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

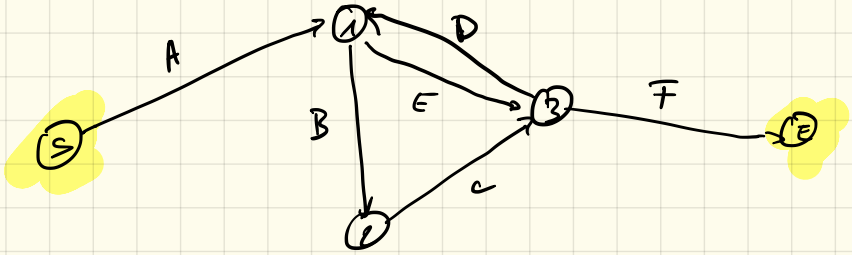
- schule@lehrer-kimmig.de
- wiki.lehrer-kimmig.de
- ab.lehrer-kimmig.de
- GFS mind. 15-20 min, gerne länger + Handout

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:

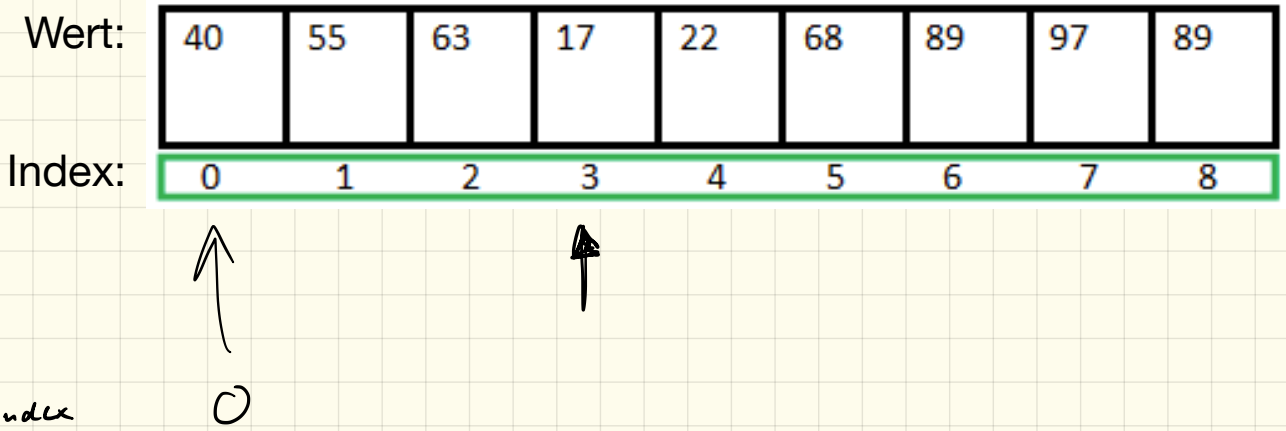
Ein Algorithmus ist eine eindeutige Handlungsvorschrift zur Lösung eines Problems oder einer Klasse von Problemen. Algorithmen bestehen aus endlich vielen, wohldefinierten Einzelschritten. Damit können sie zur Ausführung in ein Computerprogramm implementiert, aber auch in menschlicher Sprache formuliert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

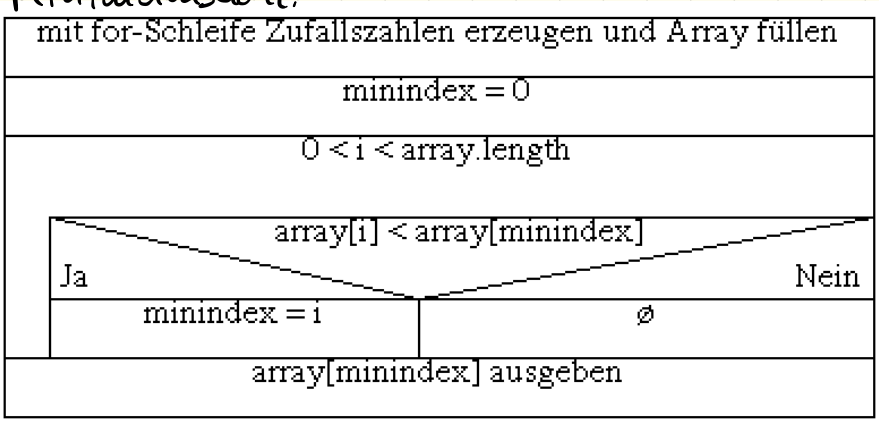
5. Zusatzaufgabe 2: Sortierte Ausgabe

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:

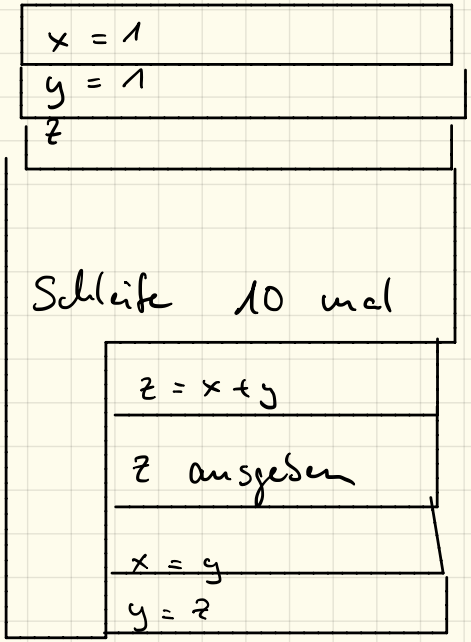
Minimumsuche

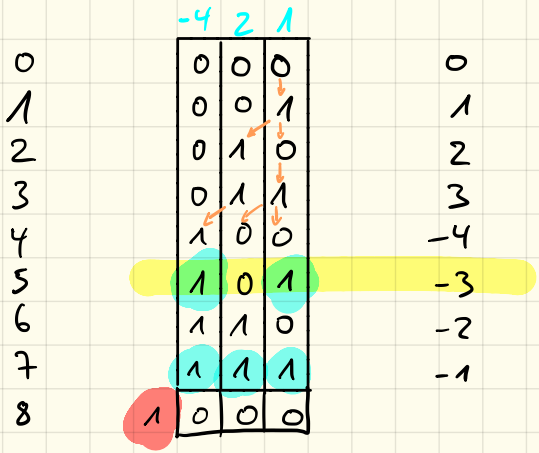


Kostenloser Struktogrammeditor unter http://www.whiledo.de/ downloadbar!

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

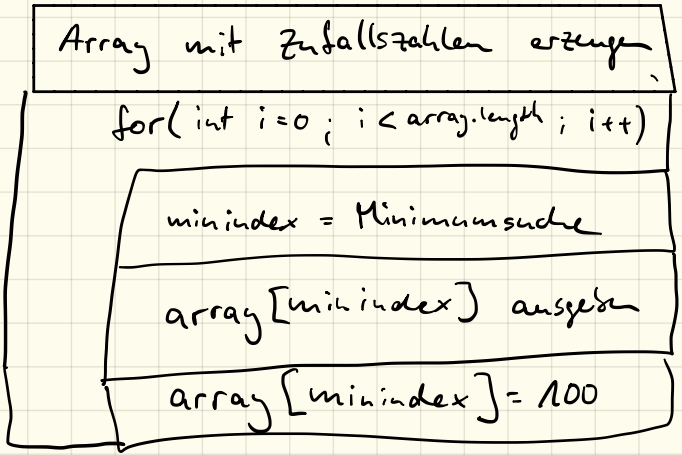
```
if (n % 2 == 0)   // n ist gerade
```

