0–13

Kompaktversion Tutorien

Von pinguinreduzierten Tutorien 0–13

Florian Sihler ○ 1. November 2022

Diese ***Kompaktversion*** ist dazu gedacht die wichtigsten Kommentare und Lösungen zu sammeln. Sie ist allerdings ***ohne jede Garantie auf Vollständigkeit*** aufzufassen. Zusätzliche Inhalte finden sich in den Folien zu den einzelnen Tutorien.

Liebe Grüße, Flo

Eine Sammlung — schnief — ohne Liebe

1. Organisatiorisches Tutorium
2. Blatt 0 — Hello World
3. Blatt 1 — Zahlen & Algorithmen
4. Blatt 2 — Raten, Typen & Booleans
5. Blatt 3 — Programmfluss & Schleifen
6. Blatt 4 — Schleifen, Vokale & Prim
7. Blatt 5 — Arrays, Matrizen & Vektoren
8. Blatt 6 — Methoden & Tic-Tac-Toe
9. Blatt 7 — Überladen & Überschatten
A. Blatt 8 — Klassen & Taxis
B. Blatt 9 — Rekursion
C. Blatt 10 — Sortierverfahren
D. Blatt 11 — Stacks und Buffer
E. Blatt 12 — Bäume und Minimax
F. Blatt 13 — Vererbung

Organisatiorisches Tutorium

KW 43

Hygienekonzept

- Wir machen ~~5G~~ **3G**
  - Geimpft
  - Genesen
  - Getestet
  - Genmanipuliert & Gechipt
  - Gelangweilt
- Präsenzlehre so lange wie möglich. (Sonst *Zoom*)
- Abstand einhalten!

Zu den Folien und Tutorien

- Folien von Florian Sihler (florian.sihler@uni-ulm.de).
- Mitschriften, Zusammenfassungen, … finden sich im Cloudstore:
  - https://cloudstore.uni-ulm.de/apps/circles/
  - Kreis: „Mitschriebe Informatik“ (ganz eingeben)
- Bei erbrachter Leistung: 3 ECTS (LP)

Übungsblätter

- Mindestens 50 % der Punkte pro Blatt
- Teilnahme am Tutorium verpflichtend für Punkte
- Maximal zwei Blätter nicht bestanden
  - Bonusaufgaben auf sechs Blättern
  - Absolviert man mindestens die Hälfte → Ein Pufferblatt mehr
- Abgabe in zweier Gruppen  
- Erlaubte Formate:
  - Bei einer Datei: PDF, Java-Quellcode
  - Bei mehreren: Archiv verwenden! (ZIP, …)
- Programme *müssen* lauffähig sein (inklusive `main`).
  (Oder entsprechend kommentiert, wo es (wieso) Probleme gibt.)

Übungsblätter II

- *Jeder* muss die Abgabe vorstellen können.
  (Unabhängig davon, wer die Aufgabe im Team gemacht hat.)
- Im Falle einer Krankheit:
  Nachricht an den Tutor mit zugehöriger Entschuldigung!

Lest die FSPO!
Verwendet die Uni-Mail!


Motivation!


Motivation!

Blatt 0 — Hello World

KW 43

Präsenzaufgabe

1 **Schau mal, ich bin schon groß!**

Entwickeln Sie einen Algorithmus, der als Eingabe eine Liste von Zahlen erhält. Anschließend soll er für jedes Element der Liste bestimmen, ob nach diesem Element noch eine größere Zahl in der Liste folgt.
Beispiel mit der Liste „1, 0, 7, 5, 2“:

1. **ja** (da $1 < 7$)
2. **ja** (da $0 < 7$)
3. **nein** (da $7 > \max\{5, 2\}$)
4. **nein** (da $5 > 2$)
5. **nein** (da letztes Element)

Welche Laufzeit hat Ihr Algorithmus im schlechtesten Fall?

Präsenzaufgabe - Lösung

Input : Eingabe als Liste von n Zahlen: $a_1, \ldots, a_n$

```
1  for i = 1 to n do
2      gefunden ← falsch;
3      for j = i+1 to n do // Wird für j > n nicht betreten.
4          if a_j > a_i then
5              gefunden ← wahr;
6              break;
7          end
8      end
9      if gefunden then Gebe aus: „ja“ else Gebe aus: „nein“ ;
10 end
```

Algorithmus 1 : Naiver Ansatz

Nach kleinem Gauß ergibt sich die Laufzeit:

$$c \cdot \frac{n(n+1)}{2} \in \mathcal{O}(n^2).$$

Präsenzaufgabe - Effizientere Lösung

Für eine optimierte/bessere Laufzeit ($\mathcal{O}(n)$):

Input : Eingabe als Liste von n Zahlen: $a_1, \ldots, a_n$
1 Erstelle Liste an n Wörtern: $w_1, \ldots, w_n$;
2 $max \leftarrow -\infty$;
3 **for** *i = n* **to** *1* **do**
4 | **if** $max > a_i$ **then** $w_i \leftarrow$ „ja“ **else** $w_i \leftarrow$ „nein“ ;
5 | **if** $a_i > max$ **then** $max \leftarrow a_i$;
6 **end**
7 **for** *i = 1* **to** *n* **do**
8 | Gebe aus: w_i;
9 **end**

Algorithmus 2 : Bezüglich der Laufzeit besserer Ansatz

Übungsblatt 0 - Aufgabe 1

- In der Konsole: `java -version`:

```
openjdk version "11.0.17" 2022-10-18
OpenJDK Runtime Environment (build 11.0.17+8-alpine-r0)
OpenJDK 64-Bit Server VM (build 11.0.17+8-alpine-r0, mixed mode)
```

- Sowie: `javac -version`:

```
javac 11.0.17
```

Übungsblatt 0 - Aufgabe 2

- Tipp, tipp, tipp, …

```
1 class Hello {
2     public static void main(String[] args) {
3         System.out.println("Hello World!");
4     }
5 }
```

- Sowie: `javac Hello.java`
- Und dann: `java Hello`.

Blatt 1 — Zahlen & Algorithmen

KW 44

Präsenzaufgabe

2 **Kommen Sie hier rein?**

Konstruieren Sie für jede der folgenden Aussagen einen booleschen Ausdruck, welcher diese überprüft.

1. Eine Person ist Teenager. (`int alter` in Jahren)
2. Es ist Vormittag. (`int uhrzeit` im 24h-Format)
3. Eine Tür ist geschlossen. (`boolean istOffen`)
4. Im Kaffee ist entweder Zucker *oder* Milch, aber nicht beides.
   (`boolean zucker` und `boolean milch`)

Präsenzaufgabe - Lösung

- Teenager: `alter >= 13 && alter <= 18` (oooder 19?)
- Vormittag: `uhrzeit >= 9 && uhrzeit <= 12`
- Tür: `!istOffen` (But who trusts the name?)
- (Guter?) Kaffee: `(zucker || milch) && !(zucker && milch)`, oder: `zucker ^ milch`.

^ entspricht dem XOR-Operator. Dieser evaluiert nur genau dann zu wahr, wenn einer der Parameter wahr und der andere falsch ist.

A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0

Übungsblatt 1 - Aufgabe 1 a)

- Hornerschema am Beispiel von $1101_{(2)}$:

$$(((1 \cdot 2 + 1) \cdot 2 + 0) \cdot 2) + 1 = (((3) \cdot 2 + 0) \cdot 2) + 1 = (6 \cdot 2) + 1 = 13_{(10)}.$$

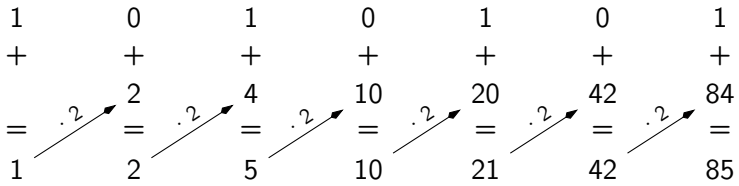
- $1010101_{(2)} = (((((1 \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1 = 85_{(10)}$
- $12345_{(8)} = ((((1 \cdot 8 + 2) \cdot 8 + 3) \cdot 8 + 4) \cdot 8 + 5) = 5349_{(10)}$
- Hexadezimal:

$1_{(16)} \mathrel{\widehat{=}} 1_{(10)}$
$2_{(16)} \mathrel{\widehat{=}} 2_{(10)}$
$3_{(16)} \mathrel{\widehat{=}} 3_{(10)}$
$4_{(16)} \mathrel{\widehat{=}} 4_{(10)}$
$5_{(16)} \mathrel{\widehat{=}} 5_{(10)}$
$6_{(16)} \mathrel{\widehat{=}} 6_{(10)}$
$7_{(16)} \mathrel{\widehat{=}} 7_{(10)}$
$8_{(16)} \mathrel{\widehat{=}} 8_{(10)}$
$9_{(16)} \mathrel{\widehat{=}} 9_{(10)}$
$A_{(16)} \mathrel{\widehat{=}} 10_{(10)}$
$B_{(16)} \mathrel{\widehat{=}} 11_{(10)}$
$C_{(16)} \mathrel{\widehat{=}} 12_{(10)}$
$D_{(16)} \mathrel{\widehat{=}} 13_{(10)}$
$E_{(16)} \mathrel{\widehat{=}} 14_{(10)}$
$F_{(16)} \mathrel{\widehat{=}} 15_{(10)}$

- $FACE_{(16)} = ((15 \cdot 16 + 10) \cdot 16 + 12) \cdot 16 + 14 = 64206_{(10)}$

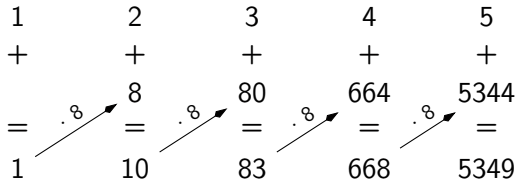
Übungsblatt 1 - Aufgabe 1 a)

- Wir können die Rechnungen auch tabellarisch veranschaulichen! Zur Erinnerung: $1010101_{(2)} = (((((1 \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1 = 85_{(10)}$.



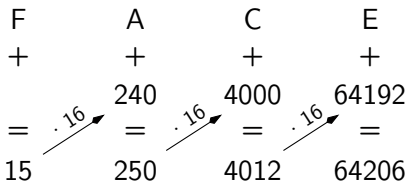
Übungsblatt 1 - Aufgabe 1 a)

- Analog: $12345_{(8)} = ((((1 \cdot 8 + 2) \cdot 8 + 3) \cdot 8 + 4) \cdot 8 + 5) = 5349_{(10)}$.



Übungsblatt 1 - Aufgabe 1 a)

- Sowie: $\text{FACE}_{(16)} = ((15 \cdot 16 + 10) \cdot 16 + 12) \cdot 16 + 14 = 64206_{(10)}$.



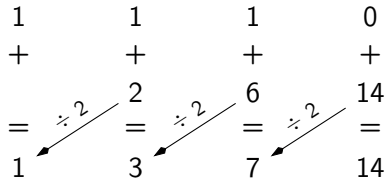
- $14_{(10)}$ zur Basis 2. Eine Vorbemerkung:

$$\begin{aligned} 14 \div 2 &= 7 & 14 \bmod 2 &= 0 \quad (\leftarrow \text{LSB}) \\ 7 \div 2 &= 3 & 7 \bmod 2 &= 1 \\ 3 \div 2 &= 1 & 3 \bmod 2 &= 1 \\ 1 \div 2 &= 0 & 1 \bmod 2 &= 1 \quad (\leftarrow \text{MSB}) \end{aligned}$$

- Damit gilt: $1110_{(2)} = (((1 \cdot 2 + 1) \cdot 2 + 1) \cdot 2 + 0 = 14_{(10)}$.
- *Hinweis: Für* b2 *genügt der Test auf gerade oder ungerade.*
- Dieses Art der Darstellung ist aber *nicht* das Hornerschema!

Übungsblatt 1 - Aufgabe 1 b)

- Natürlich geht dies auch tabellarisch. Zur Erinnerung:
  $1110_{(2)} = (((1 \cdot 2 + 1) \cdot 2 + 1) \cdot 2 + 0 = 14_{(10)}$.



Übungsblatt 1 - Aufgabe 2 a) & b)

a) Problemspezifikation:

- *Tagespreis:* positive reelle Zahl. Dargestellt mit zwei Nachkommastellen.
- *Tagespreisliste:* chronologisch sortierte Liste an Tagespreisen $[p_1, \ldots, p_n]$ mit $n \in \mathbb{N}_1$.
- *Tag:* natürlicher Index in Tagespreisliste.
- *Zwei aufeinanderfolgende Tage:* zwei aufeinanderfolgende Listenindizes: i und $i+1$ wobei $i, i+1 < n$.
- *Preisanstieg:* positive Veränderung einer reellen Zahl (dem Tagespreis): $p_{i+1} - p_i > 0$.
- *maximaler prozentualer Preisanstieg:* größter relativer Preisanstieg: $\max_{1 \leq i < n} \frac{p_{i+1} - p_i}{p_i} \cdot 100$.

Wichtig: Wir sind hier nicht in einer Programmiersprache: Typen wie „`int`“ gibt es nicht $\Rightarrow$ mathematische Notation benutzen oder eigenen Typ definieren (wie Tupel, …)!

b) Problemabstraktion:

- *Gegeben:* Chronologisch sortierte Liste $[p_1, \ldots, p_n]$ positiver reeller Zahlen. Annahme: $n > 1$ („letzte Monate“).
- *Gesucht:* Positive reelle Zahl $m = 100 \cdot \max_{1 \leq i < n} \frac{p_{i+1} - p_i}{p_i}$ ($X\,\% \mathrel{\widehat{=}} \frac{X}{100}$).

Übungsblatt 1 - Aufgabe 2 c) & d)

c) Algorithmenentwurf:

(1) Setze $\Delta_{max} = 0$.

(2) Setze $i = 1$.

(3) Solange $(i < n)$, wiederhole:

Setze $test = (p_{i+1} - p_i)/p_i$.

Wenn $test > \Delta_{max}$: Setze $\Delta_{max} = test$.

Erhöhe i um 1.

(4) Ergebnis ist $100 \cdot \Delta_{max}$.

d) Korrektheitsnachweis, Verifikation:

1. Formale vollständige Induktion, oder:
2. „Textbasiert". i wächst streng monoton an, die Schleife wird damit genau $n - 1$ mal durchlaufen. Alle mathematischen Berechnungen terminieren per Konstruktion, ebenso die Zuweisungen. Der Algorithmus *terminiert*.

   Zudem ist er *partiell korrekt*. Die maximale prozentuale Änderung ist immer größte relative Differenz für alle $[p_1, \ldots, p_{i+1}]$ im i-ten Durchlauf. Nach $n - 1$ Durchläufen: $[p_1, \ldots, p_n]$. Basisfall mit $i = 2$, für Schritt $i \to i + 1$.

Übungsblatt 1 - Aufgabe 2

Notizen

- Totale Korrektheit erfordert zwei Komponenten!
  1. *Terminiertheit*: Der Algorithmus terminiert für jede definierte Eingabe.
  2. *Partielle Korrektheit*: Der Algorithmus liefert für jede definierte Eingabe ein korrektes Ergebnis, sofern er terminiert.
- Es reicht in der Regel nicht aus:
  - „Maximum ist sicher größer als alle betrachteten Alternativen“. Es muss dann zum Beispiel auch gezeigt werden, dass alle relevanten Alternativen in Betracht gezogen werden.
  - „Der Algorithmus findet das gesuchte Ergebnis“. Wir Beweisen zwar (noch) nicht ganz formal. Dennoch sollte ein ausreichendes Verständnis für den Beweis gezeigt werden.
- Vollständige Induktion über i (meist bei Schleifen):
  - Achtet darauf klar anzugeben, über was die Induktion läuft (Länge der Einladungs-, Bekannten- oder Bedingungsliste? Das Durchschnittsalter der Person?)
  - Achtet auf eine vollständige Behauptung. Ist die unvollständig, ist es auch der Prozess an sich.
  - Induktionsanfang: Wir zeigen, dass es für ein $i \in \mathbb{N}$ gilt, in der Regel $i = 0$ oder $i = 1$.
  - Induktionshypothese: Die Behauptung gilt für (ein) n.
  - Induktionsschritt: Gilt die Behauptung für n, so gilt sie auch für $n + 1$. (Hier wird fast immer die Hypothese benutzt um den Fall $n + 1$ auf n zurückzuführen.)

Übungsblatt 1 - Aufgabe 2 d)

(1) Setze $\Delta_{max} = 0$.
(2) Setze $i = 1$.
(3) Solange ($i < n$), wiederhole:
    Setze $\texttt{test} = (p_{i+1} - p_i)/p_i$.
    Wenn $\texttt{test} > \Delta_{max}$: Setze $\Delta_{max} = \texttt{test}$.
    Erhöhe i um 1.
(4) Ergebnis ist $100 \cdot \Delta_{max}$.

Wir zeigen die partielle Korrektheit und nehmen die Termination hier als gegeben.

Für ein beliebiges aber festes $n \in \mathbb{N}$ mit $n \geq 2$ und der Eingabe $p_1, \ldots, p_n$ zeigen wir per vollständiger Induktion über i, dass in jedem Schritte gelte: $\Delta_{max} \geq \max_{1 \leq j \leq i}((p_{j+1} - p_j)/p_j)$.

IA: Mit $i = 1$ ist $\Delta_{max} = \frac{p_2 - p_1}{p_1}$ das triviale Maximum ($1 \leq j \leq 1$).

IH: Es gilt $\Delta_{max} \geq \max_{1 \leq j \leq i}((p_{j+1} - p_j)/p_j)$ für i.

IS: Wir zeigen, dass auch $\Delta_{max} \geq \max_{1 \leq j \leq i+1}((p_{j+1} - p_j)/p_j)$ gilt. Das heißt, aktuell ist $\Delta_{max} \geq \max_{1 \leq j \leq i}((p_{j+1} - p_j)/p_j)$ und in der weiteren Schleife gilt:

1. $\texttt{test} > \Delta_{max}$: Durch die Bedingung wird Δ_{max} aktualisiert. Es gilt: $\Delta_{max} = \texttt{test} \geq \max_{1 \leq j \leq i+1}((p_{j+1} - p_j)/p_j)$.
2. $\texttt{test} \leq \Delta_{max}$: Das neue Glied durch $i + 1$ ist kein neues Maximum. Es gilt: $\Delta_{max} \geq \max_{1 \leq j \leq i+1}((p_{j+1} - p_j)/p_j) \geq \texttt{test}$.

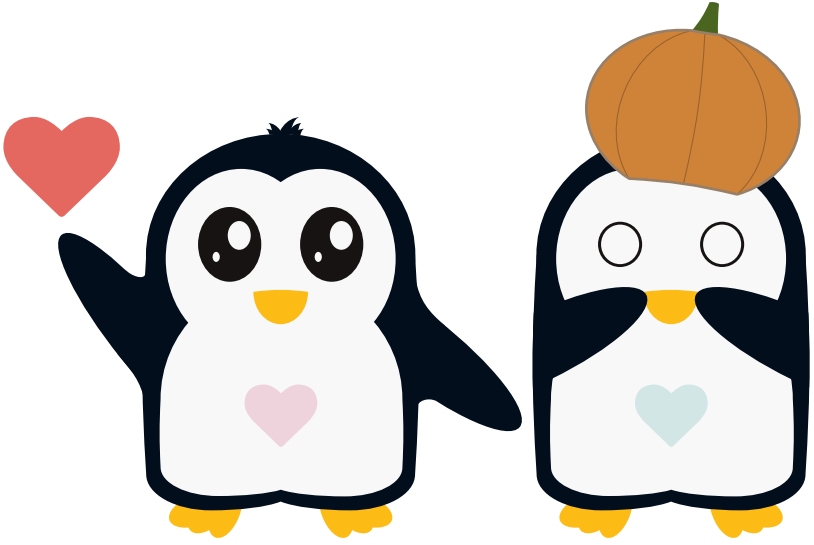
Nach $n - 1$ Schritten durch die Begrenzung $i < n$, wurden alle $p_1, \ldots, p_n$ betrachtet. Δ_{max} ist Maximum aller.

