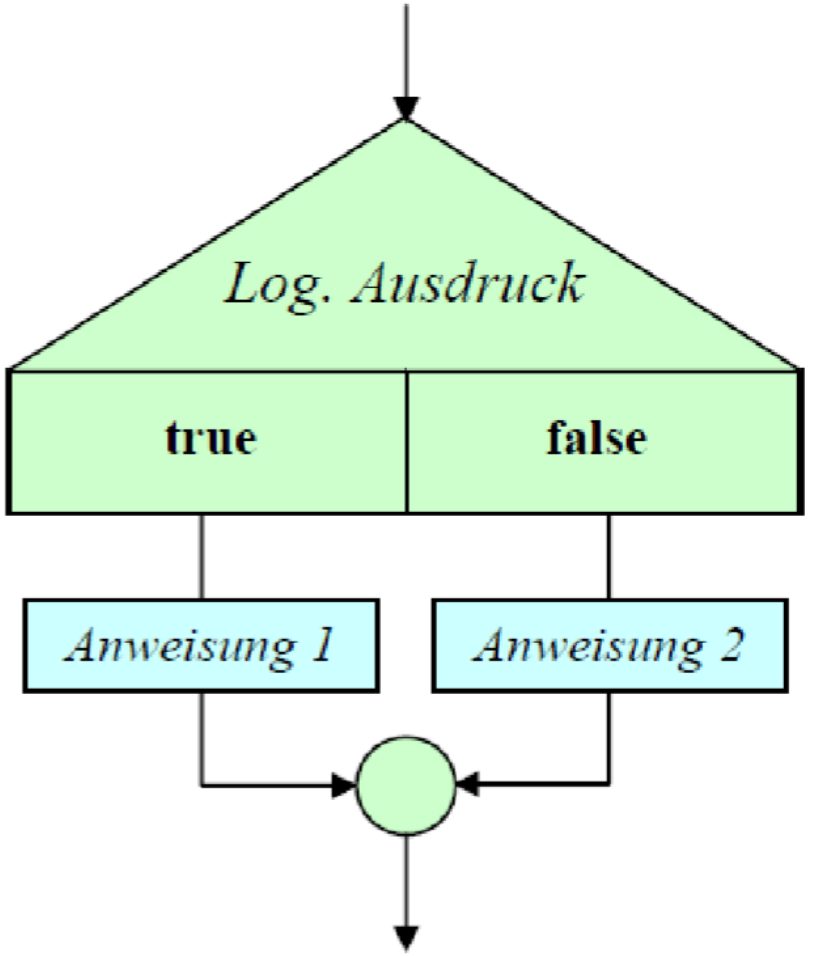
```
int anz=0;
    if (anz == 0)
        System.out.println("Die Anzahl muss > 0 sein!");
```

Der Zeilenumbruch zwischen dem logischen Ausdruck und der (Unter-)Anweisung dient, nur der Übersichtlichkeit und ist für den Compiler irrelevant. Selbstverständlich kommt als Anweisung auch ein *Block* in Frage.

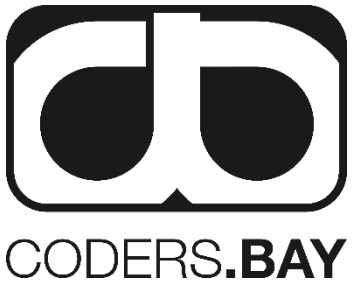
IF-ELSE-ANWEISUNG PROGRAMMABLAUFPLAN

CODERS.BAY

Soll auch etwas passieren, wenn der steuernde logische Ausdruck den Wert **false** besitzt, erweitert man die **if-Anweisung** um eine **else-Klausel**.



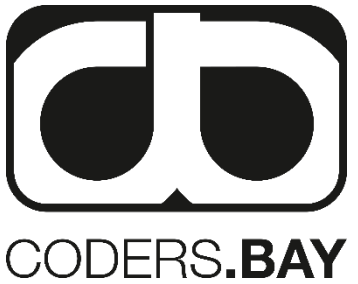
IF-ELSE-ANWEISUNG SYNTAXBESCHREIBUNG



Zur Beschreibung der if-else-Anweisung wird an Stelle eines Syntaxdiagramms eine alternative Darstellungsform gewählt, die sich am typischen Java-Quellcode-Layout orientiert:

```
if (Logischer Ausdruck)
     Anweisung 1
else
     Anweisung 2
```

IF-ELSE-ANWEISUNG SYNTAXBESCHREIBUNG

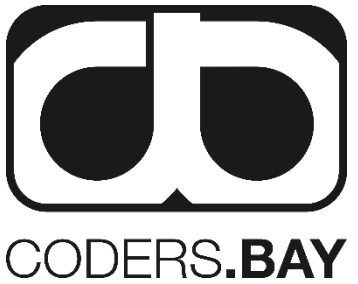


Wie bei den Syntaxdiagrammen gilt auch für diese Form der Syntaxbeschreibung:

- Für terminale Sprachbestandteile, die exakt in der angegebenen Form in konkreten Quellcode zu übernehmen sind, wird fette Schrift verwendet.

