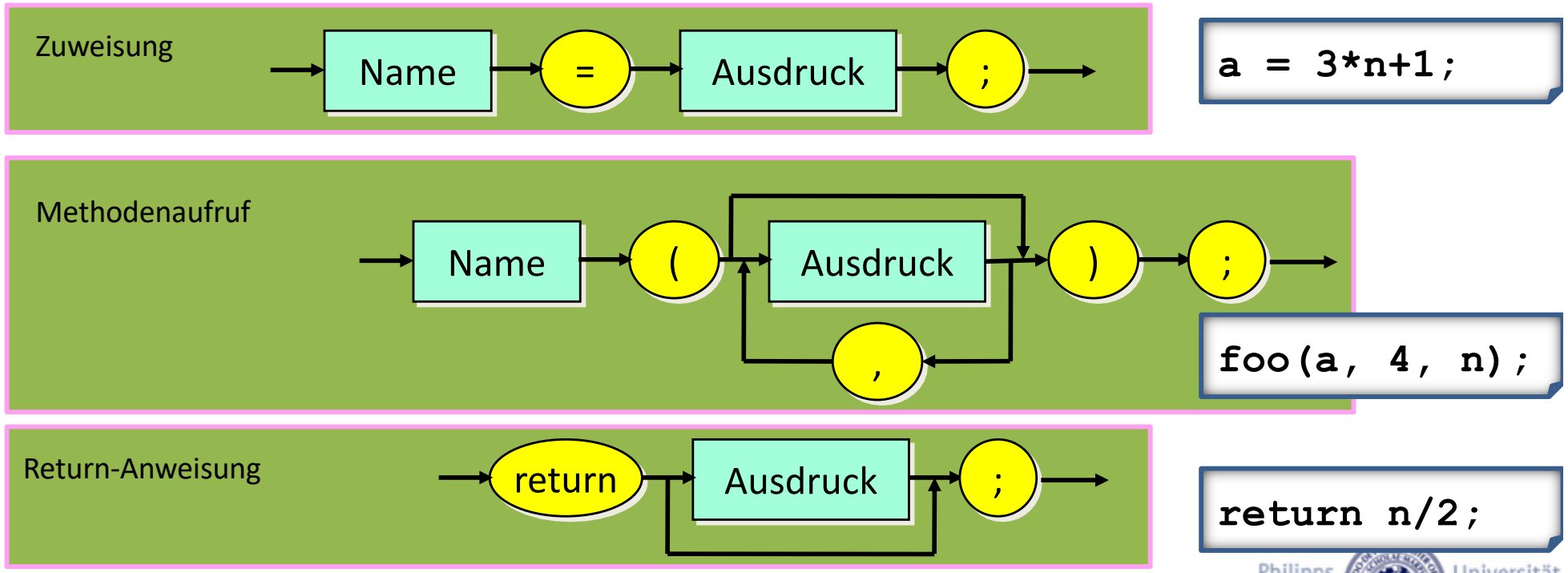
2.2.4 Erste Anweisungen

- Wie bei allen höheren Programmiersprachen gibt es auch in Java einfache und strukturierte Anweisungen, die auch als Kontrollstrukturen bezeichnet werden.
- Einige einfache Anweisungen wie z. B. die Variablendeklaration, Zuweisung und Methodenaufruf haben wir bereits behandelt.
  - Eine atomare Anweisung wird stets mit einem Semikolon abgeschlossen.



return-Anweisung in Methoden

- Methoden mit einem echten Ergebnistyp (ungleich void) müssen durch eine return-Anweisung beendet werden.
  - Der Wert in dem Ausdruck nach dem Schlüsselwort return wird als Ergebnis der Methode an den Aufrufer geliefert.
- Methoden mit dem Schlüsselwort void können durch eine leere return-Anweisung (ohne Angabe eines Werts bzw. Ausdrucks) beendet werden.
  - Ansonsten enden sie mit dem letzten Befehl im Methodenrumpf.

```
void tutNix(double x) {
        System.out.println("Hallo Tutnix " + x);
        return;
}
```

