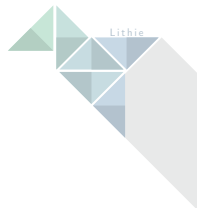


# Raute-Hashtag-Matrizen

Mathemann 5

Florian Sihler ◦ KW 48



# Präsenzaufgabe, zum Auflockern

1

Flippn' Amazing – Watch me Flip-Flip...

# Präsenzaufgabe, zum Auflockern

1

Flippn' Amazing – Watch me Flip-Flip...

Implementieren Sie die Methoden `public static void flipInPlace(char[] x)` und `public static char[] flipInCopy(char[] x)`, welche beide die Groß- und Kleinschreibung in den Arrays umkehren (nur ASCII).

# Präsenzaufgabe, zum Auflockern

1

Flippn' Amazing – Watch me Flip-Flip...

Implementieren Sie die Methoden `public static void flipInPlace(char[] x)` und `public static char[] flipInCopy(char[] x)`, welche beide die Groß- und Kleinschreibung in den Arrays umkehren (nur ASCII). `flipInPlace` soll das übergeben Array modifizieren, `flipInCopy` soll `x` aber nicht verändern und eine „geflippte“ Kopie zurückliefern.

# Präsenzaufgabe, zum Auflockern

1

Flippn' Amazing – Watch me Flip-Flip...

Implementieren Sie die Methoden `public static void flipInPlace(char[] x)` und `public static char[] flipInCopy(char[] x)`, welche beide die Groß- und Kleinschreibung in den Arrays umkehren (nur ASCII). `flipInPlace` soll das übergeben Array modifizieren, `flipInCopy` soll `x` aber nicht verändern und eine „geflippte“ Kopie zurückliefern. Zum Erstellen einer Kopie können Sie `char[] copy = x.clone()` verwenden.

# Präsenzaufgabe, zum Auflockern

1

Flippn' Amazing – Watch me Flip-Flip...

Implementieren Sie die Methoden `public static void flipInPlace(char[] x)` und `public static char[] flipInCopy(char[] x)`, welche beide die Groß- und Kleinschreibung in den Arrays umkehren (nur ASCII). `flipInPlace` soll das übergeben Array modifizieren, `flipInCopy` soll `x` aber nicht verändern und eine „geflippte“ Kopie zurückliefern. Zum Erstellen einer Kopie können Sie `char[] copy = x.clone()` verwenden. Nutzen Sie aber nicht `Character.toUpperCase` oder vergleichbare Methoden,

# Präsenzaufgabe, zum Auflockern

1

Flippn' Amazing – Watch me Flip-Flip...

Implementieren Sie die Methoden `public static void flipInPlace(char[] x)` und `public static char[] flipInCopy(char[] x)`, welche beide die Groß- und Kleinschreibung in den Arrays umkehren (nur ASCII). `flipInPlace` soll das übergeben Array modifizieren, `flipInCopy` soll `x` aber nicht verändern und eine „geflippte“ Kopie zurückliefern. Zum Erstellen einer Kopie können Sie `char[] copy = x.clone()` verwenden. Nutzen Sie aber nicht `Character.toUpperCase` oder vergleichbare Methoden, sondern machen Sie sich die Unicode-Kodierung von `char` zunutze (A–Z ist 65–90, a–z ist 97–122).

# Präsenzaufgabe, zum Auflockern

1

Flippn' Amazing – Watch me Flip-Flip...

Implementieren Sie die Methoden `public static void flipInPlace(char[] x)` und `public static char[] flipInCopy(char[] x)`, welche beide die Groß- und Kleinschreibung in den Arrays umkehren (nur ASCII). `flipInPlace` soll das übergeben Array modifizieren, `flipInCopy` soll `x` aber nicht verändern und eine „geflippte“ Kopie zurückliefern.

Zum Erstellen einer Kopie können Sie `char[] copy = x.clone()` verwenden. Nutzen Sie aber nicht `Character.toUpperCase` oder vergleichbare Methoden, sondern machen Sie sich die Unicode-Kodierung von `char` zunutze (A–Z ist 65–90, a–z ist 97–122).

Beantworten Sie weiter:



# Präsenzaufgabe, zum Auflockern

1

Flippn' Amazing – Watch me Flip-Flip...

Implementieren Sie die Methoden `public static void flipInPlace(char[] x)` und `public static char[] flipInCopy(char[] x)`, welche beide die Groß- und Kleinschreibung in den Arrays umkehren (nur ASCII). `flipInPlace` soll das übergeben Array modifizieren, `flipInCopy` soll `x` aber nicht verändern und eine „geflippte“ Kopie zurückliefern.

Zum Erstellen einer Kopie können Sie `char[] copy = x.clone()` verwenden. Nutzen Sie aber nicht `Character.toUpperCase` oder vergleichbare Methoden, sondern machen Sie sich die Unicode-Kodierung von `char` zunutze (A–Z ist 65–90, a–z ist 97–122).

Beantworten Sie weiter:

1. Was sind die Vor- und Nachteile beider Ansätze?

# Präsenzaufgabe, zum Auflockern

1

Flippn' Amazing – Watch me Flip-Flip...

Implementieren Sie die Methoden `public static void flipInPlace(char[] x)` und `public static char[] flipInCopy(char[] x)`, welche beide die Groß- und Kleinschreibung in den Arrays umkehren (nur ASCII). `flipInPlace` soll das übergeben Array modifizieren, `flipInCopy` soll `x` aber nicht verändern und eine „geflippte“ Kopie zurückliefern. Zum Erstellen einer Kopie können Sie `char[] copy = x.clone()` verwenden. Nutzen Sie aber nicht `Character.toUpperCase` oder vergleichbare Methoden, sondern machen Sie sich die Unicode-Kodierung von `char` zunutze (A–Z ist 65–90, a–z ist 97–122).

Beantworten Sie weiter:

1. Was sind die Vor- und Nachteile beider Ansätze?
2. Warum reicht keine einfache Zuweisung zur Kopie des Arrays?

# Präsenzaufgabe, zum Auflockern

1

Flippn' Amazing – Watch me Flip-Flip...

Implementieren Sie die Methoden `public static void flipInPlace(char[] x)` und `public static char[] flipInCopy(char[] x)`, welche beide die Groß- und Kleinschreibung in den Arrays umkehren (nur ASCII). `flipInPlace` soll das übergebene Array modifizieren, `flipInCopy` soll `x` aber nicht verändern und eine „geflippte“ Kopie zurückliefern. Zum Erstellen einer Kopie können Sie `char[] copy = x.clone()` verwenden. Nutzen Sie aber nicht `Character.toUpperCase` oder vergleichbare Methoden, sondern machen Sie sich die Unicode-Kodierung von `char` zunutze (A–Z ist 65–90, a–z ist 97–122).

Beantworten Sie weiter:

1. Was sind die Vor- und Nachteile beider Ansätze?
2. Warum reicht keine einfache Zuweisung zur Kopie des Arrays?
3. Warum funktioniert kein `for-each`, wenn man die Einträge modifizieren möchte?

