EINI LW/WiMa

Einführung in die Informatik für Naturwissenschaftler und Ingenieure

Vorlesung 2 SWS WS 15/16

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

Thema

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

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

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**
- Verfeinerung
  - Unterprogramme, Prozeduren, Funktionen
  - Blockstrukturierung
- Rekursion

In diesem Kapitel:

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

Kontrollstrukturen: Sequenz

- Sequenz
  - „einfachste“ Kontrollstruktur
  - Trennzeichen zwischen Anweisungen: ;
  - Zuweisungen können aneinandergereiht werden:
  - z.B. a = 3 ; b = a + 4 ;

- Anmerkungen
  - Einrückung beibehalten
  - möglichst **nicht** mehrere Anweisungen in eine Zeile
  - Lesbarkeit ist wichtig!

Block

Block: zusammengehörende Anweisungsfolge

- wird durch `{ . . . }` als zusammengehörend gekennzeichnet.
- sinnvoll z.B. bei Verzweigungen, um mehr als eine Anweisung je Situation angeben zu können.
- Blöcke erlauben in vielen Sprachen die Deklaration von Variablen, die nur innerhalb des Blockes zur Verfügung stehen.
- Begrenzungssymbole können je nach Sprache unterschiedlich sein (**`begin`**, **`end`**), die Idee ist jedoch stets gleich.

Kontrollstrukturen: Alternativen

Alternativen beruhen auf Fallunterscheidungen

- Verzweigungen steuern den Programmablauf abhängig von Bedingungen:
  - Zuweisungen werden von Bedingungen abhängig sein
  - Bedingungen werden am häufigsten in Form der **`if`**-Anweisungen formuliert:

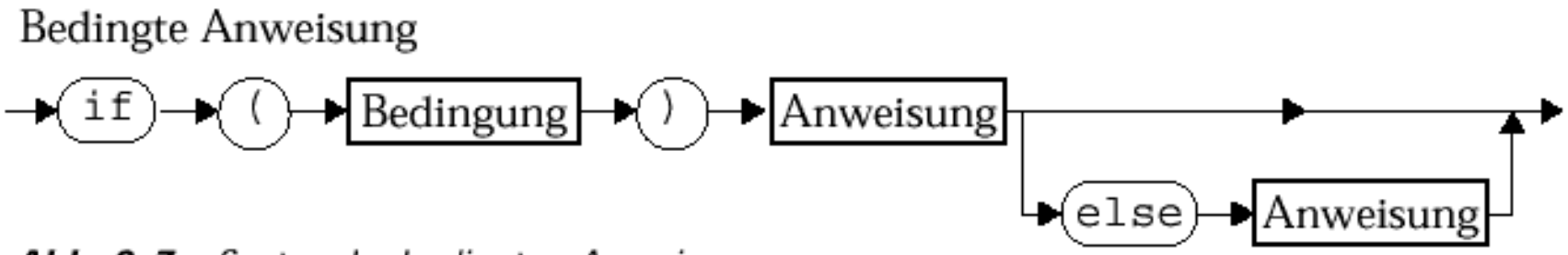


Abb. 2–7 *Syntax der bedingten Anweisung*

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

Bedingungen

- Eine **Bedingung** ist durch einen **Booleschen Ausdruck** gegeben:
  - Einfache oder komplexe Boolesche Ausdrücke
    - a<=b
    - a> 3 && v || b != c && !w
  - Beachte die Typisierung der Variablen!

- Die Bedingung wird **stets** in runde Klammern eingeschlossen (a<=b) ....

- **Bedeutung**:
  - Falls die Auswertung wahr ergibt wird die nächste Anweisung ausgeführt. Ein möglicher **`else`**–Zweig wird nicht ausgeführt.
  - Falls die Auswertung der Bedingung falsch ergibt, wird die erste Anweisung nicht ausgeführt, sondern – falls vorhanden – die Anweisung nach dem Schlüsselwort **`else`**

Beispiel: Fallunterscheidungen 1

```
int g = 5;
int k = 1;

if (g > k)
    System.out.println("g ist größer");
else
    System.out.println("g ist kleiner oder gleich");
```


In diesem Kapitel:

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

Beispiel: Fallunterscheidungen 2

```
int g,k = 0;
g = ...;
if (g == 1)
    k = k + 100;
else if (g == 2)
    k = k + 1000;
System.out.println("k = " + k);
```

- Frage: Was wird ausgegeben für die Eingabe
  - 1 ?
  - 2 ?
  - 3 ?

