Programmieren II: Java

Arrays

Prof. Dr. Christopher Auer

Sommersemester 2024

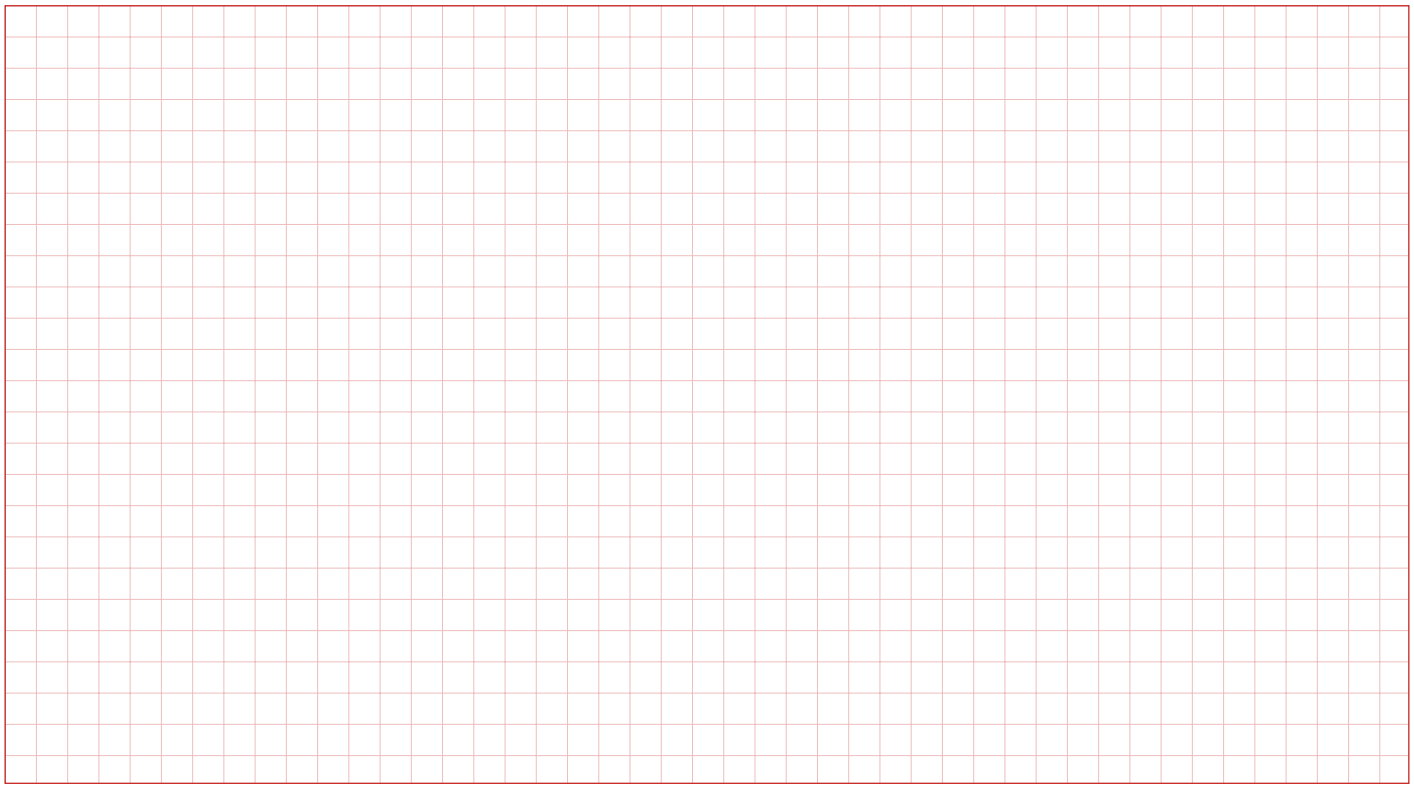
Notizen

Eindimensionale Arrays

Mehrdimensionale Arrays

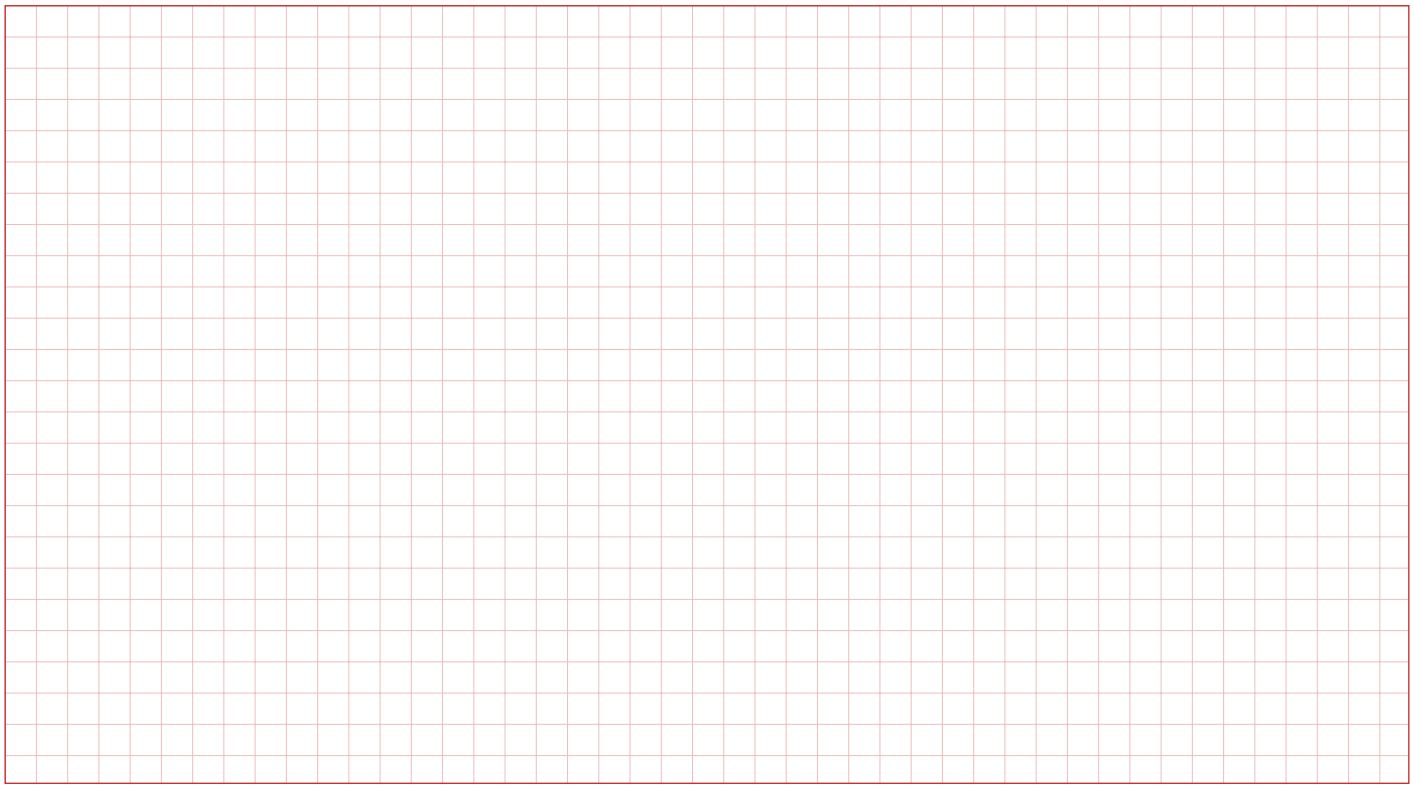
Arbeiten mit Arrays

Notizen



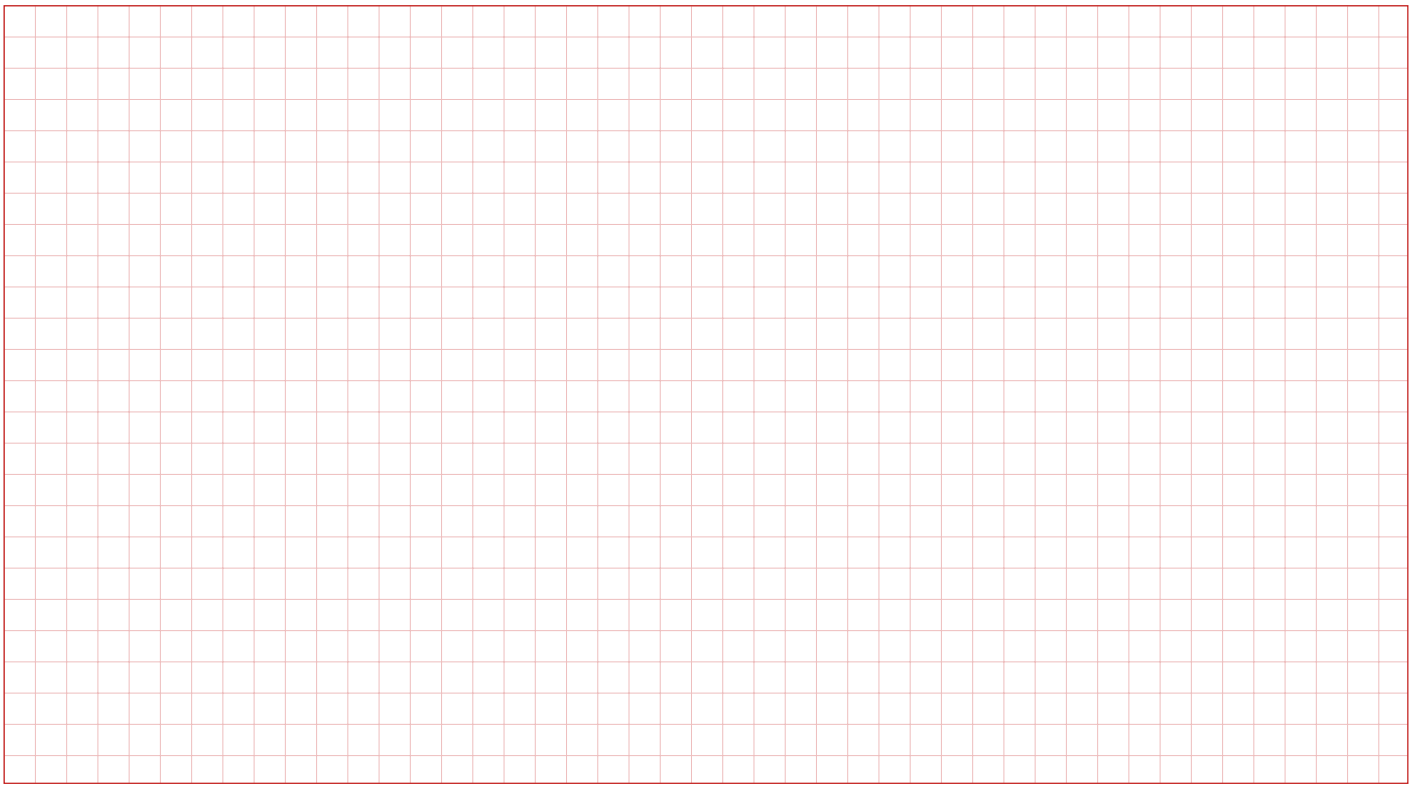
Inhalt

Notizen



Inhalt

Notizen



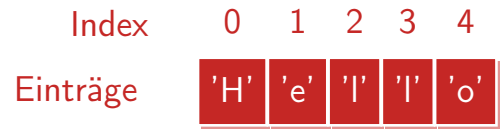
Was sind Arrays?

- Arrays (auch „Felder“)
  - sind eine Aneinanderreihung von Elementen eines Typs
  - haben eine unveränderliche Länge (Anzahl Einträge)
  - Erlauben Zugriff über einen Index (Position)
- Beispiel:

```
char[] hello = new char[5];
hello[0] = 'H'; hello[1] = 'e';
hello[2] = 'l'; hello[3] = 'l';
hello[4] = 'o';
```

(Details folgen später)

- Array `hello`
  - beinhaltet **char**s
  - hat Länge 5
- Grafische Darstellung



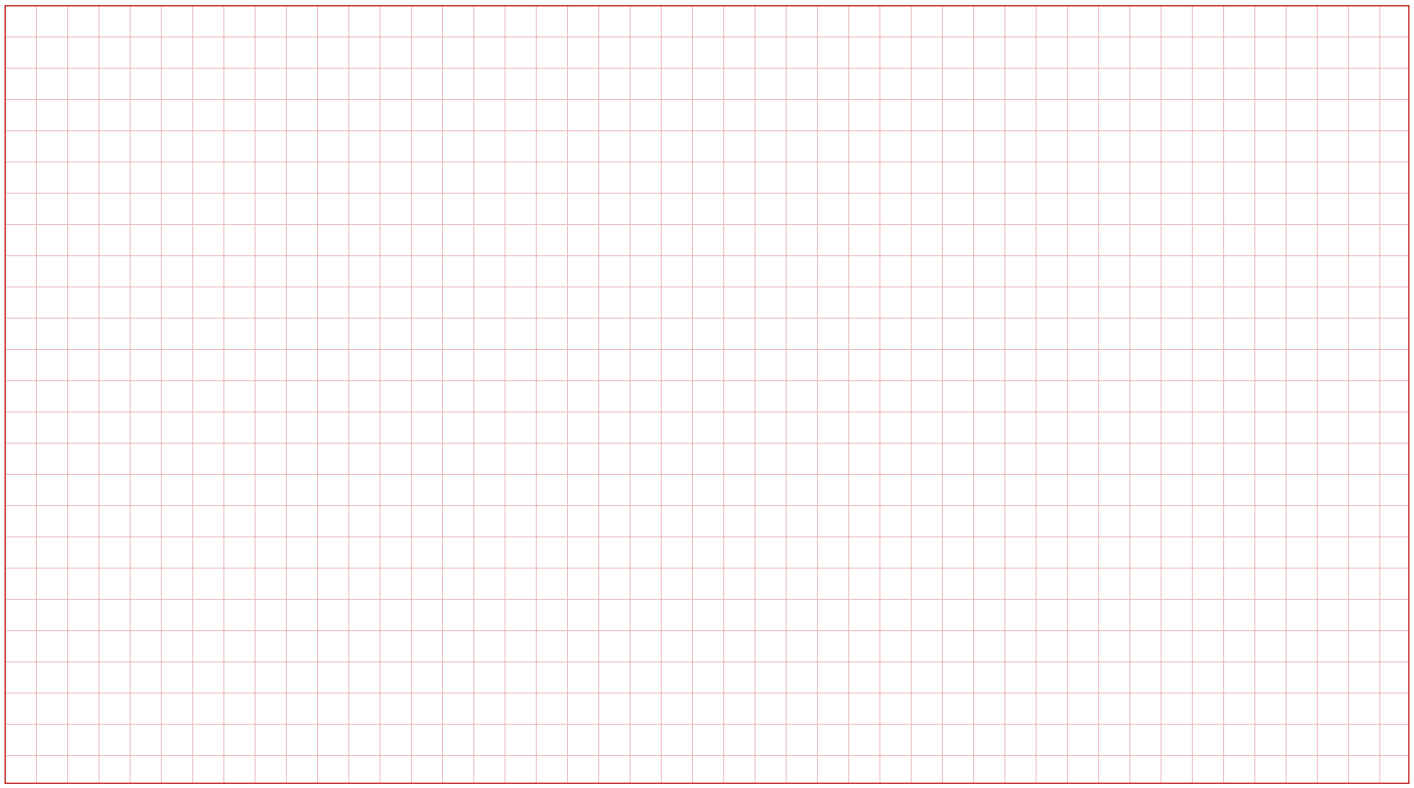
Notizen

Arraytypen

- Arrays bilden eine Familie von Typen
- Für jeden Java-Typen `T` gibt es einen „`T`-Array“
- `T[]` bezeichnet den Datentyp: Array mit Elementen von Typ `T`
- Beispiele

Arraytyp	Bezeichnung	Beispiel
`byte[]`	**`byte`**-Array	Datenstrom
`int[]`	**`int`**-Array	Zahlenfolge
`double[]`	**`double`**-Array	Messwerte
`char[]`	**`char`**-Array (`!= String`)	Encoding
`Point2D[]`	`Point2D`-Array	Polygon
`Weekday[]`	`Weekday`-Array	Arbeitstage

Notizen



Deklaration einer Array-Variable

- Deklaration einer Array-Variable `x` vom Typ `T`

```
T[] x;
```

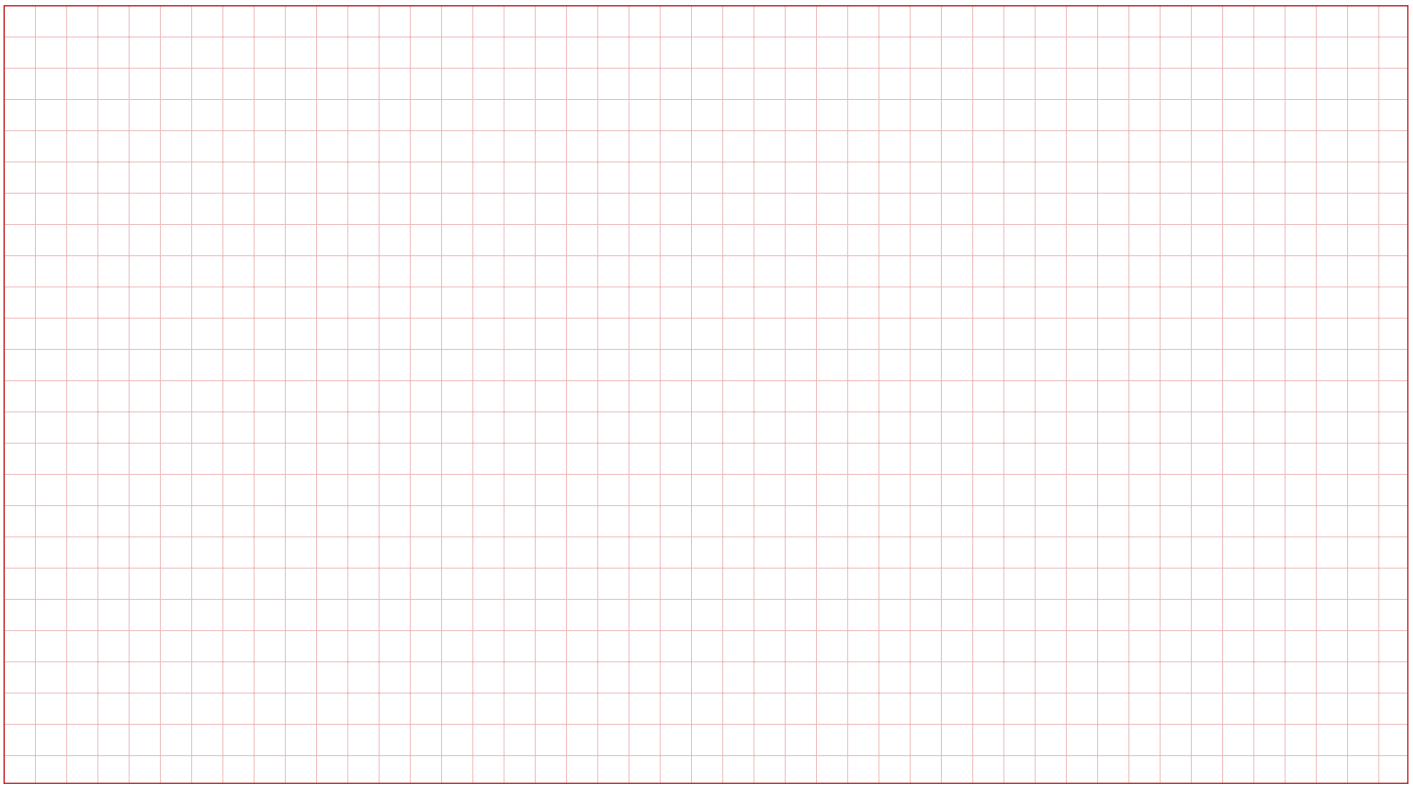
- Hinweise
  - Deklaration erzeugt keinen Array
  - Vergleiche: Deklaration einer Objektreferenz-Variable erzeugt kein Objekt

```
Point2D p;
```

  - `x` ist Referenzvariable (`T[] x = null;` möglich)
  - `x` hat (noch) keine Länge
- Beispiele

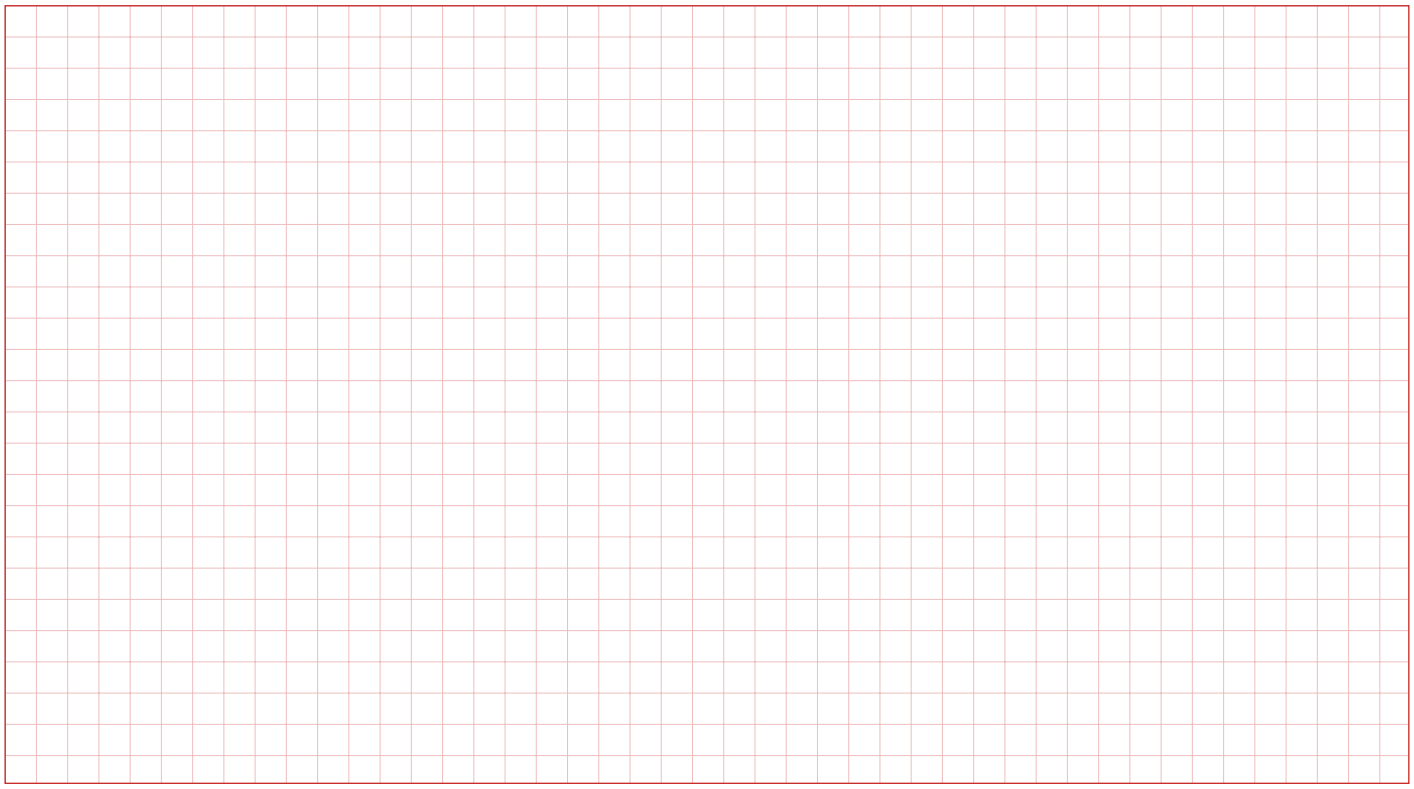
```
int[] numbers;
double[] measurements;
Point2D[] points;
Weekday[] workingDays;
```


Notizen



Inhalt

Notizen

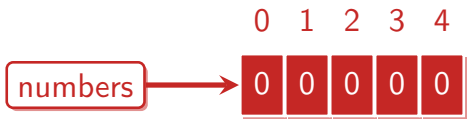


Allokieren von Arrays (eindimensional)

- Allokieren von Arrays über **new**-Operator

```
int[] numbers = new int[5];
```

  - Allokiert Speicher für Array der Größe 5
  - Größe muss hier festgelegt werden
  - Ergebnis: **int**-Array-Referenz



  - Hinweise
    - In Java werden Arrays mit Default-Werten belegt (0 bei numerischen Werten, **null** bei Referenzen)
    - Erzeugung von Array mit Objektreferenzen erzeugt keine Objekte (Default-Wert **null**)
    - Erzeugung muss nicht bei Deklaration stattfinden

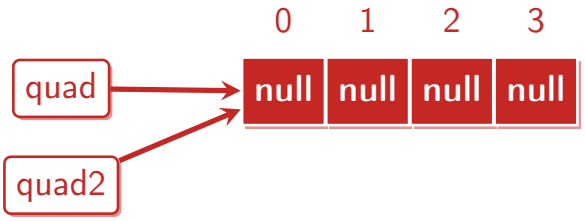
```
int[] numbers;
/* ... */
numbers = new int[5];
```


Notizen

Array-Referenzen

- Nochmals: Array-Variablen sind Referenzvariablen
- Mehrere Array-Variablen können auf gleichen Array zeigen

```
Point2D[] quad = new Point2D[4];
Point2D[] quad2 = quad;
```



- Wie bei Instanzvariablen: Lesender und schreibender Zugriff

Notizen

Größe von Arrays

- Größe von Array-Instanzen
  - wird über `length` zugegriffen

```
int numbers[] = new int[5];
System.our.println(numbers.length); // 5
```

  - ist unveränderlich
  - darf nicht negativ sein
  - darf aber 0 sein (Konsistenz)
  - muss erst bei der Erstellung feststehen

```
10 runRandomArrayLengthExample
11 int length = (int) (Math.random()*100);
12 int[] numbers = new int[2*length+20];
13 System.out.printf("numbers.length == %d%n",
14     numbers.length);
```

OneDimArrayExamples.java

```
numbers.length == 38
```


Notizen

Größe von Arrays: Beispiele

- Zugriff auf `length` nur lesend

```
Weekday[] workingDays = new Weekday[5];
workingDays.length = 4; // FEHLER Cannot change final attribute
```

- Methode `createIntArray` erstellt **`int`**-Array gegebener Länge

```
20 runCreateIntArray
21 public static int[] createIntArray(int length) {
22   System.out.printf("new int[%d]%n", length);
23   int[] numbers = new int[length];
24   System.out.printf("numbers.length == %d%n",
25       numbers.length);
26   return numbers;
27 }
```

OneDimArrayExamples.java

Notizen

Größe von Arrays: Beispiel

- Aufrufe von `createIntArray`
  - `length = 5`

```
new int[5]
numbers.length == 5
```