17.9.18

0 1 2 3 4 5 6

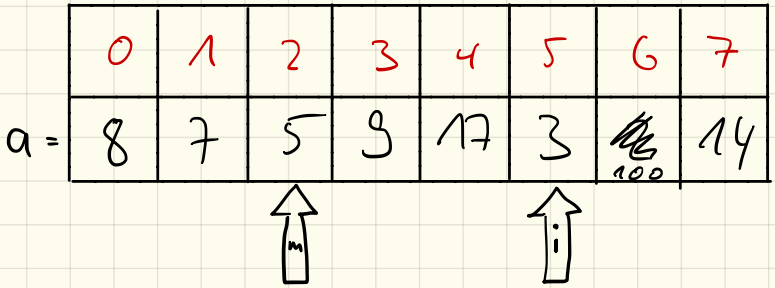
10, ~~7~~ 100, ~~100~~ 100, ~~100~~ 100, 9, 11, 17

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

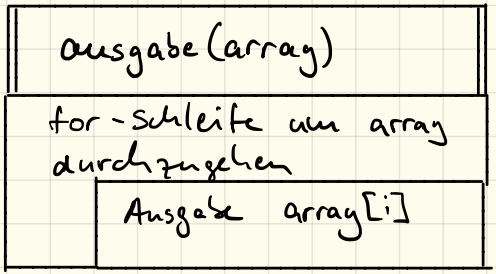
Ziel: `int[] array = zufall(20, 50);`

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);`

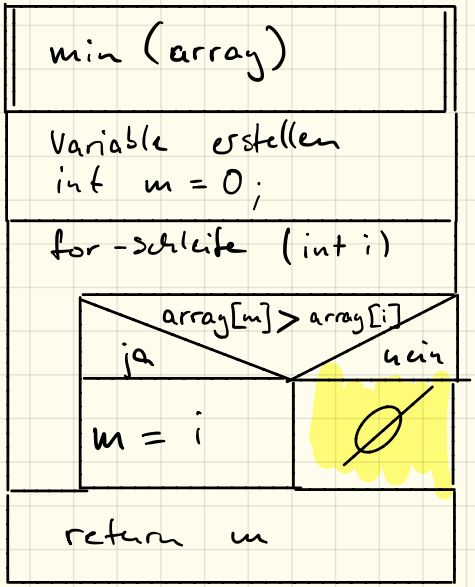



```
public static void ausgabe(int[] array){
    for(int i=0; array.length; i++){
        System.out.print(array[i]);
    }
}
```

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

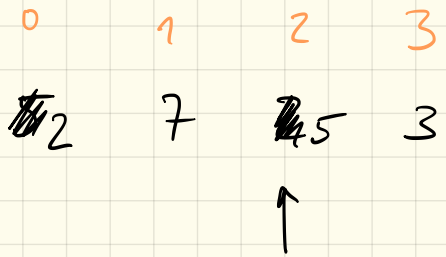
Beschreibe die beiden folgenden Begriffe im Hinblick auf Sortierverfahren:

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

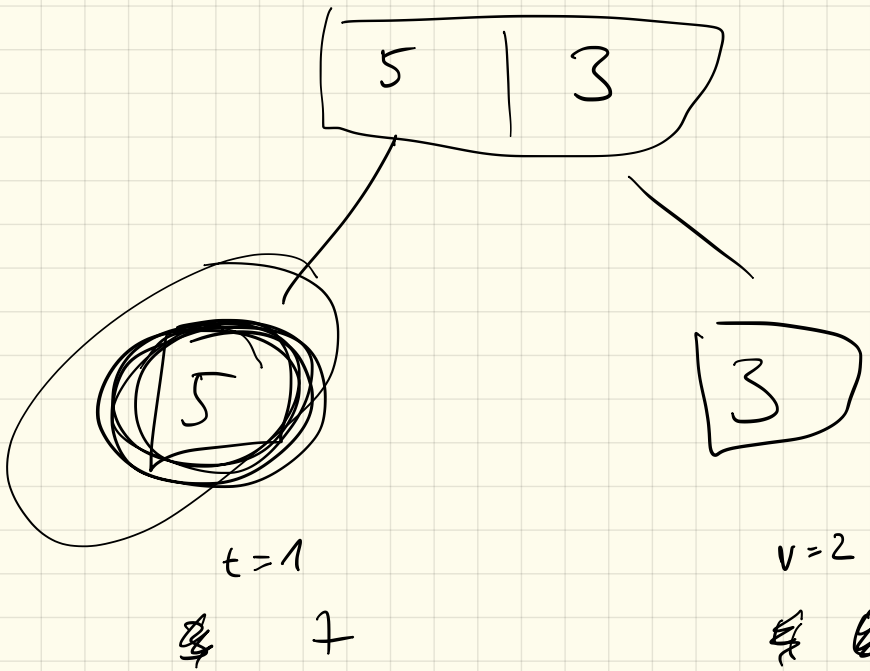
d-a-c: Aufteilen: 1002 $n + 2$

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



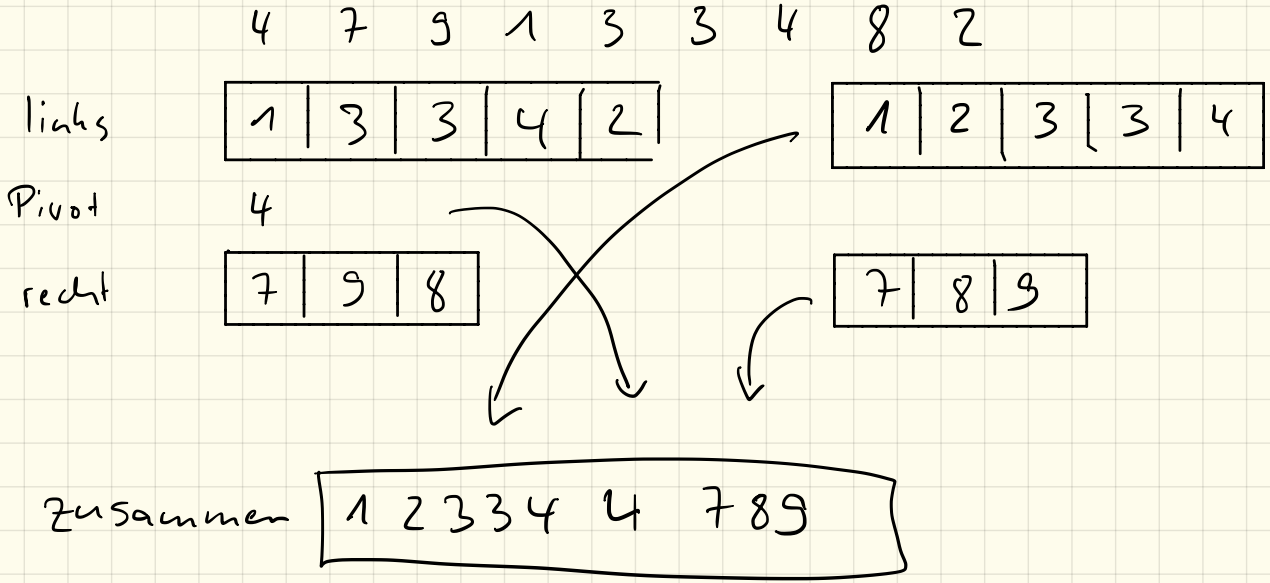
k = Mergesort(k)
g = Mergesort(g)

