Operatoren, Schleifen, Verzweigungen

Inhalt - Überblick

Rückblick

- Datentypen: boolean, char
- UNICODE-Kodierung
- Rechnen mit Binärzahlen
- Ganzzahlige Datentypen: byte, short, int, long
- Fließkommazahlen: float, double

- Variablen
- Arrays

Ausdrücke und Operatoren

Ausdrücke und Operatoren

- Ausdrücke sind die kleinsten ausführbaren Einheiten des Programms
- Beispiele:

```
// Variablenzuweisung
x = 5;

// Ausdruck mit dem Additions-Operator
x = x + 6;
```

- Ein Ausdruck besteht aus mindestens einem Operator (rot) und einem oder mehreren Operanden (blau)

[Class10.java]

Operatoren-Typen:

- Arithmetische Operatoren (+, -, /, %, usw.)
- Relationale Operatoren (==, !=, <, >, <=, >=, usw.)
- Logische Operatoren (!, &&, ||, &, ^, usw.)
- Bitweise Operatoren (~, >>, usw.)
- Zuweisungs-Operatoren (=, +=, *=, usw.)
- Sonstige (a ? b : c, usw.)

Ausdrücke haben einen Rückgabewert, der durch die Anwendung des Operators entsteht

```
5 + 6        5 * 6        state = !true
```

[Class10.java]

Eigenschaften von Operatoren

Einstellige und zweistellige Operatoren, z. B.

- Einstellig:
- Zweistellig:

Bindungsregeln, z. B.

- „Punkt vor Strich“
- ++ vor *
- ! vor ==
- usw.

Siehe: www.javabuch.de Kapitel 6.8 (Vorrangregeln)

Auswertungsreihenfolge

Erzwingen der Auswertungsreihenfolge durch runde Klammern, z. B.:

```
System.out.println(  5 + 6  * 3);
System.out.println( (5 + 6) * 3);
System.out.println(  1 << 1 + 2 );
System.out.println( (1 << 1)+ 2 );
```

Der linke Operand wird vollständig vor dem rechten Operanden ausgeführt:

```
int i = 2;
System.out.println( (i=3) * i );
```

[Class10.java]

Arithmetische Operatoren

- Addition: +
- Subtraktion: -
- Multiplikation: *
- Division: /
- Modulo / Restwertoperator: %
- Prä- und Postinkrement: ++
- Prä- und Postdekrement: --
- Vorzeichen: + oder -

siehe

[Class11.java]

Relationale Operatoren

- Vergleich von Ausdrücken; Ergebnis ist ein boolescher Wert (true / false)
- Gleich: ==
- Ungleich: !=
- Kleiner: <
- Größer: >
- Kleiner gleich: <=
- Größer gleich: >=

Vorsicht bei Strings (und anderen Referenztypen)

siehe

[Class12.java]

Boolesche Algebra

„und“
Konjunktion

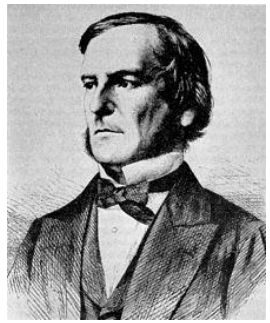
∧	0	1
0	0	0
1	0	1

„oder“
Disjunktion

∨	0	1
0	0	1
1	1	1

„nicht“
Negation

	¬
0	1
1	0



Georg Boole

0: false / falsch
1: true / wahr

Siehe: http://de.wikipedia.org/wiki/Boolesche_Algebra

[Arbeiten mit Wahrheitswertetafeln]

Oder (OR) vs. Exklusiv-Oder (XOR)

OR	0	1
0	0	1
1	1	1

XOR	0	1
0	0	1
1	1	0

- Exklusiv-Oder (XOR): „Entweder-Oder“

Logische Operatoren

- Verknüpfung von booleschen Werten (true / false)
- Logische Negation: !
- Logisches Und: && (mit short circuit evaluation)
- Logisches Oder: || (mit short circuit evaluation)
- Logisches Und: & (ohne short circuit evaluation)
- Logisches Oder: | (ohne short circuit evaluation)
- Exklusiv-Oder: ^

siehe

[Class13.java]