- Platzhalter sind durch kursive Schrift gekennzeichnet.

IF-ELSE-ANWEISUNG SYNTAXBESCHREIBUNG



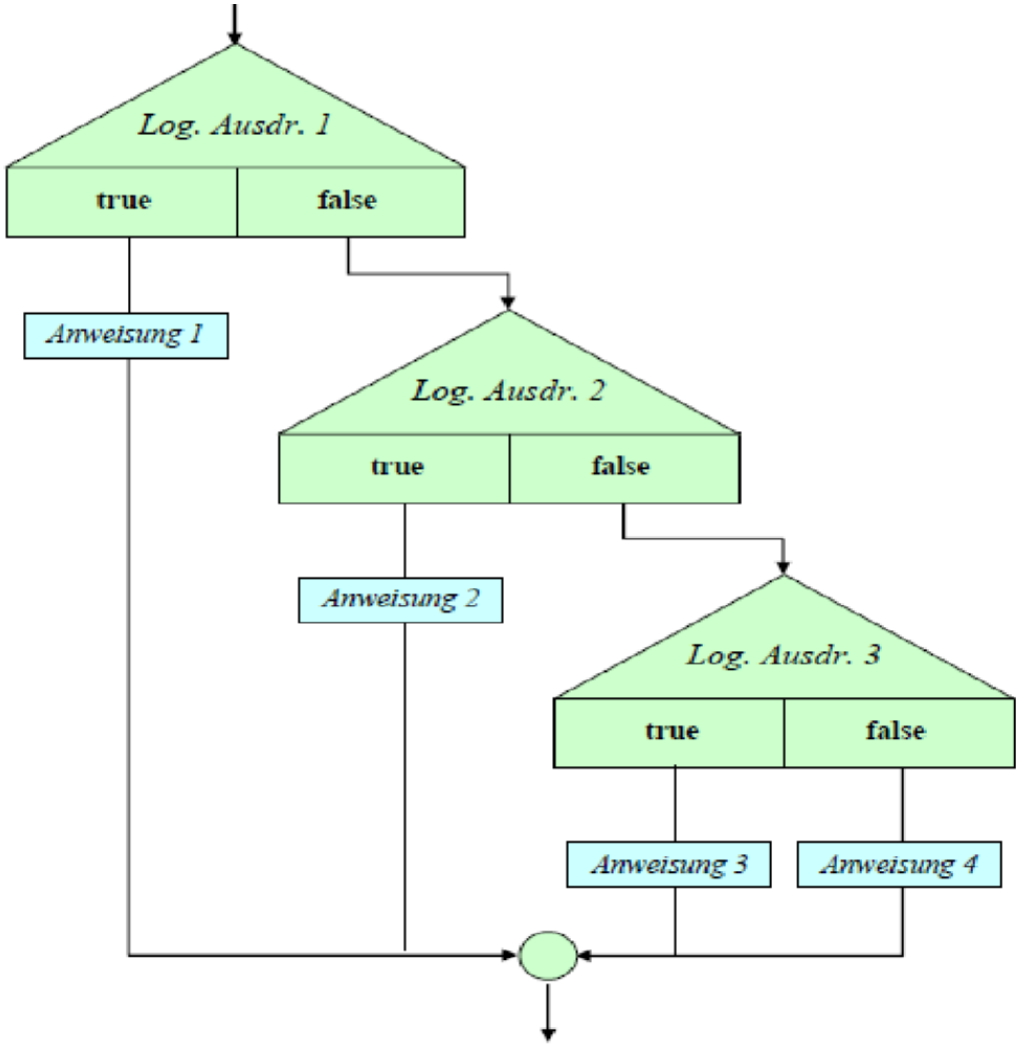
Während die Syntaxbeschreibung im Quellcode-Layout sehr übersichtlich ist, bietet das Syntaxdiagramm den Vorteil, bei komplizierter, variantenreicher Syntax alle zulässigen Formulierungen kompakt und präzise als Pfade durch das Diagramm zu beschreiben.

Wie schon bei der einfachen **if-Anweisung** gilt auch bei der **if-else-Anweisung**, dass Variablen als eingebettete Anweisungen erlaubt sind.