2.3 Elementare Kontrollstrukturen

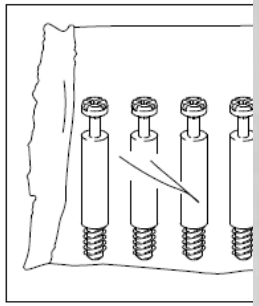
- Kontrollstrukturen sind Bestandteile von Algorithmen, die den Ablauf der Schritte innerhalb eines Algorithmus dynamisch steuern.
  - Aus einer endlichen Beschreibung können unendlich lange Prozesse entstehen.
- Dem Prozessor muss die Bedeutung der Kontrollstrukturen klar sein. Ansonsten würde zu einem Algorithmus und einer Eingabe nicht der korrekte Prozess erzeugt.
- Wichtige Kontrollstrukturen:
  - Sequenz (Anweisungsblock) – bereits kennengelernt
  - Selektion (bedingte Anweisung)
  - Schleifen

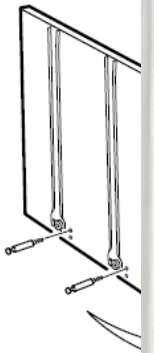
Motivation

HEMNES_Kommode_8.pdf - Adobe Acrobat Pro

File Edit View Window Help

6

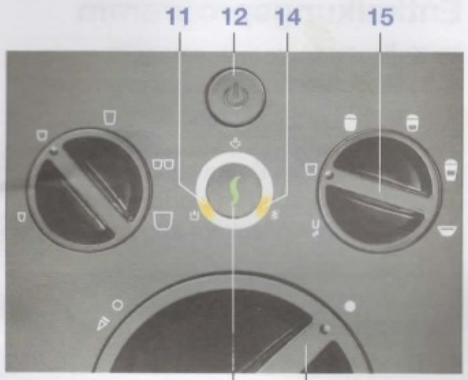




DE

Abschnitt 1 - Entkalken

1. Stellen Sie den rechten Drehknopf (15) auf das Symbol für Kaffee oder Heißwasser.
2. Stellen Sie sicher, dass die leere Padkassette eingeschoben ist und drehen Sie den Stellknopf (17) auf Position ●.
3. Stellen Sie ein mindestens 1,8 Liter fassendes Auffanggefäß (z. B. Topf) in die Mitte auf das Aufstellgitter.
4. Schalten Sie das Gerät an der Ein/Aus-Taste ⏻ (12) ein.
5. Drücken Sie gleich nach dem Einschalten beide Tasten (12) und (16) gleichzeitig und halten Sie sie so lange gedrückt, bis die Pumpe anläuft. Das Entkalkungsprogramm läuft, die Verkalkungsanzeige (14) leuchtet dauerhaft. Nach ca. 35 Minuten ist der erste Abschnitt des Entkalkungsprogramms beendet. Die Füllstandsanzeige (11) blinkt.



Abschnitt 2 - Spülen

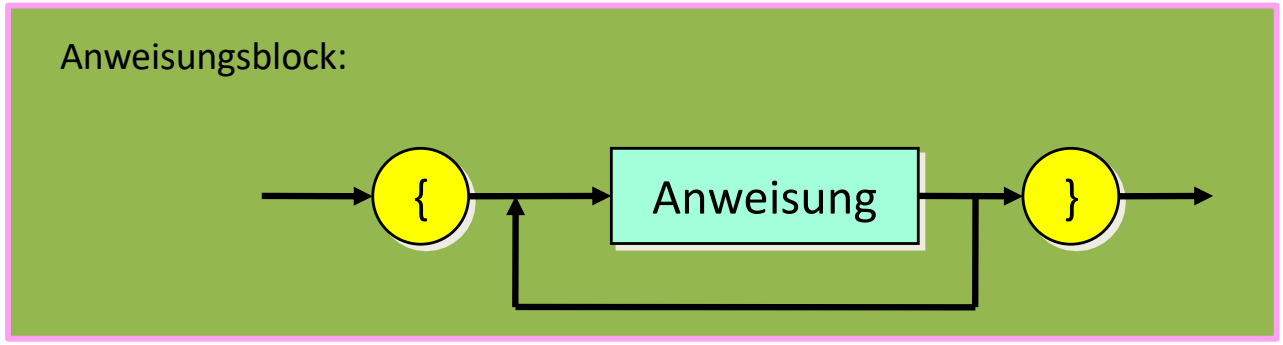
1. Füllen Sie den Wassertank bis zu dessen **max**-Marke mit Leitungswasser und setzen Sie ihn wieder in das Gerät ein.
2. Stellen Sie das entleerte Auffanggefäß wieder in die Mitte auf das Aufstellgitter.
3. Drücken Sie die grün leuchtende Start/Stopp-Taste (16). Das Spülen beginnt.
4. Sobald die Füllstandsanzeige (11) wieder blinkt, spülen Sie noch einmal. Wiederholen Sie dazu die Schritte 1. bis 3. Nach dem zweiten Spülen schaltet das Gerät in den Aus-Zustand.

