Verwendung von Feldern (🇺🇸 *Arrays*) in Java

Dozent: Prof. Dr. Michael Eichberg
Kontakt: michael.eichberg@dhbw.de, Raum 149B

Version: 1.3.1

Folien: https://delors.github.io/prog-java-arrays/folien.de.rst.html
https://delors.github.io/prog-java-arrays/folien.de.rst.html.pdf
Kontrollfragen: https://delors.github.io/prog-java-arrays/kontrollfragen.de.rst.html
Klausurvorbereitung: https://delors.github.io/prog-java-arrays/klausurvorbereitung.de.rst.html
Fehler melden: https://github.com/Delors/delors.github.io/issues

1. Eindimensionale Felder (🇺🇸 *Arrays*)

Deklaration von Feldern (*Arrays*)

Eindimensionale Felder sind Datentypen, die es ermöglichen eine Liste mit einer fixen Anzahl von Werten gleichen Datentyps zu verwalten.

- Die Tage der verschiedenen Monate können als ein Feld mit der Größe 12 abgelegt werden

> **Hinweis**
>
> Beim Programmieren beginnt der Index eines Feldes immer bei 0.

Monat (Index):	0	1	2	3	4	5	6	7	8	9	10	11
Tage:	31	29	31	30	31	30	31	31	30	31	30	31

- Variablen mit einem Feld-Datentyp werden durch den Datentyp der einzelnen Elemente gefolgt von eckigen Klammern deklariert.

Syntax: `<Typ>[] <Bezeichner>` oder `<Typ> <Bezeichner>[]`

Deklaration eines Feldes

```
1 int[] daysPerMonth;
```

Alternativ möglich, aber unüblich geworden:

```
1 int daysPerMonth[];
```

Initialisierung eines leeren Arrays:

```
1 daysPerMonth = new int[12];
2 // daysPerMonth ⟹ int[12] { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
```

Syntax: <Bezeichner> = new <Typ>[<Größe>]

Initialisierung eines Arrays mit konkreten Werten:

```
1 daysPerMonth =
2     new int[]{31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

bzw. ohne Verwendung von `new`:

```
1 int [] daysPerMonth =
2     {31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
3 // ⚠ Diese Art der Initialisierung eines Arrays direkt über
4 // "= {...}"  kann nur bei der Deklaration erfolgen.
```

Syntax: <Bezeichner> = {<Ausdruck> (, <Ausdruck>)* }

- Nach der Initialisierung lässt sich die Größe eines Feldes **nicht** mehr ändern.

  Die Länge eines Arrays kann mittels `length` abgefragt werden:

```
1 int numberOfMonths = daysPerMonth.length;
2 // numberOfMonths ⟹ 12
```

- Wird ein Feld nur deklariert und nicht initialisiert, dann hat die Variable den speziellen Wert `null`.

Zugriff auf die Elemente eines Feldes

Auf einzelne Elemente eines Feldes kann mittels eines Indexes und dem Feldzugriff-Operator `[]` lesend oder schreibend zugegriffen werden, z.B. mit `a[1]`.

- Wertzuweisung eines Feldelementes: Verwendung des Feldzugriffoperators auf der linken Seite einer Zuweisung, z.B. `a[1] = 1;`
- Auslesen eines Feldelementes: „jegliche andere Verwendung des Feldzugriffoperators“.
- Verwendung eines ungültigen Indexes führt zu einer Ausnahme/einem Laufzeitfehler (`ArrayIndexOutOfBoundsException`).

