Einstieg in die Informatik mit Java

Anweisungen

Gerd Bohlender

Institut für Angewandte und Numerische Mathematik

Gliederung

Anweisungen

Anweisungen sind die eigentlichen „Befehle“, die vom Computer ausgeführt werden sollen.

Gliederung

Ausdrucksanweisung

Syntax

Ausdruck;

Beispiel

```
a = 1;
a++;
y = Math.sin(x);
```

Achtung

Die folgenden Beispiele sind zwar mit syntaktisch korrekten Ausdrücken gebildet, sie sind aber sinnlos und deshalb verboten:

```
1;
1 + 2 + 3;
x + y;
```

Ausdrucksanweisung, erlaubte Konstrukte

Andere (auf den ersten Blick ähnliche) Konstrukte sind jedoch erlaubt, da sie über Nebeneffekte eine Wirkung haben könnten:

```
funktion();
```

(ein Funktionsaufruf, dessen Ergebnis nicht abgespeichert wird)

Gliederung

Einfache Ausgabeanweisung

Syntax

```
System.out.println( Ausdruck );
```

Der Ausdruck muss nicht unbedingt vom Typ `String` sein, sondern kann zunächst auch von einem anderen Typ sein, der dann in eine Zeichenkette konvertiert wird.

Einfache Ausgabeanweisung, Beispiele

Achtung

Zeichenketten können mit Hilfe der *Stringkonkatenation* + verknüpft werden. Allerdings besteht dabei die Gefahr der Verwechslung mit der Addition!

Beispiele

```
                                                    // Ausgabe:
System.out.println ("Ergebnis:"+1+2);               // Ergebnis:12
System.out.println ("Ergebnis:"+(1+2));             // Ergebnis:3
System.out.println (1+':'+2);                       // 61
System.out.println (""+1+':'+2);                    // 1:2
System.out.println (1+":"+2);                       // 1:2
```

Einfache Ausgabeanweisung, Zeilenende

- `System.out.println();` beendet nach Ausgabe die Zeile, die nächste Ausgabe erfolgt in einer neuen Zeile (vgl. Pascal: `writeln`).
- `System.out.print();` beendet die Zeile nicht, die nächste Ausgabe erfolgt in der gleichen Zeile (vgl. Pascal: `write`).

Gliederung

Einfache Eingabeanweisung ab Java 5

Zu Beginn eines jeden Quelltextes mit Eingabe muss mit Hilfe der `import`–Anweisung die Klasse `java.util.Scanner;` bekannt gemacht werden.
Dies geschieht mit der Zeile

```
import java.util.Scanner;
```

bzw.

```
import java.util.*;
```

am Programmanfang.
Zusätzlich muss ein Scanner erzeugt werden und die Eingabe sollte auf den US-Stil mit Dezimalpunkt umgestellt werden.
Dies geschieht mit folgenden Zeilen vor der ersten Eingabe:

```
Locale.setDefault(Locale.US);
Scanner sc = new Scanner(System.in);
```

Einfache Eingabeanweisung, Lesen von Werten

Beispiel

Einlesen einer ganzen und einer Gleitkommazahl.

```
int a    = sc.nextInt();
double b = sc.nextDouble();
```

Vor der Eingabe ist es oft sinnvoll, einen beschreibenden Text auszugeben:

Beispiel

```
double s;
System.out.print("Bitte Startwert s eingeben: ");
s = sc.nextDouble();
```

Einfache Eingabeanweisung, Beispiel

Beispiel

Einlesen einer ganzen Zahl und Ausgabe des doppelten Werts.

```
import java.util.*;
public class Demo {
  public static void main(String [] args) {
    Locale.setDefault(Locale.US);
    Scanner sc = new Scanner(System.in);
    System.out.print("Bitte i eingeben: ");
    int i = sc.nextInt();
    System.out.println("i*2=" + i*2);
  }
}
```

Gliederung

Verbundanweisung

Syntax

{ *Anweisung 1* . . . *Anweisung n* }

Mittels des geschweiften Klammerpaars { } können mehrere Anweisungen zu einer *Verbundanweisung* zusammengefaßt werden.
Die Verbundanweisung darf überall stehen, wo sonst aus syntaktischen Gründen nur eine Anweisung erlaubt ist.
Die Anweisungen werden dabei in der angegebenen Reihenfolge ausgeführt.

