EINI LogWing/WiMa

Einführung in die Informatik für Naturwissenschaftler und Ingenieure

Vorlesung 2 SWS WS 17/18

Dr. Lars Hildebrand
Fakultät für Informatik – Technische Universität Dortmund
lars.hildebrand@tu-dortmund.de
http://ls14-www.cs.tu-dortmund.de

Thema

- **Kapitel 3**
  Basiskonstrukte imperativer (und objektorientierter) Programmiersprachen

- **Unterlagen**
  - Echtle, Klaus und Michael Goedicke: *Lehrbuch der Programmierung mit Java*. Heidelberg: dpunkt-Verl, 2000. (→ ZB)
  - Gumm, Heinz-Peter und Manfred Sommer: *Einführung in die Informatik*, 10. Auflage. München: De Gruyter, 2012. (Kap. 2) (→ Volltext aus Uninetz)

Übersicht

- ✓ Variablen
- ✓ Zuweisungen
- ✓ (Einfache) Datentypen und Operationen
  - ✓ Zahlen
    `integer, byte, short, long; float, double`
  - ✓ Wahrheitswerte **`(boolean)`**
  - ✓ Zeichen **`(char)`**
  - ✓ Zeichenketten **`(String)`**
  - ✓ Typkompatibilität
- ✓ Kontrollstrukturen
  - ✓ Sequentielle Komposition, Sequenz
  - ✓ Alternative, Fallunterscheidung
  - ➢ Schleife, Wiederholung, Iteration:
    - ✓ while, do-while
    - ➢ for
- ▶ Verfeinerung
  - ▶ Unterprogramme, Prozeduren, Funktionen
  - ▶ Blockstrukturierung
- ▶ Rekursion

In diesem Kapitel:

- Prolog
- **Kontroll-strukturen**
  - Sequenz
  - Block
  - Alternative
  - **Iteration**