```
1 daysPerMonth[13]
2 ⟹ Exception java.lang.ArrayIndexOutOfBoundsException:
3     Index 13 out of bounds for length 12
```

Beispiel: Lesender Zugriff auf ein Element eines Feldes

```
1 int daysInFebruary = daysPerMonth[1]; // Index "1" ⇒ 2. Element
2 // daysInFebruary ⟹ 29
```

Syntax: <Bezeichner>[<Index>]

Beispiel: Schreibende und lesende Zugriffe

```
1 int daysPerMonth[] = new int[12]; // Deklaration
2
3 daysPerMonth[0] = 31;
4 daysPerMonth[1] = 29;
5 //…
6 daysPerMonth[10] = 30;
7 daysPerMonth[11] = 31;
8
9 IO.println("daysPerMonth[1] = " + daysPerMonth[1]);
```

Häufig greift man auf Arrays mittels einer Schleife zu:

```
1 for (int i = 0; i < daysPerMonth.length; i++) {
2     IO.println("daysPerMonth[" + i + "] = " + daysPerMonth[i]);
3 }
```

Bzw. mit einer `for-each`-Schleife, wenn der Index nicht benötigt wird:

```
1 for (int days : daysPerMonth) {
2     IO.println("days = " + days);
3 }
```

※ Hinweis

Der Index der einzelnen Elemente eines Feldes läuft von 0 bis Größe-1, wobei das erste Element den Index 0 hat.

In Java und vielen anderen moderne(re)n Programmiersprachen ist es nicht möglich auf ein Element eines Feldes zuzugreifen, das außerhalb des definierten Bereichs

liegt, da früh erkannt wurde, dass dies insbesondere in älteren Programmiersprachen (z. B. C) ein häufiger Fehler ist, der dann zu Speicherlecks führt. Dies hat dazu geführt, dass das „Weisse Hause“ die Empfehlung ausgesprochen hat, solche alten Sprachen nicht mehr zu verwenden.

Übung

1.1. Wochentagsberechnung mit Feld

Nehmen Sie Ihr Programm zur Berechnung des Wochentags und ersetzen Sie die Logik zur Bestimmung des Namens eines Wochentags durch ein Feld mit den Namen der Wochentage:

```
1 String[] dayInWeekName = ...
```

2. Referenzen auf Felder (🇺🇸 *Arrays*)

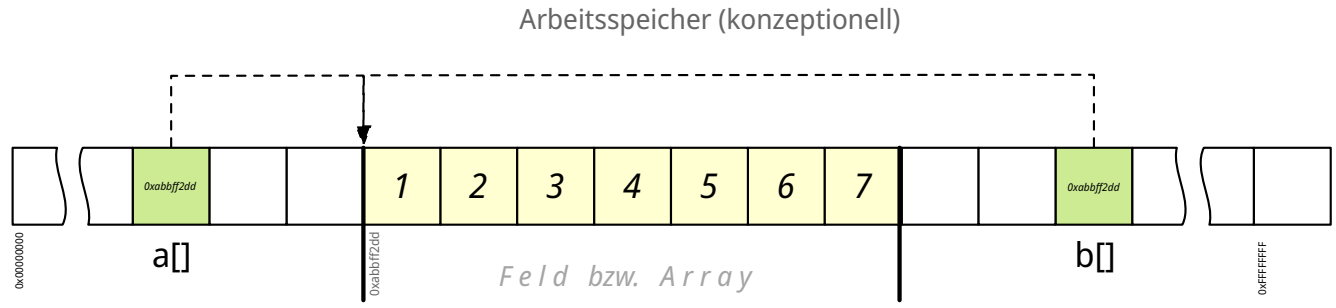
Felder sind Referenzdatentypen

Eine Variable mit einem Feld-Datentyp speichert eine (virtuelle) Speicheradresse zu den Feld-Inhalten (Werten).

```
1 int[] a = {1, 2, 3};
2 int[] b = a; // "b" referenziert das gleiche Feld wie "a".
3              // "b" ist nur eine Kopie des Zeigers auf das Feld.
4              // "b" ist keine Kopie des Feldes "a".
5 b[0] = 4;
6 IO.println(a[0]);
7 // ⟹ 4
```

Visualisierung von Referenzen auf Felder

```java
int[] a = { 1, 2, 3, 4, 5, 6, 7 };
int[] b = a;
```



※ Hinweis

Die Werte eines Referenzdatentyp werden automatisch gelöscht (*Garbage Collected*), wenn keine Referenz (Variable) auf die Inhalte mehr existiert.

Referenzdatentypen und `final`

- Der `final`-Modifizierer verhindert *nur*, dass die Referenz auf ein Feld geändert werden kann.
- Der Inhalt des Feldes kann jedoch geändert werden.

  D. h. der Nutzen von `final` ist im Zusammenhang mit Referenzdatentypen im Allgemeinen begrenzt.

