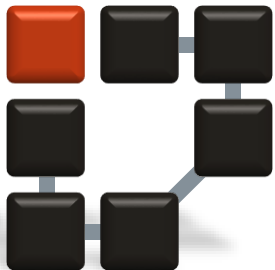


Informatik 1 für Nebenfachstudierende Grundmodul

Java – Ablaufsteuerung (Beispiele)

Kai-Steffen Hielscher
Folienversion: 16. Januar 2018



Informatik 7
Rechnernetze und
Kommunikationssysteme



FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Blöcke

- Anweisungen durch geschweifte Klammern `{ }` zu Blöcken zusammenfassbar
- Block stellt eine zusammengefasste Anweisung dar
- verwendbar, wo einzelne Anweisung erlaubt ist
- Blöcke können geschachtelt werden
- Blöcke durch Einrückungen deutlich machen
- Variablen nur bis Ende des Blocks gültig, in dem sie deklariert wurden
 - Gültigkeitsbereich

Blöcke

■ Beispiel:

```
{  
    // äußerer Block  
    double x = 2.0;  
    x = x + 1.0;  
    {  
        // erster innerer Block  
        double y;  
        y = x * 3.141592654;  
        System.out.println(y);  
    }  
    System.out.println(x);  
    {  
        // zweiter innerer Block  
        double z;  
        z = x + 44.11;  
        System.out.println(z);  
    }  
}
```

Bedingte Ausführung

- Entscheidungsanweisung *if*

```
if ( Ausdruck )  
    Anweisung
```

- zunächst Auswertung von *Ausdruck*, Ergebnis muss boolescher Wert sein, z.B. Ergebnis eines Vergleichs
- *Anweisung* wird nur ausgeführt, falls *Ausdruck* zu *true* evaluiert

Bedingte Ausführung

■ Beispiel:

```
int x = 2;
```

```
if ( x > 0 )  
    System.out.println("Die Zahl ist positiv.");
```

Bedingte Ausführung

- Alternative `if ... else`

```
if ( Ausdruck )  
    Anweisung1  
else  
    Anweisung2
```

- zunächst Auswertung von *Ausdruck*, Ergebnis muss boolescher Wert sein, z.B. Ergebnis eines Vergleichs
- *Anweisung1* wird ausgeführt, falls *Ausdruck* zu `true` evaluiert
- anderenfalls wird *Anweisung2* ausgeführt

Bedingte Ausführung

■ Beispiel:

```
int x = 2;

if ( x < 0 )
    System.out.println("Die Zahl ist negativ.");
else
    System.out.println("Die Zahl ist nicht negativ.");
```

Bedingte Ausführung

- `if ... else` mit Blöcken
- mehrere Anweisungen in Abhängigkeit vom Ergebnis eines logischen Ausdrucks ausführen

```
if ( Ausdruck ) {  
    Anweisung1.1  
    ...  
    Anweisung1.m  
} else {  
    Anweisung2.1  
    ...  
    Anweisung2.n  
}
```

Bedingte Ausführung

■ Beispiel 1:

```
int zahl = -2;
int vorzeichen = +1;
int betrag = 0;

if ( zahl < 0 ) {
    System.out.println("Die Zahl ist negativ.");
    vorzeichen = -1;
} else {
    System.out.println("Die Zahl ist nicht negativ.");
    vorzeichen = +1;
}
betrag = vorzeichen * zahl;
System.out.println("Der Betrag der Zahl ist " + betrag + ".");
```

Bedingte Ausführung

■ Beispiel 2:

```
int zahl = -2;

if ( zahl < 0 ) {
    System.out.println("Die Zahl ist negativ.");
} else if ( zahl > 0 ) {
    System.out.println("Die Zahl ist positiv.");
} else {
    System.out.println("Die Zahl ist Null.");
}
```

Bedingte Ausführung

■ Beispiel 3:

```
int zahl = -2;

if ( zahl < 0 )
    System.out.println("Die Zahl ist negativ.");
else if ( zahl > 0 )
    System.out.println("Die Zahl ist positiv.");
else
    System.out.println("Die Zahl ist Null.");
```

Bedingte Ausführung

- Verzweigung in unterschiedliche Alternativen mittels `switch ... case`

```
switch ( Ausdruck ) {  
    case Konstante1:  
        Anweisung1.1  
        ...  
        Anweisung1.m  
        break;  
    case Konstante2:  
        Anweisung2.1  
        ...  
        Anweisung2.n  
        break;  
    default:  
        Anweisung3.1  
        ...  
        Anweisung3.n  
}
```

Bedingte Ausführung

- Verzweigung in unterschiedliche Alternativen mittels `switch ... case`
 - Zunächst *Ausdruck* auswerten, Ergebnis muss vom Typ `byte`, `short`, `int` oder `char` sein
 - wenn der Wert des Ergebnisses einer der definierten Konstanten entspricht, werden die nachfolgenden Anweisungen bis zum nächsten `break` ausgeführt
 - wenn der Wert keiner Konstanten entspricht und ein `default`-Abschnitt vorhanden ist, wird dieser ausgeführt

