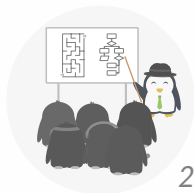


Diese Worte sucht getau ich habe

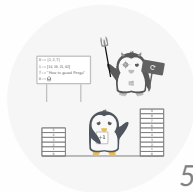
Tutorium Zwei



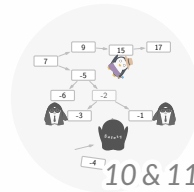
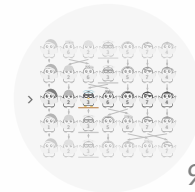
Theorie



Grundlagen



Vertiefungen



> Algorithmenkonstruktion



> Kerneigenschaften eines Algorithmus:

- Von einem Prozessor (Mensch, ...) in endlich vielen Schritten Ausführ- und Reproduzierbar
- Eine endliche Beschreibung in Elementaroperationen

> Abstrakte Variablenoperationen:

`long x;` *Deklaration* — Reservieren von Namen mit Typ long
`x = 21;` *Initialisierung* — Erste Wertzuweisung, macht die Variable verwendbar
`x = 42;` *Wertzuweisung* — Überschreiben/Ändern des vorherigen Wertes

Exkurs: Kommandozeilenparameter

M-Array me und andere Zukunftsträume

- › Arrays werden später genauer behandelt
- › Arrays sind Listen fester Größe die Elemente des gleichen Datentyps beinhalten
- › Wir notieren sie mit []:

```
int[] arr = {42, -7, 13};
```

Ein Array an Integern

```
System.out.println(arr[1]); // → -7
```

Beginnt bei 0

```
System.out.println(arr.length); // → 3
```

Liefert die Array-Größe

```
System.out.println(arr[arr.length - 1]); // → 13
```

Die Sache mit dem `String[] args`

```
public class Example {  
    public static void main(String[] args) {  
        System.out.println(args.length);  
    }  
}
```

- › Hier haben wir ein Array von Strings
- › Dieses wird von der Java Virtual Machine gefüllt
- › Beim Programmstart mit `java Example`, können wir Argumente übergeben

Ein wunderbares Beispiel

java Example Hermeline mögen Mäuse

```
public class Example {  
    public static void main(String[] args) {  
        3  
        for(int i = 0; i < args.length; i++) {  
            System.out.println(args[i]);  
        }  
    }  
}
```

args = {"Hermeline", "mögen", "Mäuse"}

i = 0 → Hermeline
i = 1 → mögen
i = 2 → Mäuse

Fürs Erste werden wir es dabei belassen.





Zahlen als Parameter
werden leider auch
zu `String`.



Wir machen
einen eigenen
Zahlenkonverter.

Leider nein. Wir greifen „nur“ vor...



int? Wie primitiv!

- › Für jeden primitiven Datentyp existiert eine „Wrapper-Klasse“ **KLASSE??**
- › Wie `System`, welches uns `out.println` liefert, gibt es `Integer` für `int`
 - `Integer.parseInt(...)`, `Double.parseDouble(...)`, ... konvertieren `"42"` zu `42`
 - Genauer liefert `Integer.parseInt("42")` die Zahl `42` zurück
 - Wir gehen hier zunächst nur von gültigen Eingaben (Zahlen) aus
- › Nun sind wir gewappnet für die Präsenzaufgabe!

Es gibt keine if-Schleifen, sondern nur if-Abfragen!
– if-schleife.de

Präsenzaufgabe

1

Wenn... Ja wenn nur die Schleife nicht wär...

Legen Sie eine Java Datei namens `ErsteSchleife.java` an (oder bearbeiten Sie die Aufgabe auf einem Blatt Papier). Lesen Sie in der main Methode eine `int` Variable namens `n` über die Kommandozeilenparameter ein und implementieren Sie eine for Schleife, die jede dritte Zahl von 1 bis einschließlich `n` ausgibt. Beispiel:

`n = 13 // → 1 4 7 10 13`

Ein Schleifenzähler

- › Mit dem Vorwissen ist es an sich nur Einsetzen:

```
public class ErsteSchleife {  
    public static void main(String[] args) {  
        int n = Integer.parseInt(args[0]);  
        for(int i = 1; i <= n; i = i + 3) {  
            System.out.println(i);  
        }  
    }  
}
```

Was, wenn es keine
Argumente gibt? Hier
hilft eine if-Abfrage.

Das Beispiel der Aufgabe hatte keine neuen Zeilen!
Dafür gibt es „print“ anstelle von „print line“.

