Informatik Kursstufe 4-stündig

Schuljahr 18/19

Organisation: 10.9.18

- kein Heft aber Ordner für Arbeitsblätter
- Klausuren 2 pro HJ
- Notenverhältnis

  50% schriftlich
  20% Projekt
  30% mündlich

  70% schriftlich falls kein Projekt

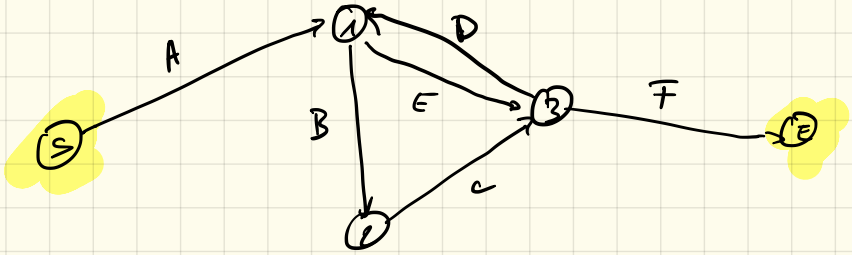
- schule@lehrer-kimmig.de
- wiki.lehrer-kimmig.de
- ab.lehrer-kimmig.de
- GFS mind. 15-20 min, gerne länger + Handout

Inhalte:

- Projektplanung, Top-down vs. Bottom-up, UML
- Programmierung
  - OOP, abstrakten Datentypen
  - Algorithmen, Sortierverfahren, Rekursion
  - Laufzeiten, Berechenbarkeit
  - Codierung, Komprimierung
- Netzwerk
  - Schichtenmodell
  - Spuren im Netz
  - Kommunikation
  - Angriffe und Schutz
- DB
  - SQL
- Rechnerarchitektur
  - Technik, logische Schaltungen
- Automatentheorie

- Kryptografie und Signaturen

- Sicherheit



A E F

A B C F
A B C D E F
A B C D B C F
A E D E F

I. Algorithmen

12.9.18

Definition Algorithmus:

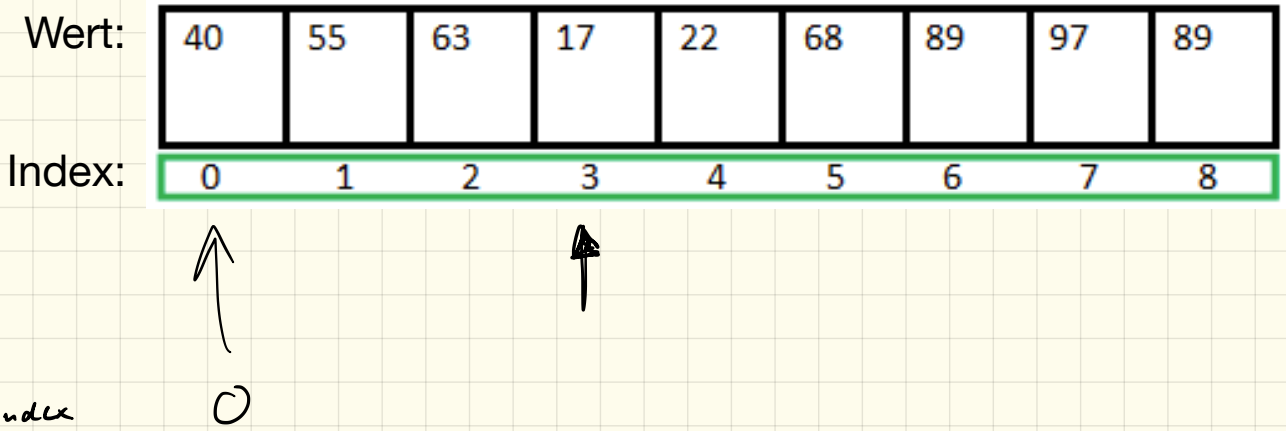
Ein Algorithmus ist eine eindeutige Handlungsvorschrift zur Lösung eines Problems oder einer Klasse von Problemen. Algorithmen bestehen aus endlich vielen, wohldefinierten Einzelschritten. Damit können sie zur Ausführung in ein Computerprogramm implementiert, aber auch in menschlicher Sprache formuliert werden. Bei der Problemlösung wird eine bestimmte Eingabe in eine bestimmte Ausgabe überführt.

(Quelle: Wikipedia „Algorithmus“, 11.9.18)

Beispiele: Anleitung, Kochrezept, ...

I.1 Minimumsuche in einem Array

Ziel ist es, innerhalb eines Arrays das kleinste Element zu finden



Minimumsuche

1. Pseudocode

Schreibe in *Pseudocode* (d. h. in „freier Sprache“), wie die Minumumsuche funktioniert, also wie in einem gegebenen, unsortierten Array, der kleinste Wert herausgefunden werden kann:

Einführung Zufallszahlen

Es bietet sich an, nicht mit einem vordefinierten Array zu arbeiten, sondern die Zahlen im Array *zufällig* zu erzeugen.
Java bietet hierfür die Möglichkeit Zufallszahlen[1] zu erzeugen.
Für eine bessere Übersichtlichkeit rechnen wir hier vorerst nur mit ganzen Zahlen. Für ganzzahlige Zufallszahlen müssen wir zunächst das Paket `java.util.Random` importieren. Anschließend können wir in unserem Programmablauf einen Zufallszahlengenerator erzeugen:

[1] Anmerkung: es können mit einem herkömmlichen Computer keine *echten Zufallszahlen* erzeugt werden, diese werden mit einem aufwändigen Algorithmus *berechnet*. Je besser dieser Algorithmus ist, desto zufälliger sehen die Zahlen aus. Wir sprechen deshalb auch von *Pseudozufallszahlen*.

```java
import java.util.Random;

class Zufallszahl {
    public static void main(String[] args) {
        // erzeuge Zufallszahlengenerator
        Random rand = new Random();

        // erzeuge Zufallszahl zwischen >=0 und < 50
        rand.nextInt(50);
    }
}
```

Listing 1: Erzeugung von Zufallszahlen

Der Parameter der `rand.nextInt`-Methode gibt dabei die obere Grenze der Zufallszahlen an. Im obigen Beispiel liegen die erzeugten Zufallszahlen also immer zwischen *inklusive* 0 und *exklusive* 50.

2. Zufallszahlengenerator

Erzeuge ein neues Java-Projekt `Sortierung`. An diesem Projekt werden wir die nächsten Wochen arbeiten.
Lege darin ein Paket `minimumsuche` mit einer Klasse `Minimum` (inklusive `main`-Methode) an.
Programmiere hier zunächst einen Zufallszahlengenerator:

a) Erzeuge zunächst ein Array, welches 20 Ganzzahlen speichern kann.

b) Befülle dieses Array mit 20 zufälligen Zahlen (zwischen 0 und 50).

c) Lasse die Werte dieses Arrays auf der Konsole kommagetrennt ausgeben.
Beispiel: `20,6,30,34,5,11,0,34,28,12,4,26,11,15,44,28,40,7,20,7`

3. Minimumsuche

Erweitere den Programmablauf aus Aufgabe 2 so, dass im zufällig befüllten Array der Minimale Wert und der zugehörige Index gesucht und ausgegeben wird. Verwende dazu den Ablauf aus Aufgabe 1.
Beispielausgabe: `Index: 6, Wert: 0`

4. Zusatzaufgabe: Minimumsuche als Methode

Um die Minimumsuche nicht jedes Mal erneut programmieren zu müssen soll nun eine passende Methode programmiert werden. Das zu durchsuchende Array soll dabei als *Parameter* an die Methode übergeben werden.
Überlege dir zunächst, was als Ergebnis der Methode zurückgegeben werden soll und programmiere anschließend diese Methode in die `Minimum`-Klasse.

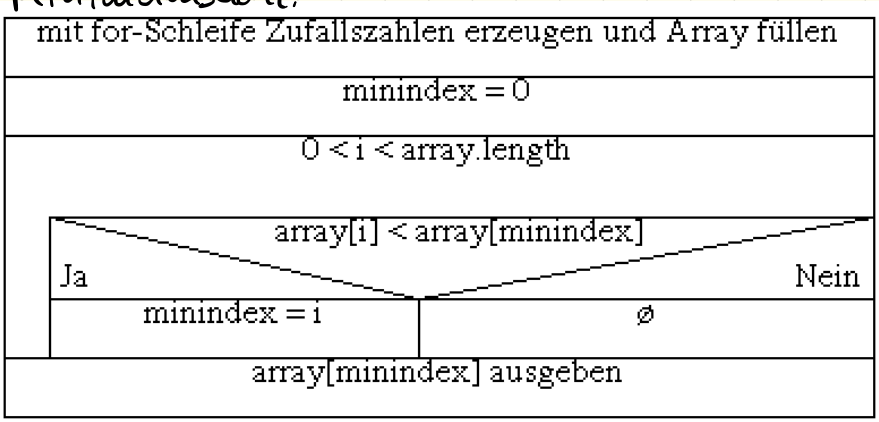
5. Zusatzaufgabe 2: Sortierte Ausgabe

Überlege dir, wie man diese Minimumssuche dazu verwenden könnte, alle Einträge des Arrays sortiert auf der Konsole auszugeben.

I.2 Struktogramme

Um Algorithmen (oder Ausschnitte daraus) darzustellen, verwendet man sogenannte Struktogramme:

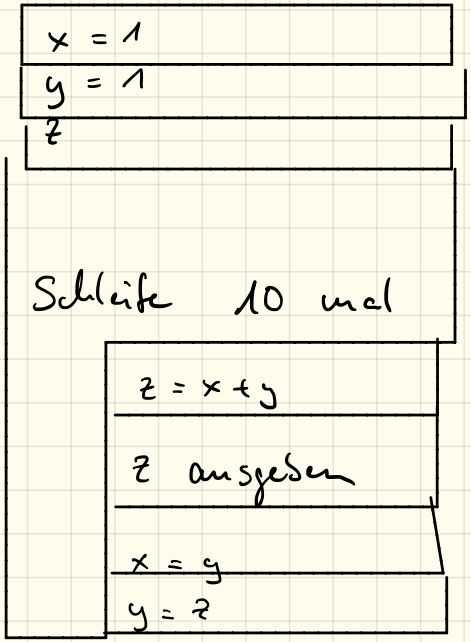
Minimumsuche

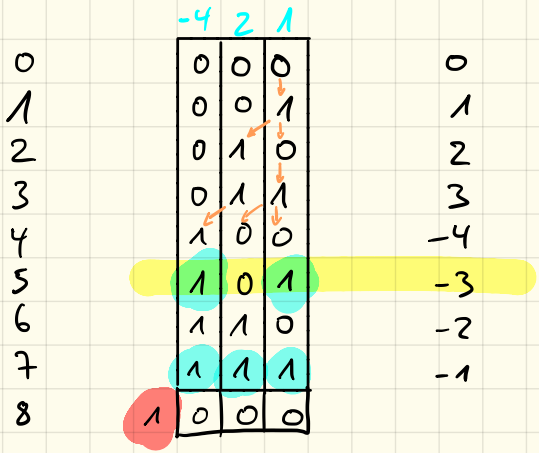


Kostenloser Struktogrammeditor unter http://www.whiledo.de/ downloadbar!

Aufgabe (gemeinsam):

Wie würde das Struktogramm aussehen, um die Fibonaccifolge (1,1,2,3,5,8,13,…) zu berechnen und ausgeben zu lassen?




-4 2 1
0 0 0 0
1 0 0 1
2 0 1 0
3 0 1 1
4 1 0 0
5 1 0 1
6 1 1 0
7 1 1 1
8 1 0 0 0
0
1
2
3
-4
-3
-2
-1

Aufgabe: Collatz-Folge

* Die Collatz-Folge beginnt bei einer beliebigen Zahl n>0
* ist n gerade, so ist die nächstes n=n/2
* ist n ungerade, so nimm als nächstes n=3n+1
* Wiederhole, bis das n=1 ist

Beispiel: 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

Wie sieht das Struktogramm für diesen Algorithmus aus?

13, 40, 20, 10, 5, 16, 8, 4, 2, 1

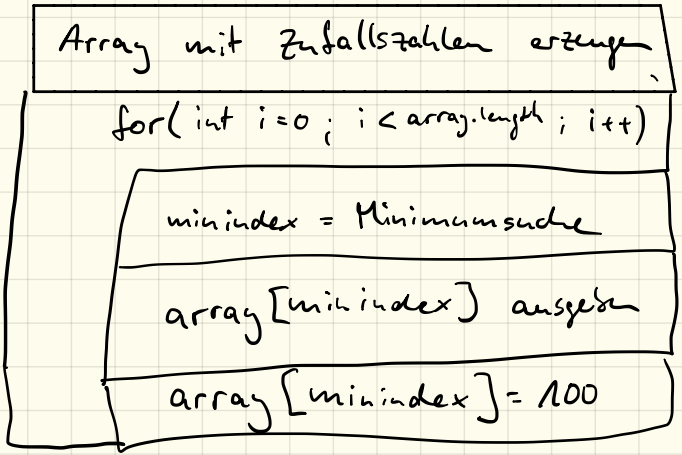
```
if (n % 2 == 0)   // n ist gerade
```

17.9.18

0 1 2 3 4 5 6

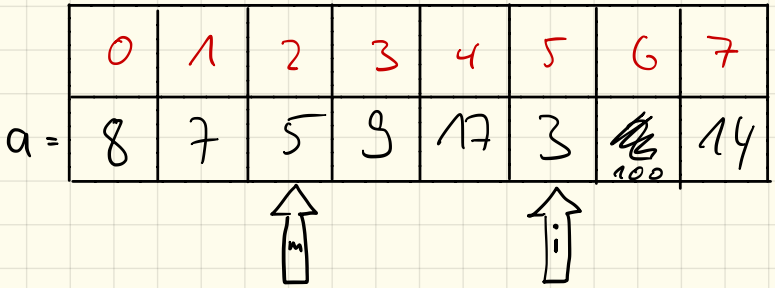
10, ~~7~~ 100, ~~100~~ 100, ~~100~~ 100, 9, 11, 17

3, 5, 7



```
int a =(int) (Math.random() * 50)
```

double



minindex = 6

i = 8

```
if(a[i] < a[minindex])
    minindex = i;
```

a[minindex] = 100

19.9.18

- Wiederholung Methoden
  - was ist eine Methode?
  - Parameter?
  - Rückgabetyp und -wert?
  - wohin schreiben?

$f(x) = x^2$

$f(5) = 25$

- Grundstrukturierung einer Klasse

- Sortierverfahren
  - SelectionSort mit Methoden
  - out-of-place vs. in-place
  - Geschwindigkeit

Methode um ein Array mit Zufallszahlen zu erzeugen:

Anzahl Elemente

Zahlen bis

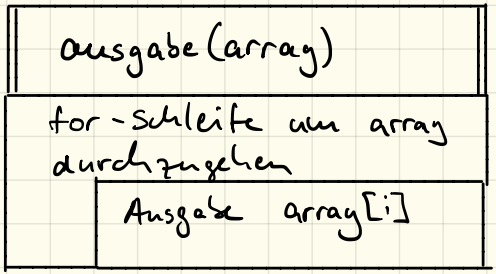
Ziel: `int[] array = zufall(20, 50);`

zufall(laenge, max)
int[] a = new int[laenge]; Array anlegen
for-Schleife, weist jedem Element Zufallszahl zu: a[i] = (int)(Math.random() * max);
Ergebnis zurückgeben return a;

```
public static int[] zufall(int laenge, int max) {
    int[] a = new int[laenge];
    for(int i=0; i<laenge; i++) {
        a[i] = (int)(Math.random() * max);
    }
    return a;
}
```

Methode, um ein Array auszugeben:

Ziel: `ausgabe(array);`

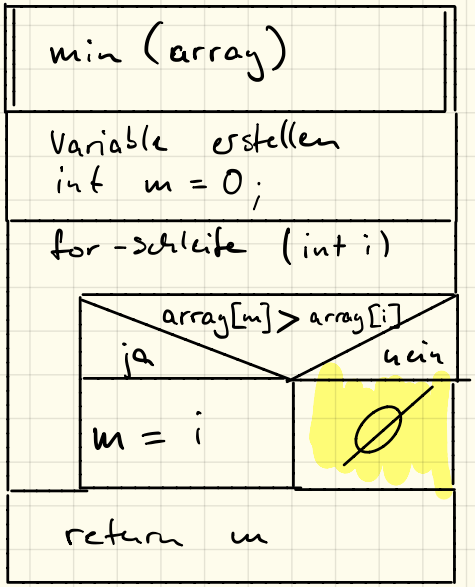



```
public static void ausgabe(int[] array){
    for(int i=0; array.length; i++){
        System.out.print(array[i]);
    }
}
```

24.9.18

Methode zur Minimumsuche gibt den Index der kleinsten Zahl zurück

```
int minindex = min ( array );
```




```
public static int min (int[] array ){

    int m = 0;
    for (int i=0 ; i<array.length ; i++) {
        if(array[m] > array[i]){
            m=i;
        }
    }
    return m;
}
```

Methode zur sortierten Ausgabe

sortierteAusgabe(array)

for-Schleife

int m = min(array)

Ausgabe array[m]

array[m] = 1000;

```
public static void sortierteAusgabe(int[] array){
    for(int i=0; i<array.length; i++){
        int m = min(array);
        System.out.println(array[m]);
        array[m] = 1000;
    }
}
```

sortiere(array)

Erstelle neues Array
int[] sortiert = new int[array.length]

for-Schleife

int m = min(array)

sortiert[i] = array[m]

array[m] = 1000;

return sortiert

```
public static int[] sortiert(int[] array){
    int[] sortiert = new int[array.length];
    for(int i=0; i<array.length; i++){
        int m = min(array);
        sortiert[i] = arrray[m];
        array[m] = 1000;
    }
    return sortiert;
}
```

SelectionSort – Teil I

1. sortierte Ausgabe

Erstelle in deinem Projekt `Sortierung` ein Paket `selectionsort`. Lege in diesem eine Klasse `Ausgabe` (inklusive `main`-Methode) an.
Erstelle ein Struktogramm und programmiere anschließend das Programm, welches:

- Ein Array mit 20 Zufallszahlen (zwischen 0 und 50) füllt.
- Dieses Array soll zunächst unsortiert ausgegeben werden.
- Mithilfe der Minimumsuche sollen wie Werte sortiert auf der Konsole ausgegeben werden.

Hinweis: du darfst natürlich den Code von letztem Mal nutzen und in die neue Klasse kopieren!
Tipp: Erstelle für die Minimumsuche eine eigene Methode, die als Rückgabewert den Index des kleinsten Elements zurückgibt.

Beispielausgabe auf der Konsole:

```
Unsortiert:
20,6,30,34,5,11,0,34,28,12,4,26,11,15,44,28,40,7,20,7
Sortiert:
0
4
5
6
7
7
...
```

2. Abspeicherung

Erstelle im Paket `selectionsort` eine Klasse `OutOfPlace` mit einer `main`-Methode.
Programmiert werden soll ein Programm, welches wie oben ein zufällig befülltes Array generiert. Anstatt die Werte direkt auszugeben, sollen diese nun in einem *neuen* Array gespeichert werden, damit diese Werte später im Programm wieder weiterverwendet werden können.
Gib zur Kontrolle nach der Sortierung mithilfe einer Schleife das sortierte Array auf der Konsole aus.

Beispielausgabe auf der Konsole:

```
Unsortiert:
20,6,30,34,5,11,0,34,28,12,4,26,11,15,44,28,40,7,20,7
Sortiert:
0,4,5,6,7,7,11,11,12,15,20,20,26,28,28,30,34,34,40,44
```

3. Zusatzaufgabe

Informiere dich über die Begriffe *out-of-place* und *in-place*. Was bedeuten diese im Hinblick auf Sortieralgorithmen?

SelectionSort – Teil II

1. Begriffe

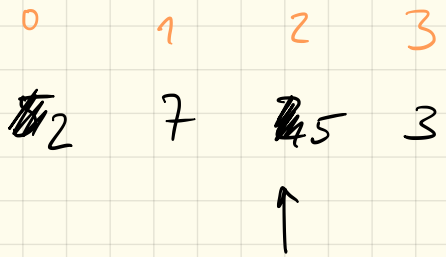
Beschreibe die beiden folgenden Begriffe im Hinblick auf Sortierverfahren:

out-of-place unsortiertes Array → sortiertes Array wird in einem neuen Array abgespeichert → doppelter Speicherplatz

in-place Elemente im unsortierten Array werden so vertauscht, dass die Sortierung entsteht → Originaldaten gehen verloren

2. *in-place*-SelectionSort

Der SelectionSort-Algorithmus kann auch in-place erfolgen. Beschreibe in freier Sprache, Pseudocode oder Ablaufdiagramm, welche Schritte hierbei durchgeführt werden müssen und was dabei zu beachten ist.




```
int tmp = array[i];   // = 5
array[i] = array[m];
array[m] = tmp;
```

3. Programmierung

Erstelle im Paket `selectionsort` eine neue Klasse `InPlace` mit `main`-Methode.
Programmiere darin den in-place-SelectionSort-Algorithmus anhand dem in Aufgabe 2 erarbeiteten Ablauf.

4. Zusatzaufgabe: Laufzeitmessung

Mit der Methode `System.currentTimeMillis()` kann man sich die Millisekunden seit dem 1.1.1970 zurückgeben lassen. Da diese Zahl sehr groß ist, reicht ein einfacher `int`-Wert nicht aus, in Java gibt es deshalb für große ganze Zahlen den Datentyp `long`. Die Methode gibt einen Wert von diesem Datentyp zurück.
Speichert man nun die Millisekunden *direkt vor* dem Sortiervorgang und zieht man diese vom Wert *direkt nach* dem Sortiervorgang ab, so erhält man die Laufzeit des Sortiervorgangs in Millisekunden.

Aufgabe: Erzeuge ein Array mit 25000, 50000, 100000, 200000 zufälligen Werten. Lasse diese dann mit deinem in-place-SelectionSort-Algorithmus sortieren und miss die dafür benötigten Zeiten.
Miss für jede Arraygröße 5 Zeiten und vergleiche die Durchschnittswerte.

Tabelle 1: Messwerte zur Sortierung von 25000 Zahlen

Messung	1	2	3	4	5	Durchschnitt
Laufzeit	330	341	329	327	327	331

Tabelle 2: Messwerte zur Sortierung von 50000 Zahlen

Messung	1	2	3	4	5	Durchschnitt
Laufzeit	1301	1291	1297	1292	1308	1298

Tabelle 3: Messwerte zur Sortierung von 100000 Zahlen

Messung	1	2	3	4	5	Durchschnitt
Laufzeit	5293	5214	5196	5240	5228	5234

Tabelle 4: Messwerte zur Sortierung von 200000 Zahlen

Messung	1	2	3	4	5	Durchschnitt
Laufzeit	21750	20881	20833	20925	20863	21050

a) Wie verändert sich die Laufzeit, wenn die Größe des Arrays verdoppelt, verdreifacht,... wird?

b) Wie lange würde es damit dauern, das Telefonbuch von Berlin mit ca. 3 Millionen Einträgen zu sortieren? *(grober Richtwert!)*

7 9 13 17 2 8 7 4

Projekt zu Sortierverfahren

Ziel

Ziel ist es, ein komplettes Projekt zu programmieren und damit verschiedene Sortierverfahren zu implementieren. Am Ende sollen die Projekte in einer kurzen Präsentation vorgestellt werden.

1. Allgemeine Kriterien

- Auch zu Hause kann und soll weitergearbeitet werden!
- Am 17. Oktober ist die Abgabe des Projektes und finden die Präsentationen statt.
- Präsentiert auch Probleme, auf die ihr gestoßen seid und berichtet, wie ihr diese umgehen konntet.
- Die Präsentation sollte nicht länger als ca. 10-15 Minuten dauern.

2. MUSS-Kriterien

Die *MUSS-Kriterien* müssen auf jeden Fall erfüllt werden. Werden nur diese erfüllt, so liegt die Endnote im Bereich von ca. 7 Punkten.

- Eine `Sortieren`-Klasse mit `main`-Methode. Hier soll ein Array einer festen Länge mit Zufallszahlen befüllt werden und anschließend mit den Sortierverfahren sortiert werden.
- Das Sortierverfahren **SelectionSort** muss **in-place** programmiert werden.
- Das Sortierverfahren **InsertionSort** muss programmiert werden.
- Der Quellcode muss kommentiert werden.

3. SOLL-Kriterien

Werden zusätzlich noch die *SOLL-Kriterien* erfüllt, so liegt die Endnote in etwa bei 11 Punkten.