Übungsblatt 1 - Aufgabe 2 e)

(1) Setze $\Delta_{max} = 0$.
(2) Setze $i = 1$.
(3) Solange $(i < n)$, wiederhole:
    Setze $test = (p_{i+1} - p_i)/p_i$.
    Wenn $test > \Delta_{max}$: Setze $\Delta_{max} = test$.
    Erhöhe i um 1.
(4) Ergebnis ist $100 \cdot \Delta_{max}$.

e) Aufwandsanalyse:

1. Tabellarisch, siehe Vorlesung, oder:
2. „Textbasiert“. Wir betrachten Stückweise:
   - Die ersten Zuweisungen 1 & 2 sind exakt zwei Elementaroperation.
   - Die äußere Schleife in 3 wird genau $n - 1$ mal durchlaufen.
     - Erste Schleifenanweisung bedarf dreier Elementaroperationen: Subtraktion, Division & Zuweisung.
     - Zweite Schleifenanweisung ist ein Vergleich und maximal eine Zuweisung.
     - Dritte Schleifenanweisung ist eine Elementaroperation (je nach Definition auch zwei).
   - Das Endergebnis ist eine Multiplikation (und je nach Definition eine Zuweisung).
3. Damit erhalten wir: $2 + (n - 1) \cdot (3 + 2 + 1) + 2 + 2 = 6n - 2$ (also $\mathcal{O}(n)$).



Blatt 2 — Raten, Typen & Booleans

KW 45

Präsenzaufgabe

3

Ju can not desieve mie!

Entwickeln Sie ein Programm, welches als Eingabe eine Zahl N erhält und mit Hilfe des Sieb des Eratosthenes alle Primzahlen zwischen 1 und N bestimmt und ausgibt. Die Idee hier ist, alle Vielfachen der Zahlen, beginnend bei 2 zu markieren, also „herauszusieben“. Danach wird mit der nächsten unmarkierten Zahl fortgefahren. Hier ein grafisches Beispiel mit $N = 16$:

Präsenzaufgabe - Naive Lösung

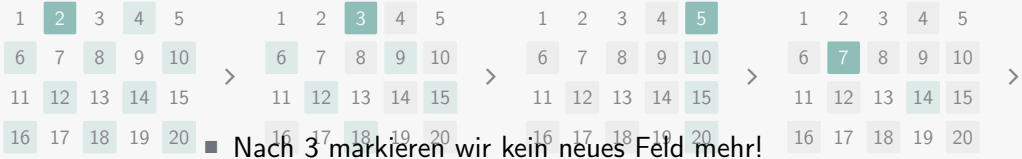
Input : Obergrenze $N \in \mathbb{N}$ (da Primzahlen)

```
1 marker = (marker_1, ..., marker_N);
2 for i = 1 to N do marker_i = false;
  // Siebe von 2 bis N
3 for i = 2 to N step 1 do
4 |   if marker_i then continue;
5 |   Gebe aus: „Prim: i“;
  |   // Markiere alle Vielfache
6 |   for j = 2 · i to N step i do marker_j = true;
7 end
```

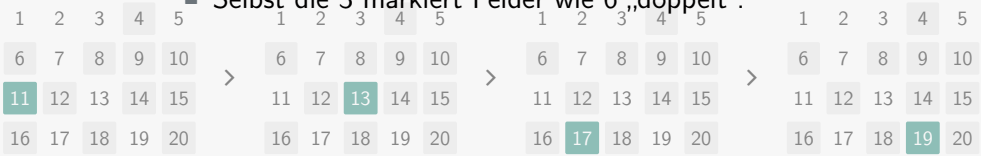
Algorithmus 1 : Naives Sieben

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90

Präsenzaufgabe - Was bemerken wir?



- Nach 3 markieren wir kein neues Feld mehr!
- Selbst die 3 markiert Felder wie 6 „doppelt“.



Präsenzaufgabe - Verbesserung

- Allgemeiner: Wir können beim Markieren bei i^2 anstelle von $2 \cdot i$ beginnen. Alle Vielfachen kleiner i^2 wurden bereits markiert!
- Damit können wir das Markieren bereits bei $\sqrt{N}$ stoppen.
- *Hinweis:* Im folgenden Pseudocode gehen wir davon aus, dass **for** nur ganzzahlige Schritte macht. Für „**for** $i = \sqrt{N} + 1$“ gehen wir also beispielsweise davon aus, dass der Algorithmus immer abrundet.

Präsenzaufgabe - Effizientere Lösung

Input : Obergrenze $N \in \mathbb{N}$

1 $\text{marker} = (\text{marker}_1, \ldots, \text{marker}_N)$;
2 **for** $i = 1$ **to** N **do** $\text{marker}_i = \text{false}$;
3 **for** $i = 2$ **to** $\sqrt{N}$ **step** *1* **do**
4 | **if** marker_i **then continue**;
5 | Gebe aus: „Prim: i“;
6 | **for** $j = i \cdot i$ **to** N **step** *i* **do** $\text{marker}_j = \text{true}$;
7 **end**
8 **for** $i = \sqrt{N} + 1$ **to** *N* **step** *1* **do**
9 | **if not** marker_i **then** Gebe aus: „Prim: i“;
10 **end**

Algorithmus 2 : Optimiertes Sieben

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90

Übungsblatt 2 - Aufgabe 1

- Wenn es heißt „dreistellige Dezimalzahl“, ist das *mit* führenden Nullen oder *ohne*?
- Wir nehmen zuerst an, dass führende Nullen *erlaubt* sind.
- Da Prozeduren als solche noch aus fernen Landen stammen, schreiben wir alles als ein Programm.

Übungsblatt 2 - Aufgabe 1 — Führende Nullen

```
1  Wähle zufällig z_1, z_2, z_3 ∈ {0, ..., 9};
2  for round = 1 to 3 step 1 do
3      guesses_left = 3;
4      while true do
5          Erbitte um Eingabe für Stelle ⌊round⌉, bei ⌊guesses_left⌉ Versuchen;
6          e = Eingabe von Nutzer mit e ∈ {0, ..., 9};
7          if e = z_round then verlasse while-Schleife;
8          else
9              guesses_left = guesses_left − 1;
10             if guesses_left ≤ 0 then
11                 Vermerke scheitern;
12                 return;
13 Verkünde Gewinn;
```

Algorithmus 3 : Rate mit führenden Nullen

Übungsblatt 2 - Aufgabe 1 — Ohne führende Nullen

- Wir wählen nun zufällig $z \in \{100, \ldots, 999\}$.
- Doch wie erhalten wir z_1, z_2 und z_3?
- Wir können beispielsweise Modulon™:

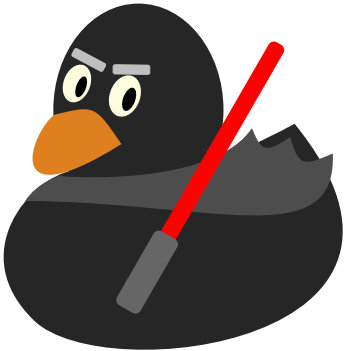
$$z_1 = \lfloor z/100 \rfloor \bmod 10 \quad z_2 = \lfloor z/10 \rfloor \bmod 10 \quad z_3 = z \bmod 10$$

- Der Rest bleibt unverändert.

Übungsblatt 2 - Aufgabe 1, ein paar Bitten

- Geht mittels verschiedener Simulationen durch den Pseudocode
- Ihr *wollt* beim Überprüfen des Pseudocodes, dass etwas nicht stimmt. (Es ist besser einen Fehler früh zu finden als sich irgendwann im Betrieb zu Fragen warum die (Mars-)Rakete auf die Sonne zu steuert.)
- Pseudocode ist nicht nur was, was in den Augen des Tutors bestehen muss!
- Nutzt Schleifen und vermeidet Redundanzen wo möglich. Bei einer Änderung sollte und darf „Alles Ersetzen“ *niemals* das Mittel der Wahl sein.
- Hardcoding löst das Problem nicht. So wird für die Aufgabenstellung: „Geben Sie von den ganzen Zahlen aus Eins bis Zehn die aus, die gerade sind“ *nicht* durch `System.out.println("2,4,6,8,10")` gelöst! (Der Compiler kann *loop-unrolling* betreiben, der Code würde dann entsprechend optimiert werden. Dies löst die Aufgabe allerdings nicht richtig!)

Übungsblatt 2 - Aufgabe 2



Die Ente ist knuffig, die bleibt 😊

Übungsblatt 2 - Aufgabe 2

- *Name einer Person*
  `String` der Name einer Person bedient sich der Welt der Buchstaben.
- *Akkustand eines Smartphone*
  Kleinstmöglich ist `byte`, das genügt für die übliche Prozentdarstellung ohne Nachkommastelle. `float` basierte Darstellungen mögen auch noch in Ordnung sein. `double` ist aber definitiv overkill (und dennoch nicht falsch).
- *Preis eines Produktes*
  Wenn wir damit nicht rechnen müssen: `float` oder `double`.
- *Ergebnis unbenoteter Prüfung*
  Da wir so nur bestanden und nicht bestanden haben, genügt ein `boolean`.

Übungsblatt 2 - Aufgabe 2

- *Raumtemperatur*
  In der Hoffnung, dass wir von Temperaturen sprechen in denen Menschen noch überleben können, sollte ein `float` absolut ausreichen.
- *Satzzeichen*
  Alle gängigen (und sogar ungängigen 😄) Satzzeichen sind als ASCII bzw. Unicode kodiert: ein `char` reicht in Java aus.

Übungsblatt 2 - Aufgabe 3 a)

I $(22 >= 21)\ \&\&\ (42 > 21) \implies true\ \&\&\ true \implies true.$

II $c\ \hat{}\ d \implies true\ \hat{}\ false \implies true.$

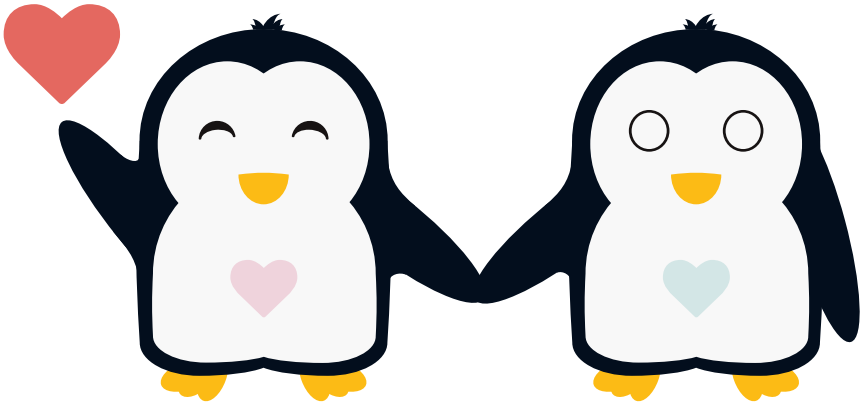
III $((42 - 21) == 21)\ \&\&\ !true \implies true\ \&\&\ false \implies false.$

IV $(42 == 2 * 21)\ \&\&\ (!true\ ||\ !false) \implies true\ \&\&\ true \implies true$

```
int a = 42; int b = 21;
boolean c = true;
boolean d = false;

if((22 >= b) && (a > b)) { // I
  if((c || d) && !(c && d)) { // II
    if(((a - b) == 21) && !c) { // III
      if((a == 2*b) && (!c || !d)) { // IV
        System.out.println("A");
      }
    } else {
      System.out.println("B");
    }
  } else {
    System.out.println("C");
  }
} else {
  System.out.println("D");
}
```

Übungsblatt 2 - Aufgabe 3 b)

- Nun, mit `true`, `true`, `false`, `true` verlassen wir bereits bei der dritten Unterscheidung das Konstrukt mit `B`.



Blatt 3 — Programmfluss & Schleifen

KW 46

Präsenzaufgabe

4 **Starring Mr. Pyramid**

Schreiben Sie ein Java-Programm, welches als Eingabe eine positive ganze Zahl N erwartet und daraufhin ein gleichseitiges Dreieck mit Hilfe von $*$- und Leerfeld-Charakteren in der Kommandozeile ausgibt. Die Ausgaben für $1 \leq N \leq 5$ sollten folgendermaßen aussehen:

Hinweis: Überlegen Sie sich eine allgemeine Formel für die Zeileneinrückung. Fügen Sie außerdem Leerzeichen zwischen $*$-en für die alternierenden Positionen in den Reihen ein.

Präsenzaufgabe

- Die ite Zeile ist immer um N – i Positionen eingerückt.
- Wir erschaffen ein Grundgerüst:

```
public class Pyramid {
    public static void main(String[] args) {
        if(args.length != 1)
            System.exit(1);
        int N = Integer.parseInt(args[0]);
        // Oder auch gerne 'n'

        // Pyramide Erschaffen +2
    }
}
```

Präsenzaufgabe

- Nun basteln wir uns eine Pyramide:

```
for (int i = 1; i <= N; i++) {
    for (int j = 1; j <= N - i; j++) {
        System.out.print("␣");
    }
    for (int j = 1; j <= i; j++) {
        System.out.print("*␣"); // zusätzliche Leerzeichen
    }
    System.out.println(); // Zeilenumbruch
}
```

- Damit ist alles komplett (Pyramid.java).

Präsenzaufgabe die Zweite

5

Code! Code! Code! Repeat.

Drücken Sie die folgenden Anweisungen durch äquivalente Schleifen der angegeben Art aus! Die Variable `int k` sei jeweils mit einem (beliebigen) Wert initialisiert.

a) Do-While:

```
for (int i = 1; i < k; ++i) {
    System.out.println(i * i);
}
```

b) For-Schleife:

```
int x = 0, i = 0;
while(i < k) { x += k * ++i; }
System.out.println("x: " + x);
```

c) For-Schleife:

```
if(k > 0){
    int i = 1, m = 0;
    while(i < k){
        if(k*i > m) m = k+i;
        ++i;
    }
    System.out.println("m: " + m);
}
```

Präsenzaufgabe 2 - Lösung

a) Wir erhalten (rechts):

```
for (int i = 1; i < k; ++i) {
    System.out.println(i * i);
}
```

```
if(k > 1) {
    int i = 1;
    do {
        System.out.println(i * i);
        ++i;
    } while (i < k);
}
```

b) Wir erhalten (rechts):

```
int x = 0, i = 0;
while(i < k) { x += k * ++i; }
System.out.println("x: " + x);
```

```
int x = 0;
for(int i = 0; i < k; i++)
    x += k * (i + 1);
System.out.println("x: " + x);
```

Präsenzaufgabe 2 - Lösung

c) Wir erhalten (rechts):

```
if(k > 0){
    int i = 1, m = 0;
    while(i < k){
        if(k*i > m) m = k+i;
        ++i;
    }
    System.out.println("m: " + m);
}
```

```
if(k > 0){
    int m = 0;
    for(int i = 1; i < k; ++i)
        if(k*i > m) m = k+i;
    System.out.println("m: " + m);
}
```

Übungsblatt 3 - Aufgabe 1 a)

- Uns reicht es, wenn dies nur für den zweiten `if-else`-Block geschieht.
- Damit wird dieser zu:

```
switch (x) {
    case 1: System.out.println("A");
        break;
    case 2: System.out.println("B");
        break;
    case 3: System.out.println("C");
        break;
    default: System.out.println("D");
        break;
}
```

Übungsblatt 3 - Aufgabe 1 a)

- Fanatiker dürfen aber auch die Kommandozeilenargumente auf diese Weise überprüfen (SwitchCaseControlled.java):

```
switch(args.length) {
    case 1: x = Integer.parseInt(args[0]); break;
    default: System.exit(1); return;
}
```

- Ist das `switch` hier sinnvoll? (Nicht wirklich. Mindestens drei Fälle sollten es schon sein, damit sich der Schreibaufwand auch reduziert beziehungsweise die Lesbarkeit verbessert.)
- Ist das `return` hier notwendig? (Nein, die JVM bricht durch das `exit` ab.)
- Warum könnte man `return` eventuell doch brauchen? (Theoretisch muss sie da nicht enden und auch wenn der Code danach nicht mehr ausgeführt wird, der Java Compiler weiß in der Regel noch nicht, dass es bei `System.exit` wirklich vorbei ist.)

Übungsblatt 3 - Aufgabe 1 b)

- Die Bedingung `(x < 0) || (x >= 3)` invertiert den Default-Fall!
- Damit genügt (SwitchCaseControlledAlternate.java)

```
switch (x) {
    case 0: System.out.println("D"); break;
    case 1: System.out.println("A"); break;
    case 2: System.out.println("B"); break;
    default:
        System.out.println("C");
}
```

Übungsblatt 3 - Aufgabe 2 a) While

- Zunächst mit While (FacultyWhileLoop.java):

```
int faculty = 1;
int i = 0;
while (i <= n) {
    if (i != 0) {
        faculty *= i;
    }
    System.out.println(i + "! = " + faculty);
    i++;
}
```

Übungsblatt 3 - Aufgabe 2 a) Do-While

- Non mit Do-While (FacultyDoWhileLoop.java):

```
int faculty = 1;
int i = 0;
do {
    if (i != 0) {
        faculty *= i;
    }
    System.out.println(i + "! = " + faculty);
    i++;
} while (i <= n);
```

Übungsblatt 3 - Aufgabe 2 b)

- „Sie möchten alle `n` Elemente einer Liste ausgeben.“

  Da wir hier das Maximum kennen und nacheinander auf alle Elemente zugreifen, empfiehlt sich eine `for`-Schleife.

- „Sie möchten solange einzelne Zeichen einlesen, bis ein ‚x‘ eingelesen wird.“

  Hier kennen wir nicht das Maximum. Da wir aber mindestens einmal einlesen müssen um auf „x“ zu prüfen, empfiehlt sich `do-while`. An sich ginge aber auch `while` wenn wir im Schleifenkopf einlesen.

Übungsblatt 3 - Zusatzaufgabe a)

- Jetzt können wir auch alles über Bord werfen.
- Da der Argumenttest nicht ternär gemacht werden kann, sind wir mal nicht fanatisch 😆 (EvenOddTernary.java):

```
System.out.println(x + ((x % 2) == 0 ? "␣ist␣gerade" :
                "␣ist␣ungerade"));
```

Übungsblatt 3 - Zusatzaufgabe b)

- So schlimm es auch ist, wir können die ternären Operationen verschachteln.
- Der Argumenttest geht aber immer noch nicht (DivisibleTernary.java):

```
String output = (x % 2) == 0 ? "␣ist␣durch␣2␣teilbar"
      : (x % 3) == 0 ? "␣ist␣durch␣3␣teilbar"
      : "␣ist␣weder␣durch␣2␣noch␣3␣teilbar";
```

Blatt 4 — Schleifen, Vokale & Prim

KW 47

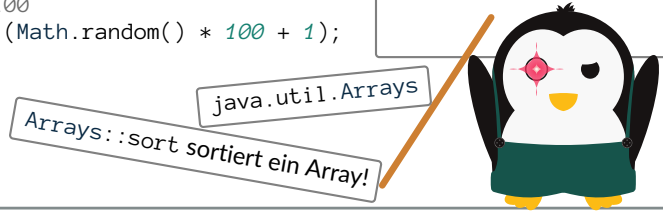
Präsenzaufgabe

6

Maximal Medium — Charakterlimit erreicht

```
public class Statistics {
    public static void main(String[] args) {
        // Zwischen 10 und 20
        int length = (int) (Math.random() * 11 + 10);
        int[] randoms = new int[arrayLength];

        for(int i = 0; i < length; i++) {
            // Zwischen 1 und 100
            randoms[i] = (int) (Math.random() * 100 + 1);
        }

        // 🐧 Platzhalterpingu
    }
}
```

Ersetzen Sie 🐧 mit Code zur Berechnung der folgenden statistische Kenngrößen für `randoms`:
1) Minimum, 2) Maximum,
3) Mittelwert und 4) Median.
Geben Sie sie aus.