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

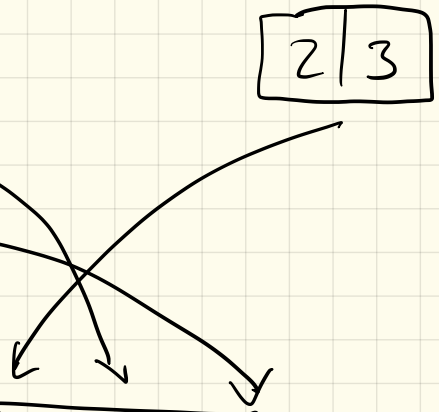
1	2	3	3	4

3 3 4 2

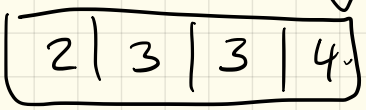
links 3 2

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

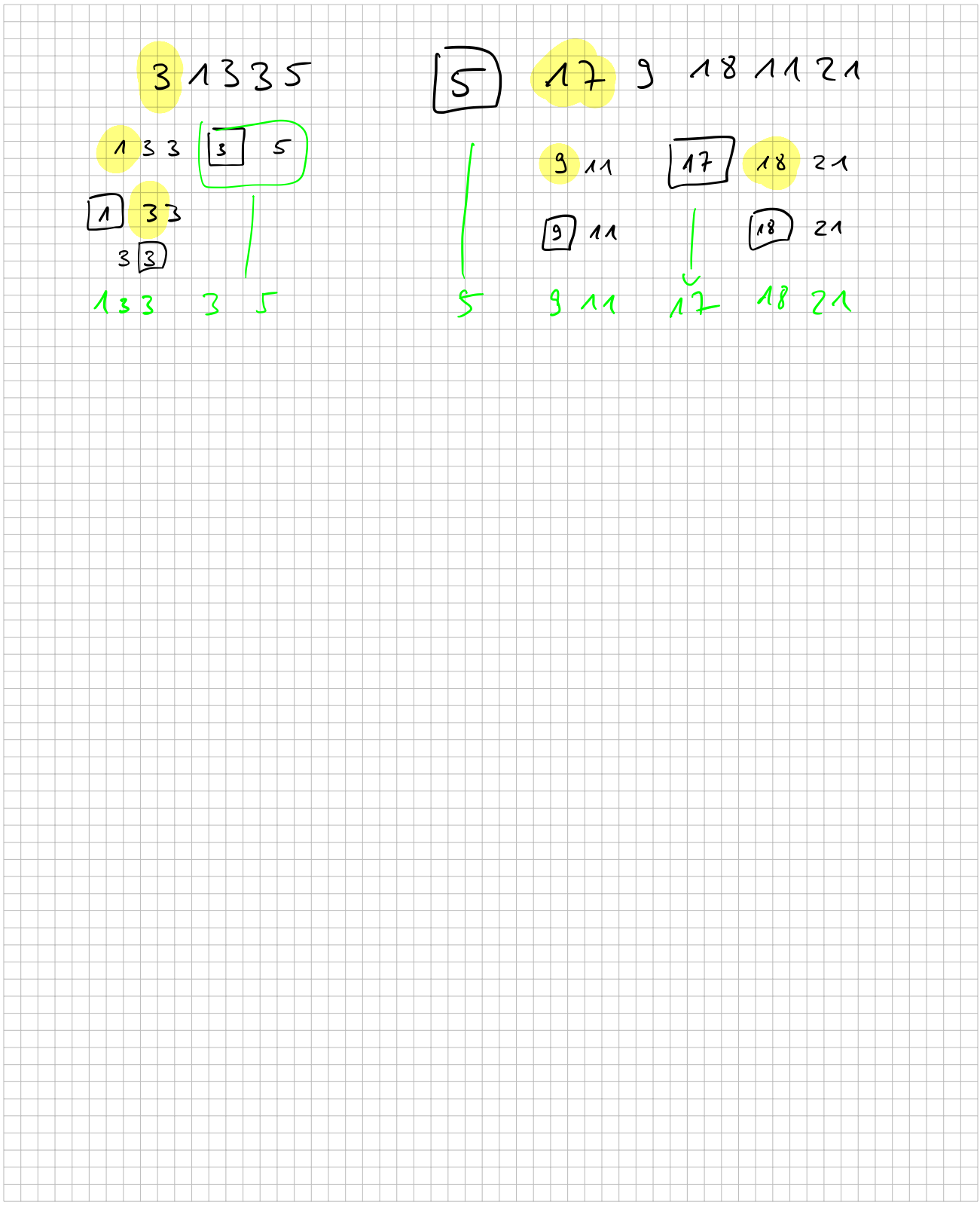
Liste 2: 50 $\xrightarrow{\text{Selection}}$ 2500