- **MergeSort** als Beispiel für die „divide-and-conquer"-Technik soll implementiert werden.
- **BubbleSort** soll programmiert werden.
- Es soll eine Zeitmessung programmiert werden um die Schnelligkeit der verschiedenen Sortierverfahren miteinander vergleichen zu können.
- Die *durchschnittlich* benötigte Zeit soll für alle Sortierverfahren für unterschiedlich große Arrays gemessen und präsentiert werden.

4. DARF-Kriterien

Werden **alle** Kriterien erfüllt, so liegt die Endnote bei etwa 15 Punkten.

- Einer der folgenden Algorithmen darf programmiert werden:
  - **HeapSort**
  - **QuickSort**
  - **TimSort**
  - oder ein selbst gewählter Sortieralgorithmus.
- Der Ablauf des Algorithmus muss dann auch präsentiert werden.
- Informiert euch und präsentiert, wann man von einem **stabilen** Sortierverfahren spricht.
- Begründet für alle Algorithmen, ob diese stabil oder nicht-stabil funktionieren.

8.10.18

7 4 7 3 1 9 10 14 5

Minimumsuche: n Schritte

Sortierung: n mal die Minimumsuche

gesamter Aufwand: n^2

8.10.18

divide-and-conquer:

14

1	2	5	6	8	9	10	12	13	14

Minimumsuche: 1 Schritt
Sortierung: n mal

gesamter Aufwand: n

Liste mit 10 Elemente:

Selection Sort: 100 Schritte

d-a-c: Aufteilen in 2 Listen: 12 Schritte

Liste 1 Sortieren: 25 Schritte

Liste 2 Sortieren: 25 Schritte

Zusammenfügen: 10 Schritte

72 Schritte

Liste mit 1000 Elemente

SelSort: 1 000 000 n^2

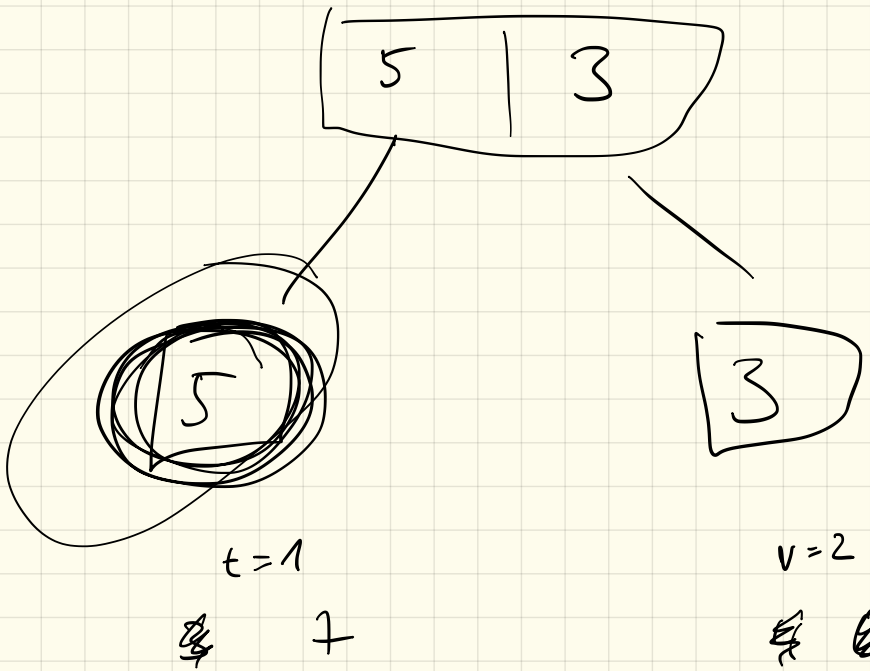
d-a-c: Aufteilen: 1002 $n + 2$

Liste 1 sortieren: 250 000 $\left(\frac{n}{2}\right)^2$

Liste 2 sortieren: 250 000 $\left(\frac{n}{2}\right)^2$

Zusammenfügen: 1000 n

502002 Schritte



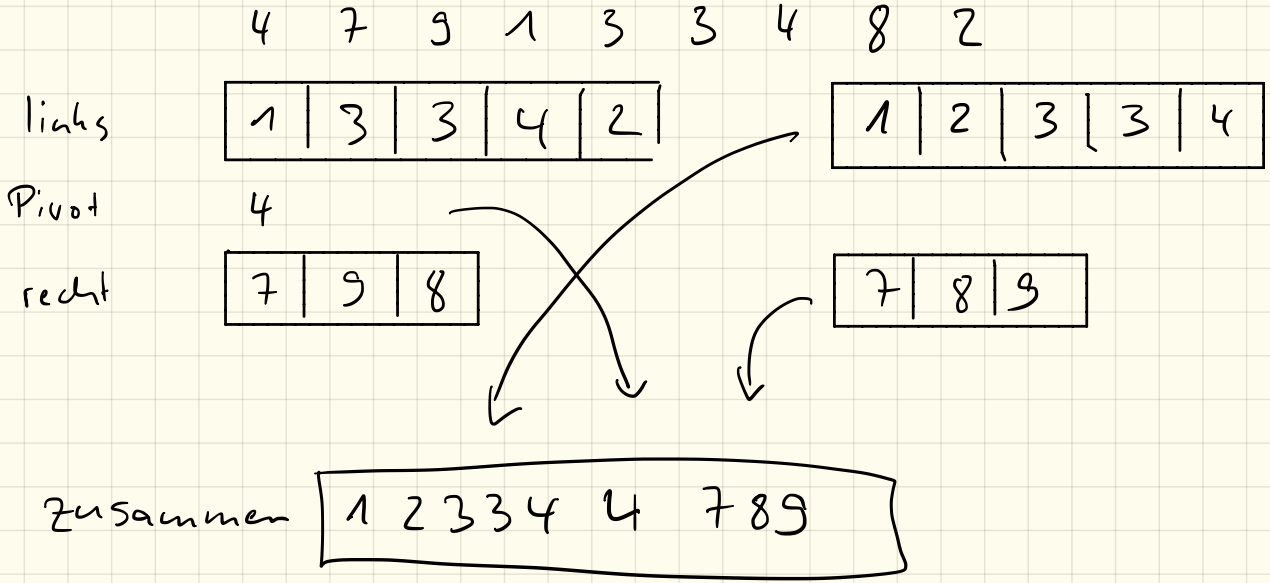
k = Mergesort(k)
g = Mergesort(g)

3 4 6

31 37 35 8 15

15.10.18

QuichSort



```
if(array.length <= 1)
    return array;
```

1	3	3	4	2

links: []

Pivot: 1

rechts: 3 3 4 2

2	3	3	4

zusammen

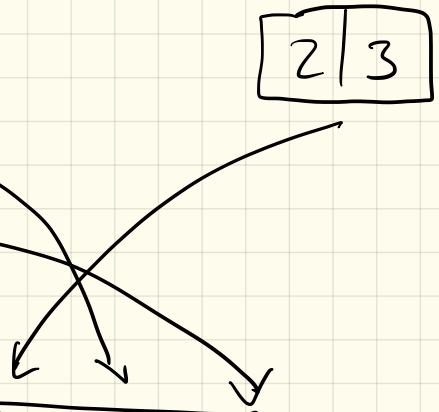
1	2	3	3	4

3 3 4 2

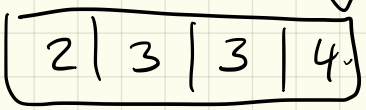
links 3 2

Pivot 3

Rechts 4



Zusammen



3 2

links 2

pivot 3

rechts []

zusammen | 2 | 3 |

Klausur 22.10.2018

Name: _____ VP: _____/24P NP: _____ mündlich: _____

1. Sortierverfahren (3P)

Beschreibe die Begriffe

a) in-place b) out-of-place c) stabil

in Bezug auf Sortierverfahren.

2. InsertionSort (4P)

Erkläre InsertionSort. Beschreibe dieses schrittweise in Pseudocode bzw. einem Struktogramm.

3. SelectionSort (4P)

Gegeben ist nachfolgender Code. Gib mit Begründung an, ob dieser SelectionSort **in-place** oder **out-of-place** arbeitet und ob das die Sortierung **stabil** verläuft.

Listing 1: SelectionSort

```
1  public class SelectionSort {
2          public static void main(String[] args) {
3                  int[] xyz = zufall(20,10);
4                  int[] sort = sortiere(xyz);
5
6                  // Ausgabe
7                  for(int i=0 ; i<sort.length ; i++) {
8                          System.out.print(sort[i] + ",");
9                  }
10         }
11
12         // generiert ein zufällig gefülltes Array
13         public static int[] zufall(int laenge, int max) {
14                 int[] a = new int[laenge];
15                 for(int i=0;i<laenge;i++) {
16                         a[i] = (int)(Math.random()*max);
17                 }
18                 return a;
19         }
20
21         // Out-of-Place-Sortierung SelectionSort
22         public static int[] sortiere(int[] array) {
23                 int[] sortiert = new int[array.length];
24                 for(int i=0 ; i<array.length ; i++) {
25                         int m = min(array, 0);
26                         sortiert[i] = array[m];
27                         array[m] = 9999;
28                 }
29                 return sortiert;
30         }
31
32         // Minimumsuche
33         public static int min(int[] array, int start) {
34                 int m = start;
35                 for(int i=start ; i<array.length ; i++) {
36                         if(array[m]>=array[i]) {
37                                 m = i;
38                         }
39                 }
40                 return m;
41         }
42 }
```

1 3 5 1 2 4

1 1 2 3 4 5

Aufgabe 3:

4. Laufzeit (5P)

Erkläre kurz, wie MergeSort funktioniert. Warum funktioniert das schneller als beispielsweise SelectionSort? Begründe z. B. mit einer kurzen Rechnung.

$n = 100 \rightarrow$ SelectionSort ~ 10000

MergeSort: Liste 1: 50 $\xrightarrow{\text{Selection}}$ 2500

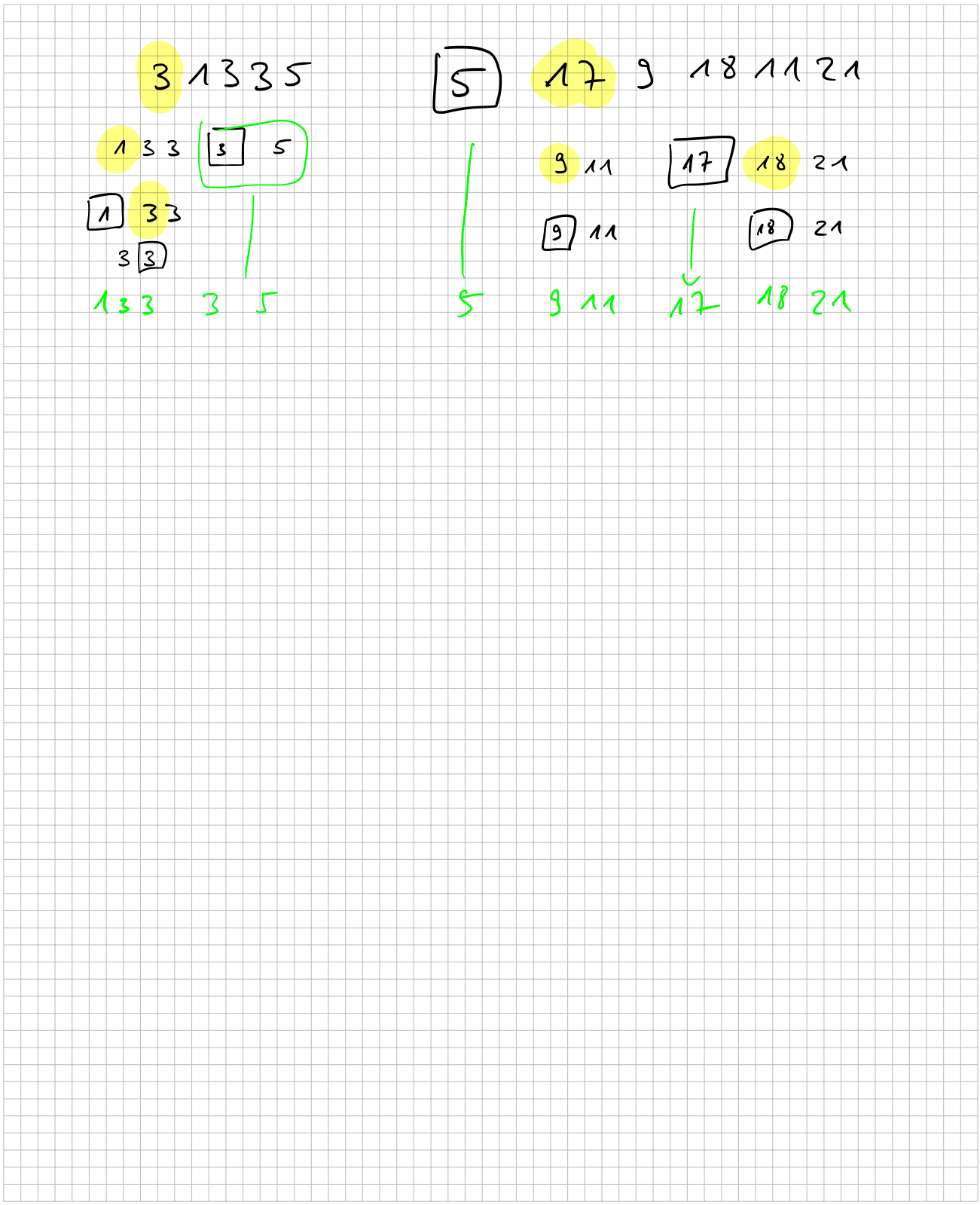
Liste 2: 50 $\xrightarrow{\text{Selection}}$ 2500

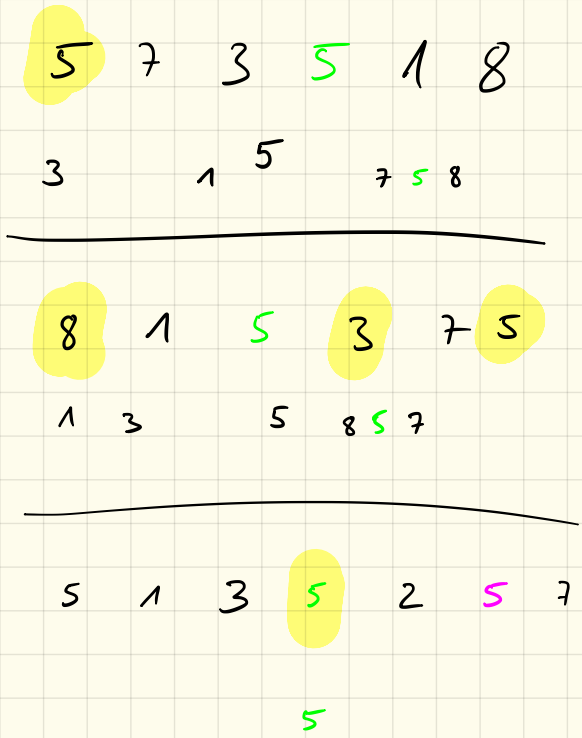
5000 Schritte

5. QuickSort (8P)

Führe mit folgendem Array den QuickSort-Algorithmus durch und beschreibe jeden Schritt den du durchführst kurz. Wann ist der QuickSort-Algorithmus besonders effizient, wann besonders ineffizient?

5	3	17	9	1	3	18	11	21	3	5




5 7 3 5 1 8
3 1 5 7 5 8
8 1 5 3 7 5
1 3 5 8 5 7
5 1 3 5 2 5 7
5

Landau - Symbole

Sel Sort out-of-place

n Schritte Minimum

n Minimumsuchen

$\Rightarrow$ immer n^2 Schritte

$\Rightarrow$ Aufwand $\Theta(n^2)$

Sel in-place

im Durchschnitt $\frac{n}{2}$ Schritte

n Minimumsuchen

$\Rightarrow$ immer $\frac{n^2}{2}$ Schritte

$\Rightarrow$ Aufwand $\Theta\left(\frac{n^2}{2}\right)$, $\Theta(n^2)$

Insertion Sort → im besten Fall n Schritte

$\Rightarrow \Theta(n)$

1 2 3 4 5 6

→ im schlechtesten Fall

6 5 4 3 2 1 n-1 Zahlen anschauen

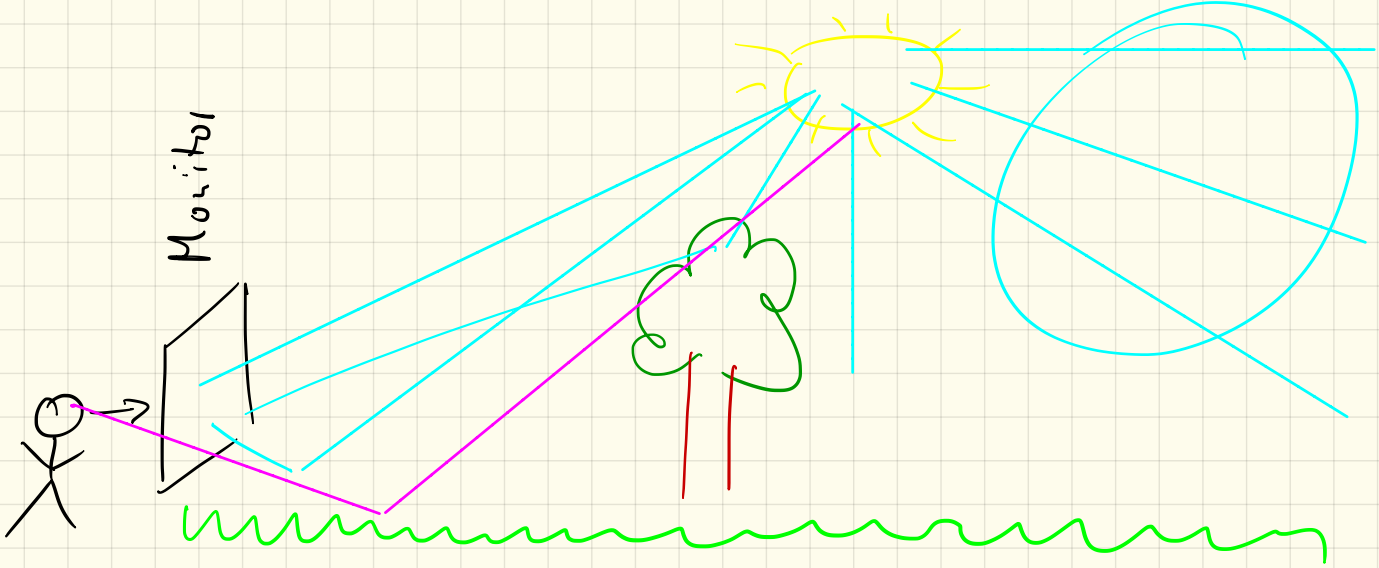
5 → 1 Stelle
4 → 2 Stellen
3 → 3 Stellen
⋮
n-1 Stellen

$\frac{n-1}{2}$ Verschiebungen im Durchschnitt

$\Rightarrow$ gesamt $\frac{(n-1)^2}{2} = \frac{n^2-2n+1}{2}$

$\Theta\left(\frac{n^2-2n+1}{2}\right)$

$\Rightarrow \mathcal{O}(n^2)$


Монитор

Objektorientierte Programmierung mit einem Raytracer

Projekt in Eclipse importieren

Da wir jetzt eine externe Funktionalität benutzen wollen, müssen wir diese Funktionen zuerst in Eclipse importieren:

1. Zuerst klicke mit der rechten Maustaste in die Projektübersicht und wähle die Funktion `Import...`
2. Wähle dann unter `General`: `Existing Projects into Workspace` und klicke auf `Next >`
3. Wähle aus dem Tauschlaufwerk im Projektverzeichnis für den Informatikkurs den Ordner `Raytracing` aus.
4. Unbedingt den Haken bei `Copy projects into workspace` setzen!
5. Mit einem Klick auf `Finish` wird das Projekt importiert

Eigenes Projekt anlegen

In unserem eigenen Projekt wollen wir die Funktionen des `Raytracing`-Projekts nutzen und müssen diese deshalb angeben wenn wir unser Projekt erstellen:

1. wähle wie bisher im Menü `File`→ `New`→ `Java Project`
2. gib den Namen `OOP` ein, wir werden die kommenden Wochen an diesem Projekt arbeiten und dieses weiterentwickeln
3. klicke **nicht** auf `Finish` sondern auf `Next >`
4. wähle im darauffolgenden Dialog `Projects` und füge das `Raytracing`-Projekt hinzu.
5. Mit einem Klick auf `Finish` wird das Projekt importiert

Damit können wir in unserem Projekt `OOP` die Funktionen des `Raytracing`-Paketes benutzen.
Für die bessere Strukturierung lege in dem Projekt ein Paket `start` an und darin eine Klasse `Start` (diese wieder mit der `main`-Methode)

Raytracer benutzen

Um den Raytracer benutzen zu können müssen wir die Pakete importieren mit `import raytracing.*;`
Anschließend legen wir in der `main`-Methode den Raytracer an mit

```
public static void main(String[] args) {
    Tracer tr = new Tracer();
}
```

Listing 1: Anlegen des Raytracers

Wenn wir so das Programm ausführen, so öffnet sich nur ein leeres, schwarzes Fenster.
Mit der Methode `tr.setPixel( x , y , r , g , b );` können wir einen einzelnen Pixel an der Koordinate $(x \mid y)$ auf einen RGB-Farbwert (r,g,b) setzen.
Hierbei ist zu beachten, dass die x-Koordinate wie gewohnt von ganz links $(x = 0)$ bis ganz rechts hochgezählt wird, die y-Koordinate jedoch von oben $(y = 0)$ nach unten hochgezählt wird! Die Fensterbreite bzw. -höhe bekommen wir mit Methode `tr.getWidth()` bzw. `tr.getHeight()`.
Die RGB-Farbwerte liegen jeweils zwischen 0 (dunkel) und 1 (volle Farbe).

1. Aufgabe

Lege das Projekt an und zeichne manuell den Anfangsbuchstaben von deinem Namen in das Fenster, indem du die einzelnen Pixel einfärbst.

2. Aufgabe

a) Lasse (mithilfe einer `for`-Schleife) eine Zeile des Fensters einfärben

b) Lasse (mithilfe einer `for`-Schleife) eine Spalte des Fensters einfärben

c) Kombiniere diese beiden Schleifen um das ganze Fenster einzufärben

d) Probiere auch unterschiedliche Farben selbst aus um dich mit dem RGB-Farbschema vertraut zu machen.

e) *Zusatzaufgabe:* Färbe das Fenster so ein, dass der Pixel in der linken oberen Ecke schwarz ist, und der Rotwert nach rechts zunimmt bis er auf der rechten Seite dann bei $r = 1$ ist. Nach unten soll der Grünwert gleichermaßen zunehmen.

Objekte sichtbar machen

In dem virtuellen Raum im Fenster (diesen nennt man auch *Szene*) sind auch einige Objekte versteckt. Du kannst die Methode `tr.trifft( x , y )` benutzen um herauszufinden, ob ein Lichstrahl, der vom Auge ausgeht und durch den Pixel $(x \mid y)$ geht, ein Objekt in der Szene trifft. Die Methode liefert als Ergebnis also einen `boolean`-Wert zurück den wir mit einer `if`-Bedingung abfragen können.

3. Aufgabe

Benutze die `for`-Schleifen von oben, um jeden Pixel des Fensters zu durchlaufen. Teste damit jeden Pixel auf einen Treffer mit einem Objekt und setze den Pixel bei einem Treffer auf eine Farbe.

4. Aufgabe

Neben der Methode `tr.trifft( x , y )` kannst du auch die Methoden `tr.rot( x , y )`, `tr.gruen( x , y )` und `tr.blau( x , y )` benutzen. Diese liefern – sofern ein Objekt getroffen wird – als Ergebnis jeweils einen `double`-Wert mit der jeweiligen RGB-Farbkomponente.
Benutze diese, um die Objekte der Szene in der passenden Farbe anzuzeigen.

