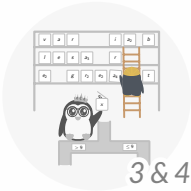
4

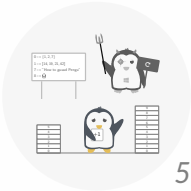
Integral Refs in Higher Dimensions

Tutorium vier

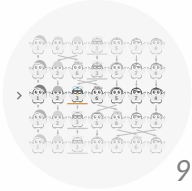
Theorie

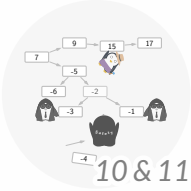
Grundlagen





Vertiefungen





Kurzwiederholung

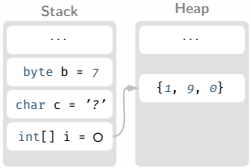
- Wir haben uns Methoden angesehen
  - Methoden mit einem Rückgabetyp „≠ `void`“ müssen für jeden Pfad diesen Typ liefern
  - Methoden haben Vor- (Parameter) und Nachbedingungen (Rückgabetyp)
  - Parameter und lokale Variablen sind nur in ihrer Methode sichtbar
- Für Arrays haben wir for-each

```java
int[] arr = { 1, 9, -2, 3 };
for(int elem : arr)
   System.out.println(elem);
```

Initialisiert automatisch mit {0, 0, 0, 0, 0}

```java
int[] arr = new int[5];
for(int i = 0; i < arr.length; i++)
   System.out.println(arr[i]);
```

- Java verwaltet Daten auf dem Stack (Primitiv) und Heap (Komplex)
  - Java kopiert, vergleicht, schützt, verwirft,… Stackelemente
  - Bei der Ablage auf dem Heap wird auf dem Stack eine Referenz zu diesem Element abgelegt (**`null`** für die „leere“ Referenz)



We can face our problem. We can arrange such facts as we have with order and method.
— Agatha Christie [Chr34]

Präsenzaufgabe

Präsenzaufgabe

1

Flip-em till 'ya hit-em

Implementieren Sie beide Methoden, welche alle im übergebenen `char`-Array enthaltenen Klein- zu Großbuchstaben, und alle Groß- zu Kleinbuchstaben machen.

```
public class Praesenzaufgabe {
    public static void
        flipInPlace(char[] c) { /* TODO */ }
    public static char[]
        flipInCopy(char[] c) { /* TODO */ }
    public static void main(String[] args) { ... }
}
```

A–Z ≙ 65–90
a–z ≙ 97–122

`flipInPlace` soll diese Änderungen direkt im übergebenen Array vornehmen, `flipInCopy` soll auf einer Kopie des Arrays arbeiten und diese zurückgeben. Überlegen Sie sich, welche Vor- und Nachteile die jeweiligen Implementierungen haben.

Flip in Place

```
public static void flipInPlace(char[] flipBuchstaben) {
   for(int i = 0; i < flipBuchstaben.length; i++) {
      if (flipBuchstaben[i] >= 65 && flipBuchstaben[i] <= 90) {
         flipBuchstaben[i] += 32;
      } else if (flipBuchstaben[i] >= 97 && flipBuchstaben[i] <= 122) {
         flipBuchstaben[i] -= 32;
      }
   }
}
```

Wer soll das mit den Zahlen verstehen?

Wir können benannte Konstanten einführen!

```
public static final int A = 65;
...
```

Flip in Place with Constants

```
public static final int A = 65; public static final int a = 97;
public static final int Z = 90; public static final int z = 122;

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

What, if there would be a better way?
What, if we want to check only for characters g–m

Flip in Place with Java-Magic

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

Implizite Typkonvertierung!
char → int

Flip in Copy, Naïve Please

There is a `clone` for every ~~Object~~ „Referenzvariable“

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

Flip in Copy — Custom Mode

Ich kannte `clone` gaaar niiischt. Unnu? Ist das nicht böse?