Präsenzaufgabe - Lösung

- Der gesamte Code ist wieder hier: Statistics.java
- Sortieren macht das finden von Mini- und Maximum leicht:

```
Arrays.sort(randoms);

int min = randoms[0]; // first
int max = randoms[length-1]; // last
```

Präsenzaufgabe - Lösung

- Es geht mir mittelmäßig. Ich komm schon drüber weg…

```
double avg = 0.0;
for(int i = 0; i < length; i++) {
    avg += randoms[i];
}
avg /= length; // avg = avg / length;
```

Präsenzaufgabe - Lösung

- Der Median ist bei ungerade vielen Elementen das mittlere und sonst der Durchschnitt der mittleren Elemente:

```
double med = 0.0;
if(length % 2 == 1) {
    med = randoms[(length / 2)];
} else {
    med = randoms[(length / 2) - 1] + randoms[(length / 2)];
    med /= 2; // med = med / 2;
}
```

Präsenzaufgabe - Lösung

- Und nun der shout-out an die Fans:

```
System.out.println("Minimum: " + min);
System.out.println("Maximum: " + max);
System.out.println("Mittelwert: " + avg);
System.out.println("Median: " + med);
```

Abgabeformalitäten

- Java-Programmieraufgaben:
  - Nichts handschriftliches oder Pseudocode.
  - Nur `.java`-Dateien
  - (Teil-)Aufgabe in separaten `.java`-Dateien.
  - Formatierung nicht wie bei Hempels unterm Sofa.

  Sonst (über Blätter hinweg):
  1. Mal: –25% der Punkte der Teilaufgabe
  2. Mal: –50% der Punkte der Teilaufgabe
  3. Mal: –100% der Punkte der Teilaufgabe

- Sie müssen **immer** lauffähig sein.
  - Fehlerhaften Code auskommentieren
  - Anmerken, was das Ziel war.

  Sonst *null Punkte*.

Abgabeformalitäten II

- Textaufgaben:
  - Handschriftlich ist Ok, sofern lesbar.
  - LaTeX ist tsööööön (😲) aber **nicht verpflichtend**.
  - Wir öffnen nur `.java`, `.txt` und `.pdf` Formate.
- Allgemein
  - Sprecht mit anderen und tauscht eure **Ideen** aus.
  - Aber bitte nicht eure Lösungen.
  - Es soll übermotivierte Tutoren geben, die an Clone-Detection arbeiten…
    und die sich alle Abgaben und Bewertungen ansehen.

Übungsblatt 4 - Aufgabe 1

- Banale Vokale CountVowels.java

```
int a = 0, e = 0, i = 0, o = 0, u = 0;

for(int j = 0; j < input.length(); j++) {
    switch(input.toLowerCase().charAt(j)) {
        case 'a': a++; break;
        case 'e': e++; break;
        case 'i': i++; break;
        case 'o': o++; break;
        case 'u': u++; break;
    }
}
```

Übungsblatt 4 - Aufgabe 1

- Und die Ausgabe:

```
System.out.println("a: " + a);
System.out.println("e: " + e);
System.out.println("i: " + i);
System.out.println("o: " + o);
System.out.println("u: " + u);
```

Übungsblatt 4 - Aufgabe 1

Schweigender Bonus

```
int a = 0, e = 0, i = 0, o = 0, u = 0;

for(int j = 0; j < input.length(); j++) {
   switch(input.toLowerCase().charAt(j)) {
      case 'a': a++; break;
      case 'e': e++; break;
      case 'i': i++; break;
      case 'o': o++; break;
      case 'u': u++; break;
   }
}

System.out.println("a: " + a);
System.out.println("e: " + e);
System.out.println("i: " + i);
System.out.println("o: " + o);
System.out.println("u: " + u);
```

```
char[] arr = input.toLowerCase().toCharArray();
int[] counters = new int[26];

for(int j = 0; j < arr.length; j++) {
   char c = arr[j];
   if('a' <= c && c <= 'z')
      counters[c - 'a'] += 1;
}

for(char c : new char[]{'a', 'e', 'i', 'o', 'u'}) {
   System.out.println(c+": " + counters[c - 'a']);
}
```

Übungsblatt 4 - Aufgabe 2

- Isch bin ene Primeel: PrimeFactors.java.
- Starting with the guardians of the C:

```
if (args.length != 1) System.exit(1);

int N = Integer.parseInt(args[0]);
if (N < 1) {
    System.exit(1);
} else if (N == 1) {
    System.out.println("Die Zahl 1 besitzt keine Primfaktorzerlegung");
    return;
}
```

Übungsblatt 4 - Aufgabe 2

- Wir verwenden die Hinweise und Sieben:

```
boolean marker[] = new boolean[N + 1];
for (int i = 0; i <= N; i++)
    marker[i] = false; // Sicha ist Sicha!

for (int i = 2; i < Math.sqrt(N); i++) {
    if (!marker[i]) {
        for (int j = i * i; j <= N; j += i) {
            marker[j] = true;
        }
    }
}
```

Übungsblatt 4 - Aufgabe 2

- Und nun mit den Primlies verhackseln:

```
System.out.print("Die␣Primfaktorzerlegung␣von␣" + N + "␣lautet:␣");
while (N >= 2) {
    for (int i = 2; i < marker.length; i++) {
        if (marker[i]) continue;
        if ((N % i) == 0) {
            N /= i; // N = N / i;
            System.out.print((N >= 2) ? i + "␣*␣" : i);
            break; // Notwendig?
        }
    }
}
System.out.println();
```

Übungsblatt 4 - Aufgabe 2 — Alternativ

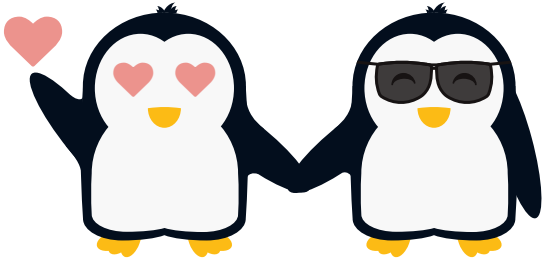
- Oder direkt wiederholt anwenden:

```
System.out.print("Die Primfaktorzerlegung von " + N + " lautet: ");
for (int i = 2; i < marker.length; i++) {
    if (marker[i]) continue;
    while ((N % i) == 0) {
        N /= i;
        System.out.print((N >= 2) ? i + " * " : i);
    }
}
System.out.println("");
```

Übungsblatt 4 - Aufgabe 2 — Die alternative Alternative

- Wir können das Sieb direkt integrieren… (PrimeFactorsAlternate.java)

```
System.out.print("Die Primfaktorzerlegung von " + N + " lautet: ");
int max = N;
for (int i = 2; i <= max; i++) {
    while ((N % i) == 0) {
        N /= i;
        System.out.print((N >= 2) ? i + " * " : i);
    }
}
System.out.println("");
```



Blatt 5 — Arrays, Matrizen & Vektoren

KW 48

Präsenzaufgabe, zum Auflockern

7 **Flippn' Amazing – Watch me Flip-Flip…**

Implementieren Sie die Methoden `public static void flipInPlace(char[] x)` und `public static char[] flipInCopy(char[] x)`, welche beide die Groß- und Kleinschreibung in den Arrays umkehren (nur ASCII). `flipInPlace` soll das übergeben Array modifizieren, `flipInCopy` soll `x` aber nicht verändern und eine „geflippte" Kopie zurückliefern.

Zum Erstellen einer Kopie können Sie `char[] copy = x.clone()` verwenden. Nutzen Sie aber nicht `Character::toUpperCase` oder vergleichbare Methoden, sondern machen Sie sich die Unicode-Kodierung von `char` zunutze (A–Z ist 65–90, a–z ist 97–122).

Beantworten Sie weiter:

1. Was sind die Vor- und Nachteile beider Ansätze?
2. Warum reicht keine einfache Zuweisung zur Kopie des Arrays?
3. Warum funktioniert kein for-each, wenn man die Einträge modifizieren möchte?

Präsenzaufgabe - Lösung

- We do se flippin place (mit `x` als `flipEm`):

```
public static void flipInPlace(char[] flipEm) {
  int shift = 'a' - 'A';
  for (int i = 0; i < flipEm.length; i++) {
    if ('A' <= flipEm[i] && flipEm[i] <= 'Z') {
      flipEm[i] += shift;
    } else if ('a' <= flipEm[i] && flipEm[i] <= 'z') {
      flipEm[i] -= shift;
    }
  }
}
```

Präsenzaufgabe - Lösung

- Und nun mit der Kopie! Wir könnten das ja nochmal schreiben…

```
public static char[] flipInCopy(char[] flipEm) {
    char[] copy = flipEm.clone();
    flipInPlace(copy);
    return copy;
}
```

Präsenzaufgabe - Lösung

- Vorteile einer Kopie (gegenüber in-place):
  - Nebeneffekte von Methoden sind generell schlecht! „out-Parameter“ sollten vermieden werden (wo nicht unbedingt notwendig).
- Nachteile einer Kopie (gegenüber in-place):
  - Der benötigte Speicher wird verdoppelt (was, wenn das Array riesig ist?)
- Was ist nun besser?
  - In der Regel die Variante mit Kopie. Die unerwarteten/ungewollten Seiteneffekte können Kaskaden schwer zu findender Fehler verursachen.
  - Sollten wirklich große Arrays erwartet werden, sollte man sich unter Umständen ein grundlegend anderes System überlegen.

Präsenzaufgabe - Lösung

- Warum reicht `char[] clone = x;` nicht für eine Kopie?

  Arrays sind komplexe Datenstrukturen („ReferenceType“). Mit `clone = x` lassen wir `clone` auf die gleiche Speicheradresse wie `x` zeigen. Wir verändern dasselbe Array.

- Warum keine for-each Schleife?

  Damit erhalten wir nur die Werte an der jeweiligen Stelle, aber nicht den Index den es zu verändern gilt. Der Iterator erzeugt eine Kopie:

  ```
  for(char c : flipEm) { /* hier haben wir nur c */ }
  ```

  Wir könnten zwar einen Zeiger inkrementieren, aber wer garantiert uns dann die Reihenfolge der Iteration? Und selbst wenn das passt… Im Endeffekt haben wir dann wieder eine for-Schleife.

Übungsblatt 5 - Aufgabe 1 a)

- Der volle Code befindet sich hier: DotProduct.java.
- Wir berechnen $d = a \cdot b = (a_1 \dots a_n) \cdot (b_1 \dots b_n) = \sum_{i=1}^{n} a_i \cdot b_i$
- Oder als Java-Code:

```
int d = 0;
for(int i = 0; i < length; i++) {
    d += vector1[i] * vector2[i];
}

System.out.println("Vector1: " + vectorToString(vector1));
System.out.println("Vector2: " + vectorToString(vector2));

System.out.println("Das Skalarprodukt ist: " + d);
```

Übungsblatt 5 - Aufgabe 1 a)

- Nun noch Ausgabe der Vektoren:

```
public static String vectorToString(int[] vector) {
    String result = "(";
    for (int i = 0; i < vector.length; i++) {
        if (i > 0) result += ", ";
        result += vector[i];
    }
    return result + ")";
}
```

Übungsblatt 5 - Aufgabe 1 b)

- Der volle Code befindet sich hier: Transpose.java.
- Wir vertauschen! Aus $a_{i,j}$ wird $a'_{j,i}$.
- Oder als Java-Code:

```
int[][] transposed = new int[M][N];
for (int i = 0; i < M; i++) {
    for (int j = 0; j < N; j++) {
        transposed[i][j] = matrix[j][i];
        System.out.print(transposed[i][j] + " ");
    }
    System.out.println("");
}
```

Übungsblatt 5 - Aufgabe 2

- Der volle Code befindet sich hier: MatrixMultiplication.java.
- Zunächst eine zweite Matrix:

```
int[][] matrix2 = new int[M][N];
for(int i = 0; i < M; i++) {
    for (int j = 0; j < N; j++) {
        matrix2[i][j] = (int) (Math.random() * 10);
        System.out.print(matrix2[i][j] + " ");
    }
    System.out.println("");
}
System.out.println("");
```

Übungsblatt 5 - Aufgabe 2

- Und nun multiplizieren wir! Da wir davon ausgehen können, dass es klappt, gilt: $c_{i,j} = \sum_{k=1}^{m} a_{i,k} \cdot b_{k,j}$.

```
int[][] result = new int[N][N];
for(int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        int tmp = 0;
        for (int k = 0; k < M; k++) {
            tmp += matrix1[i][k] * matrix2[k][j];
        }
        result[i][j] = tmp;
        System.out.print(result[i][j] + " ");
    }
    System.out.println();
}
```

Übungsblatt 5 - Zusatzaufgabe a)

- Wir machen eine Raute (Rhombus.java).
- Naiv, können wir die bekannte Dreiecksstruktur nehmen und ein Dreieck für die obere und eines für die untere Hälfte basteln:

```
// oben
for(int i = 1; i <= N; i++) {
    for(int j = 1; j <= N-i; j++) System.out.print("␣");

    for(int j = 1; j <= i; j++) {
        if((j == 1) || (j == i)) System.out.print("*␣");
        else System.out.print("␣␣");
    }
    System.out.println("");
}
```

Übungsblatt 5 - Zusatzaufgabe a)

- Für unten spiegeln wir:

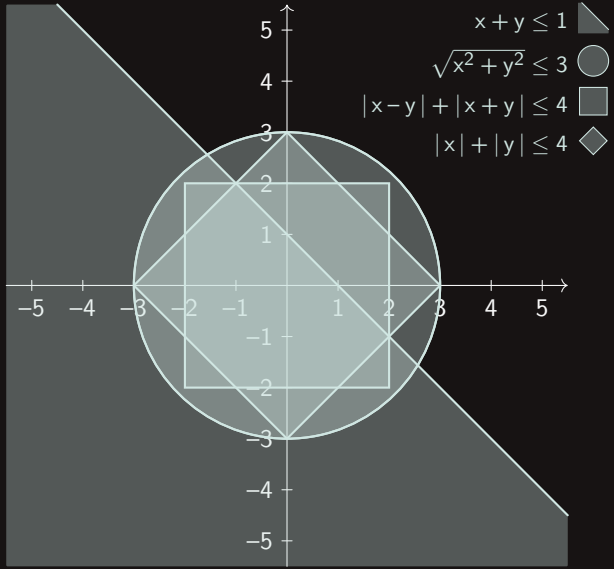
```
for(int i = N; i > 1; i--) {
    for(int j = i-1; j < N; j++) System.out.print("␣");

    for(int j = i; j >= 2; j--) {
        if((j == 2) || (j == i)) System.out.print("*␣");
        else System.out.print("␣␣");
    }
    System.out.println("");
}
```

Übungsblatt 5 - Zusatzaufgabe a)

- Negative Zahlen erlauben es, beide Hälften in einer Schleife zu behandeln:

```
for(int i = N-1; i > -N; i--) {
    for(int j = 1; j <= Math.abs(i); j++)
        System.out.print("␣");

    for(int j = 1; j <= N - Math.abs(i); j++) {
        if((j == 1) || (j == N - Math.abs(i)))
            System.out.print("*␣");
        else
            System.out.print("␣␣");
    }
    System.out.println("");
}
```

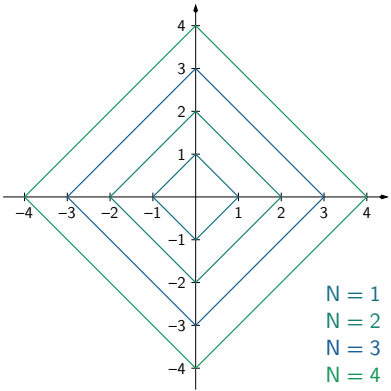
RhombusNegative.java



Manuell Layouts im Foliensatz zu ändern macht Spaß. Wirkt aber fast, als hätte ich das hier geklaut :/ Das hab ich nich!

Übungsblatt 5 - Zusatzaufgabe a) Eleganter

- Wir haben eine Menge zu der alle Punkte zählen für die gilt: $|x| + |y| = N - 1$.
- $N - 1$ rührt daher, da wir ja für $N = 1$ sonst auch in y-Richtung je 1 hätten.




```java
if(args.length != 1) System.exit(1);
int N = Integer.parseInt(args[0]) - 1;
// Diskretisieren
for (int y = N; y >= -N; y--) {
    for (int x = -N; x <= N; x++) {
        if (Math.abs(y) + Math.abs(x) == N)
            System.out.print("*");
        else System.out.print("␣");
    }
    System.out.println();
}
```

RhombusElegant.java

Übungsblatt 5 - Zusatzaufgabe b)

- Die Musterlösung (NestedRhombus.java).

```
for(int i = 1; i <= N; i++) {
    for(int j = 1; j <= N-i; j++) System.out.print("␣");
    if(i % 2 == 1) {
        for (int j = 1; j <= i; j++) {
            if (j % 2 == 1) System.out.print("*␣");
            else System.out.print("␣␣");
        }
    } else {
        for (int j = 1; j <= i / 2; j++) { // Linke Hälfte
            if (j % 2 == 1) System.out.print("*␣");
            else System.out.print("␣␣");
        }
        for (int j = i/2; j >= 1; j--) { // Rechte Hälfte
            if (j % 2 == 1) System.out.print("*␣");
            else System.out.print("␣␣");
        }
    }
    System.out.println("");
}
```

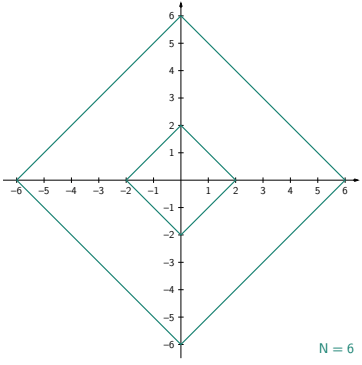
Übungsblatt 5 - Zusatzaufgabe b)

- Die Musterlösung der unteren Hälfte:

```
for(int i = N-1; i >= 1; i--) {
    for(int j = i; j < N; j++) System.out.print("␣");
    if(i % 2 == 1) {
        for (int j = 1; j <= i; j++) {
            if (j % 2 == 1) System.out.print("*␣");
            else System.out.print("␣␣");
        }
    } else {
        for (int j = 1; j <= i / 2; j++) { // Linke Hälfte
            if (j % 2 == 1) System.out.print("*␣");
            else System.out.print("␣␣");
        }
        for (int j = i/2; j >= 1; j--) { // Rechte Hälfte
            if (j % 2 == 1) System.out.print("*␣");
            else System.out.print("␣␣");
        }
    }
    System.out.println("");
}
```

Übungsblatt 5 - Zusatzaufgabe b) Eleganter

- Wir möchten nun alle Rauten durch $|x| + |y| \leq N - 1$.
- Da wir aber nur je die mit Längen –4 haben wollen: $|x| + |y| \equiv N - 1 \mod 4$. Mathe ist Liebe.




```
if (args.length != 1) System.exit(1);
int N = Integer.parseInt(args[0]) - 1;

for (int y = N; y >= -N; y--) {
    for (int x = -N; x <= N; x++) {
        int radius = Math.abs(y) + Math.abs(x);
        if (radius <= N && radius % 4 == N % 4)
            System.out.print("*");
        else System.out.print("␣");
    }
    System.out.println();
}
```

NestedRhombusElegant.java

Blatt 6 — Methoden & Tic-Tac-Toe

KW 49

Präsenzaufgabe

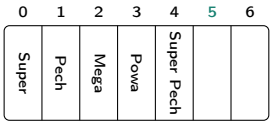
8 **Glückselig sind die Studierenden**

Implementieren Sie ein veränderbares `WheelOfFortune`.