Objektorientierte Programmierung mit einem Raytracer

Methoden

Wir können uns mit der Methode `tr.getObjekte();` alle Objekte in unserer Szene als Array holen. Hierfür müssen wir das Paket `raytracing.objekt.Objekt` importieren. Anschließend reicht der Aufruf

```
Objekt[] obj = tr.getObjekte();
```

Listing 1: Holen der Objekte

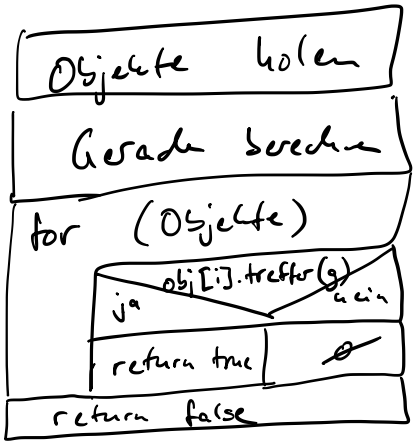
um alle in der Szene befindlichen Objekte in einem Array zu speichern.
Für jedes Objekt `obj[i]` gibt es dann widerum eine Methode `treffer(Gerade g)`, welche testet, ob die Gerade g das Objekt schneidet und dann einen entsprechenden `boolean`-Wert zurückgibt.
Die Gerade g erhalten wir widerum über die Methode `tr.getGerade(int x,int y)` welche x- und y-Koordinate eines Pixels annimmt und als Ergebnis eine `Gerade` liefert. Dazu müssen wir jedoch das Paket `raytracing.math.Gerade` importieren.

1. Aufgabe

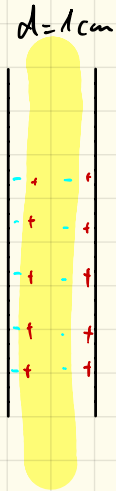
Ziel wird es sein, die Methode `tr.trifft(int x,int y)` so nachzubilden, dass wir diese später erweitern können.
Lege dazu im `OOP`-Projekt in der `Start`-Klasse eine neue Methode `trifft` an:

- die Methode soll wie bisher `public static` sein
- als Ergebnis soll die Methode einen `boolean`-Wert zurückgeben
- es werden 3 Parameter angenommen:
  - im ersten Parameter soll der `Tracer tr` übergeben werden
  - im zweiten Parameter soll die x-Koordinate übergeben werden
  - im dritten Paramter soll die y-Koordinate übergeben werden

Diese können wir anschließend in unserer `main`-Methode benutzen und damit die `tr.trifft`-Methode ersetzen.
Auf die gleiche Art können wir die Methoden `tr.rot`, `tr.gruen` und `tr.blau` ersetzen. Den Farbwert eines einzelnen Objektes in unserer Szene bekommen wir über die Methoden `obj[i].rot()`, `obj[i].gruen()` und `obj[i].blau()`.




```
public static boolean trifft(Tracer tr, int x, int y){
    Objekt[] obj = tr.getObjekte();
    Gerade g = tr.getGerade(x,y);
    for(int i=0; i<obj.length; i++){
        if(obj[i].treffer(g)){
            return true;
        }
    }
    return false;
}
```



$\varepsilon_r = 5{,}0$

$U_0 = 1000\,V$

$A = 500\,cm^2 = 5\,dm^2 = 0{,}05\,m^2$

$d = 1\,cm = 0{,}01\,m$

$$E = \frac{U}{d} = \frac{1000\,V}{0{,}01\,m} = 100000\,\frac{V}{m}$$

a) $Q = C \cdot U$ $\qquad C = \varepsilon_0 \cdot \varepsilon_r \cdot \frac{A}{d}$ $\qquad C_{mit} = \varepsilon_r \cdot C_{ohne}$

$Q_{mit} = C_{mit} \cdot U = 5 \cdot C_{ohne} \cdot U = 5 \cdot Q_{ohne}$

$\Rightarrow$ Kapazität erhöht sich auf das 5-fache

b) $U = \frac{Q}{C}$ $\rightarrow$ U sinkt auf $\frac{1}{5}$ $\rightarrow 200\,V$

$$E = \frac{U}{d} = \frac{200\,V}{0{,}01\,m} = 20000\,\frac{V}{m}$$

$$\varepsilon_0 = 8{,}85 \cdot 10^{-15} \; \frac{V_s}{Am}$$

$$C = \varepsilon_0 \cdot \frac{A}{d} = 4{,}425 \cdot 10^{-11} \; F$$

$$Q_{ohne} = C \cdot U \qquad Q_{mit} = 5 - Q_{ohne}$$

$$Q_p = \Delta Q = 4 \cdot Q_{ohne}$$
$$= 4 \cdot C \cdot U$$
$$= 1{,}77 \cdot 10^{-7} C$$

E nimmt um $\frac{4}{5}$ ab

$$Q_p = \frac{4}{5} \cdot Q_{ohne}$$
$$= 3{,}54 \cdot 10^{-8} C$$

Farbe

```
public static double rot(Tracer tr, int x, int y) {
    Objekt[] obj = tr.getObjekte();
    Gerade g = tr.getGerade(x, y);
    Farbe ergebnis = new Farbe()
    for(int i=0;i<obj.length;i++) {
        if(obj[i].treffer(g)) {
            return obj[i].rot();
        }
    }
    ergebnis.rot = 0;
    ergebnis.blau = 0;
    ergebnis.gruen = 0;
    return 0; ergebnis;
}
```

```
ergebnis.rot = obj[i].rot();
ergebnis.gruen = obj[i].gruen();
ergebnis.blau = obj[i].blau();
return ergebnis;
```

```
public class Farbe {

    public double rot;
    public double gruen;
    public double blau;

}
```

```
int a = 77;
```

```
Farbe x = new Farbe();
x.rot = 0.5;
x.gruen = 0.7;
x.blau = 0.1;
```

```
Farbe y = new Farbe();
y.rot = 0;
y.gruen = 0.1;
y.blau = 0.9;
```

Person

Farbe: Haarfarbe

Farbe: Augenfarbe

int: Größe

Farbe: Hautfarbe

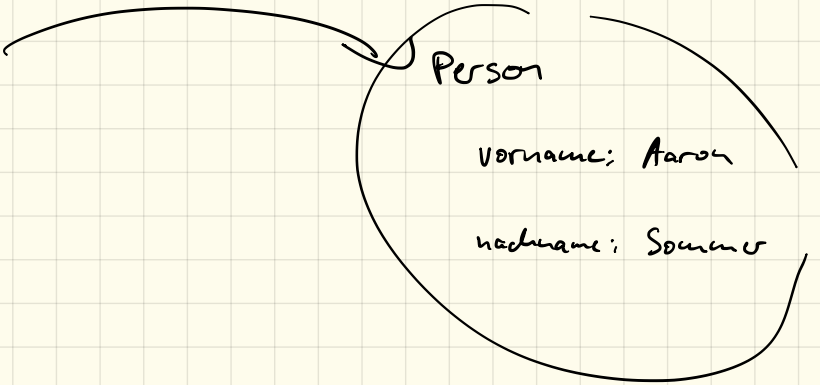
boolean: männlich

String: vorname

String: nachname

Methode Haare färben

Methode erstelle Anrede




```
Person p = new Person();
p.haarfarbe.rot = 0.7;
```

Programmiert werden sollen Bankkonten: 19.11.2018

* Jedes Bankkonto hat einen Startbetrag
* Man kann Geld abheben und einzahlen
* Man kann Geld von einem auf ein anderes Konto überweisen.
* Die Überweisung soll nur funktionieren, wenn auf dem Konto genug Geld ist.

a) Welche Eigenschaften/Attribute muss so ein Bankkonto enthalten?
b) Welche Methoden müssen programmiert werden?

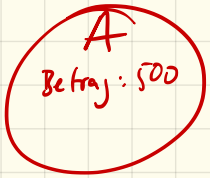
a)

Bankkonto

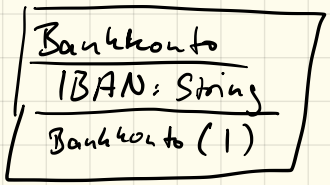
IBAN: String
BIC: String
Inhaber: String
Betrag: double

b

Ausgabe()
Einzahlen(double Betrag)
Auszahlen(double Betrag)
Überweisen(double Betrag, Bankkonto Ziel)

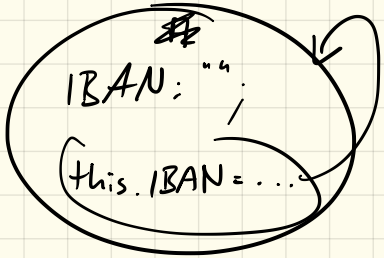


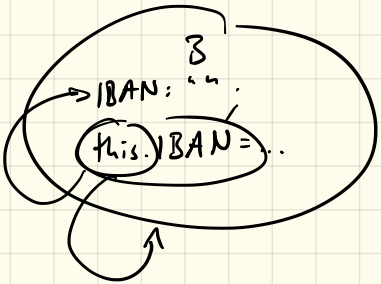
A.Ueberweisen(100, B);



Bankkonto A = new Bankkonto ("DE1");

Bankkonto B = new Bankkonto ("DE2");



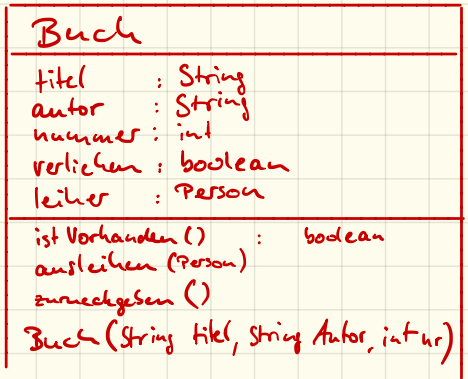


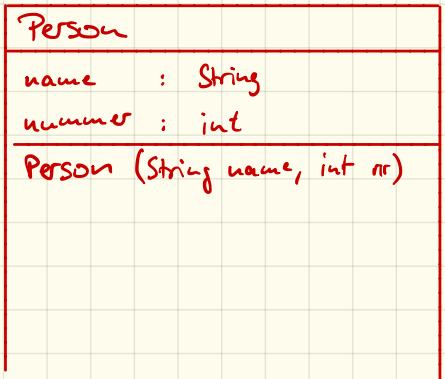
Aufgabe:

21.11.2018

Erstelle eine Bibliothek:

* Diese besteht aus Personen
  * Personen haben einen Namen und eine eindeutige Nummer
* Außerdem gibt es Bücher
  * Bücher haben einen Titel, einen Autor und ebenfalls eine Nummer
  * Außerdem ist gespeichert, ob ein Buch verliehen ist oder nicht
  * Und an wen
* In der Bibliothek gibt es viele Bücher und einige Personen
* Bücher können von einer Person ausgeliehen und wieder zurückgegeben werden





Klassen und Objekte

Objektorientierung

Bei der objektorientierten Programmierung werden mehrere *Attribute* und *Methoden* zu logischen oder realen **Objekten** zusammengefasst.

Objekte

Ein Objekt ist ein eindeutig identifizierbares Element. Dieses enthält gewisse Informationen und kann bestimmte Aktionen ausführen.
Beispiele für Objekte sind:

- **reale Dinge**, die man aus dem Alltag kennt und die sich beschreiben lassen, wie z. B. eine Person, ein Buch, ein Auto,...
- **Rollen**, die beispielsweise eine Person annehmen kann und die mit gewissen Eigenschaften verbunden ist, wie z. B. Chef, Angestellter, Kunde, Student, Schüler,...
- **Ereignisse/Vorgänge**, welche man sich vorstellen kann, aber nicht Gegenständlich existiert, wie z. B. ein Meeting, eine Bestellung, eine Schulnote,...

Klassen

Mit der Erkenntnistheorie legten Platon und Aristoteles bereits im antiken Griechenland den Grundstein zum Verständnis von Objekten und Klassen. Man spricht dabei von einer „ist ein“-Beziehung:

- Herr Herbert Huber *ist eine* Person
- Herr Huber *ist ein* Patient
- Herbert *ist ein* Kunde in der örtlichen Bibliothek
- Mein Toyota Yaris *ist ein* Auto
- Ein Tesla S *ist ein* Auto

Die konkreten „Dinge“ (Herbert, Yaris, Tesla) sind hierbei *Objekte*. Diese werden in die *Klassen* „Person“, „Patient“, „Kunde“ und „Auto“ eingeteilt. Ein einzelnes Objekt nennt man auch **Instanz**.
Diese Klassifizierung ist grundsätzlich nicht von Natur aus gegeben. Erst durch unsere Beobachtungen werden den Klassen verschiedene Attribute zugewiesen, so haben Objekte

- der Klasse *Patient* beispielsweise die Attribute „Name“, „Krankenkasse“, „Versicherungsnummer“, „Krankenvorgeschichte“,...
- der Klasse *Kunde* die Attribute „Name“, „Kundennummer“,...
- der Klasse *Auto* die Attribute „Farbe“, „Leistung“, „Sitzplätze“, „Antriebsart“,...

An den Klassen „Patient„ und „Kunde“ lässt sich erkennen, dass die selbe Person, je nach Kontext unterschiedliche Attribute haben kann.

Klassendefinition

Eine Klasse enthält Definitionen, welche *Attribute* und *Methoden* Instanzen dieser Klasse beinhalten.

Attribute

Attribute oder Eigenschaften sind bestimmte Werte, wie z. B.

- Name
- Farbe
- Kontonummer
- Alter
- Leistung
- Guthaben

Methoden

Methoden beschreiben die Möglichkeiten, die ein Objekt ausführen kann, wie z. B.

- `Ausgabe()` → Irgendetwas auf der Konsole ausgeben
- `Abheben()` → Geld vom Konto abheben
- `Zurueckgeben()` → Buch in die Bücherei bringen

`public` vs. `private`

Sowohl Attribute als auch Methoden können als `public` oder als `private` angelegt werden. Der Unterschied ligt darin, dass...

public: von überall aufrufbar und verwendbar

private: nur aus der „eigenen" Klasse verwendbar

- Attribute, die nicht öffentlich einsehbar sein sollen → private
- Attribute, die nicht frei veränderbar sein sollen → private

Anlegen von Objekten, Konstruktor

Eine Instanz kann man mit dem `new`-Befehl erzeugen:

```
1    Kunde sc = new Kunde();
```

Hierbei wird der sogenannte *Konstruktor* aufgerufen. Der Konstruktor ist...

eine Methode, die beim Anlegen einer Instanz aufgerufen wird.
Er legt die Attribute fest (initialisiert diese)

public KLASSENNAME(...) → keinen Rückgabetyp
→ Name der Methode entspricht dem Klassennamen

Autohaus

1. Vorüberlegungen

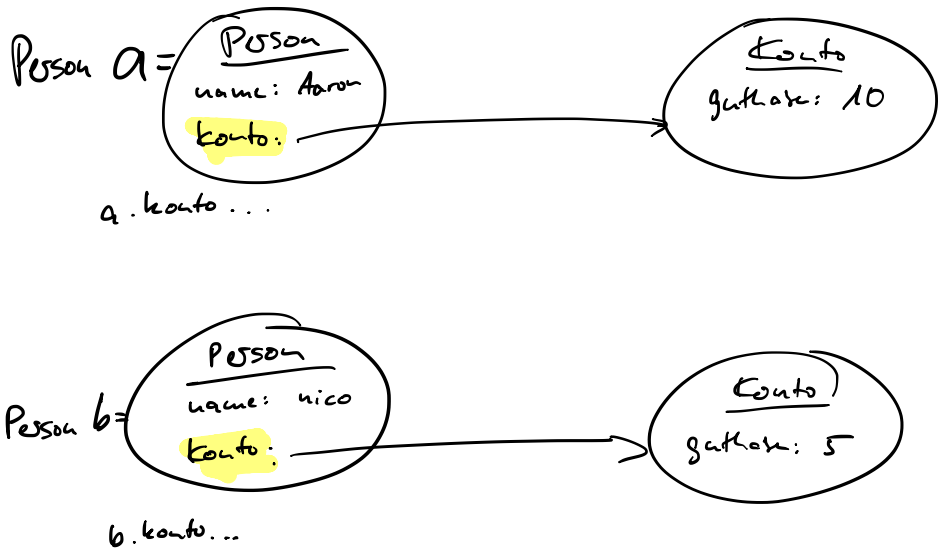
Um ein Programm von vornherein gut strukturiert programmieren zu können ist eine gute Vorplanung nötig. Diese passiert immer zuerst auf Papier und besteht darin, sich zu überlegen, welche Klassen das Programm beinhaltet und welche Attribute und Methoden diese beinhalten sollen.
Hier sollst du ein einfaches Autohaus programmieren mit den Klassen: `Auto`, `Person`, `Konto`. (Und eine Haupt-Klasse, die die `main`-Methode beinhaltet)
Beim Kauf eines Autos durch eine Person soll dabei der nötige Betrag von dessen Konto auf das Konto des Autohauses überwiesen werden (sofern möglich!)
Entwerfe zunächst also die Klassen: (beachte dazu auch die Tests auf der Rückseite!)

Auto	**Person**	**Konto**
Attribute: marke: String modell: String preis: double	*Attribute:* name: String bankkonto: Konto	*Attribute:* inhaber: String / Person guthaben: double
Methoden: Auto(modell: String, marke: String, preis: double) kaufen(kaeufer: Person, verkaeufer: Person)	*Methoden:* Person(name: String) kaufen(auto: Auto, verkaeufer: Person)	*Methoden:* Konto(inhaber: String) Einzahlen(betrag: double) Ueberweisen(betrag: double, ziel: Konto) :boolean

2. Programmierung und Tests

Programmiere diese Klassen und teste sie, indem du in der `main`-Methode

- 2 Autos anlegst
- 2 Personen anlegst (hierbei darf jeweils ein Konto auch automatisch mit angelegt werden!)
- zusätzlich eine „Person“ Autohaus anlegst
- auf das Konto einer Person genug Geld für einen Autokauf einzahlst
- die beiden Personen jeweils ein Auto kaufen lässt
- sofern nicht genug Geld auf dem Konto ist, soll der Kauf mit einer Fehlermeldung abgebrochen werden.



3.11.18

Klasse:
- Definiert Attribute (ohne Werte)
- ist Form / Vorlage

Objekt:
- hat Werte für die Attribute
- konkrete Instanz

Beispiel: Schule

- Schüler
- Lehrer
- Hausmeister
- Schulleiter

Person

vorname
nachname

Schüler

vorname
nachname
klasse

Lehrer

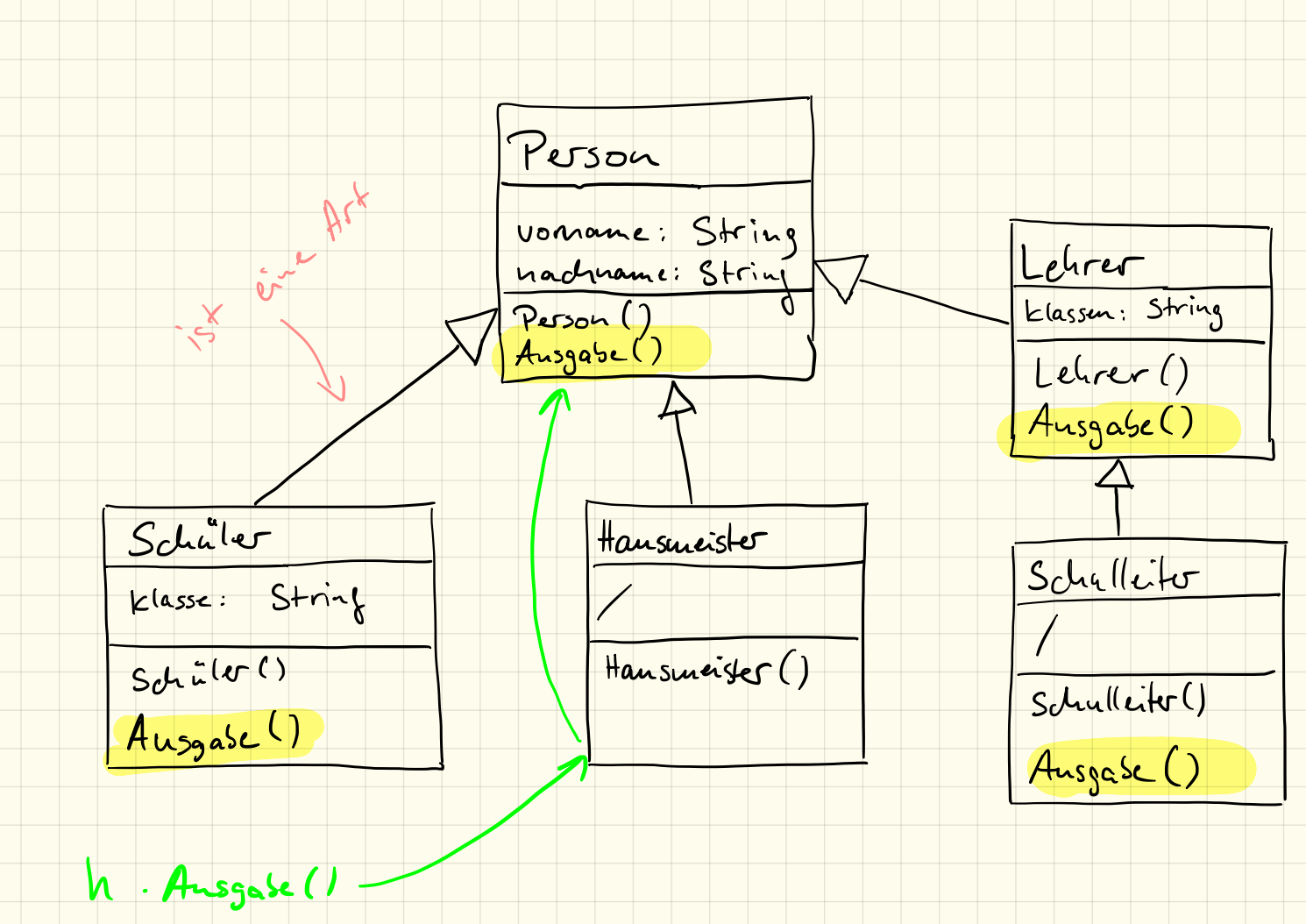
vorname
nachname
klassen[]

Schulleiter

vorname
nachname
klassen[]

Hausmeister

vorname
nachname


Person
vorname: String
nachname: String
Person()
Ausgabe()
ist eine Art
Lehrer
klassen: String
Lehrer()
Ausgabe()
Schüler
klasse: String
Schüler()
Ausgabe()
Hausmeister
/
Hausmeister()
Schulleiter
/
Schulleiter()
Ausgabe()
h.Ausgabe()

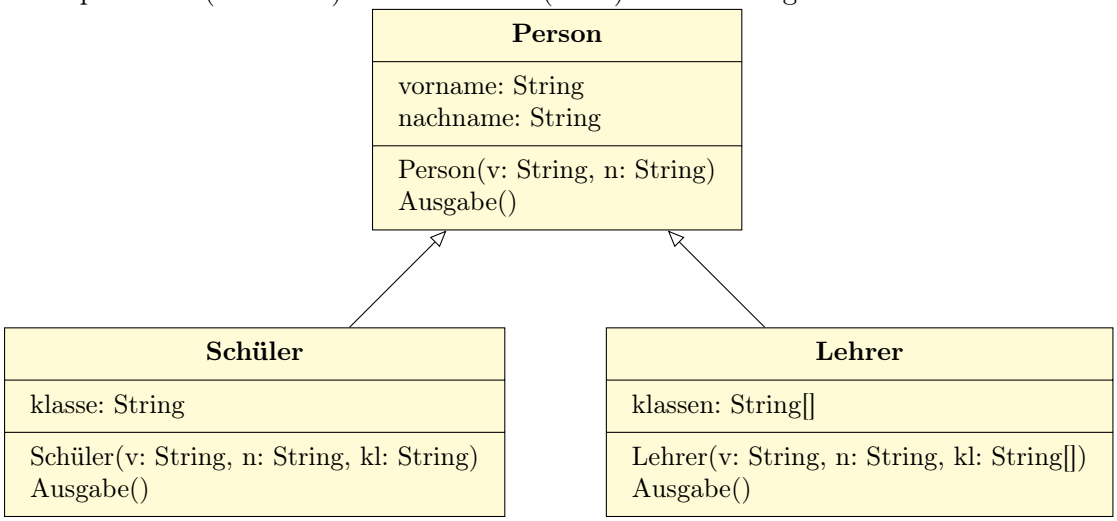
UML, Vererbung und Aggregation

1. UML

Die *Unified Modeling Language* (vereinheitlichte Modellierungssprache), kurz *UML*, ist eine Möglichkeit, Strukturen und Zusammenänge eines Programmes grafisch darzustellen.