# Präsenzaufgabe - Lösung

# Präsenzaufgabe - Lösung

- We do se flippin place (mit x als flipEm):

# Präsenzaufgabe - Lösung

- We do se flippin place (mit x als flipEm):

```
public static void flipInPlace(char[] flipEm) {
```

```
}
```

# Präsenzaufgabe - Lösung

- We do se flippin place (mit x als flipEm):

```
public static void flipInPlace(char[] flipEm) {  
    int shift = 'a' - 'A';  
  
}
```

# Präsenzaufgabe - Lösung

- We do se flippin place (mit x als flipEm):

```
public static void flipInPlace(char[] flipEm) {  
    int shift = 'a' - 'A';  
    for (int i = 0; i < flipEm.length; i++) {  
  
  
    }  
}
```



# Präsenzaufgabe - Lösung

- We do se flippin place (mit x als flipEm):

```
public static void flipInPlace(char[] flipEm) {  
    int shift = 'a' - 'A';  
    for (int i = 0; i < flipEm.length; i++) {  
        if ('A' <= flipEm[i] && flipEm[i] <= 'Z') {  
  
            } else if ('a' <= flipEm[i] && flipEm[i] <= 'z') {  
  
            }  
        }  
    }  
}
```

# Präsenzaufgabe - Lösung

- We do se flippin place (mit x als flipEm):

```
public static void flipInPlace(char[] flipEm) {  
    int shift = 'a' - 'A';  
    for (int i = 0; i < flipEm.length; i++) {  
        if ('A' <= flipEm[i] && flipEm[i] <= 'Z') {  
            flipEm[i] += shift;  
        } else if ('a' <= flipEm[i] && flipEm[i] <= 'z') {  
  
        }  
    }  
}
```

# Präsenzaufgabe - Lösung

- We do se flippin place (mit x als flipEm):

```
public static void flipInPlace(char[] flipEm) {  
    int shift = 'a' - 'A';  
    for (int i = 0; i < flipEm.length; i++) {  
        if ('A' <= flipEm[i] && flipEm[i] <= 'Z') {  
            flipEm[i] += shift;  
        } else if ('a' <= flipEm[i] && flipEm[i] <= 'z') {  
            flipEm[i] -= shift;  
        }  
    }  
}
```

# Präsenzaufgabe - Lösung

# Präsenzaufgabe - Lösung

- Und nun mit der Kopie! Wir könnten das ja nochmal schreiben...



# Präsenzaufgabe - Lösung

- Und nun mit der Kopie! Wir könnten das ja nochmal schreiben...

```
public static char[] flipInCopy(char[] flipEm) {  
    char[] copy = flipEm.clone();  
  
}
```

# Präsenzaufgabe - Lösung

- Und nun mit der Kopie! Wir könnten das ja nochmal schreiben...

```
public static char[] flipInCopy(char[] flipEm) {  
    char[] copy = flipEm.clone();  
    flipInPlace(copy);  
  
}
```



# Präsenzaufgabe - Lösung

- Und nun mit der Kopie! Wir könnten das ja nochmal schreiben...

```
public static char[] flipInCopy(char[] flipEm) {  
    char[] copy = flipEm.clone();  
    flipInPlace(copy);  
    return copy;  
}
```

# Präsenzaufgabe - Lösung

# Präsenzaufgabe - Lösung

- Vorteile einer Kopie (gegenüber in-place):

# Präsenzaufgabe - Lösung

- Vorteile einer Kopie (gegenüber in-place):
  - Nebeneffekte von Methoden sind generell schlecht! „out-Parameter“ sollten vermieden werden (wo nicht unbedingt notwendig).

# Präsenzaufgabe - Lösung

- Vorteile einer Kopie (gegenüber in-place):
  - Nebeneffekte von Methoden sind generell schlecht! „out-Parameter“ sollten vermieden werden (wo nicht unbedingt notwendig).
- Nachteile einer Kopie (gegenüber in-place):

# Präsenzaufgabe - Lösung

- Vorteile einer Kopie (gegenüber in-place):
  - Nebeneffekte von Methoden sind generell schlecht! „out-Parameter“ sollten vermieden werden (wo nicht unbedingt notwendig).
- Nachteile einer Kopie (gegenüber in-place):
  - Der benötigte Speicher wird verdoppelt (was, wenn das Array riesig ist?)

# Präsenzaufgabe - Lösung

- Vorteile einer Kopie (gegenüber in-place):
  - Nebeneffekte von Methoden sind generell schlecht! „out-Parameter“ sollten vermieden werden (wo nicht unbedingt notwendig).
- Nachteile einer Kopie (gegenüber in-place):
  - Der benötigte Speicher wird verdoppelt (was, wenn das Array riesig ist?)
- Was ist nun besser?

# Präsenzaufgabe - Lösung

- Vorteile einer Kopie (gegenüber in-place):
  - Nebeneffekte von Methoden sind generell schlecht! „out-Parameter“ sollten vermieden werden (wo nicht unbedingt notwendig).
- Nachteile einer Kopie (gegenüber in-place):
  - Der benötigte Speicher wird verdoppelt (was, wenn das Array riesig ist?)
- Was ist nun besser?
  - In der Regel die Variante mit Kopie. Die unerwarteten/ungewollten Seiteneffekte können Kaskaden schwer zu findender Fehler verursachen.



# Präsenzaufgabe - Lösung

- **Vorteile einer Kopie (gegenüber in-place):**
  - Nebeneffekte von Methoden sind generell schlecht! „out-Parameter“ sollten vermieden werden (wo nicht unbedingt notwendig).
- **Nachteile einer Kopie (gegenüber in-place):**
  - Der benötigte Speicher wird verdoppelt (was, wenn das Array riesig ist?)
- **Was ist nun besser?**
  - In der Regel die Variante mit Kopie. Die unerwarteten/ungewollten Seiteneffekte können Kaskaden schwer zu findender Fehler verursachen.
  - Sollten wirklich große Arrays erwartet werden, sollte man sich unter Umständen ein grundlegend anderes System überlegen.

# Präsenzaufgabe - Lösung

# Präsenzaufgabe - Lösung

- Warum reicht `char[] clone = x;` nicht für eine Kopie?

# Präsenzaufgabe - Lösung

- Warum reicht `char[] clone = x;` nicht für eine Kopie?  
Arrays sind komplexe Datenstrukturen („ReferenceType“).

# Präsenzaufgabe - Lösung

- Warum reicht `char[] clone = x;` nicht für eine Kopie?

Arrays sind komplexe Datenstrukturen („ReferenceType“). Mit `clone = x` lassen wir `clone` auf die gleiche Speicheradresse wie `x` zeigen.

# Präsenzaufgabe - Lösung

- Warum reicht `char[] clone = x;` nicht für eine Kopie?

Arrays sind komplexe Datenstrukturen („ReferenceType“). Mit `clone = x` lassen wir `clone` auf die gleiche Speicheradresse wie `x` zeigen. Wir verändern dasselbe Array.

# Präsenzaufgabe - Lösung

- Warum reicht `char[] clone = x;` nicht für eine Kopie?

Arrays sind komplexe Datenstrukturen („ReferenceType“). Mit `clone = x` lassen wir `clone` auf die gleiche Speicheradresse wie `x` zeigen. Wir verändern dasselbe Array.

- Warum keine for-each Schleife?

# Präsenzaufgabe - Lösung

- Warum reicht `char[] clone = x;` nicht für eine Kopie?

Arrays sind komplexe Datenstrukturen („ReferenceType“). Mit `clone = x` lassen wir `clone` auf die gleiche Speicheradresse wie `x` zeigen. Wir verändern dasselbe Array.

- Warum keine for-each Schleife?