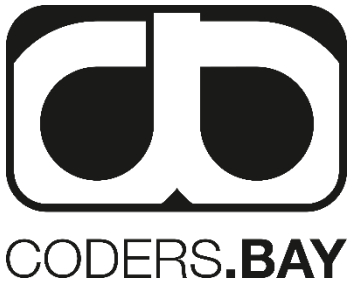
IF-ELSE-ANWEISUNG

CODERS.BAY

Eine bedingt auszuführende Anweisung darf durchaus wiederum vom **if-** bzw. **if-else-Typ** sein, so dass sich mehrere, hierarchisch geschachtelte Fälle unterscheiden lassen.



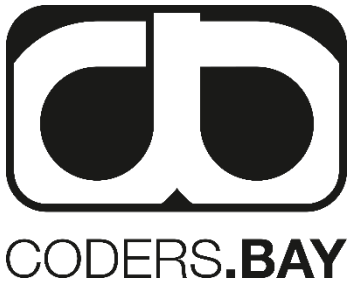
IF-ELSE-ANWEISUNG BEISPIEL



Quellcode	Ein- und Ausgabe

`int a=5, b=6, c=7;`
`if(a>b)`
`{`
`  a=b;`
`}`
`else`
`{`
`  if(a<c)`
`  {`
`  a=c;`
`  }`
`}`
	Da a gleich 5 ist und b gleich 6, ergibt sich für den Ausdruck a < b (5<6) der boolesche Wert true. Da die Bedingung wahr ist, wird der Anweisungsblock direkt unter der Bedingung durchlaufen und nicht der Anweisungsblock unter dem else-Zweig.

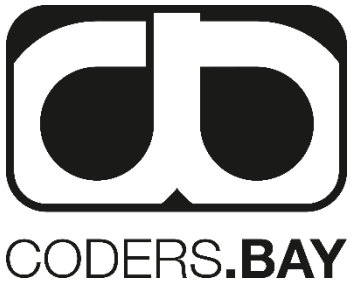
IF-ELSE-ANWEISUNG



Diesen PAP mit „sukzessiver Restaufspaltung“ realisiert z. B. eine if-else-Konstruktion nach diesem Muster:

```
if (Logischer Ausdruck 1)
     Anweisung 1
else if (Logischer Ausdruck 2)
     Anweisung 2
     .    .    .
else if (Logischer Ausdruck k)
     Anweisung k
else
     Default-Anweisung
```

IF-ELSE-ANWEISUNG

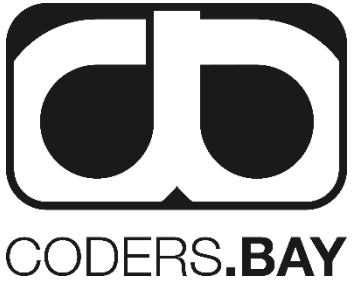


Wenn alle logischen Ausdrücke den Wert **false** annehmen, dann wird die **else-Klausel** zur letzten **if-Anweisung** ausgeführt.

Gerade wurde eine zusammengesetzte Anweisung mit spezieller Bauart als Beispiel vorgeführt. Es ist z. B. keinesfalls allgemein vorgeschrieben, dass alle beteiligten **if-Anweisungen** eine **else-Klausel** haben müssen.

Die Bezeichnung *Default-Anweisung* in der vorigen Syntaxdarstellung, die bei einer Mehrfallunterscheidung gegenüber einer verschachtelten **if-else-Konstruktion** zu bevorzugen, ist, wenn die Fallzuordnung über die verschiedenen Werte eines Ausdrucks (z. B. vom Typ **int**) erfolgen kann.