Bitweise Operatoren

- Operatoren auf Ebene der Bit-Darstellung einer Zahl
- Einerkomplement: ~
- Bitweises Oder: |
- Bitweises Und: &
- Bitweises Exklusiv-Order: ^
- Rechtsschieben mit Vorzeichen: >>
- Rechtsschieben ohne Vorzeichen: >>>
- Linksschieben: <<

siehe

[Class14.java]

Zuweisungsoperatoren

- Einfache Zuweisung: =
- Additionszuweisung: +=
- Subtraktionszuweisung: -=
- Multiplikationszuweisung: *=
- Divisionszuweisung: /=
- Modulozuweisung: %=
- und einige weitere…

siehe

[Class15.java]

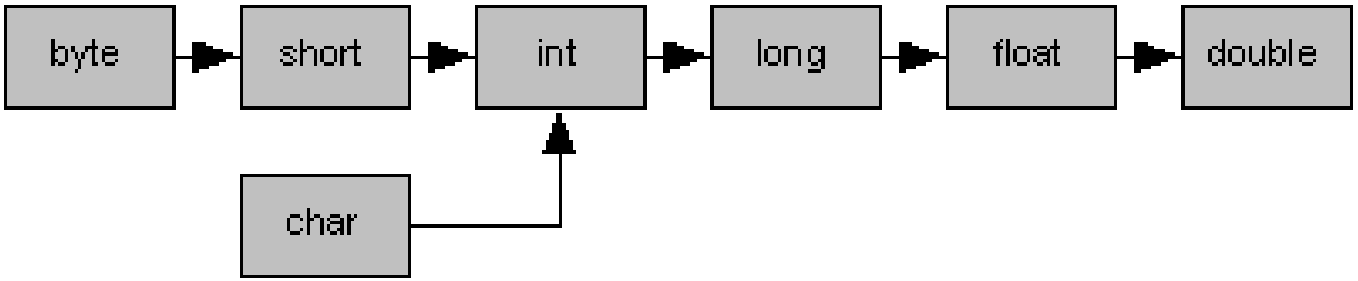
- Wenn a wahr (true) ist dann b auswerten sonst c:

```
a ? b : c
```

- Cast: Explizite Typumwandlung des Ausdrucks a zu type:

```
(type) a
```

Vergleiche:



siehe

[Class16.java]

Arithmetische Operatoren

- Zwei Strings a und b werden zu einem String zusammengehängt und c zugewiesen:

```
c = a + b
```

- Achtung: Eine Auto-Konvertierung zu String ist für fast alle Datentypen vorhanden. Vorsicht, z. B. bei

```
System.out.println("Summe von 2 + 3 = " + 2 + 3);
System.out.println("Summe von 2 + 3 = " + (2 + 3));
```

[Class17.java]

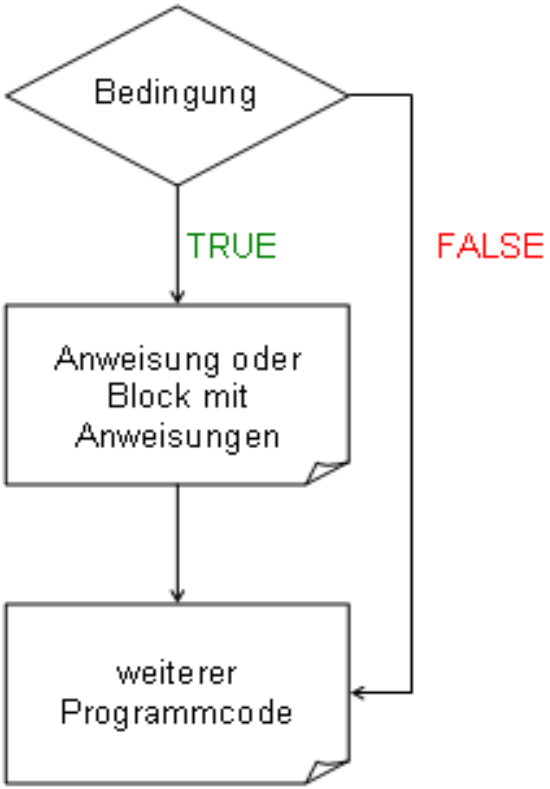
Verzweigungen

Verzweigungen

- Bestimmte Programmteile werden nur ausgeführt, wenn eine Bedingung zur Laufzeit erfüllt ist
- Die Bedingung wird zu einem logischen Datentyp (boolean) ausgewertet (true oder false)
- Java umfasst diese Verzweigungsanweisungen
  - **`if`**
  - **`if-else`**
  - **`switch`**