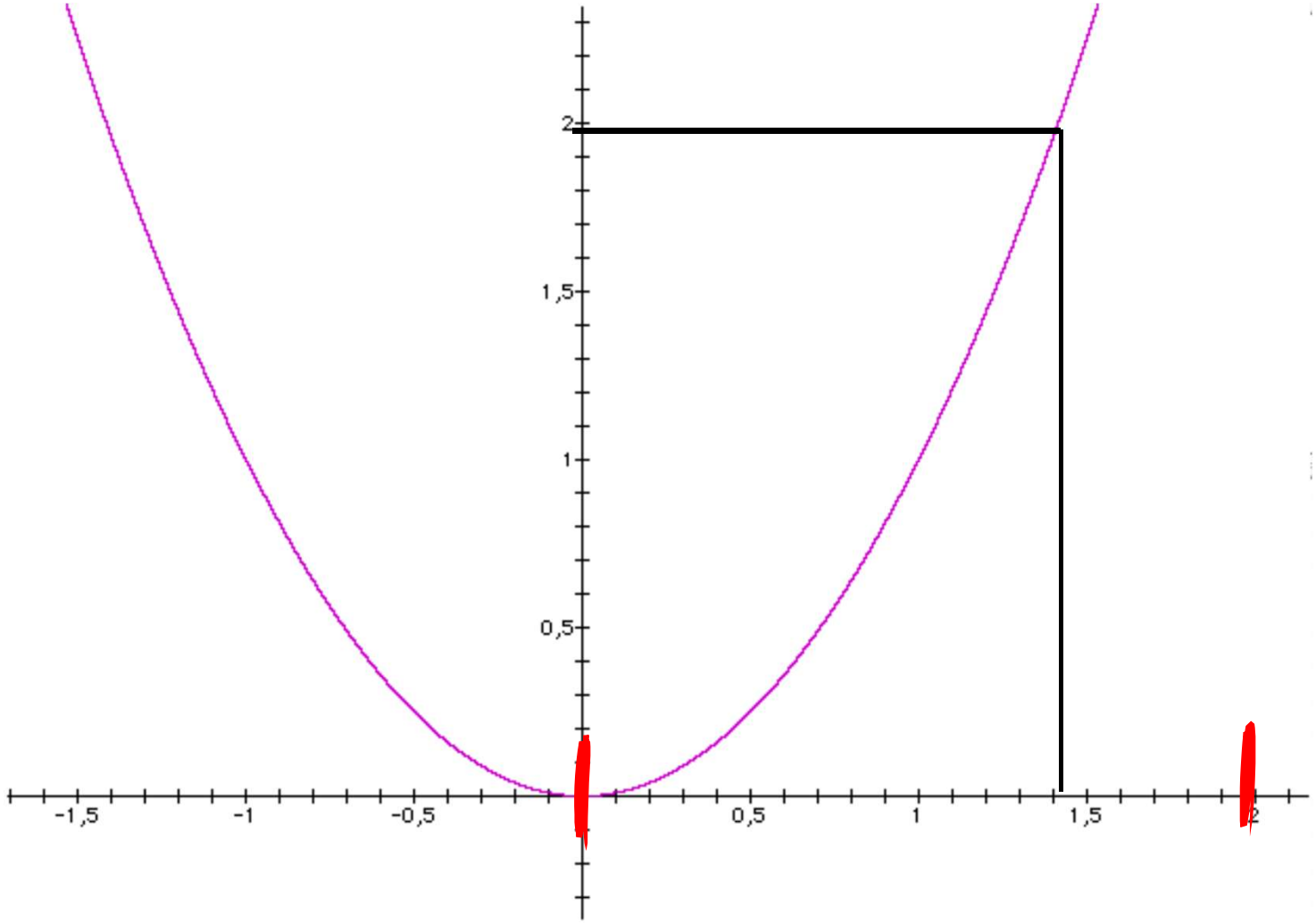
Beispiel: `do-while` (2) I

Beispiel: einfache Numerik-Funktionen

- Berechnung der Quadratwurzel **`sqrt`** für n > 0
- Nützlichkeit klar,
  - da in vielen Programmen unabhängig vom Kontext verwendbar.
  - daher auch in Bibliotheken (Libraries) stets verfügbar.
- Eine Berechnungsidee: Intervallschachtelung
  - Finde eine untere Schranke.
  - Finde eine obere Schranke.
  - Verringere obere und untere Schranke, bis der Abstand hinreichend gering geworden ist.
  - Etwas konkreter: Halbiere Intervall, fahre mit dem Teilintervall fort, das das Resultat enthält.

Beispiel: `do-while` (2) II

Quadratwurzel-Berechnung mittels Intervallschachtelung



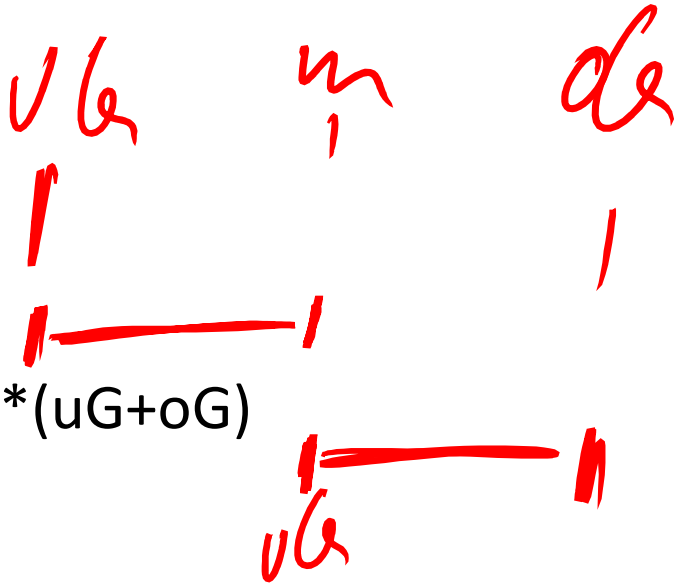
In diesem Kapitel:

- Prolog
- Kontroll-strukturen
  - Sequenz
  - Block
  - Alternative
  - Iteration

Beispiel: do-while (2) III

- Quadratwurzel-Berechnung mittels Intervallschachtelung
- Rückführung der Berechnung auf Quadrierung

- Start: Intervall [0,x+1],
  - uG =0;
  - oG = 3;
  - Mitte m = 0,5*(uG+oG)



- Algorithmus:
  - Berechne neue Mitte m = 0,5*(uG+oG)
  - Falls $m^2 > x$: oG = m
    sonst: uG = m
  - Abbruch: falls oG - uG < ε

Beispiel: do-while (2) IV


```
double x = 2.0,
       uG = 0,   oG = x + 1,  m,
       epsilon = 0.001;

do
{
  m = 0.5*(uG + oG);
  if (m*m > x)
     oG = m;
  else
     uG = m;
}

while (oG - uG > epsilon);

System.out.println (   "Wurzel " + x
                     + " beträgt ungefähr "
                     + m);
```

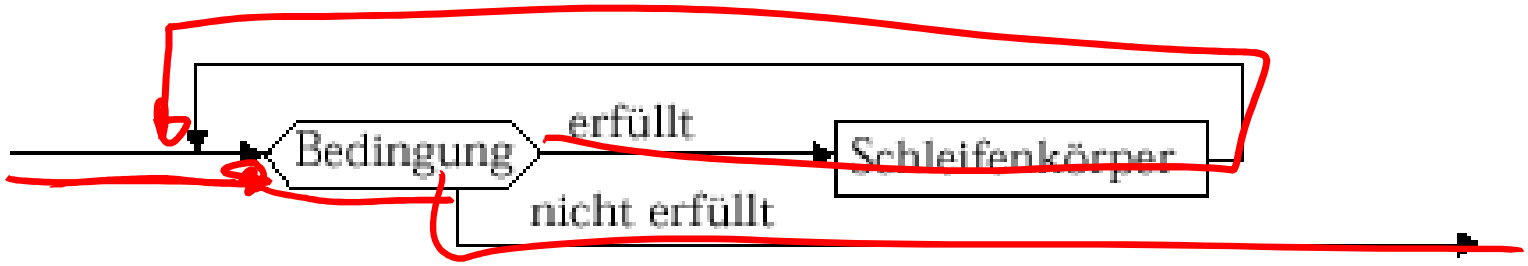
Wiederholung: Schleifen

- Drei Varianten:
  - `while` (Bedingung) { Anweisungsfolge }
  - `do` { Anweisungsfolge } `while` (Bedingung)
  - `for` (Initialisierung, Bedingung, Fortschritt) { Anweisungsfolge }
- Diese Vielfalt ist „nur“ durch Komfort begründet.
- Jede Schleife kann mittels jedes Typs programmiert werden:
  - Der Code sieht je nach Schleifentyp anders aus.
  - Das Problem gibt den geeigneten Schleifentyp vor.

Wiederholung: `while`-Schleife

```
while (Bedingung) { Anweisungsfolge }
```

- Grundsätzlich gilt, dass der Schleifenkörper solange wiederholt wird, wie die Bedingung wahr ist (auch 0-mal).
- Die Bedingung wird zu **`true`** oder **`false`** ausgewertet.
- Die Bedeutung kann auch durch ein Diagramm dargestellt werden (**Kontrollflussgraph**):

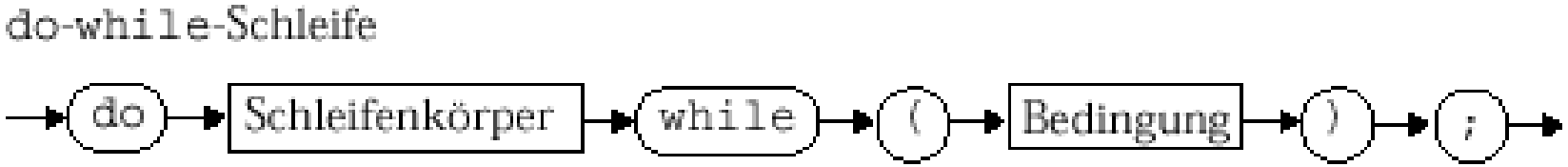


Echtle/Goedicke, Heidelberg: *Abb. 2–10*, S. 53 © dpunkt 2000.