Beispiel

```
{ x = 1; y = 2; z = 3; }
```

Achtung

Zu einer Ausdrucksanweisung gehört immer ein Semikolon!

Gliederung

Bedingte Anweisung

Syntax - Einseitige bedingte Anweisung

`if (` *Bedingung* `)` *Anweisung1*

Syntax - Zweiseitige (doppelseitige) bedingte Anweisung

`if (` *Bedingung* `)` *Anweisung1* `else` *Anweisung2*

Bei der Bedingung in den runden Klammern handelt es sich um einen boolschen Ausdruck. Ergibt der Wert des Ausdrucks `true`, so wird die erste Anweisung (*Anweisung1*) ausgeführt. Ergibt der Wert des Ausdrucks `false`, so wird bei der einseitigen bedingten Anweisung nichts, bei der zweiseitigen bedingten Anweisung die zweite Anweisung (*Anweisung2*) ausgeführt.

Bedingte Anweisung, Beispiel

Beispiel

Bestimmung des Maximums zweier ganzen Zahlen `x` und `y`.

```
int x = 21, y = 12, max;
if (x>y)
  max = x;
else
  max = y;
// ergibt: max = 21
```

Bedingte Anweisung, Achtung

Achtung

Möchte man anstelle einer einzigen Anweisung mehrere Anweisungen in den `if`– bzw. `else`–Zweigen ausführen, so verwendet man dafür eine Verbundanweisung!

Existieren keine Klammern, so wird jedes `else` immer dem nächsten davorstehenden `if` zugeordnet. Diese Zuordnung kann durch eine geeignete Klammerung geändert werden.

Bedingte Anweisung, Beispiel

Beispiel

```
int x = 5, y = 6; z = 4;
if (x>y)
    if (x>z) max = x;
    else max = z;
    // Zuordnung zum zweiten if

if (x>y)
    {if (x>z) max = x;}
    else max = z;
    // Zuordnung zum ersten if
```

Gliederung

Auswahlanweisung

Syntax

```
switch ( Ausdruck ) {
  case konstanter Ausdruck1: Anweisungsfolge1
  case konstanter Ausdruck2: Anweisungsfolge2
  ...
  default: Anweisungsfolge
}
```

Der Ausdruck darf vom Typ `byte`, `short`, `int` oder `char` sein, bei den konstanten Ausdrücken muss es sich um voneinander verschiedene zuweisungskompatible Konstanten handeln.
Stimmt der Ausdruck mit einem konstanten Ausdruck überein, so wird bei der zugehörigen Anweisungsfolge fortgefahren und die folgenden Anweisungsfolgen werden sequentiell abgearbeitet.
Der `default`–Zweig ist optional. Dieser wird ausgeführt, falls kein konstanter Ausdruck passt.

Auswahlanweisung, break

Achtung

Die einzelnen Fälle müssen sich nicht ausschließen! Um die sequentielle Abarbeitung zu verhindern, muss den Anweisungsfolgen i. allg. eine `break`–Anweisung (siehe auch Abschnitt 38) folgen!

Auswahlanweisung, Beispiel

Beispiel

```
switch (x) {
  case 1:
  case 2: x *= 4;
  case 3: x *= 8;
  case 4: x *= 16; break;
  default: x *= 2;
}
```

Für `x=1` ergibt sich der Wert `512`, für `x=2` der Wert `1024`, für `x=3` der Wert `384`, für `x=4` der Wert `64` und für alle anderen Werte von `x` jeweils der Wert `2*x`.