Bedingte Ausführung

- Verzweigung in unterschiedliche Alternativen mittels `switch ... case`

- Einfaches Beispiel:

```
int a = 2;

switch ( a ) {
    case 1:
        System.out.println("Die Zahl ist eins.");
        break;
    case 2:
        System.out.println("Die Zahl ist zwei.");
        break;
    default:
        System.out.println("Die Zahl ist weder eins noch zwei.");
}
```

Bedingte Ausführung

■ Alternativ mit `if`

```
int a = 2;

if ( a == 1 ) {
    System.out.println("Die Zahl ist eins.");
} else if ( a == 2 ) {
    System.out.println("Die Zahl ist zwei.");
} else {
    System.out.println("Die Zahl ist weder eins noch zwei.");
}
```

Schleifen

■ for-Schleife

```
for (Initialisierung; Ausdruck; Updateliste)  
    Anweisung
```

■ bzw.

```
for (Initialisierung; Ausdruck; Updateliste) {  
    Anweisung_1  
    ...  
    Anweisung_n  
}
```

Schleifen

■ `for`-Schleife

- zunächst werden in *Initialisierung* eine oder mehrere Variablen (vom gleichen Typ) initialisiert und ggf. deklariert
 - sogenannte **Schleifenvariablen**
 - falls Deklaration: Variablen nur innerhalb der Schleife gültig
- dann wird *Ausdruck* ausgewertet, Ergebnis muss boolescher Wert sein
- ist der Wert `true`, so wird die folgende *Anweisung* bzw. der folgende Anweisungsblock ausgeführt
- nach Ausführung werden Anweisungen in *UpdateListe* ausgeführt
 - mit Kommata `,` getrennte Liste von Anweisungen
- nun wird wieder *Ausdruck* ausgewertet
- wird wiederholt, bis *Ausdruck* zu `false` evaluiert

Schleifen

■ for-Schleife

■ Einfaches Beispiel

```
for(int i = 1; i <= 10; i++) {  
    System.out.println(i);  
}
```

Schleifen

- abweisende Schleife mit `while`

```
while (Ausdruck)  
    Anweisung
```

- bzw.

```
while (Ausdruck) {  
    Anweisung_1  
    ...  
    Anweisung_n  
}
```

Schleifen

■ abweisende Schleife mit `while`

- zunächst wird *Ausdruck* ausgewertet, Typ des Ergebnisses muss `boolean` sein
- ist der Wert des Ergebnisses `true`, wird *Anweisung* bzw. der Anweisungsblock ausgeführt
- danach wieder Auswertung von *Ausdruck*
- bei Wert `false` keine Ausführung der Anweisung(en), Abbruch der Schleife
- *Ausdruck* wird immer vor Ausführung der Anweisungen ausgewertet
 - wenn gleich beim ersten Versuch `false`, **keine Ausführung**
 - **abweisende Schleife**

Schleifen

■ anweisende Schleife

■ Einfaches Beispiel

```
int i = 1;

while (i <= 10) {
    System.out.println(i);
    i++;
}
```

Schleifen

- nichtabweisende Schleife mit **do**

do

Anweisung

while (*Ausdruck*);

- bzw.

do {

Anweisung_1

...

Anweisung_n

} **while** (*Ausdruck*);

Schleifen

- nichtabweisende Schleife mit **do**
 - zunächst wird *Anweisung* bzw. der Anweisungsblock ausgeführt
 - dann wird *Ausdruck* ausgewertet, Typ des Ergebnisses muss **boolean** sein
 - ist der Wert des Ergebnisses **true**, wird Ausführung der Anweisung(en) wiederholt, ansonsten wird die Schleife abgebrochen
 - *Ausdruck* wird immer erst nach Ausführung der Anweisungen ausgewertet
 - stets mindestens eine Ausführung der Anweisungen, selbst wenn gleich beim ersten Versuch **false**
 - **nichtabweisende Schleife**

Schleifen

■ nichtanweisende Schleife

■ Einfaches Beispiel

```
int i = 1;  
  
do {  
    System.out.println(i);  
    i++;  
} while (i <= 10);
```

Schleifen

■ Endlosschleifen

- wird Abbruchkriterium der Schleife nie erreicht, so endet Programm nie
- kein Algorithmus
- meist unerwünscht
- syntaktisch korrekt, wird vom Compiler meist nicht erkannt
- Vorsicht bei der Programmierung
- manchmal jedoch gewünschtes Verhalten, z.B. oft in eingebetteten Systemen oder Multitasking-Systemen
- manchmal auch mit Unterbrechung des Programmflusses sinnvoll
- Beispiel für Endlosschleife:

```
while (true) {  
    ...  
}
```

Unterbrechung des Programmflusses

- mit `break` wird Ausführung des innersten Blockes bzw. der innersten Schleife abgebrochen und mit Ausführung der Anweisung direkt nach dem Block bzw. der Schleife fortgesetzt

Unterbrechung des Programmflusses

■ in einer Endlosschleife

```
int i = 1;

while ( true ) {
    System.out.println(i);
    i++;
    if ( i > 10 )
        break;
}
```