Java Intro

Michael Inden

Freiberuflicher Consultant und Trainer

https://github.com/Michaeli71/JAVA_INTRO

Speaker Intro

- Michael Inden, Jahrgang 1971
- Diplom-Informatiker, C.v.O. Uni Oldenburg

- ~8 ¼ Jahre **SSE** bei Heidelberger Druckmaschinen AG in Kiel
- ~6 ¾ Jahre **TPL, SA** bei IVU Traffic Technologies AG in Aachen
- ~4 ¼ Jahre **LSA / Trainer** bei Zühlke Engineering AG in Zürich
- ~3 Jahre **TL / CTO** bei Direct Mail Informatics / ASMIQ in Zürich
- **Freiberuflicher Consultant, Trainer und Konferenz-Speaker**
- Autor und Gutachter beim dpunkt.verlag

E-Mail: michael.inden@hotmail.ch
Blog: https://jaxenter.de/author/minden

Kurse: Bitte sprecht mich an!

https://github.com/Michaeli71/JAVA_INTRO

Agenda

Workshop Contents

- **PART 1: Schnelleinstieg Java**
  - Erste Schritte in der JShell
  - Schnelleinstieg
    - Variablen
    - Operatoren
    - Fallunterscheidungen
    - Schleifen
    - Methoden
    - Rekursion

https://github.com/Michaeli71/JAVA_INTRO

Workshop Contents

- PART 2: Strings
  - Gebräuchliche String-Aktionen
  - Suchen und Ersetzen
  - Formatierte Ausgaben
  - Einstieg Reguläre Ausdrücke
  - Mehrzeilige Strings

- **PART 3: Arrays**
  - Gebräuchliche Array-Aktionen
  - Mehrdimensionale Arrays
  - Beispiel: Flood Fill

Workshop Contents

- **PART 4: Klassen & Objektorientierung**
  - Basics
  - Textuelle Ausgaben
  - Gleichheit == / `equals()`
  - Klassen ausführbar machen
  - Imports & Packages
  - Information Hiding
  - Vererbung und Overloading und Overriding
  - Die Basisklasse `Object`
  - Interfaces & Implementierungen

PART 3:
Arrays

Arrays

- Arrays bilden die Grundlage für viele andere Datenstrukturen.
- Arrays werden dazu verwendet, mehrere Dinge gleichen Typs zu speichern, etwa eine Menge von Namen oder Personen.
- Etwas unpraktisch wäre es, müsste man jeweils eine eigene Variable für jedes Element definieren.

```
jshell> var stadt1 = "Bremen"
stadt1 ==> "Bremen"

jshell> var stadt2 = "Hamburg"
stadt2 ==> "Hamburg"

jshell> var stadt3 = "Kiel"
stadt3 ==> "Kiel"

jshell> String[] städte = {"Bremen", "Hamburg", "Kiel"}
städte ==> String[3] { "Bremen", "Hamburg", "Kiel" }
```

Wäre var hier nicht praktischer?

Arrays

- **Ja, aber**

```
jshell> var städte = {"Bremen", "Hamburg", "Kiel"}
|  Error:
|  cannot infer type for local variable städte
|    (array initializer needs an explicit target-type)
|  var städte = {"Bremen", "Hamburg", "Kiel"};
|  ^-----------------------------------^
```

- **var nur möglich, wenn Typ eindeutig aus rechter Seite hervorgeht!**

Arrays -- Erzeugen

- **Definition wie bei normalen Variablen, Typ und dann Name, aber mit eckigen Klammern:**

```
jshell> String[] firstNames = { "Tim", "Tom", "Mike", "Peter"}
firstNames ==> String[4] { "Tim", "Tom", "Mike", "Peter" }

jshell> String firstNames[] = { "Tim", "Tom", "Mike", "Peter"}
firstNames ==> String[4] { "Tim", "Tom", "Mike", "Peter" }
```

- **Für Zahlen ebenfalls möglich:**

```
jshell> int[] primes = { 2, 3, 5, 7, 11, 13}
primes ==> int[6] { 2, 3, 5, 7, 11, 13 }
```

