Geschachtelte Arrays

- Motivation
- Deklaration
- Initialisierung und Wertzuweisung
- Zugriff auf Komponenten

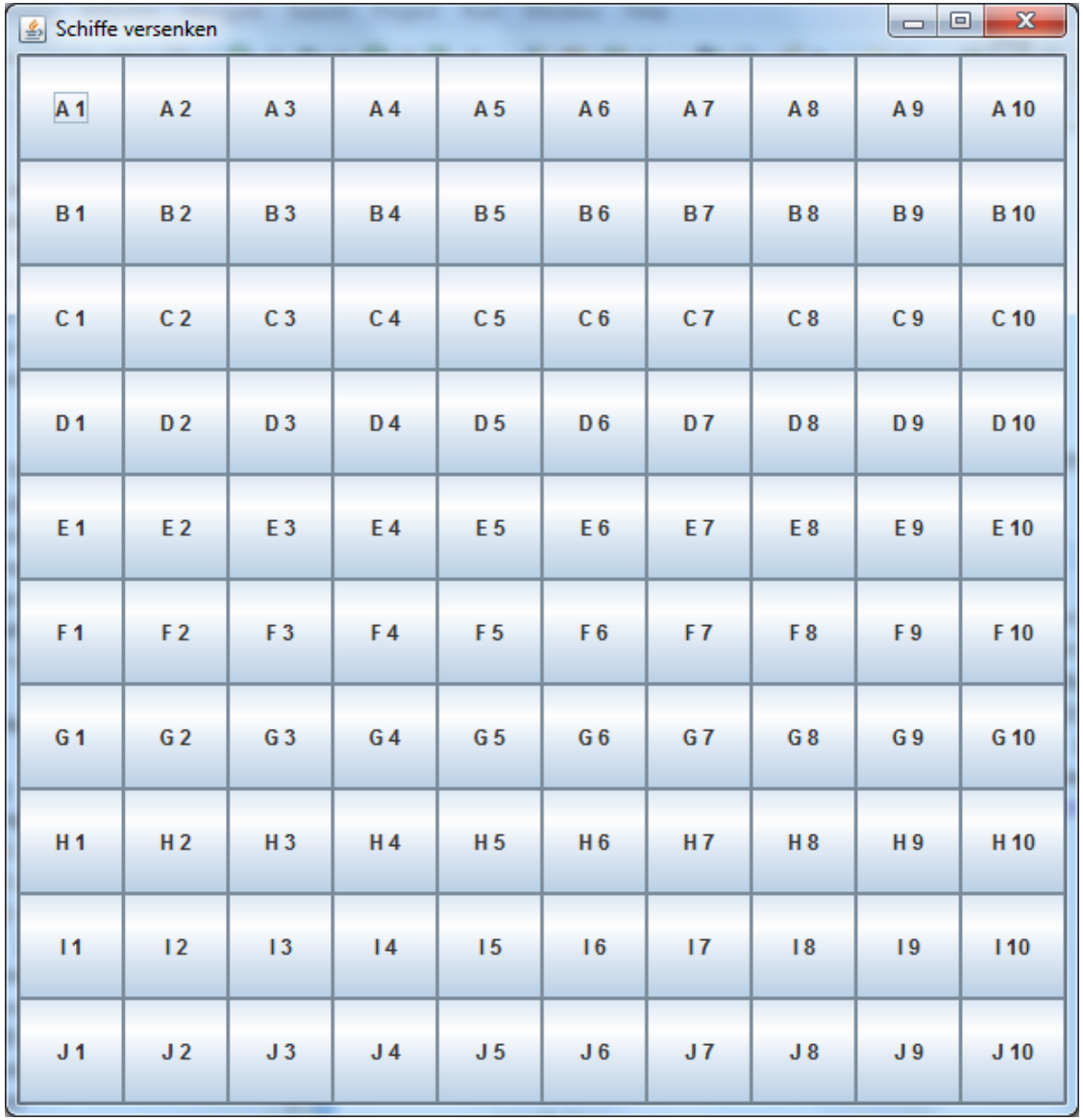
- Exkurs: Teilarrays verschiedener Länge

Beispiel für ein zweidimensionales Array: Entfernungstabelle

Element 3, 4: Zeile 3, Spalte 4 (jeweils beginnend bei 0)

	Aachen	Augsburg	Baden-Baden	Berlin	Bielefeld	Bonn
Aachen	0	593	409	633	257	91
Augsburg	593	0	258	586	600	502
Baden-Baden	409	258	0	723	522	318
Berlin	633	586	723	0	390	598
Bielefeld	257	600	522	390	0	228
Bonn	91	502	318	598	228	0

Spielfeld für Schiffe versenken



Vereinbarung

- Arrays in Java sind eigentlich *immer eindimensional* (also Vektoren), sie können aber *geschachtelt* werden;
  man bezeichnet sie deshalb oft als mehrdimensional
  (wir machen in dieser Vorlesung keinen Unterschied)

