

Informatik Kursstufe 4-stündig

Schuljahr 18/19

Organisation:

- kein Heft oder Ordner für Arbeitsblätter
- Klausuren 2 pro Hj
- Notenverhältnis

50% schriftlich
 20% Projekt
 30% mündlich

70% schriftlich falls kein Projekt

- schule @ lehrer - kimnig. de
- wiki. lehrer - kimnig. de
- ab. lehrer - kimnig. 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

- Kryptografie
und Signaturen

- Netzwerk

- Schichtenmodell
- Spuren im Netz
- Kommunikation
- Angriffe und Schutz

- Sicherheit

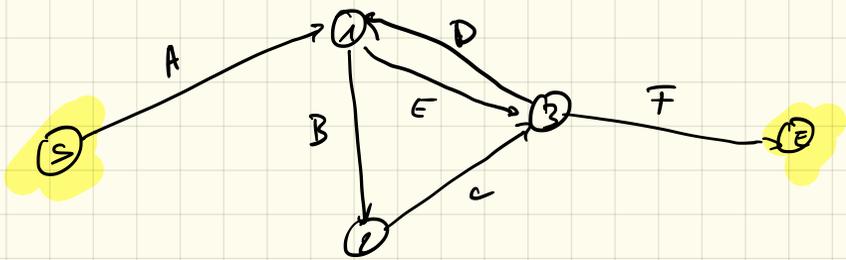
- DB

- SQL

- Rechnerarchitektur

- Technik, logische Schaltungen

- Automatentheorie



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

Definition Algorithmus:

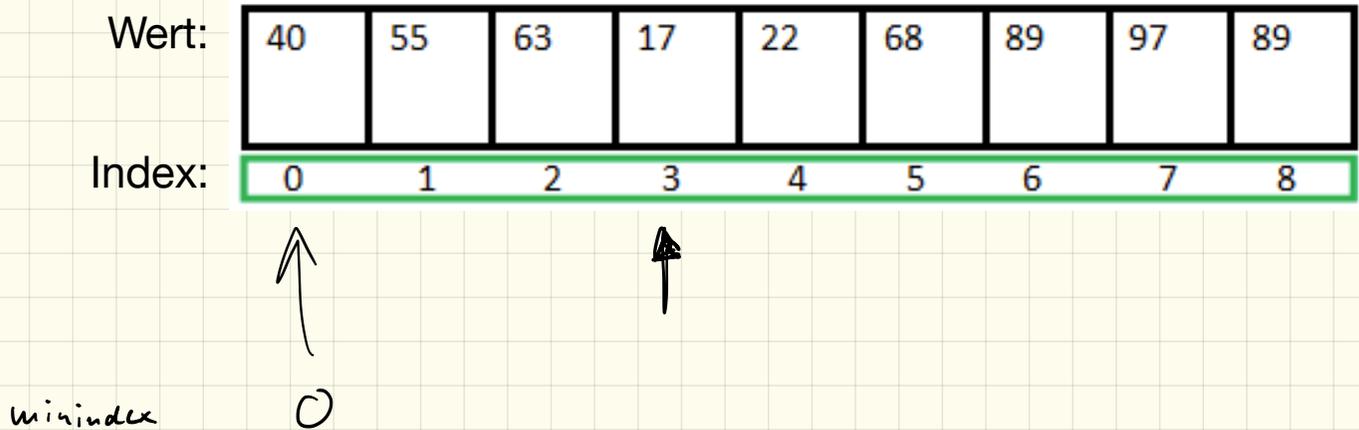
Ein Algorithmus ist eine eindeutige **Handlungsvorschrift** zur Lösung eines Problems oder einer Klasse von Problemen. Algorithmen bestehen aus endlich vielen, wohldefinierten **Einzelschritten**. Damit können sie zur Ausführung in ein Computerprogramm implementiert, aber auch in menschlicher Sprache 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




```
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:

- Erzeuge zunächst ein Array, welches 20 Ganzzahlen speichern kann.
- Befülle dieses Array mit 20 zufälligen Zahlen (zwischen 0 und 50).
- 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.

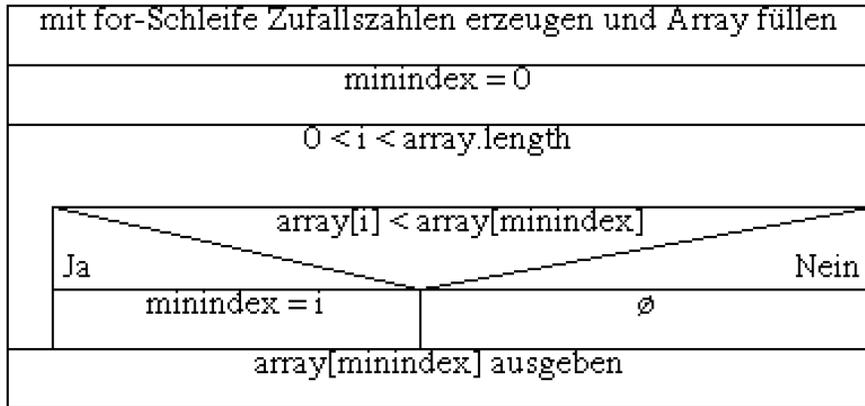
5. Zusatzaufgabe 2: Sortierte Ausgabe

Überlege dir, wie man diese Minimumsuche 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:

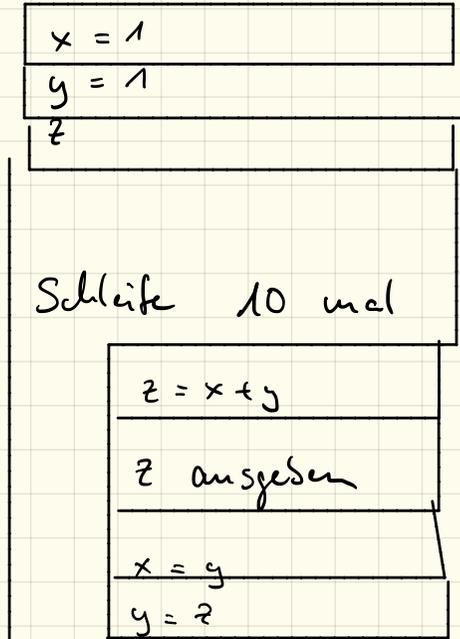
Minimunsuche:



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

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, ~~100~~, ~~100~~, ~~100~~, 9, 11, 17

3, 5, 7