2.3.1 Wiederholung: Anweisungsblock

- Ein Anweisungsblock (auch Sequenz genannt) ist eine zusammenhängende Teilfolge von Schritten innerhalb eines Algorithmus.
- Prozessoren führen einen Anweisungsblock nach folgenden Regeln aus:
  - Bearbeitung der Anweisungsblock beginnt mit dem ersten Schritt des Anweisungsblocks
  - zu einem Zeitpunkt wird dann nur ein Schritt ausgeführt
  - jeder Schritt wird genau einmal ausgeführt: keine Wiederholung, kein Auslassen
  - Reihenfolge der Bearbeitung der Schritte ist identisch mit der des Algorithmus
  - Bearbeitung des Anweisungsblocks endet mit dem letzten Schritt

Wiederholung: Anweisungsblock in Java

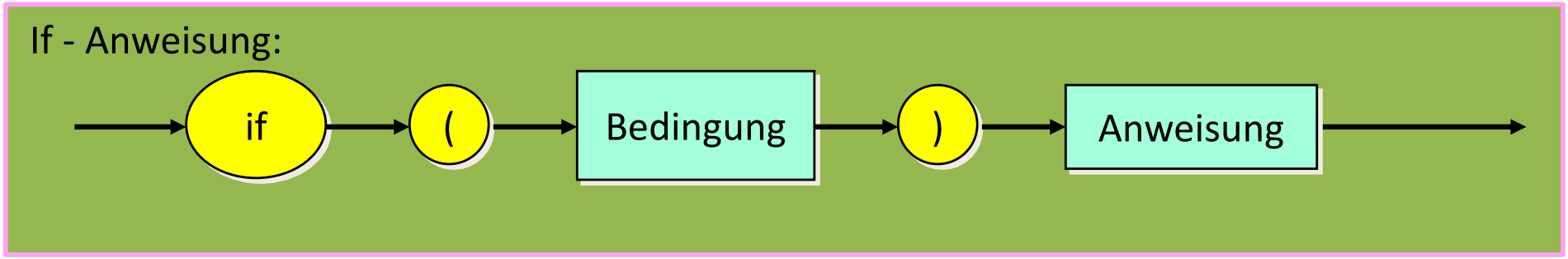
- In Java wird ein Schritt als Anweisung bezeichnet. Die Anweisungen stehen dabei in einem Paar von Klammern.



- Ein Anweisungsblock kann logisch wieder als ein einzige Anweisung aufgefasst werden!
- Die Implementierung eines Algorithmus besitzt mindestens einen Anweisungsblock, aber typischerweise gibt es mehrere Anweisungsblöcke.

2.3.2 Selektion

- Algorithmen erzeugen in Abhängigkeit von den Eingabeparametern verschiedene Prozesse.
- Bei Algorithmen benötigt man eine von einer Bedingung abhängige Selektionsmöglichkeit, mit welchen Schritten fortgefahren wird.
- In Java spricht man von der bedingten Anweisung, die mit dem Schlüsselwort `if` beginnt und folgendermaßen aufgebaut ist:



- Die Bedingung liefert entweder wahr oder falsch.
  - Zunächst wird die Bedingung ausgewertet.
  - Bei wahr wird die Anweisung ausgeführt und bei falsch übersprungen.