1. Erstellen Sie eine Klasse `WheelOfFortune` welches ein String Array `slots` fester Größe und eine Instanzvariablen `int nextSlot` besitzt.
2. `public boolean addEntry(String entry)` fügt einen Eintrag hinzu, sofern es die Größe zulässt (der Rückgabewert kennzeichnet den Erfolg).
3. `public boolean removeEntry()` entfernt den letzten hinzugefügten Eintrag wieder (der Rückgabewert kennzeichnet ob ein Eintrag entfernt wurde).
4. `public int getNumEntries()` soll die Gesamtanzahl der Einträge liefern.
5. `public String spin()` dreht das Glücksrad und liefert den Eintrag zurück. Gibt es keinen Eintrag wünschen wir einen leeren String.

Erzeugen Sie zudem eine `main`-Methode welche das Rad sinnvoll benutzt.

`nextSlot = 5`



- Eine Basis (WheelOfFortune.java):

```java
public class WheelOfFortune {
    private String[] slots;
    private int nextSlot;

    public WheelOfFortune(int numSlots) {
        nextSlot = 0;
        // keine negative Länge
        slots = new String[Math.max(numSlots, 0)];
        // Am Anfang setzen wir alle auf den leeren String
        for(int i = 0; i < slots.length; i++)
            slots[i] = ""; // vs. new String("");
    }
}
```

Von **Klassen** lassen sich durch den **Konstruktor Objekte** erzeugen. Der **Zustand** eines Objekts ist durch die **Instanzvariablen** definiert. Jedes Objekt hat seinen *eigenen Zustand*.

Präsenzaufgabe - Lösung

Hinzufügen

```
public boolean addEntry(String entry) {
    if(nextSlot == slots.length) // Sind wir voll?
        return false;

    slots[nextSlot] = entry;
    nextSlot++;
    return true;
}
```

```java
public boolean removeEntry() {
    if(nextSlot == 0) // Sind wir leer?
        return false;

    nextSlot--;
    slots[nextSlot] = "";
    return false;
}

public int getNumEntries() {
    return nextSlot;
}
```

Präsenzaufgabe - Lösung

Es geht wieder loooos

```java
public String spin() {
    // Sind wir leer?
    if(slots.length == 0)
        return "";

    int slot = (int) (Math.random() * nextSlot);
    return slots[slot];
}
```

(Pseudo-)**Zufall** ist ein spannendes Thema. Anstelle von `Math.random()` können wir uns direkt ein „Random-Objekt“ bauen:

```java
Random rnd = new Random();
rnd.nextInt(15); // {0,...,14}
```

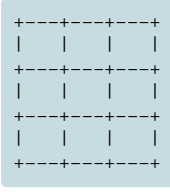
Präsenzaufgabe - Lösung


```
public static void main(String[] args) {
    int numSlots = 6;
    WheelOfFortune wheel = new WheelOfFortune(numSlots);

    for(int i = 0; i < numSlots + 1; i++) {
        String entry = "Eintrag " + (char) ('A' + i);
        if(wheel.addEntry(entry))
            System.out.println("Eintrag " + entry + " hinzugefügt");
        else System.out.println("Glücksrad ist voll");
    }

    String result = wheel.spin();
    System.out.println("Ergebnis einer Drehung: " + result);
    while(wheel.getNumEntries() > 0)
        wheel.removeEntry();
    result = wheel.spin();
    System.out.println("Drehen des leeren Glücksrades ergibt: " + result);
}
```

Übungsblatt 6 - Aufgabe 1a)

- We wanta print a board (TicTacToe.java):

```
public static void printBoard(char[][] board) {
    for (char[] row : board) {
        System.out.println("+---+---+---+");
        for (char cell : row)
            System.out.print("| " + (cell != 0 ? cell : ' ') + " ");
        System.out.println("|");
    }
    System.out.println("+---+---+---+\n");
}
```



Übungsblatt 6 - Aufgabe 1b)

- Request from the best:

```
public static char[][] getMove(int player, char[][] board) {
    System.out.println("Spieler " + player + " ist am Zug:");
    int move = 0;
    do {
        move = scanner.nextInt();
    } while (!isMoveValid(board, move));
    return makeMove(board, move, player);
}
```

Wir versuchen **lesbaren Code** zu schreiben. Anstelle einfach `isMoveValid` und `makeMove` zu integrieren, lagern wir den Code in Subroutinen aus, welche dessen Semantik als Namen tragen. In späteren Veranstaltungen ist dies im Rahmen der **Testbarkeit** und **Lesbarkeit** von besonderem Interesse.

Übungsblatt 6 - Aufgabe 1b)

- Check it and regret it:

```
private static boolean isMoveValid(char[][] board, int move) {
    return move >= 1 && move <= 9
        && board[getY(move)][getX(move)] == 0;
}

public static int getX(int cellNumber) {
    return (cellNumber - 1) % 3;
}

public static int getY(int cellNumber) {
    return (cellNumber - 1) / 3;
}
```

1	2	3
4	5	6
7	8	9

Übungsblatt 6 - Aufgabe 1b)

- Execute Order 66:

```
private static char[][] makeMove(char[][] board, int cellNumber, int player) {
    char[][] newBoard = board.clone(); // odeeeer?
    newBoard[getY(cellNumber)][getX(cellNumber)] = getPlayerSymbol(player);
    return newBoard;
}
```

- Wo wir schon bei Star Wars™sind…
- Have you ever heard the tragedy of Darth Clone**a**able the wise? Well, then prepare yourself for… something at least.


clone

Übungsblatt 6 - Aufgabe 1b) Fortsetzung

- Outstanding Deep-Copy:

```
char[][] newBoard = new char[3][3];
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        newBoard[i][j] = board[i][j];
    }
}
```

- Wer `clone` doch benutzen „muss“:

```
for(int i = 0; i < 3; i++) {
    newBoard[i] = board[i].clone();
}
```

Übungsblatt 6 - Aufgabe 1c)

- Outstanding Move:

```
public static boolean winningMove(int player, char[][] board) {
    char c = getPlayerSymbol(player);
    return winsHorizontal(board, c) || winsVertical(board, c)
        || winsDiagonal(board, c);
}
```

- Für das Spielersymbol:

```
public static char getPlayerSymbol(int player) {
    return (player == 1) ? 'x' : 'o';
}
```

Hier könnte man auch eine Enumeration verwenden.

Übungsblatt 6 - Aufgabe 1c)

- Win me Baby:

```
private static boolean winsHorizontal(char[][] board, char c) {
  return ((board[0][0] == c) && (board[0][1] == c) && (board[0][2] == c)) ||
       ((board[1][0] == c) && (board[1][1] == c) && (board[1][2] == c)) ||
       ((board[2][0] == c) && (board[2][1] == c) && (board[2][2] == c));
}

private static boolean winsVertical(char[][] board, char c) {
  return ((board[0][0] == c) && (board[1][0] == c) && (board[2][0] == c)) ||
       ((board[0][1] == c) && (board[1][1] == c) && (board[2][1] == c)) ||
       ((board[0][2] == c) && (board[1][2] == c) && (board[2][2] == c));
}

private static boolean winsDiagonal(char[][] board, char c) {
  return ((board[0][0] == c) && (board[1][1] == c) && (board[2][2] == c)) ||
       ((board[2][0] == c) && (board[1][1] == c) && (board[0][2] == c));
}
```

Übungsblatt 6 - Aufgabe 1d)

- The main:

```
int player = 1; // Start player 1
char[][] board = emptyTicTacToeBoard();

// Game loop
for (int i = 0; i < 9; i++) {
    board = getMove(player, board);
    printBoard(board);
    if (winningMove(player, board)) {
        System.out.println("Spieler " + player + " hat gewonnen!");
        return;
    }
    player = otherPlayer(player);
}
System.out.println("Unentschieden");
scanner.close(); // wichtig
```

Scanner::close ist eure **Lebensversicherung!** Schließt eine Ressource die ihr anfordert **immer** und **ohne Ausnahme** idealerweise hält die selbe Abstraktionsebene Anforderung und Abstoßung. Später machen das auto-closeables angenehmer.

Übungsblatt 6 - Aufgabe 1d)

- Initialize the board:

```
public static char[][] emptyTicTacToeBoard() {
    char[][] board = new char[3][3];
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            board[i][j] = 0;
        }
    }
    return board;
}
```

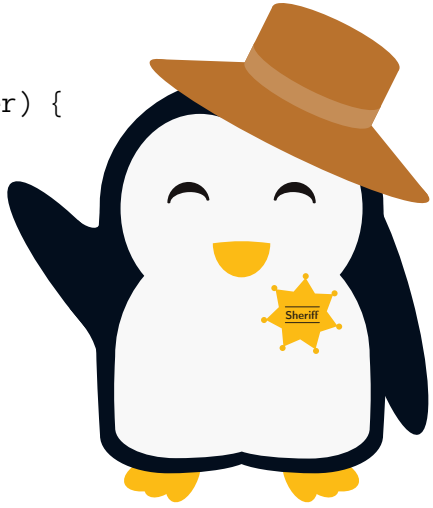
Ist eine solche Initialisierung **notwendig**? Nein. Java weist Variablen „Default-Werte“ zu („die 0“). Zahlen werden 0, booleans *false*, chars kriegen das Zeichen mit ASCII 0, ...

Dennoch, „explicit is better than implicit“. Andere Sprachen machen das andes, was das Lesen erschwert. Zudem zwingt explizit dazu, sich über den „Standardwert“ Gedanken zu machen.

Übungsblatt 6 - Aufgabe 1d)

- Toggle den Spieler in dir:

```
public static int otherPlayer(int player) {
    return player == 1 ? 2 : 1;
}
```



Blatt 7 — Überladen & Überschatten

KW 50

Präsenzaufgabe

9

Die Springer meiner Herzen sie umkreisen mich…
Und mit jeder weitren Zeile, leide ich.

Erstellen Sie eine `Circle`-Klasse mit privaten Instanzvariablen für Radius, Flächeninhalt und Umfang. Definieren Sie einen öffentlichen Konstruktor der auf Basis eines Radius die Instanzvariablen initialisiert. Erstellen Sie nun zusätzlich die Methoden:

- *Getter*-Methoden für die Instanzvariablen.
- Eine Methode um die Kreiseigenschaften auf der Kommandozeile auszugeben.
- Eine statische Methode, die für einen Kreisradius den Flächeninhalt zurückgibt.
- Eine statische Methode, die für einen Kreisradius den Umfang zurückgibt.

Bonus: Wäre es möglich zusätzliche Konstruktoren zu definieren, welche entweder nur den Flächeninhalt oder nur den Umfang als Parameter übernehmen?

Präsenzaufgabe - Lösung

- Das wichtigste Beiseite: Circle.java
- Wir gehen in die Analyse:

Anstelle von `radius * radius` geht auch `Math.pow(radius,2)` für radius^2. Die Formeln sollten natürlich bekannt sein ☺.

Erstellen Sie eine `Circle`-Klasse mit privaten Instanzvariablen für Radius, Flächeninhalt und Umfang.

Definieren Sie einen öffentlichen Konstruktor der auf Basis eines Radius die Instanzvariablen initialisiert.

```java
public class Circle {
   private double radius;
   private double area;
   private double circumference;

   public Circle(double radius) {
      this.radius = radius;
      this.area = Math.PI * radius * radius;
      this.circumference = 2 * Math.PI * radius;
   }
}
```

Präsenzaufgabe - Lösung

- Nun gilt es zu sondieren…

```
public class Circle {
    private double radius;
    private double area;
    private double circumference;
    ...
    public Circle(double radius) { ... }

    public double getRadius() { return radius; }
    public double getArea() { return area; }
    public double getCircumference() { return circumference; }
}
```

- Ein printchen Liebe, ein printchen Freude…

```
public class Circle {
    private double radius;
    private double area;
    private double circumference;

    public Circle(double radius) { ... }
    ...

    public void printProperties() {
        System.out.println("Radius:␣" + this.radius + "cm");
        System.out.println("Fläche:␣" + this.area + "cm^2");
        System.out.println("Umfang:␣" + this.circumference + "cm");
    }
}
```

- Berechne Gefechte:...

```
public class Circle {
    private double radius;
    private double area;
    private double circumference;

    public Circle(double radius) { ... }
    ...

    public static double computeArea(double radius) {
        return Math.PI * radius * radius;
    }
    public static double computeCircumference(double radius) {
        return 2 * Math.PI * radius;
    }
}
```

- Die statischen Methoden können wir rückwirkend verwenden…

```
public class Circle {
    private double radius;
    private double area;
    private double circumference;

    public Circle(double radius) {
        this.radius = radius;
        this.area = Circle.computeArea(radius);
        this.circumference = Circle.computeCircumference(radius);
    }
    ...

    public static double computeArea(double radius) { ... }
    public static double computeCircumference(double radius) { ... }
}
```

- Da wir nur einen Parameter übergeben, hilft uns Overload™ leider nicht!

```
public class Circle {
    ...
    public Circle(double radius) { ... }

    public Circle(double area) { ... }
    public Circle(double circumference) { ... }
    ...
}
```

Alle drei Konstruktoren würden durch `Circle(double)` identifiziert werden. Das erlaubt Java allerdings **nicht**.

Übungsblatt 7 - Aufgabe 1a)

- Man nennt mich Jörg, den Zeichenkettenspalter (MethodOverloading.java):

```
public static int[] separateDigits(String number) {
    int numDigits = number.length();
    int[] digits = new int[numDigits];

    for(int i = 0; i < numDigits; i++)
        digits[i] = number.charAt(i) - '0';

    return digits;
}
```

Alternativ ginge auch `digits[i] = Integer.parseInt(number.substring(i, i + 1))`

Übungsblatt 7 - Aufgabe 1b)

- Theoretisch können wir für die Verarbeitung die alte Methode benutzen:

```
public static int[] separateDigits(int number) {
    return separateDigits(Integer.toString(number));
}
```

- Das widerspricht aber der Aufgabenbeschreibung! Deswegen klammern wir uns an den ursprünglichen Wunsch…

Übungsblatt 7 - Aufgabe 1b)

- Deswegen implementieren wir das Ganze so:

```
public static int[] separateDigits(int number) {
    int numDigits = (int) (Math.log10(number) + 1);
    int[] digits = new int[numDigits];

    for (int i = numDigits - 1; i >= 0; i--) {
        digits[i] = number % 10;
        number /= 10;
    }
    return digits;
}
```

Betrachte $n = 476 = 4 \cdot 10^2 + 7 \cdot 10^1 + 6 \cdot 10^0$.

$n/10 = 4 \cdot 10^{2-1} + 7 \cdot 10^{1-1} + 6 \cdot 10^{0-1} = 47.6$

$n \bmod 10 = \cancel{4 \cdot 10^2} + \cancel{7 \cdot 10^1} + 6 \cdot 10^0 = 6$

$\log_{10} n = \log_{10}(4 \cdot 10^2 + 7 \cdot 10^1 + 6 \cdot 10^0) \approx_{\max} \log_{10}(4 \cdot 10^2) \approx 2.6$

Kleinste Ziffer: $z = n \bmod 10$, Zahl ohne kleinste Ziffer: $z = \lfloor \frac{n}{10} \rfloor$. Anzahl Ziffern: $z = \lfloor \log_{10} |n| + 1 \rfloor$ (beachte: 0). Was ist mit 100? Mit 1000?

Übungsblatt 7 - Aufgabe 1c)

- Es ist **nicht** möglich, die zweite Methode mit `public static byte[] separateDigits(String number)` zu erzeugen.
- Warum?
- Die Signatur `separateDigits(String)` enthält (in Java) *nicht* den Rückgabetyp. Daher, sind für Java beide Methoden nicht zu unterscheiden. Dies wird vom Java-Standard verboten.

Method-Signature:
Name und Parametertypen

Overloading / Überladung:
Gleicher Name, andere Signatur

Übungsblatt 7 - Aufgabe 2

- Die Ausgabe eines Programms verstehen:

```
public class VariableScope {
    static int a = 23;
    static int b = 47;
    public static void print() {
        System.out.println("a=" + a + ",␣b=" + b + "\n");
    }
    public static void print(int a, int b) {
        System.out.println("a=" + a + ",␣b=" + b + "\n");
    }
    public static void main(String[] args) {
        int a = 17, b = 24;
        print();
        print(a, b);
        System.out.println("a=" + a + ",␣b=" + b);
    }
}
```

Shadowing / Überschatten: [vielseitig, z.B.] lokale Variable mit gleichem Namen wie globale.

```
a=23, b=47↩
↩
a=17, b=24↩
↩
a=17, b=24↩
↩
```

Übungsblatt 7 - Aufgabe 2 — In Worten

- Wir erhalten „a=*23*, b=*47*↩↩“ durch den Aufruf von `print()`, welcher die Werte der zwei statischen globalen Variablen ausgibt, da diese nicht **überschattet** werden.
- „a=*17*, b=*24*↩↩“ erhalten wir durch `print(int a, int b)` welches durch die Namen der Parameter die globalen Variablen `a` und `b` überschattet. Die Parameter sind an die lokalen Variablen in `VariableScope::main` gebunden, wo sie ebenfalls die globalen Variablen **überschatten**.
- Analog erhalten wir „a=*17*, b=*24*↩↩“ durch die eben angesprochene **Überschattung** in `VariableScope::main`.

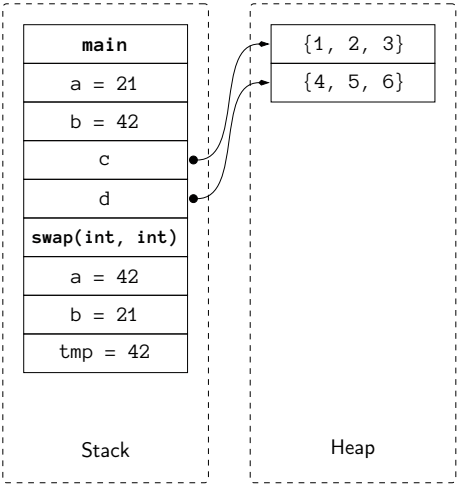
Ich verwende „↩“ um eine neue Zeile zu markieren. Auch dann, wenn ich sie nicht im Text formatiere.

Übungsblatt 7 - Aufgabe 3 Musterlösung

- Die zwei Integer Variablen `a` und `b` haben nach dem Aufruf der `swap` Methode die selben Werte wie davor. Das liegt daran, dass in Java alle Parameter *by value* übergeben werden, somit findet der Tausch der Werte in der `swap` Methode auf einer Kopie der Variablen statt.
- Die Werte der Array Variablen `c` und `d` sind nach dem Aufruf der `swap` Methode jedoch getauscht. Dies liegt daran, dass hier nur die Referenz auf das Array *by value* übergeben wird, welche nicht verändert werden kann. Beim Zugriff auf die Arrays, werden jedoch die originalen Werte und keine Kopien geliefert, und somit können diese verändert werden.

