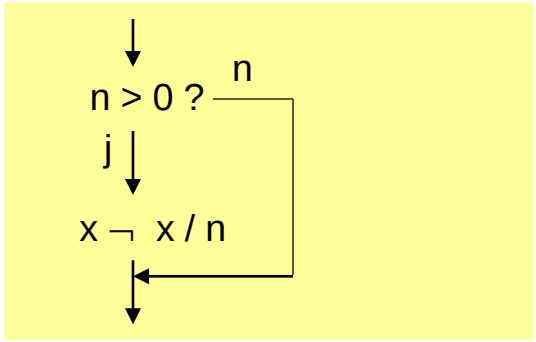
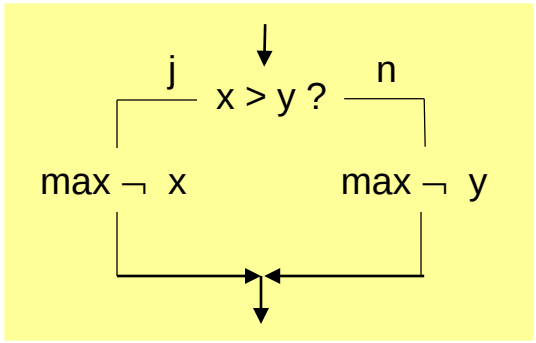
3. Verzweigungen

If-Anweisung




```
if (n > 0) x = x / n;
```

ohne else-Zweig




```
if (x > y)
   max = x;
else
   max = y;
```

mit else-Zweig

Syntax

IfStatement = "**if**" "**(**" Expr "**)**" Statement ["**else**" Statement].

Anweisungsblöcke

Wenn if-Zweig oder else-Zweig aus mehr als 1 Anweisung bestehen, müssen sie durch { ... } geklammert werden.

```
Statement = Assignment | IfStatement | Block | ... .
Block = "{" { Statement } "}".
```

Zeichen im Quelltext

Metazeichen der Grammatik, das Wiederholung ausdrückt

Beispiel

```
if (x < 0) {
   negNumbers++;
   Out.print(-x);
} else {
   posNumbers++;
   Out.print(x);
}
```

Es wird empfohlen, die Blockklammern auch dann zu schreiben, wenn im if-Zweig oder else-Zweig nur eine einzige Anweisung steht.

Vergleichsoperatoren

Vergleich zweier Werte liefert wahr (*true*) oder falsch (*false*)

	Bedeutung	*Beispiel*
==	gleich	x == 3
!=	ungleich	x != y
>	größer	4 > 3
<	kleiner	x+1 < y-1
>=	größer oder gleich	x >= y
<=	kleiner oder gleich	x <= y

arithmetische Operatoren binden stärker als Vergleichsoperatoren

Vergleiche werden z.B. in If-Anweisungen verwendet

```
if (x == 0) Out.println("x is zero");
```

Achtung: "=" ist in Java kein Vergleich, sondern eine Zuweisung

```
if (x = 0) Out.println("x is zero");  // Compiler meldet einen Fehler!
```

Einrückungen

- erhöhen die Lesbarkeit (machen Programmstruktur besser sichtbar)
- Einrückungstiefe: 1 Tabulator oder 2 Leerzeichen

```
if (n != 0)
   x = x / n;
```

```
if (x > y)
   max = x;
else
   max = y;
```

```
if (x < 0) {
   negNumbers++; Out.print(-x);
} else {
   posNumbers++; Out.print(x);
}
```

Kurze If-Anweisungen können auch in *einer* Zeile geschrieben werden

```
if (n != 0) x = x / n;
```

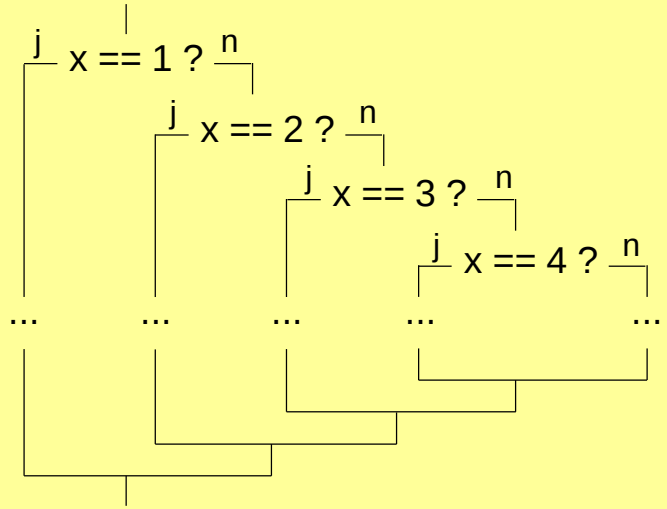
```
if (x > y) max = x; else max = y;
```

if-else-Kaskaden

Häufiges Muster, wenn man mehr als 2 Alternativen prüfen möchte

Beispiel

```
if (x == 1) {
    ...
} else if (x == 2) {
    ...
} else if (x == 3) {
    ...
} else if (x == 4) {
    ...
} else {
    ...
}
```



Alternativen werden sequentiell nacheinander geprüft

Dangling Else

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

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

Mehrdeutigkeit! Zu welchem if gehört das else?