ErsteSchleife.java

In the Mean-While

- › Wir können dies aber auch mit einer **while**-Schleife machen (links)

```
int i = 1;
while(i <= n) {
    System.out.print(i + " ");
    i += 3;
}
```

```
int i = 1;
if(n > 0) { Soll die Schleife überhaupt
    do { einmal laufen?
        System.out.print(i + " ");
        i += 3;
    } while (i <= n); Das ; nicht vergessen!
}
```

- › Oder mit einer **do-while**-Schleife (rechts)
- › „**java** ErsteSchleife 12“ liefert die Ausgabe „1 4 7 10“

ErsteSchleifeWhile.java, ErsteSchleifeDoWhile.java

- › Anstelle von $i \leftarrow i + 3$ können wir beispielsweise auch Modulo-Rechnen:

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

- › Es gibt eigentlich immer etliche Möglichkeiten...

Übungsblatt 2

Aufgabe 1: Algorithmen und Unteralgorithmen

Oft ist es zur besseren Strukturierung notwendig komplexere Algorithmen aufzuteilen. Dies ist beispielsweise der Fall, wenn Code-Elemente häufig an verschiedenen Stellen eingesetzt werden oder wenn es der Übersichtlichkeit dienen soll. Im Folgenden sollen Sie einen einfachen Sortieralgorithmus in zwei Teilalgorithmen umsetzen. Zusätzlich sollen Sie lernen die Eingaben (sog. Parameter oder Argumente) des Aufrufs auf Gültigkeit zu überprüfen. Für zukünftige Aufgaben sollen Sie dies selbstständig erkennen und umsetzen.

Indizes		Liste				Vertauschen?
i	j	l_1	l_2	l_3	l_4	
1	2	5	8	4	0	Nein, denn $5 \not> 8$
1	3	5	8	5	0	Ja, denn $5 > 4$
1	4	4	8	5	0	Ja, denn $4 > 0$
Runde vorbei, alle $j \in \{i + 1, \dots, N\}$ betrachtet!						
2	3	0	8	5	5	Ja, denn $8 > 5$
2	4	0	5	8	4	Ja, denn $5 > 4$
Runde vorbei, alle $j \in \{i + 1, \dots, N\}$ betrachtet!						
3	4	0	4	8	5	Ja, denn $8 > 5$
Fertig, alle $i \in \{1, \dots, N - 1\}$ betrachtet!						
→		0	4	5	8	

Gehen Sie zur Sortierung nach folgendem Schema vor.

Seien l_1, \dots, l_N die Werte einer Liste L der Länge N . Für jedes $i \in \{1, \dots, N - 1\}$ betrachte der Reihe nach jedes l_j mit $j \in \{i + 1, \dots, N\}$ und vergleiche l_i mit l_j . Falls $l_i > l_j$ vertausche l_i und l_j und fahre mit dem nächsten j fort.

So sortiert der beschriebene Algorithmus die Liste $(5, 8, 4, 0)$ aufsteigend zu $(0, 4, 5, 8)$. Die beiden **Markierungen** je Zeile geben an, welche Elemente in dem jeweiligen Schritt verglichen werden.

Elemente vertauschen

- a) Entwerfen Sie einen Algorithmus, der gegeben einer Liste und zwei Indizes die entsprechenden Elemente vertauscht. Stellen Sie sicher, dass die Indizes i und j gültig sind, d.h. dass sie **größer als 1** und kleiner als die Länge der Liste sind. Stellen Sie zudem sicher, dass die Länge der Liste mindestens 2 beträgt. Der Algorithmus soll für den Fall der Ungültigkeit mit einer Fehlermeldung abbrechen.

Input : List $L = (l_1, \dots, l_N)$, Index i , and Index j

```
1 if  $N < 2$  or  $i < 1$  or  $j < 1$  or  $i > N$  or  $j > N$  then
2   print „Eingabe ungültig!“ to output device;
3   stop program;
4 make new  $t$  have value  $l_j$ ;
5 make  $l_i$  have value  $l_j$ ;
6 make  $l_j$  have value  $t$ ;
```

In der Aufgabe stand etwas von „größer als 1“, aber wie wir Listen indizieren ist nicht von uns standardisiert.



Sich den Tag schön-sortieren

- b) Entwerfen Sie einen Algorithmus, der genau nach dem angegebenen Schema eine Liste sortiert. Stellen Sie auch hier sicher, dass die Liste mindestens zwei Elemente hat. Falls Sie Teilaufgabe a) nicht bearbeitet haben, können Sie annehmen, dass der Algorithmus `vertausche(L, i, j)` zur Verfügung steht.