```
public static char[] flipInCopy(char[] flipBuchstaben) {
   char[] arrayKopie = new char[flipBuchstaben.length];
   for(int i = 0; i < arrayKopie.length; i++)
      arrayKopie[i] = flipBuchstaben[i];

   flipInPlace(arrayKopie);
   return arrayKopie;
}
```

Werte primitiver Datentypen werden bei Zuweisung kopiert. *Praesenzaufgabe.java*, *PraesenzaufgabeNaiv.java*

Allgemeines

- `Clone` widmen wir uns gleich nochmal…
- *Überlegen Sie sich, welche Vor- und Nachteile die jeweiligen Implementierungen haben.*
  - 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.

Alternativen?

- Hätte man hier auch „for-each“ für flip in place nutzen können?

```
for(int i = 0; i < flipBuchstaben.length; i++) {
   if ('A' <= flipBuchstaben[i] && flipBuchstaben[i] <= 'Z') {
      flipBuchstaben[i] += shift;
   } else if ('a' <= flipBuchstaben[i] && flipBuchstaben[i] <= 'z') {
      flipBuchstaben[i] -= shift;
   }
}
```

- Nein, so könnten wir nicht mehr `flipBuchstaben` an der Stelle `i` verändern.
- Mit `for(char c : flipBuchstaben)` ist c jeweils eine Kopie (primitiver Datentyp) und nicht mehr eine Referenz auf die Stelle im Array!

Have you ever heard the tragedy of Darth Cloneable the wise? Well, then prepare yourself for… something at least.
— Florian

Clone-Wars


clone

Die Object Klasse

- `Object` ist die Klasse, welche die grundlegenden Eigenschaften für jede andere Klassen zur Verfügung stellt.
- Wir werden sie später im Rahmen von Vererbung genauer kennen lernen.
  Bis da hin: `toString` stammt beispielsweise dort her.
- Diese Klasse `Object` liefert auch die Methode `clone`, die auf jedem Objekt aufgerufen werden kann… … …
- `clone` muss aber von jeder Klasse selbst und individuell implementiert werden, wenn es denn unterstützt werden soll.

Kopien

- Da die Variable eines Objekts in Java nur eine Referenz auf das eigentliche Objekt enthält, erzeugen wir durch den folgenden Code *keine* Kopie:

```
Scanner a = new Scanner(System.in);
Scanner b = a;
```

- Oft stellen sich auch Fragen wie:
  - Was soll kopiert werden?
  - Funktioniert eine Kopie überhaupt?
  - Was ist, wenn das zu kopierende Objekt selbst wieder (zum Beispiel in den Attributen) Referenzen auf andere Objekte enthält?
  - Was, wenn diese Referenzen zirkulär sind?

Kopie-Varianten

- Deswegen unterscheiden wir zwei Arten von Kopien: *shallow* und *deep*.
  Genau genommen gibt es noch viel mehr, wie zum Beispiel *lazy*, aber das soll uns hier nicht weiter stören.
- Eine *shallow copy* kopiert „so wenig wie möglich“. Oder auch nur „die erste Hierarchieebene.“
- Eine *deep copy* kopiert das komplette Objekt, sowie alle Objekt-Referenzen die dieses Objekt wieder besitzt und so weiter.
- Während shallow copies meist von Hand geschrieben werden, wird eine deep copy meistens durch (De-)Serialisierung gelöst.
  Sonst müssten bei einer deep copy auch alle vom Objekt verwendeten Ressourcen wieder deep-copyable sein. Das Verfahren hat allerdings ebenfalls Nachteile. Stichwort: `transient`.

Kopien, ein Beispiel

- Betrachten wir die folgende Klasse (`Y` und `Z` seien ebenfalls gegeben):