2. Vererbung

Die sogenannte Vererbung ermöglicht es Informationen (Variablen) und Verhalten (Methoden / Operationen) weiterzugeben. Dies ist eine wesentliche Möglichkeit um Redundanz zu vermeiden. Die Erben fügen dann weitere Informationen und/oder Verhalten hinzu. Zwei Klassen stehen dabei zueinander als Superklasse (Erblasser) und Subklasse (Erbe) in Beziehung.[1]



Im Beispiel **spezialisieren** die Klassen `Schüler` und `Lehrer` die *Basisklasse* `Person` und fügen dieser weitere Attribute hinzu. Man spricht dabei auch von einer *„ist-eine-Art-von“*-Beziehung.
Die Vererbung wird im UML-Diagramm mit einem leeren Pfeil dargestellt.

2.1 Umsetzung in Java

In Java werden Unterklassen mit dem `extends`-Befehl angelegt:

```
1 public class Schüler extends Person {
2     [...]
3 }
```

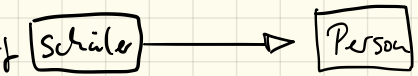
2.2 `protected` vs. `private`

Soll von einer Unterklasse auch auf Attribute der Hauptklasse zugegriffen werden, so dürfen diese nicht als `private` markiert sein. Stattdessen kann man hierzu das Schlüsselwort `protected` verwenden. Hierbei erhalten neben der eigentlichen Klasse (wie bei `private`) auch abgeleitete Unterklassen Zugriff.

[1] Quelle: `https://de.wikibooks.org/wiki/Java_Standard:_Vererbung`

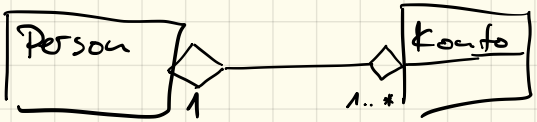
Vererbung

"ist-eine-Art"-Beziehung



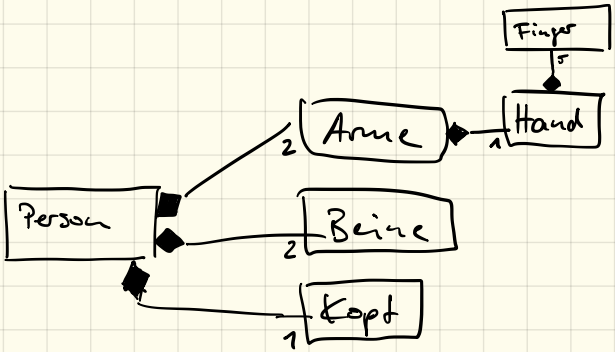
Aggregation

"hat"-Beziehung



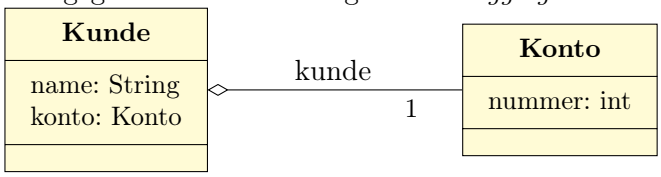
Composition

"besteht-aus"-Beziehung



3. Aggregation

Demgegenüber steht die sogenannte *Aggregation* oder auch *„hat“*-Beziehung.



Diese wird im UML-Diagramm mit einer leeren Raute dargestellt. Häufig wird auch die sogenannte *Multiplizität* als Zahl an die Verbindung geschrieben. Im Beispiel bedeutet diese, dass ein Kunde *genau ein* Konto hat.
Andere Multiplizitäten sind:

- `1`: genau eins
- `1..n`: mindestens eins, maximal n
- `1..*`: mindestens eins, ohne Maximale Anzahl
- `0..*`: muss keins enthalten, ohne maximale Anzahl (oft auch nur als `*` geschrieben)

4. Aufgabe

Bringe die folgenden Klassen und deren Beziehungen in ein UML-Diagramm:

- Pflanze
- Baum
- Stamm
- Ast
- Blatt
- Strauch
- Buche
- Eiche
- Lavendel

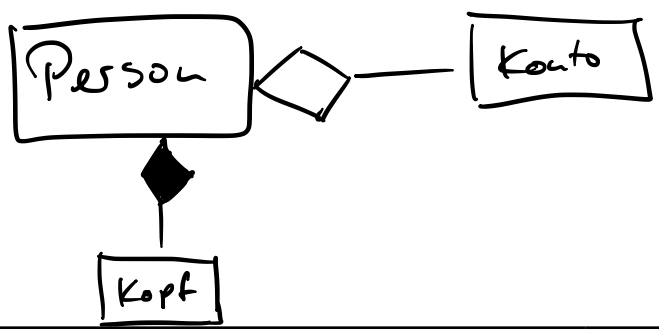
5. Aufgabe

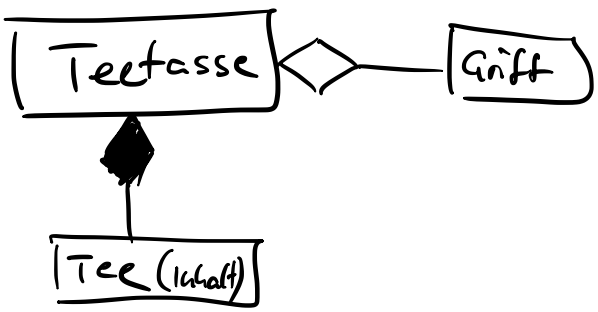
Zeichne ein UML-Diagramm mit folgenden Begriffen:

- Kamel
- Dromedar 
- Trampeltier 
- Lama 
- Höcker
- Gewicht
- Alter
- Größe
- Farbe

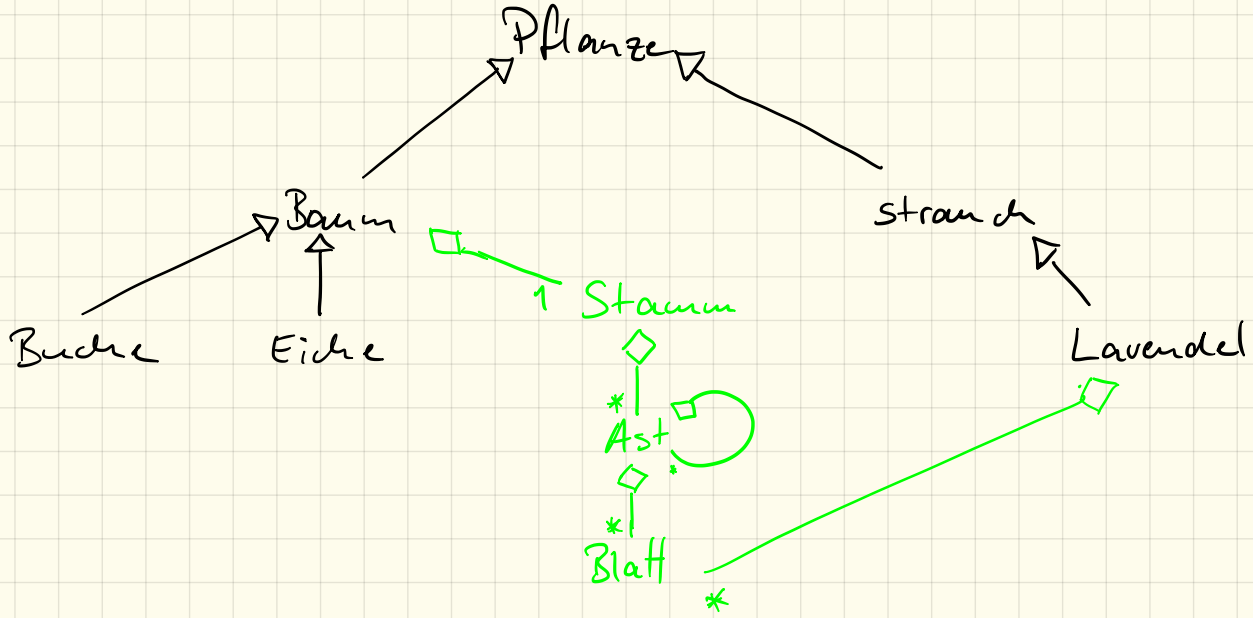
6. Aufgabe: Komposition

Informiere dich über den Unterschied zwischen *Aggregation* und *Komposition* und nenne ein Beispiel dafür.
Wie wird eine Komposition im UML-Diagramm dargestellt?

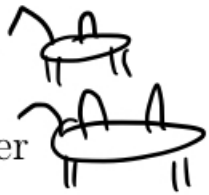




- ~~Pflanze~~
- ~~Baum~~
- ~~Stamm~~
- ~~Ast~~
- ~~Blatt~~
- ~~Strauch~~
- ~~Buche~~
- ~~Eiche~~
- ~~Lavendel~~

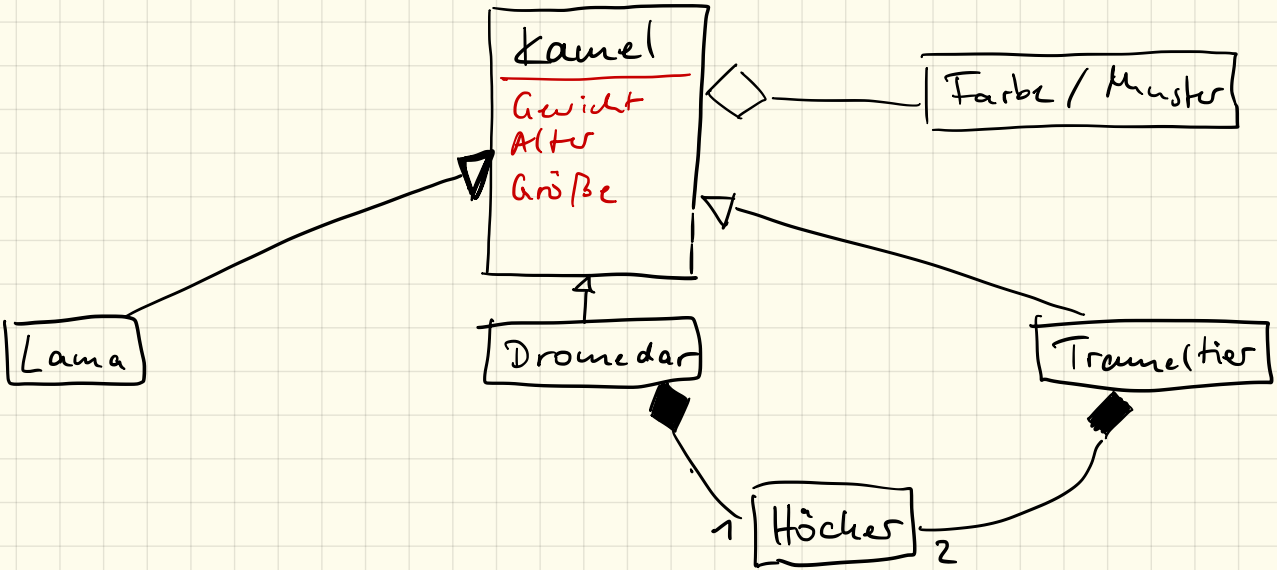


- Kamel
- Dromedar
- Trampeltier



- Lama
- Höcker
- Gewicht

- Alter
- Größe
- Farbe



Listen : ArrayList

```
ArrayList<Person> liste = new ArrayList<Person>();
```

```
Person[] liste = new Person[10]();
```

```
liste.add( new Hausmeister("Nico", "Schaser") );
```

```
liste 0 = ....
```

```
liste.size();
```

```
liste.length
```

```
System.out.print( liste.get(4) );
```

```
System.out.print( liste[4] )
```

Listen

1. Nachteile von Arrays

Um viele Dinge vom gleichen Datentyp abzuspeichern, haben wir bisher Arrays benutzt. Arrays haben aber einige Nachteile:

- Arrays sind beschränkt, d. h. nachdem ein Array einmal mit einer festen Länge initialisiert wurde kann diese nachträglich nicht mehr geändert werden um beispielsweise weitere Elemente hinzufügen zu können.
- Als Index sind nur fortlaufende Zahlen möglich.
- Bei einer Suche müssen entweder die Einträge davor sortiert werden oder alle Einträge durchsucht werden.
- Ein Element zwischendrin einfügen oder löschen und damit alle nachfolgenden Elemente zu verschieben funktioniert nur manuell, ebenso das Einfügen oder Löschen am Anfang des Arrays.
- Sortiert einfügen ist somit sehr umständlich.

2. `List`

Um vor allem das erste Problem zu lösen, nimmt man in Java sogenannte Listen. Die einfachste hierbei ist die `ArrayList`:

```
1 ArrayList<Integer> liste = new ArrayList<Integer>();
2
3 liste.add(5);
4 liste.add(8);
5
6 for(int i=0 ; i<liste.size() ; i++) {
7     System.out.println( liste.get(i) );
8 }
```

Listing 1: ArrayList

Eine Besonderheit dabei ist, dass diese Listen prinzipiell mit allen Datentypen funktionieren. Man muss den Datentyp, den man allerdings verwenden will explizit in den geschweiften Klammern angeben![1]
Hierbei ist zu beachtet, dass in den spitzen Klammern keine *primitiven Datentypen* (wie `int`, `float`,...) stehen dürfen, sondern die entsprechenden „vollwertigen“ Klassen (`Integer`, `Float`,...)
Zur Liste kann man dann mit der Methode `add()` ein weiteres Element hinzufügen.
Die Methode `size()` gibt die Länge der Liste an, mit `get(i)` kann man das Element am Index `i` abrufen (äquivalent beim Array: `[i]`)

[1] Anmerkung: das nennt sich *Generics*

3. Aufgabe: Schule

Erweitere deine Schule von letztem Mal um eine Klasse `Schule`. Diese soll eine Liste beinhalten mit allen Personen. Dazu benötigt man auch eine Methode `addPerson`, um neue Personen hinzuzufügen. Außerdem eine Methode `Ausgabe()`, die alle Personen auf der Konsole ausgibt.

a) Zeichne zuerst das Klassendiagramm (inklusive der Klassen aus letzter Stunde!)

b) Erstelle in der `Main`-Klasse[2] in der `main`-Methode eine Instanz der Schule und füge anschließend neue Personen über die Methode der Schule hinzu.

c) Rufe anschließend die `Ausgabe`-Methode der Schule auf.

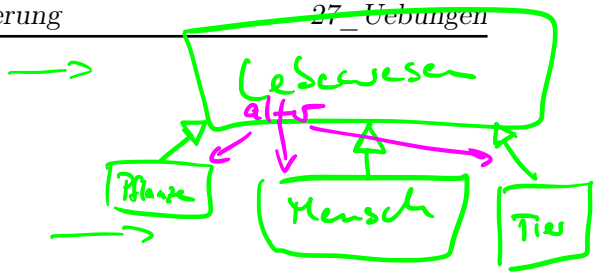
```
1  Schule dhg = new Schule("Droste-Hülshoff-Gymnasium");
2
3  Schüler as = new Schüler("Aaron","Sommer","KS1");
4  dhg.addPerson( as );
5
6  Person mc = new Schüler("Milena","Cordes","KS1");
7  dhg.addPerson( mc );
8
9  dhg.addPerson( new Lehrer("Alexander","Kimmig","10b,KS1,KS2") );
10 dhg.addPerson( new Hausmeister("Bruno","del Core") );
11 dhg.addPerson( new Schulleiter("Stefan","Maier","KS2") );
12
13 dhg.Ausgabe();
```

Listing 2: Beispielcode

d) Im Beispiel siehst du 3 Möglichkeiten, Personen hinzuzufügen. Nenne deren Unterschiede und beschreibe, welche Vor- und Nachteile diese haben.

[2] oder `Test`-Klasse, je nachdem wie du die genannt hast

Übungen zur Klausur



1. Begriffe der Objektorientierung

Beschreibe kurz folgende Begriffe (und ggf. deren Verwendung) im Zusammenhang der Objektorientierung:

- Objekt
- Klasse Definiert Attribute + Methoden, Vorlage für Objekte
- Instanz Eindeutig identifizierbares Element, ein konkretes Ding einer Klasse
- Vererbung Kindklasse erweitert Basisklasse
- Konstruktor Methode um Attribute des Objekts zu initialisieren, wird beim Anlegen einer Instanz aufgerufen
- `public`, `private`, `protected` innerhalb der Vererbungskette verwendbar
- Polymorphismus bedeutet, dass es unterschiedliche Methoden mit gleichem Namen geben kann (z.B. bei Vererbung)

überall verwendbar

nur innerhalb der Klasse verwendbar

2. Beziehungen

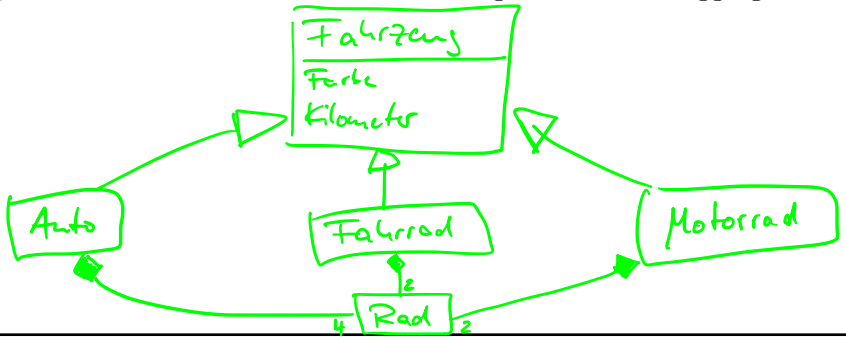
Gib an, ob es sich bei diesen Beziehungen um eine Instanz, eine Vererbung oder um eine Aggregation handelt:

ist ist Art hat

- Buch - Seite
- Buch - Medium
- Intel Pentium - Prozessor
- Monitor - Ausgabegerät
- Dackel - Hund
- J. S. Bach - Komponist
- Freiheitsstatue - Wahrzeichen
- Mensch - Trockennasenaffe
- Tee - Getränk

3. Klassendiagram

Zeichne ein Klassendiagramm zu den folgenden Begriffen: ~~Fahrzeug~~, Fahrrad, Motorrad, ~~Auto~~, ~~Rad~~, ~~Kilometerstand~~, ~~Farbe~~. Verwende sowohl die Vererbung als auch die Aggregation.



17.12.18

```
FileReader fr = new FileReader("src/ab31/input.txt");
BufferedReader br = new BufferedReader(fr);

String zeile = br.readLine();

while ( zeile != null ){
    (.....)
    zeile = br.readLine();
}
```

```
class Person {
    public static String name;

}

class Main {
    main() {
        Person.name = "Aaron";
        Person.name = "Patrick";
    }
}
```

Dateien lesen

FileReader und BufferedReader

Eine Möglichkeit, Dateien mit Java einzulesen, bietet der `FileReader` in Kombination mit dem `BufferedReader`. Hierzu müss die entsprechenden Funktionen mit `import java.io.*` eingebunden werden.

```
1 FileReader fr = new FileReader("input.txt");
2 BufferedReader br = new BufferedReader(fr);
3 String zeile = br.readLine();
```

Listing 1: Dateien lesen

Zeile 1: `FileReader` anlegen, im Konstruktor gibt man dabei den Dateinamen der einzulesenden Datei an. Diese liegt im Hauptordner des Projektes!

Zeile 2: Der `BufferedReader` dient wie der `Scanner` dazu, Texte zu lesen.[1] Hiermit wird also der Inhalt der Datei gelesen.

Zeile 3: Anschließend können wir mir der Methode `readLine()` des `BufferedReader`-Objektes eine Zeile einlesen.

Kann keine Zeile mehr gelesen werden, so liefert `br.readLine()` `null` zurück. Dies kann z. B. in einer `while`-Schleife als Bedingung genutzt werden.

1. Aufgabe

Erstelle eine einfache Textdatei `input.txt` mit einem **mehrzeiligen** Inhalt.
Programmiere anschließend eine `Main`-Klasse mit `main`-Methode, welche die Datei komplett liest und auf der Konsole ausgibt.
Hinweis: Es werden zwei Fehler `Unhandled exception type FileNotFoundException` und `Unhandled exception type IOException`. Löse diese beiden Fehler indem du auf `Add throws declaration` klickst.

2. Aufgabe

Erweitere das Programm so, dass zunächst alle Zeilen in einen einzigen `String` gespeichert werden, dass dass du mit nur einem Aufruf von `System.out.println()` die komplette Datei auf der Konsole ausgeben lassen kannst.
Was fällt dir dabei auf?

[1] Dateien könnte man auch mit dem `Scanner` lesen, ein `BufferedReader` arbeitet aber etwas effizienter.

static

Oft braucht man Methoden, die nicht an ein spezielles Objekt gekoppelt sind. Diese haben wir bisher als `static`-Methoden direkt in die Hauptklasse unter die `main`-Methode programmiert.
Zu besseren Strukturierung können wir diese ebenfalls in Klassen auslagern um das Programm übersichtlicher zu gestalten. Eine solche Klasse ist in der Hinsicht keine Vorlage für ein Objekt sondern dient lediglich als Hilfsklasse.
Um ein Methode aus dieser Klasse aufzurufen benötigt man auch keine Instanz! Man kann diese `static`-Methoden direkt über den Klassennamen aufrufen:

```
1 class Beispiel {
2     public static void Methode() {
3         [...]
4     }
5 }
```

Listing 2: Klasse Beispiel

```
1 class Main {
2     public static void main(String[] args) {
3         Beispiel.Methode();
4     }
5 }
```

Listing 3: Klasse Main

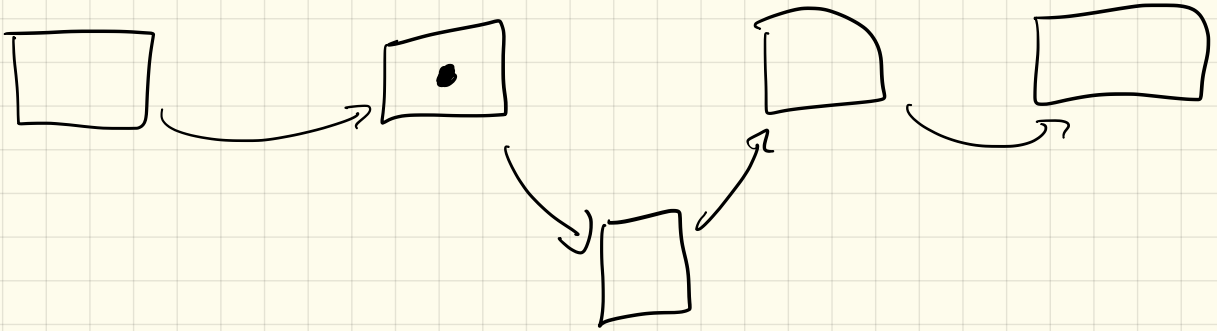
3. Aufgabe

Programmiere eine Klasse `Reader` mit einer `static`-Methode `readFile(String filename)`. Diese soll als Parameter den Dateinamen bekommen und den Inhalt der Datei als String zurückgeben.
Rufe dann aus der `main`-Methode die Methode auf:

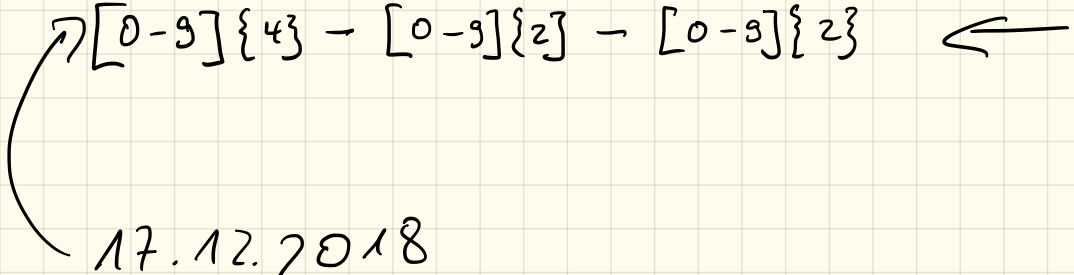
```
1 public static void main(String[] args) {
2     String datei = Reader.readFile("input.txt");
3     System.out.println(datei);
4 }
```

Listing 4: Klasse Main

Hinweis: Löse die Fehler der nichtbehandelten Exceptions jeweils mit `Add throws declaration`.