Beispiel: Fallunterscheidungen 3

Anweisungsfolge - Block

```
double winkel = ...;

if (winkel > 90.0 && winkel < 180.0)
    {
      System.out.println ("stumpfer Winkel");
      winkel = 180.0 - winkel;
    }
```

Verschachtelung von `if`-Anweisungen

```
if (...)
    if (...)
    else if ( ... )
```

- Hier tritt das Problem auf, wie ein **`else`** gebunden wird, wenn es im Prinzip zu mehreren **`if`**s gehören könnte.
  - In Java (und auch in vielen anderen Programmiersprachen) gilt:
  - Bindung immer an das innere **`if`** , es sei denn, es werden **`{ }`** gesetzt.

Beispiele zu verschachtelten `if`s

```
int  i = ...,  j = ...;
if (i == 5)
   if (j == 5)
     System.out.println ("i und j sind 5");

   else
     System.out.println ("nur i ist 5");
else
   if (j == 5)
     System.out.println ("nur j ist 5");
```

Beispiele zu verschachtelten ifs

```
int  i = ...,  j = ...;
if (i == 5)
   { if (j == 5)
     System.out.println ("i und j sind 5");
   }
else
  System.out.println ("i ist nicht 5");
```

```
if (j == 5)
    System.out.println ("j ist 5");
```


In diesem Kapitel:

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

Einrücken ist sinnvoll

Unübersichtliche Variante


```
double winkel = ...;
if (winkel <   90.0)
  System.out.println ("spitzer Winkel");
else if (winkel ==  90.0)
  System.out.println ("rechter Winkel");
else if (winkel <  180.0)
  System.out.println ("Stumpfer Winkel");
else if (winkel == 180.0)
  System.out.println ("gestreckter" +"Winkel");
```

Einrücken ist sinnvoll

Übersichtliche Variante

```
double winkel = ...;
if (winkel <   90.0)
    System.out.println ("spitzer Winkel");
else if (winkel ==  90.0)
      System.out.println ("rechter Winkel");
else if (winkel <  180.0)
   System.out.println ("Stumpfer Winkel");
else if (winkel == 180.0)
   System.out.println("gestr. Winkel");
```

In diesem Kapitel:

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

Das `switch` - Statement

Auswahl aus einer gegebenen Menge von Alternativen mittels eines `int`-Wertes:

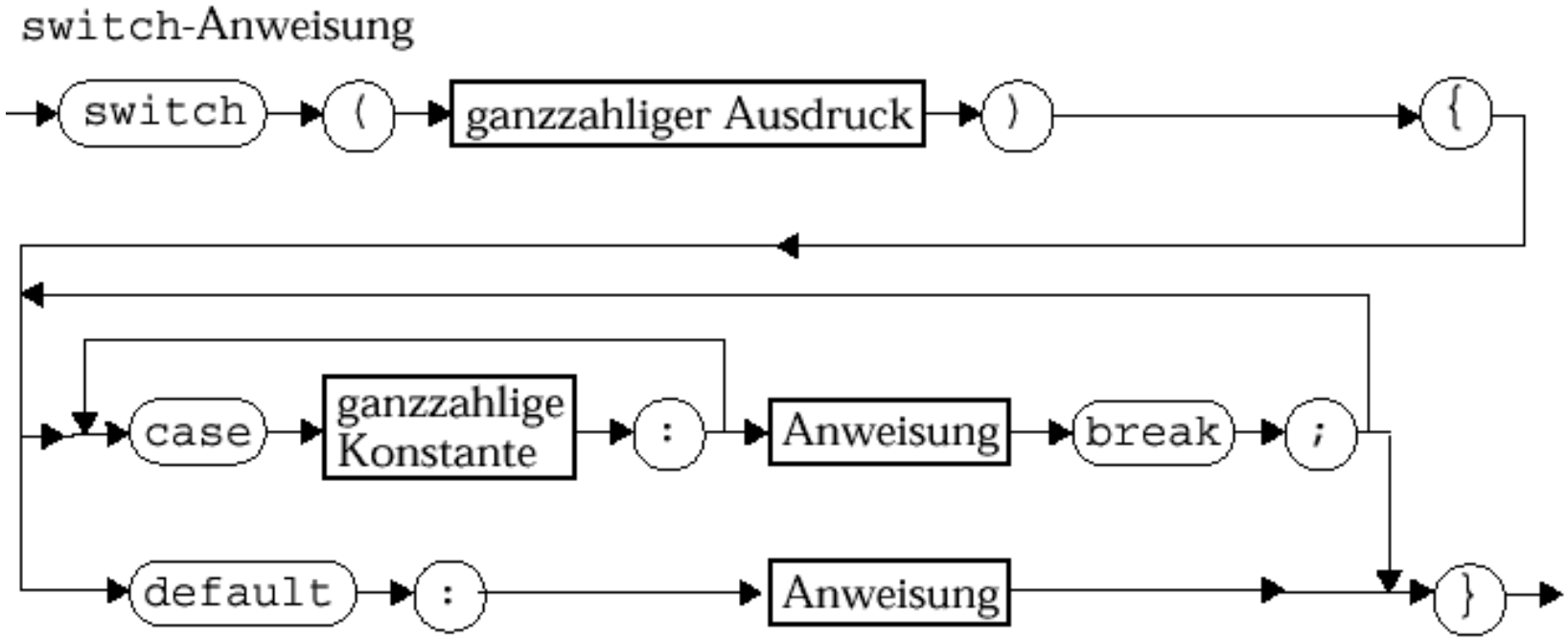


Abb. 2–8 *Syntax der switch-Anweisung*

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

In diesem Kapitel:

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

Beispiel für switch

```
int monat = 4;
int quartal;

switch (monat)
{
 case 1: quartal = 1; break;
 case 2: quartal = 1; break;
 case 3: quartal = 1; break;
 case 4: quartal = 2; break;
  . . .
  . . .
 case 11: quartal = 4; break;
 case 12: quartal = 4; break;
}
```

Beispiel für `switch`

```
int monat = 4;
int quartal;

switch (monat)
{
 case 1:  case 2:  case 3:  quartal = 1; break;
 case 4:  case 5:  case 6:  quartal = 2; break;
 case 7:  case 8:  case 9:  quartal = 3; break;
 case 10: case 11: case 12: quartal = 4; break;
}
```

- Bedingungen dürfen mehrfach vorkommen

Beispiel für `switch`

```
int monat = 4;
int quartal;

switch (monat)
{
 case 1:  case 2:  case 3:  quartal = 1; break;
 case 4:  case 5:  case 6:  quartal = 2; break;
 case 7:  case 8:  case 9:  quartal = 3; break;
 default: quartal = 4;
}
```

- `default` wird erreicht, wenn keine der Bedingungen zutrifft

Beispiel für `switch`

```
char buchstabe ='x';

switch (buchstabe)
{
 case 'a':
 case 'e':
 case 'i':
 case 'o':
 case 'u': System.out.println("Vokal!"); break;
 default: System.out.println("Konsonant!");

}
```

- jeder primitive Datentyp, der implizit in einen `int`-Wert umgewandelt werden kann, eignet sich als Kriterium

switch – Nur eine abkürzende Schreibweise


```
int zahl = 2;
char zeichen;

if(zahl==1) zeichen = '1';
 else if(zahl==2) zeichen = '2';
 else if(zahl==3 || zahl==4) zeichen = 'x';
 else zeichen = '?';
```

```
int zahl = 2;
char zeichen;

switch (zahl)
{
    case 1: zeichen = '1'; break;
    case 2: zeichen = '2'; break;
    case 3: case 4: zeichen ='x'; break;
    default: zeichen = '?';
}
```