```
class X {
    int a;
    Y y;
    Z z;
}
```

- Eine *shallow copy* würde ein neues `X`-Objekt erzeugen und die Attribute durch „ =“ zuweisen. Damit wird `a` kopiert, `y` und `z` referenzieren aber (je) dasselbe Objekt.
- Eine *deep copy* würde rekursiv auch neue Objekte von `Y` und `Z` erzeugen.
  Quizfrage: was passiert oder besser, was kann alles passieren, wenn `Y` wieder eine Referenz auf `X` enthält?
- Und was davon macht jetzt `Object::clone()`? Spoiler: Nobody knows.

Cloneable

- Das Interface Cloneable *fordert nichts* von einer gegebene Klasse.
- Es ist ein sogenanntes „Marker“ interface. Wer es implementiert, sorgt dafür, dass gewisse Methoden sich einem gewissen Vertrag unterwerfen.

  Diese Erklärung ist an sich nachträglich entstanden. Die komplette Geschichte um das `Cloneable`-Interface ist eine einzige Tragödie.
- Aus der Java-Dokumentation („contract“):

  […] Note that this interface does not contain the clone method. Therefore, it is not possible to clone an object merely by virtue of the fact that it implements this interface. Even if the clone method is invoked reflectively, there is no guarantee that it will succeed. […]

Cloneable, Schadensbericht

- Das Interface widerspricht der sonstigen Schreibweise. (Clonable vs. Cloneable.)
- Es fordert nichts und kann theoretisch von jeder Klasse implementiert werden.
- Es liefert keine weiteren Informationen über die Kopie.
- Es stellt keine Anforderungen an die Java-Syntax (Sichtbarkeit), die nicht gleichermaßen geprüft werden können.
- Der Rückgabewert der `Object::clone`-Methode ist stets `Object`, wir brauchen also einen expliziten Cast. (Beispielsweise bei `(Date) birthday.clone()`.)
- … … …

Long story short

- Manche Java Klassen implementieren `Cloneable` korrekt. (Wie `Date`.)
- Bei Arrays sind die Kopien mit `clone` immer *shallow*.
- Ansonsten haben sich andere Prinzipien durchgesetzt. So beispielsweise Copy-Konstruktoren.

Übungsblatt 4

Aufgabe 1: Referenzvariablen

Betrachten Sie den folgenden Java Code (Referenzvariablen.java):

```
1  public class Referenzvariablen {
2    public static void setzeElementVielleicht(
         int[] array, int idx, int ele) {
3      if (array == null || idx >= array.length
           || ele <= 0) {
4        System.out.println("Eingabe ungültig!");
5        return;
6      }
7      array[idx] = ele;
8      ele = 4;
9      array = null;
10   }
11
12   public static void main(String[] args) {
13     int ele = 1;
14     int idx = ele;
15     ele = 2;
16     int[] array = new int[3];
17     setzeElementVielleicht(array, idx, ele);
18     System.out.println(ele);
19     System.out.println(idx);
20     if (array != null) {
21       for (int wert : array) {
22         System.out.println(wert);
23       }
24     } else {
25       System.out.println("Array ungültig!");
26     }
27   }
28 }
```

a) *Welche Ausgaben erzeugen die Zeilen 18 und 19? Begründen Sie Ihre Antwort.*

b) *Welche Ausgaben erzeugen die Zeilen 20–26? Begründen Sie Ihre Antwort.*

Ein schneller Blick auf knappe Antworten

a) *Welche Ausgaben erzeugen die Zeilen 18 und 19?*

Z18 liefert 2, und *Z19* 1.

Zuerst die Ausgabe von `ele`, dieses wird in *Z13* mit 1 initialisiert und in *Z15* mit 2 überschrieben. Der Parameter `ele` in *Z2* besitzt einen anderen Sichtbarkeits- und Gültigkeitsbereich, zudem wird der Wert durch *Call-by-Value* kopiert. Damit hat das Unterprogramm keinen Einfluss auf den wert in `ele` aus `main`. Auf `ele` findet kein weiterer Zugriff statt, es ist 2.

Analog argumentiert es sich für `idx`, welches in *Z14* mit dem dortigen Wert von `ele` (ist 1) initialisiert wird (da es ein primitiver Datentyp ist, wird hier auch mit *Call-by-Value* der Wert kopiert). Anschließend findet kein verändernder Zugriff statt und `idx` ist 1.

b) *Welche Ausgaben erzeugen die Zeilen 20–26?*

Sie erzeugen „`0\n 2\n 0`“ (wobei \n ein neue Zeile darstellt)

Ein schneller Blick auf knappe Antworten II

a) *Welche Ausgaben erzeugen die Zeilen 18 und 19?*

Z18 liefert 2, und *Z19* 1.

b) *Welche Ausgaben erzeugen die Zeilen 20–26?*

Sie erzeugen „0\n2\n0“ (wobei \n ein neue Zeile darstellt)

Die Zielen befassen sich mit der Ausgabe von `array`. Dieses wird in *Z16* Java initialisiert hier direkt mit `{0, 0, 0}`. Das Unterprogramm erfüllt nun in *Z2* die Referenz zu `array` als Kopie (das ist *Call-by-Value*, verhält sich aber wie *Call-by-Reference*), das **`if`** in *Z3* scheitert, da `array` auf `{0, 0, 0}` verweist und `idx = 1`, `ele = 2` (aus vorheriger Antwort). *Z7* ändert damit das gemeinsame Array zu `{0, 2, 0}` (Seiteneffekt), die Neuzuweisung in *Z9* überschreibt lediglich die in den Parameter kopierte Referenz. Damit löst das **`if`** in *Z20* aus und es erfolgt besagte Ausgabe.

Heap und Stack… Again

- Das könnte man jetzt aufwändig durchanimieren…
- Ich möchte aber, dass ihr das zur Übung macht!
- Wählt einen beliebigen (sinnvollen) Punkt im Programmablauf aus und zeichnet Heap- und Stack zu diesem Zeitpunkt
  - Am Besten im Unteralgorithmus
  - Wie werden wohl „mehrdimensionale“ Arrays dargestellt?
- Hier zeige ich gleich die Animation eines anderen Codes (als Hilfe)

Sprechen über Signaturen

- Wie unterscheidet Java Methoden? Durch ihre Signatur! Und was ist das?
- Der Name und die Parametertypliste! (JLS17 8.4.2)
- Der Rückgabetyp ist dabei *kein Teil* (Hust hust CLR)
- Betrachten wir ein Beispiel:

```
public String foo(int i, double[] dubletti, boolean b) {
    ...
}
```

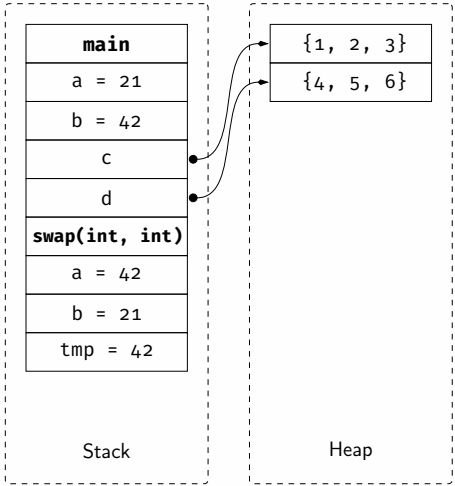
- Die Signatur lautet „`foo(int, double[], boolean)`“
- Das werden wir in Zukunft weiter behandeln und einbetten!

Der Swap-Stack-Heap-Durchlauf

```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);
   }
}
```

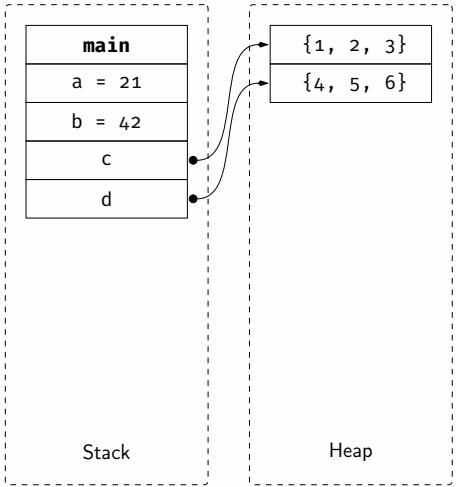


Der Swap-Stack-Heap-Durchlauf

```
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);
   }
}
```

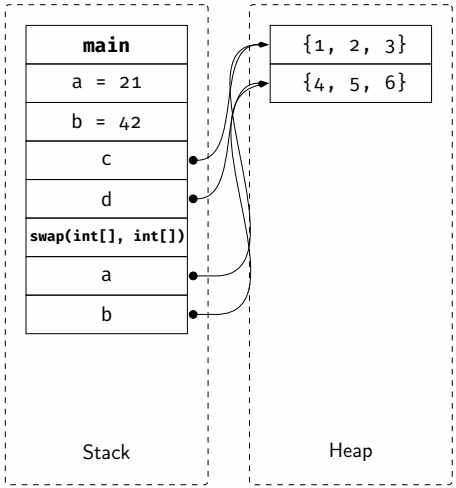


Der Swap-Stack-Heap-Durchlauf