2018-12-17



$$(x^3 - 3x^2 - 10x + 24) : (x - 2) = x^2 - x - 12$$

$$-(x^3 - 2x^2)$$

$$-1x^2 - 10x + 24$$

$$-(-x^2 + 2x)$$

$$-12x + 24$$

$$-(-12x + 24)$$

$$0$$

19.12.18

HashMap

a	5
b	10
c	3
d	7
e	19
g	1

Index
Key

Wert
Value

ArrayList

0	5
1	10
2	3
3	7
4	19
5	1
6	6

Index

Wert

```
ArrayList<Integer> name = new ArrayList<>();
```

```
HashMap<Character, Integer> liste = new HashMap<>();
```

```
HashMap<Character, Integer> liste = new HashMap<>();
liste.put('a', 5);
liste.put('a', 7);
System.out.println(liste.get('a'));

for(Map.Entry e : liste.entrySet()) {
        e.getKey();
        e.getValue();
}
```

Variable

```
String text = Reader.readFile("src/a531/input.txt");
HashMap<Character, Integer> zeichen = new HashMap<>();

for (int i=0; i< text.length(); i++) {
    char c = text.charAt(i);
    if( zeichen.get(c) == null ) {
        zeichen.put(c, 0);
    }

    zeichen.put(c, zeichen.get(c) + 1);
}
```

HashMap

Einführung

Eine `HashMap` kann unter Java mit einer `ArrayList` verglichen werden. Jedoch mit einem wichtigen Unterschied: Die „Schubladen/Fächer“, die in einer `ArrayList` (wie auch bei einem statischen Array) fortlaufend von 0 ab durchnummeriert sind, können bei einer `HashMap` beliebig benannt werden. Das bedeutet, der *Index* muss keine Zahl sein, sondern kann irgendein Objekt sein, beispielsweise Buchstaben.
Hinweis: hierfür muss das Paket `import java.util.HashMap;` eingebunden werden!

```
1 HashMap<Character, Integer> zaehler = new HashMap<>();
2 zaehler.put('a',5);
3 System.out.println( "a: " + zaehler.get('a') );
```

Listing 1: eine HashMap anlegen

Hierbei werden in den spitzigen Klammern 2 Datentypen angegeben: der erste Datentyp gibt an, womit wir unsere Schubladen bezeichnen wollen. Der zweite Datentyp gibt – wie bei einer `ArrayList` – an, welche Daten wir in den Schubladen speichern wollen. Im Beispiel wird eine `HashMap` angelegt, deren Index ein einzelner Buchstabe (`Character`) ist, und in die wir Zahlen ablegen können.
Ähnlich wie bei einer `ArrayList` können wir mit der Methode `get('a')` den Wert abrufen (s. Zeile 3), der im Fach mit dem Index *a* abgelegt wurde. Das Ablegen funktioniert mit der Methode `put('a',5)`;
Die Abfrage *aller* Einträge ist mit einer `HashMap` allerdings etwas komplizierter, da wir nicht mehr einfach durchnummerierte Fächer haben.
Hinweis: hierfür muss das Paket `import java.util.Map;` eingebunden werden!

```
1 for(Map.Entry e : zaehler.entrySet()) {
2     System.out.println(e.getKey() +": " + e.getValue());
3 }
```

Listing 2: eine HashMap durchlaufen

Mit einer sogenannten „for-each“-Schleife können wir jeden Eintrag durchgehen. Hierbei wird bei jedem Schleifendurchlauf der nächste verfügbare Eintrag in der Variablen `e` vom Datentyp `Map.Entry` abgespeichert. Mit `e.getKey()` kann man dann den Index (*engl. key*), also die „Beschriftung“ des Faches abfragen, mit `e.getValue()` bekommen wir den im Fach gespeicherten Wert.

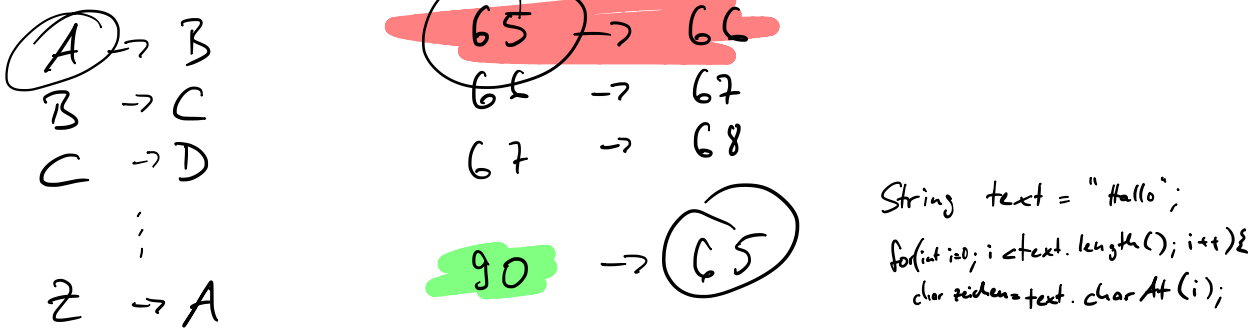
1. Aufgabe

Erstelle ein neues Paket `ab32` mit einer Klasse `Zaehlen` (inklusive `main`-Methode).
Lasse dann mithilfe der Methode von letztem Arbeitsblatt eine Textdatei einlesen, gehe dann den Text mit einer `for`-Schleife durch und zähle, wie oft jedes Zeichen in dem Text vokommt. Lasse anschließend die Häufigkeit aller Zeichen auf der Konsole ausgeben.
Hinweis: du kannst die `Reader.readFile()` Methode von letztem Mal benutzen, indem du die Klasse mit `import ab31.Reader;` einbindest!

2. Aufgabe

Programmiere eine einfache „rot13“-Verschlüsselung: hierbei sollen die Buchstaben jeweils um 13 Stellen „verschoben“ werden, d. h. aus `a` wird ein `n`, aus `b` wird `o` usw.
Betrachte hierzu nochmals die ASCII-Tabelle. Verschoben werden sollen dabei nur die Buchstaben (d. h. ASCII-Werte 65 bis 90 für die Großbuchstaben, sowie 97 bis 122 für die Kleinbuchstaben)
Für die Verschiebung musst du also zum ASCII-Wert 13 hinzuaddieren und das Ergebnis bei Bedarf (d. h. wenn der Wert über 90 ist) an das „untere“ Ende des Bereiches setzen, so dass aus einem `u` ein `h` wird.

- Programmiere hierfür eine `static`-Methode `verschluesseln(String s)` welche den übergebenen String „verschlüsselt“ und wieder zurückgibt.
- Ergänze anschließend die Methode um einen weiteren Parameter `int weite`, welcher angibt, wie weit die Buchstaben verschoben werden sollen (also anstatt fest um immer 13 Positionen zu verschieben).
- Programmiere zusätzlich eine Methode `entschluesseln(String s,int weite)`



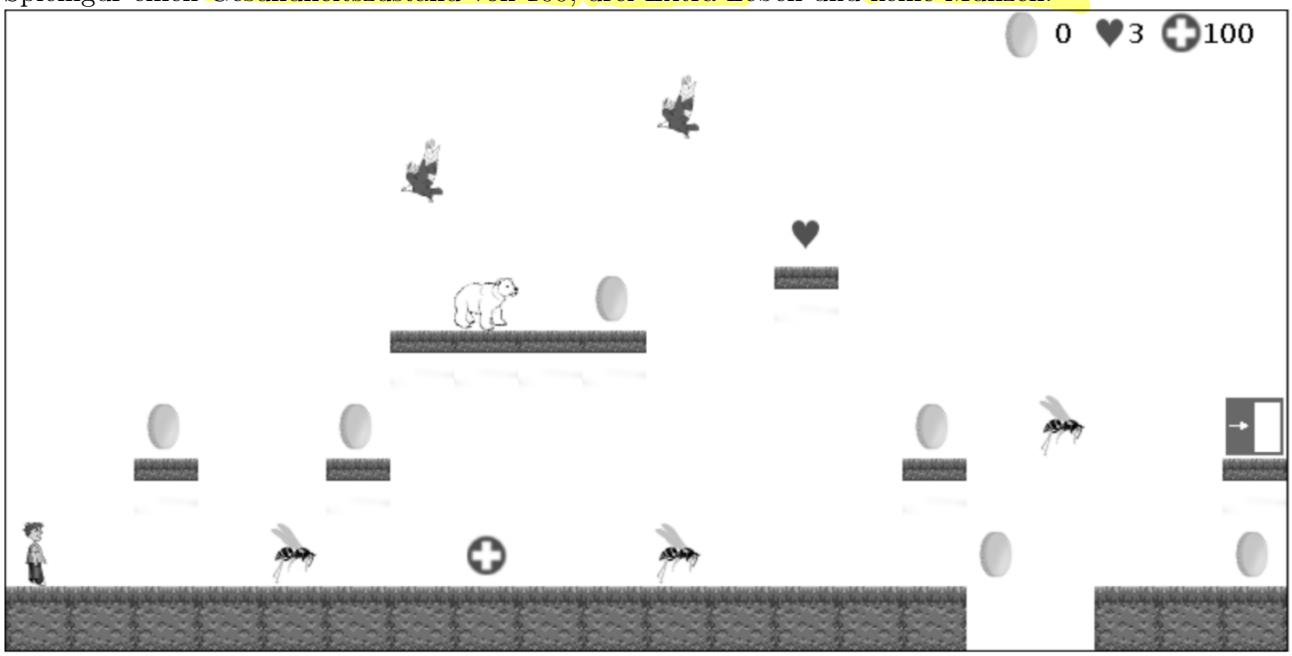

```
3
char zeichen = 'A';
int zeichenASCII = (int) zeichen;        // = 65
if(zeichenASCII >= 65 && zeichenASCII <= 90) {
  int neuASCII = zeichenASCII + 1;
  if(neuASCII > 90) neuASCII = neuASCII - 26;
  char neu = (char) neuASCII;            // = 'B'
}
```

7.1.19

Klausur 12.12.2018

Name: _____ VP: _____/20P NP: _____ mündlich: _____

Die Firma „games on demand“ bekommt den Auftrag, ein Jump'n'Run-Spiel zu entwickeln.
Hier ein Auszug aus dem Pflichtenheft:
Die Spielfigur kann per Tastatursteuerung durch eine 2D-Welt geführt werden. Darin trifft sie auf unterschiedliche Gegner: Wespen, Adler und Eisbären. In einer späteren Version des Spiels sollen weitere Gegner dazu kommen. Laufgegner wie die Eisbären sind wie die Spielfigur an den Boden gebunden und fallen beispielsweise von Terrassen herunter, wenn sie über deren Rand hinauslaufen. Fluggegner hingegen können sich durch die Luft bewegen. Darüber hinaus zeigen die Gegner weitere unterschiedliche Verhaltensweisen.
In der Welt findet die Spielfigur Münzen, Erste-Hilfe-Pakete und Leben. Zu Beginn des Spiels hat die Spielfigur einen Gesundheitszustand von 100, drei Extra-Leben und keine Münzen.



1. Aufgabe 1 (7P)

Es wird entschieden, die einzelnen Arten der Gegner als Unterklasse einer Klasse Gegner zu implementieren.

a) Stelle für diesen Fall die Beziehungen zwischen den Klassen `Gegner`, `Wespe`, `Adler`, `Eisbaer`, `Laufgegner` und `Fluggegner` in einem Klassendiagramm dar. Die Klassen sollen nur den Klassennamen, keine Attribute und Methoden enthalten.

b) Nenne Gründe, die in diesem Fall für die Modellierung mit Hilfe der Vererbung sprechen, und erläutere in diesem Zusammenhang das Grundprinzip der Vererbung.

Spieljur
private leben: int
private münzen: int
private gesund: int

c) Entwirf ein erstes UML-Klassendiagramm für die Klasse `Spielfigur` unter Berücksichtigung des Geheimnisprinzips (d. h. `public` ist so weit es geht zu vermeiden). Es ist ausreichend, die Methoden für die Initialisierung und den lesenden Zugriff auf die Attribute anzugeben.

d) Implementiere die Konstruktormethode für diese Klasse `Spielfigur`.

2. Aufgabe 2 (8P)

Eisbären besitzen eine Angriffsstärke von 10, Adler von 3 und Wespen von 1. Diese sollen über eine Methode `int getAngriffsstaerke()` ausgelesen werden.

a) Erläutere exemplarisch, wie die Klassen `Gegner` und `Eisbaer` angepasst werden müssen, damit bei jedem Gegner die korrekte Angriffsstärke ausgelesen werden kann.

Wenn die Spielfigur ein Erste-Hilfe-Paket findet, wird die Methode `gesundheitAnpassen(int a)` aufgerufen. In diesem Fall ist der übergebene Wert `a` positiv. Beim Zusammentreffen mit einem Gegner ist `a` negativ.
Beim Anpassen des Gesundheitszustands sind folgende Bedingungen zu beachten: Der Gesundheitszustand kann nicht geringer als 1 werden. Fällt er auf 0 oder weniger, verliert die Spielfigur ein Leben und der Gesundheitszustand wird wieder auf 100 gesetzt. Wenn die Spielfigur keine Extra-Leben mehr hat und ihre Gesundheit auf 0 fällt, wird die Methode `showMessage("Game over")` aufgerufen und das Spiel damit beendet. Andererseits kann der Gesundheitszustand nicht größer als 100 werden.

b) Implementiere diese Methode `gesundheitAnpassen(int a)`.

Wenn die Spielfigur auf einen Gegner trifft, hat dies Auswirkungen auf ihre Gesundheit und gegebenenfalls auf ihre Leben. Die Auswirkung berechnet sich als Produkt der Angriffsstärke des Gegners mit einer ganzzahligen Zufallszahl von 1 bis 10. Dieses Produkt wird dividiert durch einen Wert, der sich am Gesundheitszustand orientiert:

- Ist der Gesundheitszustand größer oder gleich 70 beträgt der Wert 3
- ist der Gesundheitszustand größer oder gleich 40 und kleiner als 70 beträgt er 2
- bei einem Gesundheitszustand von weniger als 40 beträgt der Wert 1

c) Implementiere die Methode `gegnerTreffen(Gegner g)`, die die Auswirkungen eines Aufeinandertreffens berechnet und dann an die Methode `gesundheitAnpassen(int a)` weitergibt.

Hinweis: Verwende als Hilfsmethode `int getZufallszahl(int min, int max)`, die eine ganzzahlige Zufallszahl `z` mit `min`≤`z`≤`max` liefert.

```
int faktor = 1;
if ( this.gesundheit >= 70 )
    faktor = 3;
else if ( this.gesundheit >= 40 )
    faktor = 2;

int z = getZufallszahl( 1, 10 );
int a = g.getAngriffsstaerke();

gesundheitAnpassen( -z * a / faktor );
```

3. Aufgabe 3 (5P)

Die Startzustände der einzelnen Levels des Spiels sind in Textdateien hinterlegt. Diese haben eine vorgegebene Struktur und bestehen aus 10 Zeilen mit jeweils 20 Zeichen.

```
00: ····E·············
01: ··········A·····
02: ·····A···········
03: ···········L·····
04: ·····E·M··B·····
05: ·····BBBB·····
06: ··M··M·····M·W··X
07: ··B··B·····B····B
08: S···W··H··W····M···M
09: BBBBBBBBBBBBBBBE·BBB
```

Abbildung 1: Beispiel-Textdatei für Level 1

Als Zeichen dienen das Leerzeichen (im Bild als · dargestellt) und die Buchstaben

A dler	**L** eben	**S** pielfigur
W espe	**M** ünze	**B** oden oder Terrasse
E isbär	**H** ilfe-Paket	**X** Ausgang

Mit Hilfe einer hier nicht zu implementierenden Methode `levelLaden(String dateiName)` wird die Textdatei in ein zweidimensionales Zeichen-Array `level` eingelesen und für das weitere Spiel zur Verfügung gestellt.

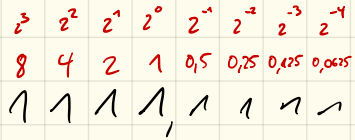
Hinweis: es soll hierbei ein statisches Array verwendet werden.

Der Spieler kann mit einem gewöhnlichen Text-Editor weitere eigene Levels erstellen. Beim Laden eines Levels wird geprüft, ob folgende Mindestbedingungen erfüllt sind:

- das Level enthält genau eine Spielfigur
- das Level enthält mindestens einen Ausgang
- die Spielfigur und Eisbären stehen auf einem festen Untergrund

a) Deklariere und initialisiere das zweidimensionale `char`-Array `level`, das Platz für 10 Zeilen mit jeweils 20 Zeichen bietet.

b) Implementiere eine Methode `boolean pruefeLevel()`, die den Inhalt des Zeichen-Arrays `level` überprüft und genau dann `true` zurückliefert, wenn die Mindestbedingungen erfüllt sind.



$0{,}75_{10} = 0{,}11_2$

Java Überblick 1/2

Datentypen

Typ	Beschreibung	Wertebereich / Beispiel
boolean	Boolescher Wert	`true, false`
char	einzelnes Zeichen (16 Bit)	alle Unicode-Zeichen
byte	ganze Zahl (8 Bit)	$-2^7 \dots 2^7 - 1$
short	ganze Zahl (16 Bit)	$-2^{15} \dots 2^{15} - 1$
int	ganze Zahl (32 Bit)	$-2^{31} \dots 2^{31} - 1$
long	ganze Zahl (64 Bit)	$-2^{63} \dots 2^{63} - 1$
float	Fließkommazahl (32 Bit)	3,14159f
double	Fließkommazahl (64 Bit)	$-1{,}79*10^{38}$
`String`	Zeichenkette	"Dies ist ein String."
int[]	ganzzahliges Feld (Array)	{3, 1, 4, 1, 5, 9}

Java kennt **primitive Typen** (boolean, char, , double) und **Referenztypen** (Objekte, Strings und Arrays).

Variablendeklaration

`(<Zugriffsart>) <Typ> <Bezeichner> (= <Wert>)`

Beispiel	Erläuterung
`private int anzahl;`	Ganzzahlige Variable mit Namen *anzahl*
`int tage = 14;`	Ganzzahlige Variable mit Bezeichner *tage* und Zuweisung des Werts 14
`boolean gesund;`	Boolesche Variable mit Bezeichner *gesund*
`public String name;`	(Öffentlich zugängliche) Text-Variable mit dem Bezeichner *name*

Referenztypen (außer `String`) müssen mithilfe des `new`-Operators erzeugt werden:

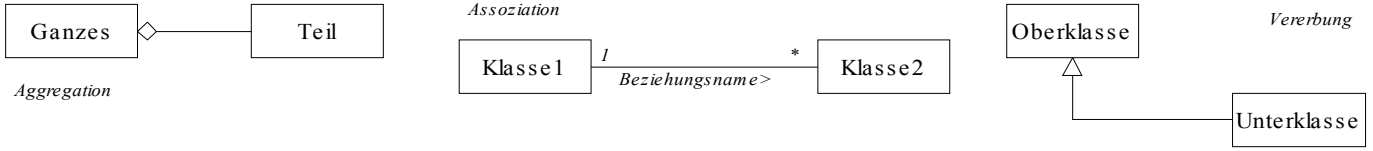
`int[] du = new int[5];`
Es wird ein leeres Feld mit dem Bezeichner *du* erzeugt, welches fünf ganzzahlige Werte aufnehmen kann.

`double[] messungen;`
Deklaration eines Feldes mit Namen *messungen*. Bevor es jedoch Werte aufnehmen kann, muss es mit dem `new`-Operator erzeugt werden.

Operatoren

+	Addition		
-	Subtraktion		
*	Multiplikation		
/	Division		*
%	Modulo		Divisonsrest
++	Inkrement		i++ entspr. i = i+1
--	Dekrement		i = i – 1
==	Vergleich	=	Zuweisung**
<	Kleiner	<=	Kleiner gleich
>	Größer	>=	Größer gleich
!=	Ungleich	!	logisches NICHT
||	logisches ODER	&&	logisches UND

* x / y ergibt den Quotienten von x und y. Sind x und y ganzzahlig, so ist auch x / y ganzzahlig (z.B. 9 / 4 liefert 2).
** a == b liefert `true` oder `false`



Methodendefinition

`(<Zugriffsart>) <Rückgabetyp> <Bezeichner> (<Parameter>) {…}`

Beispiel

```
public void hello(String name){
       System.out.print("Hallo " + name);
}

public double umfang(double radius){
       return 2*radius*3,14159;
}

public void zubettgehen(){
       ausziehen();
       waschen();
       zaehneputzen();
       schlafenlegen();
}
```

Erläuterung

Die öffentliche Methode *hello* gibt auf dem Bildschirm „Hallo XYZ" aus, wenn ihr „XYZ" beim Aufruf übergeben wurde.

Die Methode gibt den Kreisumfang bei Übergabe des Radius zurück.

Die Methode *zubettgehen* hat keinen Rückgabewert, keine Parameter und ruft nacheinander die Methoden *ausziehen, waschen, zaehneputzen* und *schlafenlegen* auf.

Klassendefinition

`(<Zugriffsart>) class <Bezeichner> (extends <Oberklasse>) {…}`

Beispiel

```
public class Quadrat{

   //Attribute
   private int laenge;
   private String farbe;


   //Methoden
   Quadrat(int seitenl){
       laenge = seitenl ;
       farbe = "rot";
   }

   public double flaeche(){
       return laenge*laenge;
   }

} //Ende Quadrat
```

Erläuterung

Kopf: **bei Vererbung** class Unterkl **extends** Oberkl, z.B. `class Quadrat extends Figur`
Deklaration der Attribute
Es werden ein ganzzahliges Attribut für die Seitenlänge sowie eine Text-Variable für die Füllffarbe des Quadrates deklariert.

Methodendefinitionen
Der Konstruktor zur Erzeugung des Objekts hat den gleichen Namen wie die Klasse selbst.

Methode, die als Rückgabewert den Flächeninhalt des Quadrats berechnet.

Ende der Klassendefinition

Java Überblick 2/2

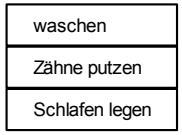
Sequenz

Jede Anweisung wird mit einem Semikolon abgeschlossen;
Mehrere Anweisungen nacheinander ergeben eine Sequenz

Beispiel

```
waschen();
zaehneputzen();
schlafenlegen();
```

Struktogramm



Fallunterscheidung (bedingte Anweisung)

Die bedingte Anweisung gibt es mit oder ohne Alternative:

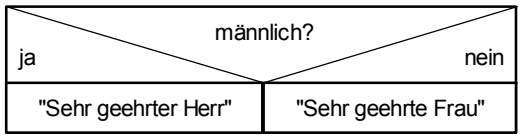
```
if (<Bedingung>) {
   <Anweisungen>
}
else {
   <Anweisungen>
}
```

```
if (<Bedingung>) {
   <Anweisungen>
}
```

Beispiel (mit Alternative):

```
if (geschl == 'm') {
   System.out.println("Sehr geehrter Herr");
}
else {
   System.out.println("Sehr geehrte Frau");
}
```

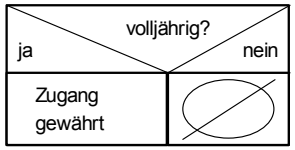
Struktogramm (mit Alternative):



Beispiel (ohne Alternative):

```
if (alter >= 18) {
   zugang = true;
}
```

Struktogramm (ohne Alternative):



Wiederholung mit fester Anzahl

```
for (<Init>; <Bedingung>; <Update>) {
      <Anweisungen>
}
```

`<Init>`	Deklaration einer ganzzahligen Zählvariablen und Zuweisung ihres Anfangswertes (z.B. `int i = 0`).
`<Bedingung>`	Solange die Bedingung (abhängig von der Zählvariablen) erfüllt ist, werden nachfolgende Anweisungen ausgeführt.
`<Update>`	Das Update erfolgt nach jedem Durchlauf und ändert die Laufvariable entsprechend der angegebenen Zuweisung (oft `i++`).

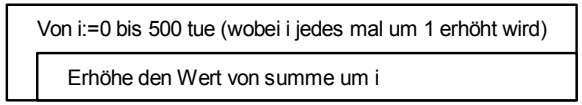
Beispiel

```
int summe = 0;

for (int i = 0; i <= 500; i++) {
      summe = summe + i;
}
```

Struktogramm und Erläuterung

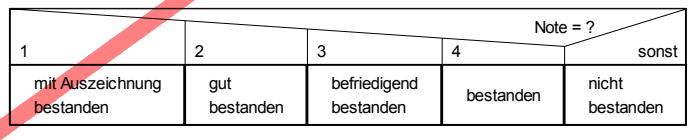
Berechnet die Summe aller ganzen Zahlen von 0 bis 500. Als Zählvariable wird die ganze Zahl *i* deklariert, ihr Anfangswert ist 0. Die Anweisung wird solange wiederholt, bis *i* den Wert 500 erreicht, wobei *i* bei jedem Durchlauf um 1 erhöht wird.