Regel: else gehört immer zum unmittelbar vorausgegangenen if.

Wenn man das nicht will, muss man die Anweisung so schreiben:

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

Darum besser immer Blockklammern setzen => vermeidet Mehrdeutigkeit

Programmbeurteilung

Welches der beiden Programme ist besser?

```
if (a > b)
    if (a > c)
        max = a;
    else
        max = c;
else
    if (b > c)
        max = b;
    else
        max = c;
```

```
max = a;
if (b > max) max = b;
if (c > max) max = c;
```

Was heißt "besser"?

- Kürze: das 2. Programm ist kürzer
- Effizienz:
  - 1. Programm braucht immer 2 Vergleiche und 1 Zuweisung
  - 2. Programm braucht immer 2 Vergleiche und im Schnitt 2 Zuweisungen
- Lesbarkeit?
- Erweiterbarkeit?

Assertionen bei If-Anweisungen

```
if (condition)
    // condition
    ...
else
    // ! condition
    ...
```

diese Assertion sollte man immer hinschreiben oder zumindest im Kopf bilden

Beispiel: Maximum dreier Zahlen berechnen

```
int a, b, c, max;
a = In.readInt(); b = In.readInt(); c = In.readInt();

if (a > b)  /* a>b */
    if (a > c)  /* a>b && a>c */ max = a;
    else  /* a>b && c>=a */ max = c;
else  /* b>=a */
    if (b > c)  /* b>=a && b>c */ max = b;
    else  /* b>=a && c>=b */ max = c;

Out.println(max);
```

3. Verzweigungen

Zusammengesetzte Vergleiche

&& Und-Verknüpfung

x	y	x && y
true	true	true
true	false	false
false	true	false
false	false	false

|| Oder-Verknüpfung

x	y	x || y
true	true	true
true	false	true
false	true	true
false	false	false

! Nicht-Verknüpfung

x	!x
true	false
false	true

Beispiel

```
if (a >= 0 && a <= 10 || a >= 100 && a <= 110) b = a;
```

Vorrangregeln

! bindet stärker als && bindet stärker als ||

Vorrangregeln können durch Klammerung umgangen werden

```
if (a > 0 && (b == 1 || b == 7)) ...
```

Kurzschlussauswertung

Zusammengesetzter Vergleich wird abgebrochen, sobald das Ergebnis feststeht

```
if ( a != 0  && b / a > 0 ) x = 0;
```

wenn *false*, ist gesamter Ausdruck *false*

äquivalent zu

```
if (a != 0)
   if (b / a > 0) x = 0;
```

```
if ( a == 0  || b / a > 0 ) x = 1;
```

wenn *true*, ist gesamter Ausdruck *true*

```
if (a == 0)
   x = 1;
else if (b / a > 0)
   x = 1;
```

Vorteile

- effizienter in der Ausführung
- erlaubt, einen Ausdruck durch einen anderen zu "schützen"

Negation zusammengesetzter Vergleiche

Regeln von DeMorgan

Augustus De Morgan,
britischer Mathematiker, 1806-1871

```
! (a && b)   Û   ! a || ! b
! (a || b)   Û   ! a && ! b
```

Diese Regeln helfen beim Bilden von Assertionen

```
if (x >= 0 && x < 10) {
    ...
} else { // ! (x >= 0 && x < 10)
    ...
}
```

⇨ ! (x >= 0) || ! (x < 10)

⇨ x < 0 || x >= 10

Beispiele zu De Morgan

! (a && b)	Û	! a || ! b
! (a || b)	Û	! a && ! b

```
if (a != 0 && b / a > 0) {
    ...
} else { // ! (a != 0 && b / a > 0)
    ...
}
```

⇨ ! (a != 0) || ! (b / a > 0)

⇨ a == 0 || b / a <= 0

```
if (empty || i < size) {
    ...
} else { // ! (empty || i < size)
    ...
}
```

⇨ ! empty && ! (i < size)

⇨ ! empty && i >= size

```
if (a <= x && x <= b || c < x && x < d) {
    ...
} else { // ! (a <= x && x <= b || c < x && x < d)
    ...
}
```

⇨ ! (a <= x && x <= b) && ! (c < x && x < d)

⇨ (! (a <= x) || ! (x <= b)) && (! (c < x) || ! (x < d))

⇨ (x < a || x > b) && (x <= c || x >= d)