```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[])
   }
}
```

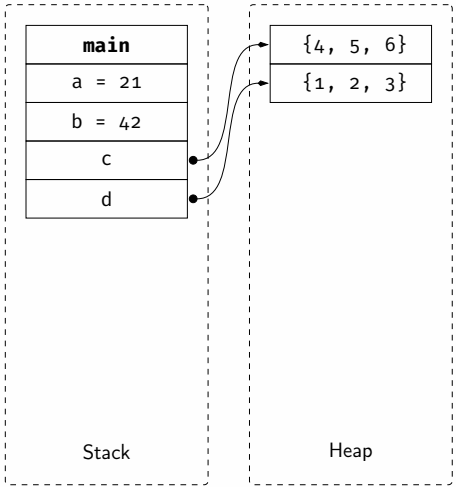


Der Swap-Stack-Heap-Durchlauf

```
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);
   }
}
```



Aufgabe 2: Mehrdimensionale Arrays

Im Folgenden sollen Methoden zum Verarbeiten von mehrdimensionalen Arrays (Matrizen) implementiert werden. Legen Sie dazu eine Java Datei `Matrix.java` *an. Im Folgenden sei* n *die Anzahl der Zeilen,* m *die Anzahl der Spalten,* i *und* j *die jeweiligen Indizes, sowie* a_{ij} *das Element in der* i*-ten Zeile und* j*-ten Spalte. Die Zeilensummennorm ist in der Mathematik die von der Summennorm abgeleitete natürliche Matrixnorm. Die Zeilensummennorm* $\|A\|_\infty$ *einer Matrix* A *entspricht der maximalen Betragssumme aller ihrer Zeilen:*

$$\|A\|_\infty = \max_{i=1,\dots,n} \sum_{j=1}^{n} |a_{ij}|$$

Beispiel: *Die Zeilensummennorm der Matrix* $A = \begin{pmatrix} 1 & -2 & -3 \\ 2 & 4 & -1 \end{pmatrix}$ *is gegeben durch* $\|A\|_\infty = \max\{|1| + |-2| + |-3|,\ |2| + |4| + |-1|\} = \max\{6, 7\} = 7.$

a) *Legen Sie ein 2-D Array mit* 3×3 *Elementen vom Typ double an und füllen Sie es mit gültigen Werten.*

b) *Implementieren Sie eine Methode* `public static double berechneBetragssumme(double[] arr)`*, die die Betragsnorm von* `arr`*, d.h. die Summe der Beträge der Elemente, berechnet und zurückgibt. Für ungültige Eingaben (*`arr` *ist* `null` *oder hat die Länge* 0*) soll* `-1.0` *zurückgegeben werden.*

c) *Implementieren Sie eine Methode* `public static double berechneZeilensummennorm(double[][] mat)`*, die ein 2-D Array als Argument annimmt, die Zeilensummennorm berechnet und diese zurückgibt. Für ungültige Eingaben (mat ist null oder hat die Länge* 0*) soll* `-1.0` *zurückgegeben werden.*

d) *Rufen Sie die Methode aus Teilaufgabe 3 mit ihrer Matrix aus und geben Sie das Ergebnis auf die Konsole aus.*

a) Ein Array Anlegen

- Hier mal etwas beliebiges

```
double[][] matrix = { { 1.5, -2, 1 }, { -5, 1, 6.1 }, { -2, -4, 1 } };
```

- Hier haben wir übrigens — Java sei dank — stark abgekürzt:

```
double[][] matrix = new double[][] { new double[] { ... }, ... };
```

- Quiz! Sind diese Statements gültig? Wenn ja, welchen Wert hat `m1`, `m2`, ...? (Warum?)

```
double[][] m1;
double[][] m2 = new double[][];
double[][] m3 = new double[2][];
int[][]    m4 = new int[1][3];
double[][] m5 = { null, {}, { null } };
```

✔ *m1 ist noch nicht initialisiert*

↯ *Java* muss *das Array initialisieren → braucht dessen Größe*

✔ *m3 ist { null, null } (Default-Wert für double[])*

✔ *m4 ist { { 0, 0, 0 } } (Default-Wert für int)*

↯ *null und {} (leer) sind valide double[], { null } aber nicht, da double nicht null sein kann.*

b) berechneBetragssumme(double[])