```
1 void main() {
2     final int[] a = {1, 2, 3};
3     IO.println(a[0] = -1);
4     IO.println(Arrays.toString(a));
5     // a = new int[]{}; // IllegalAccess.java:
6                         // error: cannot assign a value to final variable a
7 }
```

Vergleich von Feldern (*Arrays*)

Der Vergleich zweier Feldvariablen mit dem `==` (~ `==`) bzw. `!=` (~ `!=`) Operator vergleicht nicht den Inhalt der Felder, sondern die virtuelle Speicheradresse (ähnlich bei Strings).

Der Vergleich der Inhalte muss über den Vergleich der einzelnen Feldelemente erfolgen bzw. über Hilfsmethoden wie z.B. `Arrays.equals(...)`.

```
1 int[] a = {1, 2, 3};
2 int[] b = {1, 2, 3};
3 int[] c = a;
4
5 IO.println(a == b); // ⟹ false
6 IO.println(a == c); // ⟹ true
7 IO.println(Arrays.equals(a, b)); // ⟹ true
```

Konzeptionell führt Arrays.equals(...) eine Schleife über die Elemente der beiden Arrays aus und vergleicht die Werte der Elemente. Der Vergleich der Referenzen erfolgt über den Operator `==`.

Felder kopieren

Eine Kopie der Inhalte muss über das Erzeugen eines neuen Feldes und Kopie der einzelnen Feldelemente erfolgen bzw. über Hilfsmethoden wie z. B. `System.arraycopy(...)` oder `Arrays.copyOf(...)` oder `<Array>.clone()`.

Beispiel mit `<Array>.clone()`:

```
jshell> final var clone = daysPerMonth.clone();
// clone ==> int[12] { 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 }
jshell> clone[0] = 35;
jshell> daysPerMonth[0]
// ==> 31
```

Warnung

Die Methoden, z. B. `clone` und `arraycopy`, erzeugen nur flache Kopien ( *shallow copies*).

Dokumentation: java.util.Arrays (Java 23)

Felder als Methodenparameter bzw. Rückgabewert

Die Parameter und der Rückgabewert einer Methode können vom Typ eines Feldes sein.

- Bei der Übergabe eines Feldes an eine Methode bzw. der Rückgabe eines Feldes wird eine Kopie der Referenz auf das Feld erzeugt. Es wird keine Kopie der Arrays als solches erzeugt.
- Änderungen an den Feldelementen innerhalb der Methode wirken sich auf das ursprüngliche Feld aus.
- Der Rückgabewert kann direkt zur Initialisierung eines Feldes verwendet werden.

```
1  void incrementAll(int[] a) {
2      for (int i = 0; i < a.length; i++) { a[i]++; }
3  }
4
5  int[] getLengths(String[] strings) {
6      int[] lengths = new int[strings.length];
7      for (int i = 0; i < strings.length; i++) {
8          lengths[i] = strings[i].length();
9      }
10     return lengths;
11 }
12 getLengths(new String[]{"a","ab","abc"})
13 // ⟹ int[3] { 1, 2, 3 }
```

Übung

2.1. Arrays vergleichen

Schreiben Sie eine Methode, die prüft ob ein Array von `int` Werten, mit dem Beginn eines anderen Arrays von `int`-Werten übereinstimmt. Vergleichen Sie die Elemente der beiden Arrays mit Hilfe des `==` bzw. `!=` Operators.

D. h. die Methode soll `true` zurückgeben, wenn *alle Elemente* des ersten Arrays (`a`) mit den ersten Elementen des zweiten Arrays `b` übereinstimmen. Das Array `b` kann mehr Elemente enthalten als das Array `a` und diese werden ignoriert.

Die Methode soll die folgende Signatur haben und auch alle Sonderfälle abdecken!

