EINI LW

Einführung in die Informatik für Naturwissenschaftler und Ingenieure

Vorlesung 2 SWS WS 11/12

Dr. Lars Hildebrand
Fakultät für Informatik – Technische Universität Dortmund
lars.hildebrand@udo.edu
http://ls1-www.cs.uni-dortmund.de

technische universität dortmund

Dr. Lars Hildebrand

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

- **Unterlagen**
  - Gumm/Sommer, Kapitel 2
  - Echtle/Goedicke, Einführung in die Programmierung mit Java, dpunkt Verlag

Zwischenstand

Dr. Lars Hildebrand

- Variablen
  - Bezeichner, Datentyp, Speicherort, Wert
- Zuweisungen
  - Zuweisungen müssen typverträglich sein
- (Einfache) Datentypen und Operationen
  - `integer (byte, short, long; float, double)`
  - `boolean`
  - `char`
  - `String`
  - Typkompatibilität
- Kontrollstrukturen
  - Sequentielle Komposition, Sequenz
  - Alternative, Fallunterscheidung
  - Schleife, Wiederholung, Iteration (while, do while, for)
- Verfeinerung
  - Unterprogramme, Prozeduren, Funktionen
  - Blockstrukturierung
- Rekursion

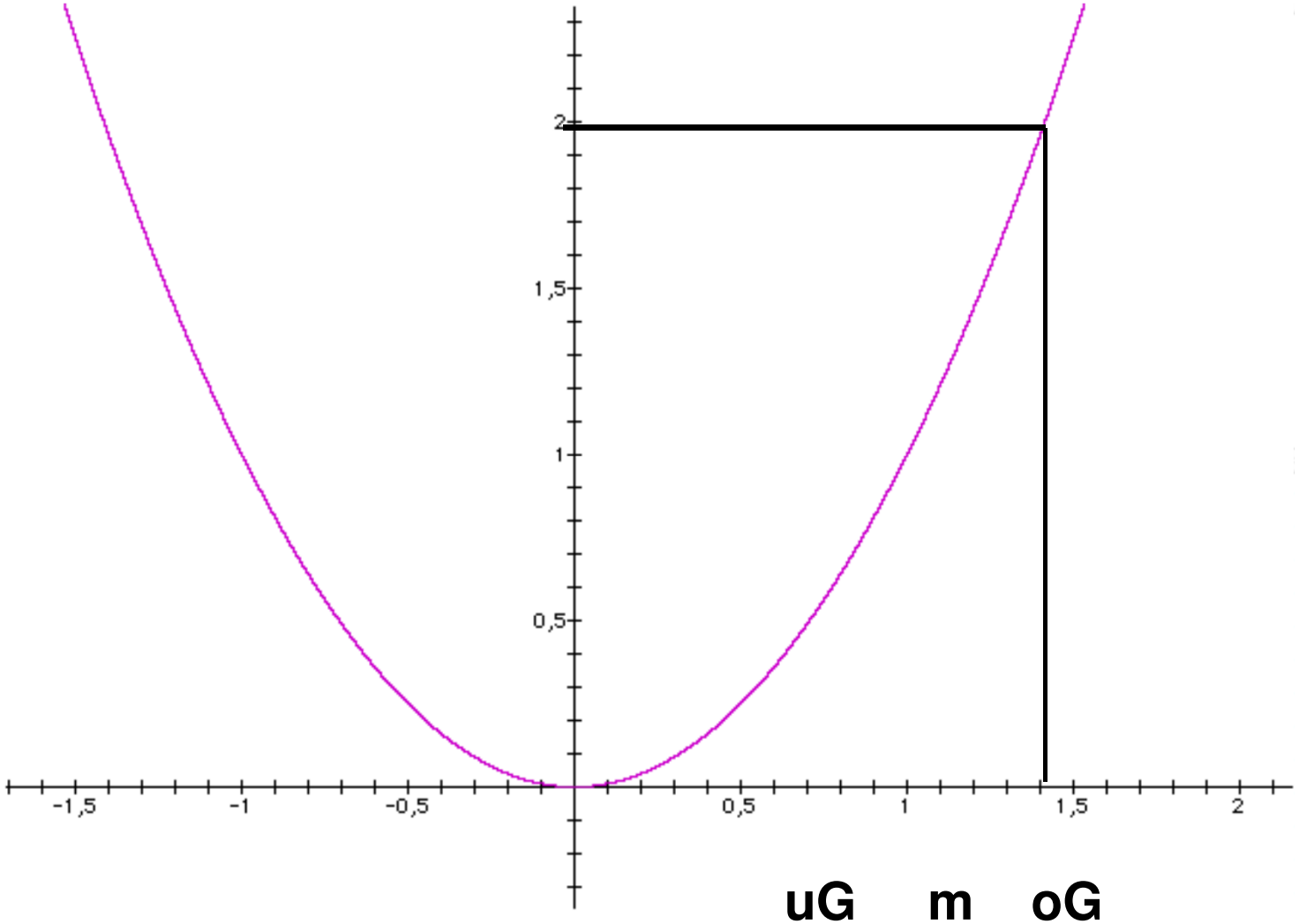
Beispiel: einfache Numerik Funktionen

- Berechnung der Quadratwurzel `sqrt( float n)` für n > 0:
- Nützlichkeit klar,
  - 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 demjenigen Teilintervall fort, das das Resultat enthält.

technische universität dortmund

Dr. Lars Hildebrand

Quadrat-Wurzel Berechnung mittels Intervallschachtelung



Beispiel do-while (2)

Dr. Lars Hildebrand

- Quadrat-Wurzel Berechnung mittels Intervallschachtelung
- Rückführung der Berechnung auf Quadrierung

- Start: Intervall [0,x+1], 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)