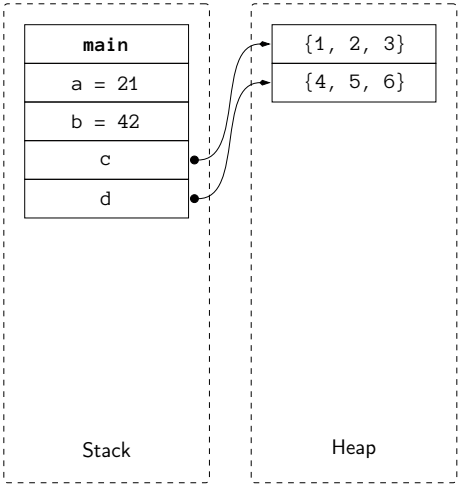
Übungsblatt 7 - Aufgabe 3

```java
public class SwapFunction {
    public static void swap(int a, int b) {
      ● int tmp = b; b = a; a = tmp;
    }
    public static void swap(int[] a, int[] b) {
        if(a.length != b.length) return;
        for(int i = 0; i < a.length; i++) {
            int tmp = b[i]; b[i] = a[i]; a[i] = tmp;
        }
    }
    public static void main(String[] args) {
        int a = 21; int b = 42;
        int[] c = {1, 2, 3};
        int[] d = {4, 5, 6};
        swap(a, b); // → swap(int, int)
        swap(c, d);
    }
}
```



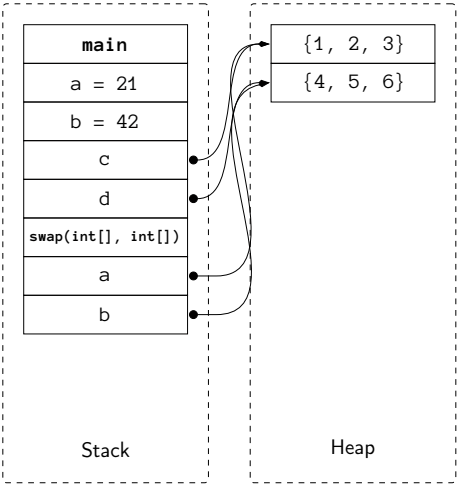
Übungsblatt 7 - Aufgabe 3

```
public class SwapFunction {
    public static void swap(int a, int b) {
        int tmp = b; b = a; a = tmp;
    }
    public static void swap(int[] a, int[] b) {
        if(a.length != b.length) return;
        for(int i = 0; i < a.length; i++) {
            int tmp = b[i]; b[i] = a[i]; a[i] = tmp;
        }
    }
    public static void main(String[] args) {
        int a = 21; int b = 42;
        int[] c = {1, 2, 3};
        int[] d = {4, 5, 6};
        swap(a, b);
        swap(c, d);
    }
}
```



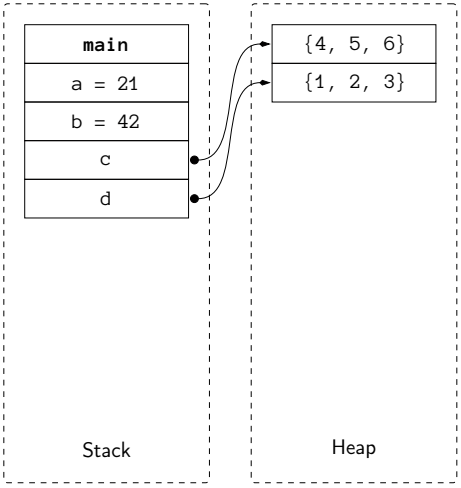
Übungsblatt 7 - Aufgabe 3

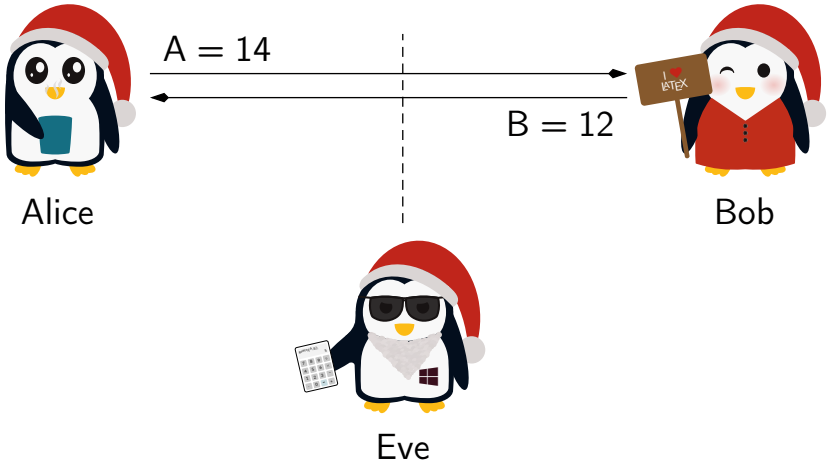
```java
public class SwapFunction {
    public static void swap(int a, int b) {
        int tmp = b; b = a; a = tmp;
    }
    public static void swap(int[] a, int[] b) {
        if(a.length != b.length) return;
        for(int i = 0; i < a.length; i++) {
            int tmp = b[i]; b[i] = a[i]; a[i] = tmp;
        }
    }
    public static void main(String[] args) {
        int a = 21; int b = 42;
        int[] c = {1, 2, 3};
        int[] d = {4, 5, 6};
        swap(a, b);
        swap(c, d); // → swap(int[], int[])
    }
}
```



Übungsblatt 7 - Aufgabe 3

```
public class SwapFunction {
    public static void swap(int a, int b) {
        int tmp = b; b = a; a = tmp;
    }
    public static void swap(int[] a, int[] b) {
        if(a.length != b.length) return;
        for(int i = 0; i < a.length; i++) {
            int tmp = b[i]; b[i] = a[i]; a[i] = tmp;
        }
    }
    public static void main(String[] args) {
        int a = 21; int b = 42;
        int[] c = {1, 2, 3};
        int[] d = {4, 5, 6};
        swap(a, b);
        swap(c, d);
    }
}
```




A = 14
B = 12
I ♥ LaTeX
Alice
Bob
Eve

Blatt 8 — Klassen & Taxis

KW 2

Präsenzaufgabe

10 **Teil mich. Nochmal. Nochmal!**

In dieser Aufgaben sollen Sie mit Hilfe des *euklidischen Algorithmus* den größten gemeinsamen Teiler zweier ganzer Zahlen `a` und `b` bestimmen. Implementieren Sie den Algorithmus einmal *iterativ* und einmal *rekursiv*. Welchen jeweiligen Vor- und Nachteil haben die beiden Lösungen?

Beispiele:

- `ggt(39, 27)` $\rightarrow$ *3*

$$39 = 1 \cdot 27 + 12$$
$$27 = 2 \cdot 12 + 3$$
$$12 = 4 \cdot \boxed{3} + 0$$

- `ggt(15, 7)` $\rightarrow$ *1*

$$15 = 2 \cdot 7 + 1$$
$$7 = 7 \cdot \boxed{1} + 0$$

Präsenzaufgabe - Lösung

- Der größter gemeinsame Teiler also: Euclid.java.
- Die Definition:

$$\text{ggt}(a, b) = \begin{cases} a & \text{wenn } b = 0 \\ \text{ggt}(b, a \bmod b) & \text{sonst.} \end{cases}$$

- In Java? Unfassbar anders natürlich!

```
public static int ggt(int a, int b) {
    if(b == 0)
        return a;
    else
        return ggt(b, a % b);
}
```

Präsenzaufgabe - Lösung

- Die iterative Variante:

```java
public static int ggt(int a, int b) {
    while(b != 0) {
        // swap a, b = b, a % b
        int tmp = a % b;
        a = b;
        b = tmp;
    }
    return a;
}
```

Präsenzaufgabe - Iteration vs. Rekursion

- Die *iterative* Lösung ist performanter und braucht weniger Speicher.
- Die *rekursive* Lösung ist kompakter und übersichtlicher.

Übungsblatt 8 - Aufgabe 1a)

- Taxi nach Heim bittedange (Taxi.java).

Implementieren Sie eine Klasse `Taxi` mit privaten, unveränderlichen Instanzvariablen für die folgenden Eigenschaften:

- Grundpreis einer Fahrt in Euro
- Den Kilometerpreis in Euro
- Den Kraftstoffverbrauch in $\frac{\text{L}}{\text{100km}}$
- Das Tankvolumen des Taxis in Liter

Fügen Sie weiterhin die folgenden privaten, veränderlichen Variablen hinzu:

- Den Tankinhalt in Litern
- Den Stand des Taxometers in Euro
- Die Gesamteinnahmen in Euro

Erstellen Sie nun einen öffentlichen Konstruktor um diese Variablen sinnvoll zu initialisieren.

```
public class Taxi {
   private final double basicCharge;
   private final double kilometerPrice;
   private final double fuelConsumption;
   private final double fueltankCapacity;

   private double remainingFuel;
   private double taxometer;
   private double totalEarnings;

   public Taxi(...) {
       ...
   }
}
```

Übungsblatt 8 - Aufgabe 1a)

```
public class Taxi {
   private final double basicCharge;       private final double kilometerPrice;
   private final double fuelConsumption;   private final double fueltankCapacity;
   private double remainingFuel;           private double taxometer;
   private double totalEarnings;

   public Taxi(double basicCharge, double kilometerPrice,
               double fuelConsumption, double fueltankCapacity) {
      this.basicCharge = basicCharge;
      this.kilometerPrice = kilometerPrice;
      this.fuelConsumption = fuelConsumption;
      this.fueltankCapacity = fueltankCapacity;

      this.remainingFuel = fueltankCapacity;
      this.taxometer = 0.0;
      this.totalEarnings = 0.0;
   }

}
```

Übungsblatt 8 - Aufgabe 1b)

- Overdriven sie mich home:

```
public double drive(double distance) {
    double requiredFuel = distance * fuelConsumption / 100;
    if(this.remainingFuel < requiredFuel)
        return -1; // Fahrt nicht möglich

    this.remainingFuel -= requiredFuel;
    this.taxometer += basicCharge + distance * kilometerPrice;

    return taxometer;
}
```

Übungsblatt 8 - Aufgabe 1c)

- Money Please:

```
public boolean pay(double amount) {
    if (amount < taxometer)
        return false;
    if (amount > taxometer)
        System.out.format("Received %.2f€ tip%n", amount - taxometer);

    taxometer = 0.0;
    totalEarnings += amount;
    return true;
}
```

Wir könnten auch **else if** benutzen. So verschmilzt aber ein Guard mit Funktionalität.

Übungsblatt 8 - Aufgabe 1d)

- Einmal auftankeeen:

```
public void refill(double pricePerLitre) {
    double refill = Math.min(
        this.totalEarnings / pricePerLitre,
        this.fueltankCapacity - this.remainingFuel
    ); // Können wir finanziell gesehen volltanken?

    this.remainingFuel += refill;
    this.totalEarnings -= refill * pricePerLitre;
}
```

Übungsblatt 8 - Aufgabe 1e)

- Eine beispielhafte (musterlösungsverdächtige) Verwendung:

```
public static void main(String[] args) {
    Taxi taxi = new Taxi(3, 0.5, 10, 50);
    double totalCost = taxi.drive(250);
    if (totalCost == -1) System.out.println("Not enough fuel left");
    else System.out.println("Trip costs " + totalCost);
    // pay
    if (taxi.pay(totalCost + 3.00)) System.out.println("Tipped 3,00€");
    // drive
    totalCost = taxi.drive(450);
    if (totalCost == -1) System.out.println("Not enough fuel left");
    // refill & drive
    taxi.refill(1.7);
    totalCost = taxi.drive(450);
    if (totalCost == -1) System.out.println("Not enough fuel left");
    else System.out.println("Trip costs " + totalCost);
}
```

Zusatzaufgabe 3/6 — Zwei Datumsies Vergleichen

- Daten vergleichen ist einfach: Sind Jahre, Tage, … identisch?
- Zu sagen, wie viele Minuten zwischen zwei Daten liegen, ist schwieriger.
  - Wie viele Minuten liegen zwischen dem 28. Februar 23:59 und dem 1. März 0:05 im selben Jahr?
  - Wir müssen Schaltjahre beachten.
  - Und Schaltsekunden!
  - Und Säkularjahre…
- In der Regel dampft man Daten daher auf ihre (Nano-/Milli-/…-)Sekunden seit einem Zeitpunkt herunter.
  - `java.util.Date` tut dies bereits (Millisekunden)!
  - Dies Manuell korrekt zu tun ist *sehr aufwändig* (Die zugehörige Methode `computeTime` in `GregorianCalendar` hat beispielsweise allein 200 Zeilen.)

Zusatzaufgabe 3/6 — Minuten zwischen zwei Daten

- Die Minuten zwischen zwei Daten `a` und `b` erhalten wir so wie folgt:

```
long aStamp = a.getTime();
long bStamp = b.getTime();
double diffInMinutes = (aStamp - bStamp) / (1000 * 60d)
```

- Magic Numbers:
  - 1000 Millisekunden bilden eine Sekunde.
  - 60 Sekunden bilden eine Minute.

- Nun eine Variante mit `SimpleDateFormat`.

```
public class Reservation {
    private final String name;
    private final Date resDate;

    private final SimpleDateFormat dateFormat =
            new SimpleDateFormat("yyyy.MM.dd␣HH:mm");
    public Reservation(String name, String date, String time)
            throws ParseException {
        this.name = name;
        this.resDate = dateFormat.parse(date + "␣" + time);
    }
}
```

Warum nicht **static**? Nun, das hat etwas mit *Threads* zu tun...

Mini-Exkurs: Mit Ausnahmen umgehen

- `ParseException` ist eine „checked-Exception“ $\Rightarrow$ wir müssen sie behandeln.
- In dieser Aufgabe können wir eine korrekte Eingabe annehmen.
- Dies erlaubt ein Unterdrücken des Fehlers:

```
try {
    ...
} catch (ParseException ex) {
    ex.printStackTrace();
}
```

- Allgemein ist aber das Weiterreichen sicherer:

```
public Reservation(...) throws ParseException {
```

Zusatzaufgabe 3/6 — toString

- `toString` ist einfach, wenn wir zusätzlich noch die übergebenen Daten speichern.
- Mit `String::format` können wir uns aber interessanteren Freuden widmen:

```
public class Reservation {
    private final String name;
    private final Date resDate;
    public String toString() {
        return String.format(Locale.ENGLISH, // March statt März
            "%s reserved for %2$tB %2$td, %2$tY at %2$tH:%2$tM",
            this.name, this.resDate
        );
    }
}
```

Format String Syntax

Zusatzaufgabe 3/6 toString

- Aber halt. Es war ja `toString(Reservation)` gefordert.
- Nun, wir könnten dies wie folgt lösen (das ist aber unüblich):

```
public class Reservation {
    private final String name;
    private final Date resDate;
    public static String toString(Reservation r) {
        return String.format(Locale.ENGLISH, // March statt März
            "%s reserved for %2$tB %2$td, %2$tY at %2$tH:%2$tM",
            r.name, r.resDate
        );
    }
    public String toString() { toString(this); }
}
```

- Gibt es Alternativen?
- Japp. Die gibt es. Beispielsweise mit `Calendar`:

```
public static String toString(Reservation reservation) {
    Calendar calendar = Calendar.getInstance();
    calendar.setTime(reservation.resDate);
    return "Reservierung für " + reservation.name
           + " am " + calendar.get(Calendar.YEAR) + "."
           + (calendar.get(Calendar.MONTH) + 1) + "."
           + calendar.get(Calendar.DAY_OF_MONTH)
           + " um " + calendar.get(Calendar.HOUR_OF_DAY) + ":"
           + calendar.get(Calendar.MINUTE)
           + " Uhr.";
}
```

Zusatzaufgabe 3/6 — toString – SimpleDateFormat

- Das `SimpleDateFormat` kann das aber auch!

```
private final String name;
private final Date resDate;
private final SimpleDateFormat dateFormat =
      new SimpleDateFormat("yyyy.MM.dd HH:mm");

public static String toString(Reservation reservation) {
   return "Reservierung␣für␣" + reservation.name
         + "␣am␣" + dateFormat.format(reservation.resDate);
}
```

- Entscheidet selbst :D

Zusatzaufgabe 3/6 — equals

- Felder vergleichen:

```
public class Reservation {
    private final String name;
    private final Date resDate;
    public boolean equals(Reservation reservation) {
        return this == reservation ||
            (Objects.equals(this.resDate, reservation.resDate)
            && Objects.equals(this.name, reservation.name));
    }
}
```

- Der `==`-Vergleich ist nicht notwendig, zeigt aber die stärkere Aussage von „identisch“ gegenüber „gleich“.
- `Object.equals(a, b)` entspricht `a.equals(b)`, kann aber auch mit `a == null` umgehen!

Zusatzaufgabe 3/6 main

- Eine Form der Verwendung:

```
public class Reservation {
  public Reservation(String n, String d, String t) throws ParseException;
  public String toString();
  public boolean equals(Reservation reservation);

  public static void main(String[] args) throws ParseException {
    Reservation r1 = new Reservation("Hugo", "2022.10.15", "14:32");
    System.out.println(r1);
    System.out.println("same: " + r1.equals(r1));
    Reservation other = new Reservation("Hugo", "2022.10.15", "14:32");
    System.out.println("other: " + r1.equals(other));
    Reservation diff = new Reservation("Hugo", "2022.01.15", "14:33");
    System.out.println("different: " + r1.equals(diff));
  }
}
```

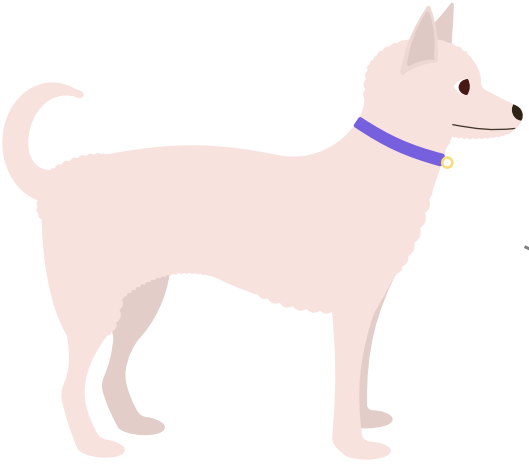
Zusatzaufgabe 3/6 — Der checkIn

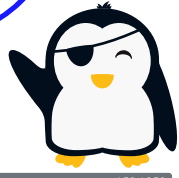
- Beim `checkIn` können wir das Minutenintervall leicht prüfen:

```java
public class Reservation {
  private final String name;
  private final Date resDate;
  public boolean checkIn(Reservation reservation) {
      long them = reservation.resDate.getTime();
      long we = this.resDate.getTime();
      double diffInMinutes = (them - we) / (MS_IN_S * S_IN_MS);
      // is it between 5 and 15 minutes?
      return diffInMinutes >= -5 && diffInMinutes <= 15 ;
  }
}
```

- Dabei ist `MS_IN_S = 1000` und `S_IN_MS = 60`.
- Optional auch `Objects.equals(this.name, reservation.name)` um den Namen zu prüfen.

Reservation.java





Blatt 9 — Rekursion

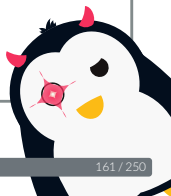
KW 3

Präsenzaufgabe

11 **Wenn du dein Kinderzimmer nicht aufräumst, …**

In dieser Aufgabe sollen Sie die *Binäre Suche* implementieren, welche nützlich ist um ein sortiertes Array effizient zu durchsuchen. Schreiben Sie die Methoden-Signatur `public static int find(int[] array, int element)`. Diese Methode soll ein *aufsteigend* sortiertes Array und das gesuchte Element als Parameter übernehmen, und daraufhin den Index des Elements im Array zurückgeben, sofern dieses vorhanden ist. Sollte das Elemente mehrfach vorhanden sein, soll der kleinste Index zurückgegeben werden. Falls das Element nicht im Array vorhanden ist, soll `-1` zurückgegeben werden. Implementieren Sie die eigentliche *binäre Suche* rekursiv. Hilfsmethoden sind gestattet.


```
...
```

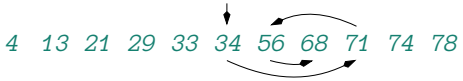


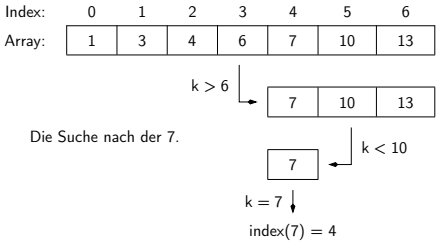
Präsenzaufgabe

11

Wenn du dein Kinderzimmer nicht aufräumst, …

Schreiben Sie die Methode `public static int find(int[] array, int element)` rekursiv. Sie soll für ein *aufsteigend* sortiertes Array mittels der *binären Suche* den Index das gesuchte Elements im Array zurückgeben, sofern dieses vorhanden ist. Wenn das Elemente mehrfach vorhanden ist, soll der kleinste Index, wenn es nicht vorhanden ist, soll –1 zurückgegeben werden. Hilfsmethoden sind gestattet.