```
1 boolean startsWith(int[] a, int[] b);
```

Übung

2.2. Skalarprodukt

Schreiben Sie eine Methode, die zwei gleich lange Arrays von `int` Werten entgegennimmt und das Skalarprodukt der beiden Arrays berechnet.

Bemerkung

Das Skalarprodukt ist die Summe der Produkte der Elemente an der gleichen Position in den beiden Arrays: `a[0] * b[0] + a[1] * b[1] + ....`

Übung

2.3. Kommandozeilenparameter

Die `main` Methode eines Java Programms bekommt - wenn der erste Parameter entsprechend spezifiziert ist - ein Feld von `String`s übergeben. Dieser Parameter wird üblicherweise `args` genannt (`void main(String[] args)`).

Nehmen Sie Ihr Programm zur Berechnung des BMIs und verwenden Sie Kommandozeilenargumente als Parameter für Ihre `bmi` Funktion.

- Prüfen Sie ob die Anzahl der Parameter korrekt ist und geben Sie eine Fehlermeldung aus, wenn dies nicht der Fall ist.
- Faktorisieren Sie (ggf.) die Funktionalität zur Berechnung des BMI in zwei Methoden:

  Eine Methode, die Strings entgegennimmt und eine die `double` Werte entgegennimmt.

Beispiel

~~Bisher:~~

```
$ java BMIBerechnen.java
Bitte geben Sie Ihr Gewicht in Kilogramm und Ihre Größe in Mete an.
```

Neu:

```
$ java BMI.java 83.1 1.89
Ihr BMI beträgt: 23.26362643822961 - Normalgewicht
```

D. h. der Nutzer übergibt direkt die Werte für das Gewicht und die Größe und wird nicht aufgefordert diese Werte einzugeben.

3. Mehrdimensionale Felder

Multidimensional Arrays

Mehrdimensionale Felder sind Datentypen, die es ermöglichen ein Feld von Feldern (gleichen Datentyps) zu verwalten.

Beispiel: Matrix für Umsätze pro Jahr (1. Dim.) und Monat (2. Dim.) bei 10 Jahren.

```
int[][] sales = new int[10][12];
```

Jahr		Jan	Feb	Mär	Apr	Mai	Jun	Jul	Aug	Sep	Okt	Nov	Dez	
sales[0] =	[	1000€	2000€	3000€	4000€	3000€	2500€	700€	8000€	2000€	1000€	1100€	1250€	]
sales[1] =	[	1200€	3200€	3200€	4500€	2000€	3000€	900€	8000€	2900€	1060€	100€	1300€	]
...														
sales[9] =	[	1000€	2000€	3000€	350€	300€	500€	600€	600€	900€	1900€	1000€	2000€	]

Jahr	Jan	...	Dez
sales[0] =	sales[0][0]	...	sales[0][9]
...	...	...	...
sales[9] =	sales[9][0]	...	sales[9][9]

Mehrdimensionale Felder werden durch den Datentyp gefolgt von mehreren Paaren von eckigen Klammern deklariert (ein Paar pro Dimension)

Syntax: `<Typ> ([])+ <Bezeichner>`
oder
`<Typ> <Bezeichner> ([])+` (unüblich)

- direkte Initialisierung eines mehrdimensionalen Feldes:

```
int [][] a = {{1,2,3},{4,5,6,7}};
// a ⟹ a ⟹ int[2][] { int[3] { 1, 2, 3 }, int[3] { 4, 5, 6, 7 } }
```

- Initialisierung mittels `new`-Operator:

  (Die Größe der einzelnen Dimensionen muss angegeben werden, kann unterschiedlich sein und kann auch Schritt-für-Schritt erfolgen.)