technische universität dortmund
Dr. Lars Hildebrand

```
double x = ...,
       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

Dr. Lars Hildebrand

- Drei Varianten
  - `while` (Bedingung) { Anweisungsfolge }
  - `do` { Anweisungsfolge } `while` (Bedingung)
  - `for` (Initialisierung, Bedingung, Fortschritt) { Anweisungsfolge }

- Diese Vielfalt ist „nur“ durch Komfort begründet

- Die allgemeinste Form ist die `while` -Schleife

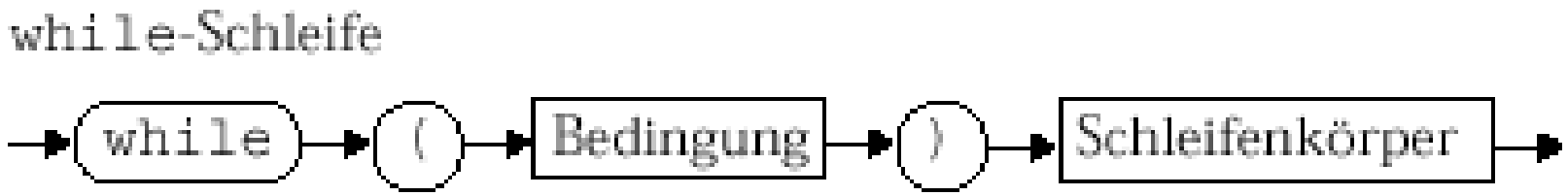


Abb. 2–9 *Syntax der while-Schleife*

Lehrbuch der Programmierung mit Java, Echtle Goedicke, Heidelberg, © dpunkt 2000

Wiederholung – `while`-Schleife

Dr. Lars Hildebrand

`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**)

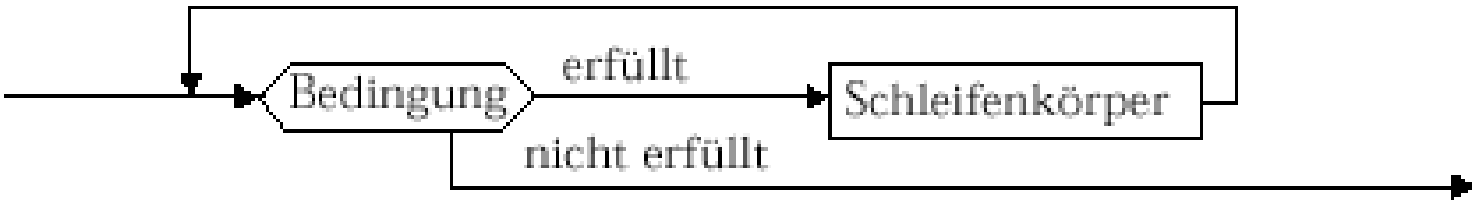


Abb. 2–10 Semantik der while-Schleife

Lehrbuch der Programmierung mit Java, Echtle Goedicke, Heidelberg, © dpunkt 2000

Wiederholung – `do-while`-Schleife

technische universität dortmund

Dr. Lars Hildebrand

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

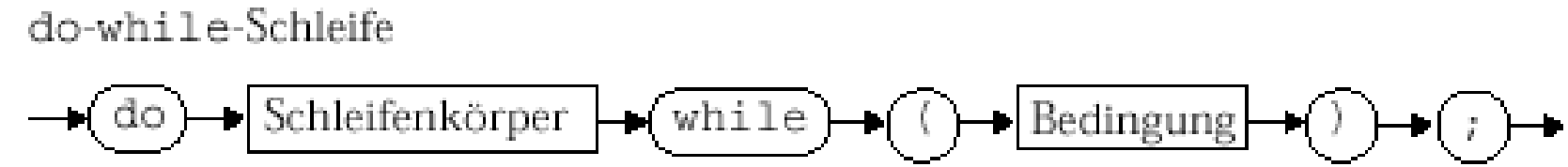


Abb. 2–12 *Syntax der do-while-Schleife*

Lehrbuch der Programmierung mit Java, Echtle Goedicke, Heidelberg, © dpunkt 2000