```
Array mit Zufallszahlen erzeugen  
for(int i=0; i<array.length; i++)  
    minindex = Minimumsuche  
    array[minindex] anspeichern  
    array[minindex] = 100
```

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

	0	1	2	3	4	5	6	7
a =	8	7	5	9	17	3	100 100	14

↑
m

↑
i

minindex = 6

i = 8

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

a[minindex] = 100

19.9.18

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

- Grundstrukturierung einer Klasse

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

$$f(x) = x^2$$

$$f(5) = 25$$

Methode um ein Array mit
Zufallszahlen zu erzeugen:

Anzahl
Elemente
Zahlen bis

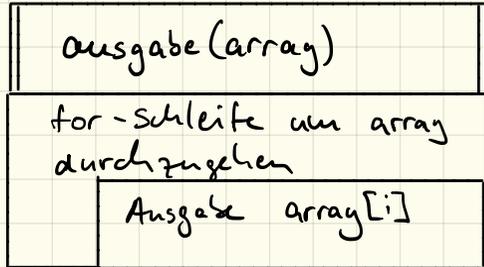
Ziel: `int[] array = zufall(20, 50);`

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

```
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);`

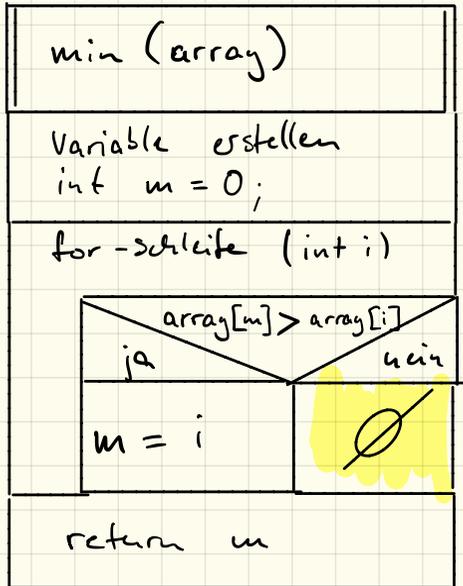


```
public static void ausgabe(int[] array) {  
    for (int i=0; array.length; i++) {  
        System.out.print(array[i]);  
    }  
}
```

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

sortierte Ausgabe (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] [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] = array[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.