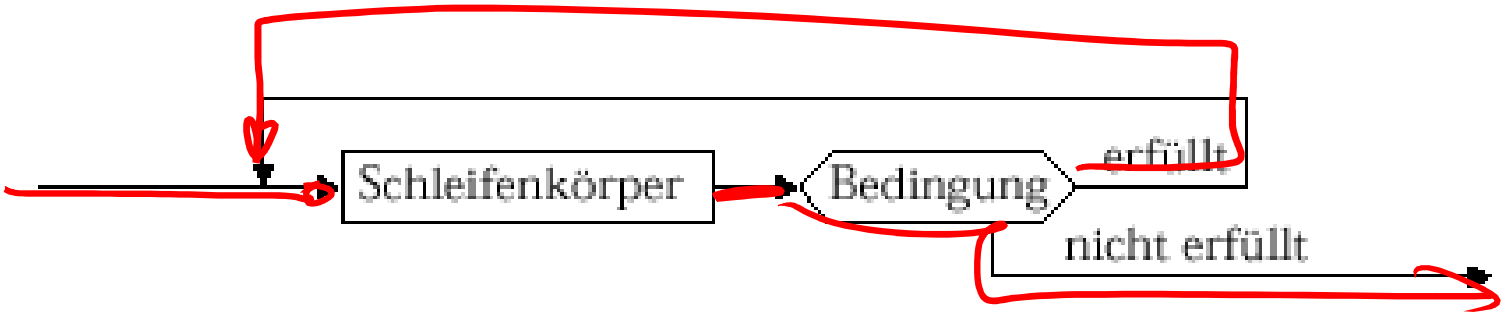
Wiederholung: do-while-Schleife

```
do { Anweisungsfolge } while (Bedingung);
```

- Durchlauf des Schleifenkörpers **mindestens 1 Mal**.
- Syntax und Semantik durch Diagramme:



Echtle/Goedicke, Heidelberg: *Abb. 2–12*, S. 56 © dpunkt 2000.



Echtle/Goedicke, Heidelberg: *Abb. 2–13*, S. 56 © dpunkt 2000.

Die 3 Teile von Schleifen

1. Vorbereitende Anweisungen:
   Deklaration von Variablen, Initialisierungen
2. Fortsetzungsanweisungen
3. Abfrage der Schleifenbedingung

- Bei Prüfung der Korrektheit eines (Teil-)Programms aus einer Schleife folgendes prüfen:
  - Werden die **Variablen**, die für die Schleifenbedingung gebraucht werden, **deklariert** und sinnvoll **initialisiert** ?
  - Werden die **Variablen**, die für die Schleifenbedingung gebraucht werden, innerhalb des Schleifenkörpers oder - sofern extra ausgewiesen - in den Fortsetzungsanweisungen **verändert**?
  - Ist eine Veränderung der Schleifenbedingung hin zum **Abbruch** der Schleife gesichert?