DANGLING-ELSE-PROBLEM



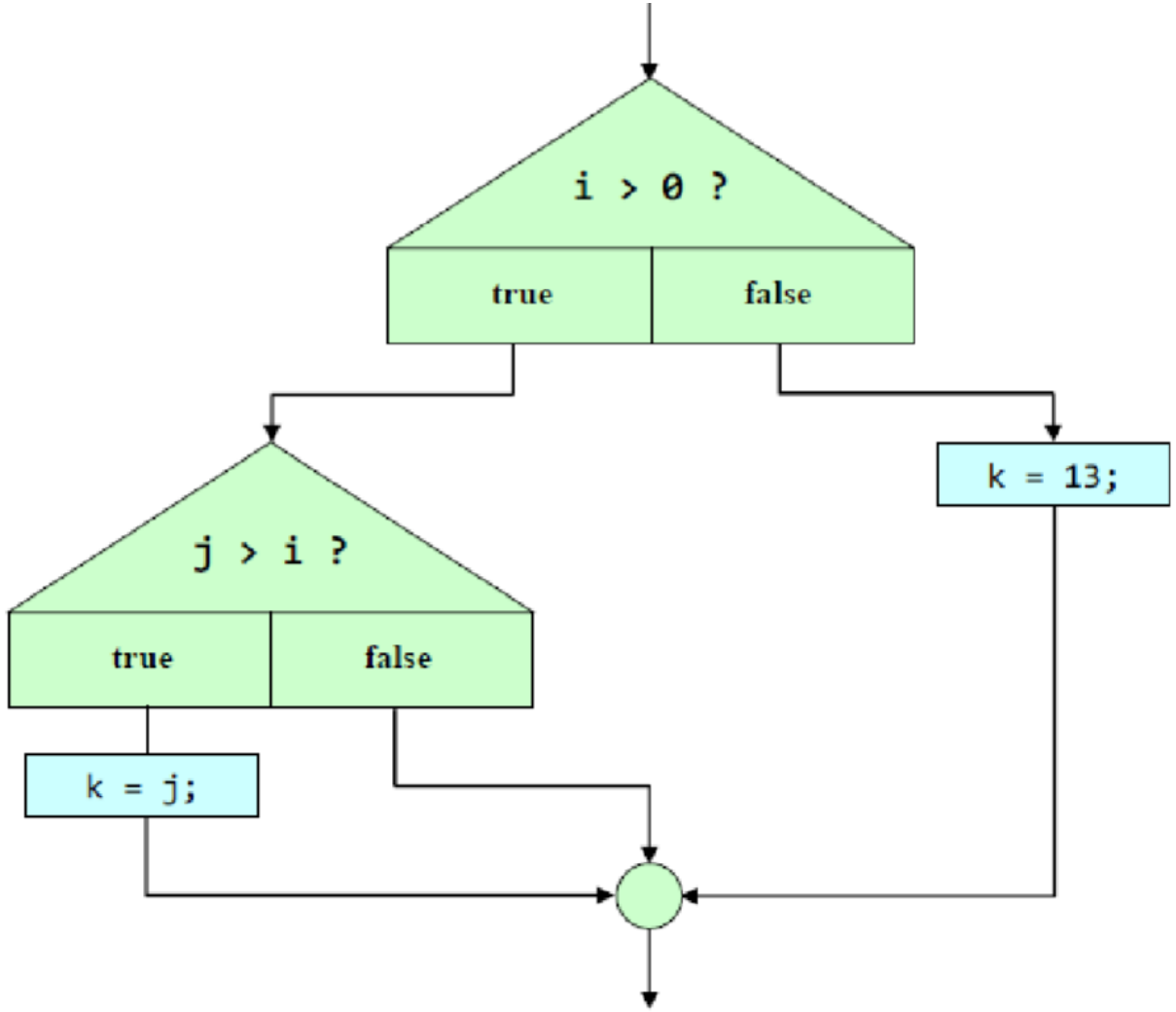
Beschreibung

Beim Schachteln von bedingten Anweisungen kann es zum **dangling-else-Problem** kommen, wobei ein Missverständnis zwischen Compiler und Programmierer hinsichtlich, der Zuordnung einer **else-Klausel** besteht. Im folgenden Code-Fragment lassen die Einrücktiefen vermuten, dass der Programmierer die **else-Klausel** auf die erste **if-Anweisung** bezogen zu haben glaubt.