Damit erhalten wir nur die Werte an der jeweiligen Stelle, aber nicht den Index den es zu verändern gilt. Der Iterator erzeugt eine Kopie:



# Präsenzaufgabe - Lösung

- Warum reicht `char[] clone = x;` nicht für eine Kopie?

Arrays sind komplexe Datenstrukturen („ReferenceType“). Mit `clone = x` lassen wir `clone` auf die gleiche Speicheradresse wie `x` zeigen. Wir verändern dasselbe Array.

- Warum keine for-each Schleife?

Damit erhalten wir nur die Werte an der jeweiligen Stelle, aber nicht den Index den es zu verändern gilt. Der Iterator erzeugt eine Kopie:

```
for(char c : flipEm) { /* hier haben wir nur c */ }
```

# Präsenzaufgabe - Lösung

- Warum reicht `char[] clone = x;` nicht für eine Kopie?

Arrays sind komplexe Datenstrukturen („ReferenceType“). Mit `clone = x` lassen wir `clone` auf die gleiche Speicheradresse wie `x` zeigen. Wir verändern dasselbe Array.

- Warum keine for-each Schleife?

Damit erhalten wir nur die Werte an der jeweiligen Stelle, aber nicht den Index den es zu verändern gilt. Der Iterator erzeugt eine Kopie:

```
for(char c : flipEm) { /* hier haben wir nur c */ }
```

Wir könnten zwar einen Zeiger inkrementieren, aber wer garantiert uns dann die Reihenfolge der Iteration?

# Präsenzaufgabe - Lösung

- Warum reicht `char[] clone = x;` nicht für eine Kopie?

Arrays sind komplexe Datenstrukturen („ReferenceType“). Mit `clone = x` lassen wir `clone` auf die gleiche Speicheradresse wie `x` zeigen. Wir verändern dasselbe Array.

- Warum keine for-each Schleife?

Damit erhalten wir nur die Werte an der jeweiligen Stelle, aber nicht den Index den es zu verändern gilt. Der Iterator erzeugt eine Kopie:

```
for(char c : flipEm) { /* hier haben wir nur c */ }
```

Wir könnten zwar einen Zeiger inkrementieren, aber wer garantiert uns dann die Reihenfolge der Iteration? Und selbst wenn das passt...

# Präsenzaufgabe - Lösung

- Warum reicht `char[] clone = x;` nicht für eine Kopie?

Arrays sind komplexe Datenstrukturen („ReferenceType“). Mit `clone = x` lassen wir `clone` auf die gleiche Speicheradresse wie `x` zeigen. Wir verändern dasselbe Array.

- Warum keine for-each Schleife?

Damit erhalten wir nur die Werte an der jeweiligen Stelle, aber nicht den Index den es zu verändern gilt. Der Iterator erzeugt eine Kopie:

```
for(char c : flipEm) { /* hier haben wir nur c */ }
```

Wir könnten zwar einen Zeiger inkrementieren, aber wer garantiert uns dann die Reihenfolge der Iteration? Und selbst wenn das passt... Im Endeffekt haben wir dann wieder eine for-Schleife.

# Übungsblatt 5 - Aufgabe 1 a)

## Übungsblatt 5 - Aufgabe 1 a)

- Der volle Code befindet sich hier: [DotProduct.java](#).

# Übungsblatt 5 - Aufgabe 1 a)

- Der volle Code befindet sich hier: [DotProduct.java](#).
- Wir berechnen  $d = a \cdot b = (a_1 \dots a_n) \cdot (b_1 \dots b_n) = \sum_{i=1}^n a_i \cdot b_i$

## Übungsblatt 5 - Aufgabe 1 a)

- Der volle Code befindet sich hier: [DotProduct.java](#).
- Wir berechnen  $d = a \cdot b = (a_1 \dots a_n) \cdot (b_1 \dots b_n) = \sum_{i=1}^n a_i \cdot b_i$
- Oder als Java-Code:



# Übungsblatt 5 - Aufgabe 1 a)

- Der volle Code befindet sich hier: [DotProduct.java](#).
- Wir berechnen  $d = a \cdot b = (a_1 \dots a_n) \cdot (b_1 \dots b_n) = \sum_{i=1}^n a_i \cdot b_i$
- Oder als Java-Code:

```
int d = 0;
```

## Übungsblatt 5 - Aufgabe 1 a)

- Der volle Code befindet sich hier: [DotProduct.java](#).
- Wir berechnen  $d = a \cdot b = (a_1 \dots a_n) \cdot (b_1 \dots b_n) = \sum_{i=1}^n a_i \cdot b_i$
- Oder als Java-Code:

```
int d = 0;
for(int i = 0; i < length; i++) {
    d += vector1[i] * vector2[i];
}
```

# Übungsblatt 5 - Aufgabe 1 a)

- Der volle Code befindet sich hier: [DotProduct.java](#).
- Wir berechnen  $d = a \cdot b = (a_1 \dots a_n) \cdot (b_1 \dots b_n) = \sum_{i=1}^n a_i \cdot b_i$
- Oder als Java-Code:

```
int d = 0;
for(int i = 0; i < length; i++) {
    d += vector1[i] * vector2[i];
}
```

```
System.out.println("Vector1:_ " + vectorToString(vector1));
System.out.println("Vector2:_ " + vectorToString(vector2));
```

## Übungsblatt 5 - Aufgabe 1 a)

- Der volle Code befindet sich hier: [DotProduct.java](#).
- Wir berechnen  $d = a \cdot b = (a_1 \dots a_n) \cdot (b_1 \dots b_n) = \sum_{i=1}^n a_i \cdot b_i$
- Oder als Java-Code:

```
int d = 0;
for(int i = 0; i < length; i++) {
    d += vector1[i] * vector2[i];
}
```

```
System.out.println("Vector1:_ " + vectorToString(vector1));
System.out.println("Vector2:_ " + vectorToString(vector2));
```

```
System.out.println("Das_Skalarprodukt_list:_ " + d);
```

# Übungsblatt 5 - Aufgabe 1 a)

# Übungsblatt 5 - Aufgabe 1 a)

- Nun noch Ausgabe der Vektoren:

## Übungsblatt 5 - Aufgabe 1 a)

- Nun noch Ausgabe der Vektoren:

```
public static String vectorToString(int[] vector) {
```

}

# Übungsblatt 5 - Aufgabe 1 a)

- Nun noch Ausgabe der Vektoren:

[illegible]



# Übungsblatt 5 - Aufgabe 1 a)

- Nun noch Ausgabe der Vektoren:

```
public static String vectorToString(int[] vector) {  
    String result = "(";  
    for (int i = 0; i < vector.length; i++) {  
  
    }  
  
}
```

# Übungsblatt 5 - Aufgabe 1 a)

- Nun noch Ausgabe der Vektoren:

```
public static String vectorToString(int[] vector) {  
    String result = "(";  
    for (int i = 0; i < vector.length; i++) {  
        if (i > 0) result += ", ";  
    }  
}
```

## Übungsblatt 5 - Aufgabe 1 a)

- Nun noch Ausgabe der Vektoren:

```
public static String vectorToString(int[] vector) {  
    String result = "(";  
    for (int i = 0; i < vector.length; i++) {  
        if (i > 0) result += ", ";  
        result += vector[i];  
    }  
  
}
```

## Übungsblatt 5 - Aufgabe 1 a)

- Nun noch Ausgabe der Vektoren:

```
public static String vectorToString(int[] vector) {  
    String result = "(";  
    for (int i = 0; i < vector.length; i++) {  
        if (i > 0) result += ", ";  
        result += vector[i];  
    }  
    return result + ")";  
}
```

# Übungsblatt 5 - Aufgabe 1 a)

# Übungsblatt 5 - Aufgabe 1 a)

- Als beispielhafte Ausgabe für `java DotProduct`:

# Übungsblatt 5 - Aufgabe 1 a)

- Als beispielhafte Ausgabe für `java DotProduct`:

```
Vector1: (-2, 0, 0, 0, -2)
```

```
Vector2: (4, -4, -3, 4, 2)
```

```
Das Skalarprodukt ist: -12
```

# Übungsblatt 5 - Aufgabe 1 b)



# Übungsblatt 5 - Aufgabe 1 b)

- Der volle Code befindet sich hier: [Transpose.java](#).