Arrays – Zugriff auf Elemente

- **Lesender und schreibende Zugriff über Indexangabe in eckigen Klammern im Bereich von 0 bis Anzahl Elemente – 1:**

```
jshell> String[] firstNames = { "Tim", "Tom", "Mike", "Peter"}
firstNames ==> String[4] { "Tim", "Tom", "Mike", "Peter" }

jshell> firstNames[2]
$10 ==> "Mike"

jshell> firstNames[2] = "Michael"
$11 ==> "Michael"

jshell> firstNames
firstNames ==> String[4] { "Tim", "Tom", "Michael", "Peter" }
```

Arrays – Zugriff auf Elemente

- **Zugriff außerhalb des erlaubten Bereichs**

```
jshell> firstNames[17]
|  Exception java.lang.ArrayIndexOutOfBoundsException: Index 17 out of bounds
for length 4
|        at (#13:1)

jshell> firstNames[-7]
|  Exception java.lang.ArrayIndexOutOfBoundsException: Index -7 out of bounds
for length 4
|        at (#14:1)
```

- **Länge / obere Grenze ermitteln**

```
jshell> firstNames.length
$15 ==> 4
```

Arrays – Durchlaufen der Elemente

- **Indexbasierte for-Schleife**

```
jshell> for (int i = 0; i < firstNames.length; i++) {
   ...>     System.out.println(firstNames[i]);
   ...> }
Tim
Tom
Michael
Peter
```

- **Wertebasierte for-Schleife:**

```
jshell> for (String currentName : firstNames) {
   ...>     System.out.println(currentName);
   ...> }
Tim
Tom
Michael
Peter
```

Arrays – Konsolenausgabe

- **Built-in**

```
jshell> firstNames
firstNames ==> String[4] { "Tim", "Tom", "Michael", "Peter" }

jshell> System.out.println(firstNames)
[Ljava.lang.String;@3f8f9dd6

jshell> System.out.println(Arrays.toString(firstNames))
[Tim, Tom, Michael, Peter]
```

Arrays – Konsolenausgabe

- **Selbstdefinierte Ausgabe**

```
jshell> void printArray(String[] values) {
   ...>     for (String val : values)
   ...>         System.out.println(val);
   ...> }
|  created method printArray(String[])
```

- **=>**

```
jshell> printArray(firstNames)
Tim
Tom
Michael
Peter
```

Was mache ich, wenn ich die Anzahl der Elemente vorab nicht kenne oder dynamisch Arrays fixer Größe erstellen möchte?

Arrays – Dynamisches Erzeugen

- **Dynamische Erzeugung fixer Länge ohne Werte:**

```
jshell> int[] top10 = new int[10]
top10 ==> int[10] { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }

jshell> String[] top5 = new String[5]
top5 ==> String[5] { null, null, null, null, null }
```

- **Nachträgliches Befüllen**

```
jshell> for (int i = 0; i < top10.length; i++)
   ...>     top10[i] = i * 2

jshell> top10
top10 ==> int[10] { 0, 2, 4, 6, 8, 10, 12, 14, 16, 18 }
```

Arrays – Dynamisches Erzeugen

- **Dynamische Erzeugung variabler Länge**

```
jshell> int[] createEmptyArray(int size)
   ...> {
   ...>     return new int[size];
   ...> }
|  created method createEmptyArray(int)

jshell> createEmptyArray(10)
$48 ==> int[10] { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }

jshell> createEmptyArray(20)
$49 ==> int[20] { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
```

java.util.Arrays

Arrays – Sortieren

- **Sortieren von Zahlen:**

```
jshell> int[] numbers = { 2, 5, 4, 3, 1, 7, 6};
numbers ==> int[7] { 2, 5, 4, 3, 1, 7, 6 }

jshell> Arrays.sort(numbers)

jshell> numbers
numbers ==> int[7] { 1, 2, 3, 4, 5, 6, 7 }
```

