

Vorlesung an der Berufsakademie Oldenburg

Unterrichtseinheit 14

Anweisungen



Dr. Dietrich Boles

- Grundlagen
- Beispiele
- Bereits bekannte Anweisungen
- do-Anweisung
- Aufgaben
- for-Anweisung
- Label-Anweisung
- break-Anweisung
- switch-Anweisung
- continue-Anweisung
- return-Anweisung

Definition:

Anweisung = Vorschrift zur Verarbeitung von Daten

Klassifikation:

- elementare Anweisungen
- zusammengesetzte Anweisungen
- Anweisungen zur Ablaufsteuerung

Imperative Programmierung
=
sequentielles Abarbeiten von Anweisungen

Bereits bekannte Anweisungen

- Deklarationsanweisung
- Zuweisungsanweisung
- Eingabe-Anweisungen
- Ausgabe-Anweisungen
- Leeraanweisung
- Blockanweisung
- if-Anweisung
- while-Anweisung

```
<do-Anweisung> ::= "do" <Anweisung>  
                    "while"  
                    "(" <boolescher Ausdruck> ")" ";"
```

Sinn und Zweck:

- Schleife mit mindestens einmaliger Ausführung einer Anweisung

Semantik:

- führe *Anweisung* aus
- berechne den Ausdruck
- falls Ausdruck `true` liefert, führe *Anweisung* erneut aus; ...
- falls Ausdruck `false` liefert, beende die do-Anweisung

Semantische Äquivalenz:

`<anweisung1> while (<bedingung>) <anweisung1>`

`<=>`

`do <anweisung1> while (<bedingung>);`

Beispiel:

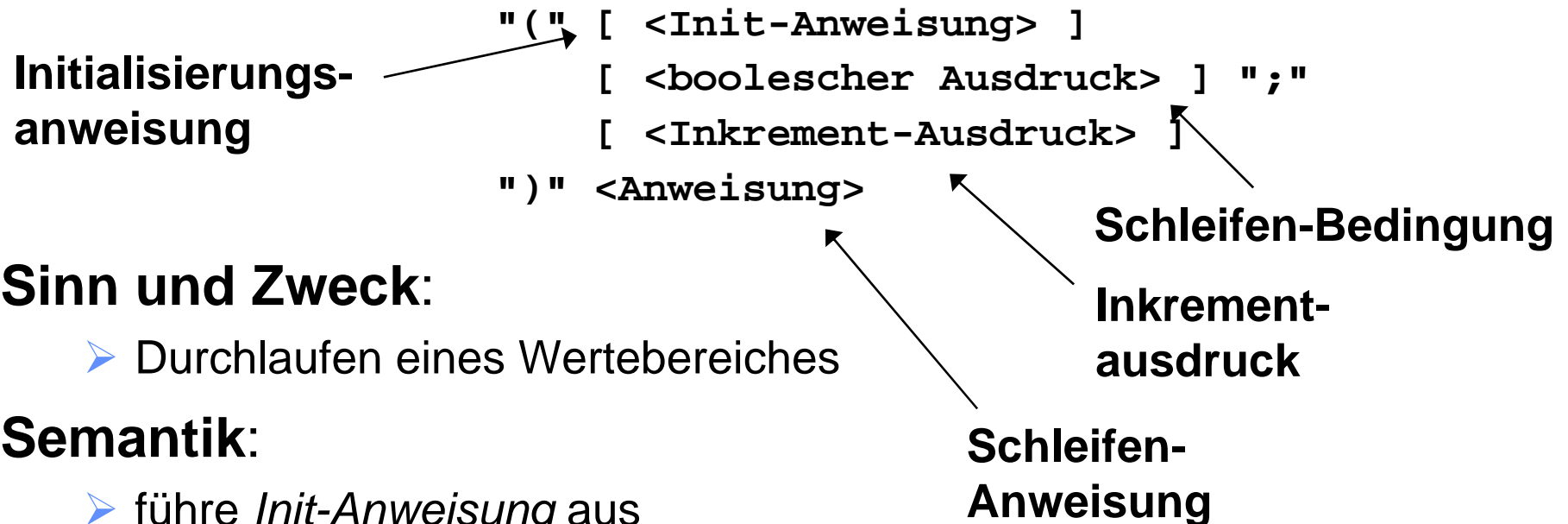
```
int anzahl = IO.readInt();
do {
    IO.println(anzahl);
    anzahl++;
} while (anzahl <= 3);
```