# Übungsblatt 5 - Aufgabe 1 b)

- Der volle Code befindet sich hier: [Transpose.java](#).
- Wir vertauschen! Aus  $a_{i,j}$  wird  $a'_{j,i}$ .

# Übungsblatt 5 - Aufgabe 1 b)

- Der volle Code befindet sich hier: [Transpose.java](#).
- Wir vertauschen! Aus  $a_{i,j}$  wird  $a'_{j,i}$ .
- Oder als Java-Code:

# Übungsblatt 5 - Aufgabe 1 b)

- Der volle Code befindet sich hier: [Transpose.java](#).
- Wir vertauschen! Aus  $a_{i,j}$  wird  $a'_{j,i}$ .
- Oder als Java-Code:

```
int[][] transposed = new int[M][N];
```

# Übungsblatt 5 - Aufgabe 1 b)

- Der volle Code befindet sich hier: [Transpose.java](#).
- Wir vertauschen! Aus  $a_{i,j}$  wird  $a'_{j,i}$ .
- Oder als Java-Code:

```
int[][] transposed = new int[M][N];  
for (int i = 0; i < M; i++) {  
  
  
  
}
```

# Übungsblatt 5 - Aufgabe 1 b)

- Der volle Code befindet sich hier: [Transpose.java](#).
- Wir vertauschen! Aus  $a_{i,j}$  wird  $a'_{j,i}$ .
- Oder als Java-Code:

```
int[][] transposed = new int[M][N];  
for (int i = 0; i < M; i++) {  
    for (int j = 0; j < N; j++) {  
  
    }  
  
}
```

# Übungsblatt 5 - Aufgabe 1 b)

- Der volle Code befindet sich hier: [Transpose.java](#).
- Wir vertauschen! Aus  $a_{i,j}$  wird  $a'_{j,i}$ .
- Oder als Java-Code:

```
int[][] transposed = new int[M][N];
for (int i = 0; i < M; i++) {
    for (int j = 0; j < N; j++) {
        transposed[i][j] = matrix[j][i];
        System.out.print(transposed[i][j] + " ");
    }
}
```

# Übungsblatt 5 - Aufgabe 1 b)

- Der volle Code befindet sich hier: [Transpose.java](#).
- Wir vertauschen! Aus  $a_{i,j}$  wird  $a'_{j,i}$ .
- Oder als Java-Code:

```
int[][] transposed = new int[M][N];
for (int i = 0; i < M; i++) {
    for (int j = 0; j < N; j++) {
        transposed[i][j] = matrix[j][i];
        System.out.print(transposed[i][j] + " ");
    }
    System.out.println("");
}
```



# Übungsblatt 5 - Aufgabe 1 b)

# Übungsblatt 5 - Aufgabe 1 b)

- Als beispielhafte Ausgabe für `java` Transpose :

# Übungsblatt 5 - Aufgabe 1 b)

- Als beispielhafte Ausgabe für `java Transpose`:

Erzeugte Matrix:

```
9 2 1 0 3
7 5 5 5 5
4 5 9 1 2
6 5 8 9 0
```

Transponierte Matrix:

```
9 7 4 6
2 5 5 5
1 5 9 8
0 5 1 9
3 5 2 0
```

# Übungsblatt 5 - Aufgabe 2

# Übungsblatt 5 - Aufgabe 2

- Der volle Code befindet sich hier: [MatrixMultiplication.java](#).

# Übungsblatt 5 - Aufgabe 2

- Der volle Code befindet sich hier: [MatrixMultiplication.java](#).
- Zunächst eine zweite Matrix:

# Übungsblatt 5 - Aufgabe 2

- Der volle Code befindet sich hier: [MatrixMultiplication.java](#).
- Zunächst eine zweite Matrix:

```
int[][] matrix2 = new int[M][N];
```





# Übungsblatt 5 - Aufgabe 2

- Der volle Code befindet sich hier: [MatrixMultiplication.java](#).
- Zunächst eine zweite Matrix:

```
int[][] matrix2 = new int[M][N];  
for(int i = 0; i < M; i++) {  
    for (int j = 0; j < N; j++) {  
  
    }  
  
}
```

# Übungsblatt 5 - Aufgabe 2

- Der volle Code befindet sich hier: [MatrixMultiplication.java](#).
- Zunächst eine zweite Matrix:

```
int[][] matrix2 = new int[M][N];  
for(int i = 0; i < M; i++) {  
    for (int j = 0; j < N; j++) {  
        matrix2[i][j] = (int) (Math.random() * 10);  
    }  
}
```

# Übungsblatt 5 - Aufgabe 2

- Der volle Code befindet sich hier: [MatrixMultiplication.java](#).
- Zunächst eine zweite Matrix:

```
int[][] matrix2 = new int[M][N];
for(int i = 0; i < M; i++) {
    for (int j = 0; j < N; j++) {
        matrix2[i][j] = (int) (Math.random() * 10);
        System.out.print(matrix2[i][j] + " ");
    }
}
```

# Übungsblatt 5 - Aufgabe 2

- Der volle Code befindet sich hier: [MatrixMultiplication.java](#).
- Zunächst eine zweite Matrix:

```
int[][] matrix2 = new int[M][N];
for(int i = 0; i < M; i++) {
    for (int j = 0; j < N; j++) {
        matrix2[i][j] = (int) (Math.random() * 10);
        System.out.print(matrix2[i][j] + " ");
    }
    System.out.println("");
}
```

## Übungsblatt 5 - Aufgabe 2

- Der volle Code befindet sich hier: [MatrixMultiplication.java](#).
- Zunächst eine zweite Matrix:

```
int[][] matrix2 = new int[M][N];
for(int i = 0; i < M; i++) {
    for (int j = 0; j < N; j++) {
        matrix2[i][j] = (int) (Math.random() * 10);
        System.out.print(matrix2[i][j] + " ");
    }
    System.out.println("");
}
System.out.println("");
```

# Übungsblatt 5 - Aufgabe 2

# Übungsblatt 5 - Aufgabe 2

- Und nun multiplizieren wir! Da wir davon ausgehen können, dass es klappt, gilt:

# Übungsblatt 5 - Aufgabe 2

- Und nun multiplizieren wir! Da wir davon ausgehen können, dass es klappt, gilt:

$$c_{i,j} = \sum_{k=1}^m a_{i,k} \cdot b_{k,j}.$$



# Übungsblatt 5 - Aufgabe 2

- Und nun multiplizieren wir! Da wir davon ausgehen können, dass es klappt, gilt:

$$c_{i,j} = \sum_{k=1}^m a_{i,k} \cdot b_{k,j}.$$

```
int[][] result = new int[N][N];
```