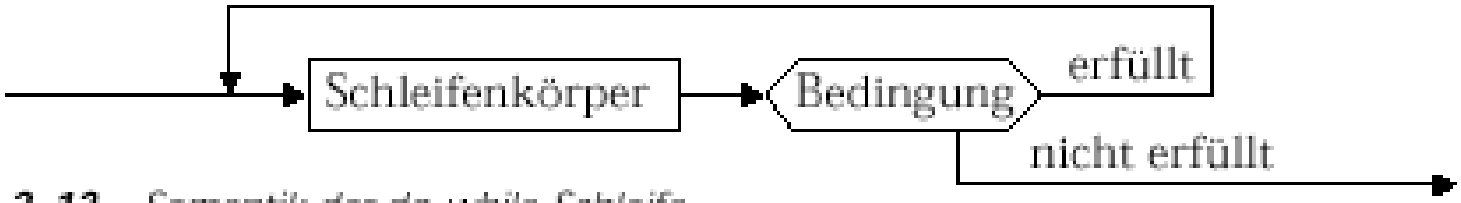


Abb. 2–13 *Semantik der do-while-Schleife*

Lehrbuch der Programmierung mit Java, Echtle Goedicke, Heidelberg, © dpunkt 2000

Die 3 Teile von Schleifen

Dr. Lars Hildebrand

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

- Daher sollten bei der Prüfung der Korrektheit eines (Teil-) Programms aus einer Schleife auch insbesondere diese Aspekte geprüft werden:
  - 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 fortschaltenden Anweisungen verändert?
  - Ist eine Veränderung der Schleifenbedingung hin zum Abbruch der Schleife gesichert?

- Weiteres spezielles Schleifenkonstrukt: -> for-Schleife

`for`-Schleife

- 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`**
  - Fortschaltende Anweisungen
    - **`i++`**

Also Beispiel:

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

- Syntax-Diagramme für die **for**-Schleife

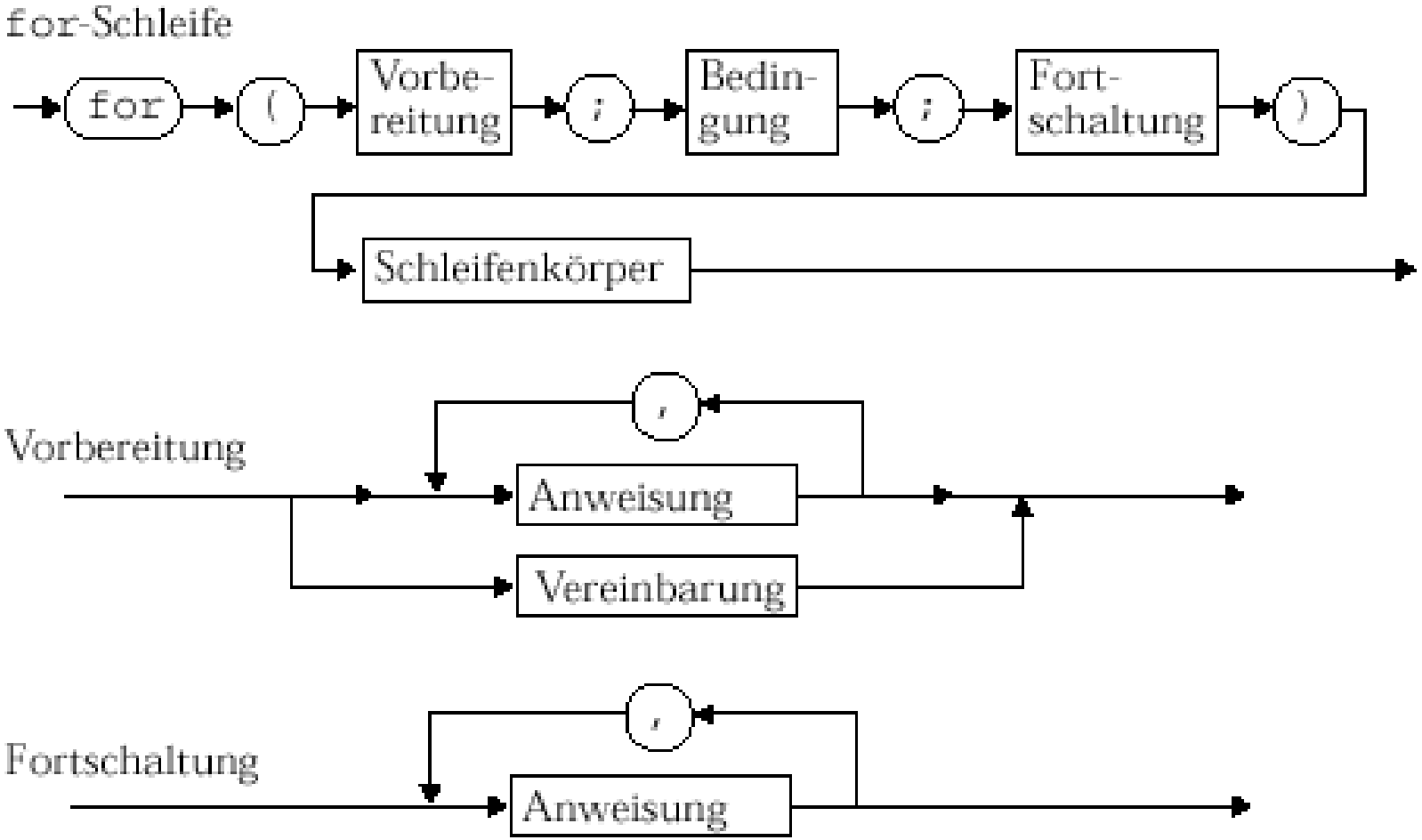


Abb. 2–14 *Syntax der for-Schleife*

Lehrbuch der Programmierung mit Java, Echtle Goedicke, Heidelberg, © dpunkt 2000

Komplexe `for`-Schleifen

Dr. Lars Hildebrand

- Beispiel