- **Sortieren von Strings:**

```
jshell> String[] names = { "Anne", "Sophie", "Barbara", "Lilija"}
names ==> String[4] { "Anne", "Sophie", "Barbara", "Lilija" }

jshell> Arrays.sort(names)

jshell> names
names ==> String[4] { "Anne", "Barbara", "Lilija", "Sophie" }
```

Arrays – Teile kopieren

- **Kopieren von Teilbereichen**

```
jshell> int[] numbers = { 1, 2, 3, 4, 5, 6, 7 }
numbers ==> int[7] { 1, 2, 3, 4, 5, 6, 7 }

jshell> int[] sliced = Arrays.copyOfRange(numbers, 2, 5);
sliced ==> int[4] { 3, 4, 5 }
```

- **Kopieren des gesamten Arrays mit Vergrösserung oder Verkleinerung:**

```
jshell> int[] enlarged = Arrays.copyOf(numbers, 10);
enlarged ==> int[10] { 1, 2, 3, 4, 5, 6, 7, 0, 0, 0 }

jshell> int[] shortened = Arrays.copyOf(numbers, 4);
shortened ==> int[4] { 1, 2, 3, 4 }
```

Sonstiges – Die Klasse Arrays

- **Erweiterungen in `java.util.Arrays`:**
  - `equals()` – Vergleicht Arrays auf Gleichheit (bezogen auf Bereiche)
  - `compare()` – Vergleicht Arrays (auch bezogen auf Bereiche)
  - `mismatch()` – Ermittelt die erste Differenz in Array (auch bezogen auf Bereiche)

Sonstiges – Die Klasse Arrays

- **`Arrays.equals()` schon lange im JDK, aber …**

- **man konnte den Vergleich leider nicht auf spezielle Bereiche einschränken**

```
final byte[] string1 = "ABC-DEF-GHI".getBytes();
final byte[] string2 = "DEF".getBytes();

System.out.println(Arrays.equals(string1, string2));      => false

System.out.println(Arrays.equals(string1, 4, 7,
                                 string2, 0, 3));         => true
```

Sonstiges – Die Klasse Arrays

- **`Arrays.compare()` neu im JDK**

- **Vergleicht – kann man auch auf spezielle Bereiche einschränken**

```
final byte[] string1 = "ABC-DEF-GHI".getBytes();
final byte[] string2 = "DEF".getBytes();

System.out.println(Arrays.compare(string1, string2));     => ABC < DEF => -3

System.out.println(Arrays.compare(string1, 4, 7,
                                  string2, 0, 3));        => DEF = DEF => 0

System.out.println(Arrays.compare(string1, 8, 11,
                                  string2, 0, 3));        => GHI > DEF => 3
```

Sonstiges – Die Klasse Arrays

- **`Arrays.mismatch()` neu im JDK**

- **Prüft auf Abweichungen im Array – kann man auch auf spezielle Bereiche einschränken**

```
//                          0123401234567890123 4
final byte[] string1 = "ABC-DEF-GHIJKLMNOPQT".getBytes();
final byte[] string2 = "ABC-dEF-GHIJKLMNOPqT".getBytes();

System.out.println(Arrays.mismatch(string1, string2));    => 4

System.out.println(Arrays.mismatch(string1, 5, 20,
                                   string2, 5, 20));      => 13
```

Mehrdimensionale Arrays

Arrays – Mehrdimensional

- **Ein mehrdimensionales Array ist ein Array, das mehrere Arrays enthält.**

- **Die Anzahl an Dimensionen wird durch die Anzahl an eckigen Klammerpaaren bestimmt:**
  - int[][] für zweidimensional oder
  - int[][][] für dreidimensional.