Gliederung

for-Schleife

Syntax

`for` (*Initialisierung; Ausdruck; Update*) *Anweisung*

- Der Initialisierungsteil wird einmal ausgeführt und dient zur Deklaration von lokalen Variablen. Er kann eine Liste von Ausdrücken zur Initialisierung von Zählern enthalten (alle mit Komma getrennt).
- Der Ausdrucksteil ist vom Typ `boolean`. Solange sich der Wert `true` ergibt, wird die Anweisung und anschließend der Updateteil wiederholt. Die Schleife wird beendet, wenn der Ausdruck `false` ist.
- Der Updateteil enthält eine Liste von Ausdrücken (alle mit Komma getrennt), welche der Reihe nach ausgeführt werden.

for-Schleife, optionale Teile

Alle drei Teile (Initialisierung, Ausdruck, Update) sind optional. Allerdings müssen in den runden Klammern insgesamt immer zwei Semikolons vorhanden sein.

- Fehlt der Initialisierer, so muss i.a. vor der Schleife die Zählvariable initialisiert werden.
- Fehlt der Ausdrucksteil, so gilt die Bedingung immer als erfüllt. Es ergibt sich eine Endlosschleife, die mit einer `break`–Anweisung verlassen werden muss!
- Fehlt das Update, so muss i.a. die Zählvariable innerhalb der Anweisung verändert werden.

for-Schleife, Beispiele

```
int i;
for (i=0; i<10; ++i) System.out.println (i);
// 0 1 ... 9, Var. i gilt ueber Schleifen-Ende hinaus.
for (int j=0; j<10; ++j) System.out.println (j);
// 0 1 ... 9, Variable j gilt nur bis Schleifen-Ende.
for (int j=0; j<100; j+=2) System.out.println (j);
// 0 2 4 ... 98
for (int j=9; j>=0; --j) System.out.println (j);
// 9 8 ... 0
for (int j=2, k=3; j+k<27; j*=2, k+=3)
    System.out.println (j);
// verwirrend, aber korrekt: 2 4 8
```

for-Schleife, Beispiele mit ausgelassenen Teilen

```
int i=5;
for (; i<10; ++i) System.out.println (i);
```

// 5 6 ... 9, Var. i gilt ueber Schleifen-Ende hinaus.

```
for (int j=0;; ++j)
  if (j<10) System.out.println (j);
  else break;
```

// 0 1 ... 9, Variable j gilt nur bis Schleifen-Ende.

```
for (int j=0; j<10; ) System.out.println (++j);
```

// 1 2 3 ... 10

```
int k=5;
for (;;)
  if (k<10) System.out.println (k++);
  else break;
```

// 5 6 7 8 9

for-Schleife, Beispiel mit double

Achtung

Das nachstehende Beispiel ist nur mit Vorsicht zu geniessen!

```
for (double x=0; x<=1; x+=0.1)
System.out.println (x);
```

Durch die inkrementelle Erhöhung in Schritten von $0,1$ kann es zu Rundungsfehlern kommen. Es ist hier deshalb nicht klar, ob die Zahl Eins erreicht wird oder nicht!

Gliederung

while-Schleife

Syntax

`while (` *Bedingung* `)` *Anweisung*

Äquivalent

`for ( ;` *Bedingung* `; )` *Anweisung*

Ist die Bedingung in den runden Klammern wahr, so wird die Anweisung solange wiederholt, bis die Bedingung falsch ist.

Achtung

Bei der Anweisung kann es sich natürlich wieder um eine Verbundanweisung handeln!

while-Schleife, Beispiel Quersumme einer natürlichen Zahl

```
import java.util.*;

public class Quersumme {
    public static void main (String [] args) {
        Scanner sc = new Scanner (System.in);
        int n, Summe = 0;
        System.out.print ("n = ");
        n = sc.nextInt();
        while (n>0) {
            Summe += n % 10;
            n     /= 10;
        }
        System.out.println("Quersumme = "+Summe);
    }
}
```

Gliederung

do-Schleife

Syntax

`do` *Anweisung* `while ( ` *Bedingung* `);`

Im Prinzip wie die `while`–Schleife, allerdings wird hier zuerst die Anweisung ausgeführt und dann die Bedingung geprüft. Das bedeutet, daß bei einer `do`–Schleife im Gegensatz zu einer `while`–Schleife die Anweisung immer mindestens einmal ausgeführt wird.