Input : List $L = (l_1, \dots, l_N)$

```
1 if N < 2 then
2   | print „Eingabe ungültig!“ to output device and stop program;
3 Make new i have value 1;
4 while i < N do
5   | Make new j have the value i + 1;
6   | while j ≤ N do
7     | if  $l_i > l_j$  then call vertausche(L, i, j);
8     | Increment j by 1;
9   | Increment i by 1;
```

Du kriegst's nicht raus? Better Call Gauß!

- c) Geben Sie eine worst-case Laufzeitabschätzung für Ihren Algorithmus aus Teilaufgabe b) an. Falls Sie Teilaufgabe a) nicht bearbeitet haben, können Sie annehmen, dass das Vertauschen (inkl. aller Überprüfungen) höchstens) neun elementare Operationen benötigt. Begründen Sie Ihre Antwort.

„Wir“ haben die a) bearbeitet. Also... Lets begin!

Input : List $L = (l_1, \dots, l_N)$, Index i , and Index j

```
1  if N < 2 or i < 1 or j < 1 or i > N or j > N then
2      print „Eingabe ungültig!“ to output device; 1
3      stop program; 1
4  make new t have value li; 1
5  make li have value lj; 1
6  make lj have value t; 1
```

} 2 (Nur schlechtester Pfad)

} 3

$$\rightarrow 1E + 1E + 1E + 1E + 1E + 3E = 8E$$

Input : List $L = (l_1, \dots, l_N)$

```

1  if  $N < 2$  then
2    | print „Eingabe ungültig!“ to output device and stop program;
3  Make new  $i$  have value 1;
4  while  $i < N$  do
5    | Make new  $j$  have the value  $i + 1$ ;
6    | while  $j \leq N$  do
7      | | if  $l_i > l_j$  then
8      | | | call vertausche( $L, i, j$ );
9      | | Increment  $j$  by 1;
10   | Increment  $i$  by 1;
    
```

Angenommen N sei 5, dann gilt für die Anzahl innerer Schleifendurchläufe:

i	#
1	4
2	3
3	2
4	1

Wir summieren also von 1 bis $N - 1$. Nach Gauß: $\frac{N^2 - N}{2}$
Gesamtdurchläufe.

$$\begin{aligned}
 &\rightarrow 1E + 1E + (N - 1) \cdot (2E + 1E) + \frac{N^2 - N}{2} (1E + 8E + 1E) \\
 &= -1E + 3 \cdot NE + 5 \cdot N^2 E - 5 \cdot NE \\
 &= 5 \cdot N^2 E - 2 \cdot NE - 1E \in \mathcal{O}(N^2)
 \end{aligned}$$

- › `System.out.print`, `Integer.parseInt`, ... sind keine Elementaroperation
- › Es sind implementierte Algorithmen, die eine Aufgabe erledigen
- › Wenn wir bestehende Algorithmen wiederverwenden, heißen sie „Unteralgorithmen“
 - Sie erlauben einen ersten Abstraktionsmechanismus um Teilprobleme auszulagern
 - Ziel ist dann die Komposition aus immer abstrakteren Bausteinen
- › (Unter-)Algorithmen sind ein wichtiger und elementarer Teil der Programmierung
- › Die Umsetzung in Java werden wir später betrachten.

Aufgabe 2: Datentypen

Welchen Datentyp würden Sie wählen um folgende Daten zu speichern? **Begründen Sie Ihre Antwort!**

a) *Die Email-Adresse einer Person*

Email-Adressen sind lange Zeichenketten welche auch Sonderzeichen enthalten. Daher ist `String` naheliegend.

b) *Den Akkustand eines Smartphones*

Ein Akkustand wird für gewöhnlich als Ganzzahl angegeben (→ `byte`, `short`, `int` und `long`). Der Akkustand reicht dabei meist von 0 % bis 100 % → `byte`.

c) *Den Notendurchschnitt einer Prüfung*

Die Noten einer Prüfung ist meist eine Fließkommazahl (→ `float`, `double`), deren Wertebereich klein mit geringer Auflösung an Nachkommastellen ist (z.B. 1–6 mit einer Nachkommastelle). Deswegen wählen wir hier `float`.

d) *Ein Satzzeichen*

Wir gehen hier von einem Satzzeichen aus, welches durch eine einzige UTF-16 Einheit dargestellt werden kann. In diesem Fall reicht ein `char` (sonst `String` das ist aber schon wieder Overkill).