- **Die einzelnen Werte kann man dann wie gewohnt in geschweiften Klammern notieren. Ein zweidimensionales Array erstellen wir wie folgt:**

```
jshell> int[][] twodim = {{1, 1, 1, 1},
   ...>                   {2, 2, 2, 2}}
twodim ==> int[3][] { int[4] { 1, 1, 1, 1 }, int[4] { 2, 2, 2, 2 } }
```

Arrays – Mehrdimensional

- Die einzelnen Werte kann man dann wie gewohnt in geschweiften Klammern notieren. Ein zweidimensionales Array erstellen wir wie folgt:

```
jshell> int[][] twodim = {{1, 1, 1, 1},
   ...>                   {2, 2, 2, 2},
   ...>                   {3, 3, 3, 3}}
twodim ==> int[3][] { int[4] { 1, 1, 1, 1 }, int[4] { 2, 2, 2, 2 }, int[4] {
3, 3, 3, 3 } }
```

- Wir sehen zunächst, dass das Array wiederum Arrays (hier gleicher Länge) enthält. Oftmals hat man es in der Praxis mit solchen rechteckigen Ausrichtungen zu tun.

Arrays – Mehrdimensional

- Dann kann man sich ein zweidimensionales Array ein wenig wie eine Schrankwand mit nummerierten Schubladen für Elemente vorstellen. Nachfolgend ist das für ein 5 × 3-Array (5 Positionen in x-Richtung und 3 Reihen in y-Richtung) schematisch dargestellt.