- Weiteres spezielles Schleifenkonstrukt: → `for`-Schleife

for-Schleife I

```
for (...; ...; ...)  { Anweisungsfolge }
```

- Die **for-Schleife** bietet eine direkte Syntax für diese drei Teile einer Schleife:
  - **Vorbereitende Anweisungen**
    - Variablenvereinbarungen, Initialisierungen
    - `int i = Startwert`
  - **Abfrage der Schleifenbedingung**
    - `i<= Endwert`
  - **Fortsetzungsanweisungen**
    - `i++`

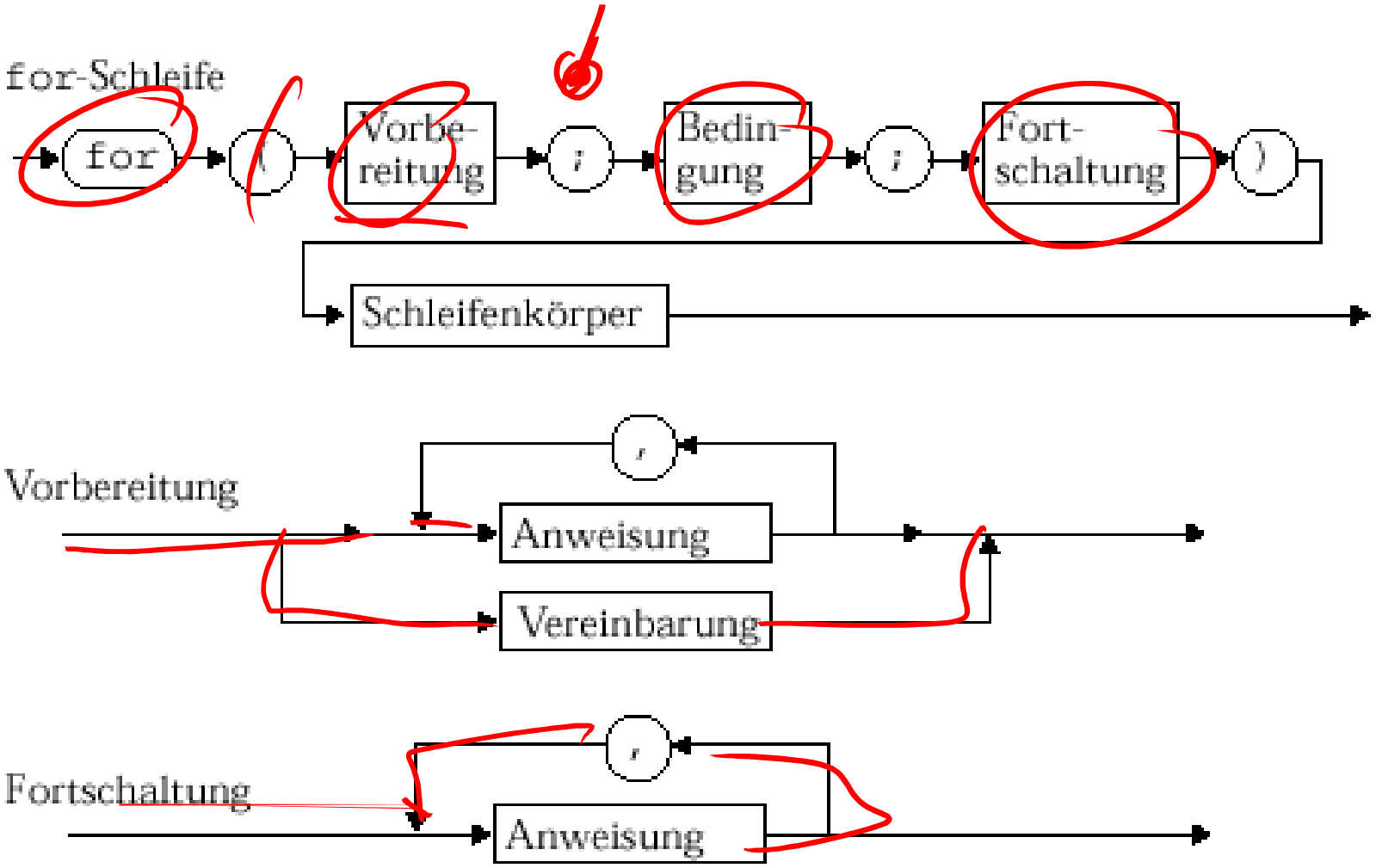
i++,
i = i+1,

Beispiel:

- `for (int i= Startwert; i<= Endwert; i++)`
  `{ ... }`

for-Schleife II

- Syntaxdiagramme für die **for**-Schleife:



Echtle/Goedicke, Heidelberg: *Abb. 2–14*, S. 58 © dpunkt 2000.