If-Anweisung in Methoden

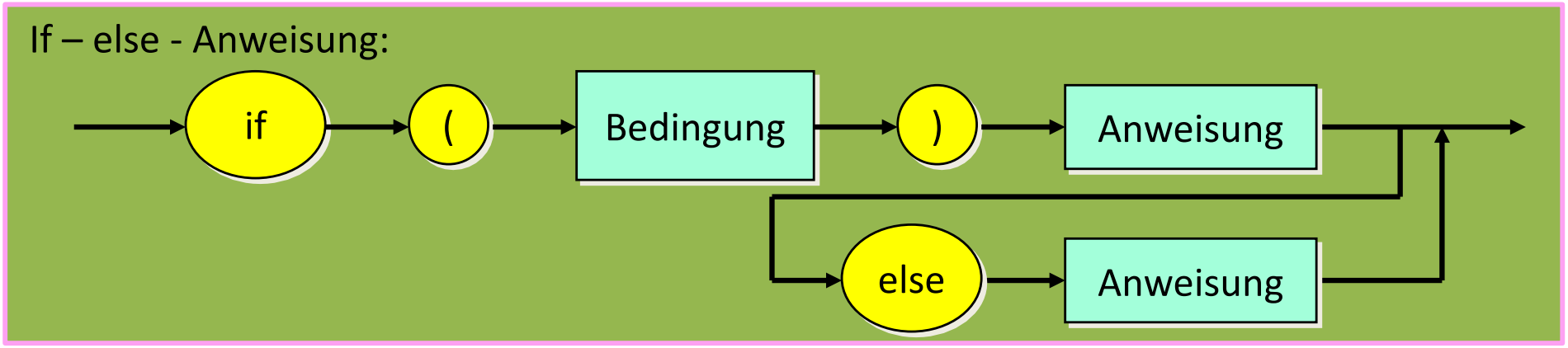
- Häufig wird eine If-Anweisung verwendet, um inkorrekte Eingabeparameter zu korrigieren.
  - Wir wollen in der Methode kugelVolumen verhindern, dass eine negative Eingabe verarbeitet wird.
- Eine Lösung

If-Anweisung in Methoden

- Häufig wird eine If-Anweisung verwendet, um inkorrekte Eingabeparameter zu korrigieren.
  - Wir wollen in der Methode kugelVolumen verhindern, dass eine negative Eingabe verarbeitet wird.
- Eine Lösung

```
double kugelVolumen (double r)  {
        final double pi = 3.14;
        if (r < 0.0)
                r = -1.0*r;
        return 1.333*pi*r*r*r;
}
```

Verallgemeinerung der Selektion



- Abarbeitung:
  - Wiederum wird zunächst die Bedingung ausgewertet.
  - Falls die Bedingung wahr ist, wird mit erster Anweisung fortgefahren.
  - Andernfalls (Bedingung falsch) wird mit zweiter Anweisung fortgefahren.
  - Nach vollständiger Abarbeitung von einer der beiden Anweisungen wird mit der nachfolgenden Anweisung (nach der if-else-Anweisung) fortgefahren.

- Eine Anweisung kann also atomar oder ein Anweisungsblock

if-else-Anweisung in Methoden

- Problem
  - Wir wollen in der Methode kreisFlaeche verhindern, dass eine negative Eingabe verarbeitet wird und der aufrufenden Methode mitteilen, dass die Eingabe nicht richtig war.
- Eine Lösung

```
double kreisFlaeche (double r)  {
        final double pi = 3.14;
        if (r >= 0.0)
                return pi*r*r;
        else
                return -1.0;
}
```

Später werden wir noch eine bessere Lösung sehen.

Verschachtelte if-Anweisungen

- Seien B1 und B2 Bedingungen und A1 und A2 Anweisungen. Wie ist folgende Anweisung auszuwerten?

```
if(B1) if(B2) A1  else A2
```

  - Welches if und else gehören zusammen?

1
```
if(B1)  if(B2) A1 else A2
```

2
```
if(B1)  if(B2) A1   else A2
```

?

Verschachtelte if-Anweisungen

- Seien B1 und B2 Bedingungen und A1 und A2 Anweisungen. Wie ist folgende A[illegible]erten?

```
if(B1) if(B2) A1
```

A2 wird ausgeführt, wenn B1 erfüllt und B2 nicht erfüllt ist.

  - Welches if und else gehören zusam[illegible]

1
```
if(B1)  if(B2) A1 else A2
```

2
```
if(B1)  if(B2) A1   else A2
```

?

A2 wird ausgeführt, wenn B1 nicht erfüllt ist.

Live Vote

PIN: KC3W

https://ilias.uni-marburg.de/vote/KC3W

Verschachtelte if-Anweisungen

- Seien B1 und B2 Bedingungen und A1 und A2 Anweisungen. Wie ist folgende Anweisung auszuwerten?

```
if(B1) if(B2) A1  else A2
```

  - Welches if und else gehören zusammen?

1
```
if(B1)  if(B2) A1 else A2
```

2
```
if(B1)  if(B2) A1   else A2
```

?