Präsenzaufgabe - Lösung

- Wir erschaffen eine Suche: BinarySearch.java. (Eine binäre Suche)

- Die Idee:
  - Wir markieren das zu durchsuchende Fenster mit `left` und `right`.
  - Ist das Fenster leer, haben wir das Element nicht gefunden.
  - Sonst prüfen wir: Ist das mittlere Element das Gesuchte?
  - Ist es kleiner, suchen wir links, ist es größer, rechts weiter.

1. Basisfall: keine Elemente mehr.
2. Basisfall: Element gefunden.
Sonst: Suche links/rechts.

- Dafür nutzen wir die Hilfsmethode:
  `binarySearch(int[] array, int left, int right, int element).`

- Wir beginnen mit der Suche im gesamten Array:
  `binarySearch(array, 0, array.length - 1, element);`

Präsenzaufgabe - Lösung

```
static int binarySearch(int[] array, int left, int right, int element) {
    // 1) Fenster ist leer
    if(right < left)
        return -1;

    int middle = left + (right - left) / 2;
    // 2) Ist das Mittlere das Gesuchte?
    if(array[middle] == element)
        return middle;
    // 3) Kleiner: suche links weiter
    else if(array[middle] > element)
        return binarySearch(array, left, middle - 1, element);
    // 4) Größer: suche rechts weiter
    else
        return binarySearch(array, middle + 1, right, element);
}
```

Präsenzaufgabe - Lösung

- Einen Teil haben wir noch nicht gemacht:

  Sollte das Elemente mehrfach vorhanden sein, soll der kleinste Index zurückgegeben werden

- Idee: Wir verringern den Index, solange das Element immer noch das Gesuchte ist:

```java
public static int find(int[] array, int element) {
    int index = binarySearch(array, 0, array.length - 1, element);
    if(index == -1) return -1;

    for(int i = 1; i <= index; i++) {
        if(array[index - i] != element)
            return index - i + 1;
    }
    return 0;
}
```

Rekursive Variante (etwas zu lang)

```java
// return findLower(array, element, index, 1);
int findLower(int[] arr, int e, int idx, int i) {
    if(i > idx)
        return 0;
    if (arr[idx - i] != e)
        return idx - i + 1;
    return findLower(arr, e, idx, i + 1);
}
```

Übungsblatt 9 - Aufgabe 1

- Die Datei befindet sich hier: CompoundInterest.java

```
for(int i = years; i > 0; i--) {
    capital *= (1 + interestRate);
}
```

```
public static
double compoundInterest(double capital, double interestRate, int years) {
    if(years == 0)
        return capital;
    else
        return compoundInterest(capital * (1 + interestRate), interestRate,
            years - 1);
}
```

Warum die herunterzählende Variante? Nun, sie spart uns eine weitere Variable in der Rekursion. So müssen wir nur prüfen, wann `years` den Wert `0` erreicht. Andernfalls wäre eine weitere Variable notwendig um das ursprüngliche `years` zu halten.

Übungsblatt 9 - Aufgabe 2

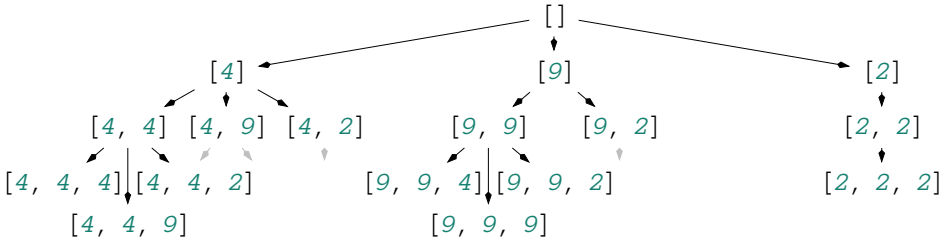
- Die Datei befindet sich hier: Palindrome.java

```
public static boolean isPalindrome(String s) { // die "Hilfsmethode"
    return isPalindromeRecursive(s.toLowerCase());
}


private static boolean isPalindromeRecursive(String s) {
    if(s.length() < 2) // Basisfall: weniger als zwei Zeichen (Abrunden)
        return true;
    else if(s.charAt(0) != s.charAt(s.length() - 1)) // Basisfall
        return false;
    else // Rekursionsfall
        return isPalindromeRecursive(s.substring(1, s.length() - 1));
}
```

Übungsblatt 9 - Aufgabe 3

Da wir die Maximalgröße der Ausgabekombinationen kennen, wird das Hinzufügen eher in der Form $[0,0,0] \rightarrow [1,0,0] \rightarrow [1,2,0] \rightarrow [1,2,3]$ erfolgen.

- Zuerst die Idee:
  - Wir verwenden ein Array um die Ausgabekombination zu halten.
  - Basisfall: das Array hat die gewünschte Länge, dann geben wir es aus.
  - Sonst: erweitern wir das Array für jedes Symbol einmal und erzeugen so eine baumartige Rekursion.
- Ein Beispiel mit $[4, 9, 2]$ und $n = 3$, mit Wiederholung:



Übungsblatt 9 - Aufgabe 3

- Die Datei befindet sich hier: Combinations.java und CombinationsRepetition.java.
- Zunächst das rekursive Ausgeben eines Arrays.
- Idee: wir inkrementieren rekursiv einen Zähler anstelle der for-Schleife.

```
public static void printArray(int[] combination, int i) {
    if(i >= combination.length) { // Basisfall
        System.out.println(); // Ende der Ausgabe
    } else { // Rekursionsfall: ausgeben und nächstes
        System.out.print(combination[i] + "␣");
        printArray(combination, i + 1);
    }
}
```

Übungsblatt 9 - Aufgabe 3

- Für das baumartige Antackern bauen wir uns eine Hilfe (zuerst mit Wiederholung):

```
static void helper(int[] array, int n, int[] combs, int start, int index)
```

- Dabei sind `array` und `n` die gegebene Eingabe.
- `combs` ist das zu konstruierende Array.
- `start` beschreibt ab welchem Feld aus `array` gezogen werden soll.
- `index` ist das aktuell in `combs` zu füllende Feld.

Übungsblatt 9 - Aufgabe 3

- Die Baumrekursion gestaltet sich wie folgt:

```
static void helper(int[] arr, int n, int[] combs, int start, int idx) {
    if (idx == n) { // Basisfall
        printArray(combs, 0);
    } else { // Sonst: Für jedes Symbol...
        for (int i = start; i < arr.length; i++) {
            combs[idx] = arr[i];
            helper(arr, n, combs, i, idx + 1);
        }
    }
}
```

- Aber halt… Arrays sind komplexe Datentypen. Warum geht das? (Tipp: simulieren.)

Übungsblatt 9 - Aufgabe 3 — Rekursionsstart

- Der Start der Rekursion ist nun recht einfach:

```
public static void printCombinations(int[] array, int n) {
    int[] combination = new int[n]; // initialisiert mit 0
    helper(array, n, combination, 0, 0);
}

private static void helper(int[] array, int n, int[] combs, int start, int index) {
    if (index == n) { printArray(combs, 0); }
    else {
        for (int i = start; i < array.length; i++) {
            combs[index] = array[i];
            helper(array, n, combs, i, index + 1);
        }
    }
}
```

Übungsblatt 9 - Aufgabe 3 — Wiederholungen vermeiden

- Wir inkrementieren `i` einfach um 1 im rekursiven Aufruf, damit es nicht wieder das selbe Feld wählen kann.

```
static void helper(int[] array, int n, int[] combs, int start, int index) {
    if (index == n) {
        printArray(combs, 0);
    } else {
        for (int i = start; i < array.length && array.length - i >= n - index; i++) {
            combs[index] = array[i];
            helper(array, n, combs, i + 1, index + 1);
        }
    }
}
```

- `start` von unten: nach der Wahl eines i-ten Elements nur noch nach $i + 1$ wählen.
- `array.length - i >= n - index` sichert optional zu, dass noch mindestens `n - index` Elemente („weiter rechts“) im Array wählbar bleiben. (Bonusfrage fürs Tutorium: Warum optional?)

You expected… a recursion gag? Read this line again!

Blatt 10 — Sortierverfahren

KW 4

Bitte anstellen…

12

Implementieren Sie eine (Linked-) `Queue` Klasse, die nach dem *FIFO*-Prinzip („first in first out“) Integer-Werte verwaltet. Implementieren Sie dafür:

- Den Konstruktor `public Queue()`, der alles *explizit* zuweist.
- `public void enqueue(int value)` — füge `value` am Ende an.
- `public boolean dequeue()` — entferne das vorderste Element (*`false`* wenn leer).
- `public int getLength()` — Anzahl der Elemente in der Queue.
- `public String toString()` — die Inhalte als String formatiert.

Bonus: Welchen Rückgabetyp würden Sie für Die Methode `public ⟨?⟩ head()` wählen, die den Wert des ersten Elements zurückgibt. Warum?

Hilfsmethoden sind erlaubt ☺.

Verwenden Sie für ihre Implementierung keine Arrays oder vorgefertigten dynamischen Datenstrukturen, sondern implementieren Sie die Queue selbst als (einfach) verkettete Liste.

```
class Queue {
  class Element {
    public final int value;
    public Element next = null;
    public Element(int value) {
      this.value = value;
  } }
  // ...
}
```

Präsenzaufgabe - Lösung

- Wir beginnen mit der Grundstruktur: Queue.java

```
public class Queue {
    class Element {
        private final int value;
        private Element next;
        // ...
    }

    private Element first;
    private Element last;
    private int length;
    // ...
}
```

Präsenzaufgabe - Lösung

Element - Alternative

```
public class Queue {
    class Element {
        private final int value;
        private Element next;
        public Element(int value) {
            this.value = value;
            this.next = null;
        }

        public void setNext(Element next) { this.next = next; }
        public Element getNextElement() { return this.next; }
        public int getValue() { return this.value; }
    }
    // ...
}
```

Oder kompakter:

```
class Element {
  public final int value;
  public Element next = null;
  public Element(int value) {
    this.value = value;
  }
}
```

```
public class Queue {
    class Element { ... }
    private Element first, last;
    private int length;

    public Queue() {
        this.first = null;
        this.last = null;
        this.length = 0;
    }
}
```

Präsenzaufgabe - Lösung

Queue - Enqueue

```
public class Queue {
    class Element {
        public final int value; public Element next = null;
        public Element(int value) { this.value = value; }
    }
    private Element first, last; private int length;
    public void enqueue(int value) {
        Element newElement = new Element(value);
        if(length == 0) { // Sonderfall: Erstes
            this.first = this.last = newElement;
        } else {
            this.last.next = newElement;
            this.last = newElement; // Verschieben
        }
        length++;
    }
}
```

Präsenzaufgabe - Lösung

Queue - Dequeue

```
public class Queue {
    class Element {
        public final int value; public Element next = null;
        public Element(int value) { this.value = value; }
    }
    private Element first, last; private int length;
    public boolean dequeue() {
        if(length > 0) {
            this.first = this.first.next; // Verschieben
            length--;
            if(length == 0)
                this.last = null;
            return true;
        }
        return false;
    }
}
```

Präsenzaufgabe - Lösung — Queue - getLength & toString

```
public class Queue {
    class Element {
        public final int value; public Element next = null;
        public Element(int value) { this.value = value; }
    }
    private Element first, last; private int length;
    public int getLength() { return this.length; }
    public String toString() {
        if(length <= 0) return "[]";

        String s = "[" + this.first.value;
        for (Element cur = this.first.next; cur != null; cur = cur.next) {
            s += ", " + cur.value;
        }
        return s + "]";
    }
}
```

Präsenzaufgabe - Lösung

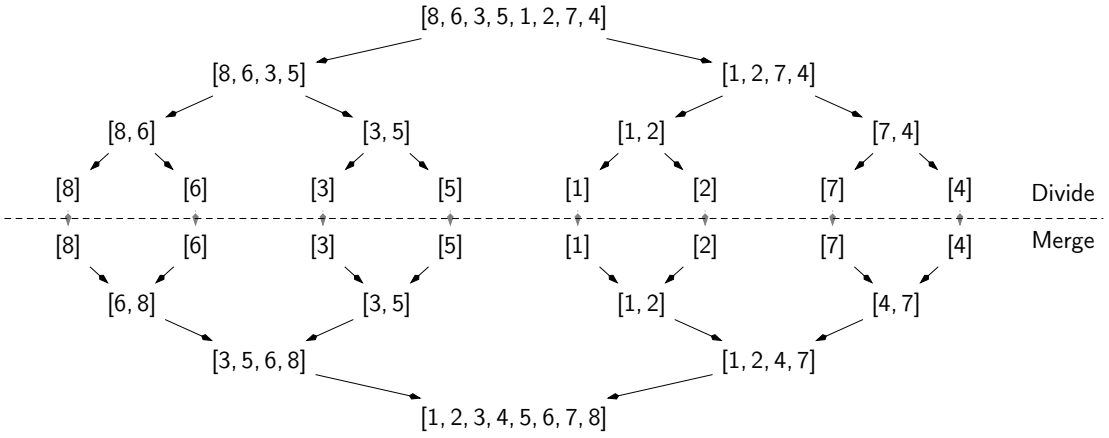
- Doch was machen wir mit `public ⟨?⟩ head()`?
- Idee: `int`.
  - Problem: Was ist, wenn die Queue kein Element hat?
  - Möglichkeit: Wir können eine Exception Werfen!
- Idee: `Integer`.
  - Warum? Es erlaubt `null` als Rückgabe.
  - Problem: Wir sollten kein `null` zurückgeben.
  - Abhilfe: Für solche Fälle gibt es `Optional<T>`.
- Idee: Ein eigener ternärer/optionaler Wert (MayQueueValue.java).

```
public class MayQueueValue {
    private final boolean stored; private final int value;
    // ...
}
```

  - Problem: Wirkt ein wenig Overkill für etwas, das eigentlich illegal ist! ⟶ Exception!

Übungsblatt 10 - Aufgabe 1

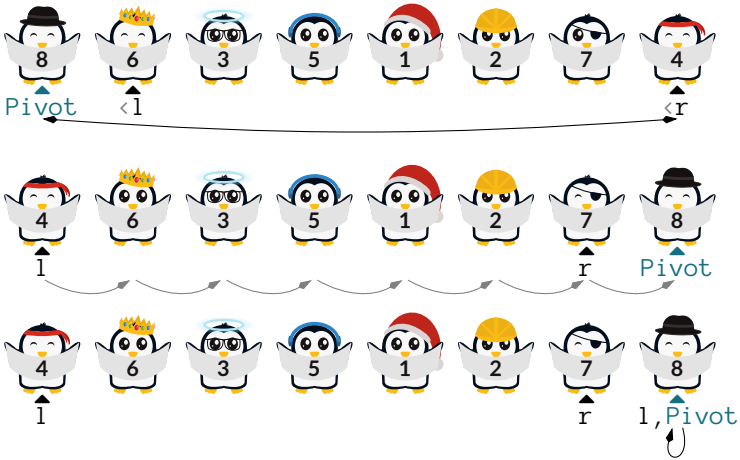
Mergesort



Übungsblatt 10 - Aufgabe 1

- Quicksort kann unterschiedlich angewendet werden.
- Wir betrachten zwei verschiedene Varianten:
  - Die Version der Vorlesung (Pivot ans Ende, Marker)
  - Eine etwas schnellere Version (nur mit Markern).
- Beide folgen dem dem bekannten Schema:
  - Wenn Liste einelementig oder leer: fertig.
  - Wähle Pivotelement und teile Liste in „kleiner“ und „größergleich“.
  - Wiederhole rekursiv für beide Teile.

Übungsblatt 10 - Aufgabe 1 — Quicksort – Vorlesungstreue



Nach der Vorlesung muss das Pivotelement ans Ende!

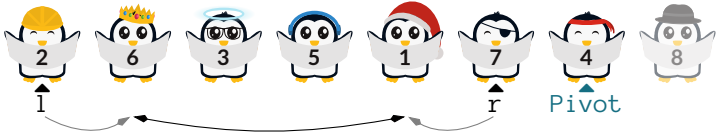
`l` geht nach rechts, solange es auf ein Element zeigt, das kleiner als das Pivotelement ist.

Es ist $l \geq r$. Tausche `l` mit dem Pivotelement (hier die selben).

Übungsblatt 10 - Aufgabe 1 — Quicksort – Vorlesungstreue

Wir wiederholen den Prozess für die linke und die rechte Hälfte. Hier gibt es nur eine linke…

Wir tauschen sofort. r zeigt auf ein Element kleiner und l auf eines größer gleich dem Pivotelement.



Aaaand we continue with the (little) steppsies. And the swappsies.

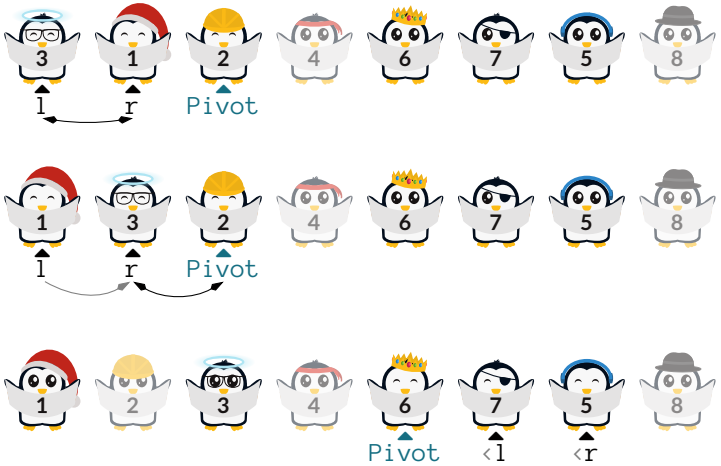
Übungsblatt 10 - Aufgabe 1 — Quicksort – Vorlesungstreue

Und sie kommen ganz nah zusammen. . . .

Es ist l = r, wir vertauschen also l und das Pivotelement.

Meet me at my happy place. We go to the left.

Übungsblatt 10 - Aufgabe 1 — Quicksort – Vorlesungstreue



Für die linke Hälfte machen wir ein swappsie.

Weiter geht l eins nach rechts, da es nun auf ein kleineres Element zeigt. Nun hat es r getroffen, wir tauschen also l und Pivot.

Eigentlich gings rekrusiv weiter. Einelementige oder sortierte Listen lassen sich durch Kommentare überspringen („passiert ja nichts“).

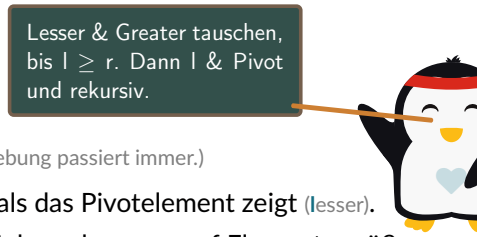
Übungsblatt 10 - Aufgabe 1 — Quicksort – Vorlesungstreue

Es geht also wieder um ein Dreierpärchen

Es zeigt l auf ein kleineres Element, es geht also eins nach rechts, trifft r und tauscht mit Pivot.

Für alle weiteren Teillisten ist l = r = Pivot. Wir verkürzen das in dieser Variante

Übungsblatt 10 - Aufgabe 1 — Quicksort – Vorlesungstreue



1. Betrachte die gegebene Teilliste (initial die Ganze).
2. Tausche das Pivotelement ans Ende.
3. Setze `l` ganz nach links und `r` ganz nach rechts.
   (Wahlweise auch `r` direkt eins links vom Pivotelement, diese Verschiebung passiert immer.)
4. Schiebe `l` nach rechts solange es auf Elemente kleiner als das Pivotelement zeigt (**l**esser).
5. Nur wenn `l` noch weiter links als `r` ist, schiebe `r` nach links, solange es auf Elemente größer gleich dem Pivotelement zeigt (g**r**eater).
6. Ist `l` nun immer noch weiter links von `r`, dann vertausche die Elemente auf die sie zeigen und wiederhole bei 4. Andernfalls, vertausche `l` und Pivotelement.
7. Wiederhole ab 1 rekursiv für die Teilliste links und rechts vom Pivotelement.