`<=>`

```
int anzahl = IO.readInt();
{
    IO.println(anzahl);
    anzahl++;
}
while (anzahl <= 3) {
    IO.println(anzahl);
    anzahl++;
}
```

for-Anweisung (1)

<for-Anweisung> ::= "for"



Sinn und Zweck:

- Durchlaufen eines Wertebereiches

Semantik:

- führe *Init-Anweisung* aus
- berechne den booleschen Ausdruck
- falls Ausdruck `true` liefert:
 - führe *Anweisung* aus;
 - berechne den *Inkrement-Ausdruck*;
 - berechne erneut den booleschen Ausdruck; ...
- falls Ausdruck `false` liefert, beende die for-Anweisung

Beispiele:

```
for (int i=0; i < 10; i++)  
    IO.println(i);
```

```
int summe = 0;  
int bis = IO.readInt();  
for (int i=1; i <= bis; i++) {  
    summe += i;  
}  
IO.println("sum(" + bis + ")=" + summe);
```

Semantische Äquivalenz:

```
for (<Init-Anweis> <bool-Ausdr>; <Inkr-Ausdr>)  
    <Anweisung1>
```

\Leftrightarrow (i.a.)

```
{ // wegen Gültigkeitsbereich der Init-Anweisung  
  <Init-Anweis>  
  while (<bool-Ausdr>) {  
    <Anweisung1>  
    <Inkr-Ausdr>;  
  }  
}
```

Äquivalenz gilt u.U. nicht bei Verwendung der continue-Anweisung

Beispiel:

```
for (int schritte = 1; schritte < 5; schritte++)  
    IO.println(schritte);
```

<=>

```
{  
    int schritte = 1;  
    while (schritte < 5) {  
        IO.println(schritte);  
        schritte++;  
    }  
}
```

Ergänzung:

- wird eine Variable in der Initialisierungsanweisung definiert, so erstreckt sich ihr Gültigkeitsbereich über die gesamte for-Anweisung (aber nicht weiter!)

Beispiel:

```
for (int zahl=0; zahl<3; zahl++) {  
    IO.println(zahl);  
}  
IO.println(zahl); // Fehler: zahl ungültig
```

```
<Label-Anweisung> ::= <Label> ":" <Anweisung>  
<Label> ::= <Bezeichner>
```

Semantik:

- keine Auswirkungen auf den Programmablauf

Beispiele:

```
deklariere: int anzahl = 3;
```

```
berechneWiederholt:  
  while (anzahl <= 3) {  
    berechne: anzahl += 2;  
  }
```

```
gibAus:  
  IO.println("hello world! ");
```

break-Anweisung (1)

`<break-Anweisung> ::= "break" [<Label>] ";"`

Sinn und Zweck:

- vorzeitiges Verlassen eines Blockes
- kein **goto**!

Semantik:

- fehlt das Label, so wird das innerste `do`, `while`, `for` oder `switch` verlassen
- existiert ein umgebendes `do`, `while`, `for` oder `switch` mit einem angegebenen Label, so wird dieses verlassen

Beispiel:

```
int erg = 0;
for (int zahl=0; zahl<5; zahl++) {
    ...
    if (erg > 10) break;
    ...
}
```

break-Anweisung (2)

Beispiel (ohne break):

```
class ZiffernOhneBreak {
    public static void main(String[] args) {
        int zahl1 = IO.readInt("erste Zahl (>0): ");
        int zahl2 = IO.readInt("zweite Zahl (>0): ");
        boolean gefunden = false;
        while (zahl1 > 0 && !gefunden) {
            int hilfsZahl = zahl2;
            while (hilfsZahl > 0 && !gefunden) {
                if (zahl1 % 10 == hilfsZahl % 10) {
                    IO.println("Enthalten gleiche Ziffer");
                    gefunden = true;
                } else {
                    hilfsZahl = hilfsZahl / 10;
                }
            }
            if (!gefunden) {
                zahl1 = zahl1 / 10;
            }
        }
    }
}
```

break-Anweisung (3)

Beispiel (mit break):

```
class ZiffernMitBreak {
    public static void main(String[] args) {
        int zahl1 = IO.readInt("erste Zahl (>0): ");
        int zahl2 = IO.readInt("zweite Zahl (>0): ");

        hauptprogramm:
        while (zahl1 > 0) {
            int hilfsZahl = zahl2;
            while (hilfsZahl > 0) {
                if (zahl1 % 10 == hilfsZahl % 10) {
                    IO.println("Enthalten gleiche Ziffer");
                    break hauptprogramm;
                } else {
                    hilfsZahl = hilfsZahl / 10;
                }
            }
            zahl1 = zahl1 / 10;
        }
    }
}
```

switch-Anweisung (1)

```
<switch-Anweisung> ::= "switch" "(" <Ausdruck> ")"  
                        "{" { <Fallunterscheid> } "  
<Fallunterscheid>   ::= <Anweisung>  
                        | "case" <const-Ausdruck> ":"  
                        | "default" ":"
```