[Class18.java]

Ausführen, wenn eine Bedingung wahr ist:



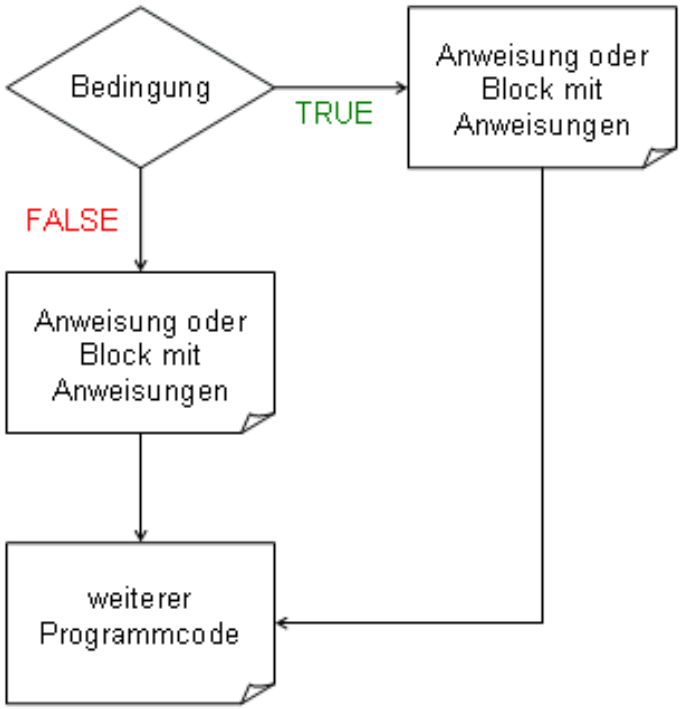

```
if (bedingung){
    anweisung;
}
```

Die Anweisung wird ausgeführt, wenn die Bedingung wahr (true) ist.

```
if (zahl1 > zahl2)
    System.out.println("Zahl 1 größer als Zahl 2");
```

[Class18.java]

Ausführen, wenn eine Bedingung wahr ist; ansonsten einen anderen Block ausführen:




```
if (bedingung){
    anweisung 1;
} else {
    anweisung 2;
}
```

Die Anweisung 1 wird ausgeführt, wenn die Bedingung wahr (true) ist. Ansonsten wird die Anweisung 2 ausgeführt.

[Class18.java]

if und if-else Verzweigung

- Mehrere Anweisungen zusammenfassen:

```
if (zahl1 > zahl2) {
    System.out.println("Zahl 1 größer als Zahl 2");
    System.out.println("Mehr gibts nicht zu sagen");
} else {
    System.out.println("Zahl 2 größer als Zahl 1");
    System.out.println("Noch eine zweite Zeile");
}
```

- Guter Stil: Immer die { } schreiben, auch bei einzeiligen if-Anweisungen. Dies hilft, Fehler zu vermeiden.
- Dangling else (s. nächste Folie)

[Class18.java]

Dangling else

- Zwei verschachtelte if Blöcke, denen nur ein else gegenübersteht
- Scheinbare Mehrdeutigkeit: Wozu gehört else?

```
int a = 1;
int b = 2;

// Dangling else
if ( a == 1)
if ( b == 1)
     System.out.println("Step 1");
else
     System.out.println("Step 2");
```

[Class18.java]

Mehrfachverzweigung für byte, short, char und int

```
int anzahl = 10;

switch (anzahl) {
case 5:
     System.out.println("Es sind 5");
     break;
case 8:
     System.out.println("Es sind 8");
     break;
case 10:
     System.out.println("Es sind 10");
     break;
case 20:
     System.out.println("Es sind 20");
     break;
default:
     System.out.println("Es sind nicht 5, 8, 10 oder 20");
}
```