## Übungsblatt 5 - Aufgabe 2

- Und nun multiplizieren wir! Da wir davon ausgehen können, dass es klappt, gilt:

$$c_{i,j} = \sum_{k=1}^m a_{i,k} \cdot b_{k,j}.$$

```
int[][] result = new int[N][N];  
for(int i = 0; i < N; i++) {
```

```
}
```

# Übungsblatt 5 - Aufgabe 2

- Und nun multiplizieren wir! Da wir davon ausgehen können, dass es klappt, gilt:

$$c_{i,j} = \sum_{k=1}^m a_{i,k} \cdot b_{k,j}.$$

```
int[][] result = new int[N][N];  
for(int i = 0; i < N; i++) {  
    for (int j = 0; j < N; j++) {  
  
  
  
  
  
  
    }  
}
```

# Übungsblatt 5 - Aufgabe 2

- Und nun multiplizieren wir! Da wir davon ausgehen können, dass es klappt, gilt:

$$c_{i,j} = \sum_{k=1}^m a_{i,k} \cdot b_{k,j}.$$

```
int[][] result = new int[N][N];  
for(int i = 0; i < N; i++) {  
    for (int j = 0; j < N; j++) {  
        int tmp = 0;  
  
        }  
    }
```

# Übungsblatt 5 - Aufgabe 2

- Und nun multiplizieren wir! Da wir davon ausgehen können, dass es klappt, gilt:

$$c_{i,j} = \sum_{k=1}^m a_{i,k} \cdot b_{k,j}.$$

```
int[][] result = new int[N][N];
for(int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        int tmp = 0;
        for (int k = 0; k < M; k++) {

        }

    }

}
```

# Übungsblatt 5 - Aufgabe 2

- Und nun multiplizieren wir! Da wir davon ausgehen können, dass es klappt, gilt:

$$c_{i,j} = \sum_{k=1}^m a_{i,k} \cdot b_{k,j}.$$

```
int[][] result = new int[N][N];
for(int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        int tmp = 0;
        for (int k = 0; k < M; k++) {
            tmp += matrix1[i][k] * matrix2[k][j];
        }
    }
}
```

# Übungsblatt 5 - Aufgabe 2

- Und nun multiplizieren wir! Da wir davon ausgehen können, dass es klappt, gilt:

$$c_{i,j} = \sum_{k=1}^m a_{i,k} \cdot b_{k,j}.$$

```
int[][] result = new int[N][N];
for(int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        int tmp = 0;
        for (int k = 0; k < M; k++) {
            tmp += matrix1[i][k] * matrix2[k][j];
        }
        result[i][j] = tmp;
    }
}
```

## Übungsblatt 5 - Aufgabe 2

- Und nun multiplizieren wir! Da wir davon ausgehen können, dass es klappt, gilt:

$$c_{i,j} = \sum_{k=1}^m a_{i,k} \cdot b_{k,j}.$$

```
int[][] result = new int[N][N];
for(int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        int tmp = 0;
        for (int k = 0; k < M; k++) {
            tmp += matrix1[i][k] * matrix2[k][j];
        }
        result[i][j] = tmp;
        System.out.print(result[i][j] + "_");
    }
}
```



## Übungsblatt 5 - Aufgabe 2

- Und nun multiplizieren wir! Da wir davon ausgehen können, dass es klappt, gilt:

$$c_{i,j} = \sum_{k=1}^m a_{i,k} \cdot b_{k,j}.$$

```
int[][] result = new int[N][N];
for(int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        int tmp = 0;
        for (int k = 0; k < M; k++) {
            tmp += matrix1[i][k] * matrix2[k][j];
        }
        result[i][j] = tmp;
        System.out.print(result[i][j] + "_");
    }
    System.out.println();
}
```

# Übungsblatt 5 - Aufgabe 2

# Übungsblatt 5 - Aufgabe 2

- Als beispielhafte (formatierte) Ausgabe für `java MatrixMultiplication`:

# Übungsblatt 5 - Aufgabe 2

- Als beispielhafte (formatierte) Ausgabe für `java MatrixMultiplication`:

Matrix 1:

```
0 4 3 1 2 2 2 1 9
8 3 7 1 9 0 1 6 1
4 9 2 2 6 6 0 6 6
```

Matrix 2:

```
6 7 8
6 7 5
2 0 2
3 2 4
```

```
5 1 8
5 0 7
9 3 4
0 6 3
5 5 8
```

Ergebnismatrix:

```
116 89 143
142 132 199
178 167 245
```

# Übungsblatt 5 - Zusatzaufgabe a)

## Übungsblatt 5 - Zusatzaufgabe a)

- Wir machen eine Raute ([Rhombus.java](#)).

## Übungsblatt 5 - Zusatzaufgabe a)

- Wir machen eine Raute ([Rhombus.java](#)).
- Naiv, können wir die bekannte Dreiecksstruktur nehmen und ein Dreieck für die obere und eines für die untere Hälfte basteln:

## Übungsblatt 5 - Zusatzaufgabe a)

- Wir machen eine Raute ([Rhombus.java](#)).
- Naiv, können wir die bekannte Dreiecksstruktur nehmen und ein Dreieck für die obere und eines für die untere Hälfte basteln:

```
// oben
```

```
for(int i = 1; i <= N; i++) {
```

```
}
```



## Übungsblatt 5 - Zusatzaufgabe a)

- Wir machen eine Raute ([Rhombus.java](#)).
- Naiv, können wir die bekannte Dreiecksstruktur nehmen und ein Dreieck für die obere und eines für die untere Hälfte basteln:

```
// oben
```

```
for(int i = 1; i <= N; i++) {  
    for(int j = 1; j <= N-i; j++) System.out.print("_");
```

```
}
```

## Übungsblatt 5 - Zusatzaufgabe a)

- Wir machen eine Raute ([Rhombus.java](#)).
- Naiv, können wir die bekannte Dreiecksstruktur nehmen und ein Dreieck für die obere und eines für die untere Hälfte basteln:

```
// oben
```

```
for(int i = 1; i <= N; i++) {  
    for(int j = 1; j <= N-i; j++) System.out.print("_");  
  
    for(int j = 1; j <= i; j++) {  
  
    }  
  
}
```

## Übungsblatt 5 - Zusatzaufgabe a)

- Wir machen eine Raute ([Rhombus.java](#)).
- Naiv, können wir die bekannte Dreiecksstruktur nehmen und ein Dreieck für die obere und eines für die untere Hälfte basteln:

```
// oben
```

```
for(int i = 1; i <= N; i++) {  
    for(int j = 1; j <= N-i; j++) System.out.print("_");  
  
    for(int j = 1; j <= i; j++) {  
        if((j == 1) || (j == i)) System.out.print("*_");  
    }  
  
}
```

# Übungsblatt 5 - Zusatzaufgabe a)

- Wir machen eine Raute ([Rhombus.java](#)).
- Naiv, können wir die bekannte Dreiecksstruktur nehmen und ein Dreieck für die obere und eines für die untere Hälfte basteln:

```
// oben
```

```
for(int i = 1; i <= N; i++) {  
    for(int j = 1; j <= N-i; j++) System.out.print("_");  
  
    for(int j = 1; j <= i; j++) {  
        if((j == 1) || (j == i)) System.out.print("*_");  
        else System.out.print("_");  
    }  
  
}
```