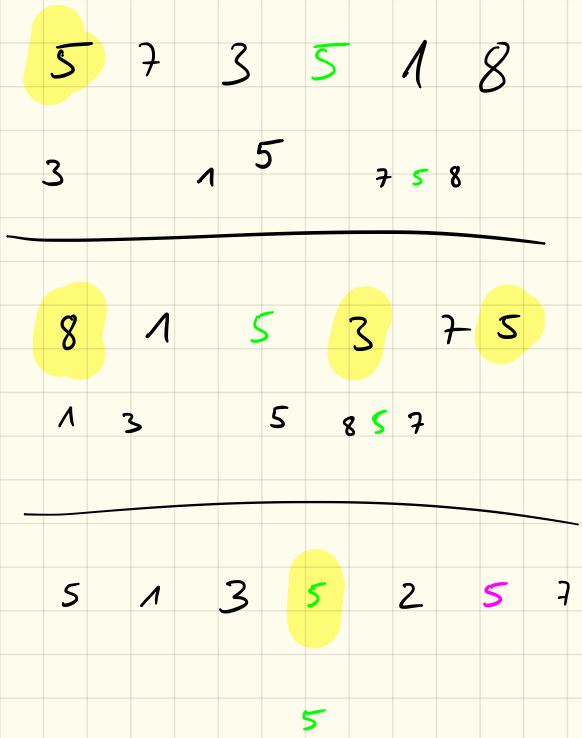
5000 Schritte

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

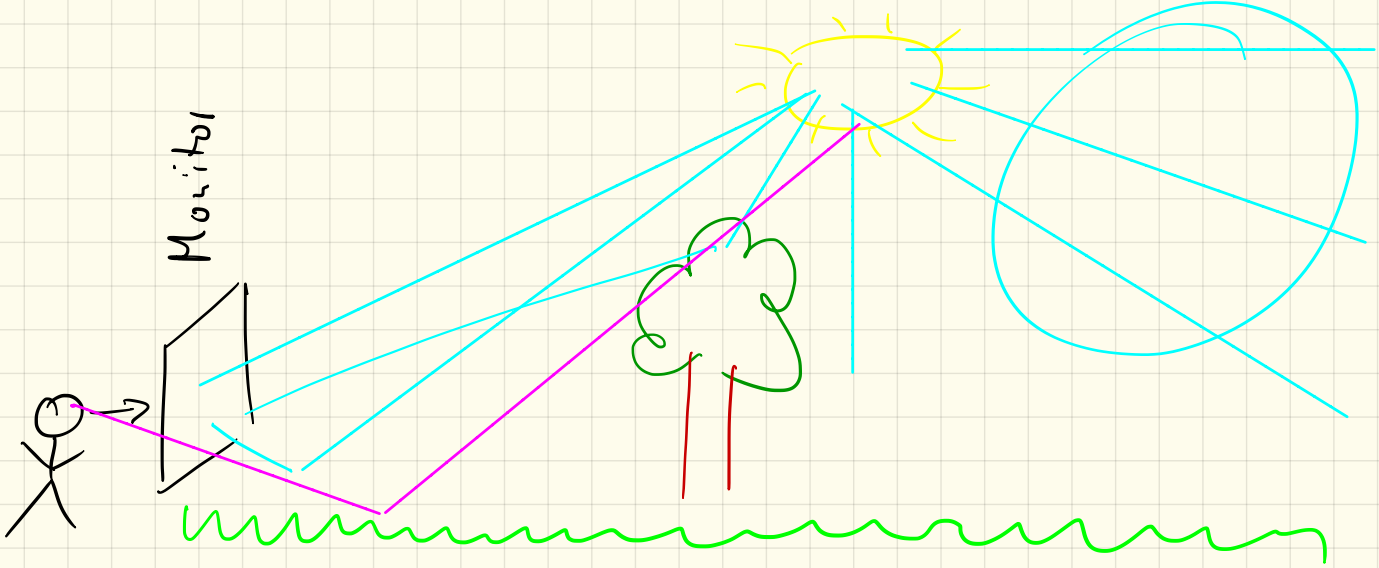
5 → 1 Stelle
4 → 2 Stellen
3 → 3 Stellen
⋮
n-1 Stellen