  - `length = 1`

```
new int[1]
numbers.length == 1
```

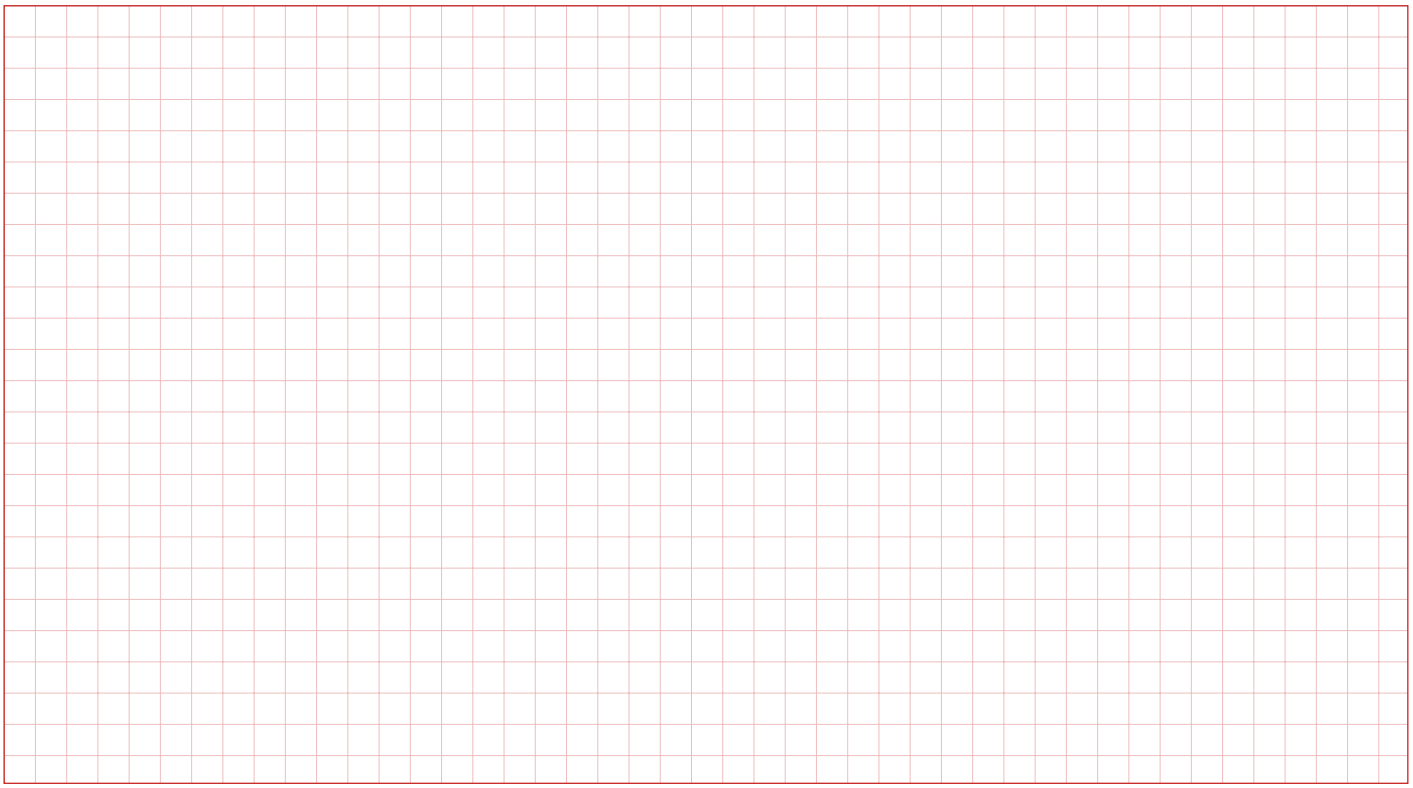
  - `length = 0`

```
new int[0]
numbers.length == 0
```

  - `length = -3`

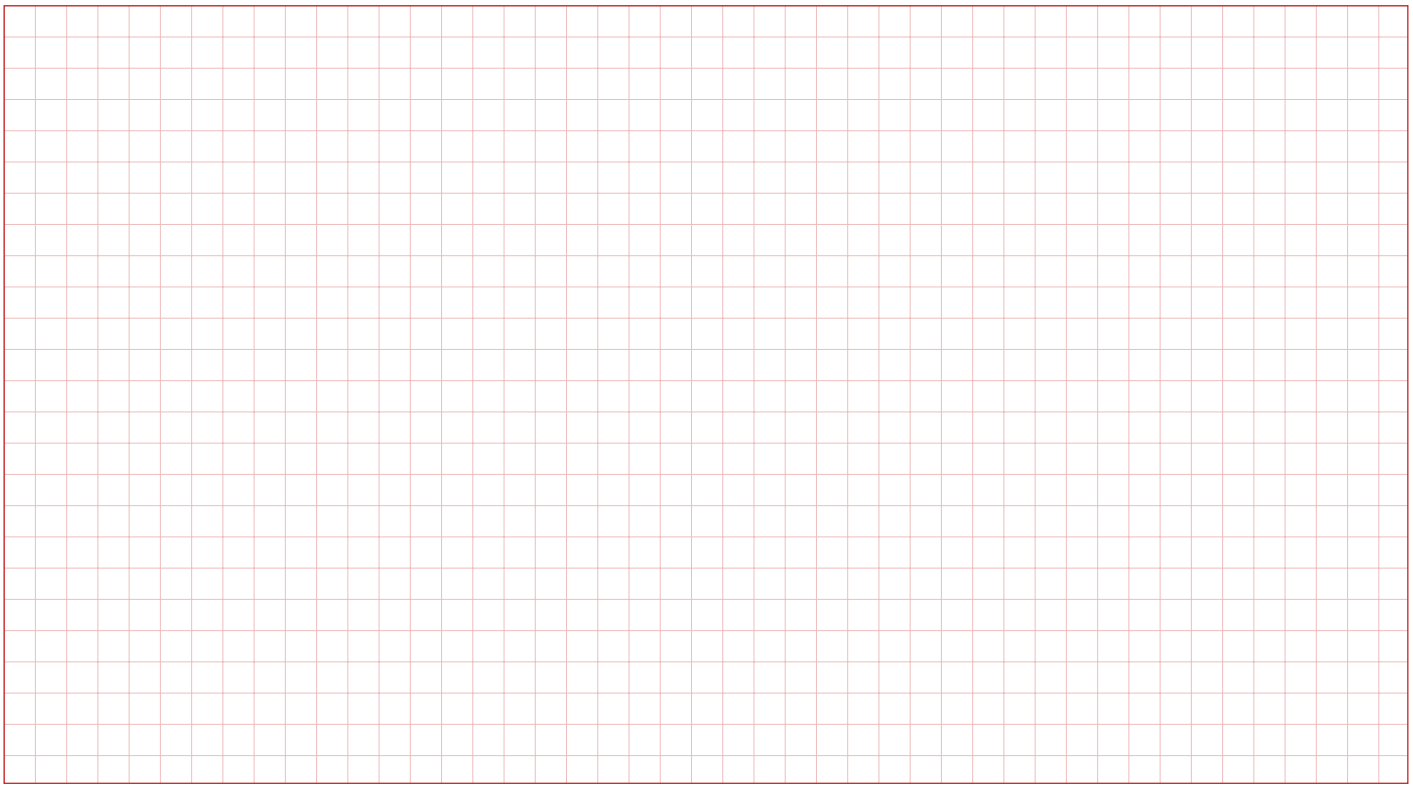
```
new int[-3]
FEHLER NegativeArraySizeException
```


Notizen



Inhalt

Notizen



Array-Literale

- Beispiel von vorher:

```
char[] hello = new char[5];
hello[0] = 'H';
hello[1] = 'e';
hello[2] = 'l';
hello[3] = 'l';
hello[4] = 'o';
```

- Manuelles befüllen von Arrays mühsam
- Array-Literale

```
char[] hello = new char[] { 'H', 'e', 'l', 'l', 'o' };
```

  - Erzeugt mit Werten vorbelegten Array
  - Größe steht zur Übersetzungszeit fest
  - `new char[]`: Größe wird durch Anzahl Einträge ermittelt
  - Werte müssen zum Typ des Arrays passen

```
int[] xs = new int[] { null, 1, 2, 3 }; // FEHLER
```


Notizen

Array-Literale

- Nur Größe des Array-Literals steht zur Übersetzungszeit fest
- Der Inhalt von Array-Literalen kann zur Laufzeit festgelegt werden

```
31  runArrayMultiplesExample
32  public static int[] arrayMultiplesExample(int n) {
33    int[] multiples = new int[] {1 * n, 2 * n, 3 * n};
34    return multiples;
35  }
```

OneDimArrayExamples.java

- `n == 1`: `[1, 2, 3]`
- `n == 3`: `[3, 6, 9]`
- `n == -4`: `[-4, -8, -12]`

Notizen

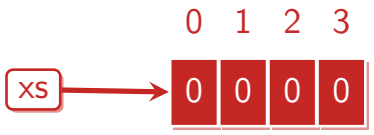
Inhalt

Notizen

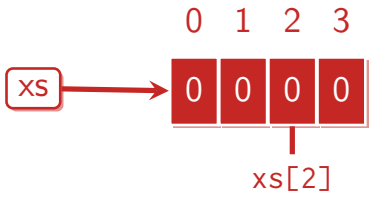
Zugriff auf Elemente eines Arrays

- Gegeben: **int**-Array mit vier Einträgen

```
int[] xs = new int[4];
```



- Elementzugriff erfolgt über Index
  - Kleinster Index: 0 (immer)
  - Größter Index: length-1 (hier 3)
  - Alle anderen Werte: `IndexOutOfBoundsException`
- `[]`-Operator: `xs[2]` gibt das Element an Stelle 2 in `xs`

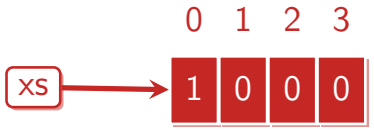


Notizen

Elementzugriff

- Schreibender Zugriff: Wie bei Variablen

```
xs[0] = 1;
```

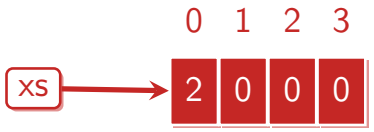


- Lesender Zugriff: Auch wie bei Variablen

```
System.out.println(xs[0]); // 1
```

- Allgemein: `array[index]` liefert LValue, d.h. etwas das einen Wert aufnehmen kann (vgl. Kapitel zu Operatoren)

```
xs[0]++;
```

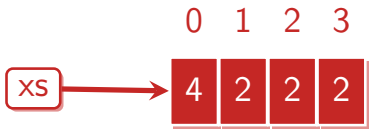


Notizen

Elementzugriff

- Wert von Index muss erst zur Laufzeit feststehen

```
for (int i = 0; i < xs.length; i++)
  xs[i] += 2;
```



- Hauptsache Index ist **int**-kompatibel

```
xs['a'] = 5;
```

  - 'a' ist **char**
  - Implizite Umwandlung in **int** ergibt 97
  - ArrayIndexOutOfBoundsException
  - Möglich, aber macht man nicht!

Notizen

Arrays überall!

- Arrays sind überall erlaubt
- Lokale Variablen

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

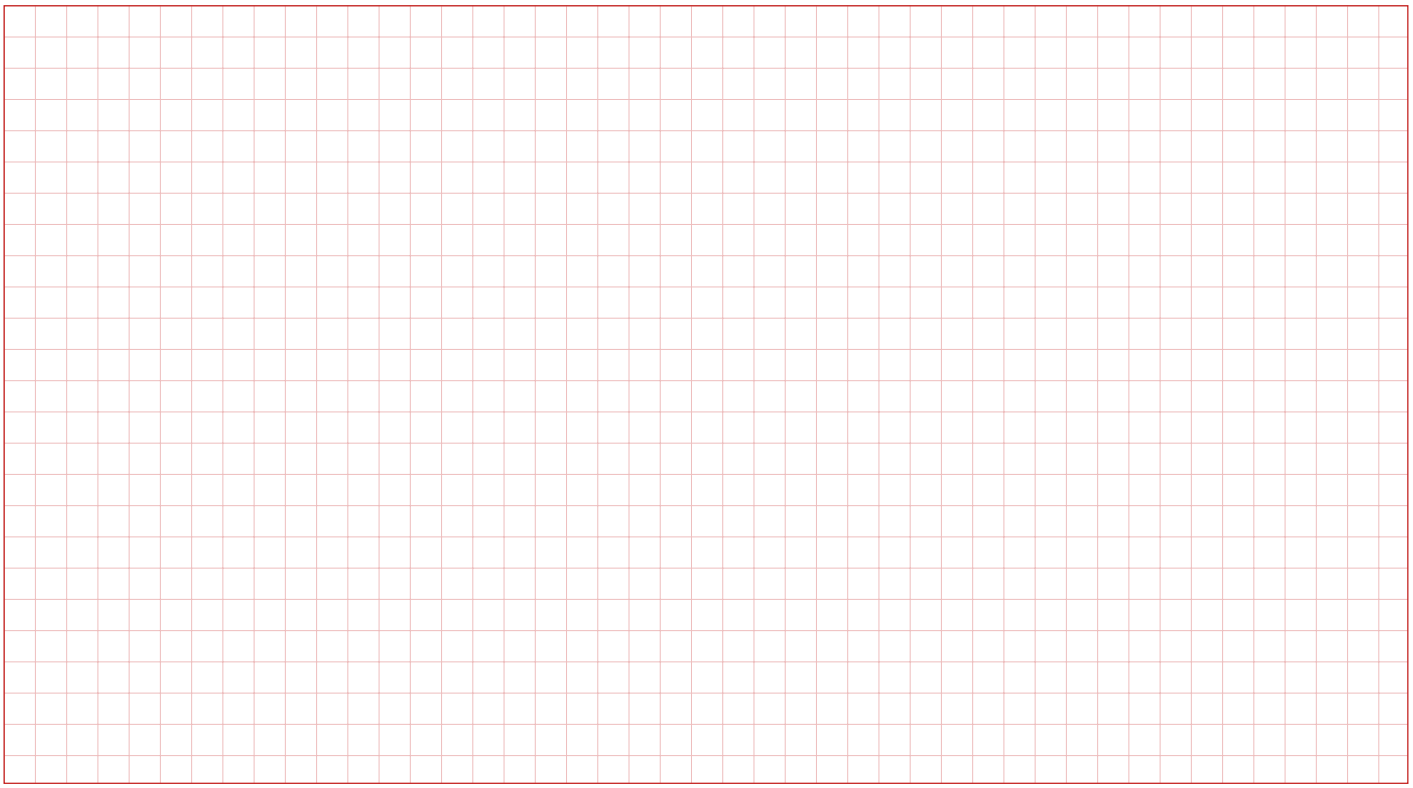
- Parameter

```
public int void sum(int[] numbers){
  int sum = 0;
  for (int i = 0; i < numbers.length; i++)
    sum += numbers[i];
  return sum;
}
```

- Rückgabewert

```
public int[] void generatePrimes(int n){
  int[] primes = new int[n];
  /* generate primes */
  return primes;
}
```


Notizen



Arrays überall!

- Objektvariablen: auch **protected**, **final**, etc.

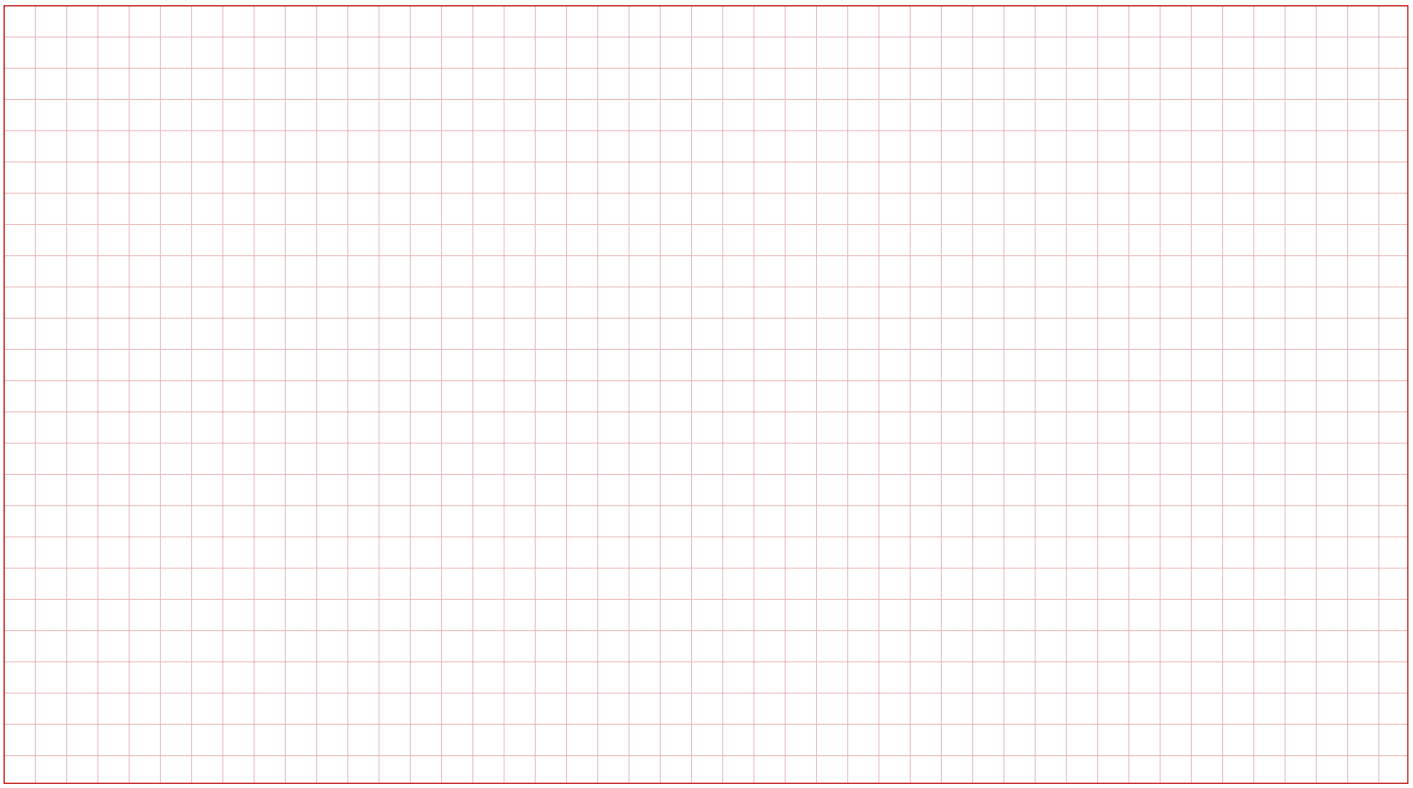
```
public class ArrayContainer{
  private int[] xs;
  public ArrayContainer(){
    this.xs = new int[10];
  }
}
```

- Klassenvariablen

```
public class ArrayContainer{
  private static int[] xs;
  /* ... */
  public void setEntry(){
    ArrayContainer.xs[0] = 1;
  }
}
```

(Eher seltener)

Notizen



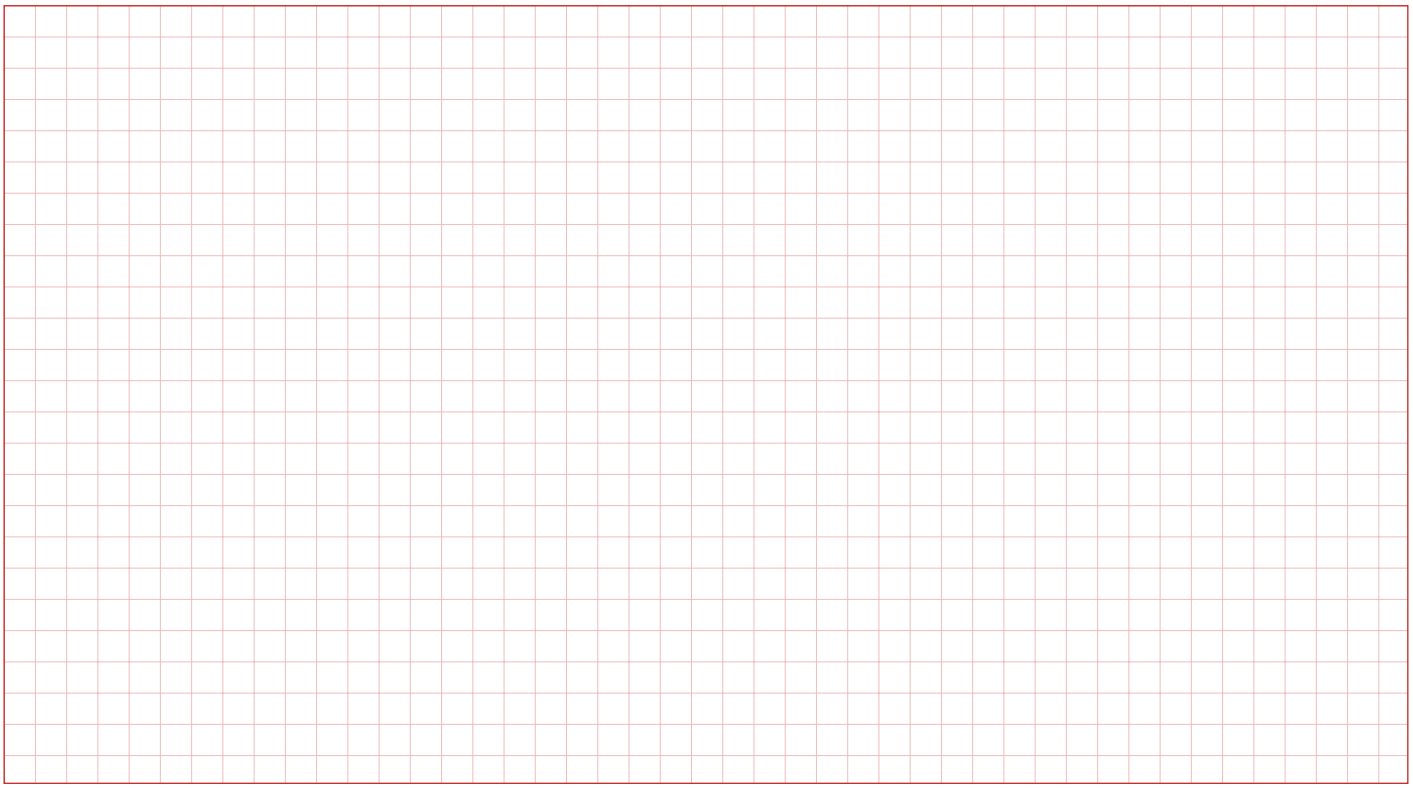
Arrays überall!

- Als Rückgabewert des Bedinungsoperator

```
int[] xs = ...;
int[] ys = ...;
(i % 2 = 0 ? xs : ys)[0] = 1;
```

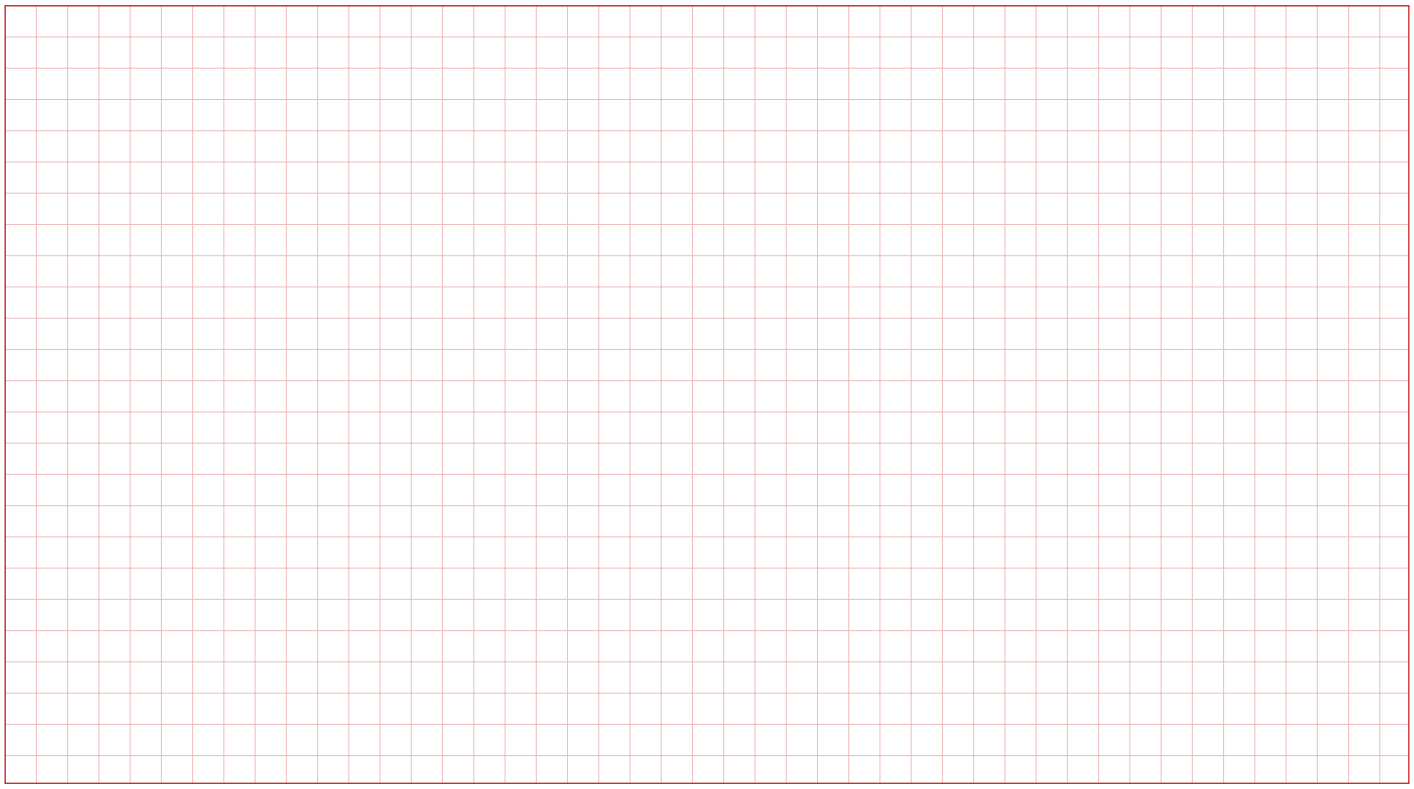
(Eher noch seltener)

Notizen



Inhalt

Notizen



Arrays aus Referenzen

- Arrays können auch Referenzen beinhalten

```
41 runArrayReferencesExample
42 Point2D[] points = new Point2D[4];
43 points[0] = new Point2D(2,3);
44 points[1] = new Point2D(-1,2);
46 points[2] = points[0];
47 points[2].move(5,5);
49 points[3].set(0,0); // NullPointerException
```

OneDimArrayExamples.java

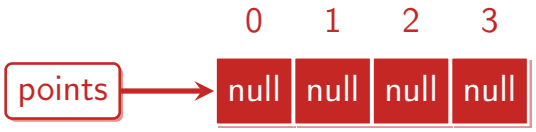
- Hinweise
  - Array wird mit **null** initialisiert
  - Insbesondere werden keine Instanzen automatisch erstellt

Notizen

Arrays aus Referenzen

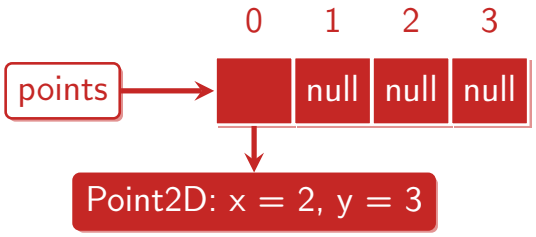
- Erzeugung des Arrays

```
Point2D[] points = new Point2D[4];
```