# Übungsblatt 5 - Zusatzaufgabe a)

- Wir machen eine Raute ([Rhombus.java](#)).
- Naiv, können wir die bekannte Dreiecksstruktur nehmen und ein Dreieck für die obere und eines für die untere Hälfte basteln:

```
// oben
for(int i = 1; i <= N; i++) {
    for(int j = 1; j <= N-i; j++) System.out.print("_");

    for(int j = 1; j <= i; j++) {
        if((j == 1) || (j == i)) System.out.print("*_");
        else System.out.print("_");
    }
    System.out.println("");
}
```

# Übungsblatt 5 - Zusatzaufgabe a)

# Übungsblatt 5 - Zusatzaufgabe a)

- Für unten spiegeln wir:

# Übungsblatt 5 - Zusatzaufgabe a)

- Für unten spiegeln wir:

```
for(int i = N; i > 1; i--) {
```

```
}
```



# Übungsblatt 5 - Zusatzaufgabe a)

- Für unten spiegeln wir:

```
for(int i = N; i > 1; i--) {  
    for(int j = i-1; j < N; j++) System.out.print("_");  
  
}
```

# Übungsblatt 5 - Zusatzaufgabe a)

- Für unten spiegeln wir:

```
for(int i = N; i > 1; i--) {  
    for(int j = i-1; j < N; j++) System.out.print("_");  
  
    for(int j = i; j >= 2; j--) {  
  
    }  
  
}
```

# Übungsblatt 5 - Zusatzaufgabe a)

- Für unten spiegeln wir:

```
for(int i = N; i > 1; i--) {  
    for(int j = i-1; j < N; j++) System.out.print("_");  
  
    for(int j = i; j >= 2; j--) {  
        if((j == 2) || (j == i)) System.out.print("*_");  
    }  
  
}
```

# Übungsblatt 5 - Zusatzaufgabe a)

- Für unten spiegeln wir:

```
for(int i = N; i > 1; i--) {  
    for(int j = i-1; j < N; j++) System.out.print("_");  
  
    for(int j = i; j >= 2; j--) {  
        if((j == 2) || (j == i)) System.out.print("*_");  
        else System.out.print("_");  
    }  
  
}
```

# Übungsblatt 5 - Zusatzaufgabe a)

- Für unten spiegeln wir:

```
for(int i = N; i > 1; i--) {  
    for(int j = i-1; j < N; j++) System.out.print("_");  
  
    for(int j = i; j >= 2; j--) {  
        if((j == 2) || (j == i)) System.out.print("*_");  
        else System.out.print("_");  
    }  
    System.out.println("");  
}
```

# Übungsblatt 5 - Zusatzaufgabe a)

## Übungsblatt 5 - Zusatzaufgabe a)

- Negative Zahlen erlauben es, beide Hälften in einer Schleife zu behandeln:

## Übungsblatt 5 - Zusatzaufgabe a)

- Negative Zahlen erlauben es, beide Hälften in einer Schleife zu behandeln:

```
for(int i = N-1; i > -N; i--) {
```

```
}
```



## Übungsblatt 5 - Zusatzaufgabe a)

- Negative Zahlen erlauben es, beide Hälften in einer Schleife zu behandeln:

```
for(int i = N-1; i > -N; i--) {  
    for(int j = 1; j <= Math.abs(i); j++)
```

```
}
```

## Übungsblatt 5 - Zusatzaufgabe a)

- Negative Zahlen erlauben es, beide Hälften in einer Schleife zu behandeln:

```
for(int i = N-1; i > -N; i--) {  
    for(int j = 1; j <= Math.abs(i); j++)  
        System.out.print("_");
```

```
}
```

## Übungsblatt 5 - Zusatzaufgabe a)

- Negative Zahlen erlauben es, beide Hälften in einer Schleife zu behandeln:

```
for(int i = N-1; i > -N; i--) {  
    for(int j = 1; j <= Math.abs(i); j++)  
        System.out.print("_");  
  
    for(int j = 1; j <= N - Math.abs(i); j++) {  
  
    }  
  
}
```

## Übungsblatt 5 - Zusatzaufgabe a)

- Negative Zahlen erlauben es, beide Hälften in einer Schleife zu behandeln:

```
for(int i = N-1; i > -N; i--) {  
    for(int j = 1; j <= Math.abs(i); j++)  
        System.out.print("_");  
  
    for(int j = 1; j <= N - Math.abs(i); j++) {  
        if((j == 1) || (j == N - Math.abs(i)))  
  
            else  
  
    }  
  
}
```

## Übungsblatt 5 - Zusatzaufgabe a)

- Negative Zahlen erlauben es, beide Hälften in einer Schleife zu behandeln:

```
for(int i = N-1; i > -N; i--) {  
    for(int j = 1; j <= Math.abs(i); j++)  
        System.out.print("_");  
  
    for(int j = 1; j <= N - Math.abs(i); j++) {  
        if((j == 1) || (j == N - Math.abs(i)))  
            System.out.print("*_");  
        else  
            System.out.print("_");  
    }  
}
```

## Übungsblatt 5 - Zusatzaufgabe a)

- Negative Zahlen erlauben es, beide Hälften in einer Schleife zu behandeln:

```
for(int i = N-1; i > -N; i--) {  
    for(int j = 1; j <= Math.abs(i); j++)  
        System.out.print("_");  
  
    for(int j = 1; j <= N - Math.abs(i); j++) {  
        if((j == 1) || (j == N - Math.abs(i)))  
            System.out.print("*_");  
        else  
            System.out.print("_");  
    }  
    System.out.println("");  
}
```

# Übungsblatt 5 - Zusatzaufgabe a)

- Negative Zahlen erlauben es, beide Hälften in einer Schleife zu behandeln:

```
for(int i = N-1; i > -N; i--) {  
    for(int j = 1; j <= Math.abs(i); j++)  
        System.out.print("_");  
  
    for(int j = 1; j <= N - Math.abs(i); j++) {  
        if((j == 1) || (j == N - Math.abs(i)))  
            System.out.print("*_");  
        else  
            System.out.print("_");  
    }  
    System.out.println("");  
}
```

RhombusNegative.java

# Übungsblatt 5 - Zusatzaufgabe a)

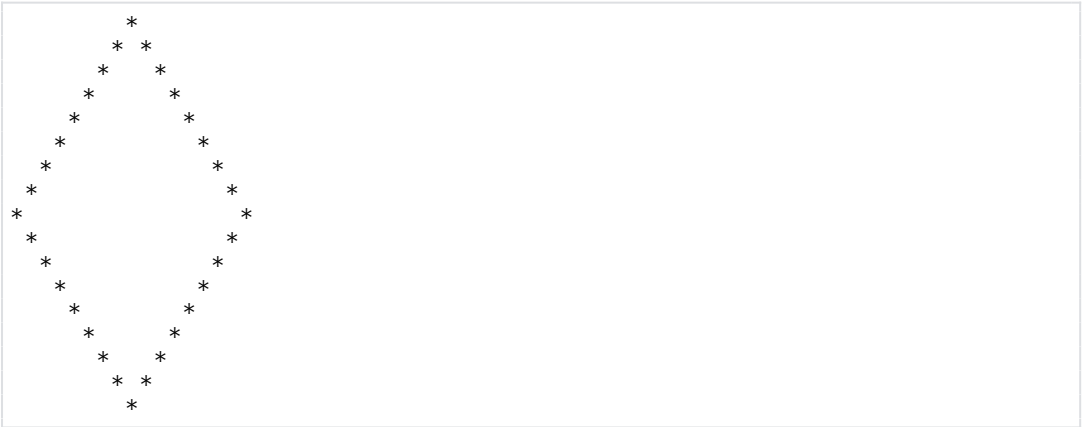


## Übungsblatt 5 - Zusatzaufgabe a)

- So ergibt sich beispielsweise `java RhombusNegative 9`:

# Übungsblatt 5 - Zusatzaufgabe a)

- So ergibt sich beispielsweise `java RhombusNegative 9`:





So tell me...

