

# **JAVA**

## **Iterationen mit for**

# Iterationen

- Iterationen
  - ◆ Dienen der wiederholten Ausführung von Programmteilen, bis eine bestimmte Bedingung eingetreten ist
  - ◆ Java kennt folgenden Kontrollanweisungen für Iterationen: for, while (in zwei Varianten), weitere for Variante ab JDK1.5
  - ◆ Syntax und Semantik stammen noch von Programmiersprache C
- Beispiele für Anwendung wiederholte Berechnungen
  - ◆ Eine mathematische endliche (unendliche) Folge oder Reihe (bis zu einem bestimmten Glied) berechnen, z.B.

$$\sum_{i=0}^{\infty} \frac{1}{2^i} = 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \dots = 2$$

- ◆ Suche nach einem bestimmten Element in einer Sammlung von Daten, z.B. alle Primzahlen von 1 bis 1000 finden.
- ◆ Numerische Verfahren für Näherungslösungen, z.B. Quadratwurzel einer Zahl

```
for (int i = 5, k = 16, j = k + 13;  
     i < k && j > k; i = i + 1, k = k+1 ) {  
    // mache etwas  
}
```

```
for ( ; ; ) {  
}
```

```
for (int i; true; )  
    ;
```

- Mehrere Variablen genau eines Typs können – mit Komma separiert – deklariert werden. Die Deklarierten Variablen sind nur innerhalb der for-Schleife sichtbar.
- Mehrere durch Komma separierte Zuweisungen erlaubt; explizit mit = oder implizit mit i++ (Kurzschreibweise für i = i + 1, i ein Zahltyp).
- BoolescherAusdruck optional. Wenn er weggelassen wird bedeutet dies „true“.

```
for (VariablenDeklaration; BoolescherAusdruck; AnweisungenFor)  
  AnweisungsBlock
```

1. *VariablenDeklaration* wird genau einmal zu Beginn ausgeführt
  - Lokale Variablen werden angelegt und initialisiert
2. BoolescherAusdruck wird ausgewertet:
  - Falls `false`: for-Schleife beendet. Programmausführung führt Anweisungen nach der for-Schleife aus
  - Falls `true`:
    - Anweisungen im *AnweisungsBlock* werden sequentiell ausgeführt
    - Anweisungen in *AnweisungenFor* werden ausgeführt
    - Es geht weiter mit Schritt 2

# for-Schleife, Semantik

- **Beispiel:** Ganzen Zahlen von 1 bis 100 auf dem Bildschirm ausgeben

```
for (int i = 1; i <= 100; i = i + 1) {  
    System.out.println(i);  
}
```

1. Lokale Variable *i* wird angelegt und mit 1 initialisiert
2. Solange *i* noch nicht 100 ist
  - *System.out.println(i)* ausführen
  - *i = i + 1* ausführen
  - Weiter mit Schritt 2
- *Alternative Implementierungen:*

```
for (int i = 1; i <= 100;) {  
    System.out.println(i);  
    i = i + 1;  
}
```

```
int i = 1;  
for (; i <= 100;) {  
    System.out.println(i);  
    i = i + 1;  
}
```

# Iterationen

- ◆ Gegeben:  $n > 0$
- ◆ Gesucht: Wert von

$$\sum_{i=0}^n \frac{1}{2^i} = 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \dots + \frac{1}{2^i}$$

```
int n = 5;  
double d = 0.0;  
  
for (int i = 0; i <= n; i++) {  
    d = d + 1.0 / Math.pow(2, i);  
}
```

# Iterationen

- ◆ Bei  $n = 50$  ist  $d = 1.9999999999999999991$
- ◆ Bei  $n \sim 60$  ist  $d = 2.0$  und ändert sich nicht mehr
- ◆ Optimierung: Wenn  $d$  sich nach einem Durchlauf nicht mehr ändert, dann for-Schleife abbrechen
- ◆ Schleife wird bei  $i = 54$  abgebrochen

```
int n = 5;
double d = 0.0;
boolean beenden = false;

for (int i = 0; ! beenden && i <= n; i++) {
    double altesD = d;
    d = d + 1.0 / Math.pow(2, i);
    beenden = (altesD == d);
    // Hier ist eine Ausnahme von der Regel
    // == nie für double zu verwenden, da wir wissen
    // dass die Reihe konvergiert
}
```



# Iterationen

- ◆ Statt `Math.pow(2,i)` die 2er Potenzen selbst berechnen (vielleicht ist das schneller)

```
int n = 5;
double d = 0.0;
double zweiHochI = 1.0;
boolean beenden = false;

for (int i = 0; ! beenden && i <= n;
      i++, zweiHochI = zweiHochI * 2.0;) {
    double altesD = d;
    d = d + 1.0 / zweiHochI;
    beenden = (altesD == d);
}
```

- ◆ Verpacken in eine Funktion

```
public static void iteriere(int n) {  
    double d = 0.0;  
    double zweiHochI = 1.0;  
    boolean beenden = false;  
  
    for (int i = 0; ! beenden && i <= n;  
        i++, zweiHochI = zweiHochI * 2.0;) {  
        double altesD = d;  
        d = d + 1.0 / zweiHochI;  
        beenden = (altesD == d);  
    }  
  
    return d;  
}
```