	0	1	2	3	4
0	0,0	0,1	0,2	0,3	0,4
1	1,0	1,1	1,2	1,3	1,4
2	2,0	2,1	2,2	2,3	2,4

Arrays – Mehrdimensional

- **Spezialfall: Nicht-rechteckiges Array**

```
jshell> int[][] twodimTriangle = {{1},
   ...>                           {1, 2},
   ...>                           {1, 2, 3},
   ...>                           {1, 2, 3, 4}}
twodimTriangle ==> int[4][] { int[1] { 1 }, int[2] { 1, 2 }, int[3]  ... },
int[4] { 1, 2, 3, 4 } }
```

Arrays – Mehrdimensional

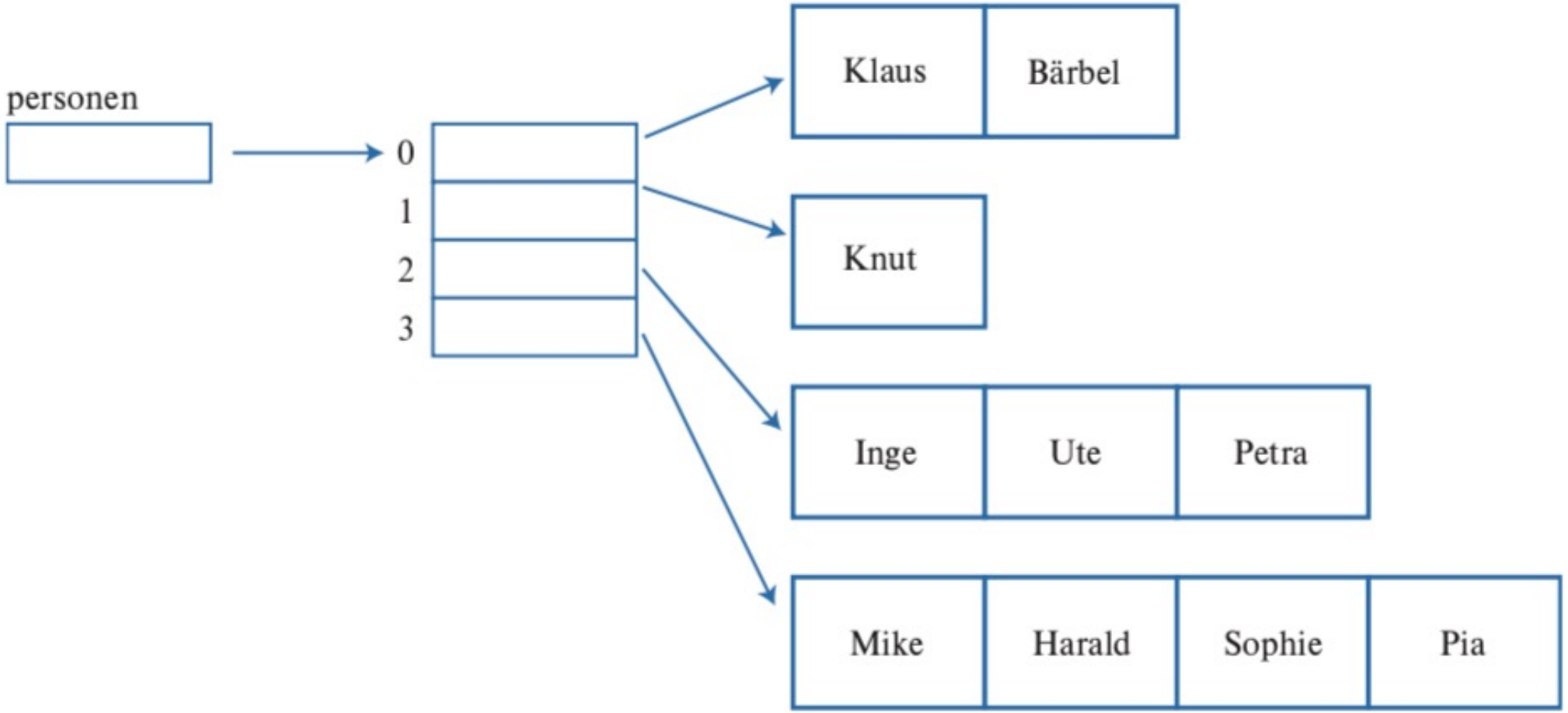
- **Ausgabe eines nicht-rechteckigen Arrays**

```
jshell> void print2dArray(int[][] values)
   ...> {
   ...>     for (int y = 0; y < values.length; y++)
   ...>     {
   ...>         for (int x = 0; x < values[y].length; x++)
   ...>         {
   ...>             System.out.print(values[y][x]);
   ...>         }
   ...>         System.out.println();
   ...>     }
   ...> }
|  modified method print2dArray(int[][])

jshell> print2dArray(twodimTriangle)
1
12
123
1234
```