0	1	2	3
2	7	4.5	3



```
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

- Wie verändert sich die Laufzeit, wenn die Größe des Arrays verdoppelt, verdreifacht, ... wird?
- 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:

$$\begin{array}{c} 2 \\ n \end{array}$$

divide-and-conquer:

~~1~~ ~~2~~ ~~5~~ ~~6~~ ~~8~~ ~~9~~ ~~10~~ ~~12~~ ~~13~~ ~~14~~

~~1~~ ~~2~~ ~~5~~ ~~6~~ ~~8~~ ~~9~~ ~~10~~ ~~12~~ ~~13~~ 14

1	2	5	6	8	9	10	12	13	14
---	---	---	---	---	---	----	----	----	----

Minimumsude: 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

$(\frac{n}{2})^2$

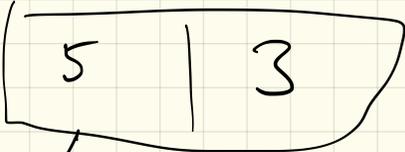
Liste 2 sortieren: 250 000

$(\frac{n}{2})^2$

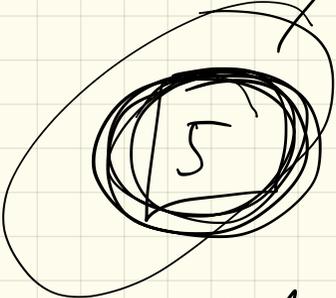
Zusammenfügen: 1000

n

502002 Schritte



$k = \text{Mergesort}(k)$
 $g = \text{Mergesort}(g)$



$t = 1$

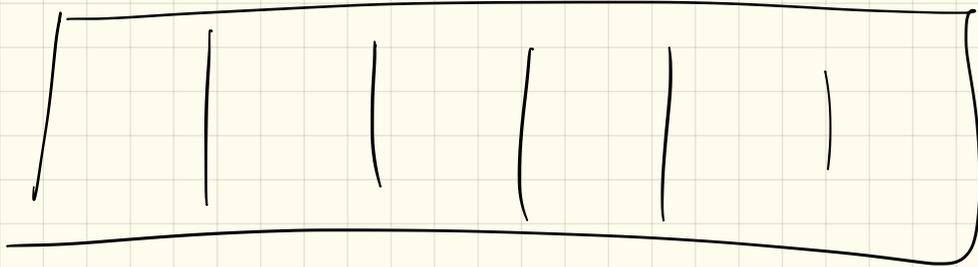
$v = 2$

~~4~~ 7

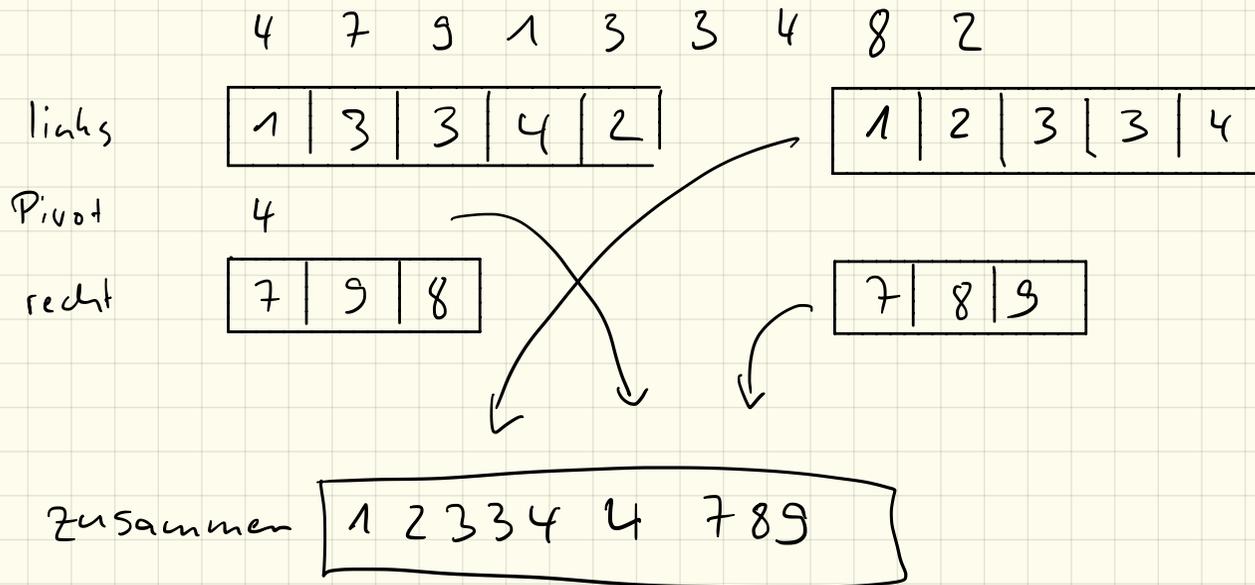
~~4~~ 6

3 4 6

31 37 35 8 15



Quick Sort



1	3	3	4	2
---	---	---	---	---

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

links: []

Pivot: 1

rechts: 3 3 4 2

2	3	3	4
---	---	---	---

Zusammen

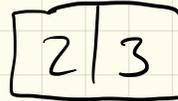
1	2	3	3	4
---	---	---	---	---

3 3 4 2

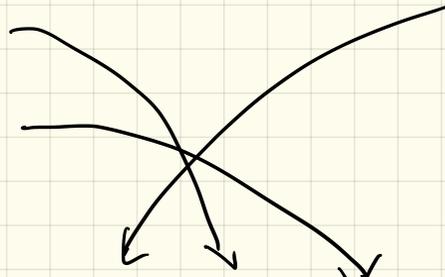
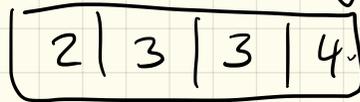
links 3 2

Pivot 3

rechts 4



Zusammen



3 2

links 2

pivot 3

rechts []

Zusammen

2 3

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.

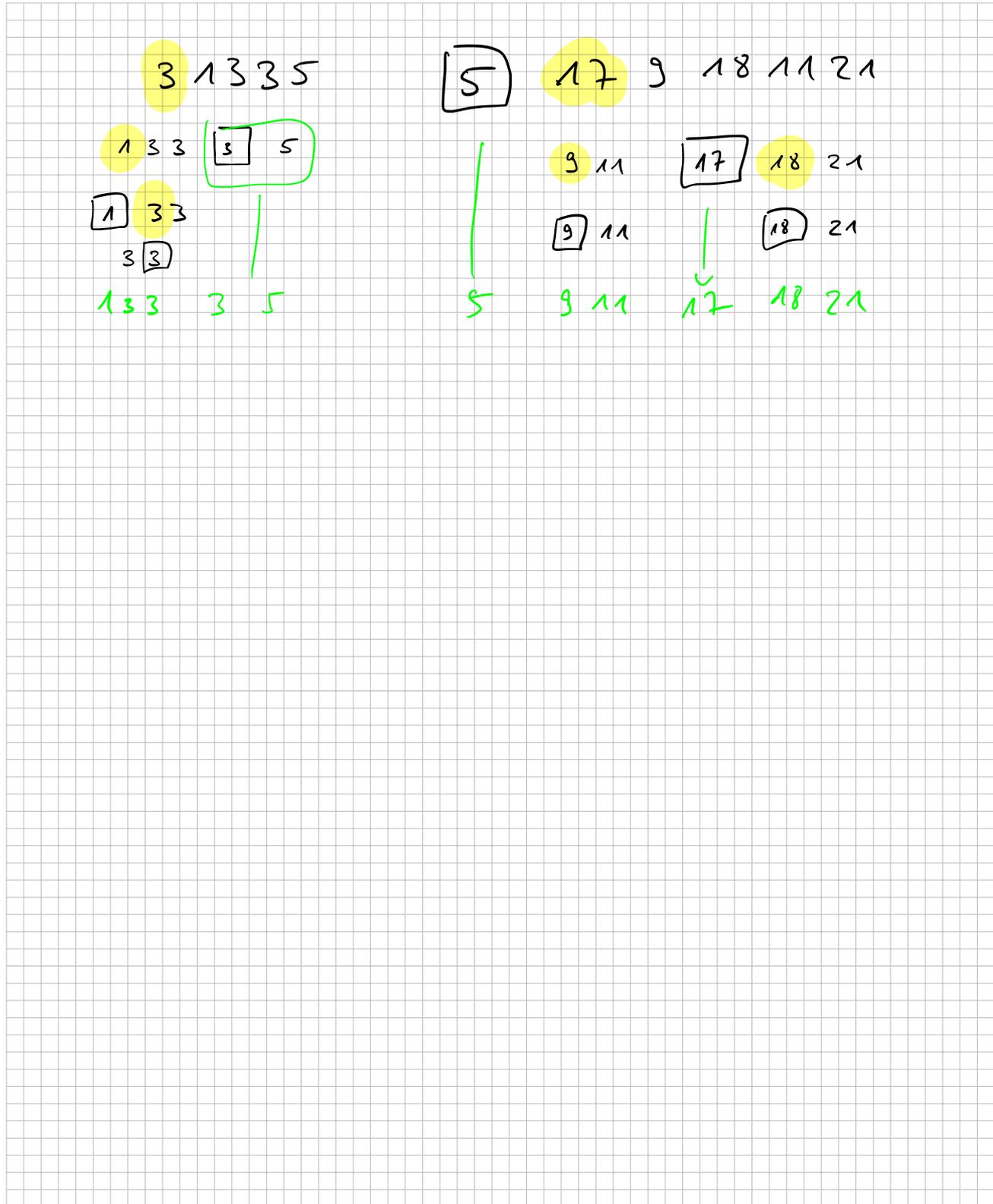
$$n = 100 \quad \rightarrow \quad \text{SelectionSort} \sim 10000$$

Merge Sort :	Liste 1 : 50	Selection	2500
	Liste 2 : 50	selection	2500
			<u>5000 Schritte</u>

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

Selektions Sort out-of-place

$\left. \begin{array}{l} n \text{ Schritte} \\ n \text{ Minimumsuchen} \end{array} \right\} \text{ Minimum}$
 \Rightarrow immer n^2 Schritte

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