[Class19.java]

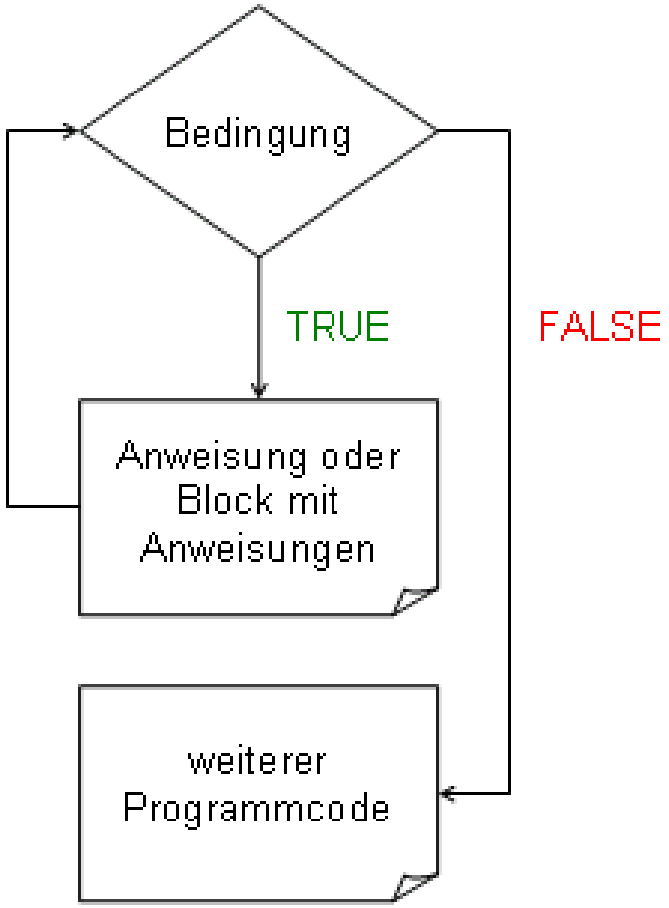
Schleifen

Schleifen wiederholen Anweisungen solange eine bestimmte Bedingung erfüllt ist.

Drei Schleifentypen

- **while**: Prüfung einer Bedingung vor der Schleifenausführung
- **do while**: Prüfung einer Bedingung nach der Schleifenausführung (nichtabweisend)
- **for**: (Zählschleife) nutzt einen Initialisierungsausdruck, eine Bedingung und einen Update-Ausdruck nach der Schleifenausführung

While Schleife

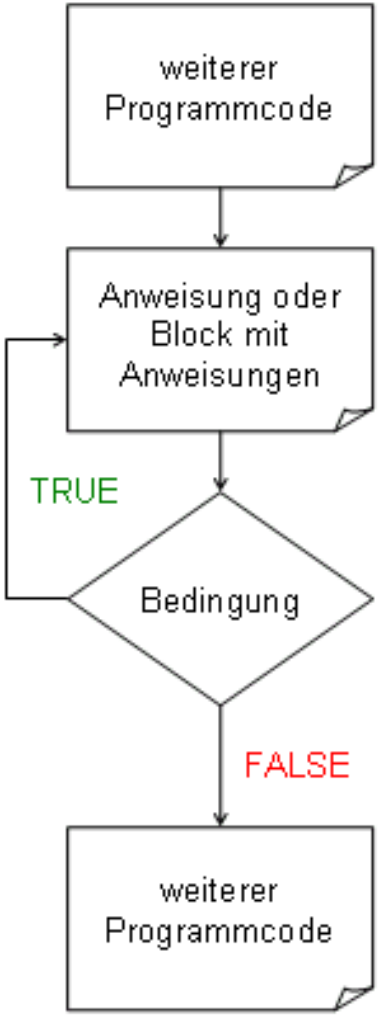



```
while (test){
    anweisung;
}
```

Zuerst wird der Ausdruck test geprüft. Ist dieser true, so wird die Schleife ausgeführt. Diese Prüfung erfolgt nach jedem Durchlauf des Schleifen-Blocks. Sobald test false ergibt, wird die Schleife nicht mehr durchlaufen. In diesem Fall wird der weitere Programmcode nach der Schleife ausgeführt.