- Erstellen eines Punktes

```
points[0] = new Point2D(2,3);
```

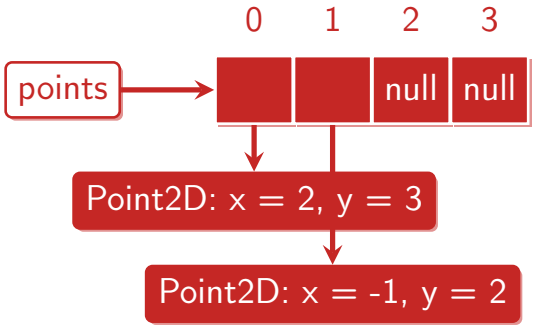


Notizen

Arrays aus Referenzen

- Erstellen noch eines Punktes

```
points[1] = new Point2D(-1,2);
```

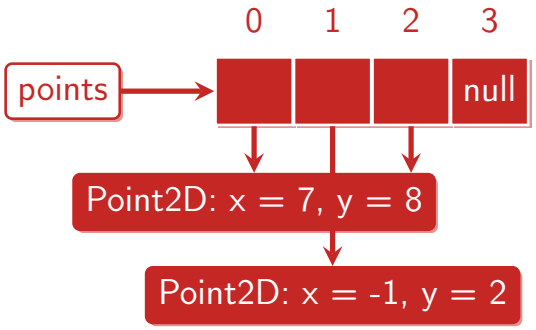


Notizen

Arrays aus Referenzen

- Zuweisung auf existierende Instanz und Methodenaufruf

```
points[2] = points[0];
points[2].move(5,5);
```



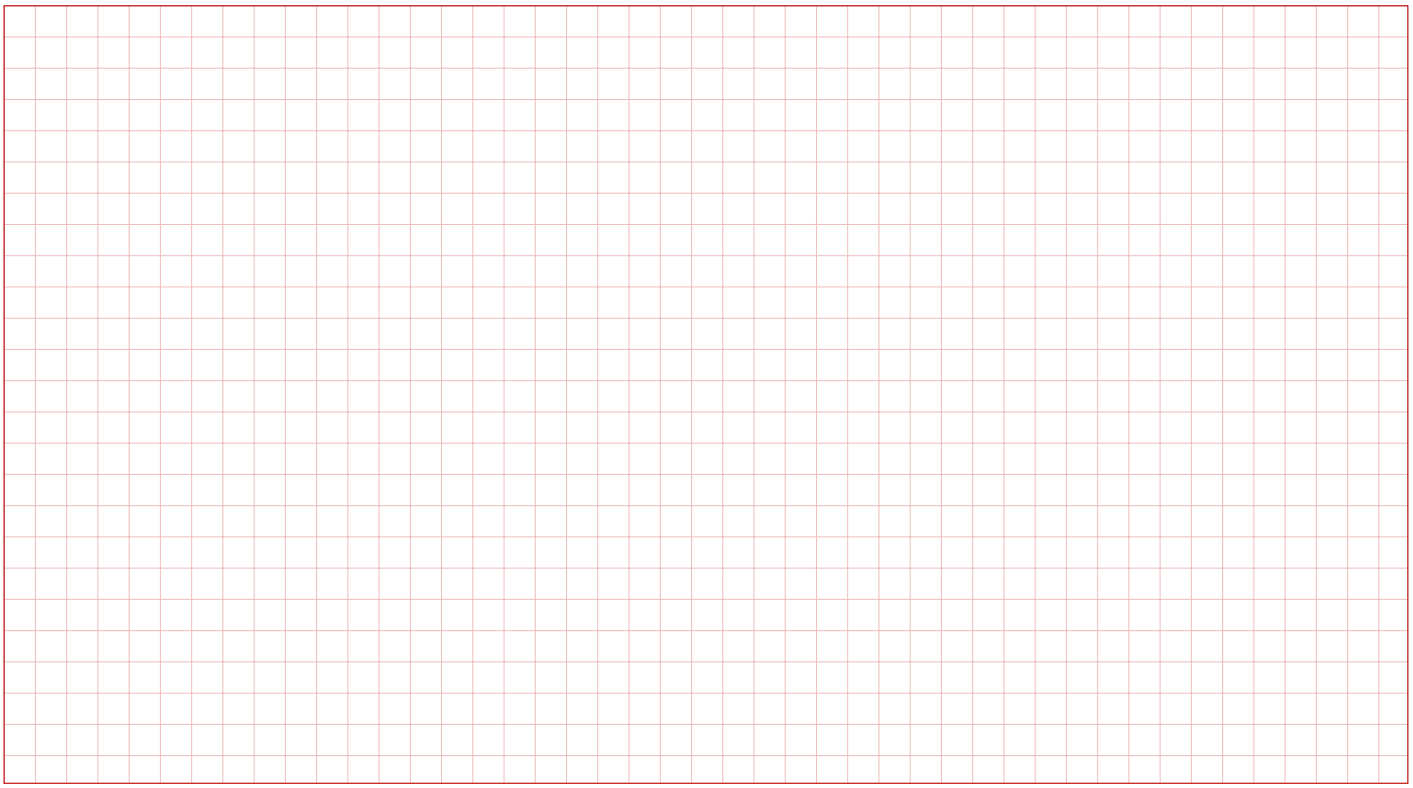
- Zugriff auf Index 3 (**`null`**)

```
points[3].set(0,0); // NullPointerException
```


Notizen

Inhalt

Notizen



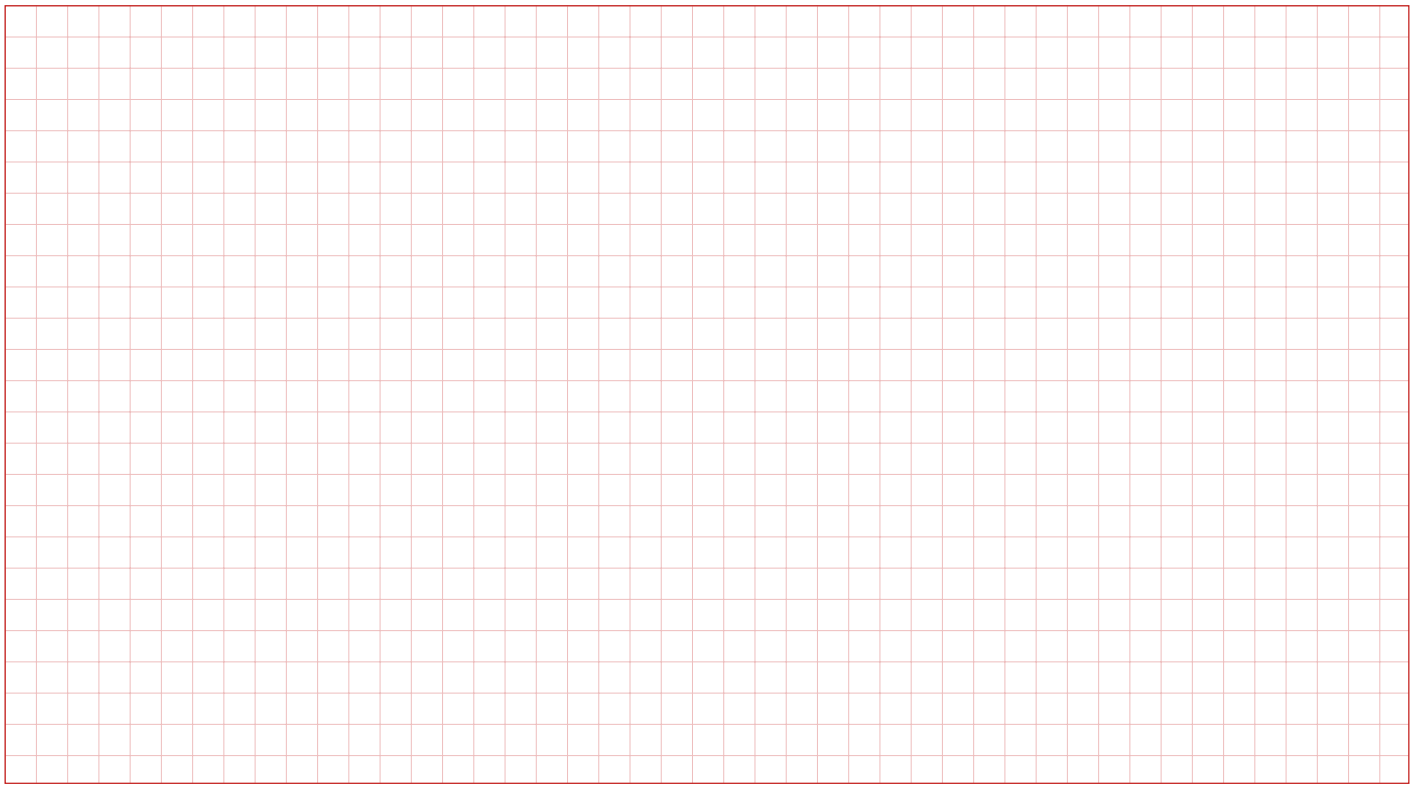
Vergleich C-Arrays mit Java-Arrays

- C- und Java-Arrays haben viele Gemeinsamkeiten, aber auch Unterschiede

Eigenschaft	Java	C
0-basiert	ja	ja
Typ	Referenz	Zeiger (`int*`)
Länge	Attribut `length`	aus Array nicht ersichtlich
Zugriffsschutz	ja	nein
`[]`-Operator	nur Arraytypen	prinzipiell jeder Typ

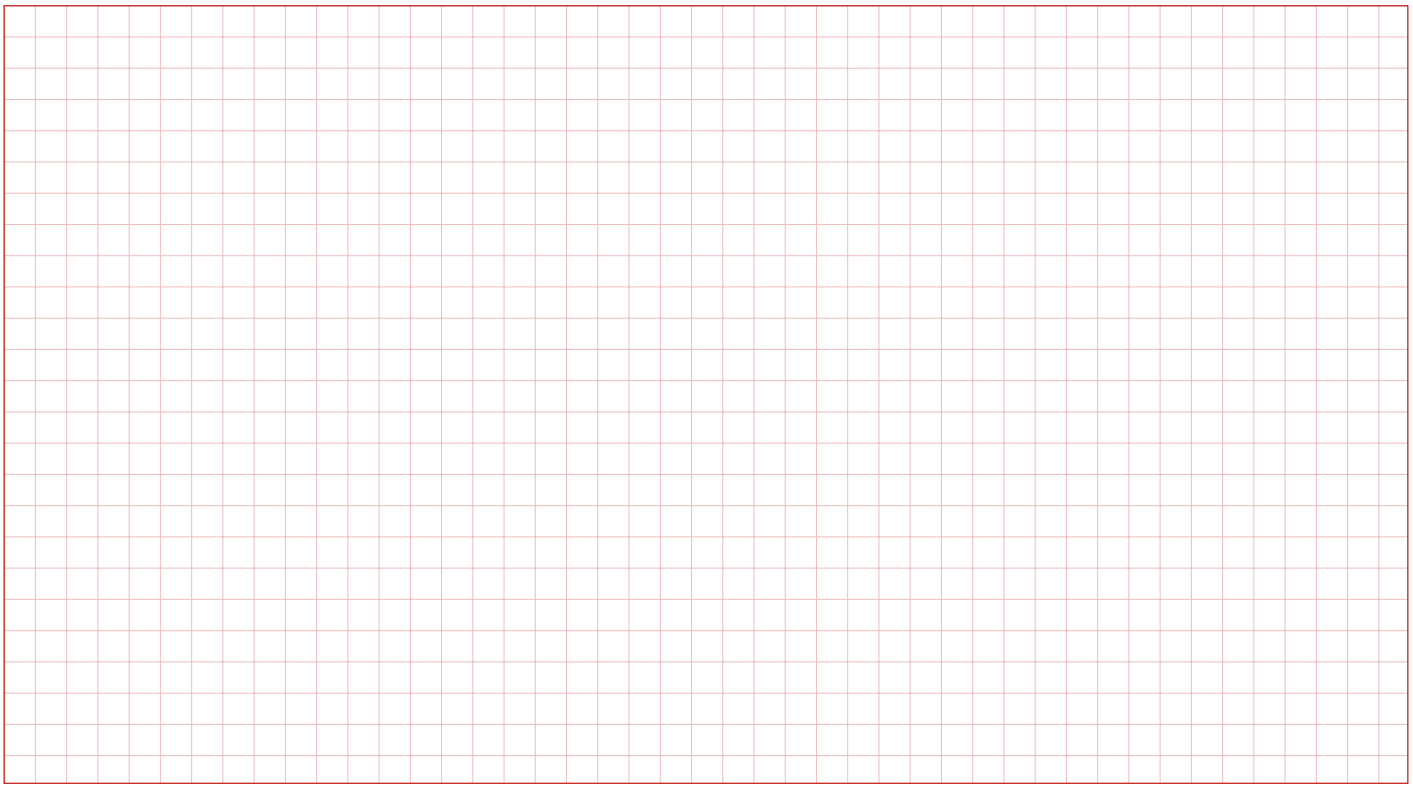
- Java-Arrays sind im Vergleich
  - komfortabler (z.B. Längenermittlung)
  - sicherer (Zugriffsschutz)
  - etwas langsamer (wegen Prüfungen)

Notizen



Inhalt

Notizen



Klassische for-Schleife

- Klassische **for**-Schleife wie in C

```
7  runArraySumExample
8  int[] numbers = new int[] {1,2,3,4,5};
9  int sum = 0;
11 for (int i = 0; i < numbers.length; i++) {
12   sum += numbers[i];
13 }
15 System.out.printf("sum = %d%n", sum);
```

ArrayIterateExamples.java

- Solche **for**-Schleifen sehen immer gleich aus
  - `i` — Laufvariable, Index
  - **`int`** `i = 0` — Starte beim ersten Index
  - `i < numbers.length` — bis zum letzten Index (beachte <!)
  - `i++` — inkrementiere Laufvariable um 1

Notizen

Varianten: Klassische for-Schleife

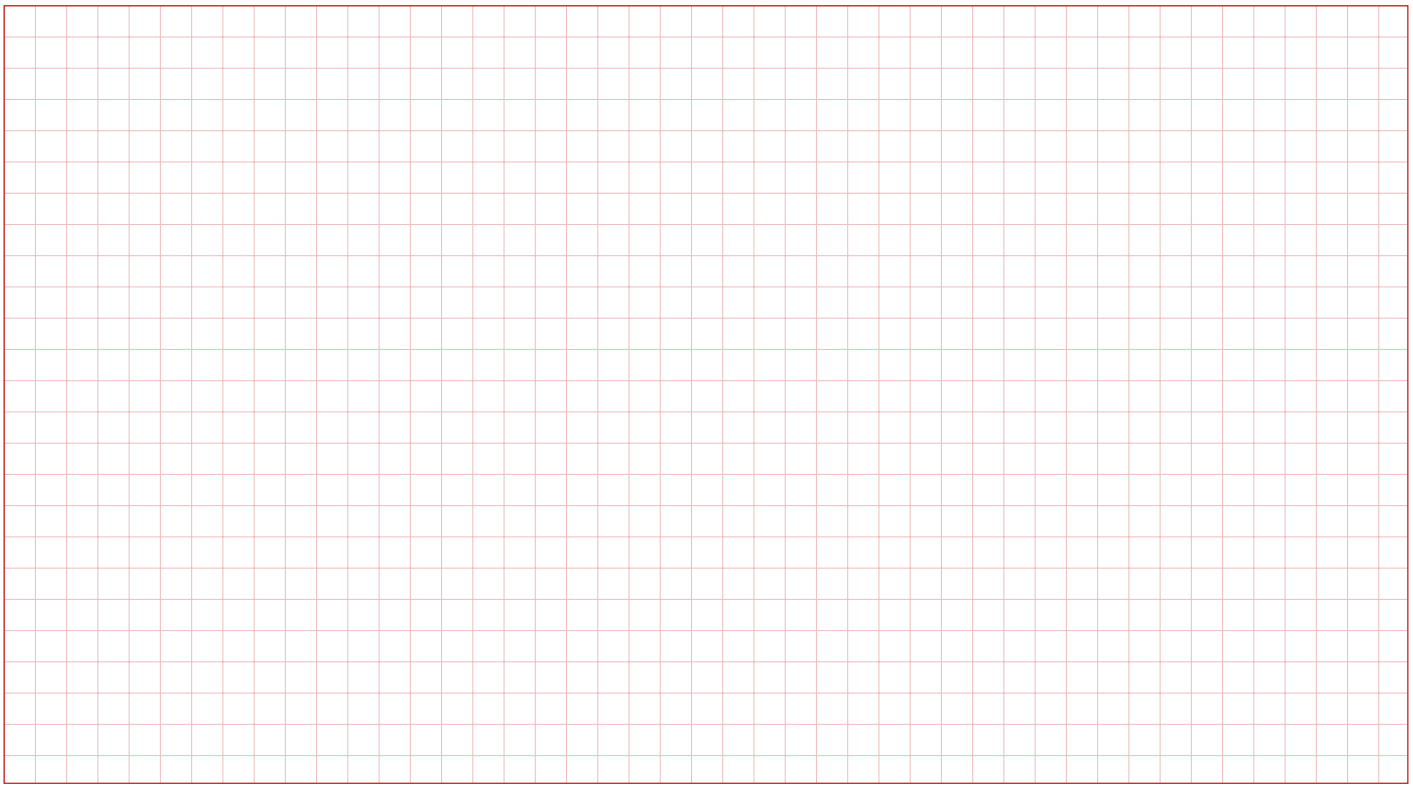
- Komplizierteres Beispiel: Bubble-Sort

runArraySortExample

```
42 int[] numbers = new int[] {4,1,8,7,2,9,3,5,6};
43 int n = numbers.length;
44 for (int i = 0; i < n-1; i++){
45   for (int j = 0; j < n-i-1; j++){
46     if (numbers[j] > numbers[j+1]){
47       int temp = numbers[j];
48       numbers[j] = numbers[j+1];
49       numbers[j+1] = temp;
50     }
51   }
52 }
53 printArray(numbers); // [ 1 2 3 4 5 6 7 8 9 ]
54
```

ArrayIterateExamples.java

Notizen



for-each-Schleife

- Beispiel von vorher mit **for**-each-Schleife (vgl. Kapitel „Grundlagen“)

```
21 runArraySumForEachExample
22 int[] numbers = new int[] {1,2,3,4,5};
23 int sum = 0;
24 for (int number : numbers) {
25   sum += number;
26 }
28 System.out.printf("sum = %d%n", sum);
```

ArrayIterateExamples.java

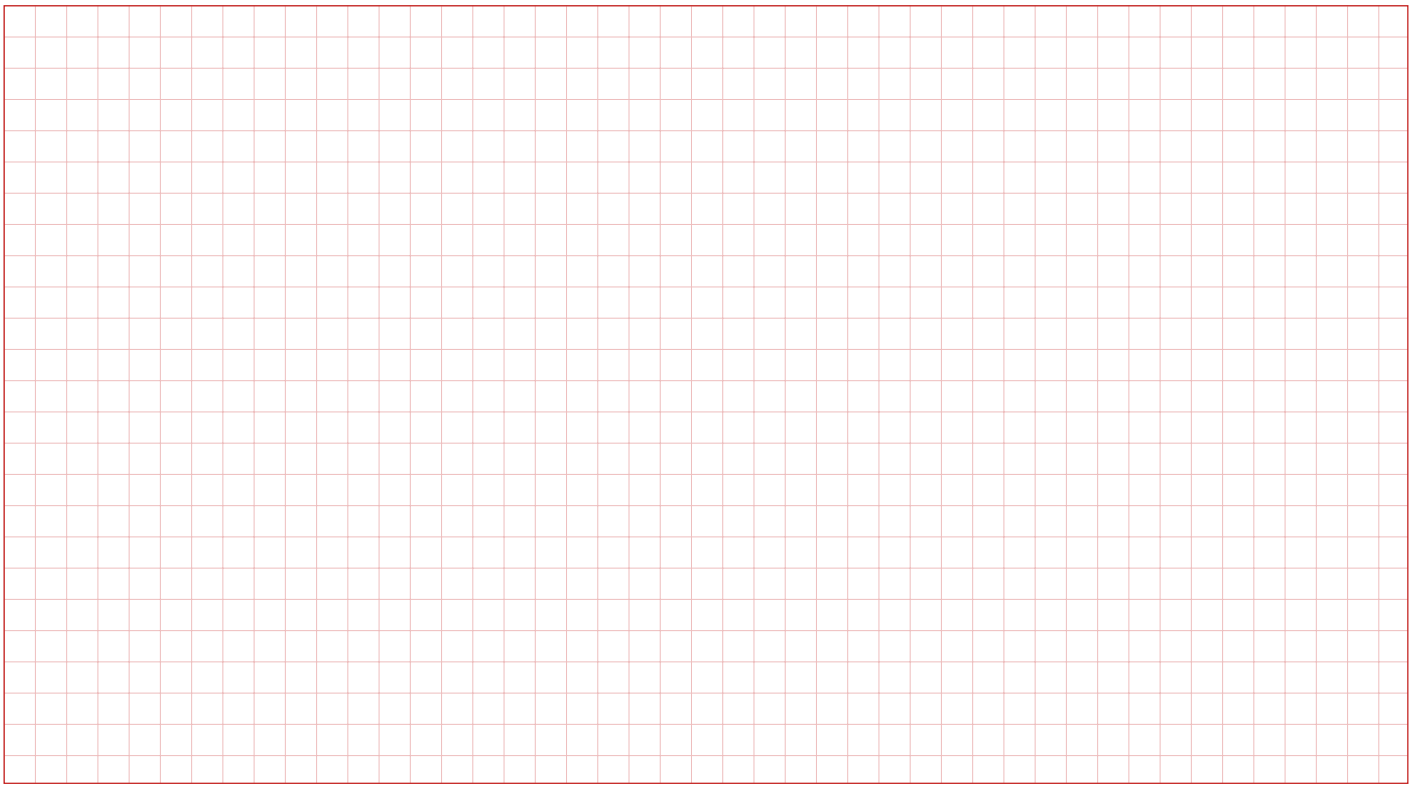
- Zur Erinnerung: Wird intern in „klassische“ **for**-Schleife umgebaut
- Ist übersichtlicher
- Aber: Kein aktueller Index verfügbar
- (Bubble-Sort nicht ohne Index-Variablen möglich)

Notizen

Klassisch vs. `for-each`

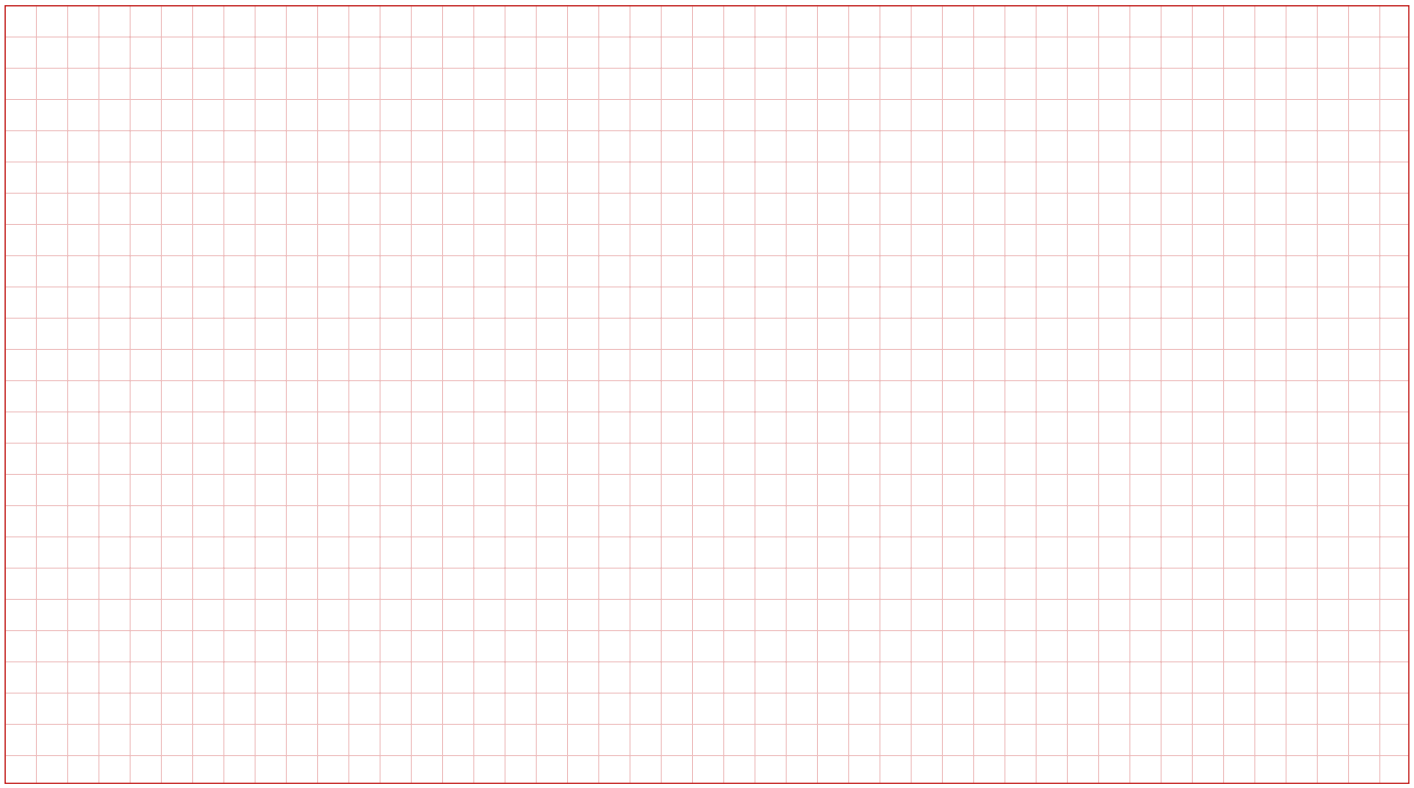
- Klassische `for`-Schleife wenn
  - Index notwendig
  - paralleles Durchlaufen von zwei (mehr) Arrays
  - Änderungen am Array (vgl. Bubble-Sort)
- `for`-each-Schleife wenn
  - nur Werte notwendig, z.B. bei Ausgabe oder Aggregationen (Summe, Suche)