Selektions Sort 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 \rightarrow im besten Fall n Schritte

$$\Rightarrow \Theta(n)$$

1 2 3 4 5 6

\rightarrow im schlechtesten Fall

6 5 4 3 2 1

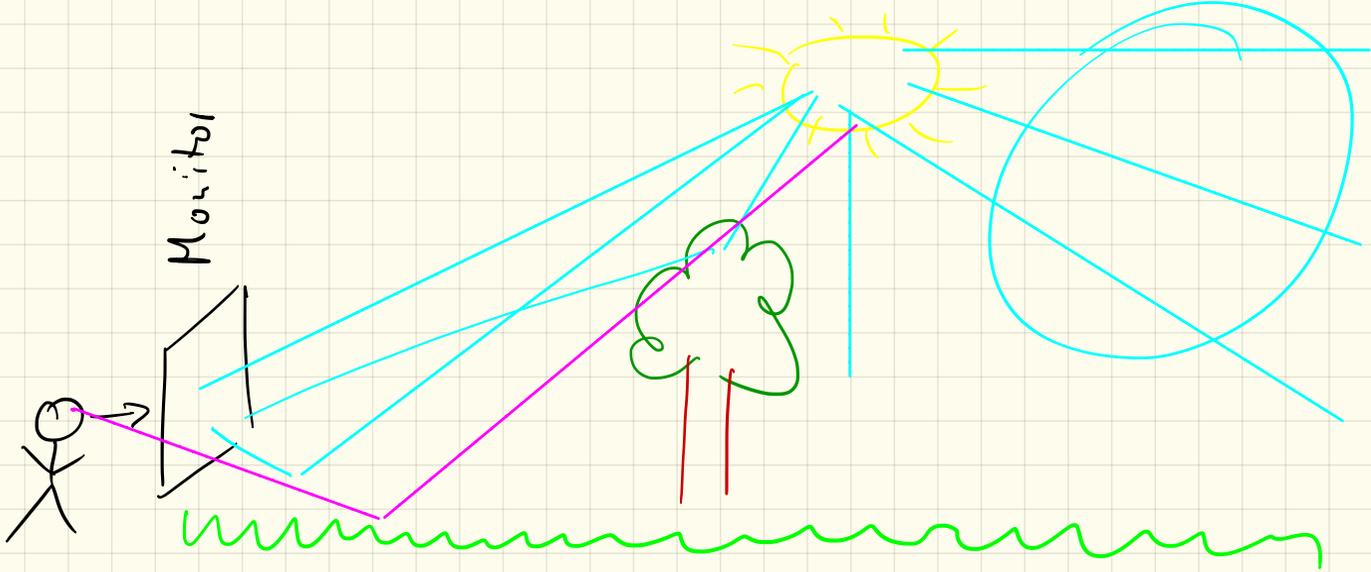
$n-1$ Zahlen ausdrehen

$$\left. \begin{array}{l} 5 \rightarrow 1 \text{ Stelle} \\ 4 \rightarrow 2 \text{ Stellen} \\ 3 \rightarrow 3 \text{ Stellen} \\ \vdots \\ n-1 \text{ Stellen} \end{array} \right\} \frac{n-1}{2} \text{ Verschiebungen} \\ \text{im Durchschnitt}$$

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

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

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



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 $(x | y)$ 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

- Lasse (mithilfe einer `for`-Schleife) eine Zeile des Fensters einfärben
- Lasse (mithilfe einer `for`-Schleife) eine Spalte des Fensters einfärben
- Kombiniere diese beiden Schleifen um das ganze Fenster einzufärben
- Probiere auch unterschiedliche Farben selbst aus um dich mit dem RGB-Farbschema vertraut zu machen.
- 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 Lichtstrahl, der vom Auge ausgeht und durch den Pixel $(x | y)$ 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 wiederum 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 wiederum 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

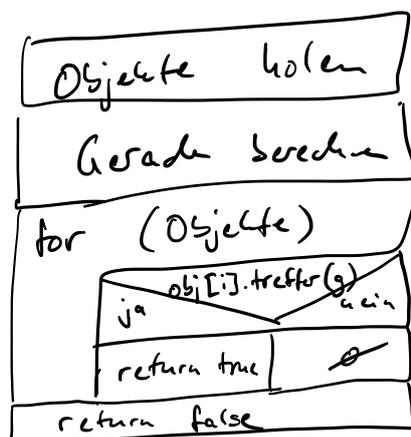
Ziel wird es sein, die Methode `tr.trifft(int x,int y)` so nachzubilden, dass wir diese später erweitern können.

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 Parameter soll die `y`-Koordinate übergeben werden

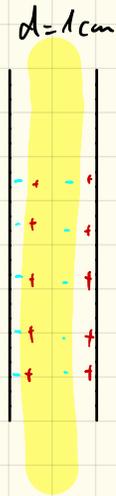
Diese können wir anschließend in unserer `main`-Methode benutzen und damit die `tr.trifft`-Methode ersetzen.

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;
}
  