Bedingungen:

- *const-Ausdrücke* vom Typ char, byte, short oder int (Literal)
- alle *const-Ausdrücke* vom selben Typ
- keine doppelte *const-Ausdrücke*
- Typ von *Ausdruck* konform zum Typ der *const-Ausdrücke*
- höchstens ein default

Sinn und Zweck:

- größere Fallunterscheidung
- effizienter als geschachtelte if-Anweisung

switch-Anweisung (2)

Semantik:

- werte *Ausdruck* aus
- falls ein *const-Ausdruck* mit dem berechneten Wert existiert:
 - springe an die entsprechende Stelle und fahre dort mit der Programmausführung fort
- falls kein entsprechender *const-Ausdruck* existiert
 - falls `default` existiert: fahre beim `default` mit der Ausführung fort
 - ansonsten: beende switch-Anweisung

Beispiel:

```
int i = IO.readInt();
switch (i) {
    case 1:  IO.println("i == 1");
             // FAHRE FORT
    case 2:  IO.println("i == (1 oder 2) ");
             break;
    case 3:  IO.println("i == 3");
             break;
    default: IO.println("i != (1/2/3) ");
}
```

switch-Anweisung (3)

Beispiel:

```
char ch = IO.readChar("char:");
int hexWert = -1;
switch (ch) {
    case '0': case '1': case '2': case '3':
    case '4': case '5': case '6': case '7':
    case '8': case '9':
        hexWert = ch - '0'; break;
    case 'a': case 'b': case 'c': case 'd':
    case 'e': case 'f':
        hexWert = (ch - 'a') + 10; break;
    case 'A': case 'B': case 'C': case 'D':
    case 'E': case 'F':
        hexWert = (ch - 'A') + 10; break;
}
if (hexWert != -1)
    IO.println(hexWert);
else
    IO.println("unguelteiges Zeichen");
```

`<continue-Anweisung> ::= "continue" [<Label>] ";"`

Sinn und Zweck:

- ans Ende eines Schleifenrumpfes springen

Semantik:

- fehlt das Label, so wird an das Ende der Schleifenanweisung des innersten `do`, `while` oder `for` gesprungen
- existiert eine umgebende Schleife mit einem angegebenen Label, so wird an das Ende der Schleifenanweisung dieser Schleife gesprungen

Beispiel:

```
int x = 0;
while (x < 10) {
    x = x + 1;
    if (x < 10) continue;
    IO.println("x == 10");
}
```

Beispiele:

```
for (int schritte=1; schritte<5; schritte++) {  
    if (schritte < 3) continue;  
    IO.println(schritte);  
}
```

!<=>

```
{  
    int schritte = 1;  
    while (schritte < 5) {  
        {  
            if (schritte < 3) continue; // Endlosschleife  
            IO.println(schritte);  
        }  
        schritte++;  
    }  
}
```

Grund für Nicht-Äquivalenz:

Bei der for-Schleife wird noch der Inkr-Ausdruck ausgewertet!

<return-Anweisung> ::= "return" [<Ausdruck>] ";"

Sinn und Zweck:

- Verlassen einer Prozedur/Funktion/Methode
- Lieferung eines Wertes

Semantik:

- die Prozedur/Funktion/Methode wird unmittelbar verlassen
- falls ein Ausdruck existiert, wird sein Wert berechnet und dieser als Funktionswert zurückgeliefert

Beispiel:

```
public static int abs(int wert) {  
    if (wert >= 0)  
        return wert;  
    else  
        return -wert;  
}
```

Beispiel 1

Schreiben Sie ein Programm zur Zinseszins-Berechnung

```
class Zinseszins {
    public static void main(String[] args) {

        System.out.println("Berechnung des Zinseszins");
        double kapital = IO.readDouble("Kapital: ");
        double zinssatz = IO.readDouble("Zinssatz: ");
        int jahre = IO.readInt("Anzahl Jahre: ");

        for (int j=0; j<jahre; j++) {
            kapital = kapital * (1.0 + zinssatz / 100.0);
        }

        System.out.println(
            "Kapital in " + jahre + " Jahren = " + kapital);
    }
}
```

Schreiben Sie ein Programm "*Rechner*", das zunächst zwei Zahlen und dann einen Buchstaben (Operatorzeichen) einliest, das Ergebnis berechnet und dieses ausgibt. Dieses soll sich wiederholen, bis der Nutzer als Operator ein 'q' (für "quit") eingibt.