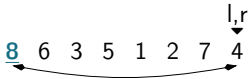
(Der „gleich“-Fall ist an sich beliebig. Man kann also auch `l` für kleiner gleich und `r` für größer als das Pivotelement verwenden. Man muss nur eine der Varianten wählen.)

Übungsblatt 10 - Aufgabe 1 — Quicksort – Nur mit Markern

- Von Hand, müssen wir aber das Pivotelement nirgends „hintauschen“.
- Im Folgenden eine Kompaktversion. Mit den Pivotelementen **P**:

1) Solange a[r] $\geq$ P und a[l] $<$ P

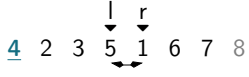
2) Es ist: l $\geq$ r. Hier tauschen wir r und p



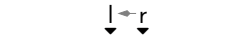
3) Rekursion links. Solange a[r] $\geq$ P und a[l] $<$ P

4) a[l] $\geq$ P und a[r] $<$ P

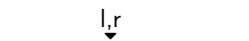
5) Solange a[r] $\geq$ P und a[l] $<$ P



6) a[l] $\geq$ P und a[r] $<$ P



7) Solange a[r] $\geq$ P und a[l] $<$ P (hier nicht, da l $\geq$ r.)

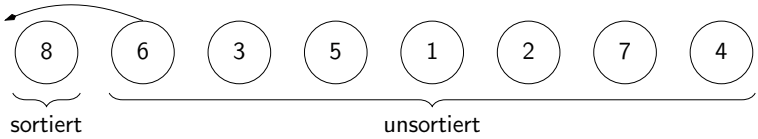


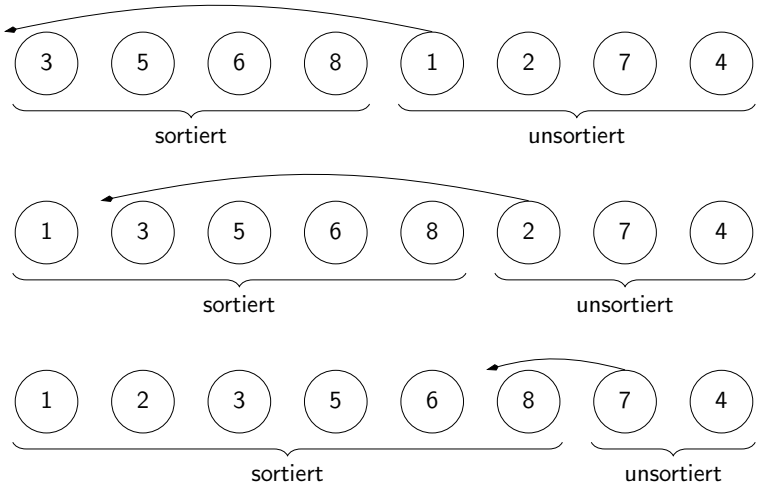
8) Es ist l $\geq$ r.

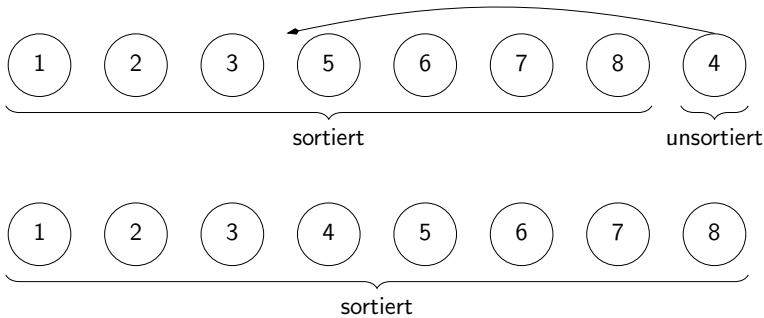
1 2 3 4 5 6 7 8

Fertig sortiert, alle weiteren Anwendungen liefern nichts.

Übungsblatt 10 - Aufgabe 1




3 5 6 8 1 2 7 4
sortiert
unsortiert
1 3 5 6 8 2 7 4
sortiert
unsortiert
1 2 3 5 6 8 7 4
sortiert
unsortiert



- Theoretisch kann man auch alle „Zwischentauschschritte“ einzeichnen.

Übungsblatt 10 - Aufgabe 2

- Iterativ ist das einfach:

```java
public static void bubbleSort(int[] arr) {
    for (int i = arr.length; i > 1; i--) {
        for (int j = 0; j < i - 1; j++) {
            if (arr[j] > arr[j + 1]) { // swap(arr, j, j + 1)
                int tmp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = tmp;
            }
        }
    }
}
```

```java
static void swap(int[] array, int i, int j) {
    int tmp = array[i];
    array[i] = array[j];
    array[j] = tmp;
}
```

Florians Didaktik Hoffnungsversuch: Eigentlich machen wir Rekursion nicht *nur um irgendwas iteratives hinzubekommen*. Es gibt schöne, wundervolle Probleme die sich mit Rekursion wunderschön und elegant lösen lassen. Tolle... wirklich tolle... schnief.

- Wir werden zweimal Rekursion benutzen um die For-Schleifen zu repräsentieren: BubbleSort.java.

Übungsblatt 10 - Aufgabe 2

```java
public static void bubbleSort(int[] array, int n) {
    if (n > 1) {
        move(array, 0, n);
        bubbleSort(array, n - 1);
    }
}
private static void move(int[] array, int i, int n) {
    if (i < n - 1) {
        if (array[i] > array[i + 1])
            swap(array, i, i + 1);
        move(array, i + 1, n);
    }
}
```

```java
public static void bubbleSort(int[] arr) {
    for (int n = arr.length; n > 1; n--) {
        for (int i = 0; i < n - 1; i++) {
            if (arr[i] > arr[i + 1])
                swap(arr, i, i + 1);
        }
    }
}
```

Übungsblatt 10 - Aufgabe 3

1. Was versteht man unter einem stabilen Sortieralgorithmus?

   Ein *stabiler* Sortieralgorithmus verändert nicht die Reihenfolge von Elementen mit dem gleichen *Sortierschlüssel*.
   (Beispiel: Wir sortieren zuerst nach Alter, danach nach Alphabet. Ein stabiler Sortieralgorithmus garantiert, dass Personen mit gleichem Alter immernoch alphabetisch sortiert sind.)

2. Welche der vier auf diesem Übungsblatt vorgekommenen Sortieralgorithmen sind stabil und welche sind instabil?

   Insertionsort, Mergesort und Bubblesort sind stabil, Quicksort ist instabil.
   (Genau genommen kann man aber jeden Algorithmus mit entsprechendem Zusatzaufwand stabil und nicht stabil implementieren. Wir halten uns hier an die Vorlesungsimplementation.)

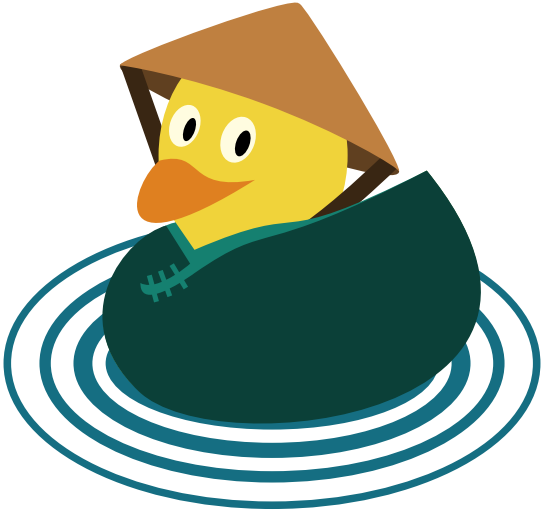
Zusatzaufgabe 4/6

- Das Gute an einer rekursiven Definition wie dieser? Wir können Sie direkt umsetzen!

```
public static int knapsack(int W, int[] weights, int[] values) {
    if(weights.length != values.length) return -1;
    return k(W, 0, weights, values);
}

private static int k(int W, int i, int[] w, int[] v) {
  if (i >= w.length) return 0; // beginnt bei 0: i > n → 0
  if (w[i] > W) return k(W, i + 1, w, v); // w_i > W_max → k(W_max, i + 1)
  else return Math.max(k(W, i + 1, w, v), // sonst: max(„ohne“,
    k(W - w[i], i + 1, w, v) + v[i]); // „mit“)
}
```

Wir nutzen `w.length` für `n` und müssen beachten, dass es bei 0 losgeht.

Knapsack.java



Wieder kein Pingu, weil süß!

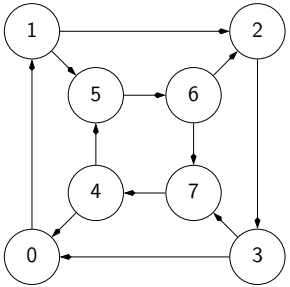
Blatt 11 — Stacks und Buffer

KW 5

Präsenzaufgabe

13 **Nächste Ausfahrt links**

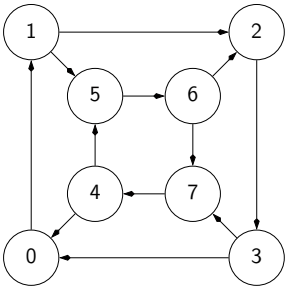
Geben Sie die Adjazenzliste für den Graphen links an und zeichnen Sie den Graphen für die Adjazenzmatrix rechts.



$$\begin{pmatrix} 0 & 3 & 0 & 0 & 6 \\ 3 & 0 & 5 & 0 & 7 \\ 0 & 5 & 0 & 1 & 9 \\ 4 & 0 & 1 & 0 & 2 \\ 6 & 7 & 9 & 2 & 0 \end{pmatrix}$$

Präsenzaufgabe - Lösung

- Zuerst eine Adjazenzliste:



$0 \to 1$

$1 \to 2 \to 5$

$2 \to 3$

$3 \to 7 \to 0$

$4 \to 5 \to 0$

$5 \to 6$

$6 \to 2 \to 7$

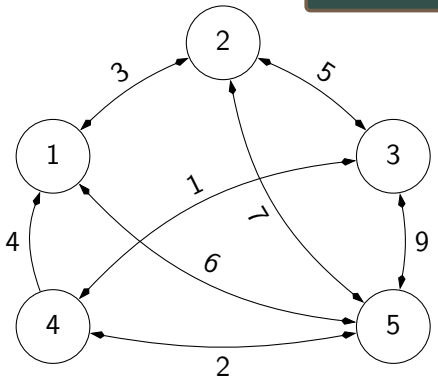
$7 \to 4$

Präsenzaufgabe - Lösung

- Nun mit der Adjazenzmatrix (Kanten wie ⟷ können auch ungerichtet dargestellt werden):

Natürlich können die Kanten in die andere Richtung auch ein anderes Gewicht aufweisen.

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0 & 3 & 0 & 0 & 6 \\ 3 & 0 & 5 & 0 & 7 \\ 0 & 5 & 0 & 1 & 9 \\ 4 & 0 & 1 & 0 & 2 \\ 6 & 7 & 9 & 2 & 0 \end{pmatrix} \end{array}$$



Übungsblatt 11 - Aufgabe 1a) Stack

- Unser Stack baut sich erstmal ganz analog zur Queue von letzter Woche (RPNStack.java).

```
public class RPNStack {
    class Element {
        public final int value;
        public Element next = null;
        public Element(int value) { this.value = value; }
    }
    private Element first;
    // private Element last;
    private int size;
}
```

- Allerdings nehmen wir bei einem Stack die Elemente nur von einer Seite.
- Wir sparen uns also `last` und nennen die Klasse nun `RPNStack`.

Übungsblatt 11 - Aufgabe 1a) Stack push

- Anstelle von `enqueue` und `dequeue` nun `push` und `pop`:

```
public class RPNStack {
    class Element { final int value; Element next = null; }
    private Element first;
    private int size;
    public void push(int value) {
        Element top = new Element(value);
        top.next = this.first;
        this.first = top;
        this.size++;
    }
}
```

Übungsblatt 11 - Aufgabe 1a) Stack pop

```
public class RPNStack {
    class Element { final int value; Element next = null; }
    private Element first;
    private int size;
    public void push(int value) { ... }

    public int pop() {
        if(this.size == 0) throw new NoSuchElementException();
        int value = this.first.value;
        this.first = this.first.next;
        this.size--;
        return value;
    }
}
```

Übungsblatt 11 - Aufgabe 1b)

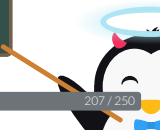
- Wir können für jede Operation eine Methode wie folgt implementieren:

```
public void subtract() {
    if(this.size < 2) throw new NoSuchElementException(); // Nötig? → Nö
    int a = this.pop();
    int b = this.pop();
    this.push(b - a); // this.push(-this.pop() + this.pop())
}
```

- Aber wo wäre da der Spaß? Wir wollen eine Enumeration! (Als Übung. Natürlich.)

```
public class RPNStack {
    class Element { final int value; Element next = null; }
    ...
   private enum BinaryOperation {
      ADD, SUB, MUL
   }
}
```

Aufgabe für die #coolenKids™ Wie kann man das mit Lambdas/Methoden Referenzen noch mehr #exciting machen?

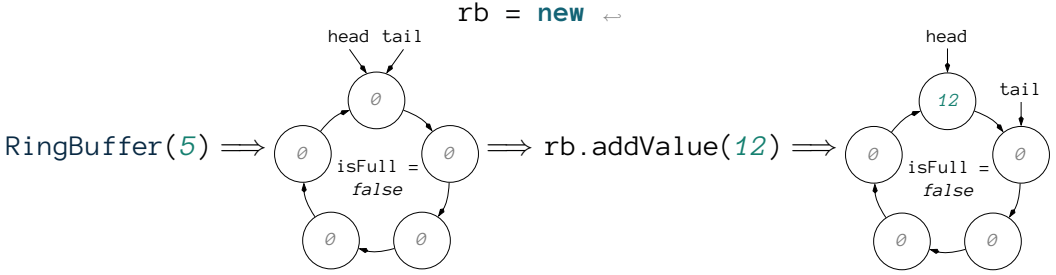


Übungsblatt 11 - Aufgabe 1b) BinaryOperation

```
public class RPNStack {
    ...
    public void push(int value) { ... }
    public int pop() throws NoSuchElementException { ... }
    private enum BinaryOperation { ADD, SUB, MUL }
    private void performBinaryOperation(BinaryOperation operation) {
        if(this.size < 2) throw new NoSuchElementException(); // Nötig? → Nopsie
        int a = this.pop();
        int b = this.pop();
        switch(operation) {
            case ADD: this.push(a + b); break;
            case SUB: this.push(b - a); break;
            case MUL: this.push(a * b); break;
        }
    }
    public void add() { this.performBinaryOperation(BinaryOperation.ADD); }
    public void subtract() { this.performBinaryOperation(BinaryOperation.SUB); }
    public void multiply() { this.performBinaryOperation(BinaryOperation.MUL); }
}
```

Übungsblatt 11 - Aufgabe 2

- Von uns wird ein einfacher Ringbuffer verlangt.
  - Idee: Eine veränderbare `Element`-Klasse.
  - Wir generieren einen Kreis mit gegebener Kapazität.
  - Wir rotieren mit `head`.



Übungsblatt 11 - Aufgabe 2 Grundkonstrukt

- Wir beginnen wieder mit der Queue-Idee (RingBuffer.java).

```
public class RingBuffer {
    class Element {
        public /* final */ int value; // Jetzt veränderbar
        public Element next = null;
        public Element(int value) { this.value = value; }
    }
    private Element head;
    private Element tail;
    private boolean isFull;
}
```

- Hier heißen Start und Ende eben `head` und `tail`.
- Weiter soll `Element` veränderbar sein.
- Anstelle der Länge speichern wir zunächst einfach mal, ob der Buffer voll ist.

Übungsblatt 11 - Aufgabe 2 — Grundkonstruktor

- Der Konstruktor übernimmt hier die Arbeit, eine „leere“ Ringliste zu konstruieren.
- Der naive Ansatz:

```
public class RingBuffer {
    class Element { int value; Element next = null; }
    private Element head, tail;
    private boolean isFull;
    public RingBuffer(int capacity) {
        this.isFull = capacity <= 0;
        if(this.isFull) return;

        Element current = new Element(0); // Magic-Number-Standard
        this.head = this.tail = current;

        // TODO: füllen
    }
}
```

Übungsblatt 11 - Aufgabe 2 — Grundkonstruktor, II

```
public class RingBuffer {
    class Element { int value; Element next = null; }
    private Element head, tail;
    private boolean isFull;
    public RingBuffer(int capacity) {
        // ...
        Element current;

        for(int i = 1; i < capacity; i++) {
            Element next = new Element(0);
            current.next = next;

            if(i == capacity - 1) // Verbindung des letztes Elementes
                next.next = this.head;

            current = next;
        }
    }
}
```

Übungsblatt 11 - Aufgabe 2


```
public class RingBuffer {
    class Element { int value; Element next = null; }
    private Element head, tail;
    private boolean isFull;
    public RingBuffer(int capacity) { ... }

    public void addValue(int value) {
        if ((this.head == this.tail) && this.isFull) // sind wir wirklich voll?
            throw new BufferOverflowException(); // java.nio!

        this.tail.value = value;
        this.tail = this.tail.next;

        // was wäre mit equals?
        if (this.tail == this.head)
            this.isFull = true;
    }
}
```

Objects.equals(tail, head) wäre keine gute Idee. Es funktioniert aus zwei Gründen: 1) *Objects*::equals handhabt ein tail = **null**, welches aber nur bei capacity = *0* auftritt (es ist ja sonst ein „Ring“) und 2) wir haben es nicht für *Element* implementiert. Die Standardimplementation ist einfach mit ==. Dennoch: Wir wollen hier die Identität und *nicht* die Gleichheit prüfen. Das ist ein Fall, in dem == viel besser ist.

Übungsblatt 11 - Aufgabe 2

Take It

```
public class RingBuffer {
    class Element { int value; Element next = null; }
    private Element head, tail;
    private boolean isFull;
    public RingBuffer(int capacity) { ... }
    public void addValue(int value) { ... }
    public int getValue() {
        if ((this.head == this.tail) && !this.isFull) // sind wir leer?
            throw new NoSuchElementException();

        this.isFull = false;
        int value = this.head.value;
        this.head = this.head.next; // Nicht direkt gefordert, aber logisch :D

        return value;
    }
}
```

Übungsblatt 11 - Aufgabe 2 Thats It

- Ein paar Ideen, die das Ganze interessanter machen.
- Anstelle einfach `this.tail.value = value` zu setzen, kann man probieren immer neue `Element`-Objekte zu erzeugen und einzufügen und wieder zu entfernen.
- Erlauben, dass die Kapazität auch nach der Erstellung verändert werden kann (vergrößern und verkleinern).
- Wenn der Ringbuffer voll ist, soll das älteste `Element` durch das neue ersetzt werden.