So tell me. . .

What is it, you *desire*?

*Math!*

So tell me...

What is it, you *desire*?

Math!

Math!

So tell me...

What is it, you *desire*?

Math!

Math!

So tell me...

What is it, you *desire*?

Math!



**Math!**

**Math!**

So tell me...

What is it, you *desire*?

**Math!**

**Math!**

Math!

Math!

L<sup>A</sup>T<sub>E</sub>X-Penguins!

So tell me...

What is it, you *desire*?

Math!

Math!

**Math!**

**Math!**

**L<sup>A</sup>T<sub>E</sub>X-Penguins!**

So tell me...

What is it, you *desire*?

**Math!**

**Math!**

**Math!**

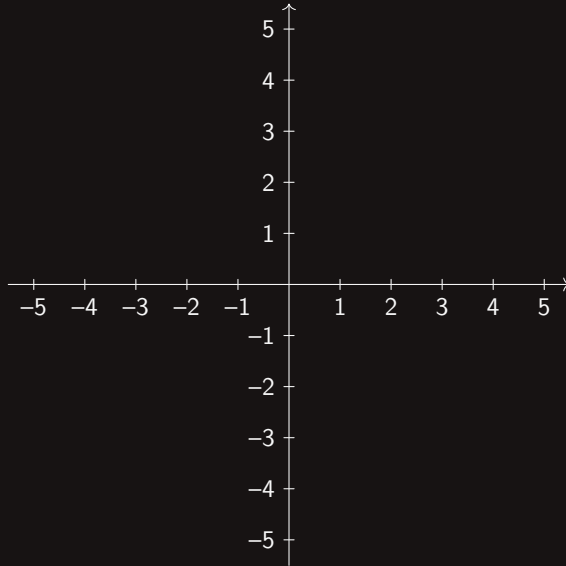
So Math it is...

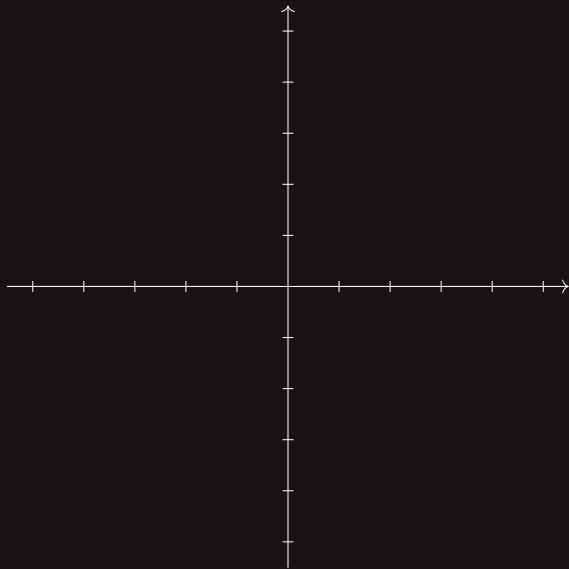
So Math it is...