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

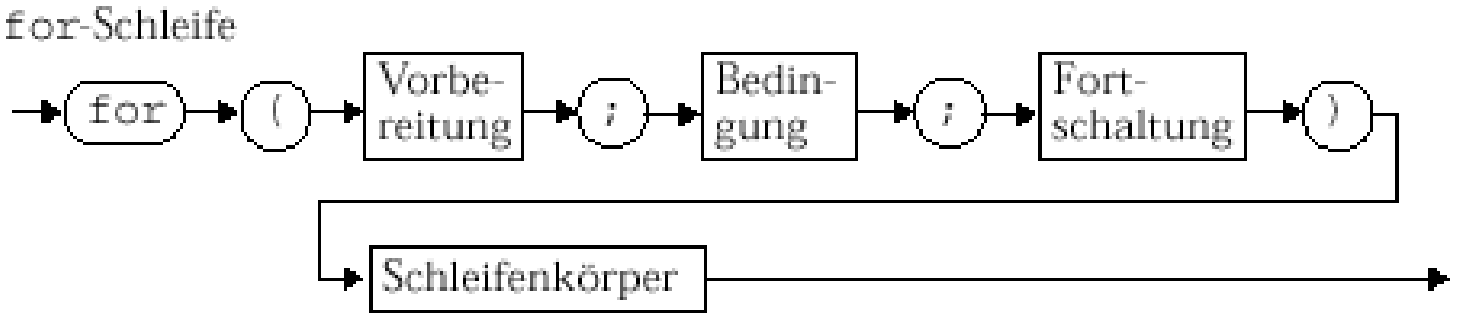
- Bemerkungen:
  - Die Schleifen-lokal vereinbarten Variablen sind nur innerhalb der Schleife gültig, dürfen aber auch außerhalb nicht noch einmal deklariert werden!
  - Die Schleifenvariablen müssen nicht unbedingt lokal vereinbart sein, es erleichtert jedoch, den Überblick über die Verwendung von Variablen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

Die Bedeutung der for-Schleife ...

technische universität dortmund

Dr. Lars Hildebrand

- Syntax



- Semantik

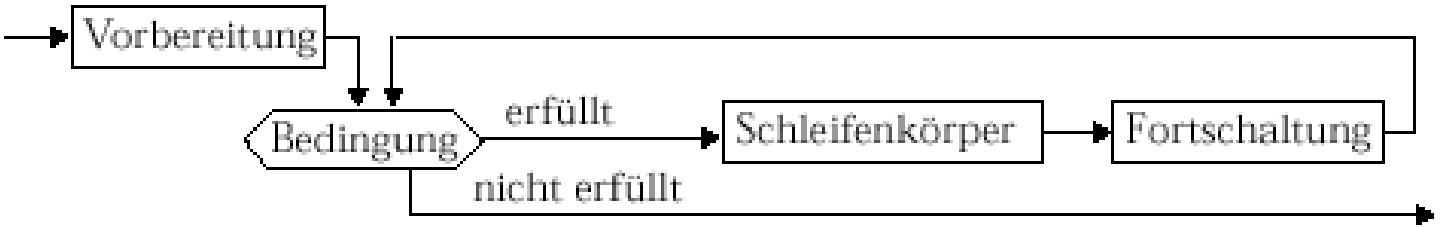


Abb. 2–15 *Semantik der for-Schleife*

Lehrbuch der Programmierung mit Java, Echtle Goedicke, Heidelberg, © dpunkt 2000

- Auch die for-Schleife kann null Mal ausgeführt werden.

technische universität dortmund

Dr. Lars Hildebrand

Das zuvor betrachtete Beispiel

- Mit Hilfe von **`while`**:

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

- Mit Hilfe von **`for`**:

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

technische universität dortmund

Dr. Lars Hildebrand

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);
     }
```