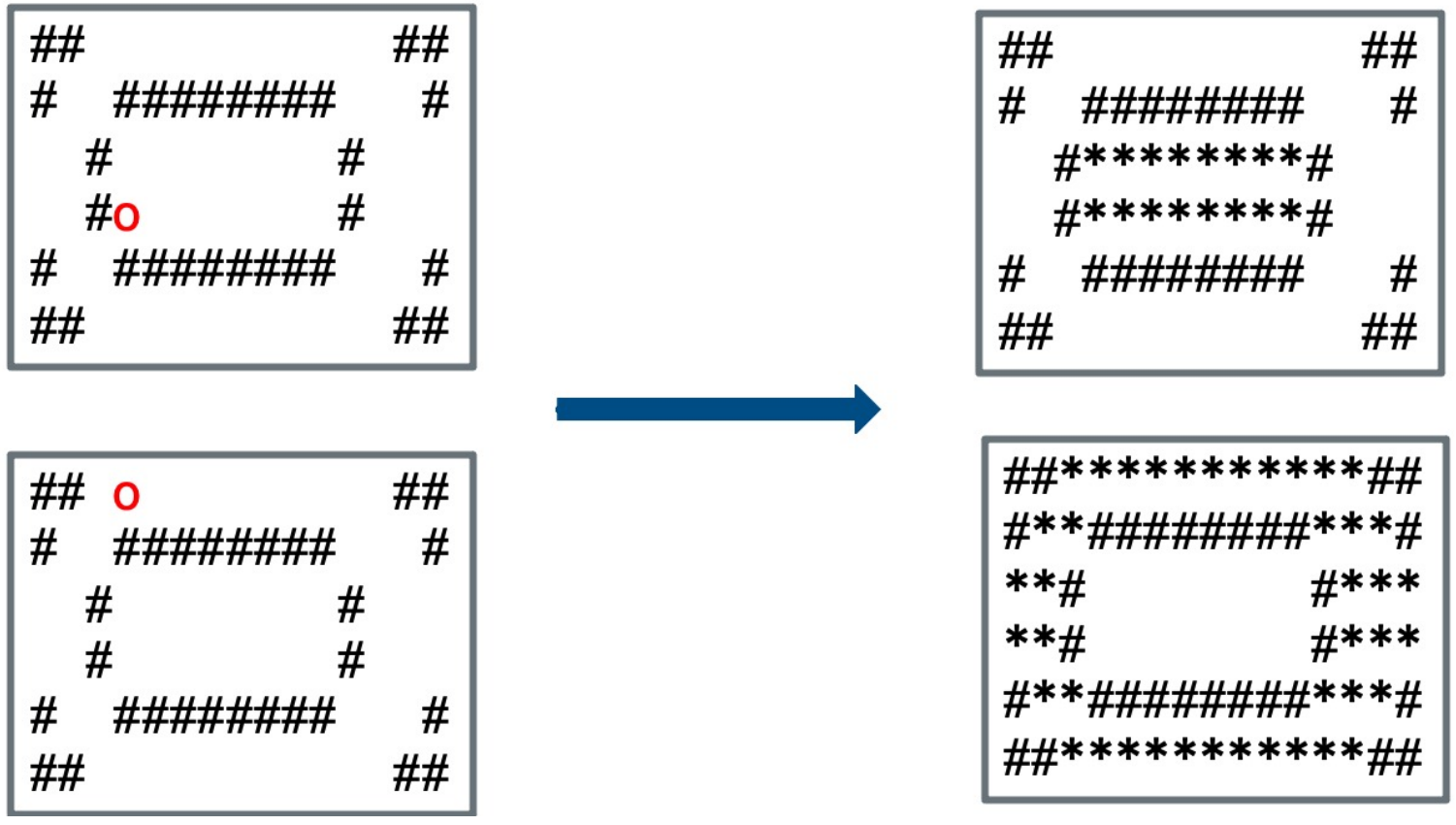
Arrays – Mehrdimensional

- **Beispiel eines nicht-rechteckigen Arrays**



Arrays – Flood Fill

Das Füllen beginnt an einer vorgegebenen Position, etwa in der linken oberen Ecke, und wird dann so lange in alle vier Himmelsrichtungen fortgesetzt, bis die Grenzen der Listen oder eine Begrenzung in Form eines anderen Zeichens gefunden wird:



Was ist denn mit einem Muster?

Arrays – Flood Fill

Erweitern wir also die Implementierung so, dass nun eine Fläche auch mit Muster gefüllt werden kann!

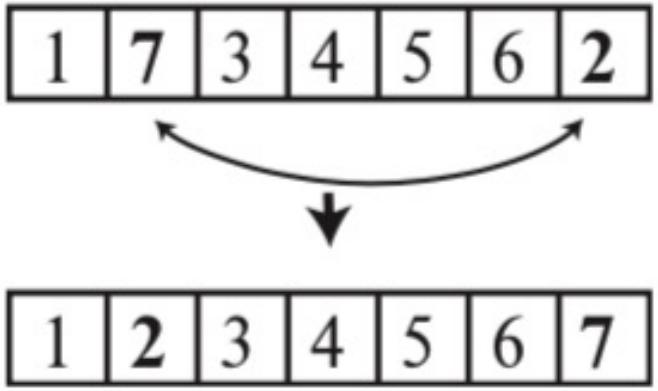
```
## o                    #
#  ##################  ##
  #         ##         ###
  #         ##          ##
#  ################     #
##                      ##
###                    ###
##                      ##
#                       #
```

```
##..|..|..|..|..|..|..|..#
#*-##################--*-##
.|#         ##|..|..|..|###
-*#         ##*--*--*--*-##
#|.################|..|..#
##--*--*--*--*--*--*--*-##
###.|..|..|..|..|..|..|###
##--*--*--*--*--*--*--*-##
#|..|..|..|..|..|..|..|..#
```

Standardfunktionalitäten

Arrays – Standardfunktionalitäten

- **Tauschen von Elementen**




```
public static void swap(int[] values, int first, int second)
{
    final int value1 = values[first];
    final int value2 = values[second];
    values[first] = value2;
    values[second] = value1;
}
```