Mehrfachauswahl

Die `switch`-Anweisung kann beliebig viele Fälle untersuchen
Die zu überprüfende Variable muss vom Typ `byte`, `short`, `int` oder `char` sein.

```
switch (<Variable>) {
   case <Wert1>: <Anweisungen1>; break;
   case <Wert2>: <Anweisungen2>; break;
   ...
   default: <Anweisungen3>; break;
}
```



Wiederholung mit Anfangsbedingung

```
while (<Bedingung>) {
      <Anweisungen>
}
```

Die Bedingung wird vor der Ausführung der Anweisungen getestet, so dass nachfolgende Anweisungen eventuell gar nicht ausgeführt werden.

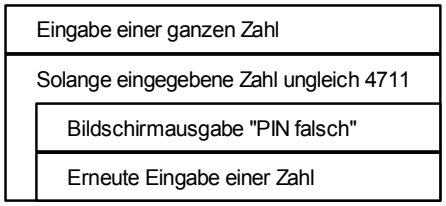
Beispiel

```
int pin = eingabe();

while (pin != 4711) {
      System.out.println("PIN falsch");
      a = eingabe();
}
```

Struktogramm und Erläuterung

Zuerst wird eine Variable pin vom Typ Integer deklariert. Die Zuweisung erfolgt über eine Methode eingabe(), welche ermöglicht, eine ganze Zahl über die Tastatur einzugeben und diese als Rückgabewert liefert. Solange pin nicht den Wert 4711 hat, wird der Fehler auf dem Bildschirm ausgegeben und zur erneuten Eingabe einer Zahl aufgefordert.



Wiederholung mit Endbedingung

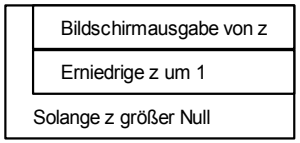
```
do {
      <Anweisungen>
} while (<Bedingung>);
```

Die Bedingung wird nach der ersten Ausführung der Anweisungen getestet, so dass die Schleife wenigstens einmal durchlaufen wird.

Beispiel

```
do {
System.out.println(z);
z--;
} while (z>0);
```

Struktogramm



$9 : 4 = 2 \text{ R } 1$

$9 / 4 = 2$

$9 \% 4 = 1$

a b c .... z

$53 - 26 = 27$

$27 - 26 = 1$

$27 \% 26 = 1$

$53 \% 26 = 1$

$7 \% 26 =$

$-33 \% 26 = -7$

a b c d e f ..... y z

Dateien ~~lesen~~ schreiben

FileWriter

Um Strings wieder in Textdateien abzuspeichern, kann man beispielsweise die Klasse `FileWriter` benutzen.

```
1 ~~FileReader~~ FileWriter fw = new FileWriter("output.txt");
2 fw.write("Dieser Text soll in die Datei abgespeichert werden!");
```

Listing 1: Dateien schreiben

Zeile 1: `FileWriter` anlegen, im Konstruktor gibt man dabei den Dateinamen der einzulesenden Datei an. Diese liegt im Hauptordner (bzw. im angegebenen Unterordner) des Projektes!

Zeile 2: Anschließend können wir mir der Methode `write()` des `FileWriter`-Objektes einen `String` in die Datei schreiben.

*Anmerkung: Wird beim Anlegen des `FileWriter`s als zweiter Parameter `true` angegeben, so wird eine eventuell vorhandene Datei **nicht** überschrieben, sondern erweitert!*

1. Aufgabe: Vorarbeit

Ändere die `verschluesseln`-Methode von letztem Mal so ab, dass die „verschobenen" Buchstaben nicht direkt auf der Konsole ausgegeben, sondern in einem `String` abgespeichert werden. Dieser String soll anschließend mit `return` zurückgegeben werden.

2. Aufgabe

Lasse wie beim letzten Mal eine Textdatei einlesen, verschlüssle diese und speichere den verschlüsselten Text mithilfe des `FileWriter`s in eine neue Textdatei ab.
Hinweis: Es werden zwei Fehler `Unhandled exception type FileNotFoundException` und `Unhandled exception type IOException`. Löse diese beiden Fehler indem du auf `Add throws declaration` klickst.

3. Aufgabe

Programmiere eine Klasse `Writer` mit einer `static`-Methode `writeFile(String filename, String text)`. Diese soll als ersten Parameter den Dateinamen bekommen und den Inhalt der Datei als als zweiten Parameter.
Rufe dann aus der `main`-Methode die Methode auf:

```
1 public static void main(String[] args) {
2     Writer.writeFile("output.txt","Dieser Text soll in die Datei");
3 }
```

Listing 2: Klasse Main

Hinweis: Löse die Fehler der nichtbehandelten Exceptions jeweils mit `Add throws declaration`.

4. Aufgabe

Lade dir auf der Homepage `http://ab.lehrer-kimmig.de` die beiden Dateien `enc1.txt` und `enc2.txt` herunter. Diese wurden mithilfe dem Cäsar-Verfahren verschlüsselt. Versuche, diese zu knacken ud die Dateien wieder zu entschlüsseln.

Huffman-Kodierung

14.1.19

a a b a b c a b c d

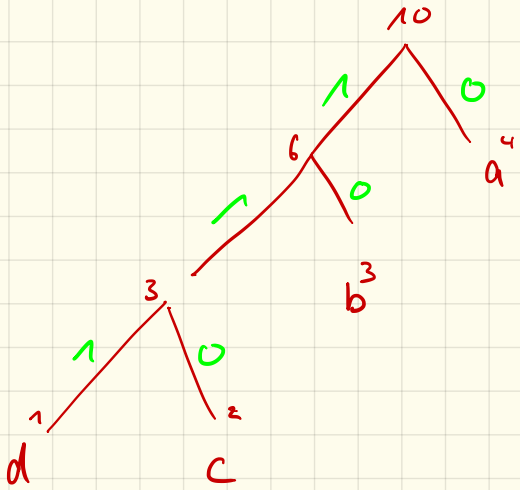
= 10 Byte oder 80 Bit

a: 4 mal 0
b: 3 mal 10
c: 2 mal 110
d: 1 mal 111

0 0 1 0 0 1 0 1 1 0 0 1 0 1 1 0 1 1 1

a a b a b c a b c d

= 19 Bit

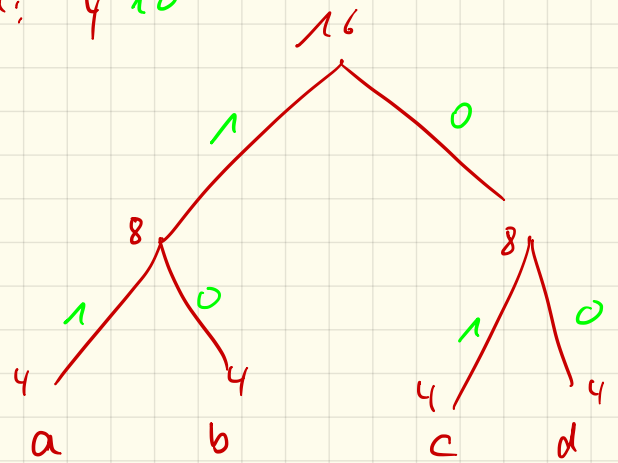


a b a b a b a b c d c d c d c d 128 Bit

a: 4 1
b: 4 0
c: 4 01
d: 4 10

11 10 11 10 11 10 11 10 01 00 01 00 01 00 01 00
a b a b a b a b c d

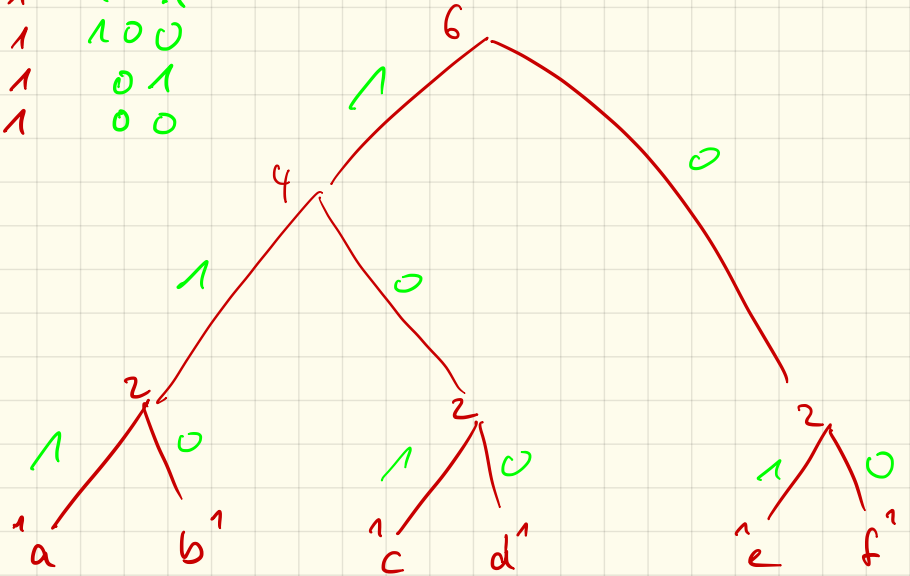
32 Bit



aaaaaaaaaabcdcfgi

a b c d e f

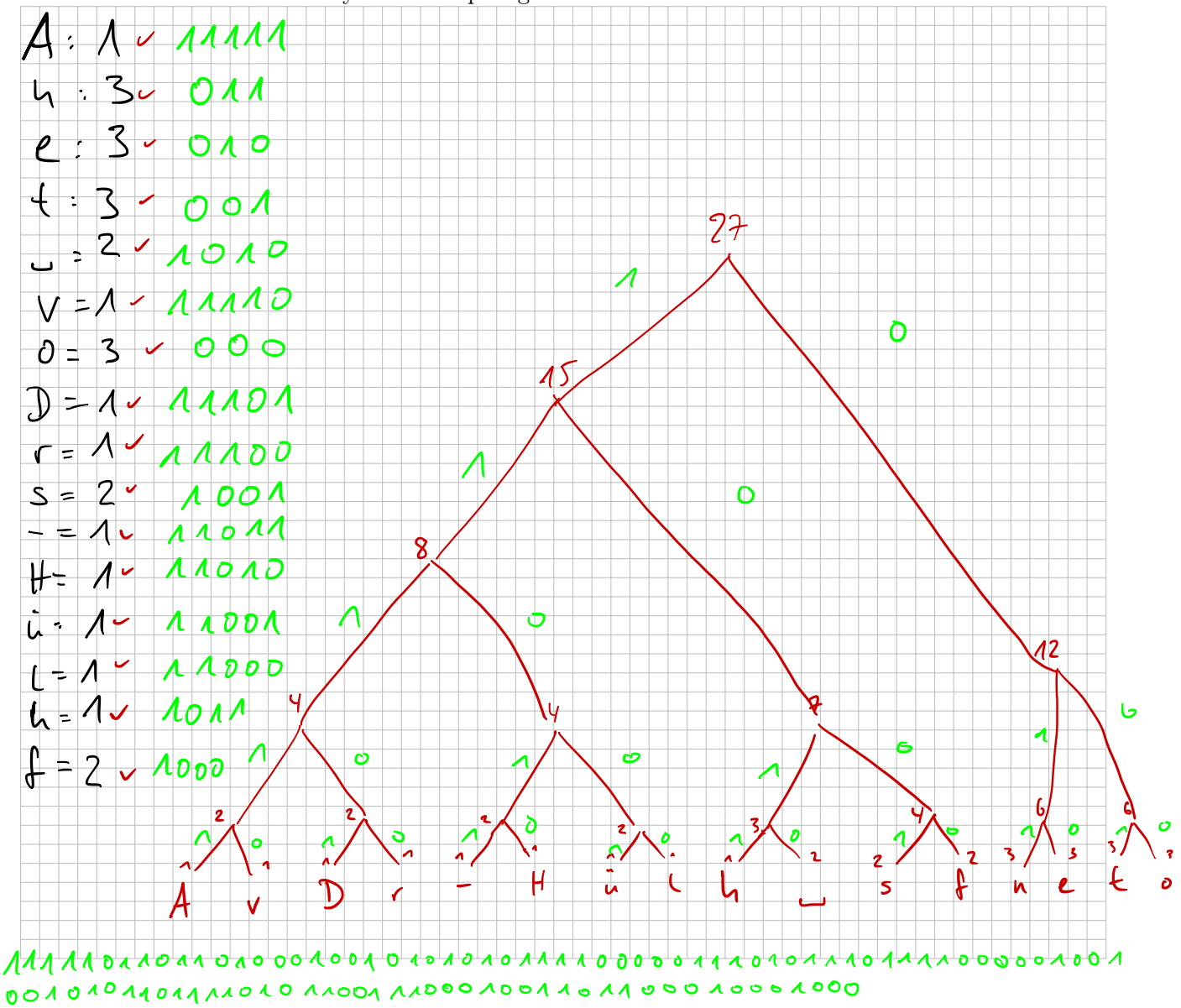
a: 1 111
b: 1 110
c: 1 101
d: 1 100
e: 1 01
f: 1 00



Huffman-Code

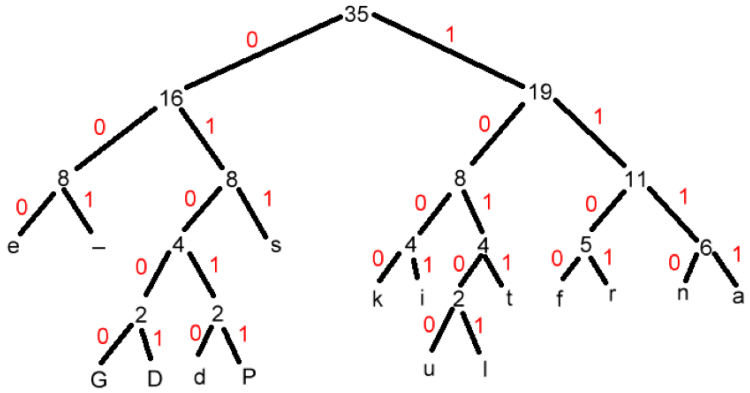
1. Codierung

Erstelle einen Huffman-Baum für **Annette von Droste-Hülshoff** und codiere diesen Text. Vergleiche anschließend die Anzahl der Bytes des ursprünglichen Textes mit denen des Huffman-Codes.



2. Decodierung

Übersetze folgenden Code. Vergleiche anschließend die Anzahl der Bytes des ursprünglichen Textes mit denen des Huffman-Codes.



01001111101100101011110000011010101000111001101100101101110110011000000100111100001110001010001010011011000000011100111111110101111101100

Das Pferd frisst keinen Gurkensalat

16.1.19

LZ77

aacaacabcabaaac

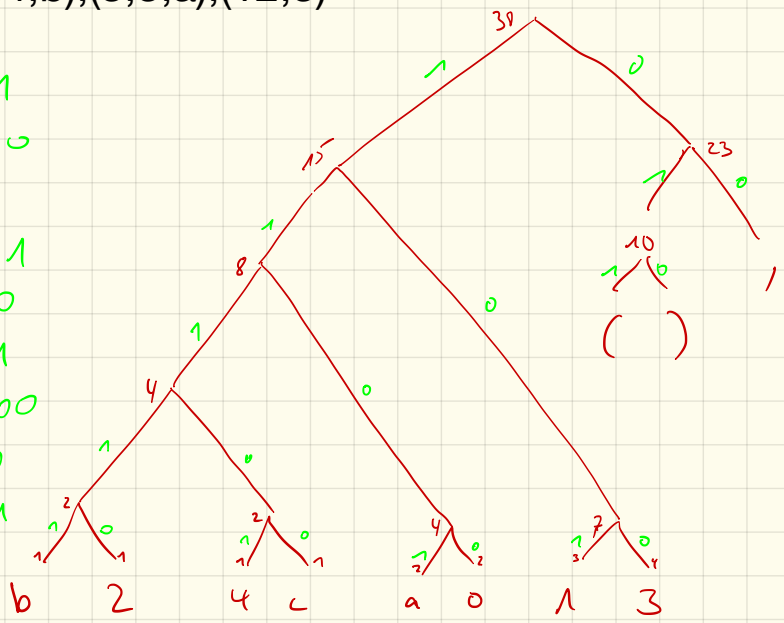
(0, 0, a), (1, 1, c), (3, 4, b), (3, 3, a), (9, 3)

(0,0,a),(1,1,c),(3,4,b),(3,3,a),(12,3)

a ac aacabcaba aac

(0,0,a),(1,1,c),(3,4,b),(3,3,a),(12,3)

(	5 mal	011
0	2 mal	1100
,	13 mal	00
a	2 mal	1101
)	5 mal	010
1	3 mal	101
c	1 mal	11100
3	4 mal	100
4	1 mal	11101
b	1 mal	11111
2	1 mal	11110

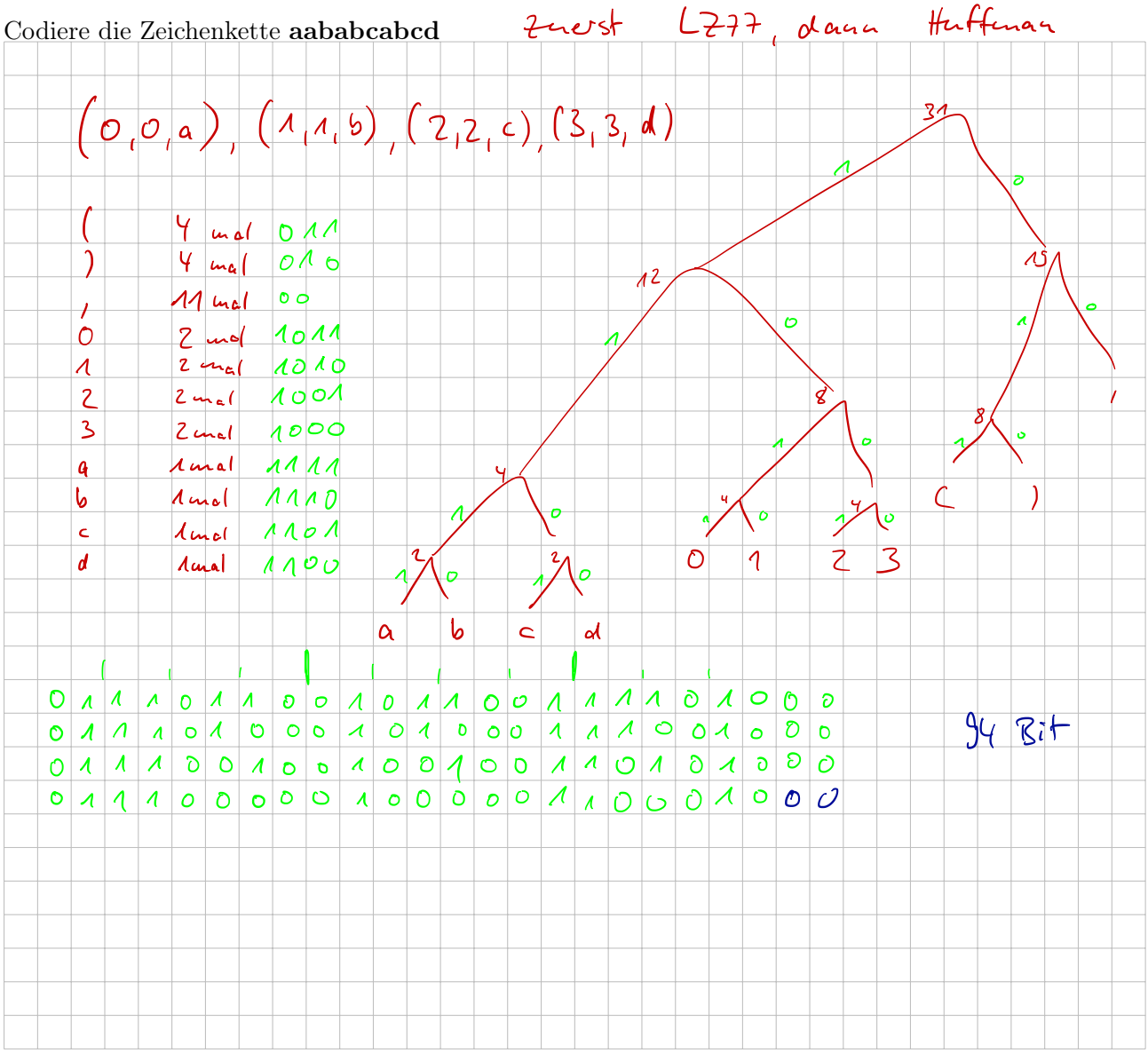


18.1.19

LZ77

1. Codierung

Codiere die Zeichenkette **aababcabcd**



2. Decodierung

Übersetze folgenden Code
(0,0,a),(0,0,b),(0,0,r),(3,1,c),(5,1,d),(7,4)

abracadabra

lol

JPG ← Fotos

PNG ← Internet / Zeichnungen

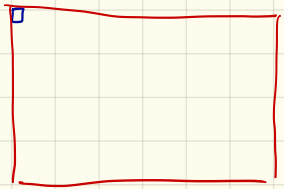
TGA ← mehrere Ebenen

BMP ← unkomprimiert

TIF ← super-RAW

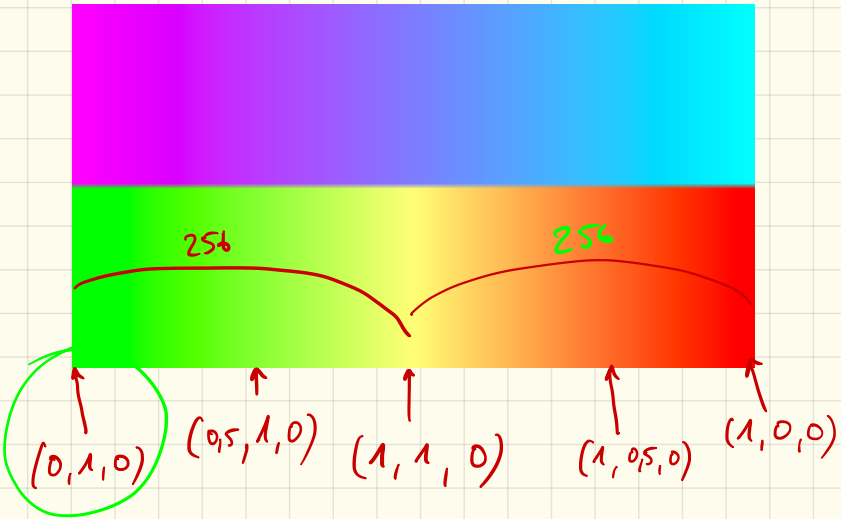
RAW ← Digitalkamera hohe Qualität

GIF



22.1.2019

PNG



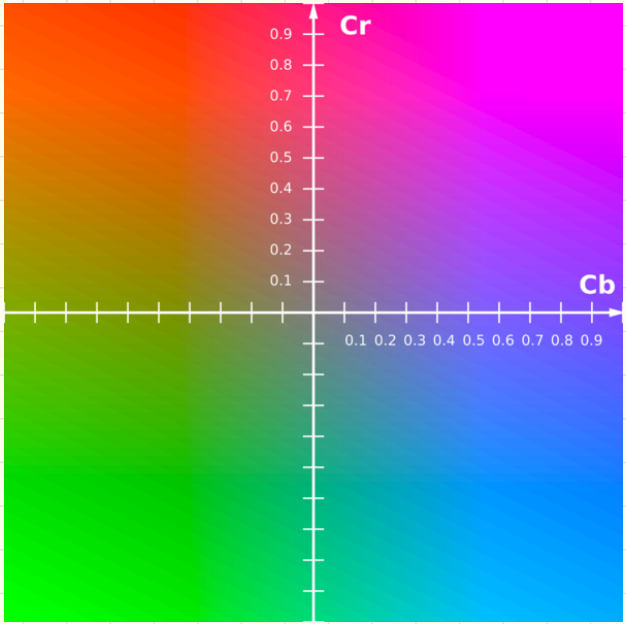
(0,255,0) (1,255,0) (2,255,0)

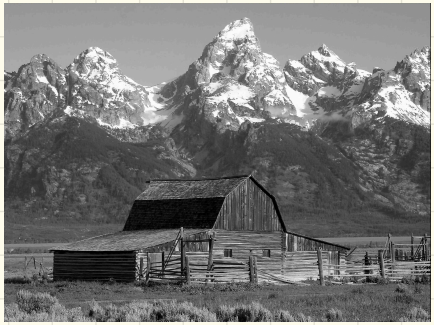
(0,255,0) (1,0,0) (1,0,0), (1,0,0)