technische universität dortmund
Dr. Lars Hildebrand

Zusammenfassung

- Drei Varianten
  - `while` (Bedingung) { Anweisungsfolge }
  - `do` { Anweisungsfolge } `while` (Bedingung)
  - `for` (Initialisierung, Bedingung, Fortschritt) { Anweisungsfolge }
- Diese Vielfalt ist „nur“ durch Komfort begründet

Schleifen dürfen verschachtelt sein

- Z.B. eine `for` - Schleife als Rumpf einer anderen `for` - Schleife:

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

innere Schleife

äußere Schleife

- Ein Durchlauf der äußeren Schleife ist eine vollständige Ausführung der inneren Schleife

i	j	
10	1	
	2	
	3	
	4	// innere Schleife einmal vollständig
20	1	// äußere Schleife in 2. Iteration

...

Verfügbarkeit von Laufvariablen

Dr. Lars Hildebrand

- Zur Erinnerung: Im Rumpf von for-Schleifen sind die Laufvariablen verfügbar
  - Z.B. kann damit 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.print (i + " " + j + ",   ");
System.out.println ();
```

äußere Schleife

innere Schleife

  - Ausgabe:
    10 1, 10 2, 10 3, 10 4, 20 1,..., 30 2, 30 3, 30 4, ...

technische universität dortmund
Dr. Lars Hildebrand

Abbruchsmöglichkeiten für Schleifendurchläufe

- 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`

Dr. Lars Hildebrand

`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 + ", ");
}
```

technische universität dortmund
Dr. Lars Hildebrand

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;
}
```

technische universität dortmund
Dr. Lars Hildebrand

Abbruchsmöglichkeiten 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`

Unüblich schwieriger Fall!

```
...
Aussen: while (bed1)
                {
                    Innen: while (bed2)
                    {
                            if (bed3) continue Aussen;
                            if (bed4) continue Innen;
                            if (bed5) break Aussen;
                     }
                }
....
```

Vorsicht mit `continue & break`

Dr. Lars Hildebrand

- 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

Zwischenstand

Dr. Lars Hildebrand

- Variablen
  - Bezeichner, Datentyp, Speicherort, Wert
- Zuweisungen
  - Zuweisungen müssen typverträglich sein
- (Einfache) Datentypen und Operationen
  - `integer (byte, short, long; float, double)`
  - `boolean`
  - `char`
  - `String`
  - Typkompatibilität
- Kontrollstrukturen
  - Sequentielle Komposition, Sequenz
  - Alternative, Fallunterscheidung
  - Schleife, Wiederholung, Iteration
- Verfeinerung
  - Unterprogramme, Prozeduren, Funktionen
  - Blockstrukturierung
- Rekursion

technische universität dortmund

Dr. Lars Hildebrand

Vielen Dank für Ihre Aufmerksamkeit!

Nächste Termine

- Nächste Vorlesung 25.11.2011, 08:30