Arrays – Standardfunktionalitäten

- **Umdrehen von Teilbereichen**

```
public static void reverse(int[] values, int start, int end)
{
    while (start < end)
    {
        swap(values, start, end);
        start++;
        end--;
    }
}
```

```
jshell> reverse(numbers, 2, 6)

jshell> numbers
numbers ==> int[9] { 1, 2, 7, 6, 5, 4, 3, 8, 9 }

jshell> reverse(numbers, 2, 6)

jshell> numbers
numbers ==> int[9] { 1, 2, 3, 4, 5, 6, 7, 8, 9 }
```

Arrays – Rotieren

- **2D Array um 90° rotieren**

```
1111          4321          4444
2222   =>     4321   =>     3333
3333          4321          2222
4444          4321          1111
```

Arrays – Rotieren

- **2D Array um 90° rotieren**

```
          x   0123
        y     ----
        0     ABCD
        1     EFGH
```

```
  xn   01                  xn   01
yn     --               yn     --
 0     DH                0     EA
 1     CG                1     FB
 2     BF                2     GC
 3     AE                3     HD
```

```
                    Orig    ->    NewX      NewY
-------------------------------------------------
rotateLeft:         (x,y)   ->    y         maxX-x
rotateRight:        (x,y)   ->    maxY-y    x
```

Arrays – Rotieren

```
public static int[][] rotate(int[][] values, boolean rotateLeft)
{
    int maxX = values[0].length - 1;
    int maxY = values.length - 1;
    final int[][] rotatedArray = new int[maxX + 1][maxY + 1];
    for (int y = 0; y < maxY + 1; y++)
    {
        for (int x = 0; x < maxX + 1; x++)
        {
            int origValue = values[y][x];
            if (rotateLeft)
            {
                rotatedArray[maxX - x][y] = origValue;
            }
            else
            {
                rotatedArray[x][maxY - y] = origValue;
            }
        }
    }
    return rotatedArray;
}
```

Arrays – Rotieren

```
int[][] values = { { 1, 1, 1, 1 },
                   { 2, 2, 2, 2 },
                   { 3, 3, 3, 3 },
                   { 4, 4, 4, 4 } };

print2dArray(values);

int[][] rotated1 = rotate(values, false);
print2dArray(rotated1);

int[][] rotated2 = rotate(rotated1, false);
print2dArray(rotated2);
```

```
1111
2222
3333
4444

4321
4321
4321
4321

4444
3333
2222
1111
```

Exercises Part 3

https://github.com/Michaeli71/JAVA_INTRO

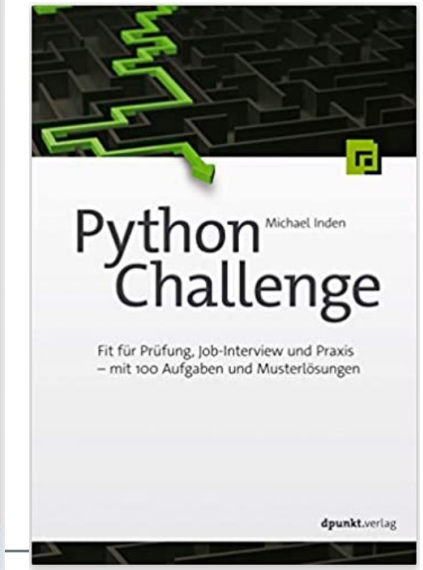
Thank You


HELP
IS ON THE
WAY


Michael Inden
Einfach Java
Gleich richtig programmieren lernen
dpunkt.verlag


5. Auflage
Michael Inden
Der Weg zum Java-Profi
Konzepte und Techniken für die professionelle Java-Entwicklung
dpunkt.verlag


Michael Inden
Java Challenge
Fit für das Job-Interview und die Praxis – mit mehr als 100 Aufgaben und Musterlösungen
dpunkt.verlag


Michael Inden
Python Challenge
Fit für Prüfung, Job-Interview und Praxis – mit 100 Aufgaben und Musterlösungen
dpunkt.verlag