JPG

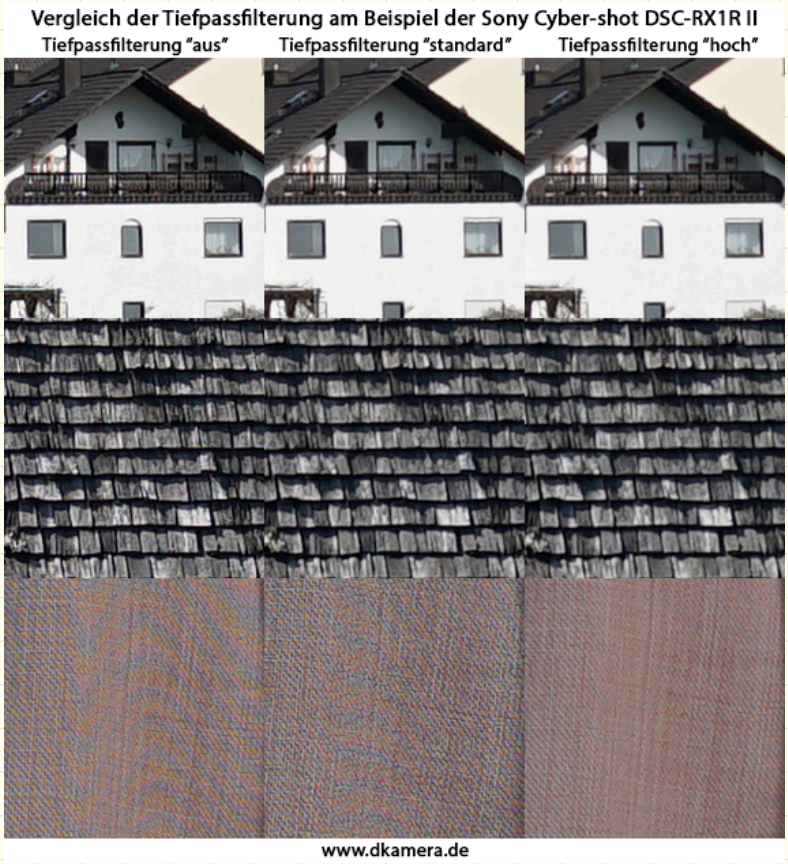
Y Cr Cb

1. Schritt: Farbumrechnung
RGB -> Y'CbCr

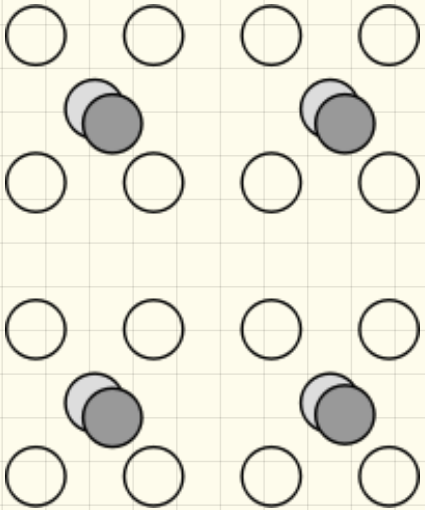




2. Schritt: Tiefpassfilterung

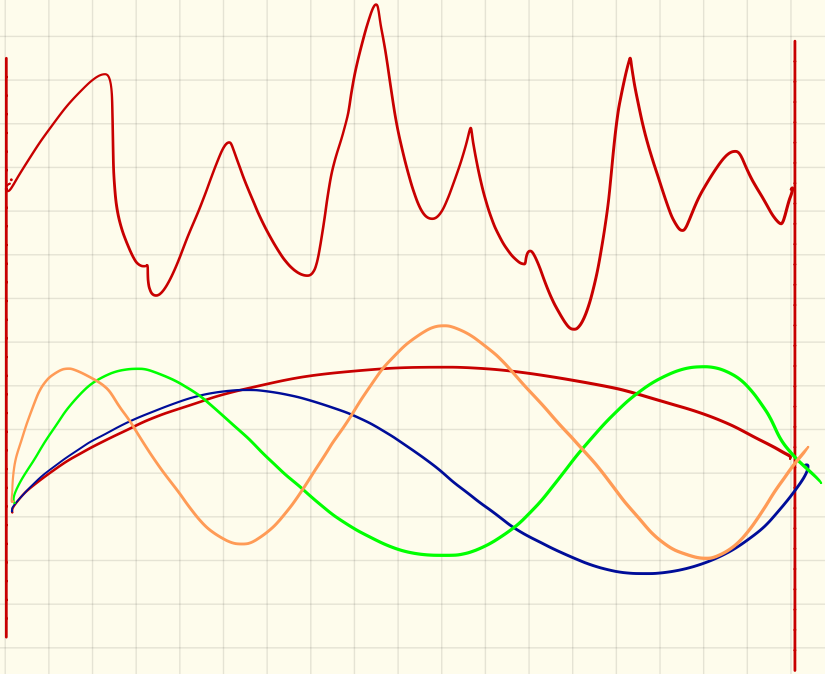


3. Schritt: Unterabtastung

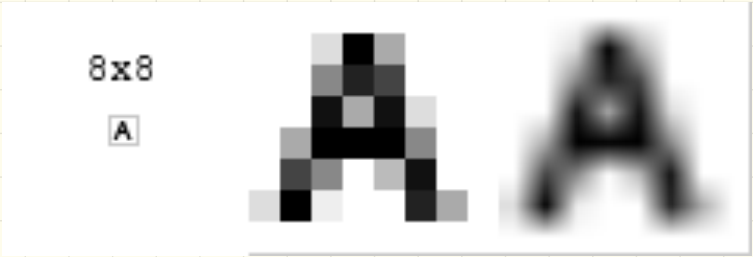


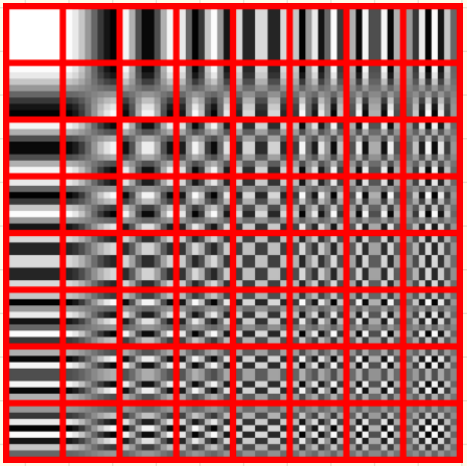
4. Schritt: diskrete Kosinustransformation

$$X_k = \frac{1}{2}(x_0 + (-1)^k x_{N-1}) + \sum_{n=1}^{N-2} x_n \cos\left[\frac{\pi}{N-1} nk\right] \qquad k = 0, \ldots, N-1$$



4. Schritt: diskrete Kosinustransformation





$$\begin{bmatrix}
6{,}1917 & -0{,}3411 & 1{,}2418 & 0{,}1492 & 0{,}1583 & 0{,}2742 & -0{,}0724 & 0{,}0561 \\
0{,}2205 & 0{,}0214 & 0{,}4503 & 0{,}3947 & -0{,}7846 & -0{,}4391 & 0{,}1001 & -0{,}2554 \\
1{,}0423 & 0{,}2214 & -1{,}0017 & -0{,}2720 & 0{,}0789 & -0{,}1952 & 0{,}2801 & 0{,}4713 \\
-0{,}2340 & -0{,}0392 & -0{,}2617 & -0{,}2866 & 0{,}6351 & 0{,}3501 & -0{,}1433 & 0{,}3550 \\
0{,}2750 & 0{,}0226 & 0{,}1229 & 0{,}2183 & -0{,}2583 & -0{,}0742 & -0{,}2042 & -0{,}5906 \\
0{,}0653 & 0{,}0428 & -0{,}4721 & -0{,}2905 & 0{,}4745 & 0{,}2875 & -0{,}0284 & -0{,}1311 \\
0{,}3169 & 0{,}0541 & -0{,}1033 & -0{,}0225 & -0{,}0056 & 0{,}1017 & -0{,}1650 & -0{,}1500 \\
-0{,}2970 & -0{,}0627 & 0{,}1960 & 0{,}0644 & -0{,}1136 & -0{,}1031 & 0{,}1887 & 0{,}1444
\end{bmatrix}$$

5. Schritt: Quantisierung

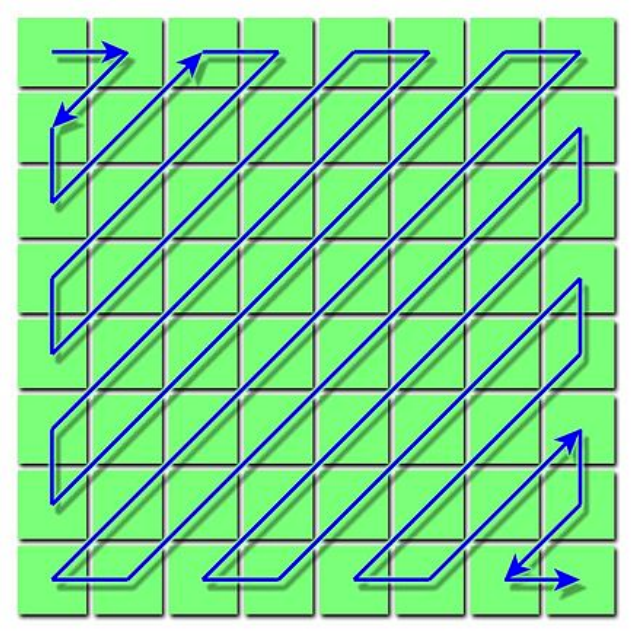
$$Q = \begin{bmatrix} 10 & 15 & 25 & 37 & 51 & 66 & 82 & 100 \\ 15 & 19 & 28 & 39 & 52 & 67 & 83 & 101 \\ 25 & 28 & 35 & 45 & 58 & 72 & 88 & 105 \\ 37 & 39 & 45 & 54 & 66 & 79 & 94 & 111 \\ 51 & 52 & 58 & 66 & 76 & 89 & 103 & 119 \\ 66 & 67 & 72 & 79 & 89 & 101 & 114 & 130 \\ 82 & 83 & 88 & 94 & 103 & 114 & 127 & 142 \\ 100 & 101 & 105 & 111 & 119 & 130 & 142 & 156 \end{bmatrix}$$

$$F = \begin{bmatrix} 782{,}91 & 44{,}93 & 172{,}52 & -35{,}28 & -20{,}58 & 35{,}93 & 2{,}88 & -3{,}85 \\ -122{,}35 & -75{,}46 & -7{,}52 & 55{,}00 & 30{,}72 & -17{,}73 & 8{,}29 & 1{,}97 \\ -2{,}99 & -32{,}77 & -57{,}18 & -30{,}07 & 1{,}76 & 17{,}63 & 12{,}23 & -13{,}57 \\ -7{,}98 & 0{,}66 & 2{,}41 & -21{,}28 & -31{,}07 & -17{,}20 & -9{,}68 & 16{,}94 \\ 3{,}87 & 7{,}07 & 0{,}56 & 5{,}13 & -2{,}47 & -15{,}09 & -17{,}70 & -3{,}76 \\ -3{,}77 & 0{,}80 & -1{,}46 & -3{,}50 & 1{,}48 & 4{,}13 & -6{,}32 & -18{,}47 \\ 1{,}78 & 3{,}28 & 4{,}63 & 3{,}27 & 2{,}39 & -2{,}31 & 5{,}21 & 11{,}77 \\ -1{,}75 & 0{,}43 & -2{,}72 & -3{,}05 & 3{,}95 & -1{,}83 & 1{,}98 & 3{,}87 \end{bmatrix}$$

$$F^Q = \begin{bmatrix} 78 & 3 & 7 & -1 & 0 & 1 & 0 & 0 \\ -8 & -4 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

5. Schritt: Quantisierung

6. Schritt: Umsortierung



7. Schritt: Entropiecodierung (z.B. Huffman)

Datenbanken

23.1.19

Schüler

INT(4)	CHAR(3)	CHAR(45)	CHAR(45)	INT
Schüler-ID	Klasse	Vorname	Nachname	gewähltes Projekt
1	11	Milena	Cordes	NULL
2	11	Nico	Schaser	NULL
3	11	Patrick	Mayer	NULL
4	11	Aaron	Sommer	2

Projekte

INT(2)	CHAR(45)	TEXT	INT(3)	INT(3)	DECIMAL(4,2)
Projekt-ID	Name	Beschreibung	TN min	TN max	Kosten
1	Eislaufen	Schwenningen	50	200	8
2	Wanderrally	Rottweil	20	80	0
3	Fußball	DSH	5	32	0

Datenbanken

1. Datenbankmodelle

Datenbanken werden verwendet, wenn viele *strukturierte* Daten abgespeichert werden sollen. Zumeist wird dabei ein *relationales Datenbankmodell* zusammen mit *MySQL* verwendet.
Informiere dich zunächst, wie eine *relationale Datenbank* aufgebaut ist und welche anderen Datenbankmodelle es gibt.

2. Entwurf einer Datenbank

Häufigste Datentypen

INT(x) Eine ganze Zahl mit maximal x Stellen.

DECIMAL(x,y) Eine Fließkommazahl mit maximal x Stellen, **davon** y Nachkommastellen.

CHAR(x) Eine Zeichenkette mit maximal x Zeichen.

TEXT Ein beliebig langer Text.

DATETIME Ein Datum inklusive Uhrzeit.

Tabellen für Projektverwaltung

Ziel soll sein, eine (sehr einfache) Datenbank für eine Projektverwaltung (z. B. für den Wintersporttag) zu erstellen.
Überlege dir zunächst, welche *Tabellen* die Datenbank dafür enthalten soll, also welche Daten du später in der Datenbank speichern willst. Gib anschließend für jede Tabelle die *Attribute*, also deren Spalten inklusive den Datentypen an.

3. Umgang mit phpMyAdmin

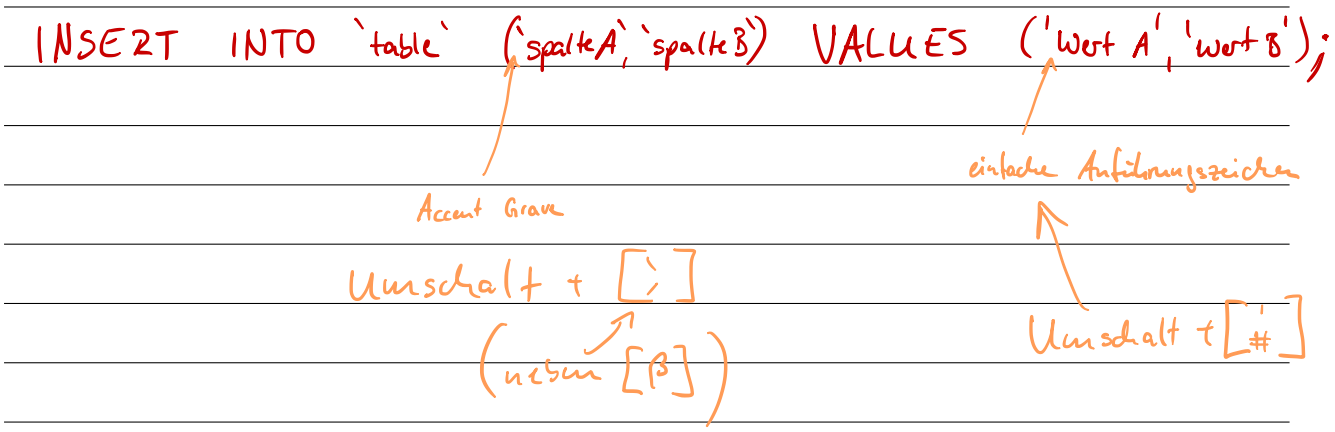
phpMyAdmin ist ein Web-basiertes Verwaltungstool um Datenbanken einfach verwalten zu können. Ihr könnt dieses mit der Adresse http://phpmyadmin.lehrer-kimmig.de (auch von Zuhause) aufrufen.
Jeder hat einen eigenen Benutzernamen: dhg_XXX wobei XXX der Benutzername des Schul-Systems ist. XXX ist gleichzeitig auch das Passwort!
Im linken Bereich seht ihr (bisher zwei) Datenbanken, wobei die *information_schema* vorerst uninteressant für uns ist. Diese beinhaltet Daten die für den Betrieb der Datenbank erforderlich sind. Wählt also eure Datenbank 181920_XXX aus und legt darin mit dem Assistenten die besprochenen Tabellen an.
Macht euch anschließend mit dem Umgang mit dem Tool vertraut und legt einige Datensätze in euren Tabellen an und löscht diese wieder.

SQL-Befehle – Teil 1

SQL steht im allgemeinen Sprachgebrauch als Abkürzung für *Structured Query Language*, leitet sich jedoch vom usrprünglichen Vorgänger *SEQUEL* (*Structured English Query Language*) ab. Dieser Name ist allerdings ein eingetragenes Warenzeichen, weshalb die Sprache in *SQL* umbenannt wurde.
Die „Sprache“ wird zur Abfrage von Daten aus Datenbanken benutzt. Eines der meistbenutzten *relationalen* Datenbanksysteme – da kostenlos für alle Systeme verfügbar und auf Webservern schon vorinstalliert – ist *MySQL*. Andere Datenbanksysteme, welche mit SQL arbeiten sind z. B. *MariaDB*, *PostgreSQL* oder *Microsoft Access*.
Zur einfachen Verwaltung einer MySQL-Datenbank gibt es unterschiedliche Tools, wir arbeiten mit *phpMyAdmin*, dieses setzt auf die Programmiersprache *PHP*.

1. Einfügen von Datensätzen

Mithilfe *phpMyAdmin* können wir auf einfache Weise neue Datensätze in eine bestehende Tabelle einfügen der zugrundeliegende SQL-Befehl wird direkt angezeigt.
Füge in verschiedene Tabellen **einige** Datensätze ein und analysiere den SQL-Befehl. Wie ist dieser für das Einfügen von neuen Datensätzen aufgebaut?



2. Anzeige bzw. Abfrage und Suche von Datensätzen

Lasse deine Datensätze anzeigen und analysiere den SQL-Befehl. Wie ist dieser für das Anzeigen bzw. Abfragen der Datensätzen aufgebaut?

SELECT * FROM `table`;

Mithilfe der Suchfunktion können wir die Ergenisse auch filtern. Suche nach Datensätzen und analysiere, was sich gegenüber der reinen Anzeige ändert.

```
SELECT * FROM `table` WHERE `spalte A` = 'Wert';
```

3. Löschen von Datensätzen

Ebenso können wir mit *phpMyAdmin* auch Datensätze per Mausklick wieder löschen. Lösche einige Datensätze ein und analysiere den SQL-Befehl. Wie ist dieser für das Löschen von Datensätzen aufgebaut?

```
DELETE FROM `table` WHERE `spalte A` = 'Wert';
```

SQL-Befehle – Teil 2

1. Einfügen von Datensätzen

`INSERT`	Befehl: Daten sollen eingefügt werden
`INTO` `` `tabelle` ``	Auswahl der Tabelle
`(` `` `spalteA` `` `,` `` `spalteB` `` `,...)`	Auswahl der Spalten
`VALUES`	Befehl: jetzt folgen die einzufügenden Daten
`('wertA','wertB',...)`	Daten

Zu beachten: die Reihenfolge der Daten muss mit der Reihenfolge der angegebenen Spalten übereinstimmen!

2. Abfrage von Datensätzen

SELECT	Befehl: Daten sollen ausgewählt werden
Alle Spalten → * `` `spalteB`,`spalteC` ``	Auswahl der auszugebenden Spalten
FROM `` `tabelle` ``	Auswahl der Tabelle
WHERE `` `spalte` = 'Wert' ``	optional: Angabe eines Suchfilters

3. Löschen von Datensätzen

DELETE	Befehl: Daten sollen gelöscht werden
FROM `` `tabelle` ``	Auswahl der Tabelle
WHERE `` `spalte` = 'Wert' ``	optional: Angabe eines Suchfilters

4. Aktualisieren

UPDATE

`` `tabelle` ``

SET `` `spalteA` = 'Wert' ``

WHERE `` `spalteB` = 'Wert' ``

SQL-Übungen

Tabellen

Nimm bei folgenden Aufgaben an, dass folgende Tabellen in deiner Datenbank existieren:

Anbieter
Anbieter-ID: `INT`
Name: `CHAR(100)`
E-Mail: `CHAR(100)`

Teilnehmer
Teilnehmer-ID: `INT`
Vorname: `CHAR(100)`
Nachname: `CHAR(100)`

Projekte
Projekt-ID: `INT`
Name: `CHAR(100)`
Preis: `DECIMAL(6,2)`
Plätze: `INT`
Anbieter: `INT`

Wahlen
Wahl-ID: `INT`
Schüler: `INT`
Projekt: `INT`

1. Einfügen von Datensätzen

Notiere die Befehle für folgende Aufgaben:

a) Lege einen Anbieter „Kimmig“ mit der E-Mail-Adresse „a.kimmig@dhg-rw.de“ an

```
INSERT INTO 'Anbieter' ('Anbieter-ID', 'Name', 'E-Mail')
VALUES ('1', 'Kimmig', 'a.kimmig@dhg-rw.de');
```

b) Ein neues Projekt mit dem Namen „Skifahren“ von diesem Anbieter wird in das Sortiment aufgenommen. Es kostet „30,50 Euro“ und hat 50 Plätze.

```
INSERT INTO 'Projekt'
VALUES ('1', 'Skifahren', '30.5', '50', '1');
```

2. Ausgabe von Datensätzen

Notiere die Befehle für folgende Aufgaben:

a) Lasse alle `Anbieter-ID`s ausgeben

```
SELECT `Anbieter-ID` FROM `Anbieter`;
```

b) Lasse nur die `Anbieter-ID` vom Anbieter mit dem Namen „Kimmig“ ausgeben

```
SELECT `Anbieter-ID` FROM `Anbieter` WHERE
`Name` = 'Kimmig';
```

c) Lasse `Name` und `Preis` aller Projekte ausgeben, welche mehr als 50 Plätze haben.

```
SELECT `Name`,`Preis` FROM `Projekte` WHERE `Plätze`>'50';
```

3. Löschen von Datensätzen

Notiere die Befehle für folgende Aufgaben:

a) Lösche alle Projekte

```
DELETE FROM `Projekte`;
```

b) Lösche alle Projekte die weniger als 10 Plätze haben.

```
DELETE FROM `Projekte` WHERE `Plätze`<'10';
```

UPDATE `tabelle` SET `spalteA` = 'Wert' WHERE ...

Übungen zu SQL-Abfragen

1. Tabellenstruktur

Wichtig bei folgenden Aufgaben ist, dass in deiner Datenbank folgende Tabellen angelegt sind:

Anbieter
Anbieter-ID: `INT`
Name: `CHAR(100)`
E-Mail: `CHAR(100)`

Käufer
Käufer-ID: `INT`
Vorname: `CHAR(100)`
Nachname: `CHAR(100)`

Produkte
Produkt-ID: `INT`
Name: `CHAR(100)`
Preis: `DECIMAL(6,2)`
Anzahl: `INT`
Anbieter: `INT`

Bestellungen
Bestellung-ID: `INT`
Produkt: `INT`
Anzahl: `INT`
Preis: `DECIMAL(6,2)`
Datum: `DATETIME`
Käufer: `INT`
Anbieter: `INT`

2. Import von Daten

Importiere die Datei `Daten.sql` aus dem Tauschverzeichnis in deine Tabellen.

3. Abfragen von Datensätzen

Schreibe alle Schritte auf, die zur Lösung der Aufgaben nötig sind! Schreibe die Befehle inklusive einer kurzen Beschreibung auf.

a) Liste alle Produkte vom Anbieter `Amazon` auf.

b) Liste alle Bestellungen von `Hans Fröhlich` auf.

c) Liste alle Produktnamen auf, die `Hans Fröhlich` bestellt hat.

d) Liste alle Produktnamen auf, die `Thomas Tischler` gekauft hat.

e) Liste alle Anbieter (mit Name und E-Mail) auf, von denen `Maximilian Mayer` etwas bestellt hat.

f) Liste alle Anbieter auf, dessen Produkte einen Lagerbestand von weniger als 20 haben.