```



$$\epsilon_r = 5,0$$

$$U_0 = 1000 \text{ V}$$

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

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

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

a) $Q = C \cdot U$

$$C = \epsilon_0 \cdot \epsilon_r \cdot \frac{A}{d}$$

$$C_{\text{mit}} = \epsilon_r \cdot C_{\text{ohne}}$$

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

\Rightarrow Kapazität erhöht sich auf das 5-fache

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

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

$$\epsilon_0 = 8,85 \cdot 10^{-12} \frac{As}{Vm}$$

$$C = \epsilon_0 \cdot \frac{A}{d} = 4,425 \cdot 10^{-11} F$$

$$Q_{\text{ohne}} = C \cdot U$$

$$Q_{\text{mit}} = 5 \cdot Q_{\text{ohne}}$$

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

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

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

Farbe

```
public static rot 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 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 green;
```

```
    public double blau;
```

```
}
```

```
int a = 77;
```

```
Farbe x = new Farbe();
```

```
x.rot = 0.5;  
x.green = 0.7;  
x.blau = 0.1;
```

```
Farbe y = new Farbe();
```

```
y.rot = 0;  
y.green = 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

vorname: Aaron

nachname: Sommer

```
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.

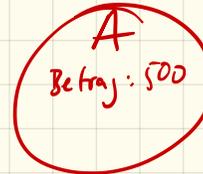
- a) Welche Eigenschaften/Attribute muss so ein Bankkonto enthalten?
b) Welche Methoden müssen programmiert werden?

a)

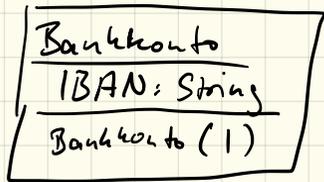
Bankkonto
IBAN: String
BIC: String
Inhaber: String
Betrag: double

b)

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

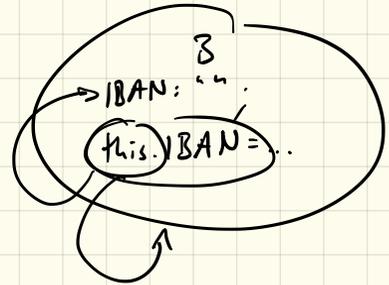
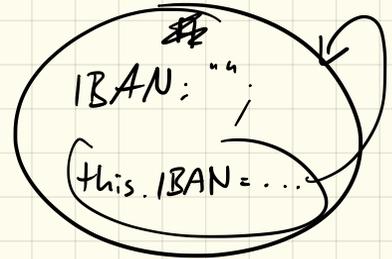


A. Überweisen (100, B);



`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

Buch	
titel	: String
autor	: String
nummer	: int
verliehen	: boolean
leiher	: Person
istVorhanden ()	: boolean
ausleihen (Person)	
zurueckgeben ()	
Buch (String titel, String Autor, int nr)	

Person	
name	: String
nummer	: int
Person (String name, int nr)	

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
- Alter
- Farbe
- Leistung
- Kontonummer
- 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 liegt darin, dass...

<code>public</code> :	von überall aufrufbar und verwendbar
<code>private</code> :	nur aus der „eigenen“ Klasse verwendbar
• Attribute, die nicht öffentlich einsehbar sein sollen	→ <code>private</code>
• Attribute, die nicht frei veränderbar sein sollen	→ <code>private</code>

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)	
<code>public KLASSENNAME(...)</code>	→ keinen Rückgabetyp
	→ Name der Methode entspricht dem Klassennamen

Autohaus

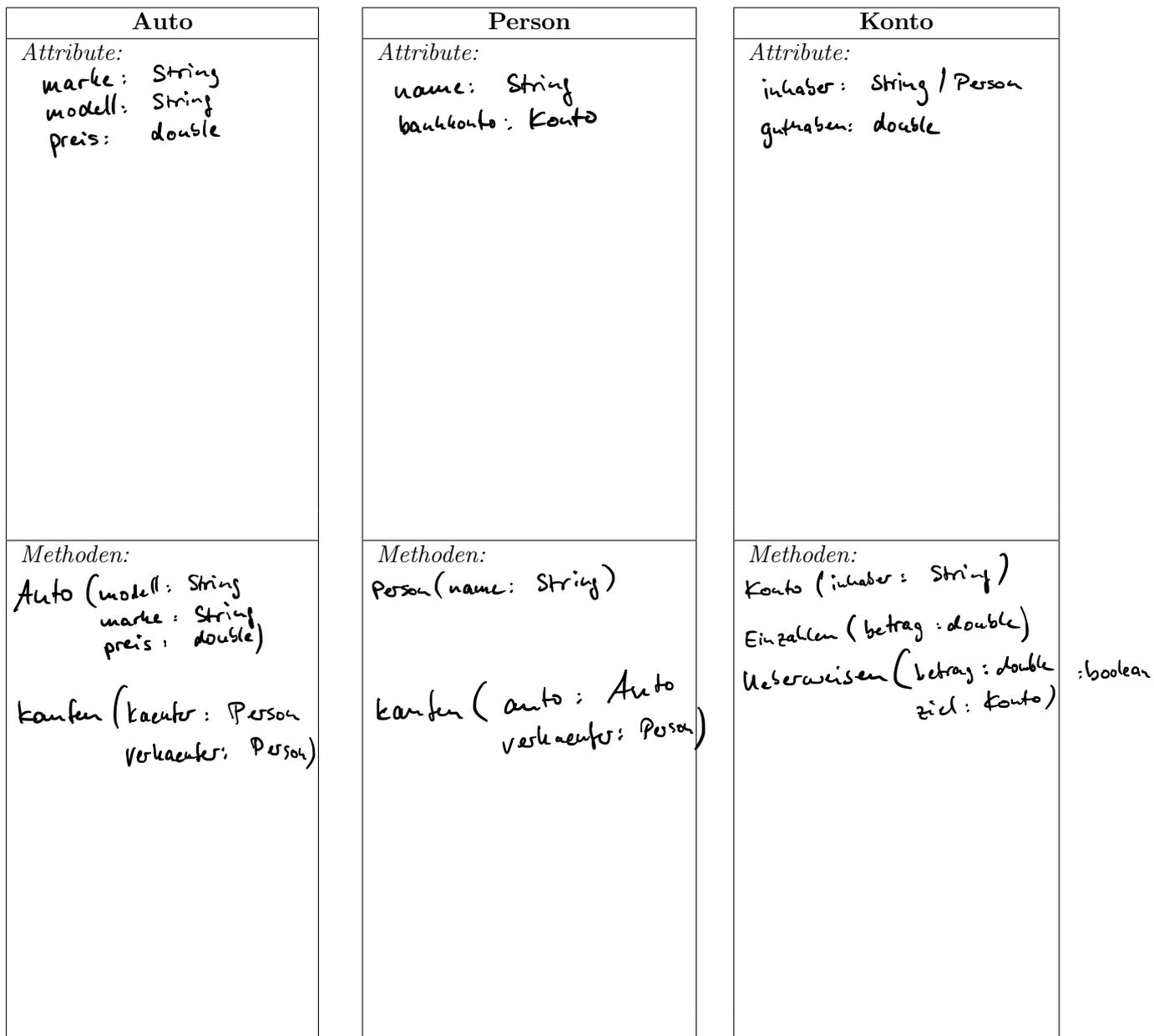
1. Vorüberlegungen

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!)