Notizen



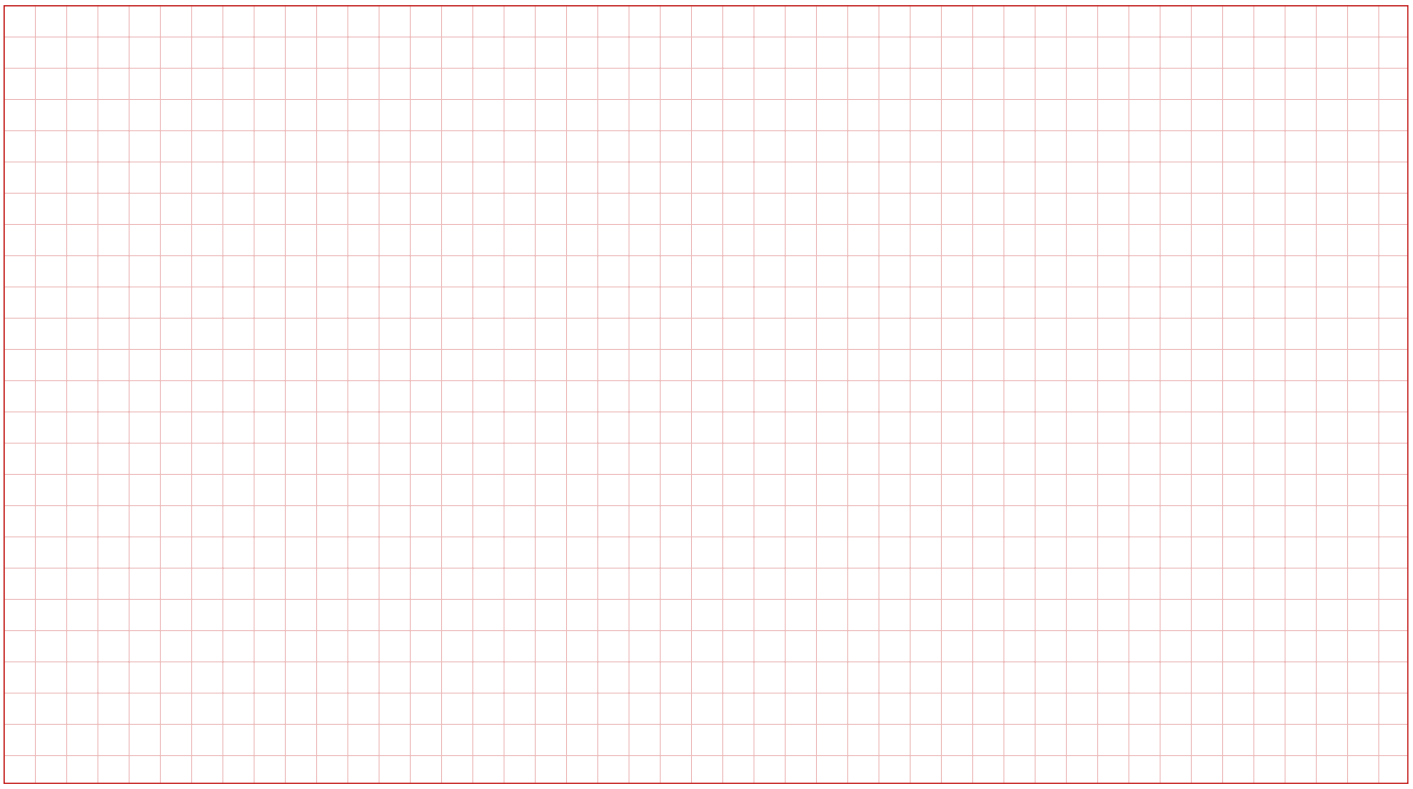
Inhalt

Notizen



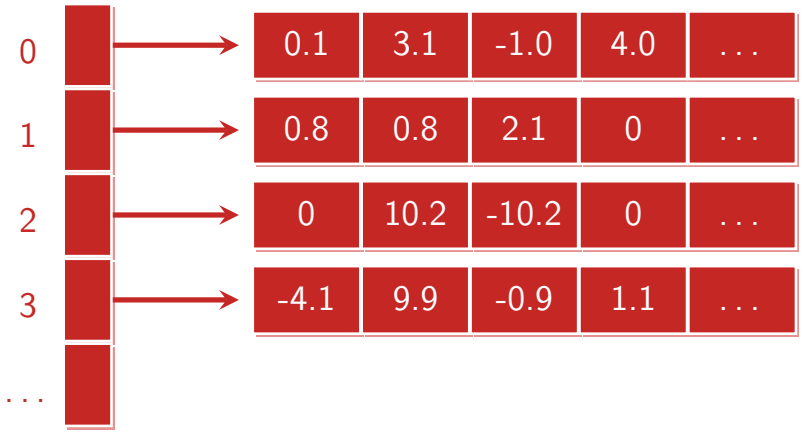
Inhalt

Notizen



Array von Arrays

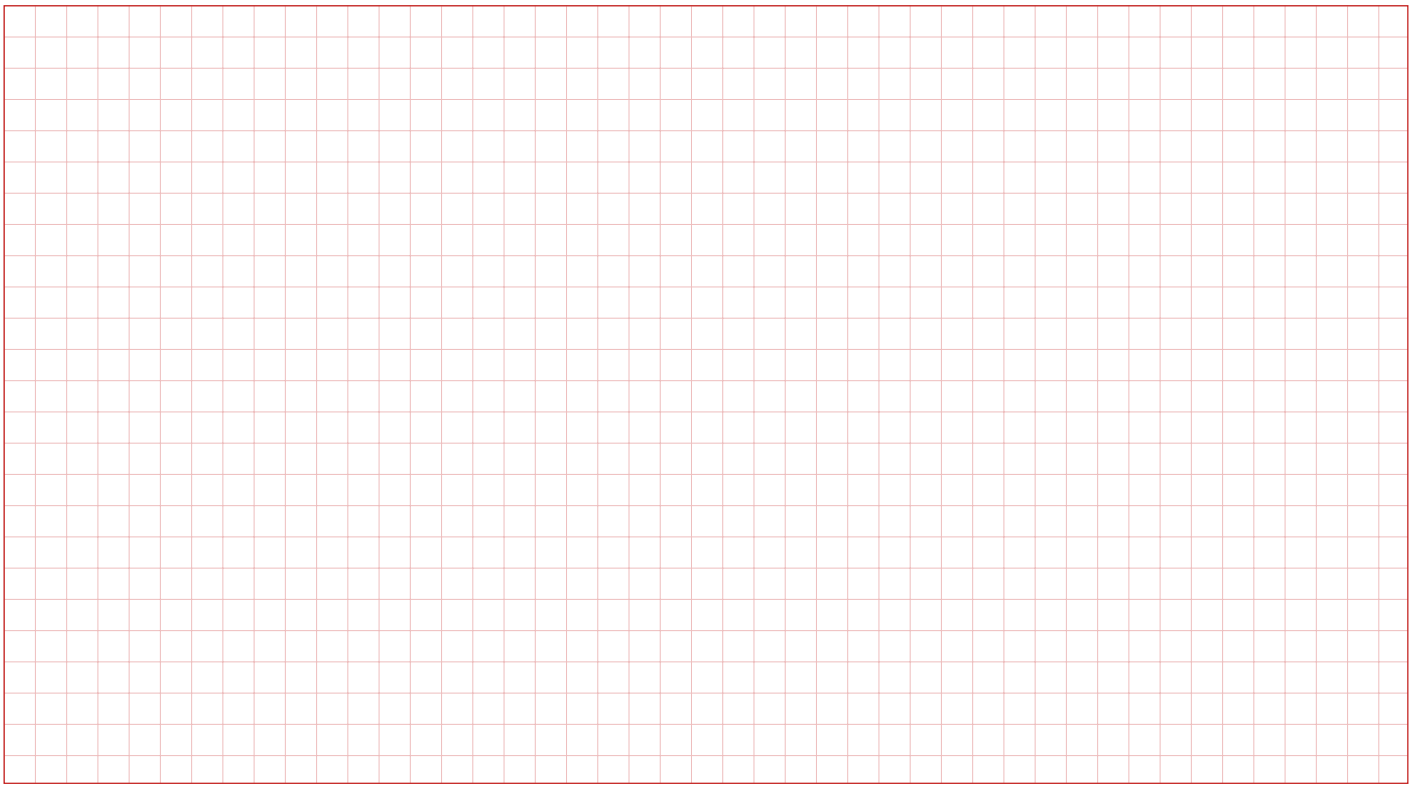
- Arrays können von beliebigem Java-Typ sein
- Also auch Arrays
- Beispiel `double[][]`
- Ein Array von **`double`**-Arrays
- Oder: zweidimensionaler **`double`**-Array
- Struktur



Notizen

Inhalt

Notizen



Beispiel: Matrix

- Beispiel: 3×3-Einheitsmatrix erstellen

```
30  runIdentityMatrixExample
31  double[][] matrix = new double[3][]; // 3 Zeilen
33  for (int i = 0; i < matrix.length; i++)
34    matrix[i] = new double[3]; // 3 Spalten je Zeile
36  matrix[0][0] = 1;
37  matrix[1][1] = 1;
38  matrix[2][2] = 1;
40  printMatrix(matrix);
```

RectangularArrayExamples.java

```
1,00  0,00  0,00
0,00  1,00  0,00
0,00  0,00  1,00
```


Notizen

Beispiel: Matrix

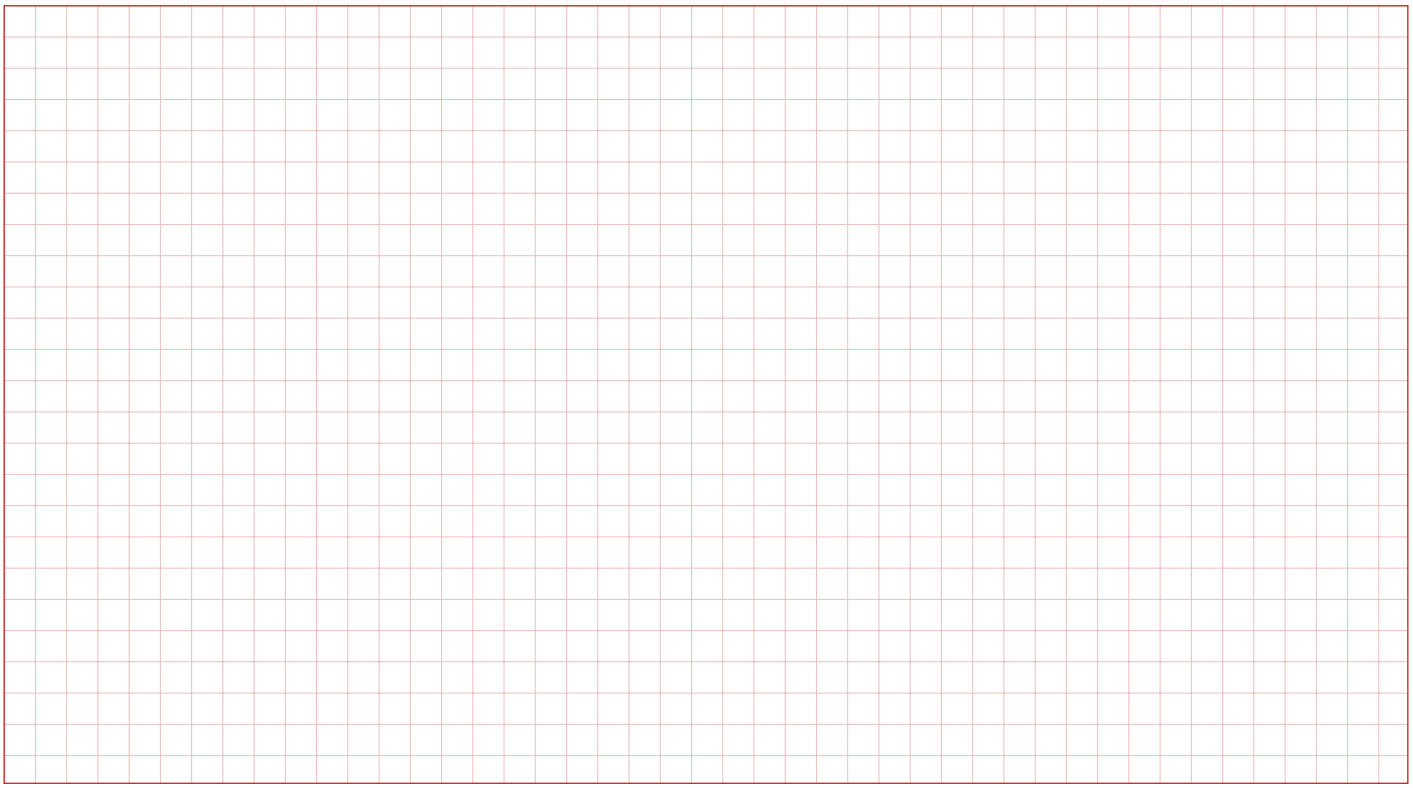
- Immer der gleiche umständliche Code

```
double[][] matrix = new double[3][]; // 3 Zeilen
for (int i = 0; i < matrix.length; i++)
  matrix[i] = new double[3]; // 3 Spalten je Zeile
```

- Äquivalent und kürzer („syntatic sugar“)

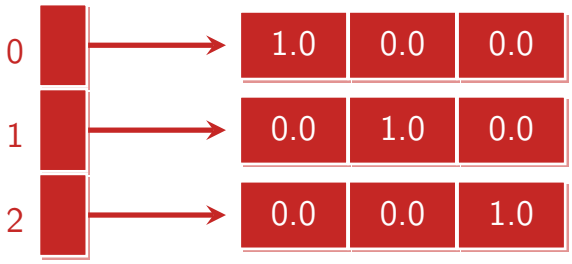
```
double[][] matrix = new double[3][3];
```


Notizen



Beispiel: Matrix

- Struktur



- Was ist `matrix[1]`?
- **`double`**-Array mit den Einträgen `{0,1,0}`!
- Was ist `matrix[1][2]`?
- Eintrag am Index `2` im Array `{0,1,0}` $\rightarrow$ `0`

Notizen

Allgemein: Allokieren und Zugriff

- Allokation

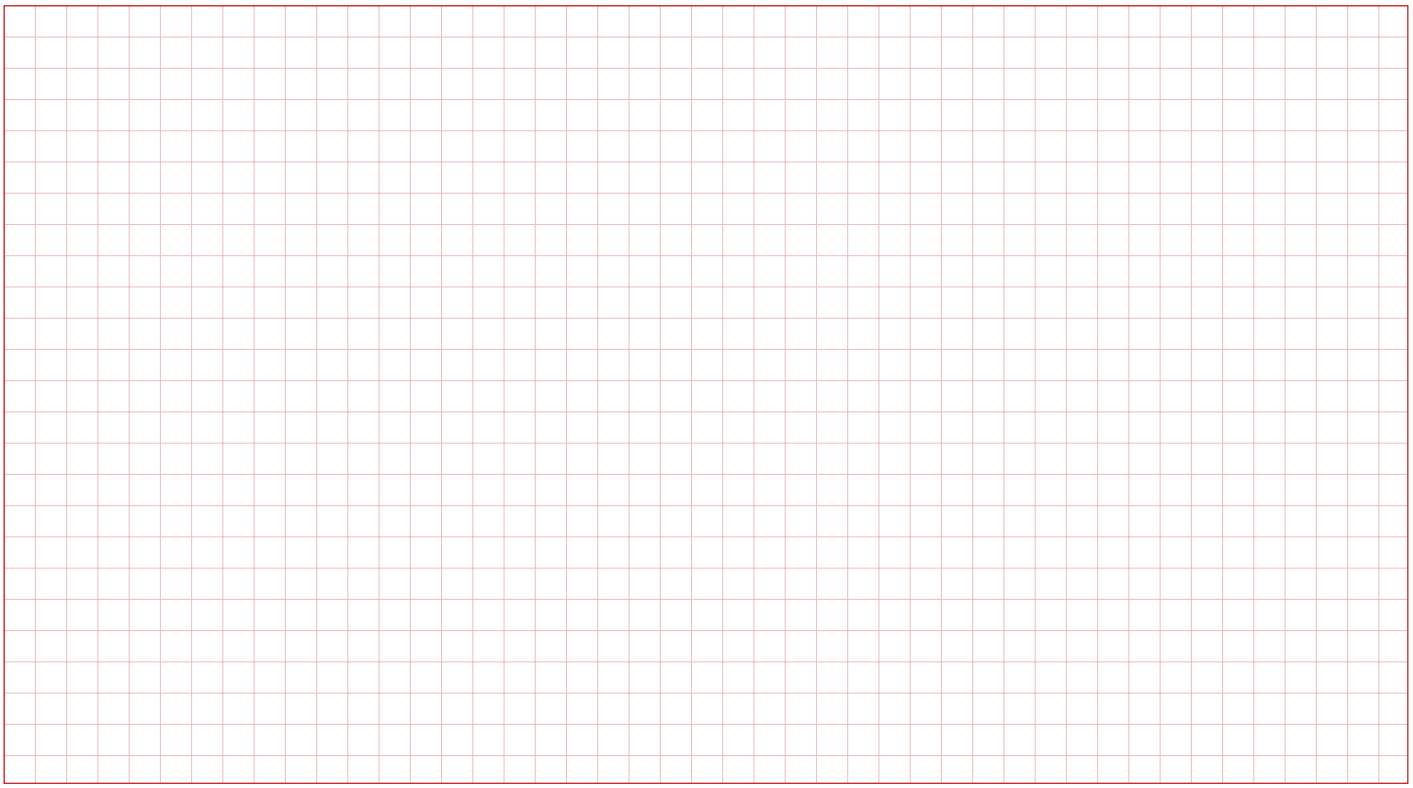
```
T[][] a = new T[m][n];
```

  Erstellt
  - Array der Länge `m` bestehend aus
  - Arrays der Länge `n` vom Typ `T`
- Zugriff

```
a[i][j]
```

  - Greife im `T`-Array mit Index `i`
  - auf das Element mit Index `j` zu
- Denkstütze
  - a entspricht der ganzen Matrix
  - `a[i]` entspricht der `i`-ten Zeile
  - `a[i][j]` entspricht dem `j`-ten Eintrag der `i`-ten Zeile

Notizen



Durchlaufen eines zweidimensionalen Arrays

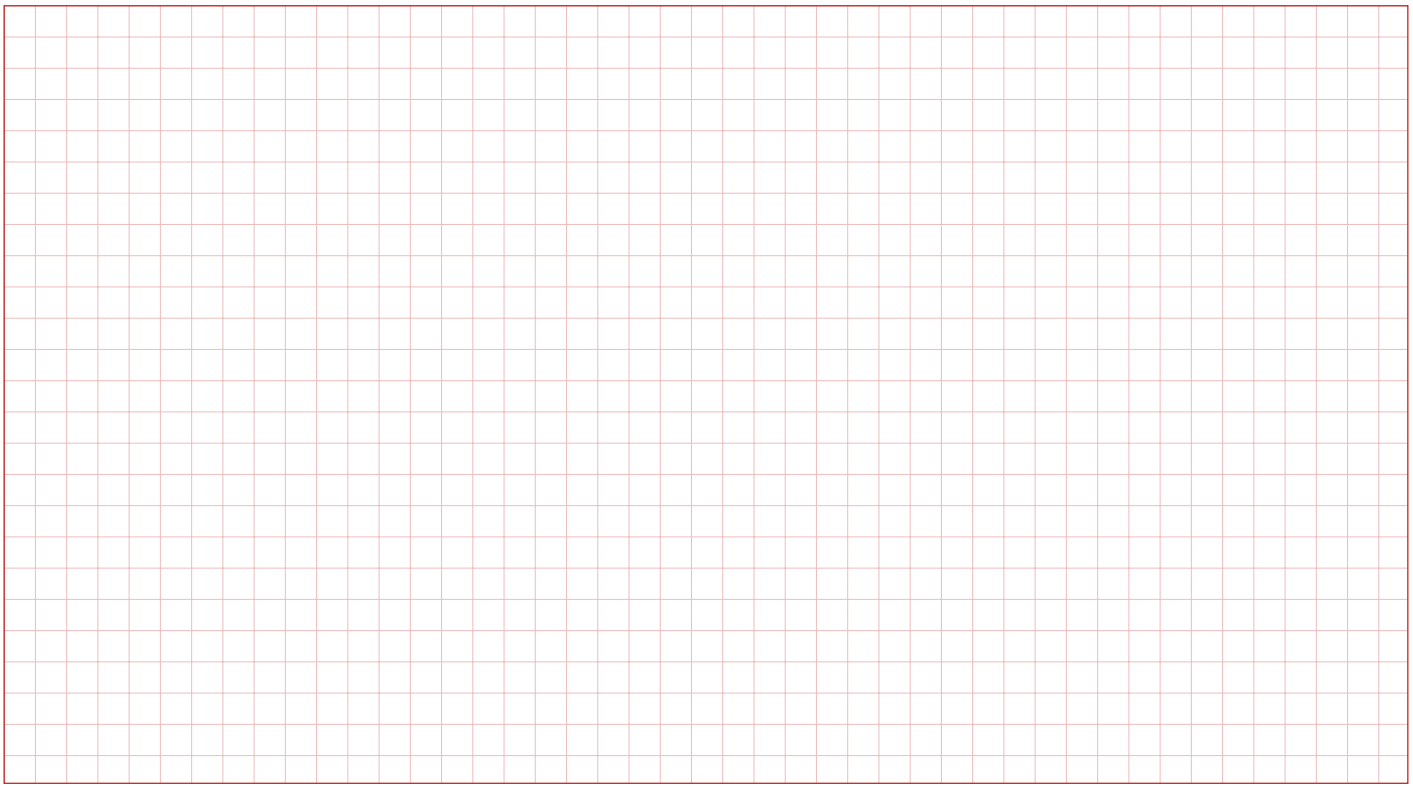
- Beispiel: `printMatrix`

```
6  private static void printMatrix(double[][] matrix){
7    for (int i = 0; i < matrix.length; i++){
8      for (int j = 0; j < matrix[i].length; j++){
9        System.out.printf("% 2.1f ", matrix[i][j]);
10     }
11     System.out.println();
12   }
13 }
```

RectangularArrayExamples.java

- Äußere Schleife: Durchläuft Zeilen der Matrix (`matrix.length` viele)
- Innere Schleife: Durchläuft Einträge der `i`-ten Zeile (`matrix[i].length` viele)
- `printf` gibt Eintrag in Zeile `i` und Spalte `j` aus (`matrix[i][j]`)

Notizen



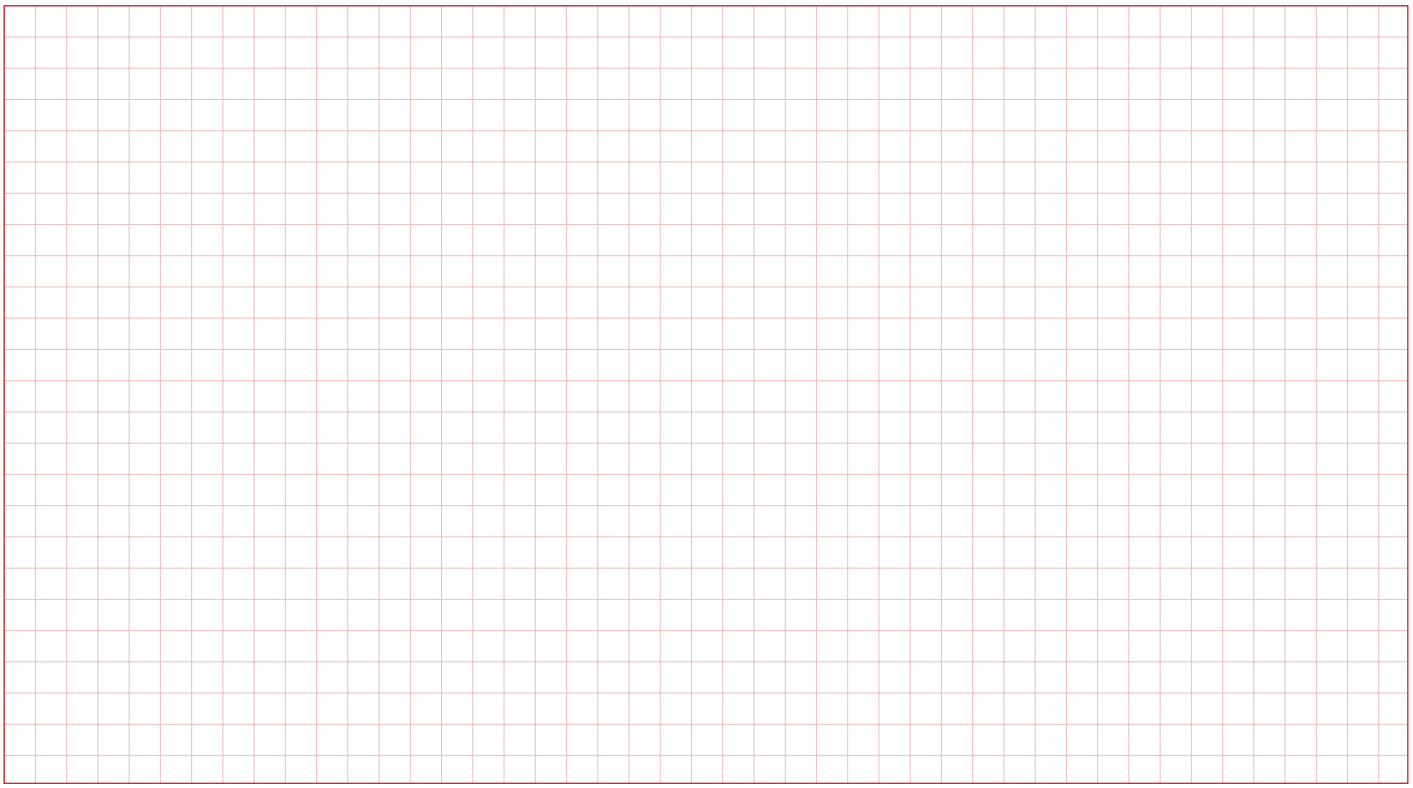
Durchlaufen eines zweidimensionalen Arrays

- Funktioniert auch mit **for**-each-Schleife

```
17 private static void printMatrixForEach(double[][] matrix){
18   for (double[] row : matrix){
19     for (double entry : row){
20       System.out.printf("% 2.1f ", entry);
21     }
22     System.out.println();
23   }
24 }
```

RectangularArrayExamples.java

Notizen



Zweidimensionale Array-Literale

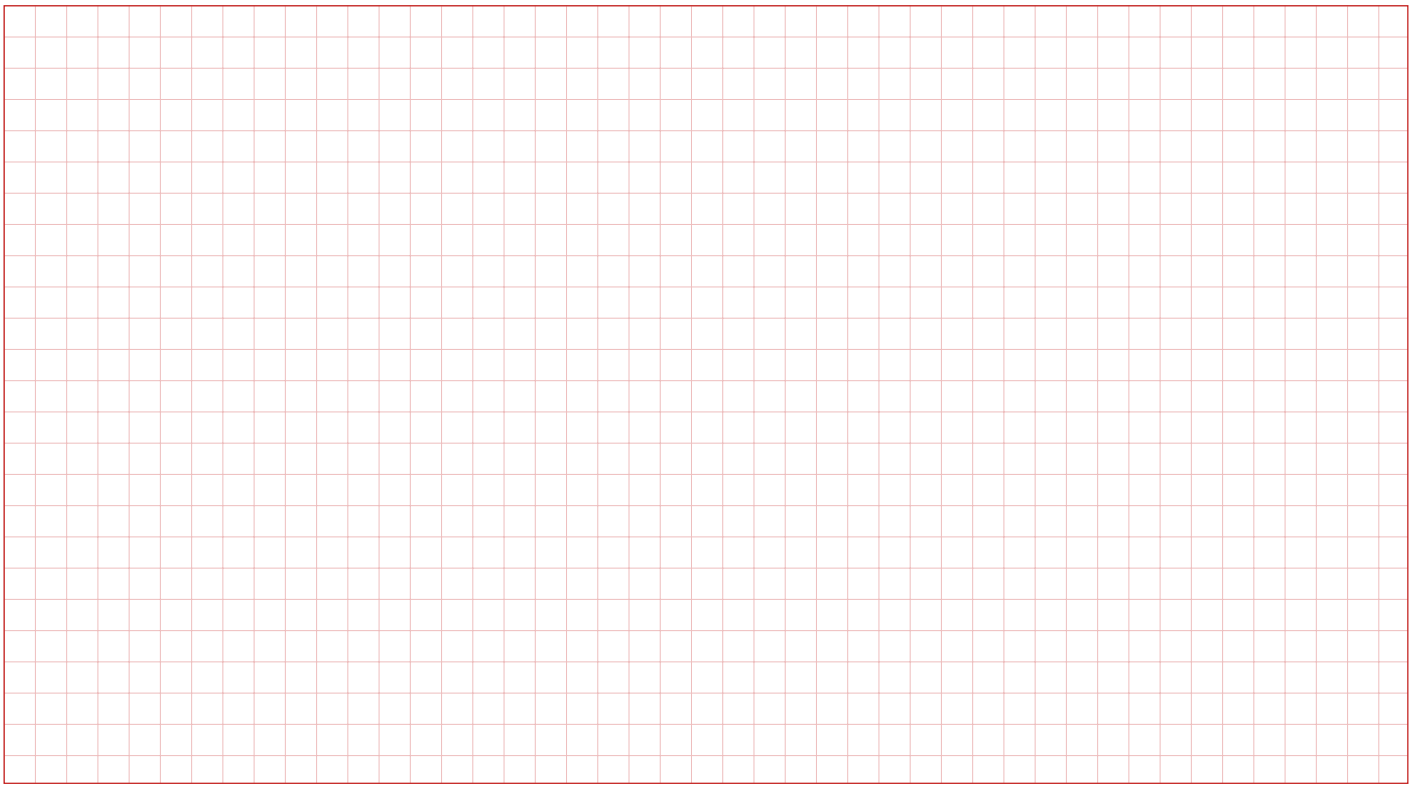
- Array-Literale gibt es auch für zweidimensionale Arrays

```
46 runIdentityMatrixLiteralExample
47 double[][] matrix = new double[][]
48 {
49     {1,0,0}, // Zeile 0
50     {0,1,0}, // Zeile 1
51     {0,0,1} // Zeile 2
52 };
54 printMatrix(matrix);
```