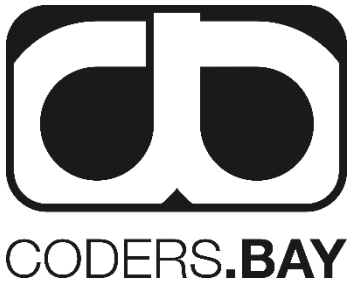
DANGLING-ELSE-PROBLEM

CODERS.BAY

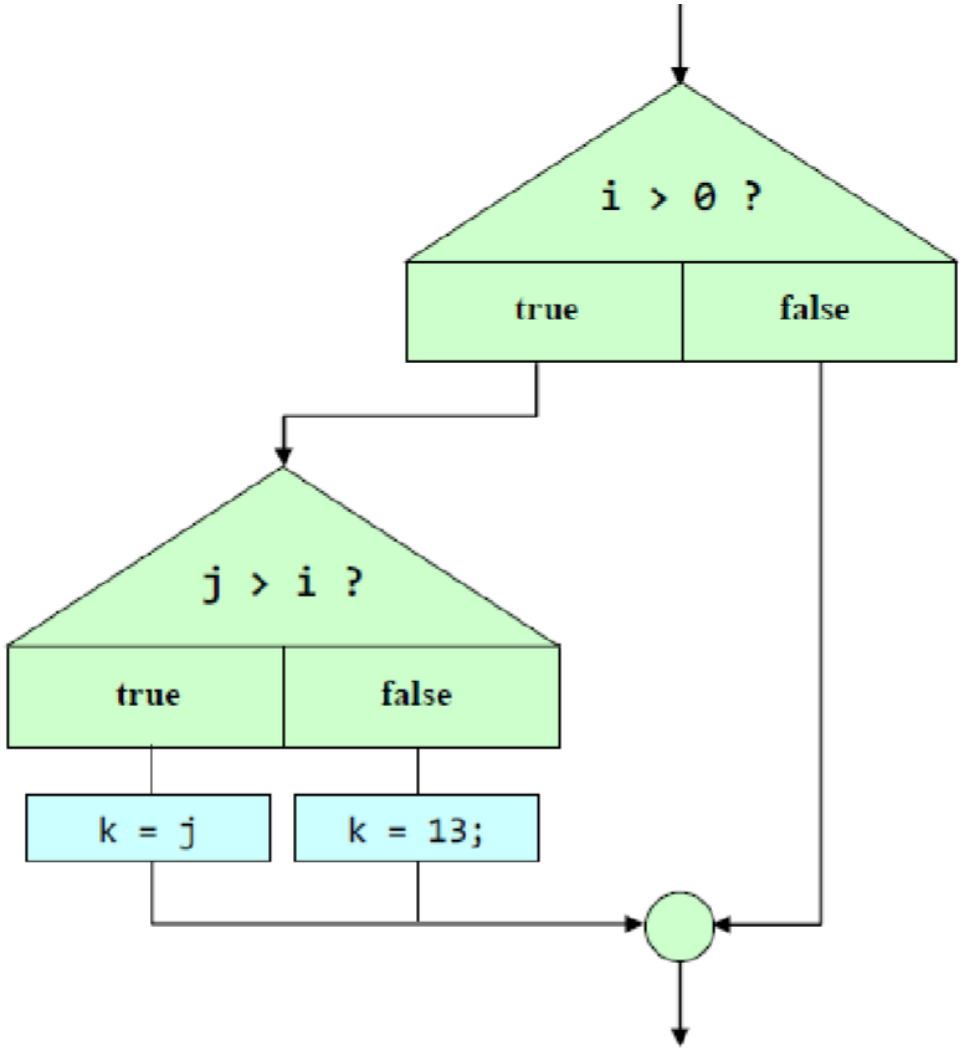
```
if (i > 0)
    if (j > i)
        k = j;
else k = 13;
```



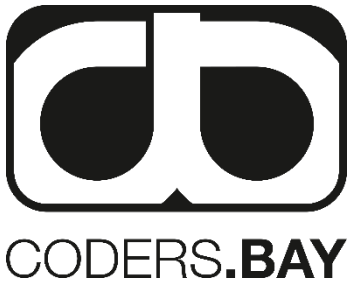
DANGLING-ELSE-PROBLEM



Der Compiler ordnet eine **else-Klausel** jedoch dem in Aufwärts-richtung nächstgelegenen if zu, das nicht durch Blockklammern({}) block clamps abgeschottet ist und noch keine **else-Klausel** besitzt. Im Beispiel bezieht er die **else-Klausel** also auf die *zweite* **if-Anweisung**, so dass de facto folgender Programmablauf resultiert:



BLOCKKLAMMERN ODER LEERE ANWEISUNG

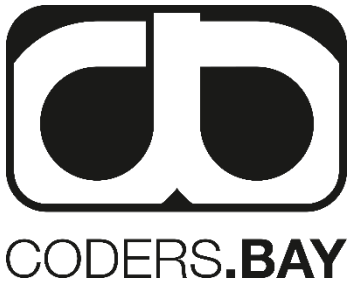


Bei i <= 0 geht der Programmierer fest vom neuen k-Wert 13 aus, was beim tatsächlichen Programmablauf keinesfalls garantiert ist.

```
if (i > 0)
      {if (j > i)
            k = j;}
else
      k = 13;
```

Mit Hilfe von Blockklammern kann man die gewünschte Zuordnung erzwingen:

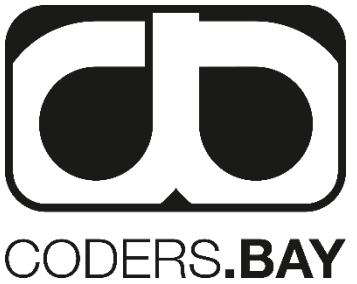
BLOCKKLAMMERN ODER LEERE ANWEISUNG



Alternativ könnte man auch dem zweiten **if** eine **else-Klausel** spendieren, und dabei eine leere Anweisung verwenden:

```
if (i > 0)
    if (j > i)
        k = j;
    else
        ;
else
    k = 13;
```

