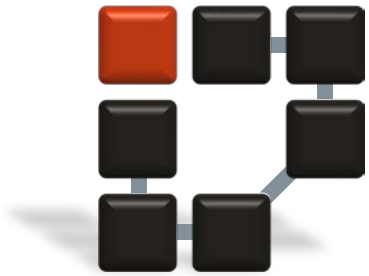
Informatik 1 für Nebenfachstudierende Grundmodul

Java – Anweisungen und Ablaufsteuerung

Kai-Steffen Hielscher
Folienversion: 24. Januar 2017



Informatik 7
Rechnernetze und
Kommunikationssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Inhaltsübersicht

- Kapitel 3 - Java
  - Einführung
  - Lexikalische Struktur
  - Datentypen und Variablen
  - Operatoren und Ausdrücke
  - **Anweisungen und Ablaufsteuerung**

Anweisungen und Ablaufsteuerung

- Anweisungen
- Blöcke
- Bedingte Ausführung
  - `if`
  - `switch`
- Schleifen
  - `for`
  - Abweisende Schleife: `while`
  - Nichtabweisende Schleife: `do`
  - Endlosschleifen
- Unterbrechung des Programmflusses
  - `break`
  - `continue`
  - `return` (später)

Anweisungen

- Deklarationen von Variablen
- Ausdrucksanweisungen
  - Zuweisungen von Werten an Variablen
  - Methodenaufrufe
- leere Anweisung
  - besteht nur aus einem Semikolon `;`

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:

```
{
    double x = 2.0;
    x = x + 1.0;
    {
        double y;
        y = x * 3.141592654;
        System.out.println(y);
    }
    System.out.println(x);
    {
        double z;
        z = x + 44.11;
        System.out.println(z);
    }
}
```

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

- 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

- `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

- 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`
  - Beispiel

```
int a, b;
…
switch ( a ) {
  case 1:
    b = 10;
  case 2:
  case 3:
    b = 20;
    break;
  case 4:
    b = 30;
    break;
  default:
    b = 40;
}
```

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
  - Beispiel

```
int x = 5;
int f = 1;

if(x > 0) {
    for(int i = 1; i <= x; i++) {
        f *= i;
    }
}

System.out.println("Die Fakultät von " + x + " ist " + f + ".");
```

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

- 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

- 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

- mit `break` wird Ausführung des innersten Blockes bzw. der innersten Schleife untergebrochen und mit Ausführung der Anweisung direkt nach dem Block bzw. der Schleife fortgesetzt
- zusätzlich lassen sich Marken (Bezeichner gefolgt von Doppelpunkt `:`) definieren, dann kann die Unterbrechung mittels
  `break marke;`
  über mehrere Hierarchieebenen erfolgen, nicht nur aus dem innersten Block
- macht den Programmablauf unübersichtlich, für Anfänger kompliziert, nach Möglichkeit vermeiden

Unterbrechung des Programmflusses

- mittels `continue` wird in Schleifen der aktuelle Schleifendurchlauf unterbrauchen, aber nicht die ganze Schleife beendet, sondern mit dem nächsten Durchlauf weitergemacht
  - bei `for` wird nach `continue` die Updateliste ausgeführt
- auch hier bei geschachtelten Schleifen Fortsetzung auf anderer Hierarchieebene mittels Marken möglich

Unterbrechung des Programmflusses

- in Methoden wird `return` verwendet, um Ausführung der Methode zu beenden und im aufrufenden Programmteil weiterzumachen
- Rückkehr aus einer Methode
- gegebenenfalls können dabei Rückgabewerte zurückgeliefert werden
- später mehr dazu