- Auflösung
  - Wie auch bei anderen Programmiersprachen üblich, verwendet man hier die Regel: Ein else ergänzt das letzte unergänzte if. (Antwort 1)

Klammerung von if-Anweisungen

- Es soll ein Algorithmus geschrieben werden, um die Anzahl der Tage zu berechnen. Die Regeln hierfür sind:
  - Jedes durch 4 teilbare Jahr ist ein Schaltjahr mit Ausnahme eines Jahrs, das durch 100, aber nicht durch 400 teilbar ist.

Zeilen-nummern

```
1   if (jahr % 4 != 0)  // Jahre nicht durch 4 teilbar
2       tage = 365;
3   else // durch 4 teilbar
4       if (jahr % 100 != 0) // nicht durch 100 teilbar
5           tage = 366;
6       else // durch 100 teilbar
7           if (jahr % 400 != 0)  // nicht durch 400 teilbar
8               tage = 365;
9           else
10              tage = 366; // durch 400 teilbar
11   return tage;
```

- *Wie könnte zu dem Codeschnipsel eine komplette Methode aussehen?*

Live Vote

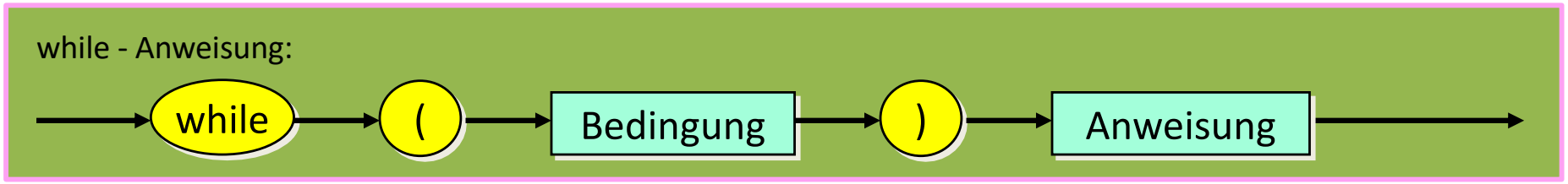
Klammerung von if-Anweisungen

```
1   if (jahr % 4 != 0)  // Jahre nicht durch 4 teilbar
2       tage = 365;
3   else // durch 4 teilbar
4       if (jahr % 100 != 0) // nicht durch 100 teilbar
5           tage = 366;
6       else // durch 100 teilbar
7           if (jahr % 400 != 0)  // nicht durch 400 teilbar
8               tage = 365;
9           else
10              tage = 366; // durch 400 teilbar
11  return tage;
```

- Weitere Fragen
  - *Welche Zeilen werden bei folgender Eingabe ausgeführt?* - 2048, 2015, 2800, 2200
  - *Die Zuweisung in welcher Zeile wird jeweils bei der Eingabe für "jahr" ausgeführt?* – 2800, 2200
  - Welche Eingabe für "jahr" führt dazu, dass die Zuweisung in Zeile 5 ausgeführt wird? Welche führ zu der Ausführung von Zeile 10?

2.3.3 while-Schleife

- Bei vielen Problemen werden Schritte solange wiederholt wie eine Bedingung wahr ist.
  - In fast allen Programmiersprachen wird deshalb als Kontrollstruktur die while-Schleife angeboten. In Java lautet die genaue Syntax:



- Verhalten der while-Schleife
  - Die Bedingung wird ausgewertet. Wenn das Ergebnis wahr ist, wird die Anweisung abgearbeitet und die Bedingung erneut ausgewertet. Wenn das Ergebnis wahr ist, wird die Anweisung abgearbeitet und die Bedingung erneut ausgewertet usw….
  - Wenn die Bedingung zum ersten Mal falsch ist, wird die Anweisung **nicht ausgeführt** und die **while-Schleife beendet**.
- Problem:
  - Wenn die Schleifenbedingung nie falsch liefert ➔ **keine Terminierung**.