Achtung

Die Bedingung hat genau die gegenteilige Wirkung wie in Pascal bei `repeat ... until`.

Vergleich do- und while-Schleife

Beispiel

Vergleich von `do`– und `while`–Schleife.

```
int n = 5;
int m = 5;

do n /= 2;
while (n>=10);              // ergibt n=2

while(m>=10) m /= 2;        // ergibt m=5
```

Beispiel

Einlesen einer positiven Zahl.

```
do p = sc.nextInt ();
while (p<=0);
```

Gliederung

break-Anweisung

Syntax

```
break;
```

Die `break`–Anweisung steht innerhalb der Auswahl–Anweisung oder innerhalb von `for`–, `while`– und `do`–Schleifen. Diese werden bei Erreichen der `break`–Anweisung sofort verlassen. Es existiert auch eine Variante mit zugehöriger Marke, d. h. beim Erreichen der `break`–Anweisung wird die markierte Anweisung verlassen. Diese Variante kann auch bei anderen Anweisungen (z. Bsp. Verbundanweisungen) eingesetzt werden.

break-Anweisung, Beispiel

Beispiel

```
draussen:                              // gesetzte Marke
for (int i=0; i<10; ++i)               // aeussere Schleife
    for (int j=0; j<100; ++j) {        // innere Schleife
        if (...) break;                // verlaesst innere Schleife
        if (...) break draussen;       // verlaesst die mit "draussen"
                                       // markierte aeussere Schleife
    }
```

Achtung

Dieser Programmierstil ist verwirrend und macht den Quelltext schnell unübersichtlich und fehleranfällig!

Gliederung

continue-Anweisung

Syntax

```
continue;
```

Die `continue`–Anweisung steht nur innerhalb von `for`–, `while`– und `do`–Schleifen. Dabei wird bei Erreichen der `continue`–Anweisung der aktuelle Schleifendurchlauf sofort beendet und der nächste begonnen.
Analog zur `break`–Anweisung existiert auch hier eine Variante mit zugehöriger Marke, an die das Programm bei Erreichen der `continue`–Anweisung „springt“, d. h. der aktuelle Schleifendurchlauf der markierten Schleife wird abgebrochen und der nächste begonnen.

continue-Anweisung, Beispiel

Beispiel

```
for (int i=0; i<10; ++i) {
    System.out.print (" a = ");
    int a = sc.nextInt ();
    System.out.print (" b = ");
    int b = sc.nextInt ();
    if (b==0) continue;
    System.out.println (a/b);
}
```

continue-Anweisung, Beispiel continue mit Marke

Beispiel

```
aussen:
for (int i=1; i<=4; ++i) {
    for (int j=1; j<=4; ++j) {
        if (j==2) continue;
        if (i==2) continue aussen;
        System.out.println (i*j);
        // Ausgabe: 1 3 4 3 9 12 4 12 16
    }
}
```

Achtung

Wie die Verwendung der `break`–Anweisung ist auch die Verwendung der `continue`–Anweisung sehr verwirrend! Mit beiden Anweisungen sollte, wenn überhaupt, nur sehr sparsam umgegangen werden!

Gliederung

Leere Anweisung

Syntax

```
; // Ausdrucksanweisung ohne Ausdruck
{} // Verbundanweisung ohne Anweisungen
```

Leere Anweisungen sind wirkungslos. Sie sind manchmal aus syntaktischen Gründen sinnvoll.

Leere Anweisung, Vorsicht!

Achtung

manchmal führen leere Anweisungen zu versteckten Fehlern, wie in den folgenden Beispielen:

```
if (2 > 3);
   System.out.print ("nanu");
```

Ausgabe: nanu

```
if (2 < 3)
   System.out.print ("klar...");
else;
   System.out.print ("nanu");
```

Ausgabe: klar...nanu

```
int i=33;
for (i=0; i<10; i++);
   System.out.print (i);
```

Ausgabe: 10