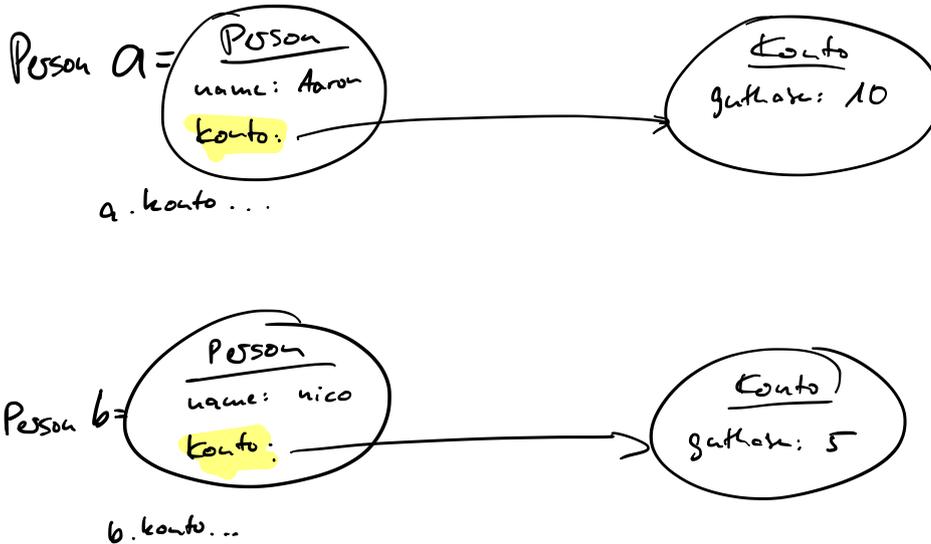
Entwerfe zunächst also die Klassen: (beachte dazu auch die Tests auf der Rückseite!)



2. Programmierung und Tests

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

- 2 Autos anlegst
- 2 Personen anlegst (^{mass}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

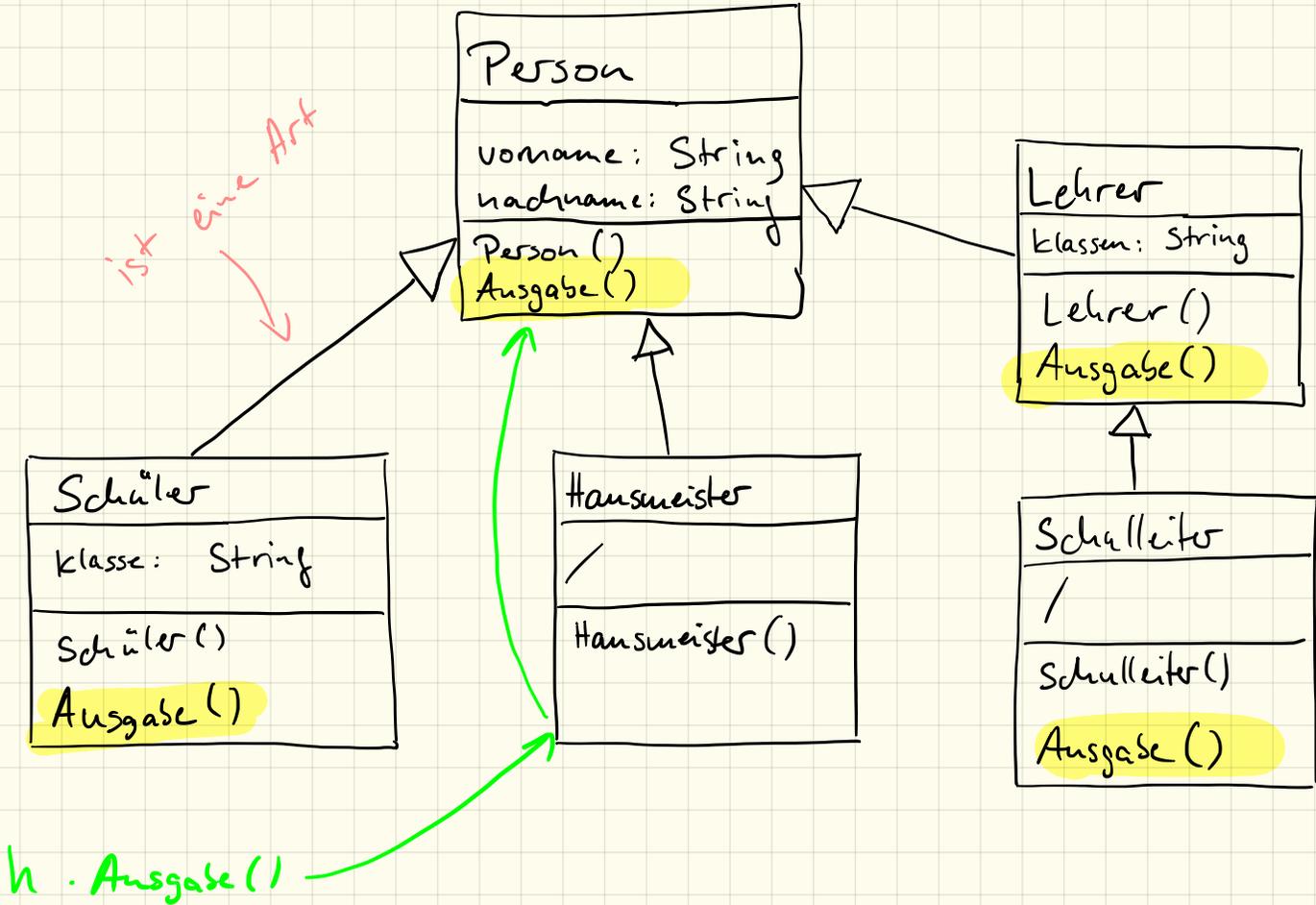
Vorname
nachname
klassen []

Schulleiter

Vorname
nachname
klassen []

