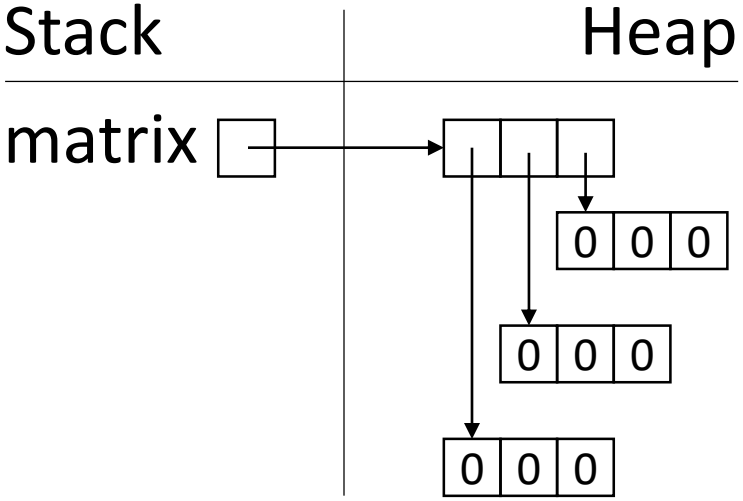
mehrdimensionale Arrays

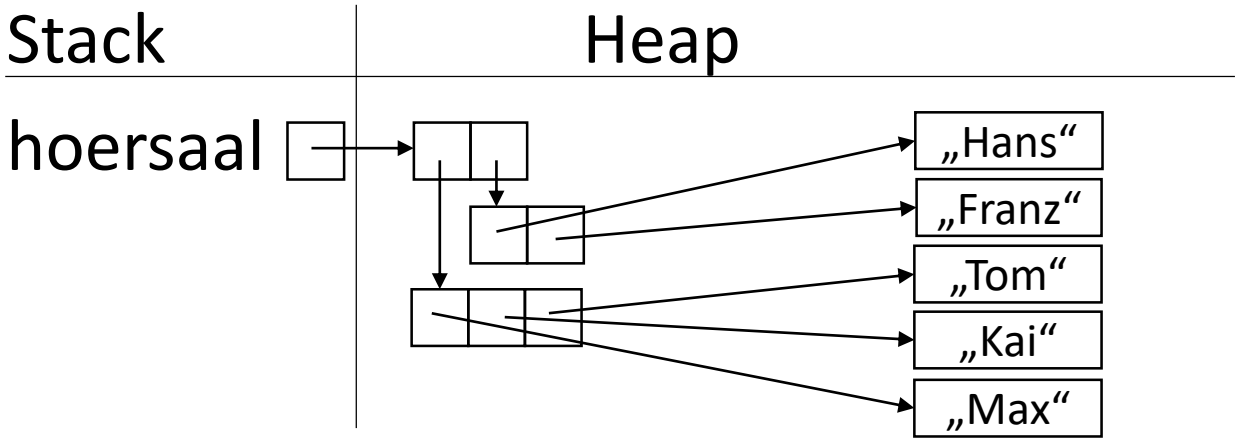
2-dim Arrays sind Arrays mit 1-dim Arrays als Komponenten

Beispiel:

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




```
String[][] hoersaal =
 { { „Max“, „Kai“, „Tom“ },
   { „Hans“, „Franz“ } };
```