Regeln für switch

Bemerkungen

- Die Bedingung wird vollständig ausgewertet
- Zur Fallunterscheidung dienen nur Konstanten
- Wird eine Übereinstimmung mit einer Konstanten gefunden, wird bei der zugehörigen Anweisung fortgesetzt
- Wird keine Übereinstimmung gefunden, wird bei **`default`** fortgesetzt.
- Die Reihenfolge von **`case`** und **`default`** ist beliebig
- **`break`** beendet die **`switch`**-Anweisung sofort
- Für Bedingung und Konstanten sind nur folgende Datentypen zugelassen
  - **`byte, short, int`**
  - **`char`**

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**
- Verfeinerung
  - Unterprogramme, Prozeduren, Funktionen
  - Blockstrukturierung
- Rekursion

Iteration

- Bisher sind die besprochenen Programme einmal durchgelaufen
  - jede Anweisung wurde höchstens einmal ausgeführt

- Beispiele für Wiederholungen:
  - Mathematische Folgen und Reihen
  - Verarbeitung wiederkehrender Vorgänge (Buchungen...)
  - Primzahltest:
    - Ist die Zahl n eine Primzahl?
    - teste ob 2,3,..., $\sqrt{n}$ Teiler von n sind.

Schleifen

- 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

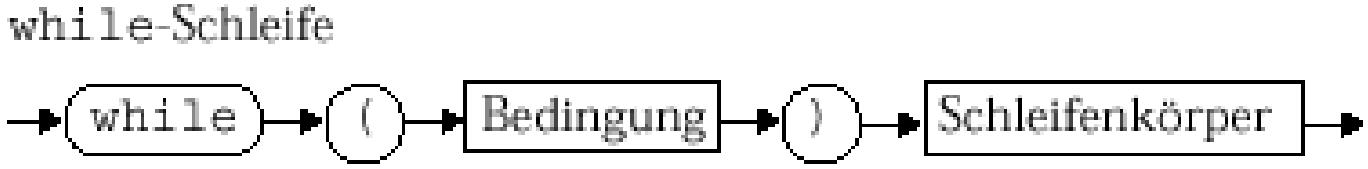


Abb. 2–9 *Syntax der while-Schleife*

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

`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**)

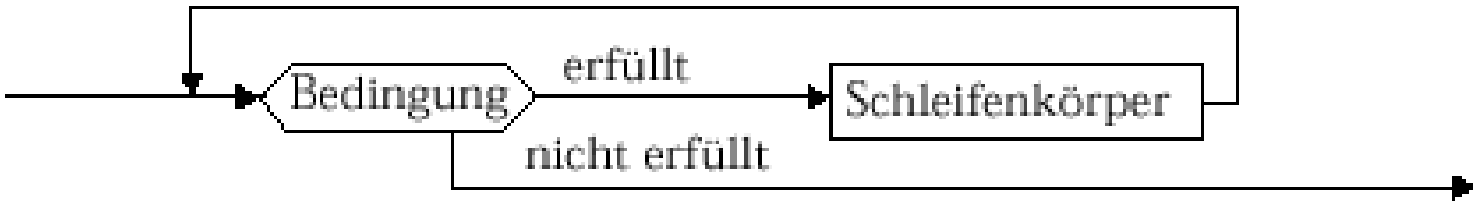


Abb. 2–10 *Semantik der while-Schleife*

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

Beispiel: Reihenberechnung

```
int  i = 1,  a = 2;
```

```
while (i < 100)
{
   a = 4*a;
  System.out.println("i=" + i + "\t" + "a=" + a);
  i++;
};
```

- In 3 Zeilen werden 99 Ausführungen von Zeilen beschrieben.
- Kleine Fehler haben große Auswirkungen:
  z.B.: i statt i++
- Die häufigsten Fehler in Schleifen
  - Bedingung verändert sich nicht oder ist falsch.
  - Bedingung signalisiert falsches Ende.
  - Falsche Initialisierung.

Beispiel: Primzahltest (1)

Algorithmus-Idee

- Teste, ob 2,3,..., √n Teiler von n sind (kann natürlich optimiert werden!)