$\frac{n-1}{2}$ 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

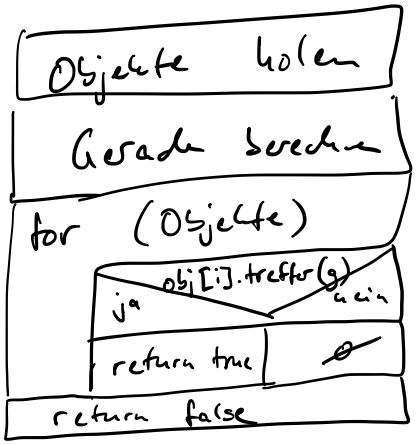
um alle in der Szene befindlichen Objekte in einem Array zu speichern.
Für jedes Objekt `obj[i]` gibt es dann widerum eine Methode `treffer(Gerade g)`, welche testet, ob die Gerade g das Objekt schneidet und dann einen entsprechenden `boolean`-Wert zurückgibt.
Die Gerade g erhalten wir widerum über die 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

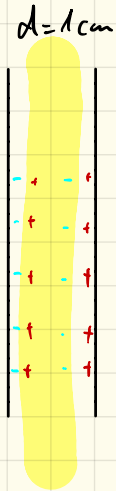
Ziel wird es sein, die Methode `tr.trifft(int x,int y)` so nachzubilden, dass wir diese später erweitern können.
Lege dazu im `OOP`-Projekt in der `Start`-Klasse eine neue Methode `trifft` an:

- 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

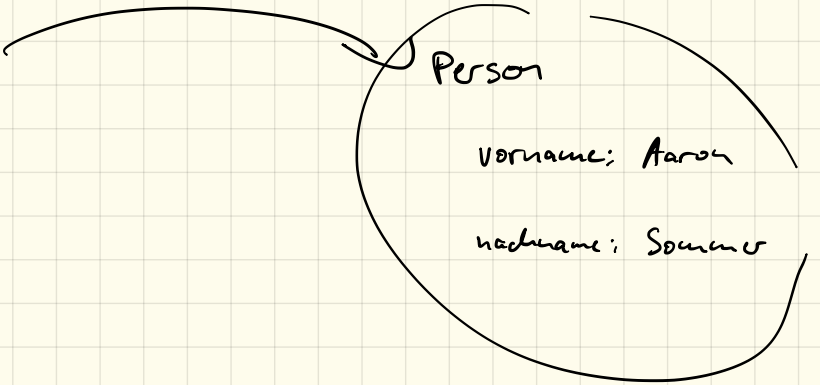
boolean: männlich

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?

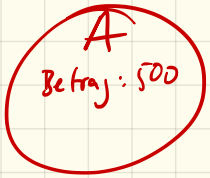
a)

Bankkonto

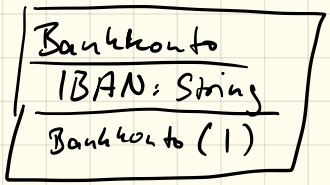
IBAN: String
BIC: String
Inhaber: String
Betrag: double

b

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

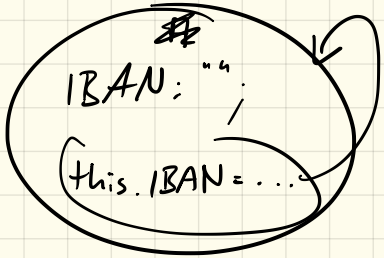


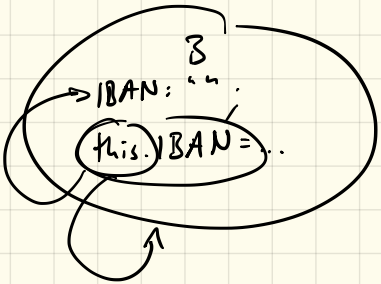
A.Ueberweisen(100, B);



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