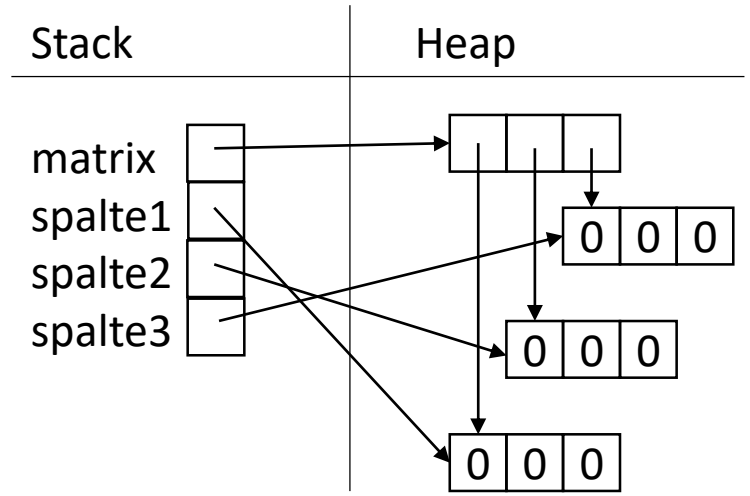


mehrdimensionale Arrays

2-dim Arrays sind Arrays mit 1-dim Arrays als Komponenten

Beispiel:

```
int[][] matrix = new int[3][];
int[] spalte1 = new int[3];
int[] spalte2 = new int[3];
int[] spalte2 = new int[3];
matrix[0] = spalte1;
matrix[1] = spalte2;
matrix[2] = spalte3;
```



mehrdimensionale Arrays

x-dim Arrays sind Arrays mit x-1-dim Arrays als Komponenten

Beispiel:

```
int[][][] quader = new int[3][][];
quader[0] = new int[3][];
quader[1] = new int[3][];
quader[2] = new int[3][];
```

Die Dimensionen und die Nachbarn können unterschiedliche Längen haben:

```
quader[0][0] = new int[1];
quader[0][1] = new int[3];
```

…

mehrdimensionale Arrays

- Aufgaben 38 + 39

Methoden

- Wiederkehrende Programmteile:
  - Sollten nicht mehrmals implementiert werden, sondern an einer Stelle angeboten werden
    - Änderungen einfacher (man vergisst keine Stelle)
    - Fehlersuche effektiver (der Ort des Fehlers lässt sich stärker einschränken)
- Komplexe Programme
  - Resultieren aus komplexen Problemstellungen
  - Sollten nicht an einem Stück implementiert werden
  - Probleme sollten in Teilprobleme zerlegt werden → Komplexität reduzieren
    - Teilprobleme implementieren…
    - … und zu einem Programm zusammensetzen.

Methoden

- OOP
  - Objekte kommunizieren über Methoden miteinander
  - Methoden stellen die Definition der Nachrichten dar …
  - … und beschreiben damit die Funktionalität von Klassen/Objekten.

Methoden

- Syntax:
  - Methodenkopf

```
[Sichtbarkeit][Modifikatoren]Rückgabetyp Bezeichner
( [Parameterliste] ) [throws Ausnahmeliste]
```

  - Methodenrumpf

```
{
        [Anweisungen]
}
```

[] - optional

Methoden - Sichtbarkeiten

- public
  - Für alle Objekte/Klassen sichtbar
- „package“ – standard, darfd nicht hingeschrieben werden
  - Innerhalb des selben Pakets sichtbar
- protected
  - package + sichtbar für Unterklassen (Vererbung -> nächste Woche)
- private
  - NUR für die eigene Klasse sichtbar

Jetzt erstmal alles public

Methoden - Modifikatoren

- static
  - Statische Methode → ohne Objekt nutzbar
- final
  - Unveränderlich (Vererbung -> nächste Woche)

Weitere Modifikatoren folgen später im Lehrgang

Methoden - Rückgabetyp

- primitiv
  - byte, short, int, long, float, double, char, boolean
- Referenztypen
  - String, Random, …
  - Wrapper Klassen (Byte, Short, Integer, Long, Float, Double, Character, Boolean)
  - Arrays (primitiv: int[], long[]; Referenztypen: String[], …)
- Kein Rückgabetyp:
  - void = „die Leere“

Methoden - Bezeichner

- lowerCamelCase
  - Beginnt mit einem kleinen Buchstaben
  - Jedes folgende Wort mit einem Großbuchstaben
  - Sonderzeichen vermeiden
  - Methodennamen stellen grundsätzlich Verben dar oder beginnen mit einem Verb

Methoden - Parameterliste

- Kein Parameter
  - ()
  - Zum Beispiel: PrintStream.println();
- >0 Parameter (feste Anzahl)
  - (int i, String s, Point p, int[] iArr)
  - Zum Beispiel: Math.pow(double a, double b);
- >=0 (variable Anzahl)
  - (Object… o) entspricht Object[] o und muss am Ende der Parameterliste stehen!
  - Zum Beispiel: PrintStream.printf(String format, Object… args);
  - Alle Werte nach dem String werden in einem Array vom Typ Object[] abgelegt.