Umsetzung

- Wir prüfen ein konkretes n
- Solange ***kein Teiler gefunden*** und ***die Grenze nicht erreicht***: erhöhe den Teiler um eins
- Die Bedingung „kein Teiler gefunden“ wird in Boolescher Variablen **`istPrimzahl`** gespeichert

```
while ( teiler <= wurzel  &&  istPrimzahl == true )
   if (n % teiler == 0)
       istPrimzahl = false;
     else
    teiler++;
```

Beispiel: Primzahltest (2)

```
01 int  n, wurzel,  teiler = 2;
02 boolean  istPrimzahl = true;
03
04 n = 42;
05
06 wurzel = (int) java.lang.Math.sqrt((float) n);
07
08 while ( (teiler <= wurzel) && (istPrimzahl == true) )
09  if (n % teiler == 0)
10              istPrimzahl = false;
11    else
12    teiler++;
13
14 System.out.println (n + " prim: " + istPrimzahl);
```

Schleifen sind schwieriges Programmkonstrukt

- Einfaches aber effizientes Verfahren ist die Darstellung der Werteverläufe über Tabellen:

n	Wurzel	Teiler	istPrimzahl
21	4	2	true
21	4	3	**false**

2 Iterationen

n	Wurzel	Teiler	istPrimzahl
37	6	2	true
37	6	3	true
37	6	4	true
37	6	5	true
37	6	6	**true**

5 Iterationen

- Auch hier: ggfs. Klammern und Einrückung zur Erhöhung der Lesbarkeit
- Schleifen sind schwierig, aber ohne sie kommt man nicht aus
  - Warum schwierig ?
  - Warum nötig ?

do-while-Schleife

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

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

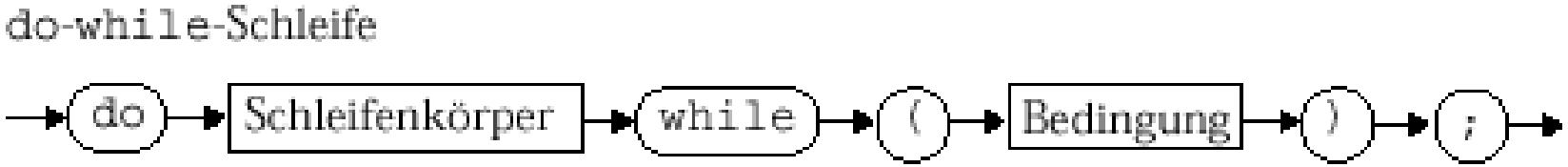


Abb. 2–12 *Syntax der do-while-Schleife*

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

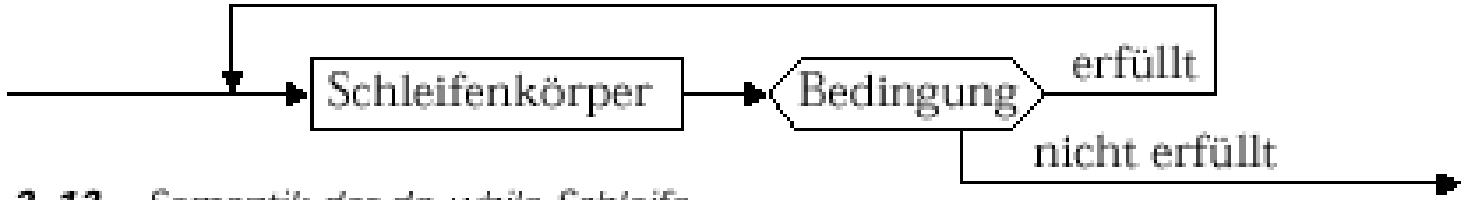


Abb. 2–13 *Semantik der do-while-Schleife*

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

Beispiel do-while (1)

```
import java.util.Scanner;

public class Beispiel {

  public static void main(String[] args) {

      Scanner scan = new Scanner(System.in);

      int  summe = 0,  anzahl = 0;

      do {
           summe = summe + scan.NextInt();
           anzahl++;
      } while (summe <= 100);

      System.out.println("Summe: " + summe +
                         ", Anzahl: " + anzahl);
  }
}
```

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

Beispiel do-while (2)