SQL-Joins

Einführung

Mit einem `JOIN` verbindet man mehrere Tabellen anhand von übereinstimmenden Werten miteinander. Damit man eine solche Abfrage durchführen kann, müssen wir uns sogenannte **Schlüssel** definieren. So ein Schlüssel dient dazu, einen Datensatz innerhalb einer Tabelle *eindeutig* zu identifizieren, wir nennen diesen deshalb **Primärschlüssel**
Verwendet man einen solchen Primärschlüssel aus Tabelle A in einer anderen Tabelle B zur „Verknüpfung“, so nennen wir ihn in Tabelle B auch **Fremdschlüssel**

Beispiel

Eine solche Verknüpfung zweier Tabellen können wir erreichen, indem wir den `SELECT`-Befehl erweitern:

```
SELECT
        `A-ID`,`spalteA1`,`spalteA2`
FROM
        `tabelleA`
WHERE
        `spalteA3`='Wert';
```

Listing 1: normaler `SELECT`-Befehl

```
SELECT
        `spalteA1`,`spalteA2`,`spalteB1`,`spalteB2`
FROM
        `tabelleA`,`tabelleB`
WHERE
        `spalteA3`=`spalteB3`;
```

Listing 2: `SELECT`-Befehl mit Verknüpfung

Wir geben also bei der Abfrage mehrere Tabellen an und müssen mittels dem `WHERE`-Filter bestimmen, welche Attribute aus `tabelleA` und `tabelleB` übereinstimmen müssen.
Achtung: Es kann vorkommen, dass in beiden Tabellen ein Attribut mit demselben Namen existiert. Um diese Attribute *eindeutig* benennen bzw. auswählen zu können müssen wir bei diesen den Tabellennamen vornan stellen, z. B. `` `tabelleA`.`spalte` `` und `` `tabelleB`.`spalte` ``

1. Abfragen von Datensätzen

Löse die folgenden Aufgaben mithilfe von JOIN-Abfragen

a) Liste alle Produkte vom Anbieter `Amazon` auf.

b) Liste alle Bestellungen von `Hans Fröhlich` auf.

c) Liste alle Produktnamen auf, die `Hans Fröhlich` bestellt hat.

d) Liste alle Produktnamen auf, die `Thomas Tischler` gekauft hat.

e) Liste alle Anbieter (mit Name und E-Mail) auf, von denen `Maximilian Mayer` etwas bestellt hat.

f) Liste alle Anbieter auf, dessen Produkte einen Lagerbestand von weniger als 20 haben.

2. Sortieren der Ausgabe

Informiere dich, wie man die Abfrage sortiert ausgeben kann:

ORDER BY `spalte` (DESC) Rückwärts

Lasse das Bestelldatum den Namen der zugehörigen Bestellungen anzeigen, sortiert nach dem Bestelldatum.

3. Zusatzaufgabe

Informiere dich über die Begriffe bzw. die Unterschiede, beschreibe diese in eigenen Worten und gib ein einfaches Beispiel dafür an.

a) `INNER JOIN` und `OUTER JOIN`

b) `LEFT JOIN` und `RIGHT JOIN`

Statt dem Join wie bisher (mehrere Tabellen angeben, Verknüpfung mit WHERE) können wir auch den JOIN-Befehl nutzen:

SELECT * FROM `tabelle1` JOIN `tabelle2` ON (`spalteA` = `spalteB`);

Gruppierungen

1. Abfragen von Datensätzen

Löse die folgenden Aufgaben mithilfe von JOIN-Abfragen

a) Liste alle Zeitpunkte auf, an denen `Gundula Gans` etwas bestellt hat, sortiert nach dem neuesten Bestelldatum.

---

b) Liste alle Zeitpunkte auf, an denen `Gundula Gans` etwas bestellt hat inklusive des Produktnamens und der bestellten Anzahl.

---

c) Liste auf, wie viele Artikel bisher *insgesamt* bestellt wurden.

---

d) Berechne den bisherigen Gesamtumsatz.

---

2. Gruppieren der Ergebnisse

Die Funktionen `COUNT` und `SUM` berechnen jeweils den Wert **aller** Ergebnisse. Die volle Funktionalität entfalten diese erst, wenn sie zusammen mit dem `GROUP BY`-Befehl verwendet werden.
Mithilfe von diesem Befehl können Ergebnisse „gruppiert“ werden, die Funktionen werden dann für jede Gruppe separat berechnet.

```
SELECT
        `Anbieter-ID`,`Anbieter`.`Name`,COUNT(*)
FROM
        `Anbieter`,`Produkte`
WHERE
        `Anbieter-ID`=`Anbieter`
GROUP BY
        `Anbieter-ID`;
```

Listing 1: `GROUP BY`-Befehl

Diese Abfrage listet für jeden Anbieter die zugehörige ID, den Namen und die Anzahl der angebotenen Produkte auf.
Anstatt also die gesamte Zeilenanzahl auszugeben, wird nur die Zeilenanzahl für jede `Anbieter-ID` separat ausgegeben.

3. Abfragen von Datensätzen mit Gruppierung

Löse die folgenden Aufgaben

a) Liste pro Kunde auf, wie viele Bestellungen dieser getätigt hat.

b) Liste pro Kunde auf, wie viele Bestellungen dieser getätigt hat, sortiert nach der Anzahl der Bestellungen.

c) Liste pro Kunde auf, wie viel Geld dieser bisher ausgegeben hat.

d) Liste den Namen der Produkte auf zusammen mit der Anzahl, wie oft diese schon bestellt wurden.

13.2.19

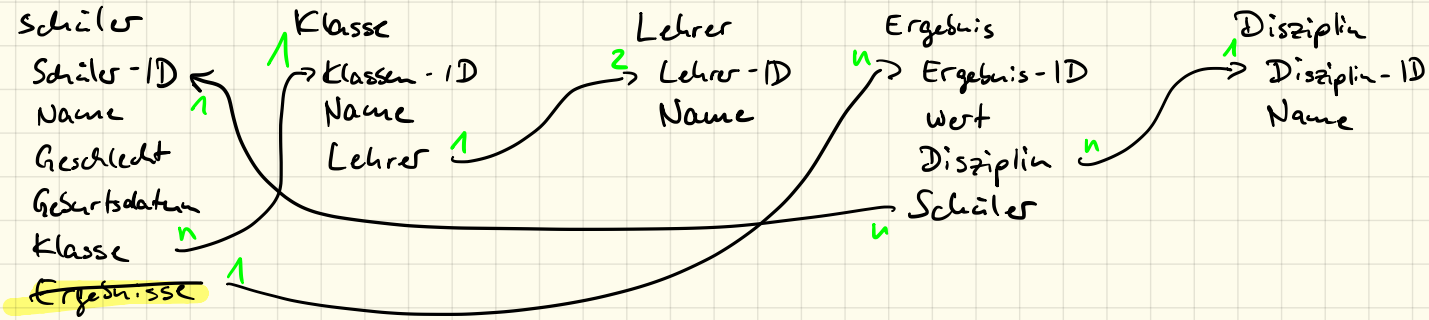
Bisher wurde am „Konrad-Zuse-Gymnasium“ die Auswertung der Bundesjugendspiele mit Hilfe einer Tabellenkalkulation durchgeführt. Es ist geplant, stattdessen in Zukunft eine Datenbank zu verwenden.

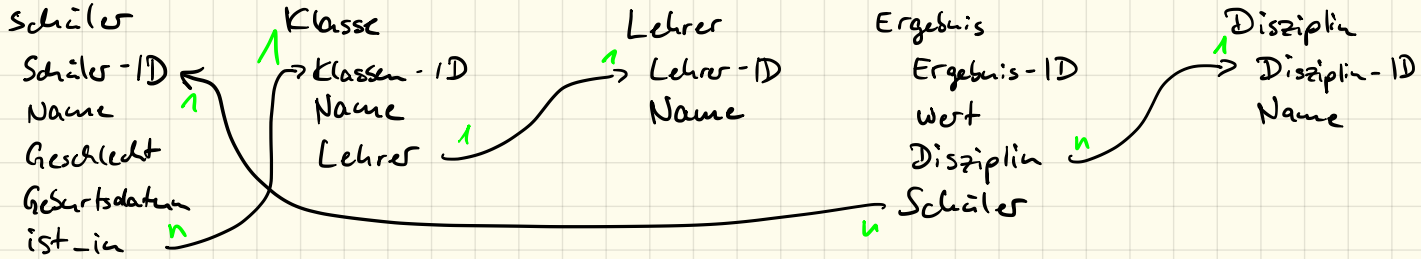
Name	Geschlecht	Geb.-Datum	Klasse	Klassenlehrer	Disziplin	Ergebnis
Peter Müller	m	3.2.2005	6a	Herr Meyer	50m	8,6
Max Schnell	m	8.5.2005	6a	Herr Meyer	50m	7,4
….						
Manuela Bach	w	18.1.2003	8b	Frau Moser	75m	12,4
….						
Peter Müller	m	3.2.2005	6a	Herr Meyer	Weitsprung	3,45
Max Schnell	m	8.5.2005	6a	Herr Meyer	Weitsprung	4,41
….						
Peter Müller	m	3.2.2005	6a	Herr Meyer	Schlagball 80g	27,0
….						

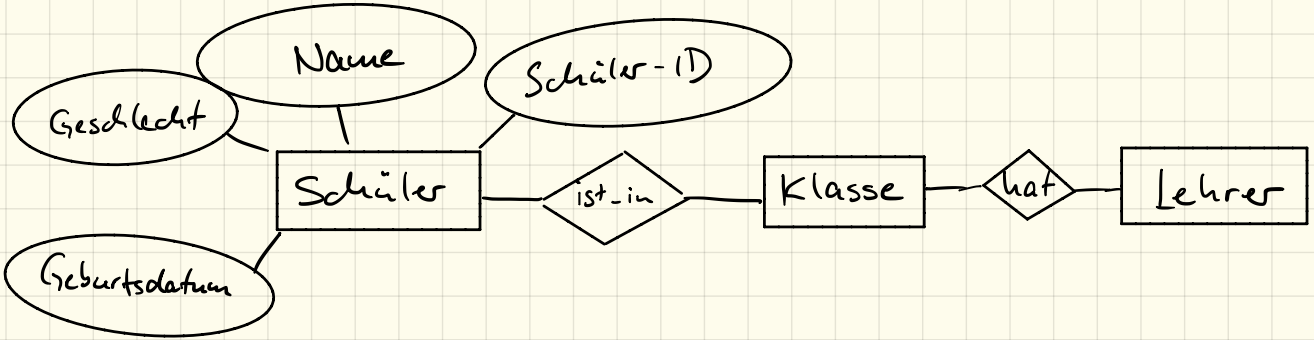
Erläutern Sie, welche Probleme sich bei Verwendung einer Tabelle mit der obigen Struktur ergeben können. Gehen Sie dabei insbesondere auf die Begriffe „Redundanz“ und „Dateninkonsistenz“ ein.

Name	Geschlecht	Geb.-Datum	Klasse	Klassenlehrer	Disziplin	Ergebnis
Peter Müller	m	3.2.2005	6a	Herr Meyer	50m	8,6
Max Schnell	m	8.5.2005	6a	Herr Meyer	50m	7,4
....						
Manuela Bach	w	18.1.2003	8b	Frau Moser	75m	12,4
....						
Peter Müller	m	3.2.2005	6a	Herr Meyer	Weitsprung	3,45
Max Schnell	m	8.5.2005	6a	Herr Meyer	Weitsprung	4,41
....						
Peter Müller	m	3.2.2005	6a	Herr Meyer	Schlagball 80g	27,0
....						

Entwerfen Sie ein Entity-Relationship-Diagramm für die geplante Datenbank, das die obigen Probleme vermeidet. Kennzeichnen Sie die Schlüsselattribute in Ihrem Entwurf.







Ergebnis

Disziplin

Entity-Relationship-Diagramme

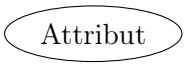
Einführung

Die grafische Darstellung von Entitätstypen (Tabellen einer Datenbank) und Beziehungen dazwischen wird *Entity-Relationship-Diagramm* oder *ER-Diagramm* genannt.[1]
Zur Darstellung existieren unterschiedliche Notationsformen, die gängigsten sind die *Chen-Notation* (die wir verwenden wollen) und die *UML-Notation*.

1. Chen-Notation

In der Chen-Notation existieren vier Darstellungen:

Entitäten oder Objekte können in einer Datenbank als (Einträge in einer) Tabelle angesehen werden.



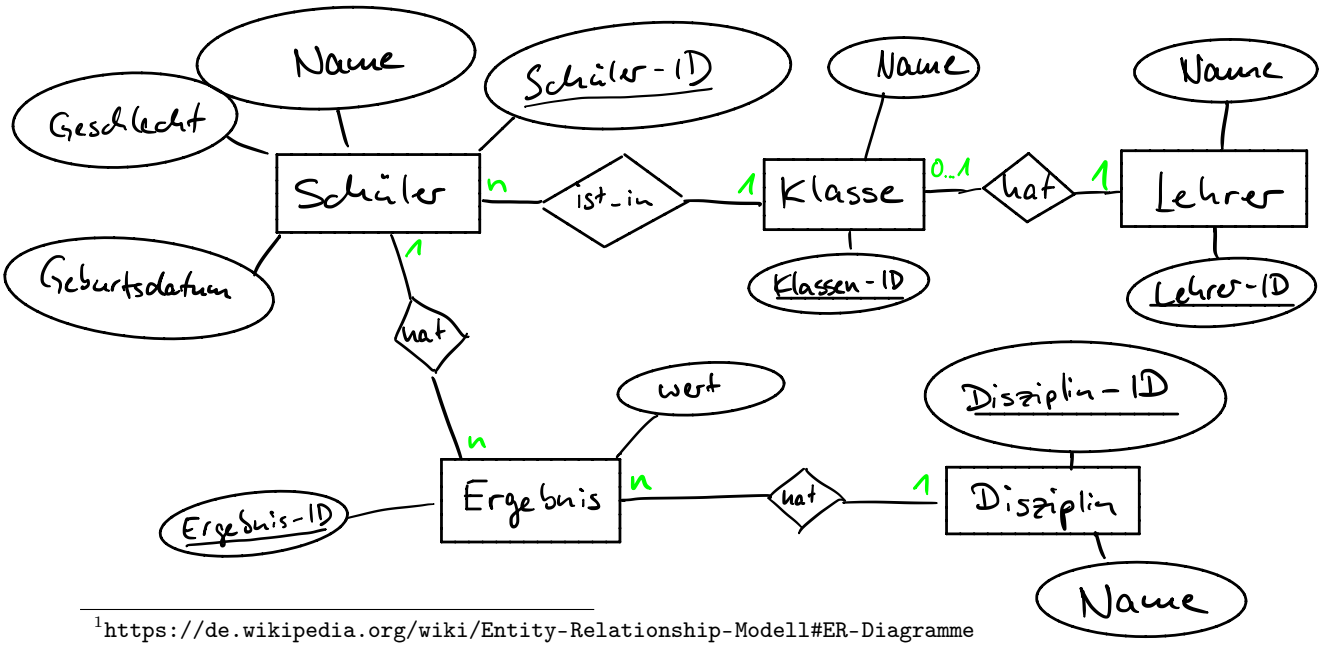
Attribute oder Eigenschaften der Entität sind Tabellenspalten

Verhältnis bzw. Zusammenhang zischen zwei Tabellen. Hierbei wird jeweils die Kardinalität angegeben, d. h. welche Anzahl davon existieren.

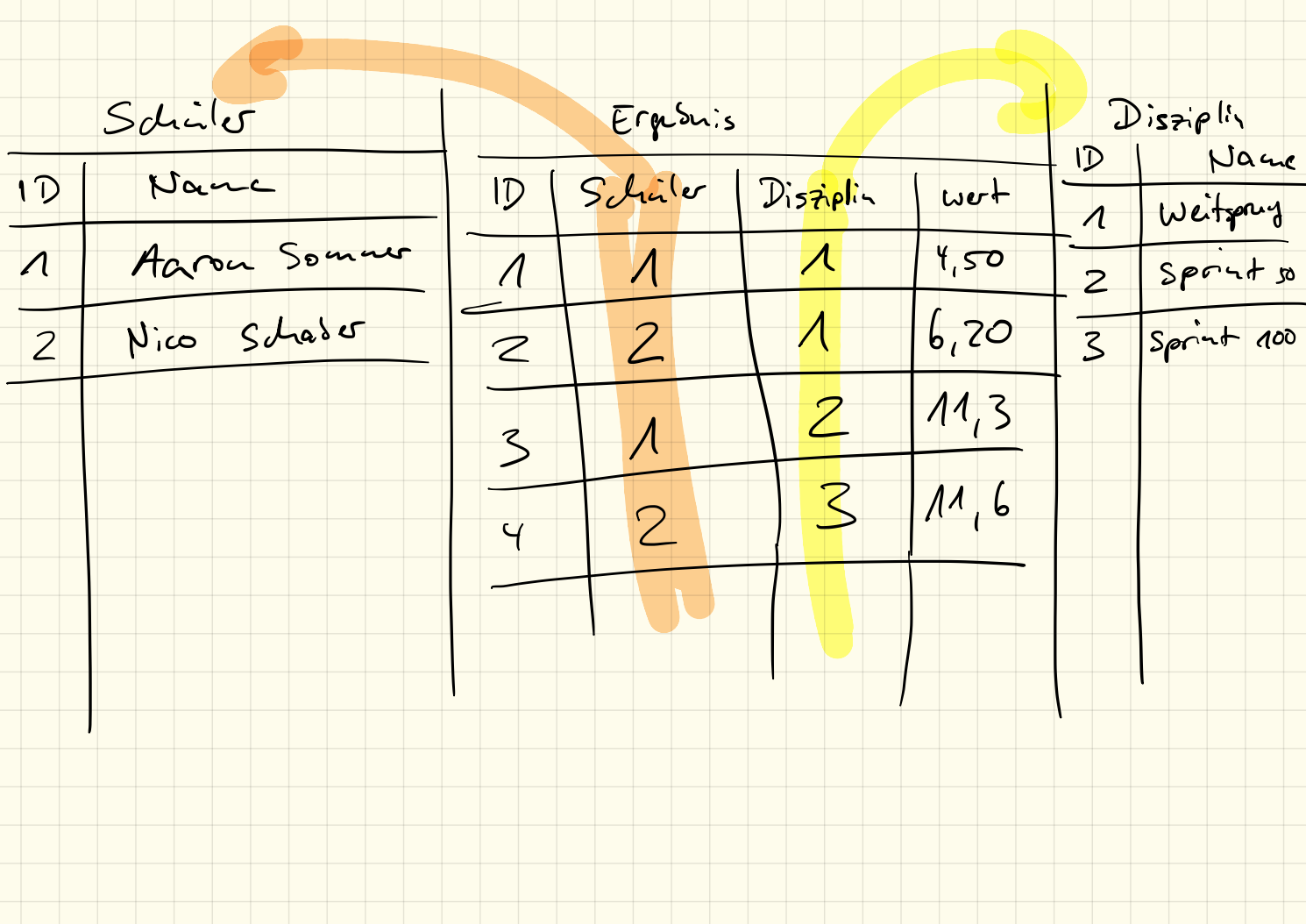
Abgeleitete Attribute sind solche, die nicht als Tabellenspalte explizit existieren, sondern sich aus anderen Spalten berechnen. Beispielsweise kann das *Alter* als abgeleitetes Attribut angesehen werden wenn das Geburtsdatum bereits als Attribut existiert.

2. Aufgabe

Vervollständige das ER-Diagramm von letzter Stunde.



[1] https://de.wikipedia.org/wiki/Entity-Relationship-Modell#ER-Diagramme



Schüler

ID	Name
1	Aaron Sommer
2	Nico Schaber

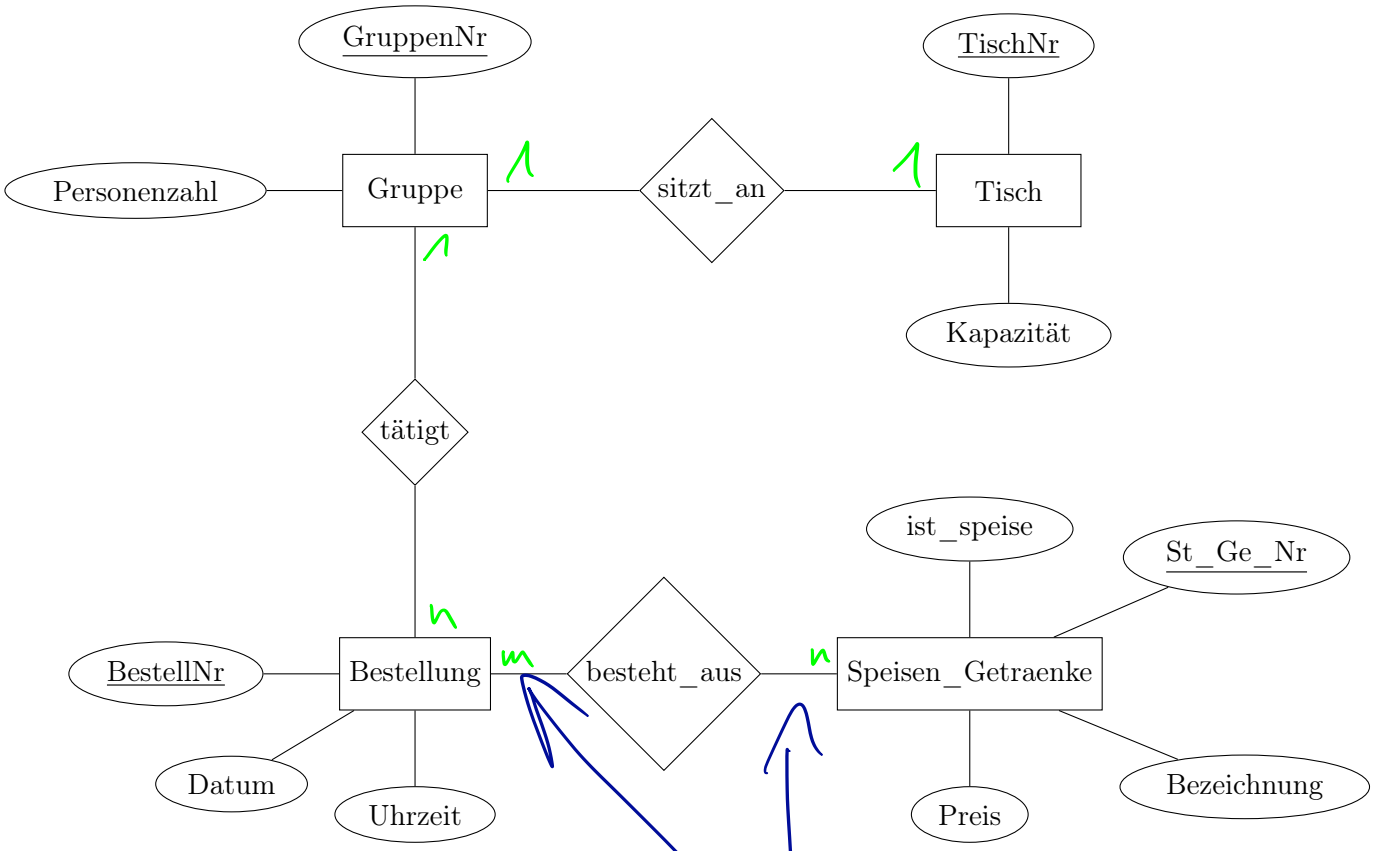
Ergebnis

ID	Schüler	Disziplin	wert
1	1	1	4,50
2	2	1	6,20
3	1	2	11,3
4	2	3	11,6

Disziplin

ID	Name
1	Weitsprung
2	Sprint 50
3	Sprint 100

3. Aufgabe



a) Gib die Kardinalitäten an

b) Überführe das Diagramm in eine Tabellenstruktur

c) Gib eine passende SQL-Abfrage an:

- Wie viele Speisen stehen auf der Speisekarte?
- Was wurde zum jeweiligen Preis an Tisch 7 bestellt?
- Wie hoch ist der Rechnungsbetrag an den einzelnen Tischen?

Bei einer m-n-Verknüpfung brauchen wir eine Hilfstabelle

Tisch
TischNr
Kapazität

Gruppe
GruppenNr
Personenzahl

Tisch

Bestellung
BestellNr
Datum
Uhrzeit
Gruppe

Speisen_Getränke
St_Ge_Nr
ist_speise
Bezeichnung
Preis

bestehtaus
Bestellung
speisen_Getränke

Bestellung

ID	Gruppe
1	1
2	2
3	1
4	1

besteht_aus

Bestellung	Sp-Ge
1	1
1	2
1	3
2	1
2	4
3	2
4	4

Sp-Ge

ID	Name
1	Cola
2	Fanta
3	Pizza
4	Pommes