```java
public static double berechneBetragssumme(double[] arr) {
    if (arr == null || arr.length == 0) {
        System.out.println("Eingabe ungültig!");
        return -1.0;
    }
    double norm = 0;
    for (double element : arr) {
        norm = norm + Math.abs(element);
    }
    return norm;
}
```

c) berechneZeilensummennorm(double[][])

```
public static double berechneZeilensummennorm(double[][] mat) {
    if (mat == null || mat.length == 0) {
        System.out.println("Eingabe ungültig!");
        return -1.0;
    }
    double max = Double.MIN_VALUE;
    for (double[] array : mat) {
        double norm = berechneBetragssumme(array);
        if (norm == -1.0) return -1.0; // Fehler in Unteralgorithmus?
        else if (norm > max) max = norm;
    }
    return max;
}
```

Zu negativen Werten

- „Magic-Numbers“ sind generell schlecht
  - Wie bei den Zeichen „A“ vs. 65
  - Hier können Konstanten eine erste Abhilfe schaffen
- Betrachten wir die `-1.0`
  - Was ist, wenn `-1.0` auch ein gültiger Rückgabewert ist?
  - Wie können wir den Aufrufer auf das Problem hinweisen?
  - Was, wenn wir mit dem Fehler eine Nachricht mitgeben möchten?
  - Wie können wir Verbindungen zwischen Fehlern ausdrücken?
- Dafür werden wir später Exceptions betrachten

d) Ausgabe

- Ein super kompliziertes Aufrufchen (für zwei Punkte!):
  `System.out.println(berechneZeilensummennorm(matrix));`

- Der gesamte Code findet sich in der Datei Matrix.java

Aufgabe 3: Numerische Integration

In dieser Aufgabe sollen Sie ein einfaches Verfahren zur numerischen Integration implementieren. Legen Sie dazu eine Java Datei namens `NumerischeIntegration.java` *an und implementieren Sie alle folgenden Teilaufgaben innerhalb dieser Datei. Die Funktion* $f(x) = \exp(-x^2)$ *besitzt keine Stammfunktion bestehend nur aus elementaren Funktionen, was eine Berechnung schwierig macht. Man ist jedoch an guten Approximationen interessiert. Wir beschränken uns auf das Intervall* $[0, 1]$*, suchen also möglichst genaue Approximationen für das folgende Integral:* $A_f = \int_0^1 \exp(-x^2)dx$

Eine bekannte Methode ist die Approximation des Integrals durch eine Summe von Trapezflächen. Dazu zerlegt man das Intervall $[0, 1]$ *in* $n \geq 1$ *Segmente* $[x_i, x_{i+1}]$ *mit* $0 = x_1 < x_2 < \ldots < x_{n-1} < x_n = 1$.

Die Fläche auf dem Intervall $[x_i, x_{i+1}]$ *erhält man dann durch ein Trapez mit Eckpunkten* $a = (x_i, 0)$, $b = (x_{i+1}, 0)$, $c = (c_{i+x}, f(x_{i+1}))$ *und* $d = (x_i, f(x_i))$*. Die Fläche* A_i *eines derartigen Trapezes kann für* $x_{i+1} \geq x_i$ *durch* $A_i = (x_{i+1} - x_i)(f(x_i) + f(x_{i+1}))/2$ *berechnet werden. Die Approximation des Integrals ergibt sich dann durch Summierung der einzelnen Trapezflächen:* $A_f \approx A_f(n) = \sum_{i=1}^{n} A_i$.

a) Daten Einlesen und b) `berechneF(double)`

- Das Einlesen erfolgt hier leicht:
  ```
  int n = Integer.parseInt(args[0]);
  ```
- Ein Scanner ist hier theoretisch auch möglich
  - Den wollen wir dann aber auch immer schließen!
  - Uns reichen Kommandozeilenparameter
- Und nun noch berechnen ($f(x) = \exp(-x^2)$):
  ```
  public static double berechneF(double x) {
     return Math.exp(-(x*x));
  }
  ```
- Natürlich kann man hier auch `Math.pow(x, 2.0)` oder so benutzen

c) berechneTrapezFlaeche(double, double)

- Ein ungültiger Eckpunkt liegt vor, wenn $x_2 < x_1$:

```
public static double berechneTrapezFlaeche(double x1, double x2) {
   if(x2 < x1)
      return 0;
   else
      return (x2 - x1) * ((berechneF(x1) + berechneF(x2)) / 2.0);
}
```

- Auch hier handelt es sich um unspektakuläres übernehmen der Aufgabenstellung

d) `trapezVerfahren(int)`

- Wir können hier über die Schritte (i < n) oder über x terminieren
  Das Verhalten Umsetzung unterscheidet sich bei Rundungsfehlern!