```
int [][] a = new int[2][];
a[0] = new int[3];
a[1] = new int[5];
// a ⟹ int[2][] { int[3] { 0, 0, 0 }, int[5] { 0, 0, 0, 0, 0 } }
```

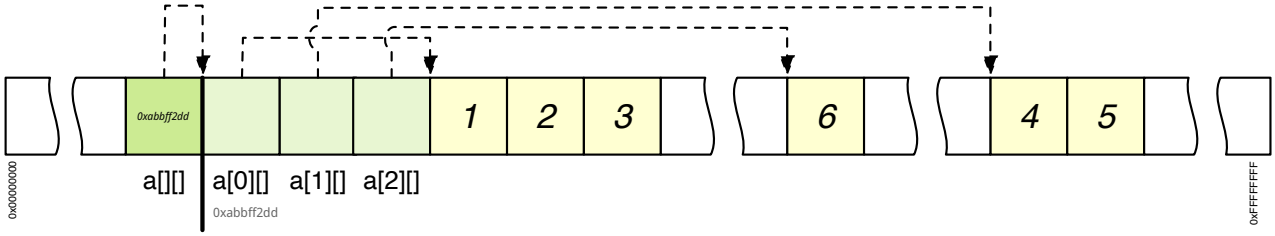
Auf die einzelnen Dimensionen eines mehrdimensionalen Feldes kann mittels einer Folge von Feldzugriff-Operatoren `[]` mit Indizes zugegriffen werden.

Der erste Feldzugriff-Operator liefert das Element der 1. Dimension, der zweite Feldzugriff-Operator liefert das Element der 2. Dimension, usw.

```
int [][][] a = {{{1,2,3},{4,5,6,7}},{{8,9,10},{11,12,13,14}}};

var x = a[0] // ==> int[2][] { int[3] { 1, 2, 3 }, int[4] { 4, 5, 6, 7 } }

var y = a[1][1] // ==> int[4] { 11, 12, 13, 14 }
```


0xabbff2dd
1
2
3
6
4
5
0x00000000
0xFFFFFFFF
a[][]
a[0][]
a[1][]
a[2][]
0xabbff2dd

Übung

3.1. Matrixmultiplikation

Schreiben Sie eine Methode `multiply`, die zwei 2-dimensionale Matrizen von `int` Werten entgegennimmt und die Matrixmultiplikation der beiden Matrizen berechnet.[1]

Dokumentieren Sie die Anforderungen an die Parameter und dokumentieren Sie den Rückgabewert. Überprüfen Sie die Anforderungen an die übergebenen Argumente mit `assert`-Anweisungen.

```
<TYP> multiply(<TYP> a, <TYP> b)
```

Lesen Sie die Matrizen mit Hilfe von `readln` Anweisungen schrittweise ein. Implementieren Sie die Funktionalität in einer Methode `initMatrix`. Schreiben Sie weiterhin eine Methode `printToConsole`, die eine Matrix auf der Konsole ausgibt.

Beispiel für die `main` Methode:

```
1  void main() {
2
3      final int[][] a = {
4          {1, 2, 3},
5          {4, 5, 6},
6          {7, 8, 9},
7          {10, 11, 12}
8      };
9      final int[][] b = {
10         {9, 8, 7},
11         {6, 5, 4},
12         {3, 2, 7}
13     };
14
15     final var c = multiply(a, b);
16     printToConsole(c);
17 }
```

---

[1] Matrixmultiplikation: Die Verrechnung erfolgt Zeile mal Spalte.

Übung

3.2. Sattelpunkte

Schreiben Sie eine Methode `printSaddlePoints` (`void printSaddlePoints(int [][] m)`), die die Sattelpunkte einer $n \times m$ Matrix von `int` Werten berechnet und auf der Konsole ausgibt. Die Sattelpunkte einer Matrix sind die Elemente der Matrix (n x m), die in Ihrer Zeile am kleinsten sind und in der Spalte am größten.

Beispiel

Die rechte Matrize hat zwei Sattelpunkte bei $(0, 1)$ und $(0, 2)$ jeweils mit dem Wert 0.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & -3 \\ 10 & -2 & -4 \end{pmatrix}$$

※ Hinweis

Verwenden Sie Ihre Methoden zum Einlesen und Ausgeben von Matrizen von vorher.

Verwenden Sie Methoden aus der vorherigen Übung wieder (zum Beispiel zum Einlesen einer Matrize bzw. zur Ausgabe.)

Übung

3.3. java.util.Arrays

Lesen Sie ein Array von der Kommandozeile ein und sortieren Sie es numerisch. D. h. wandeln Sie die Zahlen in echte Zahlen um und sortieren Sie danach. Danach geben Sie das sortierte Array aus. Schreiben Sie ein Java-Skript.

Beispiel