Beispiel:

```
$ java Rechner
Zahl 1 eingeben:
-23<CR>
Zahl 2 eingeben:
2<CR>
Operator eingeben:
*<CR>
-46
Zahl 1 eingeben:
0<CR>
Zahl 2 eingeben:
0<CR>
Operator eingeben:
q<CR>
$
```

```
class Rechner {
    public static void main(String[] a) {
        hauptprogramm:
        while (true) {
            int zahl1 = IO.readInt("Zahl 1 eingeben:");
            int zahl2 = IO.readInt("Zahl 2 eingeben:");
            char operator = IO.readChar("Operator eingeben:");

            switch (operator) {
                case '+': IO.println(zahl1+zahl2); break;
                case '-': IO.println(zahl1-zahl2); break;
                case '*': IO.println(zahl1*zahl2); break;
                case '/': IO.println(zahl1/zahl2); break;
                case '%': IO.println(zahl1%zahl2); break;
                case 'q': break hauptprogramm;
                default: IO.println("ungueltiger Op"); break;
            }
        }
    }
}
```

Beispiel 3 / Aufgabe

Schreiben Sie ein Programm "*Rechteck*", das zunächst eine Zahl *hoehe* einliest, die größer als 1 ist, dann eine Zahl *breite* einliest, die größer als 1 und kleiner als 10 ist, und anschließend ein "Rechteck" mit der Höhe *hoehe* und der Breite *breite* in folgender Gestalt auf den Bildschirm ausgibt:

Beispiel:

```
$ java Rechteck
Hoehe eingeben (>1):
4<CR>
Breite eingeben (1 < b < 10):
5<CR>
+----+
|    |
|    |
|    |
+----+
$
```

Beispiel 3 / Lösung (1)

```
class Rechteck {
    public static void main(String[] a) {

        // richtige Hoehe einlesen
        int hoehe = IO.readInt("Hoehe eingeben (>1):");
        while (hoehe < 2) {
            IO.println("Eingabefehler: Hoehe <= 1");
            hoehe = IO.readInt("Nochmal Hoehe eingeben (>1):");
        }

        // richtige Breite einlesen
        int breite = IO.readInt("Breite eingeben (1 < b < 10):");
        while (breite < 2 || breite > 9) {
            IO.println("Eingabefehler: Breite nicht zw. 2 und 9");
            breite = IO.readInt("Breite eingeben (1 < b < 10):");
        }
    }
}
```

Beispiel 3 / Lösung (2)

```
// Rechteck ausgeben
// erste Zeile ausgeben
IO.print('+');
for (int i=2; i<breite; i++)
    IO.print('-');
IO.println('+');

// mittlere Zeilen ausgeben
for (int j=2; j<hoehe; j++) {
    IO.print('|');
    for (int i=2; i<breite; i++)
        IO.print(' ');
    IO.println('|');
}

// letzte Zeile ausgeben
IO.print('+');
for (int i=2; i<breite; i++)
    IO.print('-');
IO.println('+');
}
}
```

Aufgabe 1

Schreiben Sie ein Java-Programm `Sterne`, das zunächst die Eingabe eines positiven `int`-Wertes `anzahl` erwartet. Anschließend soll das Programm `anzahl`-mal ein Sternchen auf den Bildschirm ausgeben. Realisieren Sie das Programm viermal, und zwar einmal mit einer while-Anweisung, einmal mit einer do-Anweisung, einmal mit einer for-Anweisung und einmal mit dem Konstrukt while (true) und einer break-Anweisung.

Beispiel:

```
$ java Sterne  
Eingabe: 5  
Ausgabe: *****
```

```
$ java Sterne  
Eingabe: 7  
Ausgabe: *****
```

Aufgabe 2

Schreiben Sie ein Java-Programm `Raute`, das zunächst die Eingabe eines `int`-Wertes `durchmesser` erwartet. Überprüfen Sie die Eingabe. Es soll ein ungerader Wert zwischen 5 und 50 sein. Anschließend soll das Programm eine Raute mit dem entsprechenden Durchmesser auf den Bildschirm ausgeben.

Beispiel:

```
$ java Raute  
Eingabe: 7
```