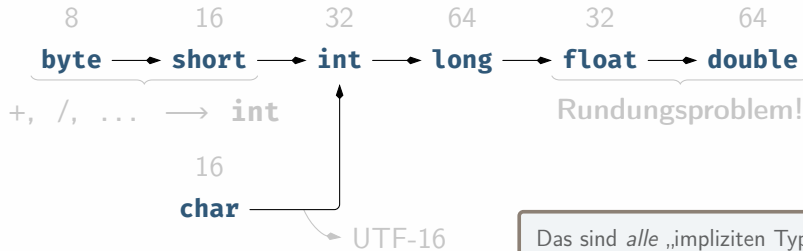
- › Eine eindeutig richtige Antwort gibt es nicht (Begründungen!).
- › Begründungen wie „weniger Speicherplatz“ sind problematisch(er)
 - Java kann für ein `byte` auch 32 bit oder mehr (64 bit, ...) reservieren
 - Analog wird `boolean` oft als `byte` umgesetzt ([JVMS17 2.3.4](#))
 - Das hat mit Geschwindigkeit, Parallelisierung, Sicherheit, ... zu tun
 - Für uns in Eidl sind die Details nicht wichtig

Ein wenig mehr über primitive Datentypen

void ist in Java kein Datentyp!
JLS17 4.2 & 14.8



boolean



Das sind *alle* „impliziten Typkonvertierungen“ in Java.
String hat da nichts zu suchen.



> **byte** fasst $[-2^7 \text{ bis } 2^7 - 1]$

Das reicht aus, um alle Primzahlen zwischen 2 000 und 3 000 zu zählen. 127 Sekunden sind $\approx 0,035$ Stunden.

> **short** fasst $[-2^{15} \text{ bis } 2^{15} - 1]$

32 767 Sekunden sind $\approx 9,1$ Stunden.

> **char** fasst $[0 \text{ bis } 2^{16} - 1]$

Das reicht für alle Code Units von UTF-16!

> **int** fasst $[-2^{31} \text{ bis } 2^{31} - 1]$

2 147 483 647 Sekunden sind $\approx 68,05$ Jahre.

> **long** fasst $[-2^{63} \text{ bis } 2^{63} - 1]$

9 223 372 036 854 775 807 Sekunden sind $\approx 292\,271\,023\,045,3$ Jahre ≈ 658 Millionen mal „älter“, als das Universum.

[JLS17 4.2.1](#), [JVMS17 2.3.1](#)

- > `float` schweift zwischen $\pm 2^{-126}$ bis zu $(2 - 2^{-23}) \cdot 2^{127}$

Joa... Das sind mehr als $5 \cdot 10^{30}$ Jahre.

- > `double` schweift zwischen $\pm 2^{-1022}$ bis zu $(2 - 2^{-52}) \cdot 2^{1023}$

So zählen sie sich leicht, die Atome des Universums!

- > Dabei sollten wir die Rundungsprobleme aber nicht außer Acht lassen
- > Jenseits des Primitiven liefert Java natürlich mehr
 - `BigInteger` begrenzt die Anzahl Bits durch einen `int`
 - `BigDecimal` erlaubt Skalieren durch `int` und volle Kontrolle übers Runden

Aufgabe 3: Boolesche Ausdrücke

Betrachten Sie den folgenden Java Code-Ausschnitt. Werten Sie die folgenden Ausdrücke in der Reihenfolge, in der sie ausgeführt werden (d.h. von oben nach unten), aus. Geben Sie dazu für jede Zeile (beginnend mit der zweiten Zeile) den neuen Wert der Variablen *b* an.

```
boolean b = true;
```

```
b = !(true || false) || b;  b ist true
```

```
b = !(true && false);      b ist true
```

```
b = (!(true && true) || (false || false));  b ist false
```

```
b = !b;  b ist true
```

a++, a-- → !a, -a, ++a, --a → a * b, a / b, a % b → a + b, a - b
→ a == b, a < b, ... → a ^ b → a && b
→ a || b → ...