Hausmeister

Vorname
nachname



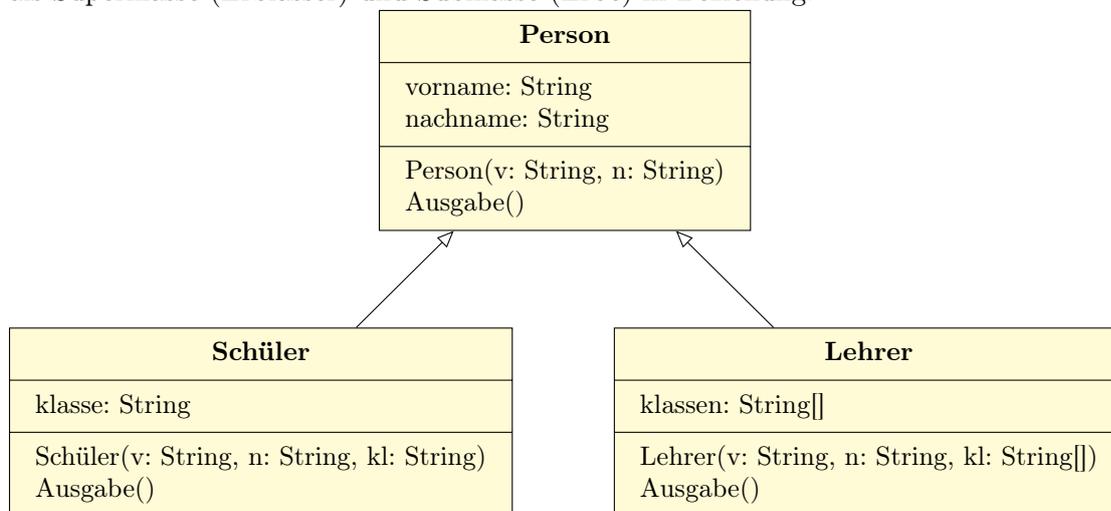
UML, Vererbung und Aggregation

1. UML

Die *Unified Modeling Language* (vereinheitlichte Modellierungssprache), kurz *UML*, ist eine Möglichkeit, Strukturen und Zusammenhänge 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`

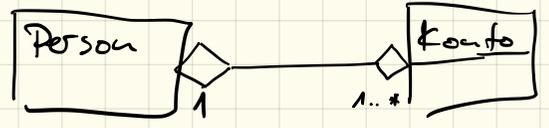
Soll von einer Unterklasse auch auf Attribute der Hauptklasse zugegriffen werden, so dürfen diese nicht als `private` markiert sein. Stattdessen kann man hierzu das Schlüsselwort `protected` verwenden. Hierbei erhalten neben der eigentlichen Klasse (wie bei `private`) auch abgeleitete Unterklassen Zugriff.

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

Vererbung "ist-eine-Art" - Beziehung 

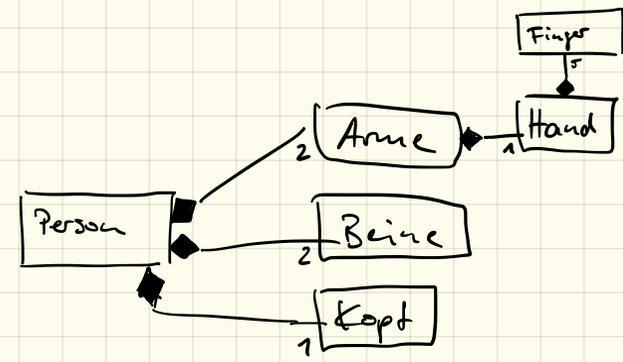
Aggregation

"hat" - Beziehung



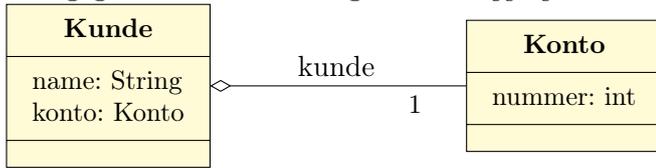
Composition

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

5. Aufgabe

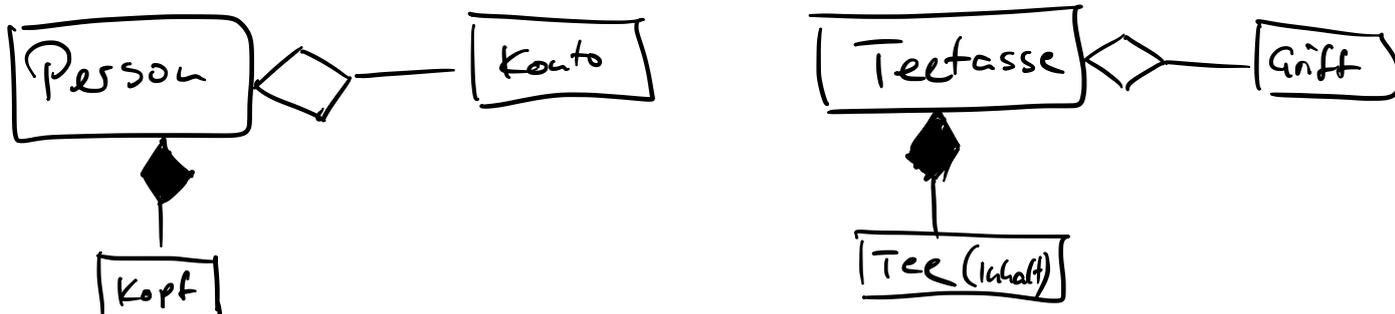
Zeichne ein UML-Diagramm mit folgenden Begriffen:

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

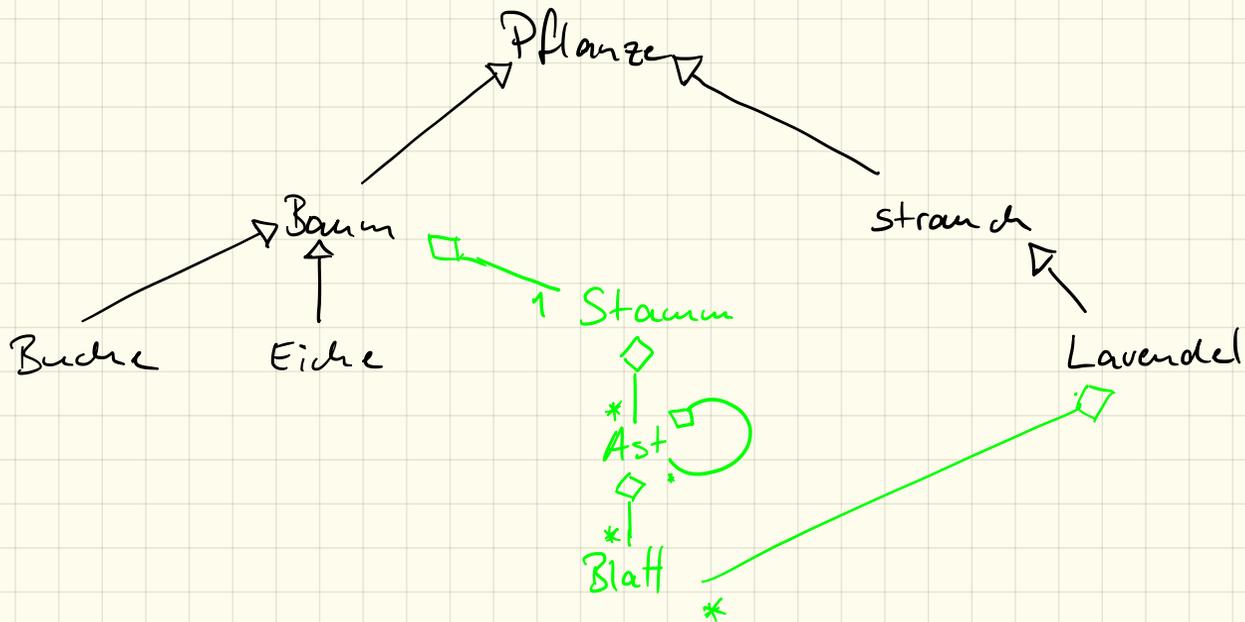
• ~~Blatt~~

• ~~Strauch~~

• ~~Buche~~

• ~~Eiche~~

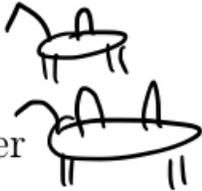
• ~~Lavendel~~



• ~~Kamel~~

• ~~Dromedar~~

• ~~Trampeltier~~



• ~~Lama~~

• ~~Höcker~~

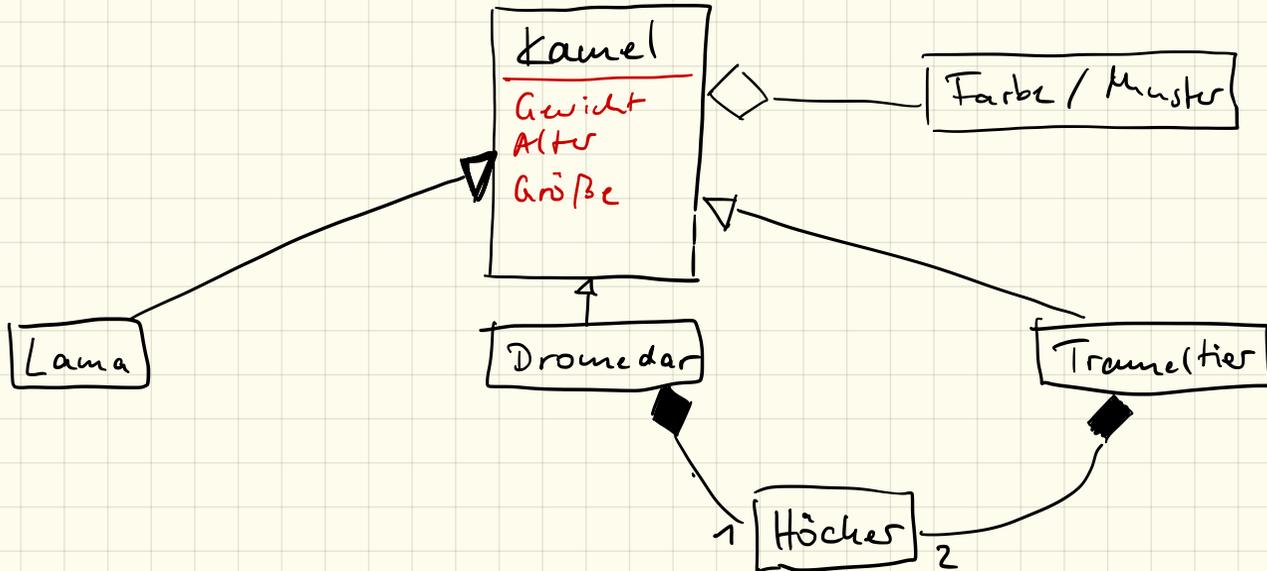
• Gewicht



• Alter

• Größe

• Farbe



Listen : ArrayList

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

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

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

```
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 beachten, 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]`)

¹Anmerkung: 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.

- Zeichne zuerst das Klassendiagramm (inklusive der Klassen aus letzter Stunde!)
- Erstelle in der `Main`-Klasse² in der `main`-Methode eine Instanz der Schule und füge anschließend neue Personen über die Methode der Schule hinzu.
- Rufe anschließend die `Ausgabe`-Methode der Schule auf.

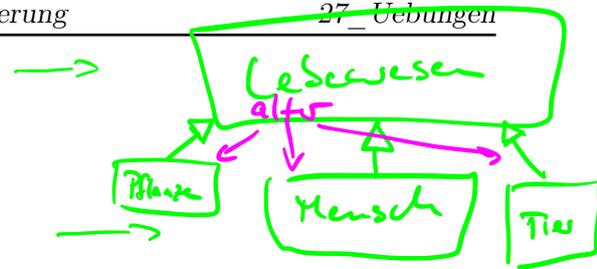
```
1 Schule dhg = new Schule("Droste-Hülshoff-Gymnasium");
2
3 Schüler as = new Schüler("Aaron", "Sommer", "KS1");
4 dhg.addPerson( as );
5
6 Person mc = new Schüler("Milena", "Cordes", "KS1");
7 dhg.addPerson( mc );
8
9 dhg.addPerson( new Lehrer("Alexander", "Kimmig", "10b, KS1, KS2") );
10 dhg.addPerson(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

Listing 2: Beispielcode

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

²oder `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*
- *überall verwendbar* `public, private, protected` *innerhalb der Vererbungsreihe verwendbar*
- Polymorphismus *bedeutet, dass es unterschiedliche Methoden mit gleichem Namen geben kann (z.B. bei Vererbung)*

2. Beziehungen

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

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

3. Klassendiagramm

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