Beispiel: einfache Numerik Funktionen

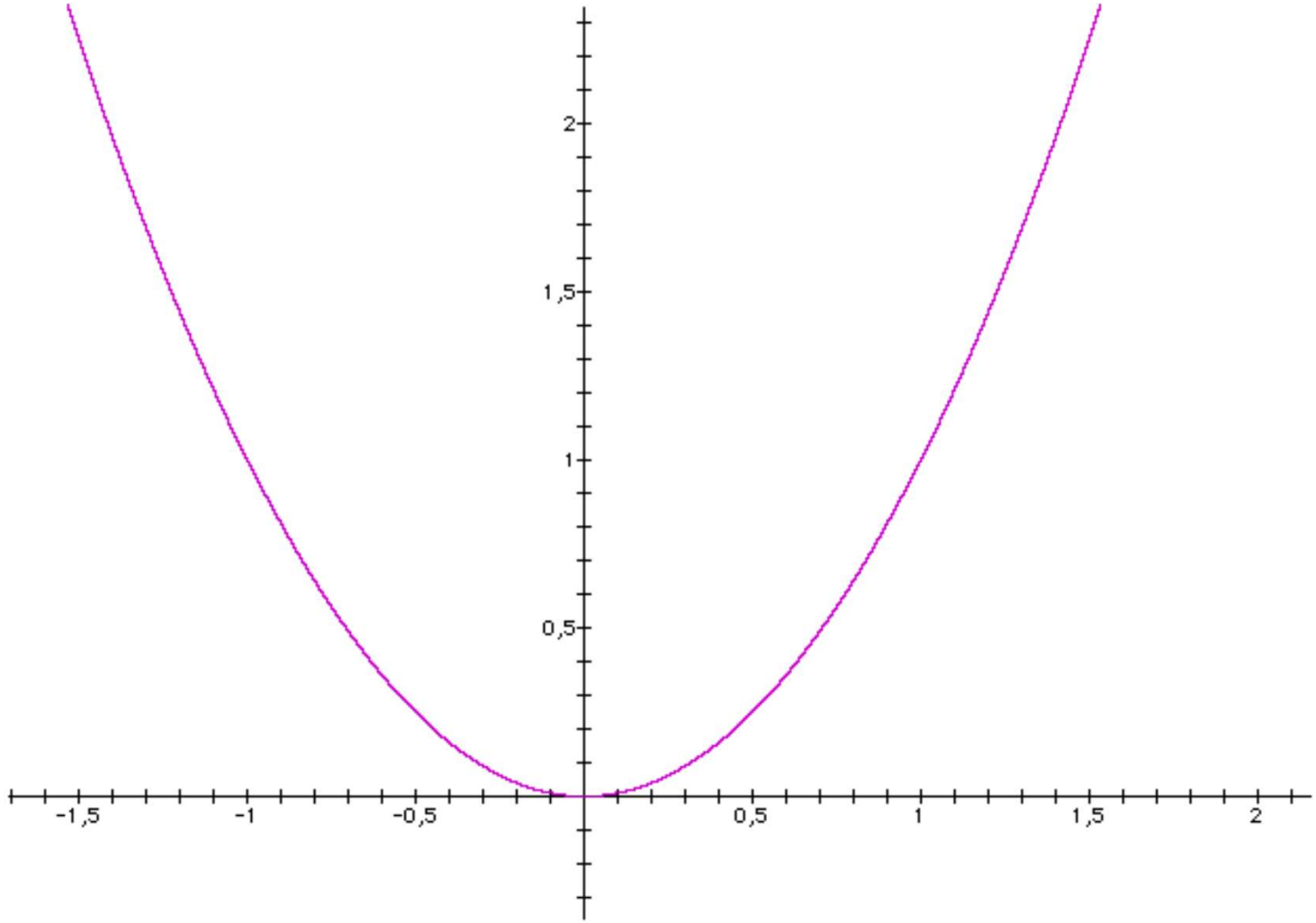
- Berechnung der Quadratwurzel **`sqrt`** für n > 0

- 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.