Aufgabe 4: Variablen anlegen und verarbeiten

In der Vorlesung haben Sie gelernt, wie in Java Variablen deklariert und initialisiert werden. Nun sollen Sie in einem kleinen Beispiel diese Konzepte selbst umsetzen. Betrachten Sie die `.java`-Datei namens `Variablen.java` mit folgendem Inhalt:

```
class Variablen {  
    public static void main(String[] args) {  
  
    }  
}
```

- Erweitern Sie den Code so, dass zwei `int` Variablen deklariert werden. Initialisieren Sie diese **anschließend** mit einem beliebigen gültigen Wert und addieren Sie sie. Das Ergebnis dieser Addition soll in eine neue dritte `int` Variable gespeichert werden. Geben Sie das Ergebnis dann über den Aufruf `System.out.println(name)` aus, wobei Sie `name` entsprechend ersetzen.
- Geben Sie als Kommentar im Java Code aus Teilaufgabe a) jeweils an, ob es sich bei dieser Zeile um eine Deklaration, Initialisierung oder Zuweisung handelt.

Simple Adds and Stuff

- a) Erweitern Sie den Code (Deklarieren, Initialisieren, Addieren und Ausgeben)
- b) Geben Sie an, wo Deklarationen, Initialisierungen und Zuweisungen stattfinden.

```
class Variablen {  
    public static void main(String[] args) {  
        int zahl1, zahl2, ergebnis; Alle drei Variablen werden deklariert  
        zahl1 = 40; Eine Initialisierung mit Wert 40  
        zahl2 = 2; Eine Initialisierung mit Wert 2  
        ergebnis = zahl1 + zahl2; Eine Initialisierung mit dem Ergebnis der Summe  
        System.out.println(ergebnis); Eine Ausgabe (für die b irrelevant)  
    }  
}
```

Ist es wichtig, dass die letzte Initialisierung mit einer Summe ist? Nö. Es ist einfach nur der erste Wert, den Ergebnis erhält.



Aussicht: Übungsblatt 3

- › Wenn wir Java-Code möchten, steht das da (ab jetzt wird das der Standardfall sein).
- › In Aufgabe 1 suchen wir *einen* booleschen Ausdruck der das Programm abbildet.
Vergleiche hierzu die Präsenzaufgabe von letzter Woche.
- › Für die Schleifen der 2. Aufgabe, kann man sich die Alternativen dieser Präsenzaufgabe nochmal ansehen.
- › Allgemein gilt: Wenn Pseudocode oder ein Ansatz gegeben ist, soll der Algorithmus auch diesen Verwenden oder darauf aufbauen!

Aufgabe 3: Algorithmen in Java implementieren

- › Eine abstrakte Notation wird konkret.
- › Viele einzelne Punkte werden nun kollektiv Wichtig
 - Einhalten der Java-Syntax
 - Welcher Datentyp passt am besten?
 - Welche Art von Schleife, Fallunterscheidung, ... eignet sich?
 - Was drückt der Pseudocode implizit aus, ist aber in Java explizit notwendig?
 - Und umgekehrt: was müssen wir in Java nicht schreiben?
 - ...
- › Kommentare empfehlen sich da, wo der Gedanke des Algorithmus hinter Java-Gebrabbel verschwindet. (If you are interested, consider looking up literate programming [Knu84])

Aufgabe 4: Iterative Wurzelberechnung

- › Für die Fließkommazahl gibt es `Float.parseFloat` oder `Double.parseDouble`
- › Verwendet das Heron-Verfahren wie angegeben (keine alternative Wurzelberechnung)
- › Ihr dürft gerne eure Entscheidungen in den Kommentaren begründen.
- › An welcher Stelle kommen Unteralgorithmen zum Einsatz?
Und...Wo könnten sie noch hilfreich sein?

Der zeitliche Abstand [...] läßt den wahren Sinn, der in einer Sache liegt, erst voll herauskommen. Die Ausschöpfung des wahren Sinnes aber, der in einem Text oder einer künstlerischen Schöpfung gelegen ist, kommt nicht irgendwo zum Abschluß, sondern ist in Wahrheit ein unendlicher Prozeß.
— Hans-Georg Gadamer [Gad60, unverändert, p. 282]

Abschließendes

[Gad60] Hans-Georg Gadamer. „Wahrheit und Methode“. 1960

[Knu84] Donald Ervin Knuth. „Literate programming“. 1984

- › Kommandozeilenparameter landen im `args`-Array (als `String`)
- › `Integer.parseInt("13")` liefert die vom `String` beschriebene Ganzzahl (`13`)
- › Unteralgorithmen sind ein elementarer Programmiermechanismus
- › Javas implizite Typkonvertierungen existieren nur für primitive Datentypen
 - `byte` → `short` → `int` → `long` → `double` und `char` → `int`
 - „Zahlen von klein nach groß, `char` zu `int`“