RectangularArrayExamples.java

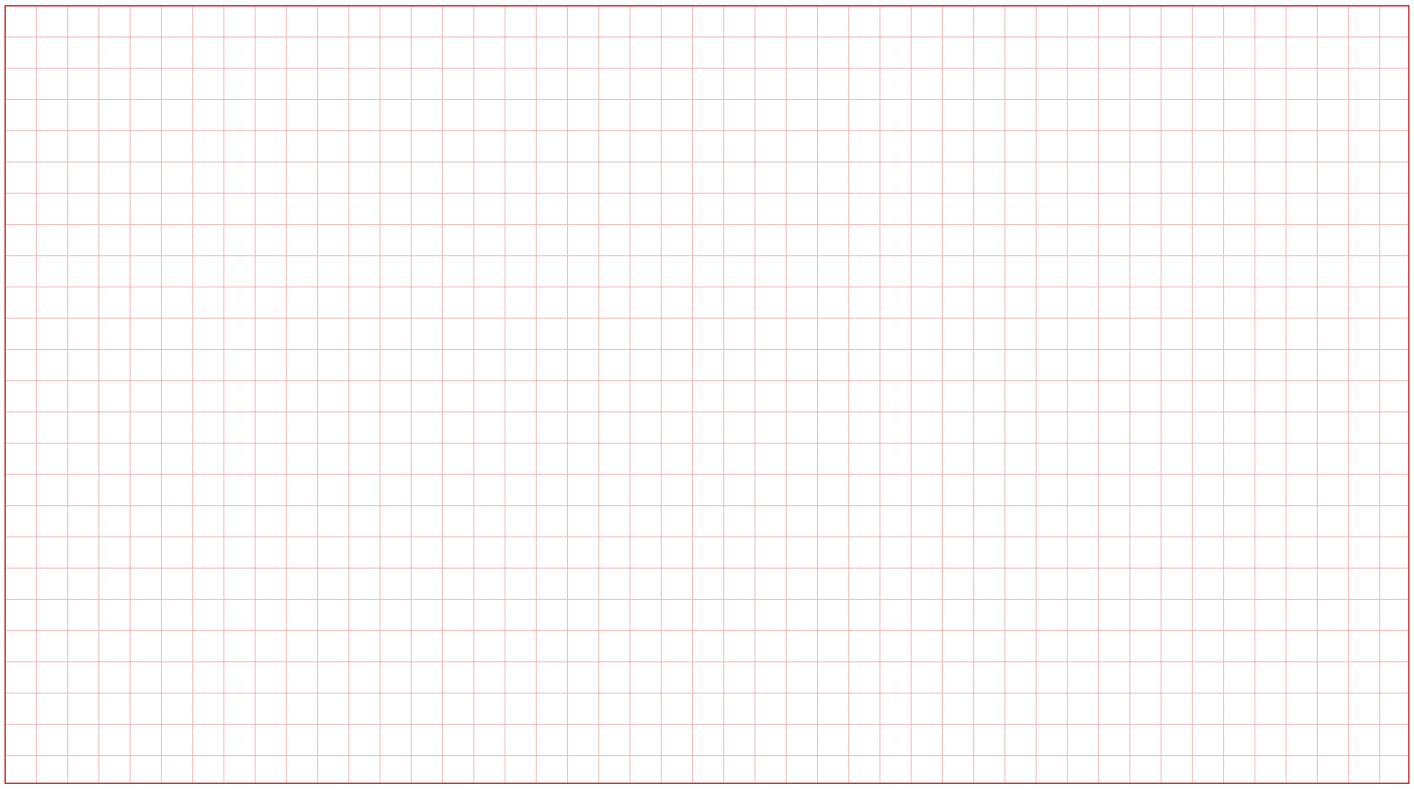
- Geschachtelte Arrays werden wieder über Array-Literal definiert
- Anzahl der Einträge implizit durch Literal gegeben
- Formatierung ist optional...
- macht Code aber übersichtlich

Notizen



Inhalt

Notizen



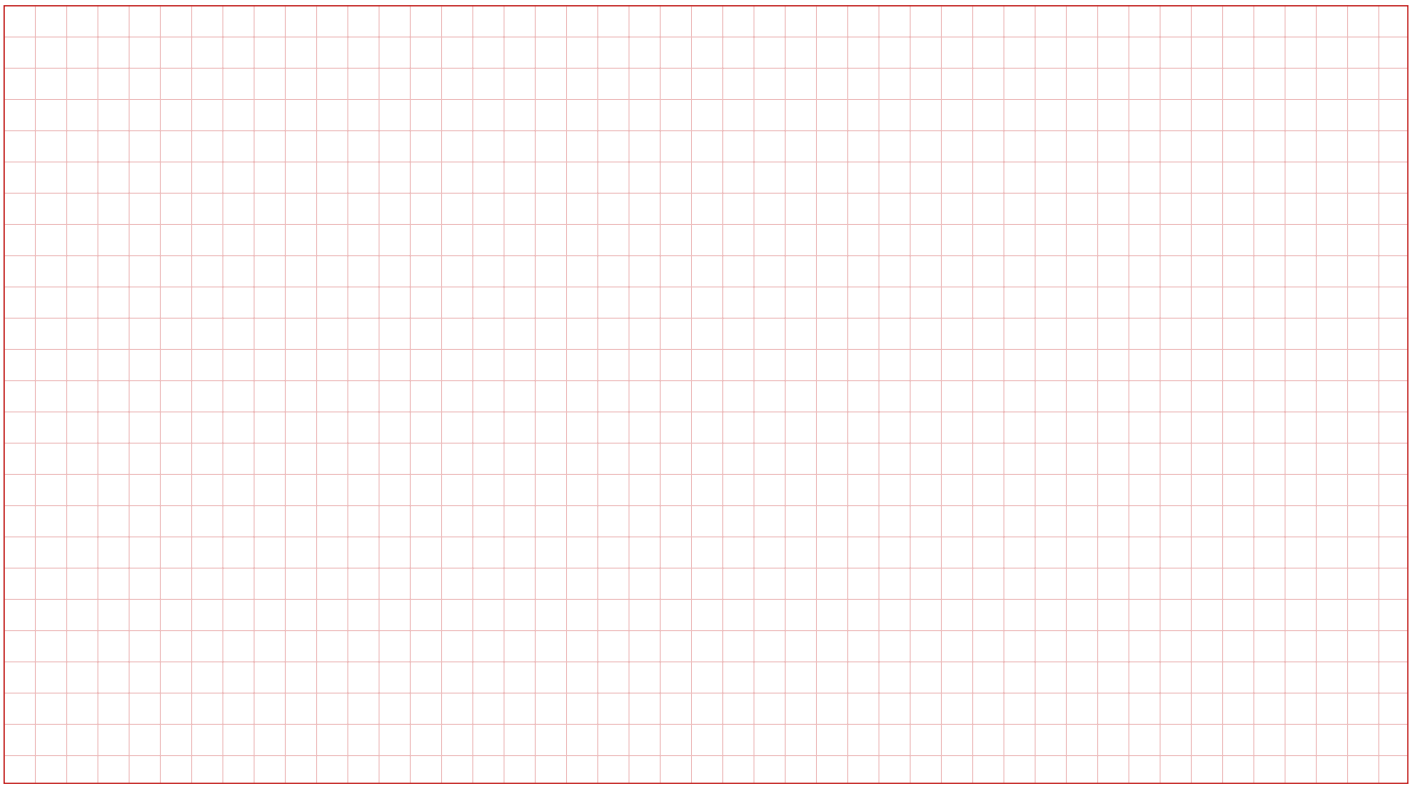
Mehrdimensionale Arrays

- Was mit zwei Dimensionen funktioniert...
- funktioniert auch mit mehr Dimensionen

```
double[][][] threeDim = new double[3][4][5];
```

- Prinzip bleibt das Gleiche (nur eine Dimension mehr)
- Anschauung: Quaderförmige Matrix
  - Breite: 3
  - Höhe: 4
  - Tiefe: 5
- Zugriff über `threeDim[i][j][k]`
  - Zeile `i`
  - Spalte `j`
  - Tiefe `k`

Notizen



Mehrdimensionale Arrays: Beispiel

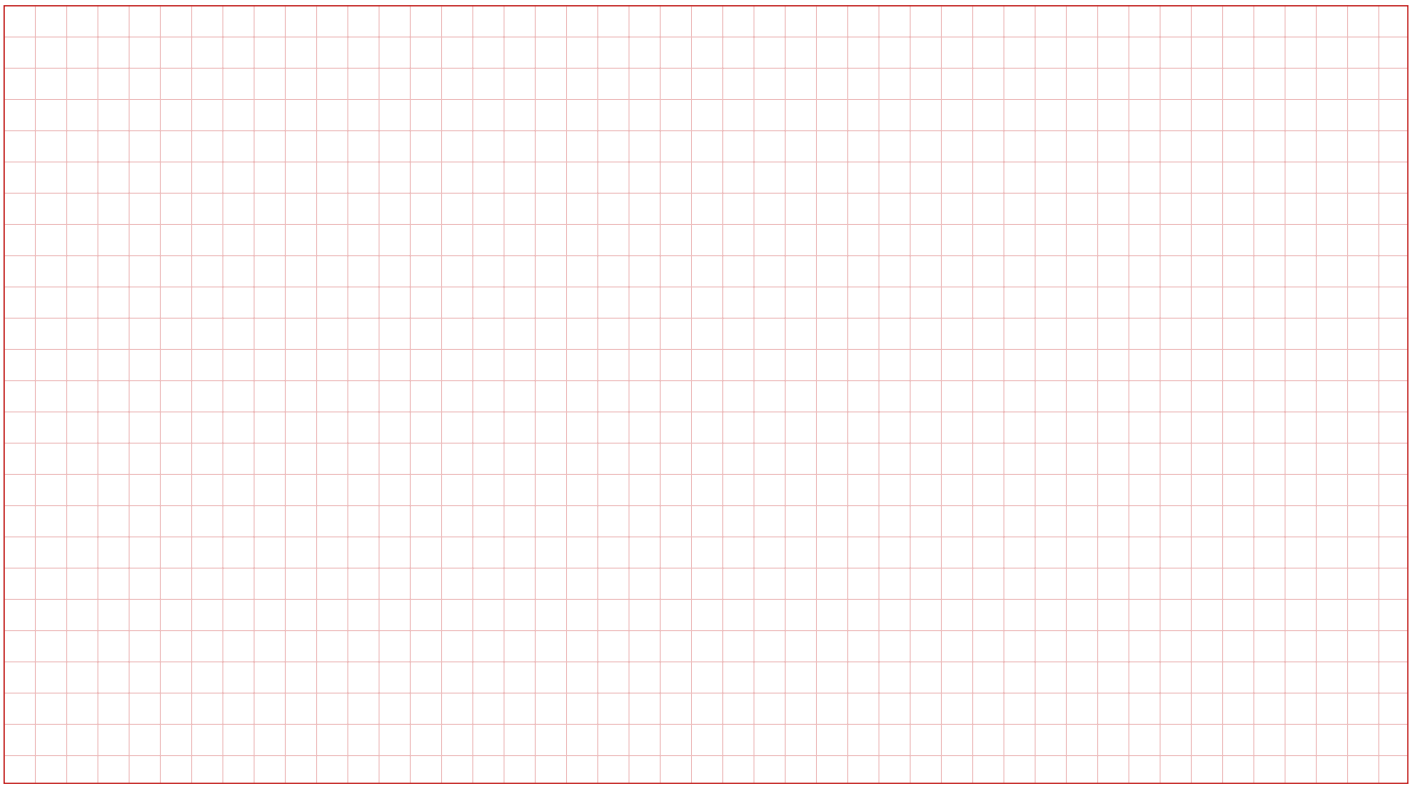
runThreeDimArrayExample

```
72 int[][][] threeDim = new int[3][4][5];
73
75 for (int i = 0; i < threeDim.length; i++)
76   for (int j = 0; j < threeDim[i].length; j++)
77     for (int k = 0; k < threeDim[i][j].length; k++)
78       threeDim[i][j][k] = i + j + k;
80 printMatrix3D(threeDim);
```

RectangularArrayExamples.java

```
m[0][0][0] = 0
m[0][0][1] = 1
...
m[2][3][3] = 8
m[2][3][4] = 9
```


Notizen



Mehrdimensionale Arrays: Hinweise

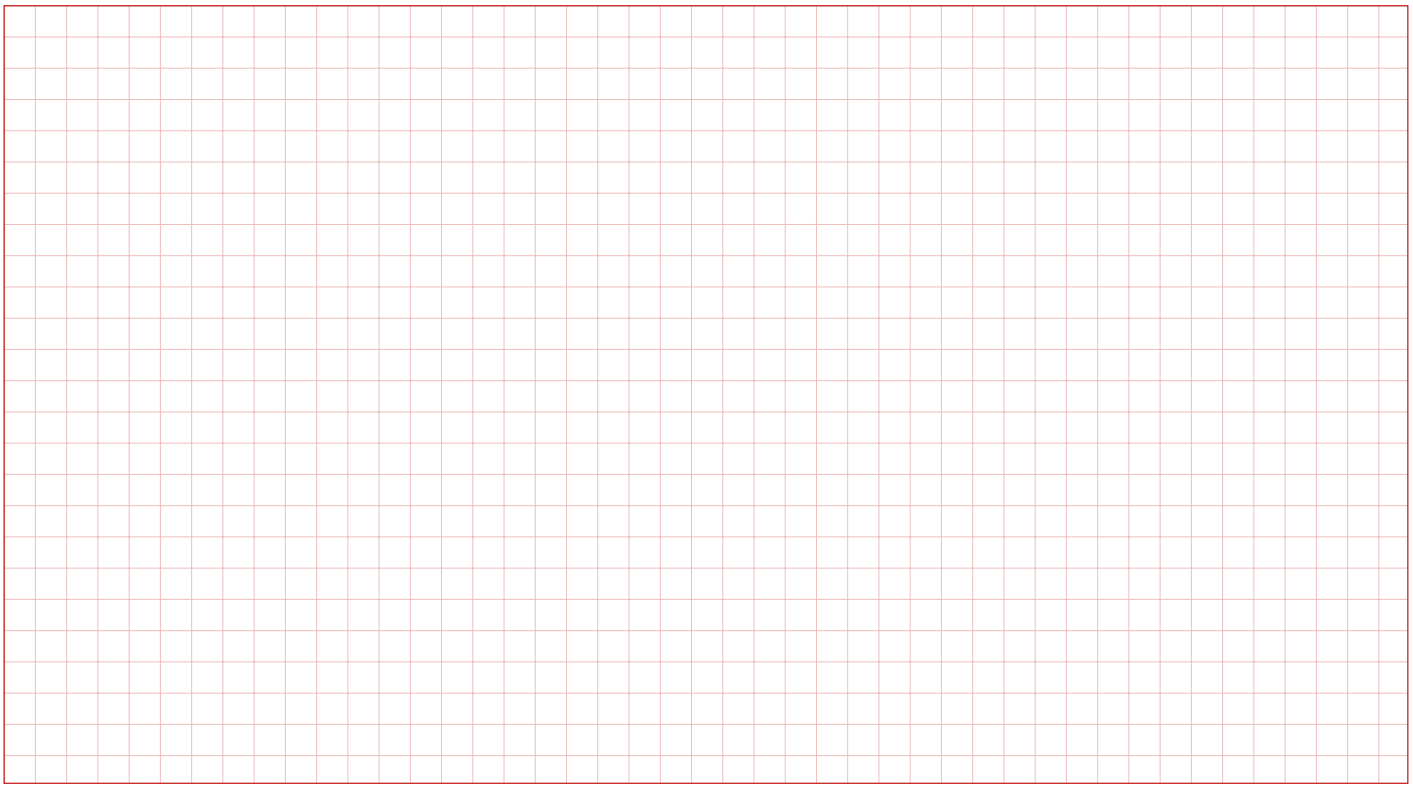
- Mehr Dimensionen möglich

```
double[][][][] fourDim = new double[4][5][6][7];
```

  - Aber irgendwann wird es unübersichtlich
  - Andere Datenstrukturen eventuell geeigneter
- Array-Literale funktionieren auch hier (aber unübersichtlich)
- Referenztypen
  - Hier nur mehrdimensionale Arrays auf primitiven Typen
  - Prinzipiell natürlich auch Referenztypen möglich

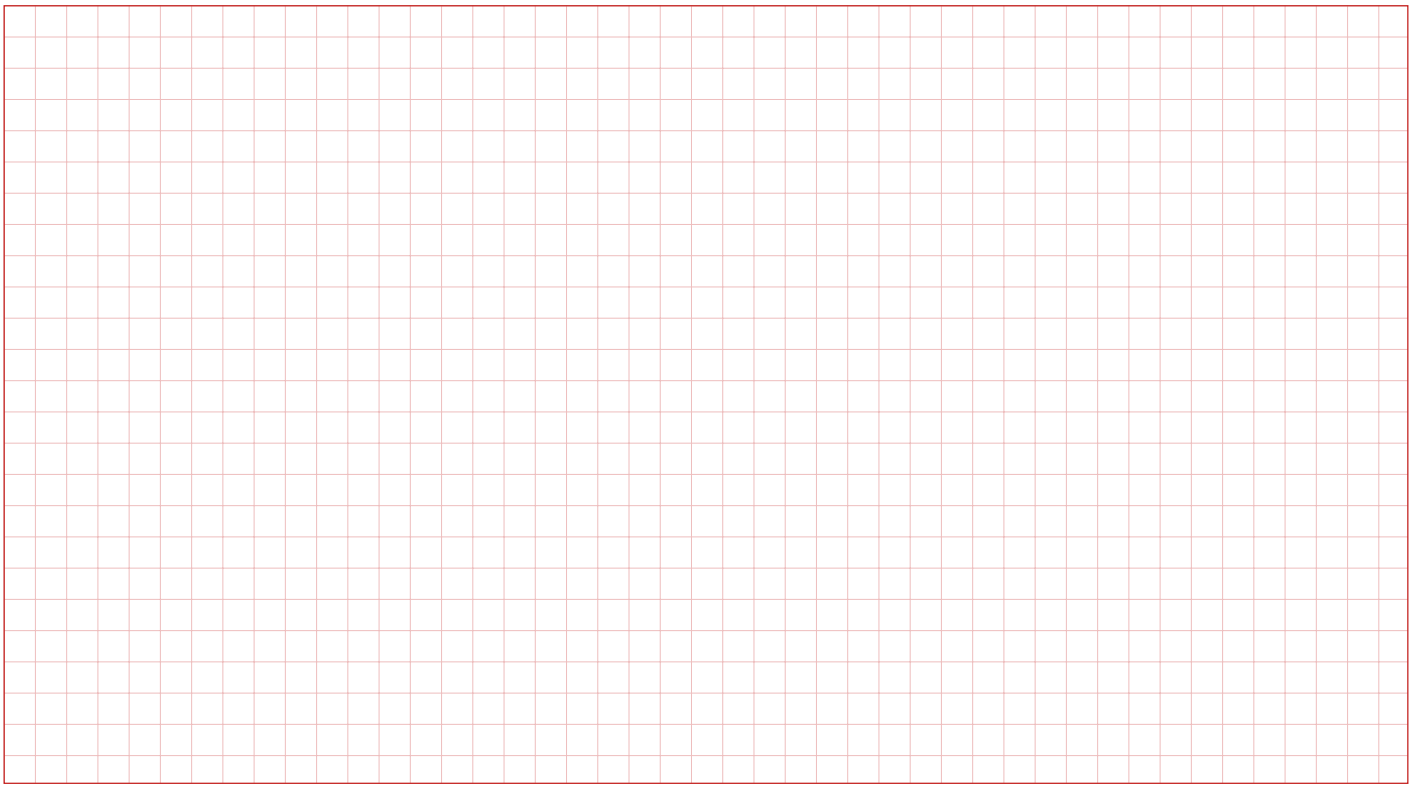
```
Point2D[][][] points = new Point2D[10][10][10];
```


Notizen



Inhalt

Notizen



Nicht-Rechteckige Arrays

- Zur Erinnerung: Folgende zwei Code-Schnipsel sind äquivalent

```
double[][] matrix = new double[3][3];
```

```
double[][] matrix = new double[3][];
for (int i = 0; i < matrix.length; i++)
  matrix[i] = new double[3];
```

- Was passiert in folgendem Code? Nicht-Rechteckiger Array!

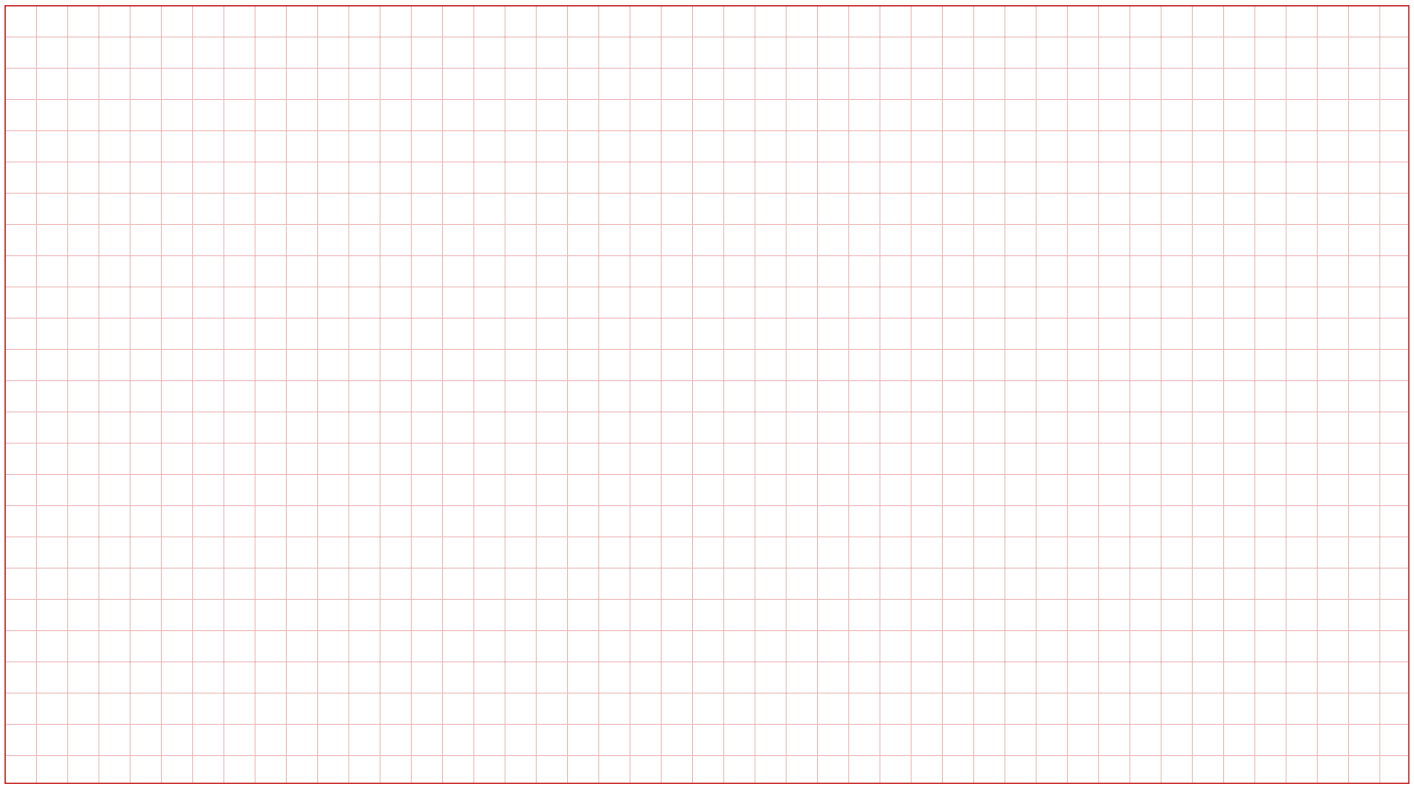
```
double[][] triangle = new double[3][];
for (int i = 0; i < triangle.length; i++)
  triangle[i] = new double[i+1];
```

Notizen

Nicht-Rechteckige Arrays: Beispiele

Array	Anwendungsbeispiel
`double[][] m`	Messwerte `m[i][j]`: `j`-ter Messwert am Tag `i`
`char[][] l`	Zeilen `l[i][j]`: `j`-tes Zeichen in Zeile `i`
`String[][] t`	Wörter `t[i][j]`: `j`-tes Wort in Zeile `i`
`Point2D[][] p`	Polygone `p[i][j]`: `j`-ter Punkt von Polygon `i`

Notizen



Nicht-Rechteckige Arrays: Polygone

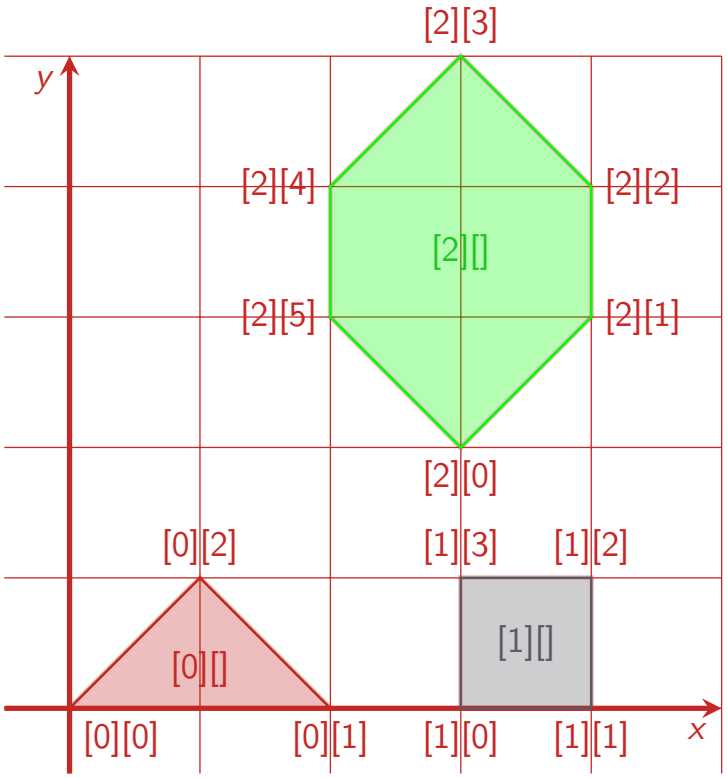
- Beispiel: Drei Polygone durch ihre Eckpunkte gegeben

```
 9  Point2D[][] polygons = new Point2D[3][];
11  polygons[0] = new Point2D[] { // three points
12    new Point2D(0,0), new Point2D(2,0), new Point2D(1,1)
13  };
15  polygons[1] = new Point2D[] { // four points
16    new Point2D(3,0), new Point2D(4,0),
17    new Point2D(4,1), new Point2D(3,1)
18  };
20  polygons[2] = new Point2D[] { // six points
21    new Point2D(3,2), new Point2D(4,3), new Point2D(4,4),
22    new Point2D(3,5), new Point2D(2,4), new Point2D(2,3)
23  };
```

