

Schleifen Teil 3

Noch mehr Schleifen?

- Die bisherigen Schleifenarten (**for** und **while**) sind **kopfgesteuert**
 - Im Schleifenkopf wird geprüft, ob der Schleifenrumpf nochmals ausgeführt werden soll
 - Konsequenz: Unter Umständen wird der Schleifenrumpf gar nicht durchlaufen
- Manchmal sind **fußgesteuerte** Schleifen nützlich
 - Im Schleifenfuß wird geprüft, ob der Schleifenrumpf nochmals ausgeführt werden soll
 - Konsequenz: Der Schleifenrumpf wird in jedem Fall ein Mal ausgeführt

Die do-while-Schleife

Beispiel:

```
double d = 5.21;  
do {  
    // Anweisungen  
    // ...  
} while ( d > 0.7 );
```

do Schlüsselwort, das den Beginn einer do-while-Schleife anzeigt.

{ bzw. } Beginn und Ende des Schleifenrumpfs, also der Anweisungen, die wiederholt werden sollen.

while (d > 0.7) Schleifenfuß mit Bedingung. Die Prüfung erfolgt nach jedem Durchlauf des Schleifenrumpfs.

Weitere Algorithmen - Heronverfahren

Wurzelberechnung nach Heron

- Die Quadratwurzel einer Zahl a kann näherungsweise bestimmt werden durch

$$x_{n+1} = \frac{1}{2} \cdot \left(x_n + \frac{a}{x_n} \right)$$

mit $x_0 = a$

Wurzelberechnung - Beispiel

- Gesucht wird die Wurzel von $a = 2$ mittels $x_{n+1} = \frac{1}{2} \cdot \left(x_n + \frac{a}{x_n} \right)$

$$x_0 = 2$$

$$x_1 = \frac{1}{2} \cdot \left(2 + \frac{2}{2} \right) = 1,5$$

$$x_2 = \frac{1}{2} \cdot \left(1,5 + \frac{2}{1,5} \right) = 1,41\bar{6}$$

$$x_3 = \frac{1}{2} \cdot \left(1,41\bar{6} + \frac{2}{1,41\bar{6}} \right) \approx 1,41422$$

...

- Aufhören, wenn $|x_{n+1} - x_n| < \varepsilon$ für ein fest gewähltes ε

Aufgabe 1

- Entwirf einen Algorithmus in Pseudocode, der mittels $x_{n+1} = \frac{1}{2} \cdot \left(x_n + \frac{a}{x_n} \right)$ einen Näherungswert für die Quadratwurzel einer beliebigen Zahl berechnet.
 - Die Berechnung soll enden, sobald $|x_{n+1} - x_n| < 0.0001$
 - Die einzelnen Folgenglieder x_n müssen **nicht** alle gespeichert werden!
- Stelle den Algorithmus mit einem Struktogramm dar.

Aufgabe 2

- Implementiere den Algorithmus in einer Methode `float berechneWurzel(float r)` in Java.
 - Möglicherweise steht die do-while-Schleife nicht zufällig auf diesen Folien ;-)
 - Teste die Methode für verschiedene Zahlen (einschließlich negativen).

Weitere Algorithmen - Sieb des Eratosthenes

Primzahlssuche mit dem Sieb des Eratosthenes

- Verfahren, um alle Primzahlen bis zu einer bestimmten Obergrenze n zu finden
- Vorgehen:
 1. Schreibe alle Zahlen von 2 bis n auf
 2. Wähle die kleinste “unbearbeitete” und nicht gestrichene Zahl:
 - Streiche alle Vielfachen dieser Zahl
 3. Falls die nächstgrößere nicht gestrichene Zahl höchstens so groß ist wie \sqrt{n} , wiederhole Schritt 2, ansonsten stoppe
 4. Alle nicht gestrichenen Zahlen sind Primzahlen

Primzahlssuche - Beispiel (1)

- Wir suchen alle Primzahlen bis einschließlich 36

	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

Primzahlssuche - Beispiel (2)

- Vielfache von 2 streichen

	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

Primzahlssuche - Beispiel (3)

- Vielfache von 3 streichen

	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

Primzahlssuche - Beispiel (4)

- Vielfache von 5 streichen

	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

Primzahlssuche - Beispiel (5)

- Die nächste unmarkierte Zahl ist die 7, aber da $7 > \sqrt{36} = 6$, ist der Algorithmus beendet.
 - Alle nicht gestrichenen Zahlen sind Primzahlen!

	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

Primzahlssuche - Beispiel (5)

- Die nächste unmarkierte Zahl ist die 7, aber da $7 > \sqrt{36} = 6$, ist der Algorithmus beendet.
 - Alle nicht gestrichenen Zahlen sind Primzahlen!

	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

- Optimierung: Bei der Suche nach zu streichenden Vielfachen der aktuell kleinsten Zahl p kann man bei p^2 beginnen, da alle kleineren Vielfachen (also $2p, 3p, \dots, (p-1) \cdot p$) bereits gestrichen wurden.

Aufgabe 3

- Entwirf einen Algorithmus in Pseudocode, der mit dem Sieb des Eratosthenes alle Primzahlen bis zu einer frei wählbaren Obergrenze findet.
 - Orientiere Dich an der verbalen Beschreibung auf der entsprechenden Folie (klick mich).

Aufgabe 4

- Schreibe eine Java-Methode **findePrimzahlen**(int max), die alle Primzahlen kleiner oder gleich max findet und ausgibt.
 - Orientiere Dich an Deinem Pseudocode von Aufgabe 3 und “verfeinere” die einzelnen Operationen zu funktionierendem Java-Code.
- Zusatzaufgabe: Visualisiere das Verfahren.
 - Stelle die Zahlen von 2 bis max in einem rechteckigen Raster dar.
 - Markiere die gestrichenen Zahlen schrittweise farbig.

Punkte im Raum

- Punkte im Raum verfügen zusätzlich zur x- und y-Koordinate über eine z-Koordinate, die angibt, wie weit der Punkt “nach hinten” verschoben ist.
- Um aus diesen sogenannten Weltkoordinaten die passenden Bildschirmkoordinaten zu berechnen, kann man einfach die x- bzw. y-Koordinate durch die z-Koordinate teilen und die Koordinaten der Bildschirmmitte dazu addieren:

```
bildX = weltX / weltZ + mitteX;  
bildY = weltY / weltZ + mitteY;
```

- Diese Umrechnung geht davon aus, dass der Ursprung des Weltkoordinatensystem in der Mitte der Bildebene liegt und die positive Z-Achse in den Bildschirm hinein zeigt

Aufgabe 5

- Schreibe ein Programm, das ein “3D-Sternenfeld” zeichnet, also Punkte/kleine Kreise, die sich auf den Betrachter zubewegen
 - Verwende Arrays, um die Weltkoordinaten der einzelnen Punkte zu speichern.
 - Verwende die Methode `void draw()`, um Code wiederholt auszuführen und mit jedem Durchlauf die z-Koordinate aller Punkte zu verringern.