[Class20.java]

Do while Schleife

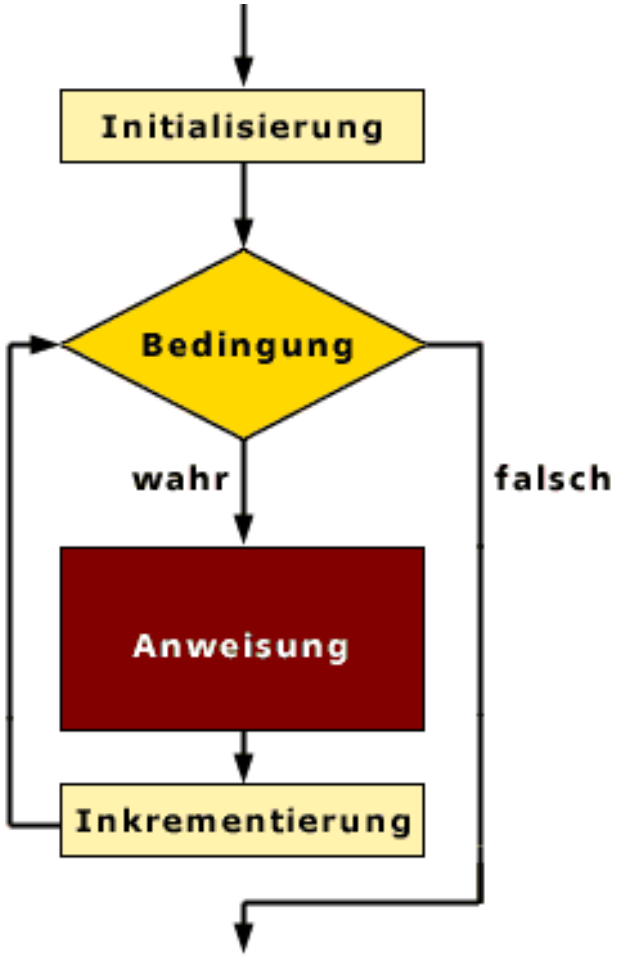



```
do{
    anweisung;
} while (test);
```

Zuerst wird der Schleifenblock (nach dem Wort do) ausgeführt. Am Ende dieser Ausführung wird geprüft, ob test true ergibt. Ist dies der Fall, so wird der Schleifenblock wiederum ausgeführt. Sobald test false ergibt, wird die Schleife nicht mehr ausgeführt und das Programm läuft weiter mit dem Code nach der Schleife.

[Class20.java]

For Schleife




```
for(init; test; update) {
    anweisung;
}
```

- init wird einmal vor dem Start der Schleife ausgeführt
- test (Bedingung) wird vor jedem Durchlauf getestet
- update wird nach jedem Durchlauf ausgeführt
- Alle drei Komponenten sind optional; eine fehlende Laufbedingung (test) wird durch true ersetzt, es entsteht eine Endlosschleife

[Class20.java]

Endlos-Schleife: Eine Schleife, deren Bedingung immer true ist.

Mit einer Schleife durch ein Array „laufen“ (Iteration):

```
int[] array = { 6, 5, 7, 8, 9, 4, 3, 2, 1 };

for (int a = 0; a < array.length; a++) {
    System.out.print("Array in der Stelle ");
    System.out.println(a + ": " + array[a]);
}
```

[Class20.java]

Zusätzliche Kontrolle für Schleifen mit

- **break**
  - beendet den Schleifenblock an der Stelle, an der die break-Anweisung steht
  - setzt das Programm mit der ersten Anweisung nach dem Schleifenblock fort
- **continue**
  - springt ans Ende des Schleifenblocks
  - setzt die Schleife mit der nächsten Iteration fort

[Class20.java]