Übungsblatt 11 - Zusatzaufgabe 5/6 Bubblesort

```java
static void bubbleSort(int[] arr) {
    for (int n = arr.length; n > 1; n--) {
        for (int i = 0; i < n - 1; i++) {
            if (arr[i] > arr[i + 1])
                swap(arr, i, i + 1);
        }
    }
}
```

```java
static void bubbleSort(int[] arr, int n) {
    if (n > 1) {
        move(arr, 0, n);
        bubbleSort(arr, n - 1);
    }
}

static void move(int[] arr, int i, int n) {
    if (i < n - 1) {
        if (arr[i] > arr[i + 1])
            swap(arr, i, i + 1);
        move(arr, i + 1, n);
    }
}
```

Show me the moves (ShakerSort.java)

Übungsblatt 11 - Zusatzaufgabe 5/6 — Ein Blick auf move

```java
static void moveUp(int[] array, int index, int end) {
    if (index < end) { // Exemplarisch mit dem unteren
        if (array[index] < array[index - 1])
            swap(array, index, index - 1);
        return moveUp(array, index + 1, end);
    }
}

static void moveDown(int[] array, int index, int end) {
    if (index > end) {
        if (array[index] < array[index - 1])
            swap(array, index, index - 1);
        return moveDown(array, index - 1, end);
    }
}
```

```java
static void move(int[] arr, int i, int n) {
  if (i < n - 1) {
    if (arr[i] > arr[i + 1])
      swap(arr, i, i + 1);
    move(arr, i + 1, n);
  }
}
```

Übungsblatt 11 - Zusatzaufgabe 5/6 Ping Pong

```java
static void shakerSort(int[] array, int n) {
    int offset = array.length - n;
    if (n > array.length / 2) { // perförmänce büüst
        moveUp(array, offset + 1, n);
        moveDown(array, n - 1, offset);

        shakerSort(array, n - 1);
    }
}
```

```java
static void bubbleSort(int[] arr, int n) {
  if (n > 1) {
    move(arr, 0, n);
    bubbleSort(arr, n - 1);
  }
}
```

Übungsblatt 11 - Zusatzaufgabe 5/6 Effizienter

```
static boolean moveUp(int[] array, int index, int end, boolean swapped) {
    if (index < end) {
        if (array[index] < array[index - 1]) {
            swap(array, index, index - 1);
            swapped = true;
        }
        return moveUp(array, index + 1, end, swapped);
    }
    return swapped;
}

static boolean moveDown(int[] array, int index, int end, boolean swapped) {
    if (index > end) {
        if (array[index] < array[index -1]) {
            swap(array, index, index - 1);
            swapped = true;
        }
        return moveDown(array, index - 1, end, swapped);
    }
    return swapped;
}
```

Übungsblatt 11 - Zusatzaufgabe 5/6 Effizienter, II

```
public static void shakerSort(int[] array, int n) {
    int offset = array.length - n;
    if (n > array.length / 2) {
        if(!moveUp(array, offset + 1, n, false)) return;
        if(!moveDown(array, n - 1, offset, false)) return;

        shakerSort(array, n - 1);
    }
}
```


Li(s)t!

Blatt 12 — Bäume und Minimax

KW 6

Präsenzaufgabe

14

Ich binde mich gleich doppelt an dich

In dieser Aufgabe sollen Sie ihre ersten Erfahrungen mit Java Interfaces machen. Im Moodle finden Sie als Vorlage für diese Aufgabe die Klasse `DoubleLinkedList` welche eine doppelt-verkettete Liste repräsentiert die Integer Werte speichern kann. Außerdem das Interface `SeekAndRemove`, welches drei verschiedene Methoden definiert um Elemente zu suchen und diese zu Entfernen.

- Die Methode `public void remove(int value)` soll die Liste beginnend beim vordersten Element durchsuchen, und das erste Element entfernen welches den als Parameter übergebenen Wert hat.
- Die Methode `public void removeLast(int value)` soll die Liste beginnend beim hintersten Element durchsuchen, und das erste Element entfernen welches den als Parameter übergebenen Wert hat.
- Die Methode `public void removeAll(int value)` soll alle Elemente entfernen die den als Parameter übergebenen Wert haben.

Präsenzaufgabe

15

Ich binde mich gleich dreifach-doppelt an dich

Gegeben sei die `DoublyLinkedList`-Klasse. Implementieren Sie das Interface `SeekAndRemove`:

```
public class DoublyLinkedList implements SeekAndRemove {
    class Element { final int value; Element prev = null, next = null; }

    private Element head, tail = null;
    private int length = 0;

    public DoublyLinkedList() { }
    public void addFront(int value) { ... }
    public void addBack(int value) { ... }
}
```

Hilfsmethoden sind erlaubt. Diese können hier auch helfen, redundanten Code zu vermeiden.

```
public interface SeekAndRemove {
  // Entferne erstes Element mit Wert <v>
  public void removeFirst(int v);
  // Entferne letztes Element mit Wert <v>
  public void removeLast(int v);
  // Entferne alle Elemente mit Wert <v>
  public void removeAll(int v);
}
```

Präsenzaufgabe - Lösung

- Wir schreiben eine Methode, die ein Element gegeben der Referenz entfernt:

```
private void removeElement(Element element) {
    if(element == null || length == 0) return; // Nichts zu tun

    if(element == this.head) this.head = element.next; // Erstes
    if(element == this.tail) this.tail = element.prev; // Letztes

    if(element.prev != null) // überspringe prev
        element.prev.next = element.next;
    if(element.next != null) // überspringe next
        element.next.prev = element.prev;
    length--;
}
```

Präsenzaufgabe - Lösung — removeFirst(int)

- Jetzt ist die Erfüllung des Interfaces leicht (SeekAndRemove.java und DoublyLinkedList.java):

```
public void removeFirst(int value) {
    Element current = head; // von vorne nach hinten...
    while(current != null) {
        if(current.value == value) {
            removeElement(current);
            break;
        }
        current = current.next;
    }
}
```

Präsenzaufgabe - Lösung removeLast(int)

```
public void removeLast(int value) {
    Element current = tail; // von hinten nach vorne...
    while(current != null) {
        if(current.value == value) {
            removeElement(current);
            break;
        }
        current = current.prev;
    }
}
```

Präsenzaufgabe - Lösung

removeAll(int)

```
public void removeAll(int value) {
    Element current = head; // von wo nach wo ist egal
    while(current != null) {
        if(current.value == value)
            removeElement(current);

        current = current.next;
    }
}
```

Übungsblatt 12 - Aufgabe 1a)

- Ein Dateien Palast: IntegerNode.java, Queue.java und BinaryTree.java.
- Wir bauen einen binären Baum (ich glaub es kaum. Zauuuuun):

```
public BinaryTree(int[] items) {
    this.root = buildTree(items, 0);
}
private static IntegerNode buildTree(int[] items, int index) {
    if (index >= items.length) return null;

    IntegerNode node = new IntegerNode(items[index]);
    node.setLeftChild(buildTree(items, 2 * index + 1)); // left
    node.setRightChild(buildTree(items, 2 * index + 2)); // right
    return node;
}
```

Übungsblatt 12 - Aufgabe 1b)

- Der Breitendurchlauf:

```
public void breadthFirstTraversal() {
    Queue queue = new Queue();
    queue.enqueue(root);

    while (!queue.isEmpty()) {
        IntegerNode node = queue.dequeue();
        if(node == null) continue;

        queue.enqueue(node.getLeftChild());
        queue.enqueue(node.getRightChild());

        System.out.print(node.getValue() + "␣");
    }
}
```

Übungsblatt 12 - Aufgabe 2.1)

- Zusätzliche Freu(n)de: StringNode.java und BinaryExpressionTreeStringNode.java

```
public class StringNode {
    private String item;
    private StringNode left;
    private StringNode right;

    public void setRightChild(StringNode right) { this.right = right; }
    public StringNode getRightChild() { return this.right; }
    public void setLeftChild(StringNode left) { this.left = left; }
    public StringNode getLeftChild() { return this.left; }
    public void setItem(String item) { this.item = item; }
    public String getItem() { return this.item; }

    public boolean isLeaf() { return left == null && right == null; }

    public StringNode(String v) { this.item = v; this.right = this.left = null; }
}
```

Übungsblatt 12 - Aufgabe 2.2)

- Die Grundlage des Baumes ist wie bekannt:

```
public class BinaryExpressionTree {
    private StringNode root;
    public BinaryExpressionTree(String[] items) {
        this.root = buildTree(items, 0);
    }
    private static StringNode buildTree(String[] items, int index) {
        if (index >= items.length) return null;

        StringNode node = new StringNode(items[index]);
        node.setLeftChild(buildTree(items, 2 * index + 1));
        node.setRightChild(buildTree(items, 2 * index + 2));
        return node;
    }

    // ...
}
```

Übungsblatt 12 - Aufgabe 2.2)

- Die Auswertung

```
public class BinaryExpressionTree {
  private StringNode root;
  // ...
  public double evaluate() { return evaluate(root); }
  private double evaluate(StringNode node) { // Postorder
    if (node == null) throw new IllegalStateException("Traversing empty");
    if (node.isLeaf()) return Double.parseDouble(node.getItem());
    double valueLeft = evaluate(node.getLeftChild());
    double valueRight = evaluate(node.getRightChild());

    switch (node.getItem()) { // Get the Operator
       case "+": return valueLeft + valueRight;
       case "-": return valueLeft - valueRight;
       case "*": return valueLeft * valueRight;
       case "/": return valueLeft / valueRight;
       default: throw new IllegalArgumentException("...");
    }
  }
}
```

Übungsblatt 12 - Aufgabe 3

- Man *kann* das Zeichnen naiv implementieren (DrawBinaryTreeNaive.java).
- Das machen wir hier nicht. We know Math. We love Math (DrawBinaryTree.java).
- Zunächst die Formatierung einer Zahl:

```
static String numberFormat(int[] arr, int i) {
    if(i >= arr.length) return "␣␣";
    return arr[i] > 9 ? Integer.toString(arr[i]) : "␣" + arr[i];
}
```

- Mit `String::format` geht auch: `String.format("%-2d", arr[i]);`

Übungsblatt 12 - Aufgabe 3

- Oder der erweiterbare Weg (wie könnte man das erweitern):

```
public static String numberFormat(int[] array, int i) {
    if(i >= array.length) return "␣".repeat(2);
    int number = array[i];
    int size = (int) Math.log10(number) + 1;
    return number + "␣".repeat(Math.abs(2 - size));
}
```

- Hinweis: Allgemein werde ich `String::repeat` benutzen, anstelle von sowas:

```
public String repeat(String str, int n) {
    String ret = "";
    for(int i = 0; i < n; i++) ret += str;
    return str;
}
```

Ich werde nicht auf die Erweiterbarkeit eingehen. Warum? Manche 2er kommen von den Binärbaumen, manche von der Breite der Zahlen, manche vom Padding. Das ist eine nette Übung.

- Was wir brauchen:
  - Die Größe des Baumes durch $\log_2$ aus der Länge des Arrays:

    ```
    int height = (int) (Math.log(array.length) / Math.log(2));
    ```

  - Die maximale Breite des Baumes. Jeder Knoten braucht dabei 2 Zeichen und zwischen den Knoten gibt es zwei Zeichen Platz. (Wie ginge das mit Konstanten schöner, lesbarer und flexibler?)

    ```
    int width = (2 + 2) * (int) Math.pow(2, height);
    ```

  - Die Position des j-ten Knoten auf Ebene i (man kann auch einfach hochzählen):

    ```
    int index = (int) Math.pow(2, i) + j - 1;
    ```

- Den Abstand links und den Zwischen-Abstand auf Ebene i (erstmal ohne Padding):

```
int levelWidth = (int) Math.pow(2, i);
int padding = width / levelWidth;
int leftPad = width / (levelWidth * 2) - 2; // halb links, halb rechts
```

Übungsblatt 12 - Aufgabe 3 draw(int[])

- High-Level ist das Zeichnen leicht beschrieben:

```
static void draw(int[] array) {
    int height = (int) (Math.log(array.length) / Math.log(2));
    int width = (2 + 2) * (int) Math.pow(2, height);

    for (int i = 0; i <= height; i++) {
        drawNumbers(array, width, i);
        if (i < height) // Für die nächste Ebene:
            drawLines(array, width, i + 1);
    }
}
```

Übungsblatt 12 - Aufgabe 3 drawNumbers(int[], int, int)

```
static void drawNumbers(int[] array, int width, int currentHeight) {
    int levelSize = (int) Math.pow(2, currentHeight);
    int padding = width / levelSize;
    int leftPadding = width / (levelSize * 2) - 2;
    System.out.print("␣".repeat(leftPadding));
    for (int j = 0; j < levelSize; j++) { // "Breitendurchlauf"
        if (j > 0) // Abstand dazwischen
            System.out.print("␣".repeat(padding - 2));
        System.out.print(formatNumber(array, levelSize + j - 1));
    }
    System.out.println();
}
```

Übungsblatt 12 - Aufgabe 3 — drawLines(int[], int, int)

```
static void drawLines(int[] array, int width, int currentHeight) {
    int levelSize = (int) Math.pow(2, currentHeight);
    int padding = width / levelSize;
    int leftPadding = width / (levelSize * 2) - 2;
    System.out.print("␣".repeat(leftPadding) + "+"); // Mind. eine
    for (int j = 0; j < levelSize; j++) {
        if (levelSize + j - 1 >= array.length) break; // Zahlen vorbei?
        if (j > 0 && j % 2 == 0) // Abstand dazwischen
            System.out.print("␣".repeat(padding - 1) + "+");
        System.out.print("-".repeat((padding - 2) / 2) + "+");
    }
    System.out.println();
}
```

Übungsblatt 12 - Zusatzaufgabe III

- Man muss nicht jedes mal neu prüfen (EfficientMinimax.java), wir betrachten es hier aber nur mit (Minimax.java).
- Zunächst, kann man überhaupt noch ziehen:

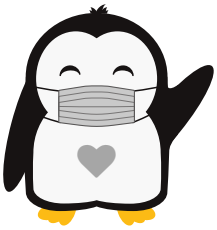
```
static boolean isFull(char[][] board) {
    for (int y = 0; y < 3; y++) {
        for (int x = 0; x < 3; x++) {
            if (board[y][x] == EMPTY)
                return false;
        }
    }
    return true;
}
```

```
static final int LOSE_HUMAN = 10;
static final int WIN_HUMAN = -10;
static final int DRAW = 0;
static final char PLAYER_COMP = 'X';
static final char PLAYER_HUMAN = 'O';
static final char EMPTY = 0;
```

```
static int minimax(char[][] board, char player) {
    // Blatt? Ist es schon vorbei?
    if (winningMove(PLAYER_HUMAN, board)) return WIN_HUMAN;
    else if (winningMove(PLAYER_COMP, board)) return LOSE_HUMAN;
    else if (isFull(board)) return DRAW;
    int bestScore = player == PLAYER_HUMAN ? LOSE_HUMAN : WIN_HUMAN;
    for (int y = 0; y < 3; y++) {
        for (int x = 0; x < 3; x++) {
            if (board[y][x] != EMPTY) continue; // Schon belegt
            board[y][x] = player; // Abstieg: Simuliere Zug
            int score = minimax(board, otherPlayer(player));
            board[y][x] = EMPTY; // Aufstieg: Verarbeite Ergebnis
            // Man verliert nur, wenn es keine andere Möglichkeit gibt:
            if(player == PLAYER_HUMAN) bestScore = Math.min(score, bestScore);
            else bestScore = Math.max(score, bestScore);
        }
    }
    return bestScore;
}
```

- Wir probieren jeden Zug und wählen den besten:

```
static char[][] computerMove(char[][] board) {
    System.out.println("Computer ist am Zug:");
    int bestScore = WIN_HUMAN;
    int bestMove = 1;
    for (int y = 0; y < 3; y++) {
        for (int x = 0; x < 3; x++) {
            if (board[y][x] != EMPTY) continue;
            board[y][x] = PLAYER_COMP;
            int score = minimax(board, PLAYER_HUMAN);
            board[y][x] = EMPTY;
            if (score > bestScore) {
                bestScore = score; bestMove = y * 3 + x + 1; // => Nummerierung
            }
        }
    }
    return makeMove(board, bestMove, PLAYER_COMP);
}
```



Blatt 13 — Vererbung

KW 7

Übungsblatt 13 - Aufgabe 1 a) EquilateralTriangle

- Gimme those classes (EquilateralTriangle.java und Square.java):

```
public class EquilateralTriangle extends RegularPolygon {
    public EquilateralTriangle(int sideLength) {
        super(3, sideLength);
    }

    @Override
    public double getArea() {
        return 0.25 * Math.sqrt(3) * sideLength * sideLength;
    }
}
```

Übungsblatt 13 - Aufgabe 1 a) Square

```java
public class Square extends RegularPolygon {
    public Square(int sideLength) {
        super(4, sideLength);
    }

    @Override
    public double getArea() {
        return sideLength * sideLength;
    }
}
```

Übungsblatt 13 - Aufgabe 1 b-d)

- Ist die Implementation von `getArea()` Überladung oder Überschreiben?

  Überschreiben: die „implementierende“ Methode hat die selbe Signatur (und ist sichtbar)

  (siehe JLS17 8.4.8.1 für die genauen Regeln).

- Könnte die abstrakte Klasse auch durch ein Interface ersetzt werden?

  Das ist aus zwei Gründen nicht möglich:

  1. Die (abstrakte) Klasse hat einen Konstruktor.
  2. Die (abstrakte) Klasse enthält Instanzvariablen.

  Schnittstellen definieren das Verhalten. *Keinen* Zustand!

- Es wird der Modifier **`protected`** für die Instanzvariablen verwendet. Ginge auch **`private`**?

  Für `numSides` ist dies möglich, diese Variable wird nur in `RegularPolygon` gebraucht. Für `sideLength` allerdings nicht, wir benötigen diese für `getArea()`.

  (Man kööönte aber in diesem speziellen Fall auch eine neue Variable machen…)

Übungsblatt 13 - Aufgabe 1 e)

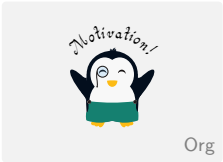
```
public class PolygonTest {
   public static void main(String[] args) {
      RegularPolygon polygon = new EquilateralTriangle(3); // A 1
      EquilateralTriangle triangle = (EquilateralTriangle)polygon; // A 2
      Square square = new RegularPolygon(4, 4); // A 3
   }
}
```

1. Das ist fein, da `EquilateralTriangle` von `RegularPolygon` erbt und damit eine Spezialisierung darstellt! (Polymorphie)
2. Durch die explizite Typkonvertierung ist dies fein. Implizit würde dies nicht funktionieren! So geht das aber, da `polygon` tatsächlich auf eine Instanz vom Typ `EquilateralTriangle` zeigt.
3. Böse, wir können keine Instanz einer abstrakten Klasse erzeugen!

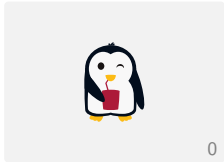
Übungsblatt 13 - Aufgabe 1 f)

```
public class PolygonAreaTest {
    public static void main(String[] args) {
        RegularPolygon polygon = new EquilateralTriangle(3);
        System.out.println(polygon.getArea());

        polygon = new Square(2);
        System.out.println(polygon.getArea());
    }
}
```

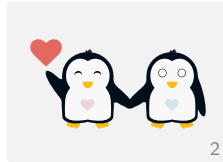
- Beide fein! Durch *dynamische Bindung* wird `EquilateralTriangle::getArea` und `Square::getArea` für `polygon.getArea()` aufgerufen. So wird zunächst der Flächeninhalt des Dreiecks (3.87911...) und dann der Flächeninhalt des Quadrates (4) ausgegeben.



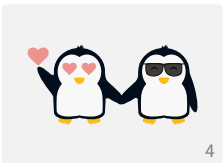
Org

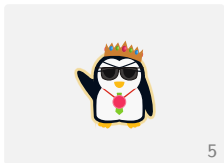

0

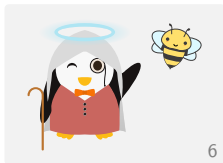

1

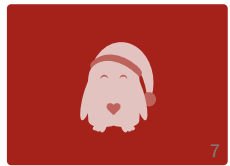

2


3

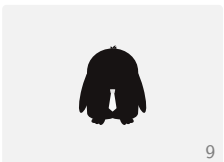

4

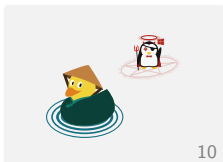

5


6

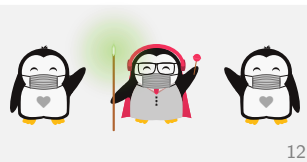

7


8


9


10

11


12

Hat Spaß gemacht. Viel Erfolg an der Klausur! Bis dann und… *waddle on*!

Hat Spaß gemacht. Viel Erfolg an der Klausur! Bis dann und… *waddle on*!