NonRectangularArrayExamples.java

Notizen

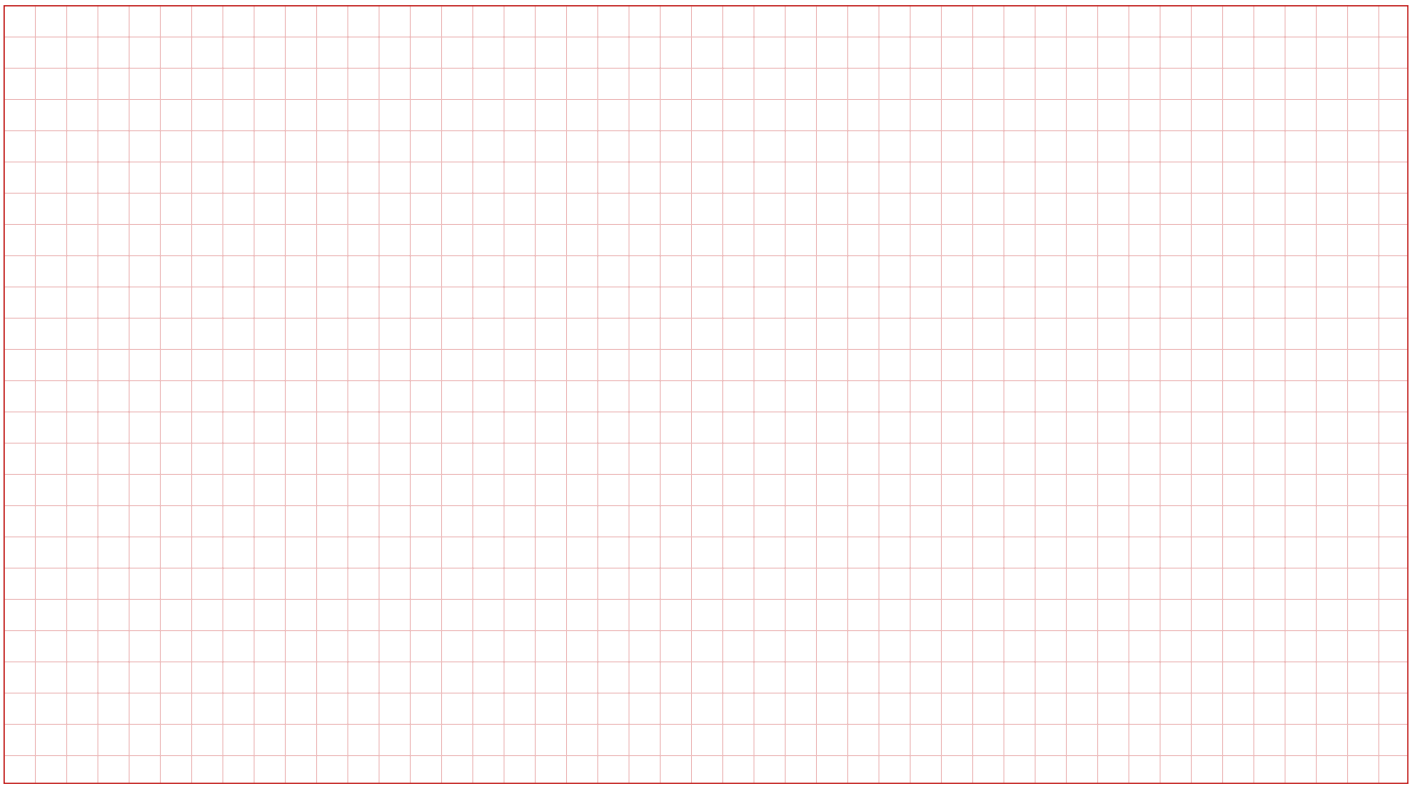
Nicht-Rechteckige Arrays: Polygone



Notizen

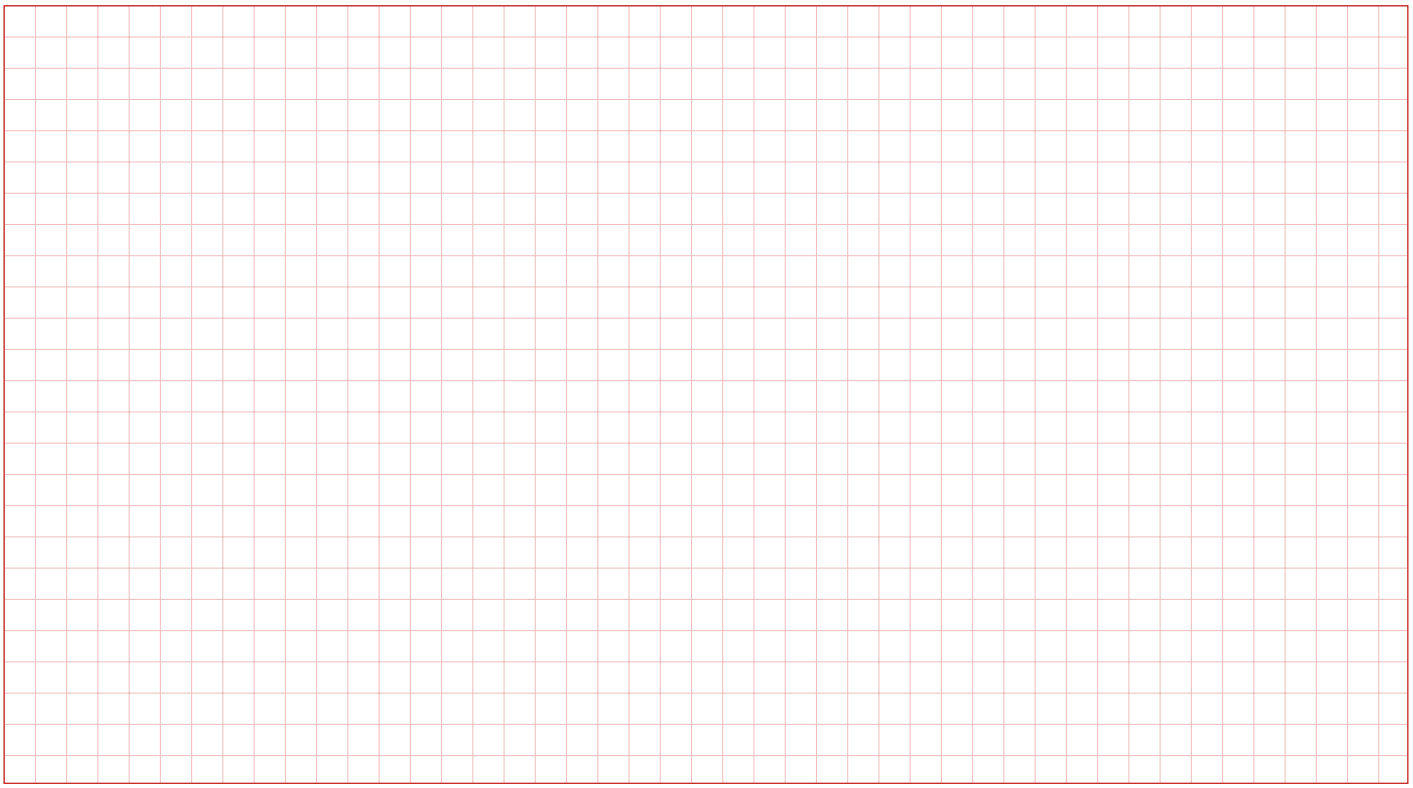
Inhalt

Notizen



Inhalt

Notizen



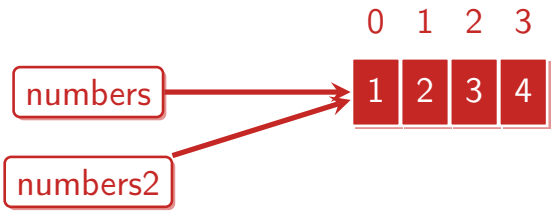
„Kopieren“ durch Wertzuweisung

- Zur Erinnerung: Array ist ein Referenztyp
- Zuweisung kopiert nur Referenz und nicht Inhalt

```
58 // snippet: runArrayAssignmentExample
59 int[] numbers = new int[] {1,2,3,4};
60 int[] numbers2 = numbers;
```

ArrayCopyExamples.java

```
numbers == numbers2? true
Eintraege identisch? true
```



- Wie kopieren wir einen Array?

Notizen

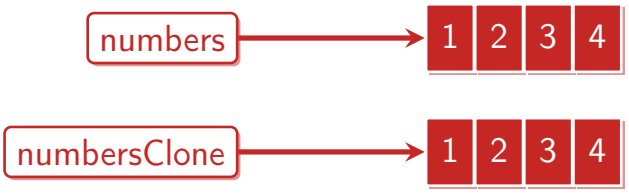
Kopieren: Zu Fuß

- Kopieren über `for`-Schleife

```
67  runArrayCopyManualExample
68  int[] numbers = new int[] {1,2,3,4};
70  int[] numbersClone = new int[numbers.length];
71  for (int i = 0; i < numbers.length; i++)
72    numbersClone[i] = numbers[i];
```

ArrayCopyExamples.java

```
numbers == numbersClone? false
Eintraege identisch? true
```



Notizen

Kopieren zu Fuß: Kochrezept

- Kochrezept: Kopieren von Array `T[] a`
  1. Array gleichen Typs/gleicher Länge anlegen

```
T[] clone = new T[a.length];
```

  2. Kopieren über **for**-Schleife

```
for (int i = 0; i < a.length; i++)
  clone[i] = a[i];
```

- Hinweis: Einträge werden über Wertzuweisung kopiert!
- Äquivalente Alternative: `clone`

runArrayCopyCloneExample

```
79 int[] numbers = new int[] {1,2,3,4};
80 int[] numbersClone = numbers.clone();
81
```

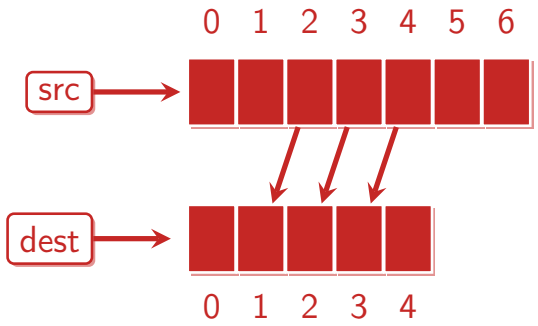
ArrayCopyExamples.java

Notizen

Kopieren über System.arraycopy

```
System.arraycopy(
  T[] src, int srcPos,
  T[] dest, int destPos, int length)
```

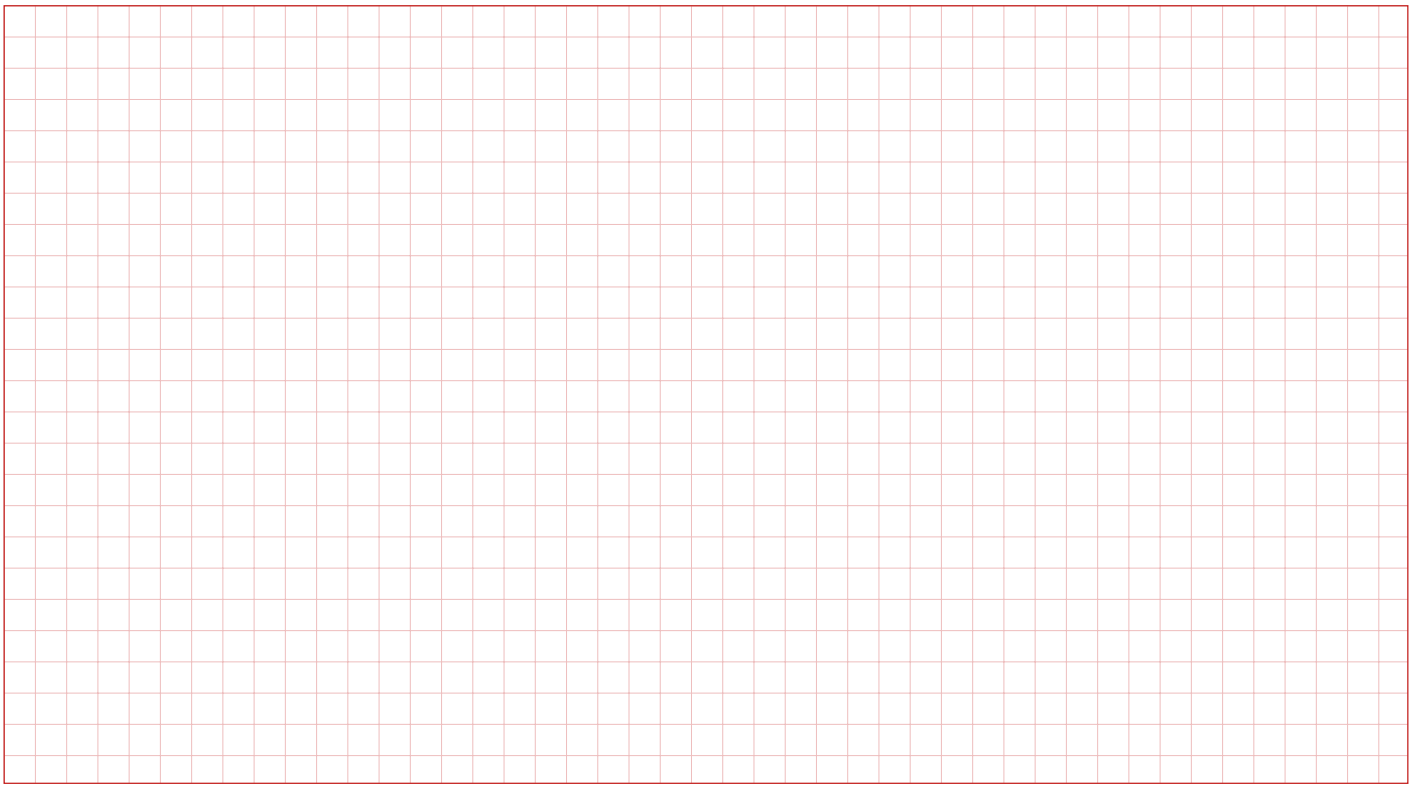
- Kopiert über Wertzuweisung
  - `length` viele Elemente
  - in `src` ab Element `srcPos`
  - nach `dest` ab Element `destPos`
- Veranschaulichung: `System.arraycopy(src, 2, dest, 1, 3);`



Notizen

Inhalt

Notizen



Tiefe Kopien von Arrays

- Bisher: Flache Kopien
- Was passiert wenn Einträge Referenzen sind?

```
88 runArrayFlatCopyExample
89 Point2D[] triangle = new Point2D[]{
90   new Point2D(0,0), new Point2D(2,0), new Point2D(1,1)
91 };
93 Point2D[] triangle2 = triangle.clone();
```

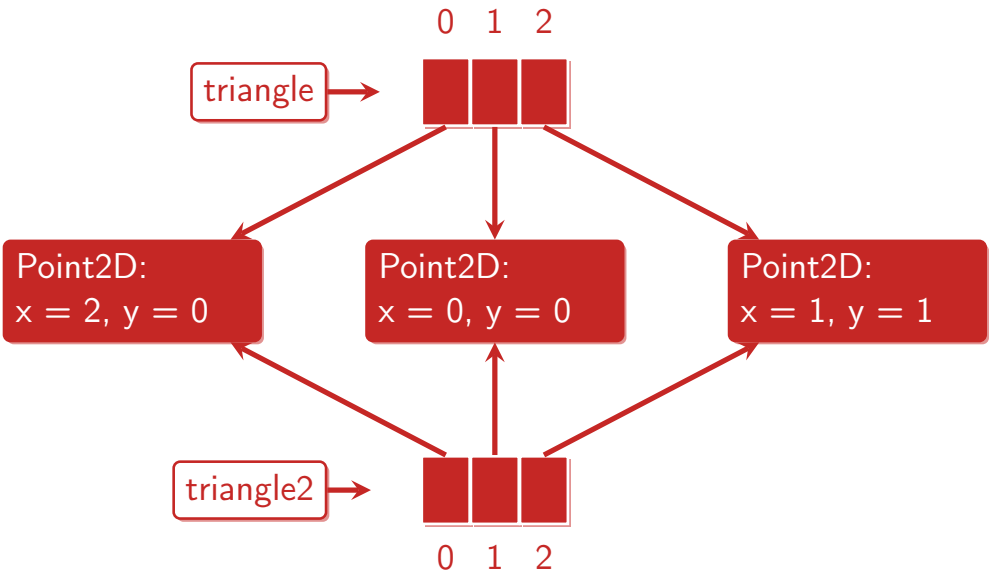
ArrayCopyExamples.java

```
triangle == triangle2? false
Eintraege identisch? true
Eintraege wertgleich? true
```


Notizen

Tiefe Kopien von Arrays

- Veranschaulichung des vorherigen Beispiels



Notizen

Tiefe Kopien von Arrays

- Tiefe Kopie über Kopier-Konstruktor

```
100 runArrayDeepCopyManualExample
101 Point2D[] triangle = new Point2D[]{
102   new Point2D(0,0), new Point2D(2,0), new Point2D(1,1)
103 };
105 Point2D[] triangle2 = new Point2D[triangle.length];
107 for (int i = 0; i < triangle.length; i++)
108   triangle2[i] = new Point2D(triangle[i]);
```

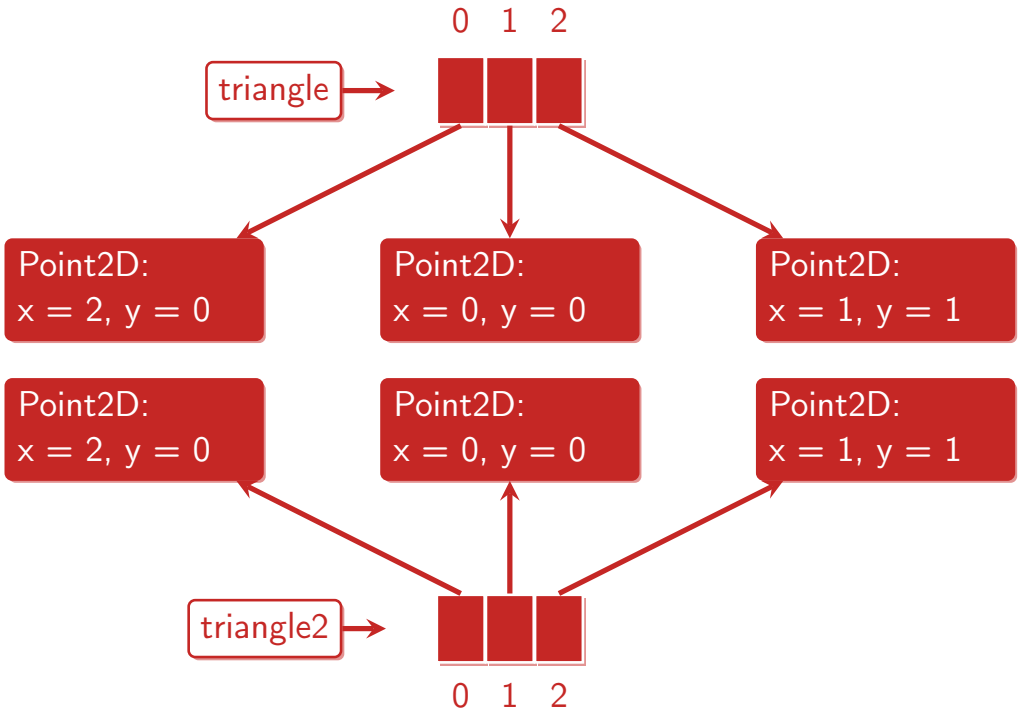
ArrayCopyExamples.java

```
triangle == triangle2? false
Eintraege identisch? false
Eintraege wertgleich? true
```


Notizen

Tiefe Kopien von Arrays

- Veranschaulichung des vorherigen Beispiels



Notizen

Tiefe Kopien von Arrays

- Alternative wenn Klasse `clone`-Methode implementiert

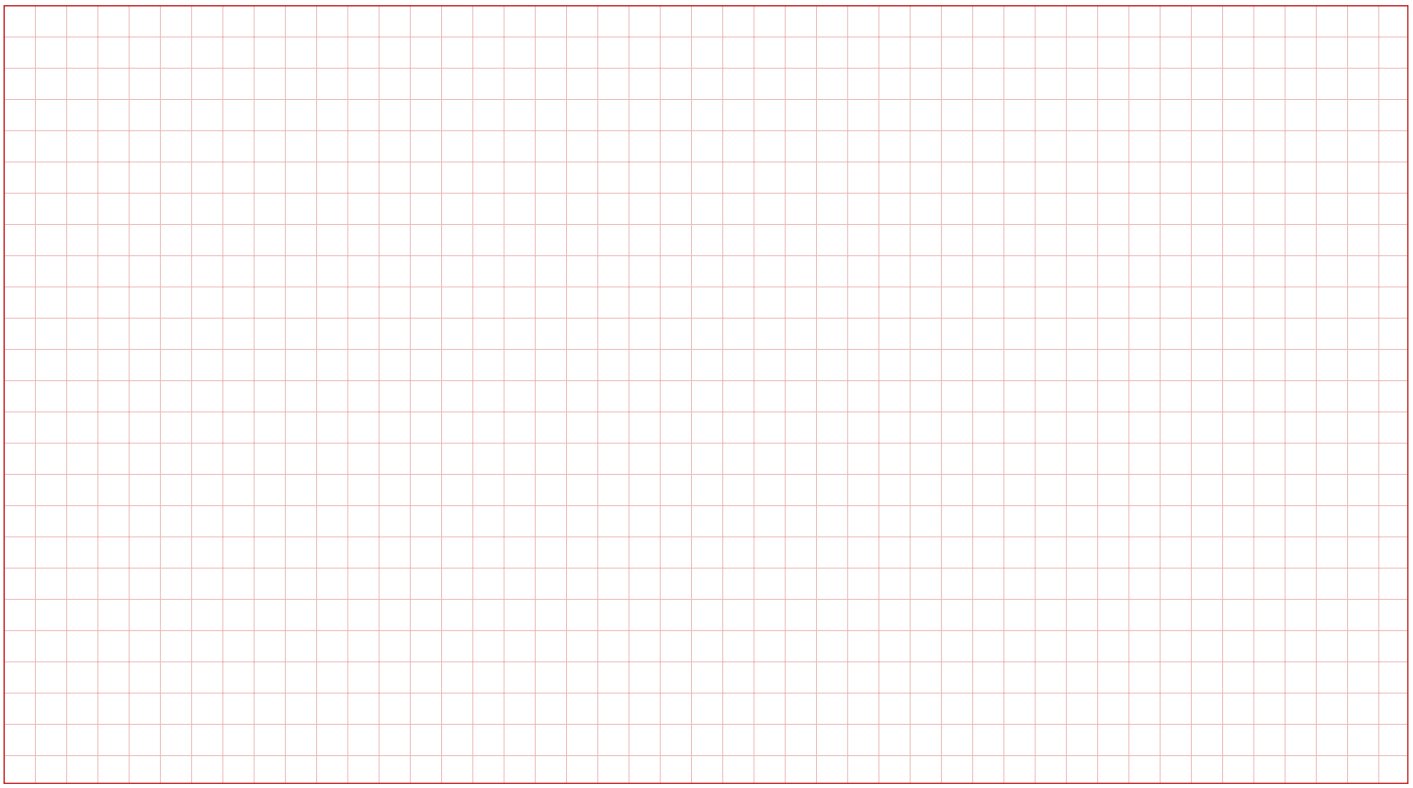
```
115 runArrayDeepCopyCloneExample
116 Point2D[] triangle = new Point2D[]{
117   new Point2D(0,0), new Point2D(2,0), new Point2D(1,1)
118 };
120 Point2D[] triangle2 = new Point2D[triangle.length];
122 for (int i = 0; i < triangle.length; i++)
123   triangle2[i] = triangle[i].clone();
```

ArrayCopyExamples.java

```
triangle == triangle2? false
Eintraege identisch? false
Eintraege wertgleich? true
```

- Führt zu gleichem Ergebnis

Notizen



(Tiefe) Kopien von mehrdimensionalen Arrays

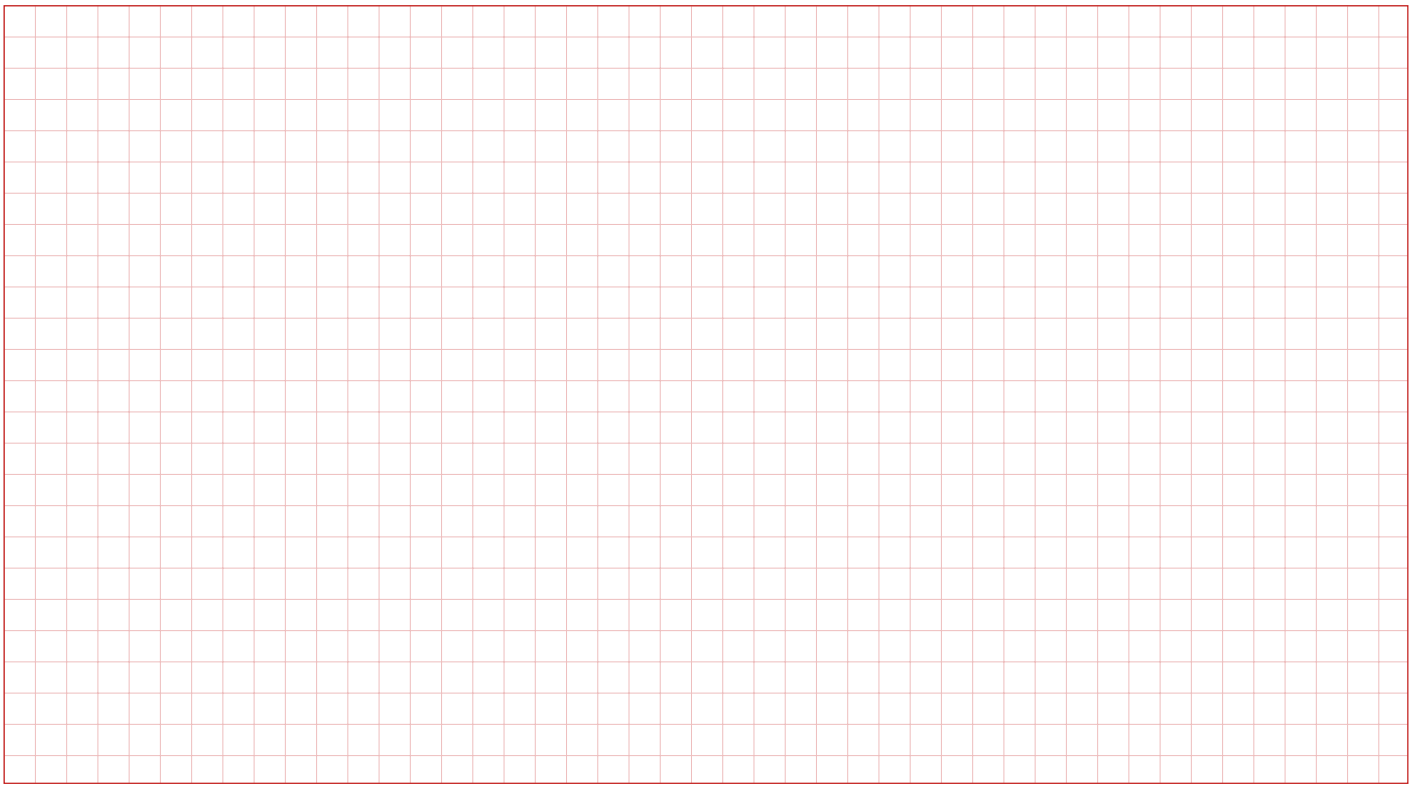
- Zur Erinnerung: Beispiel mit drei Polygonen in mehrdimensionalen und nicht-rechteckigem Array `Point2D[][] polygons`
- Tiefe Kopie von `polygons` erstellen

```
144 Point2D[][] clone = new Point2D[polygons.length][];
146 for (int i = 0; i < polygons.length; i++){
147   clone[i] = new Point2D[polygons[i].length];
149   for (int j = 0; j < polygons[i].length; j++){
150     clone[i][j] = polygons[i][j].clone();
151   }
153 }
```