Methoden – Call-By-Value

- Java kennt nur Call-By-Value
  - Primitive Parameter:
    - Der Wert des Parameters wird in den Stack-Bereich der Methode kopiert.
  - Referenztypen:
    - Die Referenz wird in den Stack-Bereich der Methode kopiert.

  - => Die Variablen des aufrufenden Programmteils können nicht verändert werden,
  - => Objekte von Referenztypen werden nicht kopiert
    - => Objekte können verändert werden, falls es Methoden dazu gibt

Methoden - Ausnahmebehandlung

- Kommt später, aber:
  - Hinter dem Schlüsselwort throws lassen sich Ausnahmen auflisten, welche von dieser Methode ausgelöst werden können.
  - Das heißt, diese Ausnahmen werden nicht in der Methode behandelt, sondern der Aufrufer muss sich darum kümmern, oder
  - Er schreibt an seine Methode auch die gleiche Ausnahme dran und liefert sie einfach weiter an den übergeordneten Aufrufer.

  - Sobald eine Ausnahme die JavaVM erreicht, also in der main-Methode ausgelöst wird, wird das Programm sofort beendet.

Methoden - Runpf

- beliebige Anzahl an Anweisungen und
- return:
  - Besitzt die Methode einen Rückgabetyp != void, dann:
    - Muss mindestens einmal am Ende der Methode das Schlüsselwort return gefolgt vom Rückgabewert (Variable oder Konstante) stehen.
    - Zum Beispiel: return 0; return 0L; return „Teststring“; return result; return;
    - Hinter return darf kein Code mehr folgen: unerreichbar => Compiler Fehler
    - Guter Stil:
    - => Jede Methode hat genau ein Ende und damit ein return.
    - => Bei vielen return Anweisungen verliert man den Überblick => Lesbarkeit leidet.

Methoden - Überladung

- Methoden unterscheiden sich durch ihre Signatur:
  - Signatur = Kombination aus Bezeichner und Parameterliste
  - Mehrere Methoden mit dem selben Bezeichner, aber unterschiedlicher Parameterliste bezeichnet man als Überladen.
  - Beispiele: addieren

```
public static int addieren( int a, int b ){
      return a + b;
}
public static double addieren( double a, double b ){
      return a + b;
}
public static int addieren( int a, int b, int c ){
      return a + b + c;
}
```

Methoden - Defaultparameter

- Defaultparameter existieren bei Java nicht
  - Kann mit durch Überladen simuliert werden:

```
// "Haupt-Methode"
public static int addieren( int a, int b, int c, int d ){
      return a + b + c + d;
}
// Hilfsmethode ruft Haupt-Methode auf
// -> implementiert die Lösung nicht selbst!
public static int addieren( int a, int b, int c ){
      return addieren(a, b, c, 0);
}
// Hilfsmethode ruft Haupt-Methode auf
// -> implementiert die Lösung nicht selbst!
public static int addieren( int a, int b ){
      return addieren(a, b, 0, 0);
}
```

Methoden

- Aufgaben: 40 - 45

Methoden - Rekursion

- Rekursive Methoden rufen sich selbst auf.
  - Zum Beispiel:

```
int sum( int n ){
        return n + sum( n – 1 );
}
```

  - Dabei müssen die Parameter auf Akzeptanz geprüft werden.
  - Alle Probleme lassen sich iterativ (mit Schleifen) und rekursiv lösen,
  - meist ist ein rekursiver Ansatz kürzer (eleganter) zu programmieren,
  - der iterative Ansatz ist aber im allgemeinen Ressourcen schonender

Methoden

- Aufgaben: 46 - 49

Weitere Aufgaben:

- Aufgaben 50 - 57