Beispiele: Sternenmuster

- Schreibe eine Methode, um in einer Zeile beginnend von der ersten Position k Sterne auszugeben.

```
/** Die Methode printStars gibt Sterne in einer Zeile aus.
  * @param k Anzahl der Sterne
  */
void printStars(int k) {
        int i = 0;
        while (i < k) { // Blockanweisung
                System.out.print("*");
                i = i + 1;
        }
}
```

Beispiele: Sternenmuster

- Methode um n Zeilen mit Sternen auszugeben, n > 0.
  - Jede Ausgabe in Zeile i beginnt an der ersten Position und gibt i Sterne aus.

```
/** Die Methode printStarRows gibt Zeilen mit abnehmender
  * Anzahl an Sternen aus.
  * @param k Anzahl der Sterne
  */
void printStarRows(int k) {
       while (k > 0) { // Blockanweisung
              printStars(k);
              System.out.println();
              k = k - 1;
       }
}
```

Beispiel: Bisektionsverfahren

- Problem
  - Gegeben ist eine stetige reelle Funktion f: R → R, zu der wir nicht analytisch die Nullstellen berechnen können.
  - Wir nutzen ein Näherungsverfahren, bei dem ein reellwertiges x gesucht ist, so dass

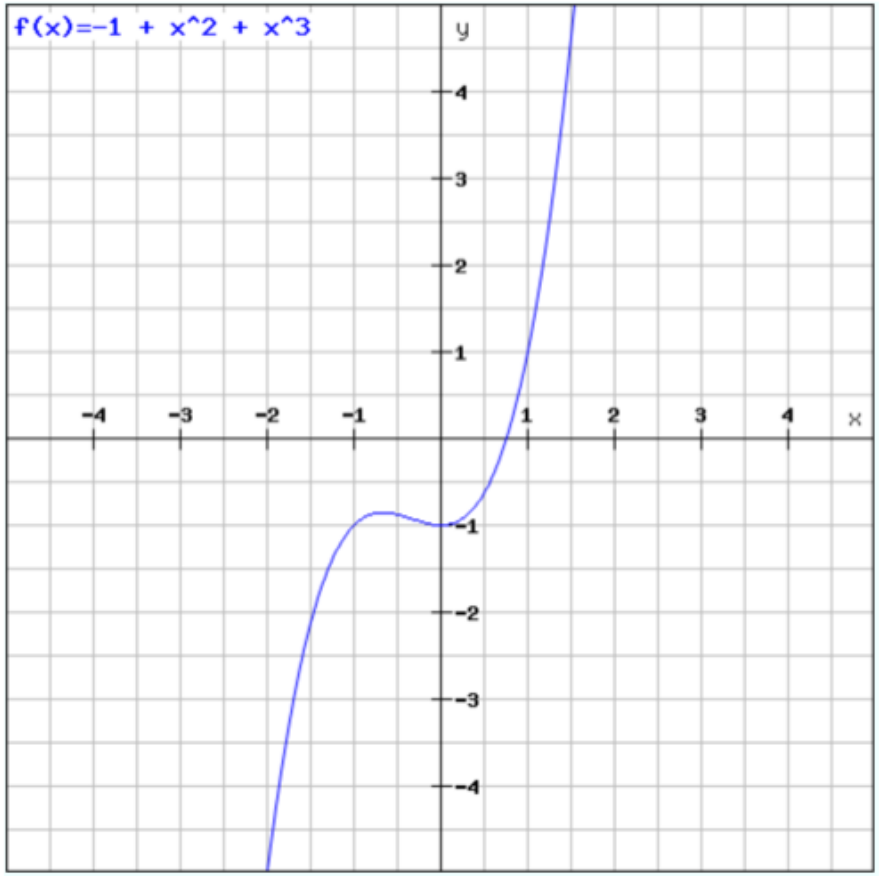
$$| f(x) | < \varepsilon$$

    ist.
  - Wenn wir zwei Punkte a, b kennen mit a < b und f(a)*f(b) < 0 lässt sich ein sogenanntes Bisektionsverfahren anwenden.

- Die Lösung gibt es dann Live in der Vorlesung.

Beispielfunktion

- Gegeben die Funktion $f(x) = -1 + x^2 + x^3$
- Wir sehen: die Nullstelle liegt zwischen 0 und 1



Beispiel: Berechnung von π

- Die Kreiszahl kann näherungsweise über die von dem Mathematiker Gottfried Wilhelm Leibniz entwickelten Summenformel berechnet werden.

$$\frac{\pi}{4} = \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k+1)} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} \mp \cdots$$

- Entwicklung einer Methode für die Summenformel
  - Iterative Berechnung mit einer while-Schleife
  - Abbruch der Schleife nach dem i-ten Schritt, falls die Änderungen der partiellen Summe des i-ten Schritts im Vergleich zum (i-1)-ten Schritt unter einer Schranke ε liegt.
    - ε ist dabei ein Eingabeparameter der Methode
  - Die Methode liefert dann den Wert der partiellen Summe nach i Durchläufen.