ArrayCopyExamples.java

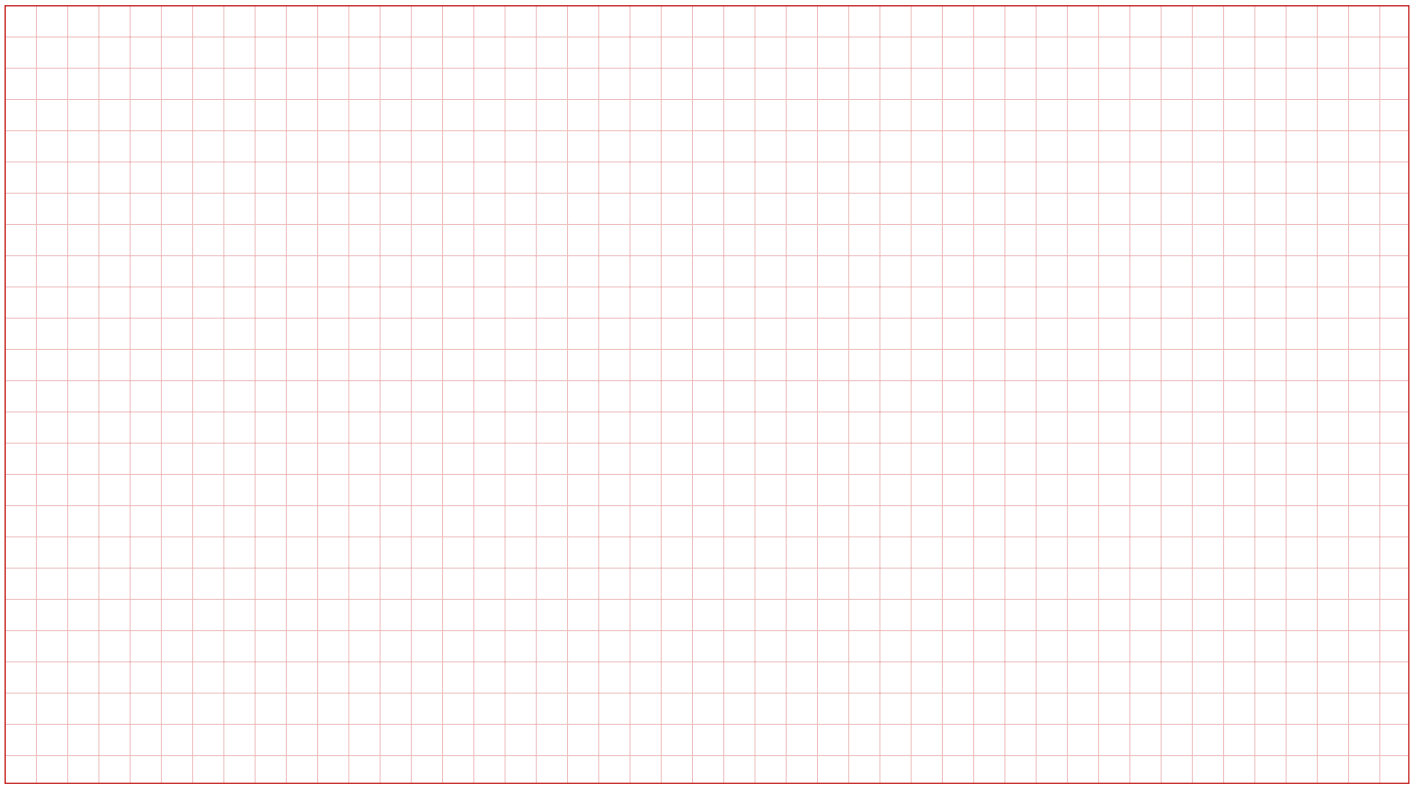
- Prinzip: Mischung aus Erstellung von Array und tiefer Kopie

Notizen



Inhalt

Notizen



Identität von zwei Arrays

- Array ist Referenztyp
- Identität zweier Referenzvariablen heißt, die zeigen auf dieselbe Instanz

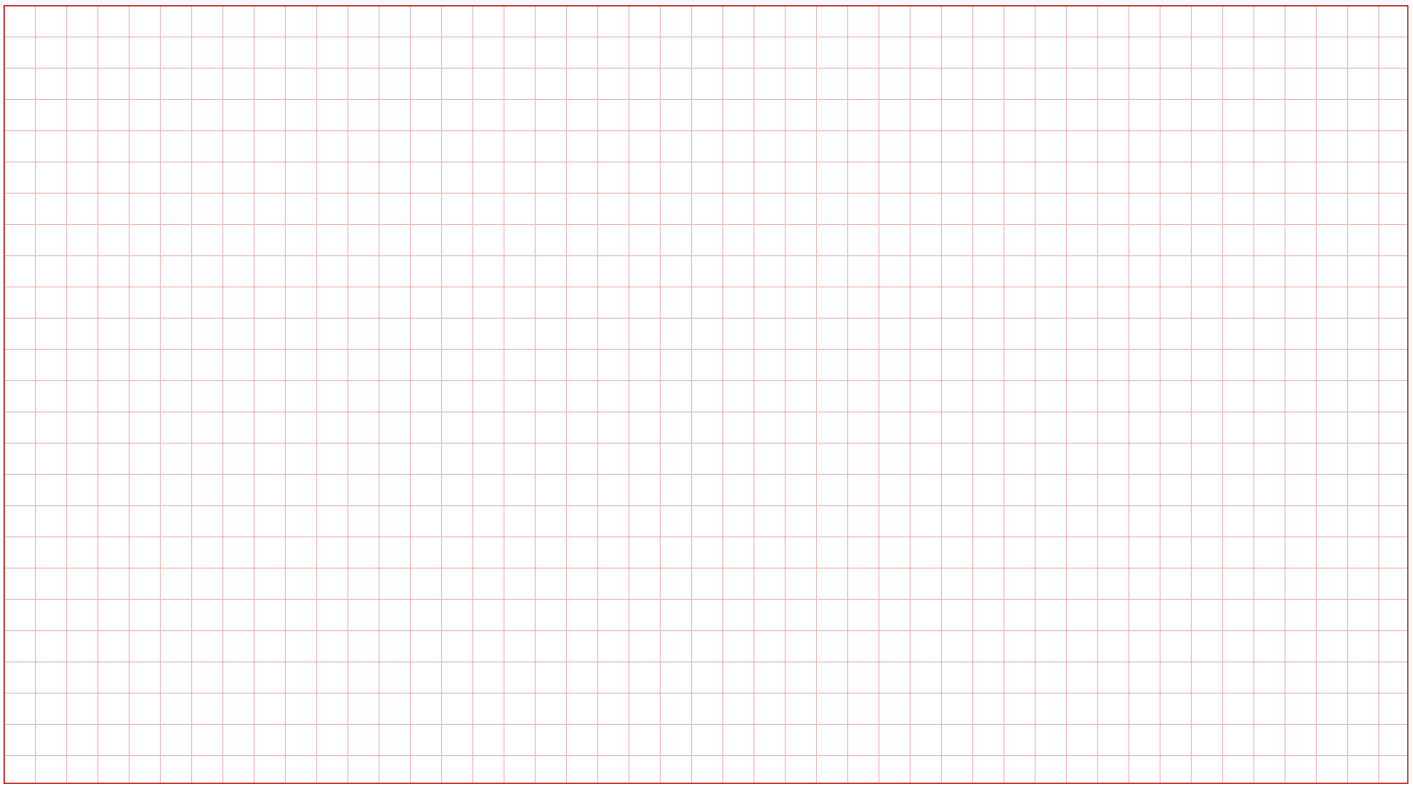
```
14 runArrayIdentityExample
15 int[] numbers = new int[] {1,2,3,4};
16 int[] numbers2 = numbers;
18 System.out.printf("numbers == numbers2 : %b%n",
19     numbers == numbers2);
```

ArrayEqualsExamples.java

```
numbers == numbers2 : true
```

- Entspricht: Wertzuweisung der Referenzvariablen bei Kopieren

Notizen



equals-Methode von Array

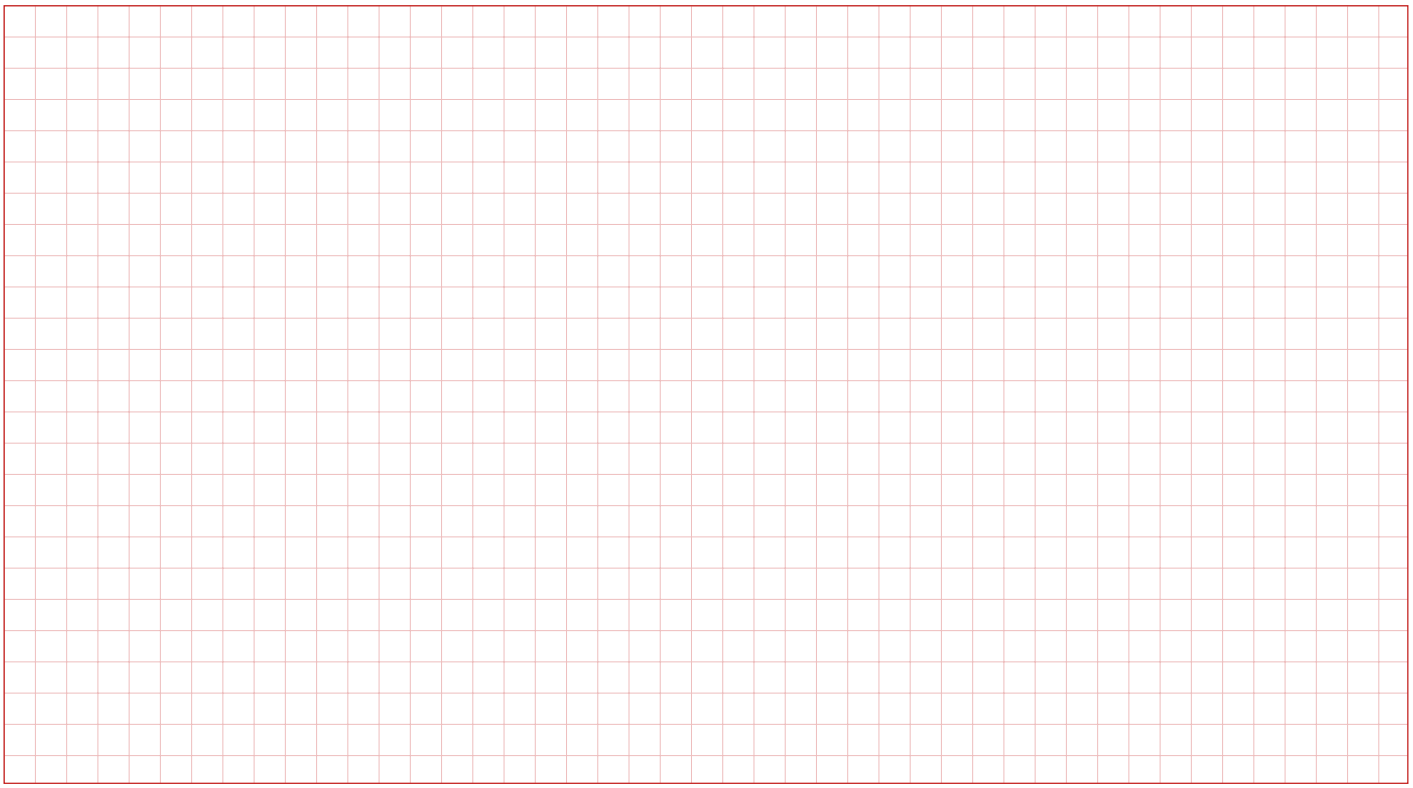
- Jeder Array implementiert equals-Methode
- Aber: Diese prüft nur Identität (äquivalent zu oben)

```
25 runArrayEqualsExample
26 int[] numbers = new int[] {1,2,3,4};
27 int[] numbers2 = numbers;
28 int[] numbersClone = numbers.clone();
30 System.out.printf("numbers.equals(numbers2): %b%n",
31     numbers.equals(numbers2));
33 System.out.printf("numbers.equals(numbersClone): %b%n",
34     numbers.equals(numbersClone));
```

ArrayEqualsExamples.java

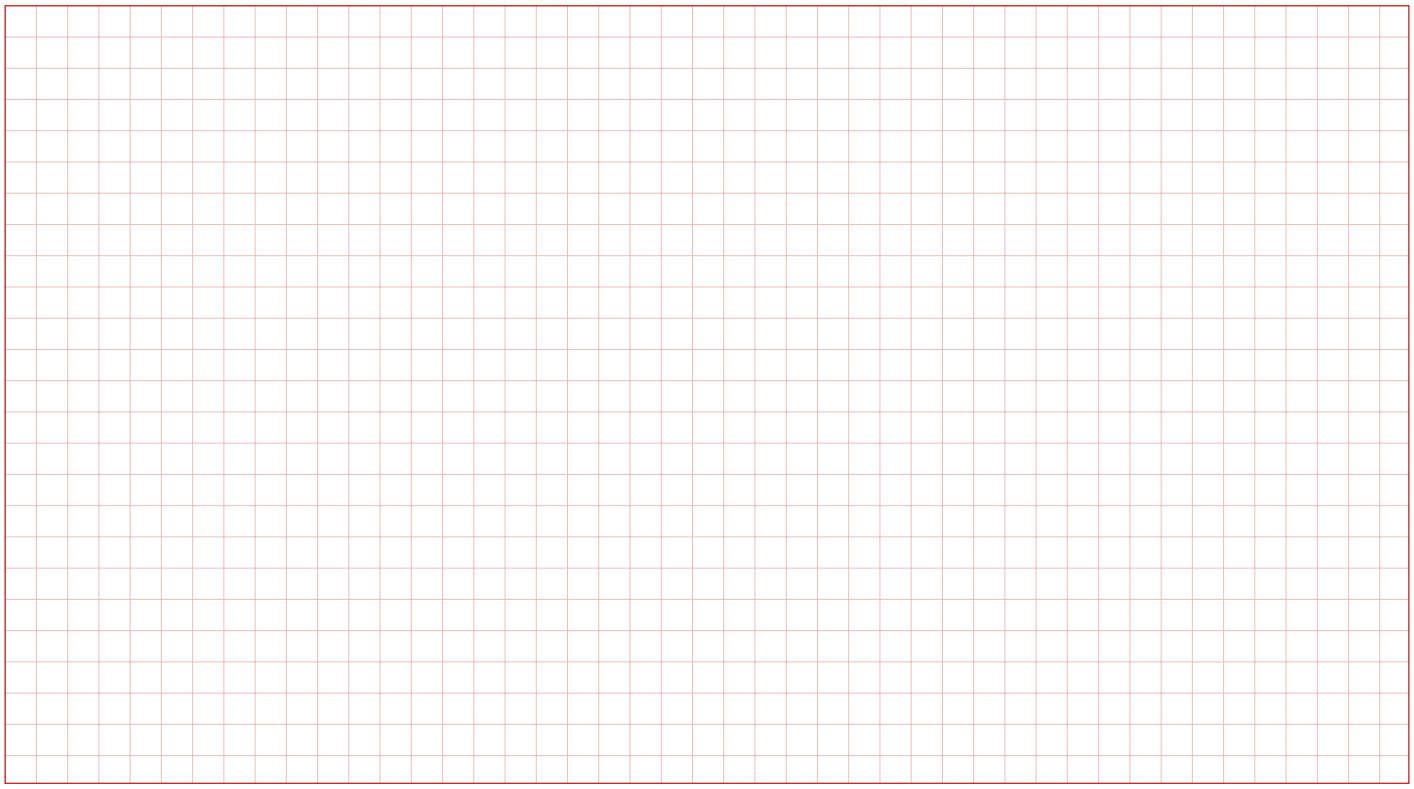
```
numbers == numbers2 : true
numbers == numbersClone : false
```


Notizen



Inhalt

Notizen



Vergleich von Arrays: Zu Fuß

- Methode vergleicht zwei **int**-Arrays

runArrayEqualsIntByFoot

```
40 public static boolean arrayEqualsIntByFoot(
41     int[] a, int[] b) {
42   if (a.length != b.length)
43     return false;
44
46   for (int i = 0; i < a.length; i++){
47     if (a[i] != b[i])
48       return false;
49   }
50   return true;
51 }
```

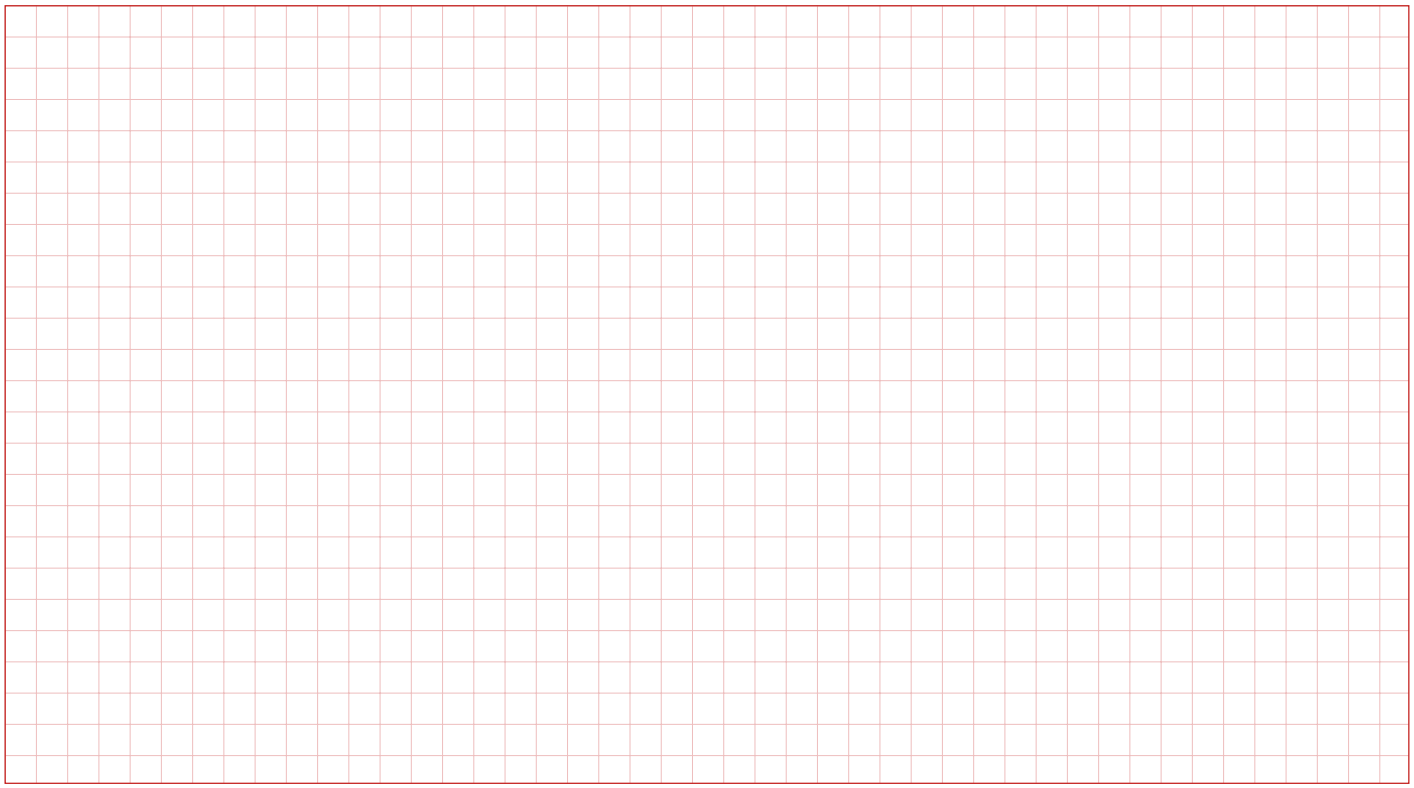
ArrayEqualsExamples.java

Notizen

Inhaltlicher Vergleich von Arrays: Zu Fuß

- Aufrufe der vorherigen Methode
  - a={1,2,3}, b={1,2,3} : **true**
  - a={1,2,3,4}, b={1,2,3} : **false**
  - a={}, b={1,2,3} : **false**
  - a={1,2,3}, b={1,2,4} : **false**
- Entspricht: T[].clone() beim Kopieren
- Hier: Wertvergleich von primitiven Typen mit == möglich
- Was ist mit Referenztypen?

Notizen



Inhaltlicher Vergleich von Arrays: Zu Fuß

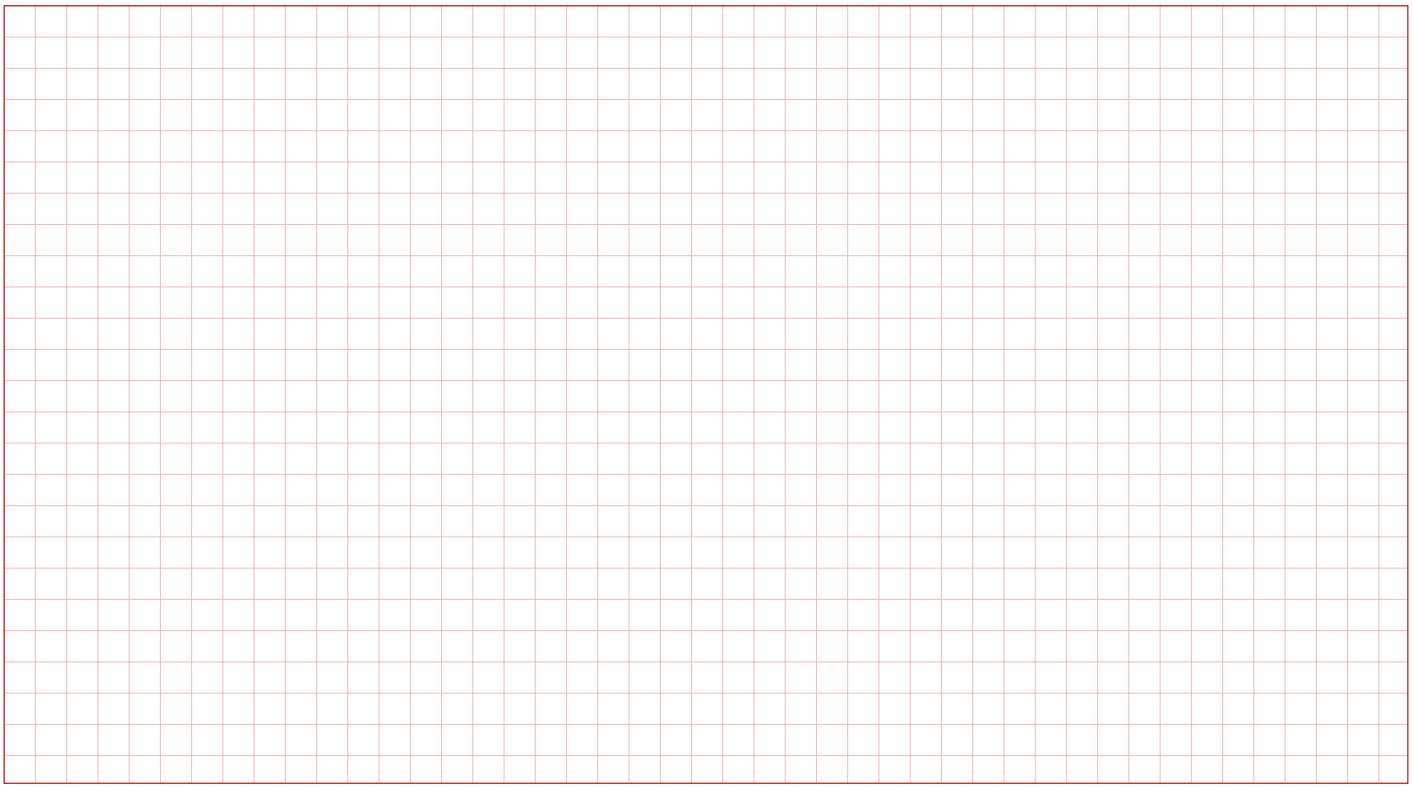
- Referenztypen mit `Objects.equals()` vergleichen

```
55 public static boolean arrayEqualsPoint2DByFoot(
56     Point2D[] a, Point2D[] b) {
58   if (a.length != b.length)
59     return false;
61   for (int i = 0; i < a.length; i++){
62     if (!Objects.equals(a[i], b[i]))
63       return false;
64   }
65   return true;
66 }
```