In diesem Kapitel:

- Prolog
- **Kontroll-strukturen**
  - Sequenz
  - Block
  - Alternative
  - **Iteration**

Komplexe `for`-Schleifen

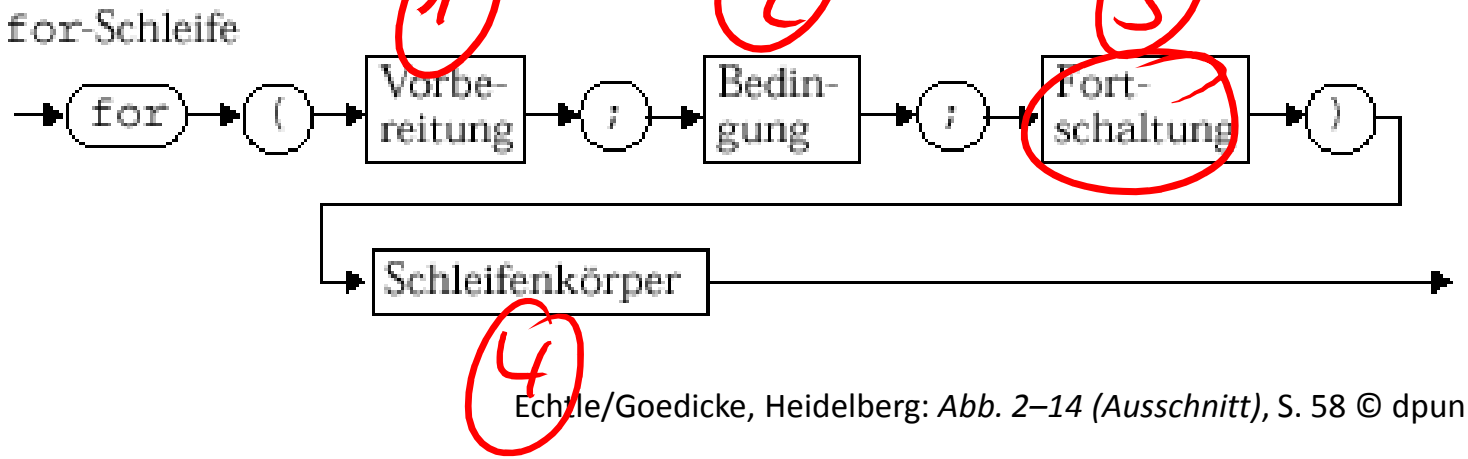
- Beispiel

```
for (int i=2,j=10 ; // Start
     i<=5 ;         // Ende
     i++ ,j--)      // Weiter
  { ... }           // Rumpf
```

- Bemerkungen:
  - Die schleifenlokal vereinbarten Variablen sind nur **innerhalb der Schleife gültig**, dürfen aber auch außerhalb **nicht noch einmal deklariert** werden!
  - Die Schleifenvariablen müssen nicht unbedingt lokal vereinbart sein, es verbessert jedoch den Überblick über die Verwendung von Variablen (Lokalität der Verwendung).
  - Mehrfache Fortsetzungsanweisungen sind eher unüblich.

Die Bedeutung der `for`-Schleife ...

- Syntax



Echtle/Goedicke, Heidelberg: *Abb. 2–14 (Ausschnitt)*, S. 58 © dpunkt 2000.

- Semantik

Echtle/Goedicke, Heidelberg: *Abb. 2–15*, S. 59 © dpunkt 2000.

- Auch die **`for`**-Schleife kann 0-mal ausgeführt werden.