```
$ ./sort 3 2 1 4 5 6 7 8 9 10
[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]
```

Studieren Sie die verfügbaren Methoden der Klasse `Arrays` (https://docs.oracle.-com/en/java/javase/23/docs/api/java.base/java/util/Arrays.html). Suchen Sie nach einer Methode zum sortieren von Arrays mit dem Datentyp `int`. Suchen Sie weiterhin nach einer Methode, um das Array „vernünftig“ auszugeben.

※ Hinweis

(Für Mac und Linux Nutzer!)

Denken Sie daran, dass die erste Zeile eines echten Java-Skripts mit einem Shebang anfangen muss und der Dateiname des Skripts nicht mit *.java* enden darf. Denken Sie auch daran, dass das Skript ausführbar sein muss (z. B. Rechte 755).

Alternativ schreiben Sie eine Java Datei mit der entsprechenden Logik und führen Sie diese mit `java` aus.

!! Wichtig

Die *erste Zeile* eines *echten* Java-Skripts:

```
#!/usr/bin/env java --source 23 --enable-preview -ea
```

4. Varargs-Methoden

Methoden mit einer variablen Anzahl von Parametern (varargs)

Beispiel

```
1  void printAll(String separator, String... strings) {
2      for (int i = 0; i < strings.length; i++) {
3          print(strings[i]);
4          if (i < strings.length - 1) {
5              print(separator);
6          }
7      }
8  }
9
10 printAll(" + ", "2", "2", "3")
11 // ⟹ 2 + 2 + 3
```

- Pro Methode kann es nur einen *varargs* Parameter geben und dies muss der letzte Parameter sein.

  Syntax für den letzten Methodenparameter:

  `<Typ>... <Bezeichner>`

- Der Compiler erzeugt ein Array, das die übergebenen Parameter enthält.

  Best Practice: das varargs-Array sollte die Methode nicht verlassen.

- Die Methode verhält sich wie eine Methode mit einem Array als Parameter.

  ⇒ D. h. die Methoden können auch mit einem Array aufgerufen werden und sind bezüglich der Signatur nicht unterscheidbar. Demzufolge ist es auch nicht möglich zwei entsprechende Methoden zu definieren:

  ```
  1 double sum(double... values) { ... }
  ```

  ```
  1 double sum(double[] values) { ... } // nicht möglich
  ```

  ⇒ Die Methode kann mit `null` als Wert aufgerufen werden. In diesem Fall wird kein Array erzeugt sondern `null` übergeben.

  ```
  1 IO.println(sum(null)); // ⟹ ???
  ```

Übung

4.1. varargs

Schreiben Sie eine Methode `join`, die eine beliebige Anzahl von `int` Arrays (`int[]`) entgegennimmt und daraus ein Array erzeugt.

Dokumentieren Sie die Methode ausführlich. Achten sie darauf alle Sonderfälle abzudecken.

Beispiel

```
jshell> var r = join(new int[]{1,2}, new int[]{3,4})
r ==> int[4] { 1, 2, 3, 4 }

jshell> var r = join(
        new int[]{1,2},
        new int[]{3,4},
        new int[]{5,6,7,8,9})
r ==> int[9] { 1, 2, 3, 4, 5, 6, 7, 8, 9 }
```