IF-ELSE-ANWEISUNG KONDITIONALOPERATOR

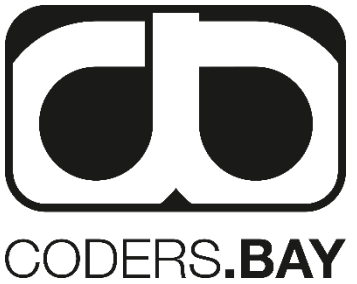


Gelegentlich kommt als Alternative zu einer simplen **if-else-Anweisung**, die zur Berechnung eines Wertes bedingungsabhängig zwei unterschiedliche Ausdrücke benutzt, der Konditionaloperator in Frage, z. B.:

if-else-Anweisung	Konditionaloperator

`double arg = 3.0, d;`
`if (arg > 1)`
`    d = arg * arg;`
`else`
`    d = arg;`
`System.out.println("d="+d); //d=9.0`
	`double arg = 3.0, d;`
	`d = (arg > 1) ? arg * arg : arg;`
	`System.out.println("d="+d); //d=9.0`

IF-ELSE-ANWEISUNG BEISPIEL



Quellcode

```
int days, month=2;
boolean isLeapYear=true;
if (month == 4)
    days = 30;
else if (month == 6)
    days = 30;
else if (month == 9)
    days = 30;
else if (month == 11)
    days = 30;
else if (month == 2)
    if (isLeapYear) // Sonderbehandlung im Fall eines Schaltjahrs
        days = 29;
    else
        days = 28;
else
    days = 31;
System.out.println("Days = "+ days);
```