Beispiele

Bedingungen konstruieren

```
// Liegt das eigegebene Datum zwischen 2000 und 2020?
if (2000 <= year && year <= 2020) {
    System.out.println("Das Jahr liegt im gültigen Bereich (2000-2020).");
}

// Liegt der eigegebene Monat zwischen Mai und Oktober?
if (5 <= month && month <= 10) {
    System.out.println("(1) Der Monat liegt im gültigen Bereich (05-10).");
}

// Sind beide Eingaben gültig?
// (schlechte Lösung, hier werden die Bedingungen von oben nochmals geprüft)
if (5 <= month && month <= 10 && 2000 <= year && year <= 2020){
    System.out.println("(1) Beide Eingaben sind gültig");
}
```

Schlechte Lösung

[Class 21.java]

Bedingungen konstruieren

```
// Liegt das eigegebene Datum zwischen 2000 und 2020?
boolean yearValid = 2000 <= year && year <= 2020;
if (yearValid) {
    System.out.println("(2) Das Jahr liegt im gültigen Bereich (2000-2020).");
}

// Liegt der eigegebene Monat zwischen Mai und Oktober?
boolean monthVaild = 5 <= month && month <= 10;
if (monthVaild) {
    System.out.println("(2) Der Monat liegt im gültigen Bereich (05-10).");
}

if (monthVaild && yearValid){
    System.out.println("Beide Eingaben sind gültig");
}
```

Bessere Lösung: Boolean-Varibalen nutzen

[Class 21.java]

Bedingungen konstruieren

```
// Wurde als Monat Februar eingegeben oder wurde der Mai 2010
// eingegeben?
// (Schlechte Lösung)
if (month == 2) {
    System.out.println("Es wurde ein Feburar oder Mai 2010 eingegeben.");
}
if (month == 5) {
    if (year == 2010) {
        System.out.println("Es wurde ein Feburar oder Mai 2010 eingegeben.
    }
}
```

Schlechte Lösung: If unnötigerweise verschachtelt

[Class 22.java]

Bedingungen konstruieren

```
// Wurde als Monat Februar eingegeben oder wurde der Mai 2010
// eingegeben?
// Bessere Lösung: Mit logischen Operatoren arbeiten
if (month == 2 || (month == 5 && year == 2010)) {
    System.out.println("Es wurde ein Feburar oder Mai 2010 eingegeben.");
}
```

Bessere Lösung: Logische Operatoren nutzen

[Class 22.java]

Schleifen beenden (break)

```
// Eine Schleife mit break abbrechen
do {
    Scanner sc = new Scanner(System.in);

    System.out.println("(1) Geben Sie eine Jahreszahl ein: ");
    int year = sc.nextInt();

    System.out.println("(1) Geben Sie den Monat ein: ");
    int month = sc.nextInt();

    // Wurde April 2011 eingegeben?
    if (month == 4 && year == 2011) {
        System.out.println("(1) Es wurde April 2011 eingegeben.");
        break;
    } else {
        System.out.println("(1) Es wurde NICHT April 2011 eingegeben.");
    }

} while (true);
```

[Class 23.java]

Schleifen beenden (boolean Variable)

```
// Eine boolean-Variable zum Schleifen-Abbruch nutzen
boolean condition = true;
do {
    Scanner sc = new Scanner(System.in);

    System.out.println("(2) Geben Sie eine Jahreszahl ein: ");
    int year = sc.nextInt();

    System.out.println("(2) Geben Sie den Monat ein: ");
    int month = sc.nextInt();

    // Wurde April 2011 eingegeben?
    condition = month == 4 && year == 2011;
    if (condition) {
        System.out.println("(2) Es wurde April 2011 eingegeben.");
    } else {
        System.out.println("(2) Es wurde NICHT April 2011 eingegeben.");
    }

} while (!condition);
```

[Class 23.java]

Schleifen verschachteln

```
// Auf ein zweidimensionales Array mit zwei
// verschachtelten Schleifen zugreifen

int[][] array = {{1,2,3}, {11,12,13}, {111,112}};

for (int outer = 0; outer < array.length; outer++){
    for (int inner = 0; inner < array[outer].length; inner++){
        System.out.print("Inhalt an Stelle ");
        System.out.println(outer +"/" + inner + ": " + array[outer][inner]);
    }
}
```

[Class 24.java]