... und ein paar Beispiele: (1)

Das zuvor betrachtete Beispiel:

- mit **`while`**:

```
int  i = 1,  a = 2;
while (i < 100)
  {
     a = 4*a;
     i++;
  }
```

- mit **`for`**:

```
for (int i = 1, a = 2; i < 100; i++)
{   a = 4*a; }
```

... und ein paar Beispiele: (2)

Drucken einer Funktionstabelle:

$f(x) = x^2 - 8$

für $x \in \{-5,-4,-3, \ldots, 10,11,12\}$

- Programmausschnitt (mit **getrennter** Deklaration & Initialisierung):

```
int  x, y;
for (x = -5; x <= 12 ; x++)
{
   y = x*x -8;
   System.out.println("x= " + x +
                      " y= " + y);
}
```

Einige Vorschläge für `for`-Schleifen

- Allgemeiner Aufbau:

```
for ( Vorbereitung; Bedingung; Fortsetzung) {...}
```

- In 1er-Schritten durch die Schleife laufen:

```
for ( int counter = Anfangswert; counter <= Endwert;
counter ++ ) {...}
```

- In Schritten der Größe *n* durch die Schleife laufen:

```
for ( int counter = Anfangswert; counter <= Endwert;
counter +=n ) {...}
```

counter = counter + n;

- Rückwarts zählen:

```
for ( int counter = Anfangswert; counter >= Endwert;
counter-- ) {...}
```

Verfügbarkeit von Laufvariablen

- Zur Erinnerung: Im Rumpf von **`for`**-Schleifen sind die Laufvariablen verfügbar.

- Damit kann z.B. eine innere Schleife von dem Wert einer Laufvariablen der äußeren Schleife abhängig gemacht werden:

```
for (int i=10; i<= 30; i=i+10)
  for (int j=1; j<= 4; j++)
    System.out.println(i + " " + j + ", ");
```

. . .

- Ausgabe:

  10 1, 10 2, 10 3, 10 4, 20 1,..., 30 2, 30 3, 30 4, ...

Schleifen

Zusammenfassung

- Drei Varianten
  - `while (Bedingung) { Anweisungsfolge }`
  - `do { Anweisungsfolge } while (Bedingung)`
  - `for (Initialisierung; Bedingung; Fortsetzung) { Anweisungsfolge }`
- Diese Vielfalt ist „nur“ durch Komfort begründet.
- Jede Schleife kann mittels jedes Typs programmiert werden:
  - Der Code sieht je nach Schleifentyp anders aus.
  - Das Problem gibt den geeigneten Schleifentyp vor.

`continue` & `break`

Abbruchmöglichkeiten für Schleifendurchläufe

- Abbruch bisher nur bei Prüfung der Bedingung vor/nach Durchlauf des Schleifenkörpers
- Zusätzliche Möglichkeiten durch spezielle Anweisungen:
  - Mit **`continue`** kann die Ausführung eines Schleifenrumpfs abgebrochen und mit der **nächsten Iteration** (nach Prüfung der Schleifenbedingung) fortgesetzt werden.
  - Mit **`break`** kann die komplette Ausführung einer Schleife beendet werden.

Beispiel zu `continue & break` (1)

`continue`

Aufgabe: Zahlen von 1 - 20 sollen ausgegeben werden, die nicht durch 3 teilbar sind.

```
for (int i = 1; i <= 20; i++)
{
    if ( (i % 3) == 0) continue;
    System.out.print(i + ", ");
}
```

Beispiel zu `continue & break` (2)

break

Aufgabe: Zahlen von 1 - 20 sollen aufsummiert werden, bis die Summe zum ersten Mal größer als 100 ist.

```
int summe = 0;
for (int i = 1; i <= 20; i++)
{
    summe = summe + i;
    if ( summe > 100) break;
}
```

In diesem Kapitel:
- Prolog
- Kontroll-strukturen
  - Sequenz
  - Block
  - Alternative
  - Iteration

continue & break

Abbruchmöglichkeiten für Schleifendurchläufe

- Besonderheit in Java: Anweisungen können benannt werden:
  - `Benennung: Anweisung;`
  - Beispiel: `Startwert: istPrimzahl = true;`
- Um bei geschachtelten Schleifen eine äußere Schleife für **`break`** oder **`continue`** zu identifizieren, muss der jeweilige Schleifenkopf mit einem Namen versehen und in der **`break`**– bzw. **`continue`**-Anweisung angegeben werden.