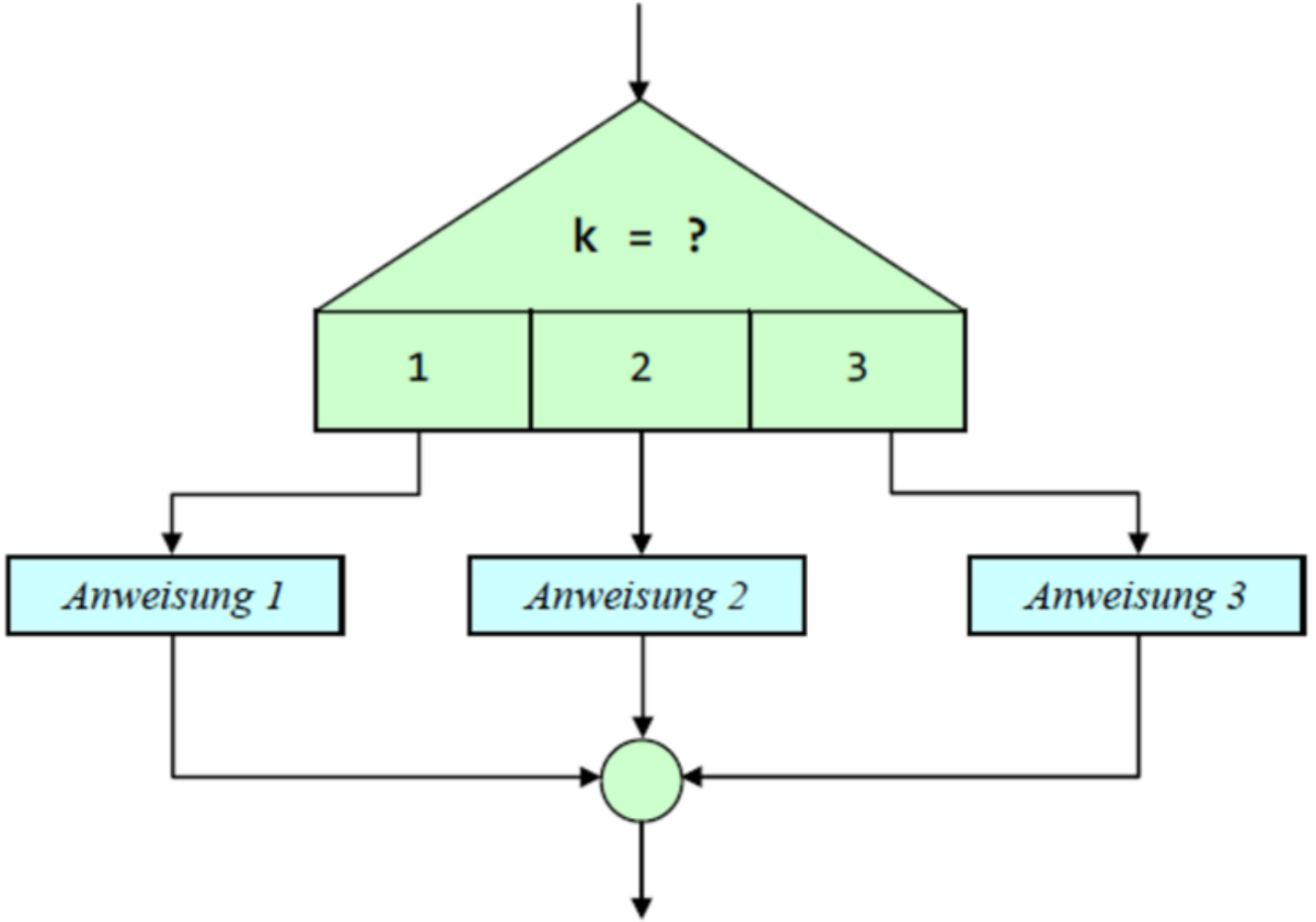
Ein- und Ausgabe

Days = 29

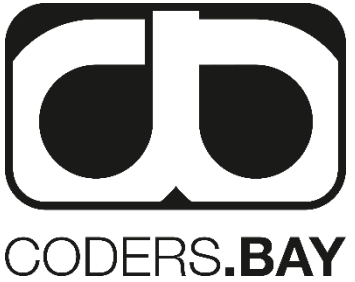
SWITCH-ANWEISUNG PROGRAMMABLAUFPLAN

CODERS.BAY

Wenn eine Fallunterscheidung mit mehr als zwei Alternativen in Abhängigkeit vom Wert *eines* Ausdrucks vorgenommen werden soll dann ist eine **switch-Anweisung** weitaus handlicher als eine verschachtelte **if-else-Konstruktion**.