- Deklaration
  - int [ ][ ] ai1, ai2; → ai1 und ai2 sind zweidimensionale array of int
  - char [ ][ ] ... [ ] ac1, ac2; → ac1 und ac2 sind …fach geschachtelte bzw. ...dimensionale array of char
  - In Java sind *beliebig viele* Schachtelungsebenen zulässig
  - Die Anzahl der Klammerpaare zeigt die Tiefe der Schachtelung an
- Formal
  - *Elementtyp* [ ][ ] ... [ ] *Variable4*, *Variable5*;

Initialisierung und Wertzuweisung

- Initialisierung
  - int [ ][ ] matrix = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

- Formal
  - *Typ* [ ][ ] ... [ ] *Variable* = { {...{{...}, {...},...{...}}...} };
  - Die Tiefe der Schachtelung von Deklaration und Initialisierungsausdruck muss übereinstimmen (hier: beide 2)

- Nachträgliches Anlegen
  - int [ ][ ] matrix;
    matrix = new int[10][12];

$$\begin{pmatrix} 0_{0,0} & 0_{0,1} & \cdots & 0_{0,11} \\ 0_{1,0} & 0_{1,1} & \cdots & 0_{1,11} \\ \cdots & \cdots & \cdots & \cdots \\ 0_{9,0} & 0_{9,1} & \cdots & 0_{9,11} \end{pmatrix}$$

  - *Typ* [ ][ ] ... [ ] *Variable*;
    Variable = new *Typ*[*Länge1*][*Länge2*]...[*Länge n*];
  - Die Anzahl der Klammern bei der Initialisierung muss mit der Schachtelungstiefe des Arrays übereinstimmen

Zugriff auf Komponenten

- matrix[i][j]
  Nicht: matrix[i, j] → das wäre – zum Beispiel in C – die Notation für den Zugriff auf ein zweidimensionales Array!
- Formal: *Variable* [*Index1*][*Index2*] ... [*Index n*]
- Ist die Anzahl der Indexklammern *gleich* der Schachtelungstiefe des Arrays, so wird als Ergebnis ein Wert vom Komponententyp des Arrays geliefert
- Ist die Anzahl der Indexklammern *größer* als die Schachtelungstiefe des Arrays, so meldet der Compiler einen Fehler
- Ist die Anzahl der Indexklammern *geringer* als die Schachtelungstiefe des Arrays, so wird als Ergebnis ein (eventuell mehrfach geschachteltes) Array geliefert, dessen Schachtelungstiefe der Anzahl der fehlenden Klammern entspricht und das denselben Komponententyp wie das ursprüngliche Array hat

Wertzuweisung bei mehrfach geschachtelten Arrays

- Wird ein (mehrfach) geschachteltes Array einem anderen zugewiesen, gilt findet wie schon bei eindimensionalen Arrays eine Deskriptoren-Zuweisung statt
- Das bedeutet, im Deskriptor wird die Referenz auf das ursprüngliche Array-Objekt durch eine Referenz auf das neue Array-Objekt ersetzt
- Wird ein Teil-Array zugewiesen, so wird entsprechend eine Referenz auf das neue Teil-Array gesetzt

- Alternative
  Elementweises Kopieren der (Teil)-Arrays in (geschachtelten) Schleifen

Exkurs: Arrays mit verschiedener Dimensionslänge 1/3

- Zum Array-Typ gehört in Java *nur der* Komponenten*typ*, *nicht* die *Länge* der Dimension
- Daher sind unterschiedlich lange Teil-Arrays möglich (was mit mehrdimensionalen Arrays nicht möglich wäre!)
- Beispiel: Pascal‘sches Dreieck

```
int [ ][ ] pascal = { { 1 }, { 1, 1 }, { 1, 2, 1 } };
```

- Erlaubt sind hier genau die folgenden Zugriffe

```
pascal[0][0],
pascal[1][0], pascal[1][1],
pascal[2][0], pascal[2][1], pascal[2][2]
```

- Zur Erinnerung:
  Die Länge der einzelnen Teil-Arrays lässt sich mittels “.length“ abfragen; Beispiel:
  pascal[1].length →
  pascal[2].length →

Exkurs: Arrays mit verschiedener Dimensionslänge 2/3

- Gegeben sei die folgende Vereinbarung

```
int [ ][ ][ ][ ] a1, a2;
```

- Bekannt ist bereits das folgende Anlegen eines der Arrays

```
a1 = new int[3][3][2][4];
```

- Von rechts her dürfen Dimensions*längen* ausgelassen werden (es müssen jedoch alle Schachtelungsebenen aufgeführt werden)

```
a2 = new int[3][3][ ][ ];
```

- Die fehlende Dimensionslänge kann nachträglich angegeben werden:

```
a2[1][0] = new int[2][4];
a2[1][1] = new int[5][ ];       // immer noch eine ausgelassen
```

Exkurs: Arrays mit verschiedener Dimensionslänge 3/3

- Gegeben sei die folgende Vereinbarung

```
int [ ][ ][ ][ ] a;
```

- Es ist nicht erlaubt, Dimensionslängen anzugeben, wenn übergeordnete Dimensionslängen ausgelassen wurden

```
a = new int[3][3][ ][2];
```
→ Compilerfehler

- Es ist nicht erlaubt, Dimensionen auszulassen

```
a = new int[3][4][ ];
```
→ Compilerfehler

Haben Sie Fragen?