3. Verzweigungen

Datentyp boolean

George Boole: Mathematiker, 1815-1864

Datentyp wie *int* mit den beiden Werten *true* und *false*

Beispiele

```
boolean p, q;
p = false;
q = x > 0;
p = p || q && x < 10;
```

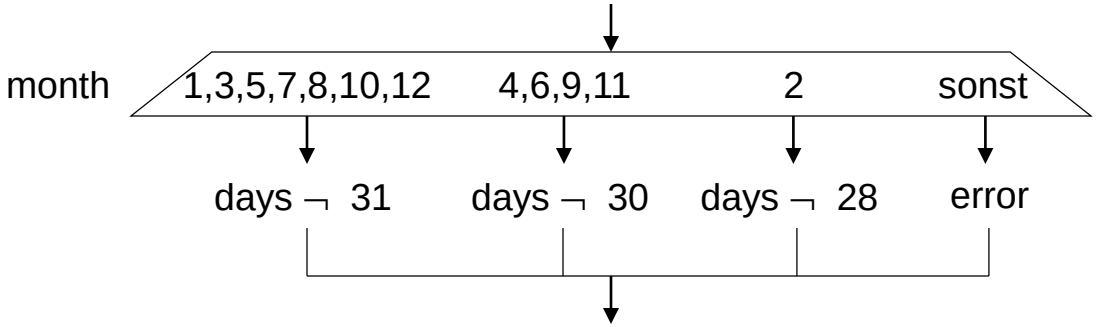
Beachte

- Jeder Vergleich liefert einen Wert vom Typ *boolean*
- Boolesche Werte können mit &&, || und ! verknüpft werden
- Boolesche Werte können in *boolean*-Variablen abgespeichert werden ("flags")
- Namen für *boolean*-Variablen sollten mit Adjektiv beginnen: equal, full

3. Verzweigungen

Switch-Anweisung

Mehrwegverzweigung



In Java

```
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 = 28; break;
   default:
      Out.println("error");
}
```

Semantik der Switch-Anweisung

Switch-Ausdruck

```
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 = 28; break;
   default:
      Out.println("error");
}
```

Break-Anweisung

- springt ans Ende der Switch-Anweisung
- wenn *break* fehlt, läuft Programm über nächste Case-Marke weiter (häufige Fehlerursache!!)

Semantik

1. berechne Switch-Ausdruck
2. springe zur passenden Case-Marke
   - wenn keine passt, springe zu default
   - wenn kein default angegeben, springe ans Ende der Switch-Anweisung

Bedingungen

1. Switch-Ausdruck ganzzahlig, *char* oder *String*
2. Case-Marken müssen Konstanten sein
3. ihr Typ muss zum Typ des Switch-Ausdrucks passen
4. Case-Marken müssen voneinander verschieden sein

Syntax der Switch-Anweisung


```
Statement       = Assignment | IfStatement | SwitchStatement | BreakStatement | ... | Block.

SwitchStatement = "switch" "(" Expr ")" "{" {LabelSeq  StatementSeq} "}".
LabelSeq        = Label {Label}.
StatementSeq    = Statement {Statement}.
Label           = "case" ConstExpr ":"
                  | "default" ":".

BreakStatement  = "break" ";"
```

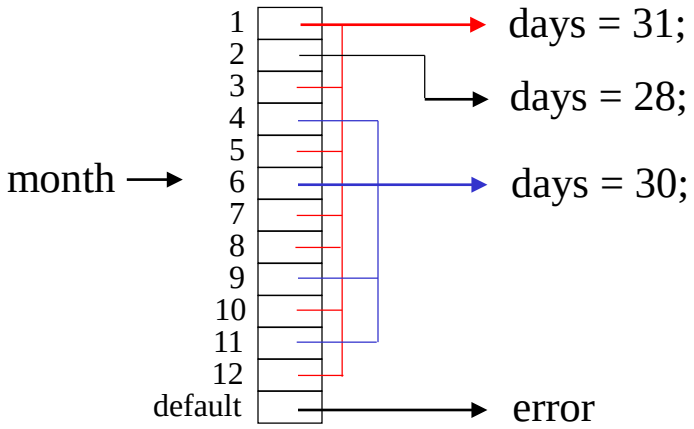
Unterschied zwischen If und Switch

```
if (month==1 || month==3 || month==5
   || month==7 || month==8 || month==10
   || month==12)
   days = 31;
else if (month==4 || month==6
   || month==9 || month==11)
   days = 30;
else if (month==2)
   days = 28;
else Out.println("error");
```

prüft Bedingungen sequentiell

```
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 = 28; break;
   default:
      Out.println("error");
}
```

benutzt Sprungtabelle



Bewertung der Switch-Anweisung

\+ schneller

\- speicheraufwändiger, z.B. bei
case 1: ... case 10000:

3. Verzweigungen

Bedingte Ausdrücke

Häufiges Muster

```
if (a > b) {
   max = a;
} else {
   max = b;
}
```

kann geschrieben werden als

```
max = a > b ? a : b;
```

- kürzer, aber nicht schneller
- nicht unbedingt lesbarer

Syntax

```
CondExpr = BoolExpr "?" Expr ":" Expr.
```

können Ausdrücke sein, nicht nur Variablen

Kann als Teilausdruck verwendet werden

```
size = c * (a > b ? a : b) + 1;
```

```
if (a > b) {
   size = c * a + 1;
} else {
   size = c * b + 1;
}
```