SWITCH-ANWEISUNG

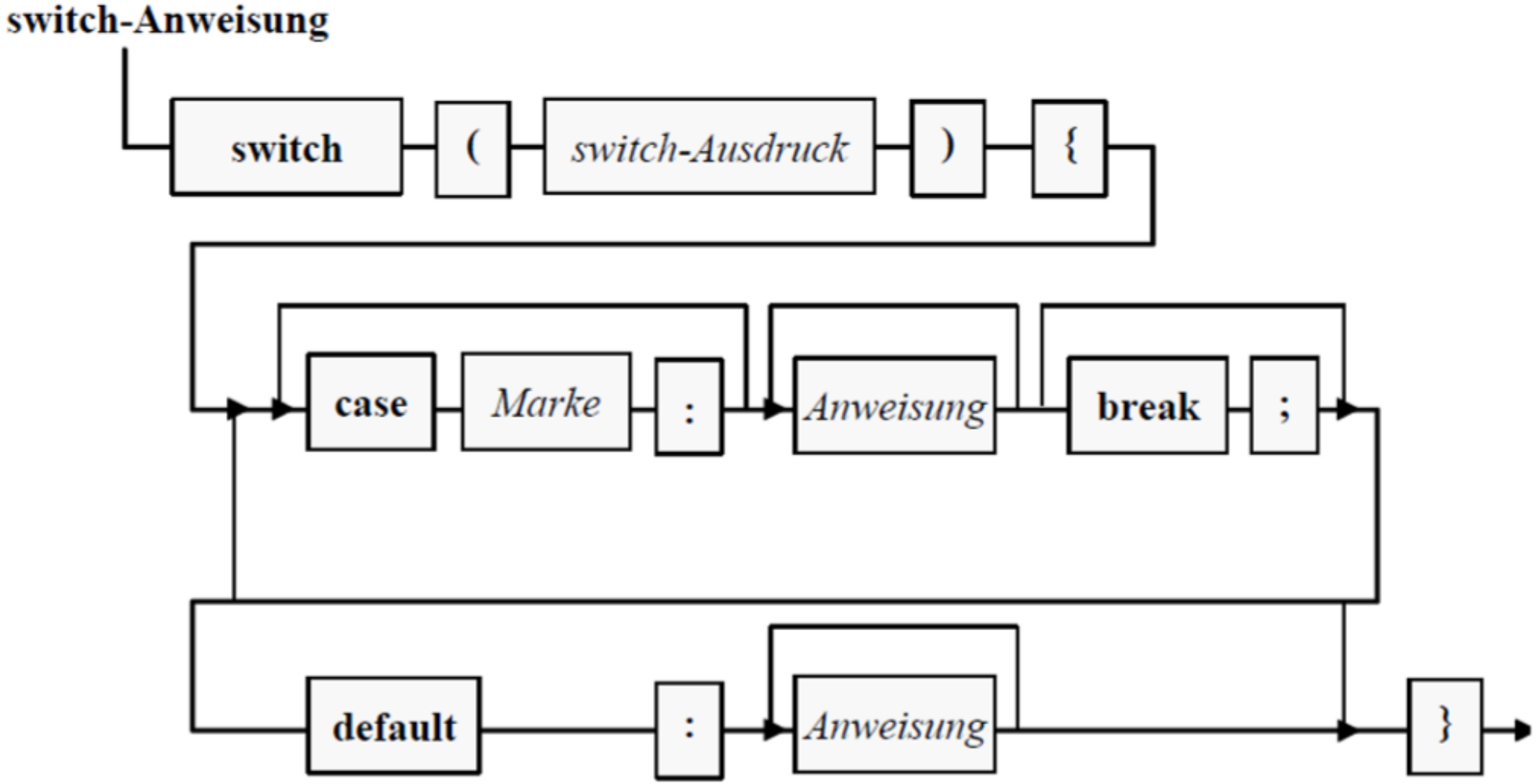


In Bezug auf den Datentyp des steuernden Ausdrucks ist Java recht flexibel und erlaubt:

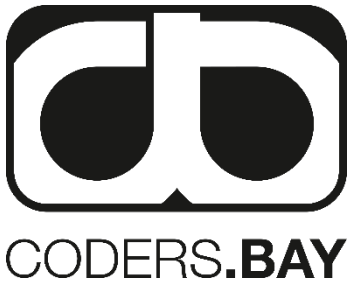
- Integrale primitive Datentypen: **byte**, **short**, **char** oder **int** (nicht long!)
- Zeichenfolgen (Objekte der Klasse **String**)
- Aufzählungstypen (siehe unten)
- Verpackungsklassen (siehe unten) für integrale primitive Datentypen: **Byte**, **Short**, **Character** oder **Integer** (nicht Long!)

SWITCH-ANWEISUNG SYNTAXDIAGRAMM

CODERS.BAY



SWITCH-ANWEISUNG BEISPIEL



Einfaches und sinnfreies Exemplar zur Erläuterung der Syntax:

Quellcode	Ausgabe

```
class Prog1 {
    public static void main(String[] args) {
// Variablendeklaration mit Initialisierung (Kurzform)
        int wert=5;
        switch(wert)
        {
            case 2: break;
            case 3: //Mach irgendwas
                    break;
            case 4:
            case 5: System.out.println("Wert ist gleich 4 oder 5"); break;
            default:   // Mach irgendwas standardmäßiges
        }
    }
}
```

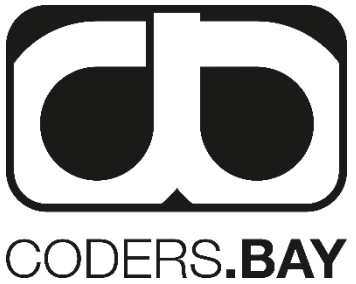
```
Der Wert
ist gleich
4 oder 5
```

SWITCH-ANWEISUNG

Als **case-Marken** sind konstante Ausdrücke erlaubt, deren Wert schon der Compiler ermitteln kann (z.B. Literale, Konstanten oder mit konstanten Argumenten gebildete Ausdrücke). Außerdem muss der Datentyp einer Marke kompatibel zum Typ des **switch-Ausdrucks** sein.

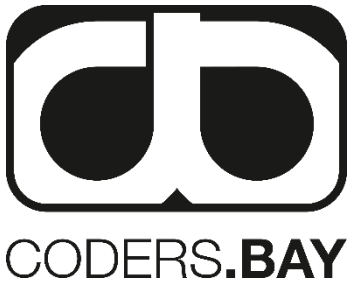
Stimmt beim Ablauf des Programms der Wert des **switch-Ausdrucks** mit einer case-Marke überein, dann wird die zugehörige Anweisung ausgeführt, ansonsten (falls vorhanden) die **default-Anweisung**.

SWITCH-ANWEISUNG



Nach der Ausführung einer „angesprungenen“ Anweisung wird die **switch-Konstruktion** jedoch nur dann verlassen, wenn der Fall mit einer **break-Anweisung** abgeschlossen wird. Ansonsten werden auch noch die Anweisungen der nächsten Fälle (ggf. inkl. **default**) ausgeführt, nach unten entweder durch eine **break-Anweisung** gestoppt wird, oder die **switch-Anweisung** endet. Mit dem etwas gewöhnungsbedürftigen **Durchfall-Prinzip** kann man für geeignet angeordnete Fälle mit wenig Schreibaufwand kumulative Effekte kodieren, aber auch ärgerliche, Programmierfehler durch vergessene **break-Anweisungen** produzieren.

SWITCH-ANWEISUNG



Soll für mehrere Werte des **switch-Ausdrucks** dieselbe Anweisung ausgeführt werden, setzt man die zugehörigen **case-Marken** hintereinander und lässt die Anweisung auf die letzte Marke folgen. Leider gibt es keine Möglichkeit, eine *Serie* von Fällen durch Angabe der Randwerte (z. B. *von a bis z*) festzulegen.