Java-Methode

```
/**
 *  Der Algorithmus berechnet approximativ die Kreiszahl pi
 *  @param epsilon Abbruch der Iteration, wenn | Summand | < epsilon
 *  @return Approximativer Wert für die Zahl pi
 */
double getPi(double epsilon) {
        double sum = 1.0;                   // Erste partielle Summe
        double sign = -1.0;                 // Vorzeichen
        double nenner = 3.0;                // Divisor
        while (1.0/nenner > epsilon) {      // Abbruchkriterium
                sum = sum + sign/nenner;    // Nächste partielle Summe
                nenner = nenner + 2;        // Erhöhung des Divisors
                sign = -1.0*sign;           // Vorzeichenwechsel
        }
        return 4*sum;                       // Ergebnis
}
```

Beispiel: Euklidscher Algorithmus

- Gegeben
  - Zwei ganze Zahlen m und n mit m > 0 und n > 0.
- Berechnung des größten gemeinsamen Teilers z
  - z ist Teiler von m und z ist Teiler von n
  - z ist der größte gemeinsame Teiler

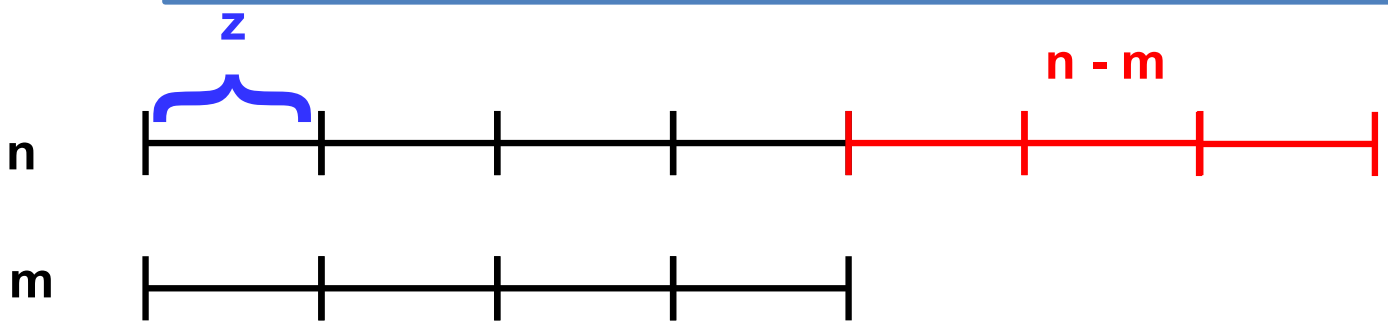
Beispiel: Euklidscher Algorithmus

- Idee des Algorithmus
  - Annahme: n > m, z sei ggt von n und m

Mathematische Überlegungen

Man kann sich überlegen:
1. z teilt auch n-m
2. es gibt keinen größeren Teiler für n-m und m

Zudem gilt noch eine Eigenschaft:
- Das Problem für n-m und m ist einfacher!
- Im besten Fall gilt sogar: n-m == m ➔ Dann sind wir fertig!



Algorithmus in Java

```
/**
 *  Der Algorithmus berechnet den ggt von zwei ganzen Zahlen > 0
 *  @param n erste ganze Zahl, n > 0
 *  @param m zweite ganze Zahl. m > 0
 *  @return größter gemeinsamer Teiler von n und m
 */
int ggt(int n, int m) {
        while (n != m)
                if (n > m)                      // Hier gilt n > m
                        n = n - m;
                else                            // Hier gilt m > n
                        m = m - n;
        // Nach der Schleife gilt n == m
        return n;
}
```

Zusammenfassung

- Wichtige Konzepte der imperativen Programmierung
  - Datentypen: Wertemenge und Operationen
  - Variablen
    - Zuweisungsoperator
  - Algorithmen und Methoden
    - Funktionen: Methoden mit echtem Datentyp als Rückgabe
    - Prozeduren: Methode mit dem Schlüsselwort void.
    - Parameterliste
  - Kontrollstrukturen
    - Anweisungsblock: {…}
    - Bedingte Anweisung: if (B) … else …
    - While-Schleife: while (B) …