```
public static double trapezVerfahren(int n) {
    if (n < 1) {
        System.err.println("Eingabe ungültig");
        return -1.0;
    }
    ...
}
```

d) trapezVerfahren(int)

```
public static double trapezVerfahren(int n) {
    ...
    double schrittweite = 1.0 / n;
    double x = 0.0;
    double A = 0.0;
    while (x <= 1.0 - schrittweite) {
        A = A + berechneTrapezFlaeche(x, x + schrittweite);
        x = x + schrittweite;
    }
    return A;
}
```

Ginge hier auch nur 1/n?

Nö, dann wäre das mit $n > 1$ immer 0 (da Ganzzahldivison).

Auch möglich wäre soetwas:

```
for(double x = 0; x <= 1 - schrittweite; x += schrittweite) {
    ...
}
```

e) Ausgabe

- Wieder eine fancy Ausgabe!
  ```
  System.out.println(trapezVerfahren(n));
  ```
- Der gesamte Code befindet sich hier: NumerischeIntegration.java
- Und wie wechseln wir die Funktion nun einfach aus?
  - Mit Eidl-Wissen eher weniger
  - Aber Lambdas sind hier hilfreich!

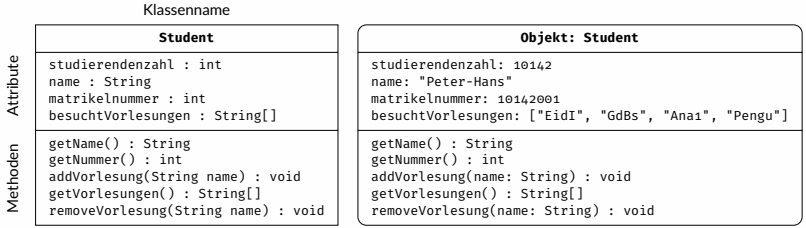
Aussicht: Übungsblatt 5

Aufgaben 1 und 2

- Hier helfen varargs!
  - Einfach nur Arrays aber in (un-)gruselig!
  - Und in Punktig! (Hier driftet Müde-Flo wohl ein bisschen ab…)
- Sonst haben wir ja schonmal was mit einer Quersumme gemacht (Übungsblatt 3, Aufgabe 3)
- Mehr Tipps kann ich dazu auch nicht geben…
- Und wegen Aufgabe 2…
  - Da haben wir heute ja schon was mit Zeichen gemacht!
  - Die Aufgabe lässt sich schön elegant lösen

Aufgabe 3: Objektorientierung — Klassenentwurf

- Wir möchten einen *Klassenentwurf* (Und Objektlies)
- Gerne von Hand (dann aber sehr deutlich):
  - Sonst gibt es draw.io
  - Man kann das aber auch mit LaTeX machen
  - Oder mit anderen Zeichenprogrammen…
  - Generierte Klassenentwürfe sind aber verboten!



There aren't that many people in the world who are good programmers and there are not that many people in the world who are good writers, and here we are expecting them to be both.
— Donald E. Knuth [Knu99, p.649]

Abschließendes

Zusammenfassend

[Chr34] Agatha Christie. *Murder on the Orient Express*. 1934

[Knu99] Donald Ervin Knuth. *Digital Typography*. 1999

- In manchen Fällen *muss* Java initialisieren (Arrays!)
  - Ganzzahlen werden `0`, Fließkommazahlen `0.0` (im jeweiligen Typ, also `0.0f`, …)
  - `boolean` wird `false`, `char` wird `'\0'` (Unicode-„Null“)
  - Referenzdatentypen (wie Arrays) werden **`null`**
- Java unterscheidet Methoden durch ihre Signatur (Name & Parametertypliste)
- Wir haben einen Blick auf Clone(s) geworfen (I've seen the dark side now)
- Auch wenn Heap und Stack nicht Klausurrelevant sind, helfen sie bei Vielem!
  - Call-by-Value und Call-by-Reference (Seiteneffekte, …)
  - Was genau schützt **`final`**, …