Beispiel do-while (2)

Quadrat-Wurzel Berechnung mittels Intervallschachtelung



Beispiel do-while (2)

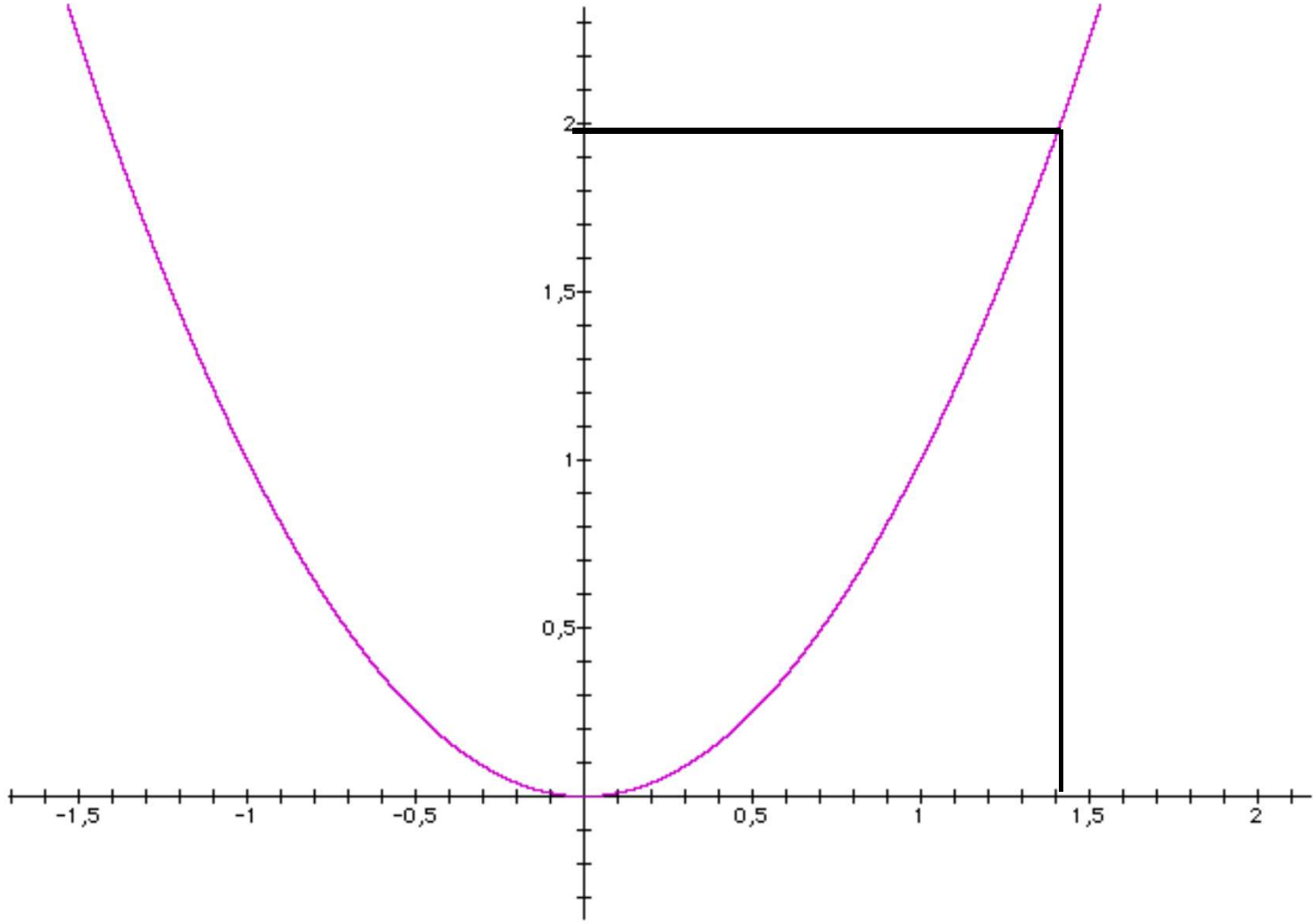
- Quadrat-Wurzel Berechnung mittels Intervallschachtelung
- Rückführung der Berechnung auf Quadrierung

- 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)

Quadrat-Wurzel Berechnung mittels Intervallschachtelung



Beispiel do-while (2)

```
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);
```

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

Zwischenstand

- 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

Vielen Dank für Ihre Aufmerksamkeit!

Nächste Termine

- Nächste Vorlesung – WiMa 26.11.2015, 08:15
- Nächste Vorlesung – LogWIng 27.11.2015, 08:15