ArrayEqualsExamples.java

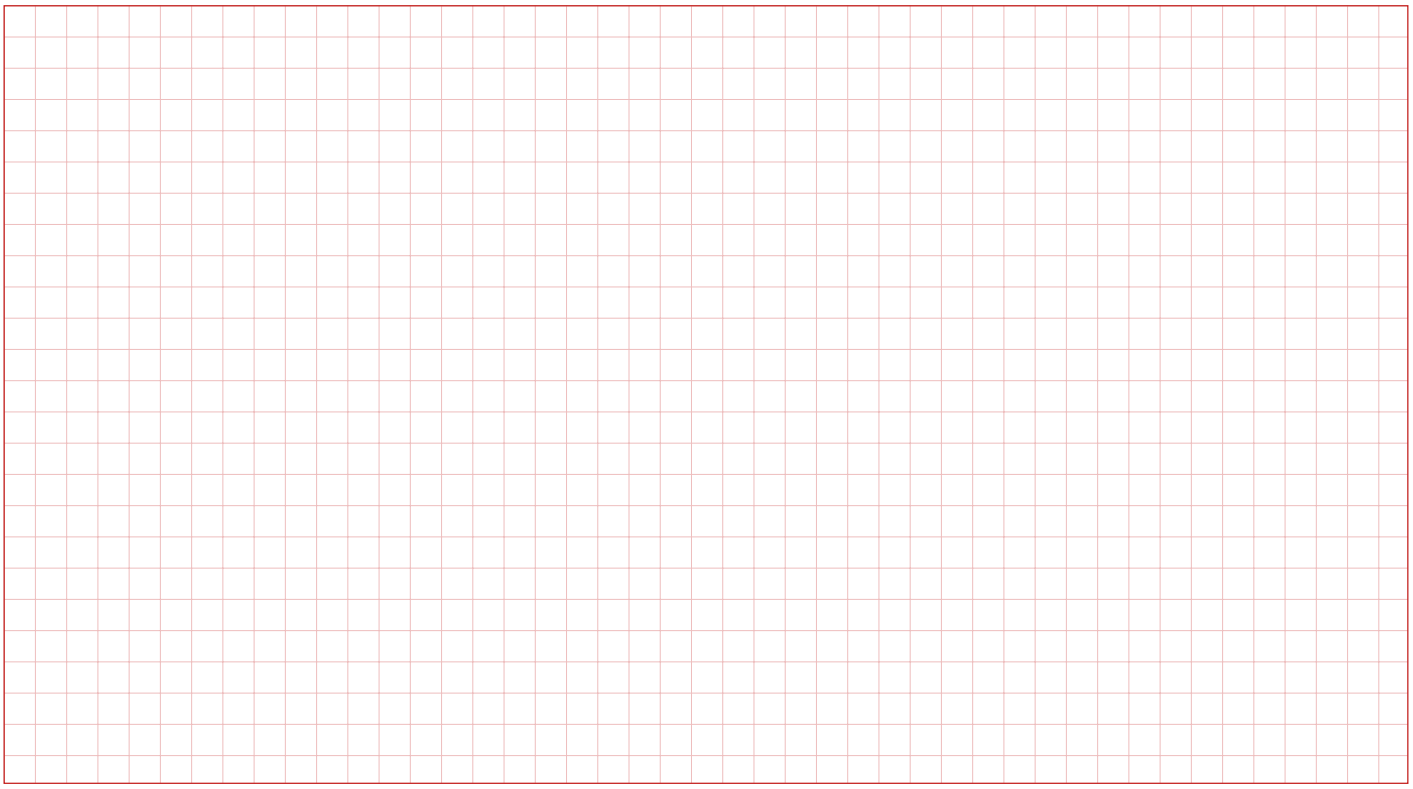
- Entspricht: tiefer Kopie des Arrays

Notizen



Inhalt

Notizen



Inhaltlicher Vergleich von Arrays über Arrays.equals

- Hilfsmethode Arrays.equals implementiert inhaltlichen Vergleich
- Beispiel

```
71 runArrayArraysEqualsExample
72 Point2D[] triangle = new Point2D[]{
73   new Point2D(0,0), new Point2D(2,0), new Point2D(1,1)};
74 Point2D[] triangle2 = new Point2D[]{
75   new Point2D(0,0), new Point2D(2,0), new Point2D(1,1)};
76 Point2D[] triangle3 = new Point2D[]{
77   new Point2D(0,0), new Point2D(2,0), new Point2D(1,2)};
79 boolean t1EqualsT2 = Arrays.equals(triangle, triangle2);
80 boolean t1EqualsT3 = Arrays.equals(triangle, triangle3);
```

ArrayEqualsExamples.java

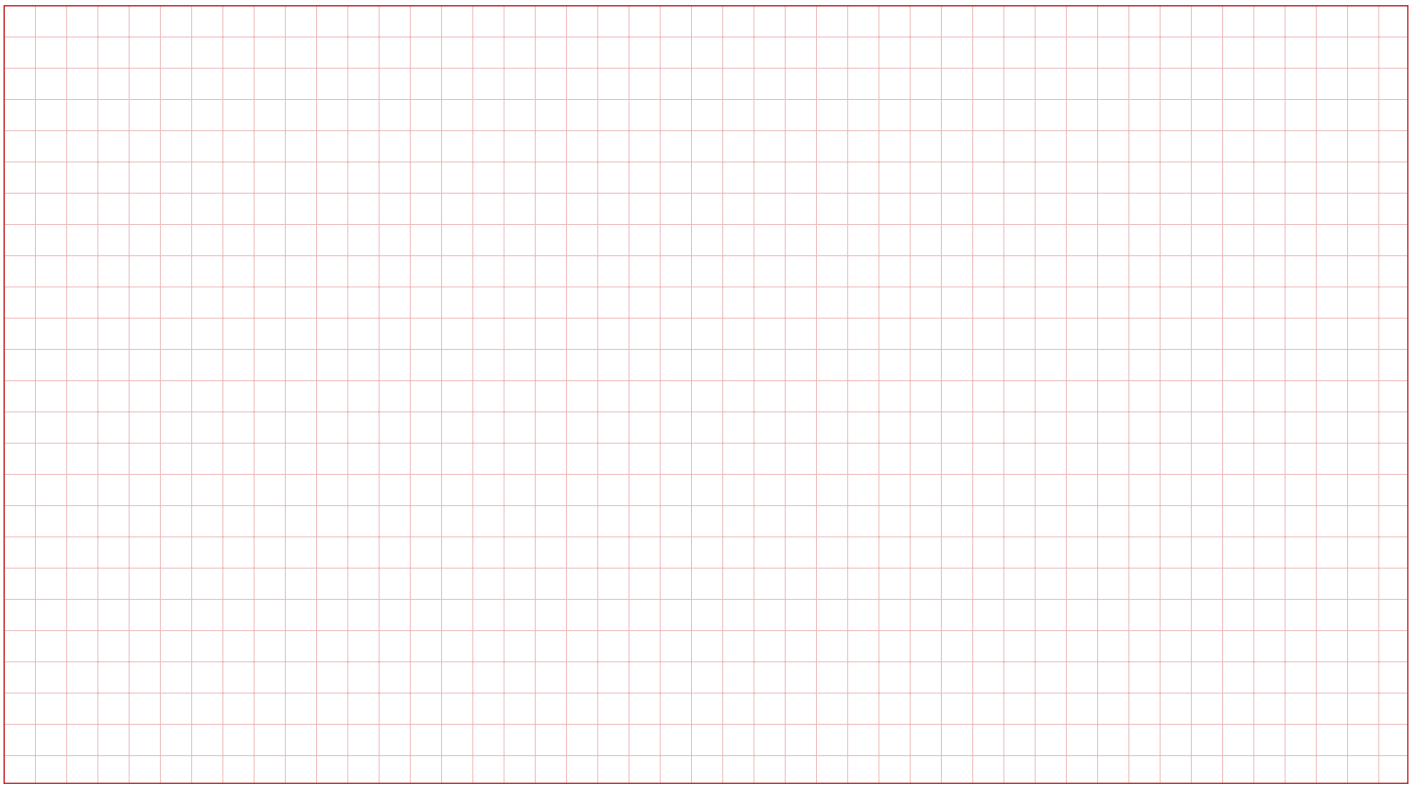
```
Arrays.equals(triangle, triangle2) : true
Arrays.equals(triangle, triangle3) : false
```


Notizen

Funktionsweise von Arrays.equals

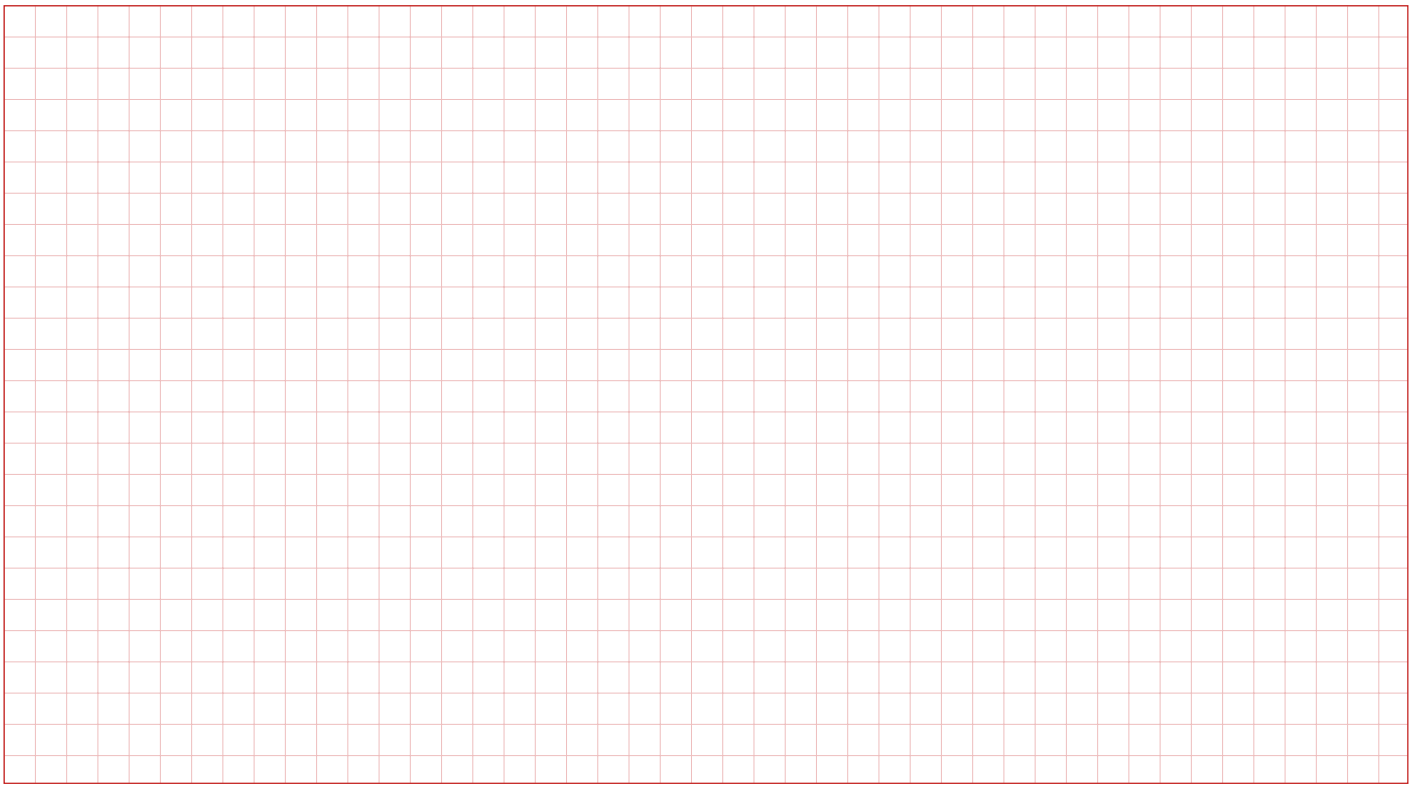
- Funktionsweise von Arrays.equals
- Wie bei unserer Zu-Fuß-Implementierung
  1. Größe vergleichen
  2. Element für Element vergleichen
     - Primitive Typen mit ==
     - Referenztypen mit equals

Notizen



Inhalt

Notizen



Mehrdimensionale Arrays

- Auch über Arrays.equals — oder?

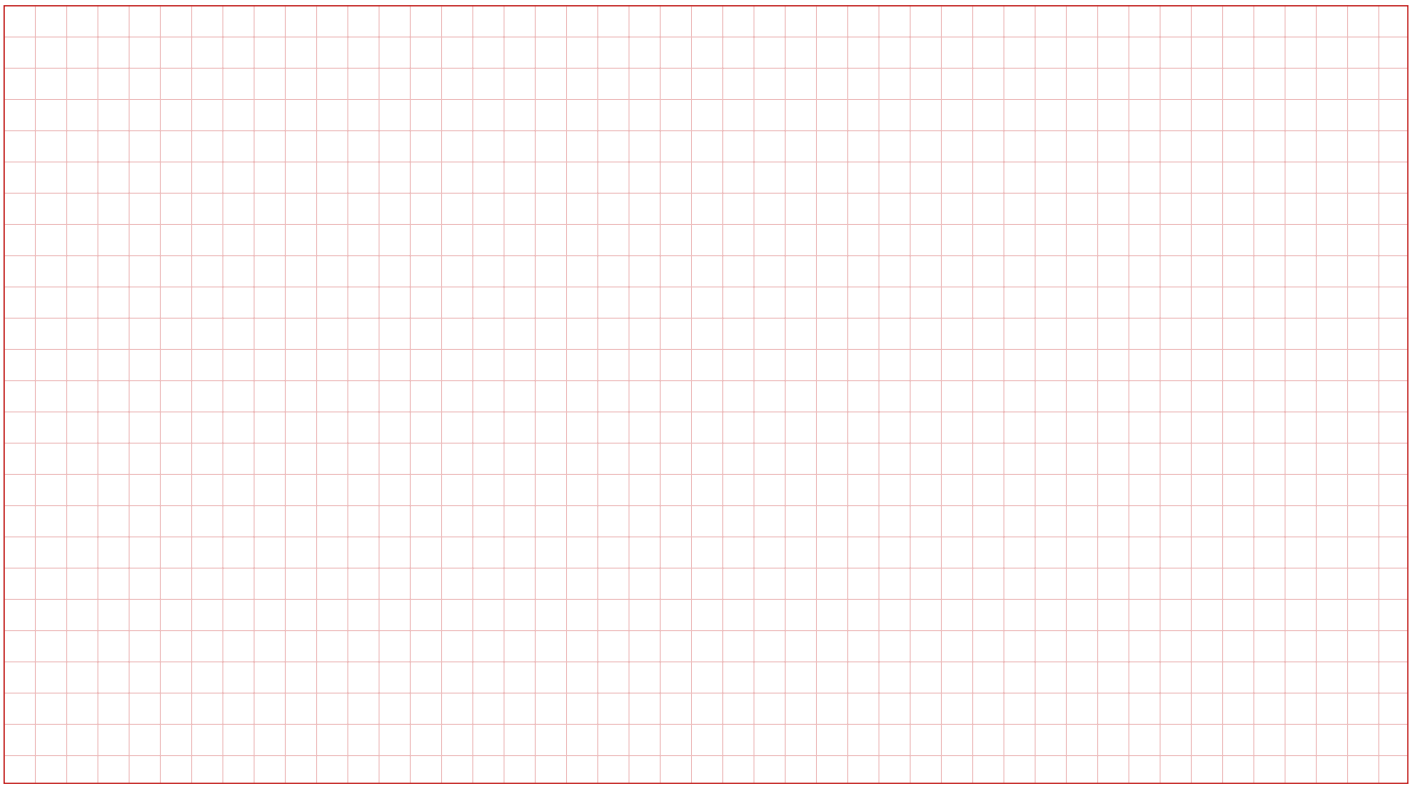
```
88 runArrayTwoDimensionalEqualsExample
89 int[][] a1 = { {1,2,3}, {4,5,6}, {6,8,9} };
90 int[][] a2 = { {1,2,3}, {4,5,6}, {6,8,9} };
92 System.out.printf("Arrays.equals(a1,a2) : %b%n",
93     Arrays.equals(a1,a2));
```

ArrayEqualsExamples.java

```
Arrays.equals(a1,a2) : false
```

- Ergebnis: **false** — obwohl beide Arrays wertgleich sind!
- Begründung:
  - Arrays.equals ruft equals auf den „Zeilen"-Arrays a1[i].equals(a2[i]) auf
  - Aber: equals von Array prüft nur Identität
  - a1[i] und a2[i] sind zwar wertgleich, aber nicht identisch (unterschiedliche Instanzen)

Notizen



Mehrdimensionale Arrays

- Tiefer Vergleich von mehrdimensionalen Arrays über `Arrays.deepEquals()`

```
99  runArrayTwoDimensionalDeepEqualsExample
100 int[][] a1 = { {1,2,3}, {4,5,6}, {7,8,9} };
101 int[][] a2 = { {1,2,3}, {4,5,6}, {7,8,9} };
103 System.out.printf("Arrays.deepEquals(a1,a2) : %b%n",
104     Arrays.deepEquals(a1,a2));
```

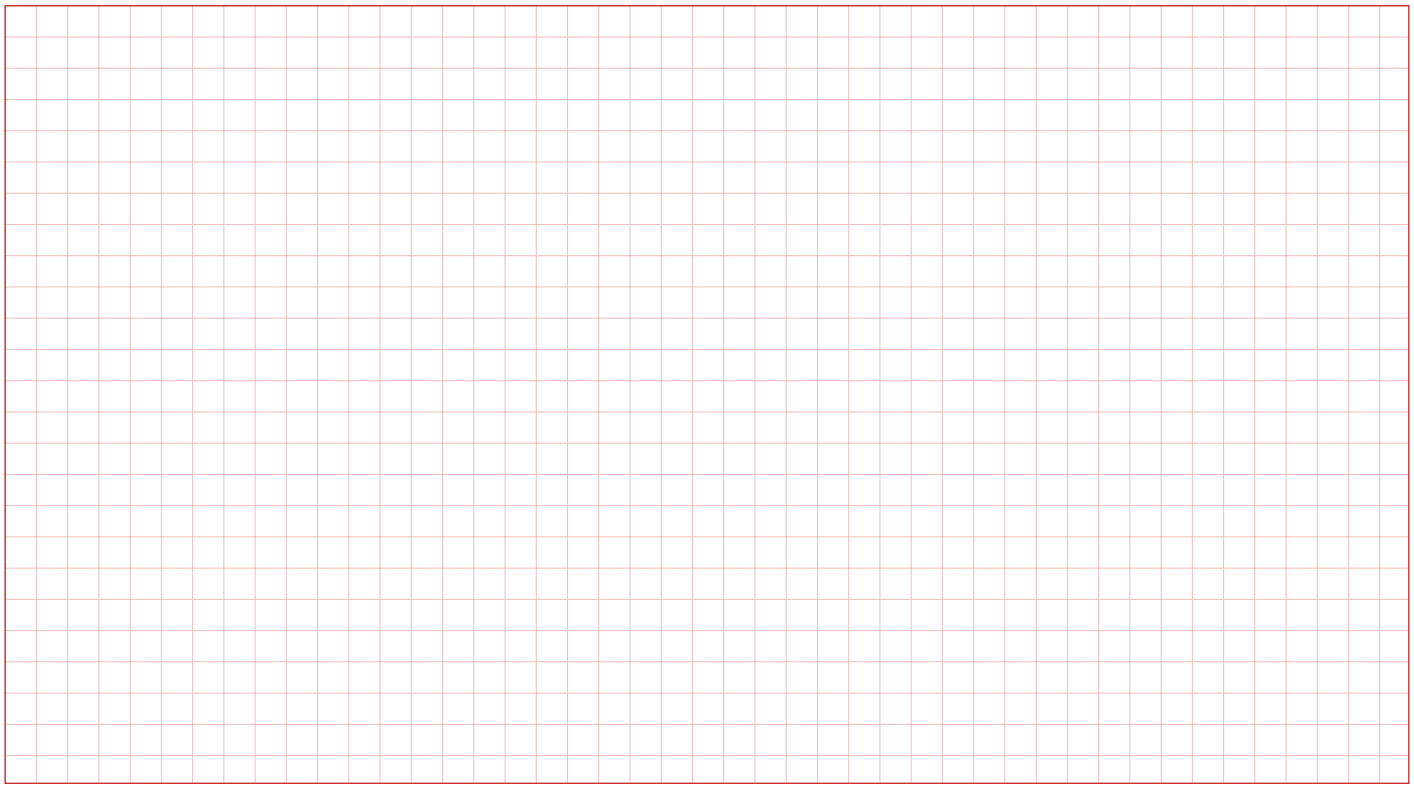
ArrayEqualsExamples.java

```
Arrays.deepEquals(a1,a2) : true
```

- Funktioniert auch mit mehr Dimensionen und Referenztypen
- Funktionsweise:
  - Prinzipiell wie `Arrays.equals`
  - Sind Array-Elemente wieder Arrays , wird deepEquals rekursiv aufgerufen

Notizen

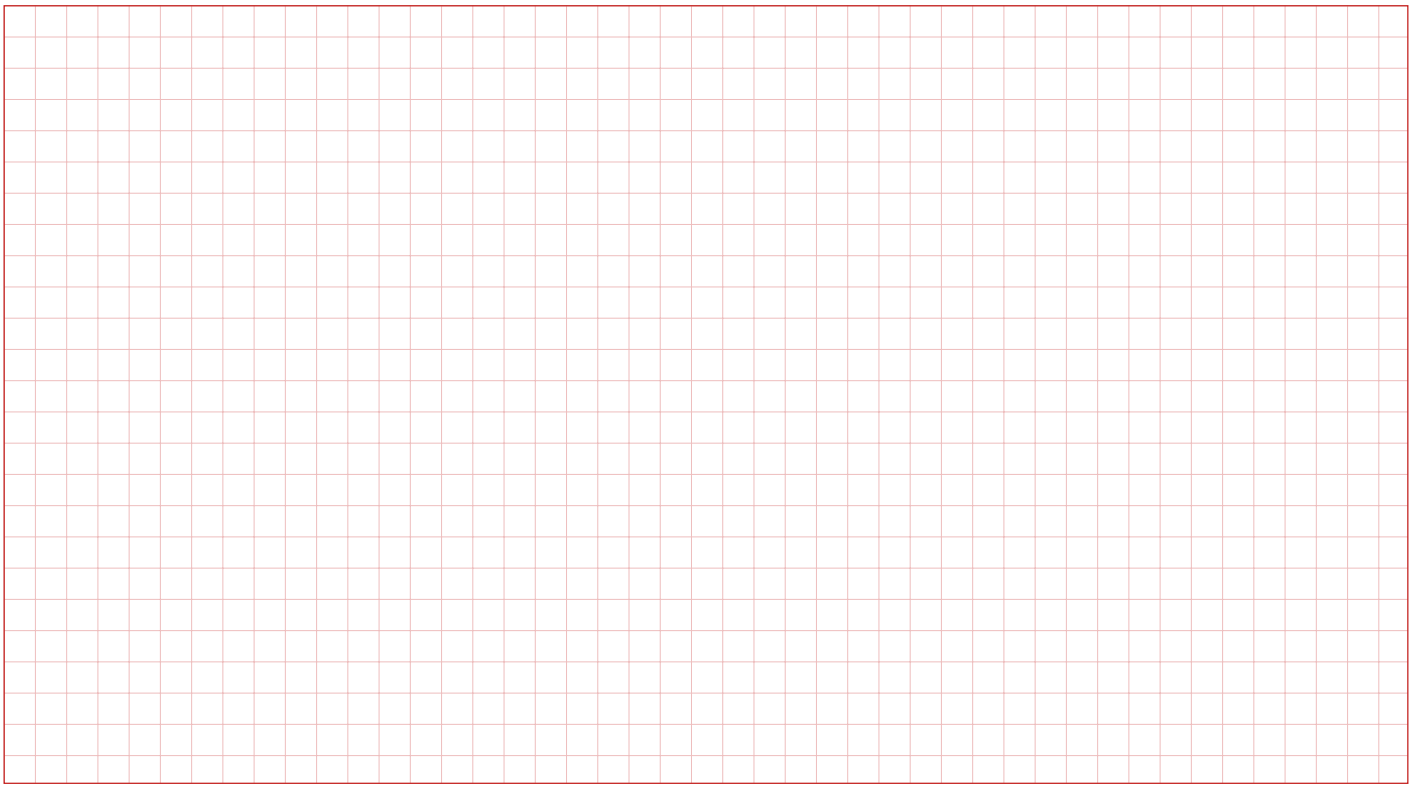
Notizen



Zusammenfassung: Vergleich von Arrays

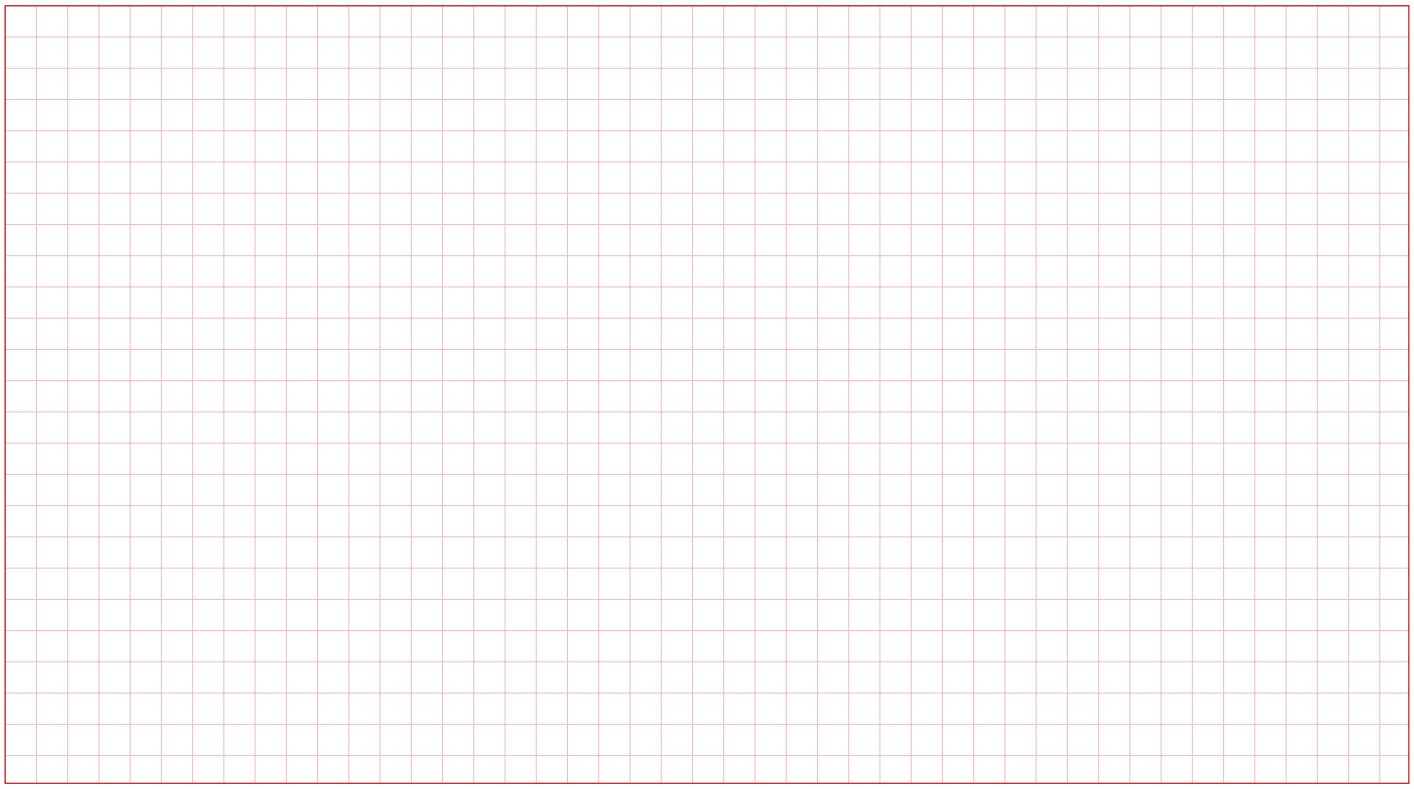
- Wann nimmt man was?
  - Identität: `==`
  - Eindimensionale Arrays: `Arrays.equals`
  - Mehrdimensionale Arrays: `Arrays.deepEquals`
- Vorsicht
  - `T[].equals` prüft nur Identität
  - `Arrays.deepEquals` kostet bei tiefen, großen Arrays viel Zeit

Notizen



Inhalt

Notizen



Die Hilfsklasse `Arrays`

- Hilfsklasse `Arrays` enthält nützliche Methoden für Arrays
  - `binarySearch` — Binärsuche in sortieren Arrays (in $\mathcal{O}(\log n)$)
  - `compare` — lexikographischer Vergleich von Arrays (vgl. `String.compareTo`)
  - `copyOf/copyOfRange` — kopieren von Arrays
  - `equals/deepEquals` — vergleichen von Arrays
  - `fill` — belegen von Arrays mit Werten
  - `hashCode` — Hashwerte von Arrays berechnen
  - `mismatch` — suchen des ersten Unterschied in zwei Arrays
  - `parallelPrefix/SetAll/Sort` — paralleles Aggregieren/Initialisieren/Sortieren
  - `stream` — umwandeln in Streams (s. Programmieren III)
  - `sort` — sortieren von Arrays
  - `toString` — umwandeln in String

Notizen

- Vergleiche `https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/util/Arrays.html`