Beispiel zu continue & break (3)

Unüblich schwieriger Fall!

```
01  ...
02  Aussen: while (bed1)
03                 {
04                    Innen: while (bed2)
05                    {
06                          if (bed3) continue Aussen;
07                          if (bed4) continue Innen;
08                          if (bed5) break Aussen;
09                    }
10                 }
11  ....
```

Vorsicht mit `continue & break`!

- Nur in übersichtlichen Fällen und sparsam verwenden!

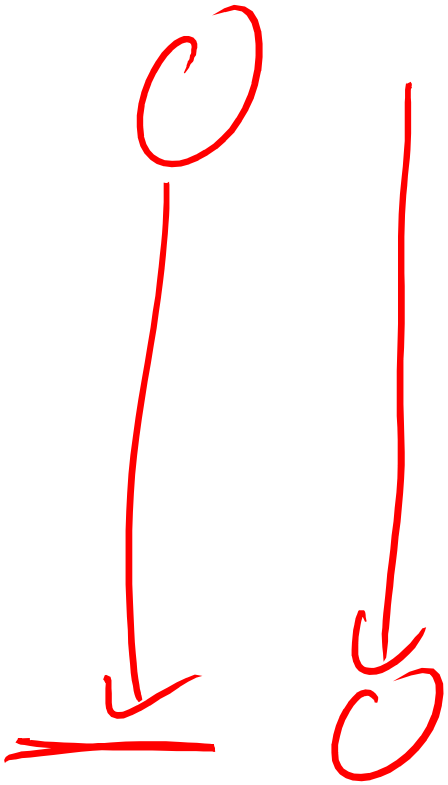
- Typische Einsatzgebiete:
  - hoch optimierte Bibliotheken
  - schnelles Verlassen von Schleifen, wenn Resultat klar ist

- Programme werden durch die Verwendung dieser Sprachkonstrukte schnell unübersichtlich!

Kontrollstrukturen: Iteration

Artikel im EINI-Wiki:

- → **Schleife**
  - →Schlüsselwörter
    - →break-Statement
    - →continue-Statement
- → **Kopfgesteuerte Schleife**
- → **Fußgesteuerte Schleife**
- → **Zählschleife**
- → **Endlosschleife**
- → **Laufvariable**



Zwischenstand

- ✓ Variablen
- ✓ Zuweisungen
- ✓ (Einfache) Datentypen und Operationen
  - ✓ Zahlen
    `integer, byte, short, long; float, double`
  - ✓ Wahrheitswerte **`(boolean)`**
  - ✓ Zeichen **`(char)`**
  - ✓ Zeichenketten **`(String)`**
  - ✓ Typkompatibilität
- ✓ Kontrollstrukturen
  - ✓ Sequentielle Komposition, Sequenz
  - ✓ Alternative, Fallunterscheidung
  - ✓ Schleife, Wiederholung, Iteration: while, do-while, for
- ➢ Verfeinerung
  - ➢ Unterprogramme, Prozeduren, Funktionen
  - ➢ Blockstrukturierung
- ▶ Rekursion

Übersicht

Vielen Dank für Ihre Aufmerksamkeit!

Nächste Termine

- Nächste Vorlesung – WiMa 23.11.2017, 08:15
- Nächste Vorlesung – LogWing 24.11.2017, 08:15

Bleiben Sie noch 5 Minuten!