Programmierkurs Java

UE 5 - Anweisungen

Dr.-Ing. Dietrich Boles

Gliederung

- Grundlagen
- Deklarationsanweisung
- Zuweisung
- Leeranweisung
- Blockanweisung
- if-Anweisung
- while-Anweisung
- do-Anweisung
- for-Anweisung
- Label-Anweisung
- break-Anweisung
- switch-Anweisung
- continue-Anweisung
- return-Anweisung
- Beispiele
- Zusammenfassung

Definition:

Anweisung = Vorschrift zur Verarbeitung von Daten

Klassifikation:

- elementare Anweisungen
- zusammengesetzte Anweisungen
- Anweisungen zur Ablaufsteuerung

Imperative Programmierung

=

(sequentielles) Abarbeiten von Anweisungen

Deklarationsanweisung

```
<Variablendefinition> ::= <Datentyp>
                          <Bezeichner> "=" <Ausdruck>
                          {"," <Bezeichner> "=" <Ausdruck>}
                          ";"
```

Sinn und Zweck:

- Einführung einer oder mehrerer Variablen

Semantik:

- Auswertung der Initialisierungsausdrucks
- Reservierung von Speicherplatz (typ-spezifisch)
- Zuweisung des Initialisierungswertes zum Speicherbereich

Beispiele:

```
int elf = 2 + 9;
char zifferNeun = '9';
boolean bedingung1 = true, bedingung2 = false;
```

Zuweisung(sanweisung)

```
<Zuweisung> ::= <Variable> "=" <Ausdruck> ";"
```

Sinn und Zweck:

- Änderung des Wertes einer Variablen

Semantik:

- Auswertung des Ausdrucks
- Zuweisung des berechneten Wertes an die Variable

Beispiele:

```
int summe = 0, wert = 47;     // Deklaration
summe = summe + wert * 3;     // Zuweisung
wert = 2 * wert * wert;       // Zuweisung
```

Blockanweisung

```
<Block> ::= "{" {<Anweisung>} "}"
```

Sinn und Zweck:

- Zusammenfassung von Anweisungen zu einer (zusammengesetzten) Anweisung

Beispiel:

```
int summe = 0, zahl = 100;
while (zahl > 0) {
    summe = summe + zahl;
    zahl = zahl - 1;
}
```

Blockanweisung / Gültigkeitsbereich

- Der Gültigkeitsbereich eines Variablennamens ist der Block, in dem die Variable definiert wird, sowie aller inneren Blöcke, und zwar **nach** der Stelle seiner Definition.
- Im Gültigkeitsbereich einer Variablen dürfen keine weiteren gleichnamigen Variablen definiert werden.

```
{
    int wert = zahl; // Fehler
    int zahl = 5;
    zahl = zahl * 7;
    {
        zahl = zahl / 5;
        double zahl = 9.3; // Fehler
        char zeichen = 'z';
    }
    System.out.println(zahl);
    System.out.println(zeichen); // Fehler
}
```

Blockanweisung / Lebensdauer

- Die Lebensdauer einer Variablen beginnt bei der Ausführung der Variablendefinition (→ Reservierung von Speicherplatz)
- Die Lebensdauer einer Variablen endet nach dem Verlassen des Blockes, in dem sie erzeugt wurde (→ Freigabe von Speicherplatz)

```
{
    int zahl = 5;
    zahl = zahl * 2;
    {
        char zeichen = 'a';
        System.out.println(zeichen);
    }
    System.out.println(zahl);
}
```

```
;
```

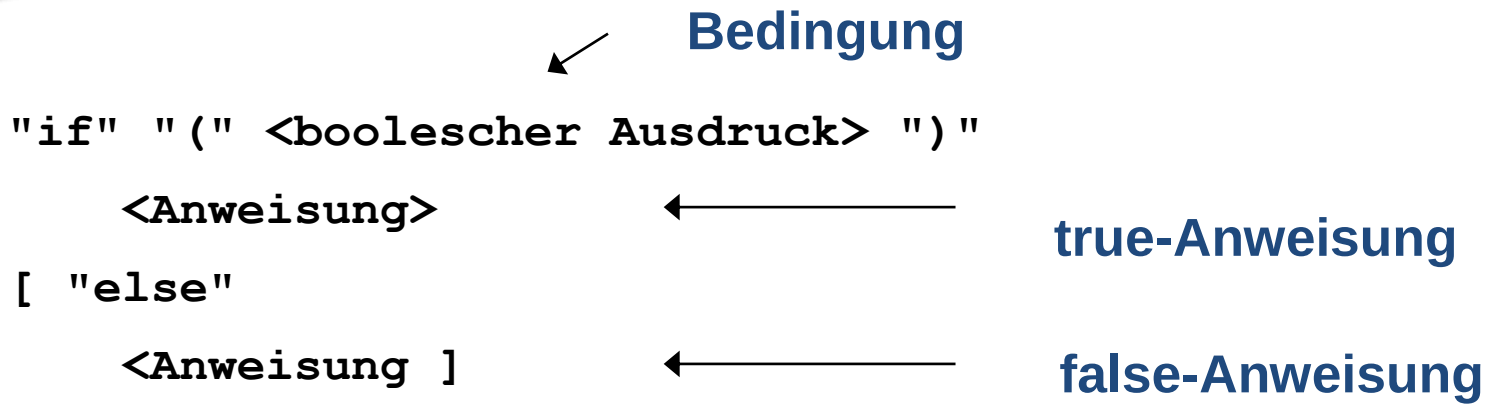
Sinn und Zweck:

- „Platzhalter“

Beispiele:

```
double zufall = Math.random();
if (zufall < 0.5)
    ;
else
    zufall = zufall - 0.5;
```

if-Anweisung (1)



Sinn und Zweck:

- Ausführung von Anweisungen unter bestimmten Bedingungen

Semantik:

- Werte Bedingung aus
- Wenn Bedingung wahr ist, führe true-Anweisung aus
- Wenn Bedingung falsch ist, führe (falls vorhanden) false-Anweisung aus

if-Anweisung (2)

```
char zeichen = IO.readChar("Zeichen: ");
if ('a' <= zeichen && zeichen <= 'z') {
    System.out.println("Kleinbuchstabe");
} else if ('A' <= zeichen && zeichen <= 'Z') {
    System.out.println("Großbuchstabe");
} else if ('0' <= zeichen && zeichen <= '9') {
    System.out.println("Ziffer");
} else {
    System.out.println("Sonderzeichen");
}
```

while-Anweisung (1)

```
"while" "(" <boolescher Ausdruck> ")" <Anweisung>
```

Schleifenbedingung **Schleifenanweisung**

Sinn und Zweck:

- Wiederholte Ausführung von Anweisungen unter bestimmten Bedingungen

Semantik:

(1) Werte Bedingung aus

(2) Falls die Bedingung falsch ist, beende die while-Anweisung.

(3) Falls die Bedingung wahr ist, führe Schleifenanweisung aus und fahre bei (1) fort.

while-Anweisung (2)

```
int zahl = IO.readInt("Zahl (>=0): ");
int spiegelzahl = 0;
while (zahl > 0) {
    spiegelzahl = spiegelzahl * 10 + zahl % 10;
    zahl = zahl / 10;
}
System.out.println(spiegelzahl);
```

do-Anweisung (1)

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

Schleifenanweisung

Schleifenbedingung

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

do-Anweisung (2)

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"
                    "(" [ <Init-Anweisung> ]
                        [ <boolescher Ausdruck> ] ";"
                        [ <Inkrement-Ausdruck> ]
                    ")" <Anweisung>
```

Initialisierungs-anweisung

Schleifenbedingung

Inkrement-ausdruck

Schleifenanweisung

Sinn und Zweck:

- Durchlaufen eines Wertebereiches

Semantik:

- führe *Initialisierungsanweisung* aus
- berechne *Schleifenbedingung*
- falls *Schleifenbedingung* `true` liefert:
  - führe *Schleifenanweisung* aus;
  - berechne den *Inkrementausdruck*;
  - berechne erneut die *Schleifenbedingung*; ...
- falls *Schleifenbedingung* `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);
```

for-Anweisung (3)

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
```

Semantische Äquivalenz:

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

          <=> (i.a.)

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

Label-Anweisung

```
<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;
while (true) {
    erg++;
    if (erg > 5) break;
    System.out.print(erg);      // 12345
}
System.out.println(erg * erg); // 36
```

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;
} } } }
```

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); seit Java 5 auch `enum`; seit Java 7 auch `String`
- 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

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) ");
}
System.out.println("Fertig");
```

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("ungueltiges Zeichen");
```

continue-Anweisung (1)

```
<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 void main(String args[]) {
    ...
    if (wert >= 0)
        return;  // Programmende
    ...
}
```

Zusammenfassung

- Imperative Programmierung: (sequentielle) Abarbeitung von Anweisungen
- Anweisung: Vorschrift zur Verarbeitung von Daten/Werten

- Anweisungstypen:
  - Deklarationen
  - Zuweisungen
  - Anweisungssequenz
  - Kontrollstrukturen
    - bedingte Anweisungen (if, switch)
    - Wiederholungsanweisungen, Schleifen (while, do, for)
  - …