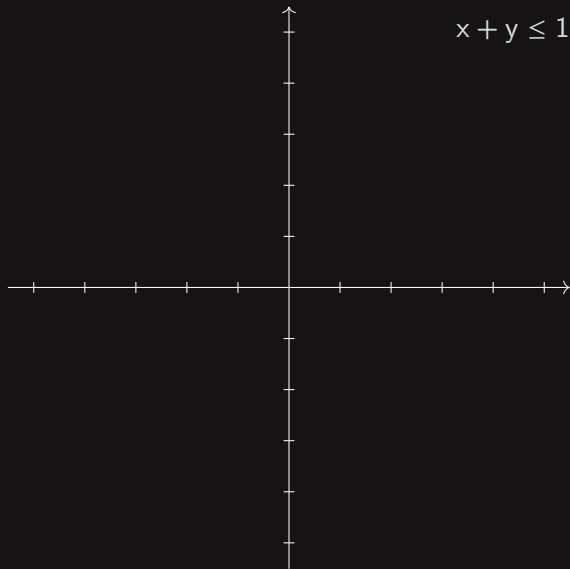
Love > Real ♥ > Math

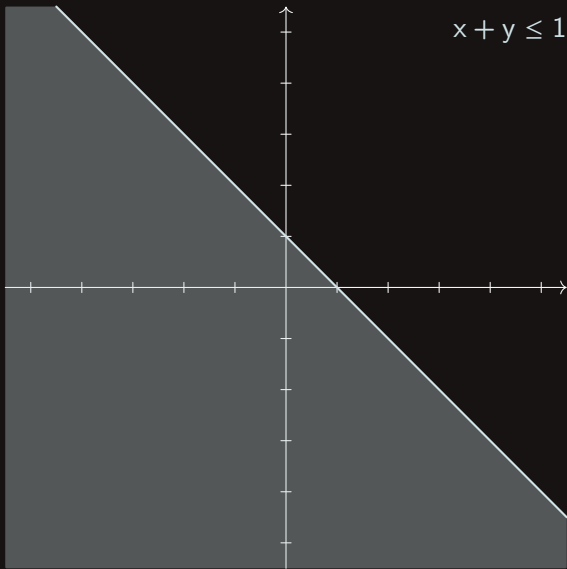
Love > Real ♥ > Math

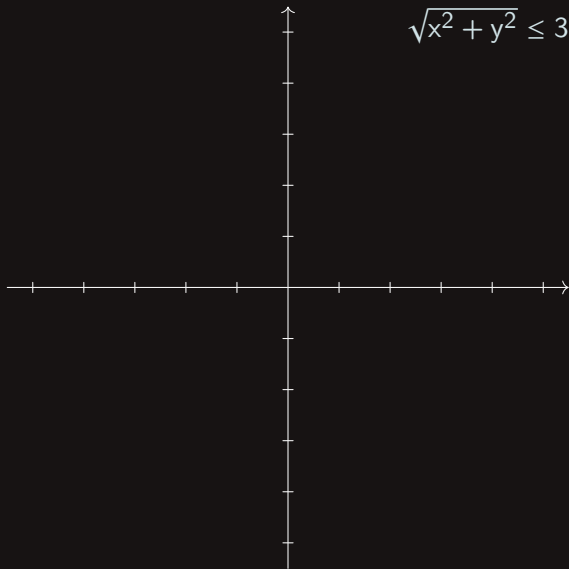




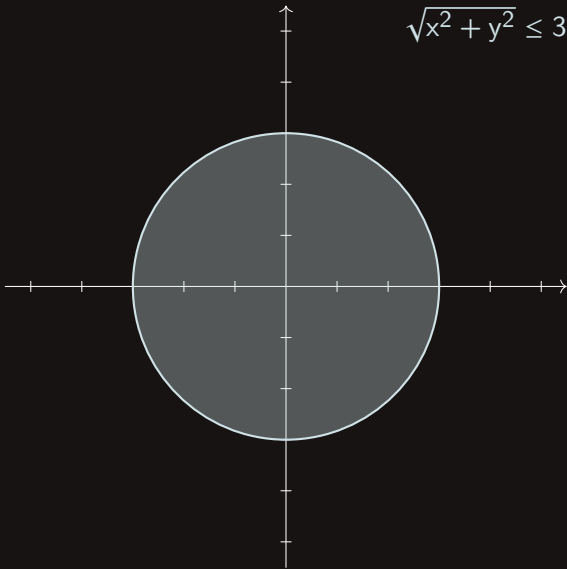




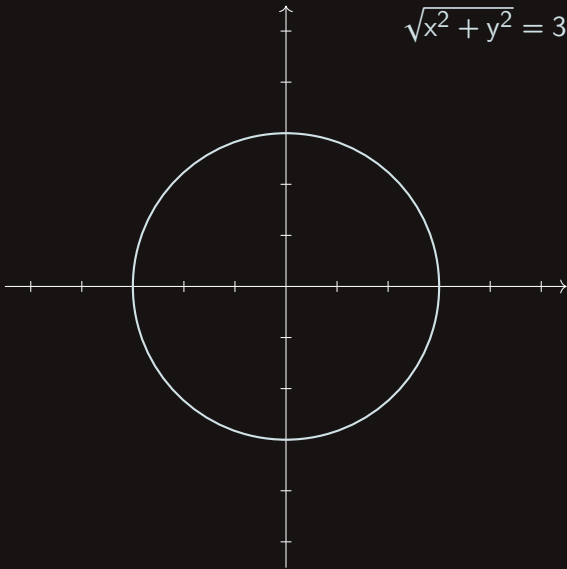


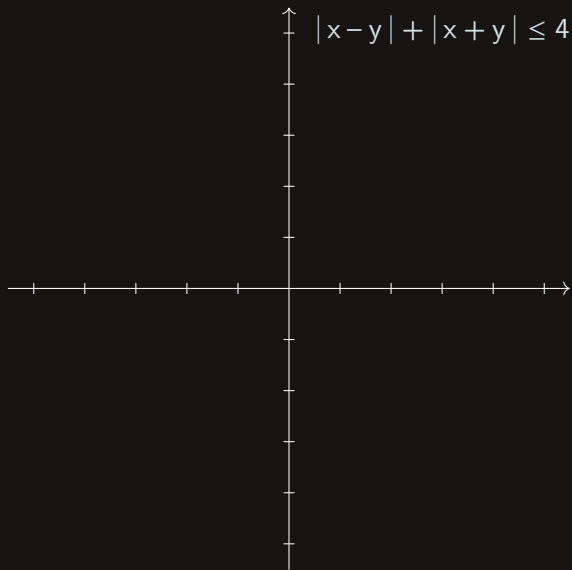


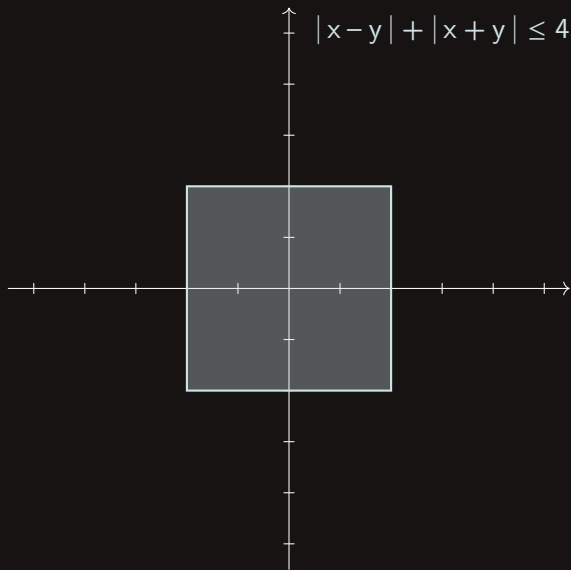
$$\sqrt{x^2 + y^2} \leq 3$$

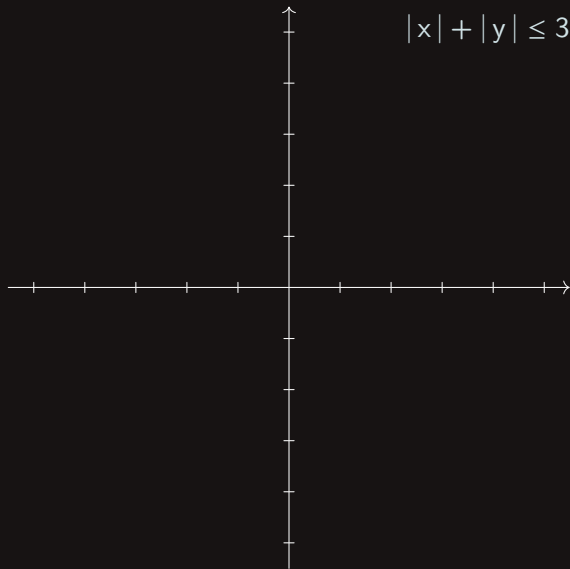


$$\sqrt{x^2 + y^2} = 3$$



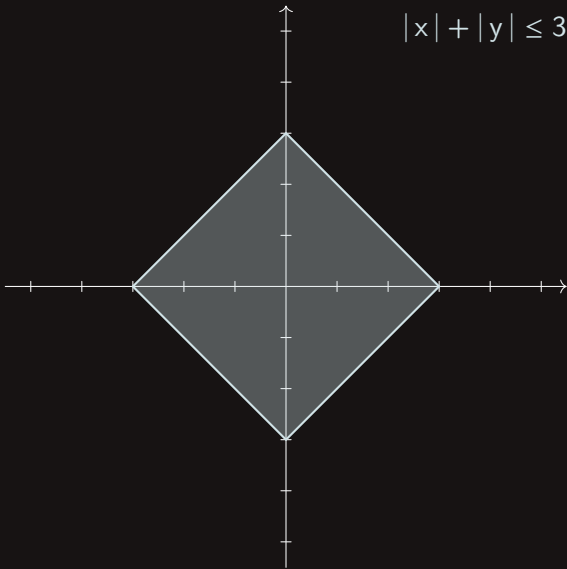








$$|x| + |y| \leq 3$$





Interessiert?

Interessiert?

Signed Distance Functions

# Interessiert?

## Signed Distance Functions

For the horde. For the rabbit hole!

Inigo Quilez • DeepSDF

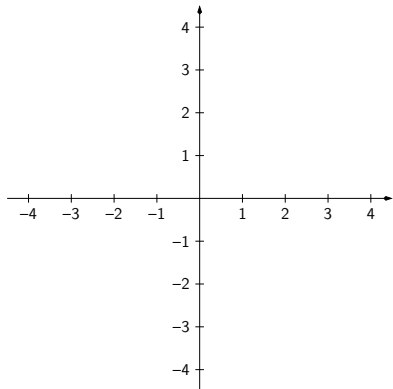


- Wir haben eine Menge zu der alle Punkte zählen für die gilt:  $|x| + |y| = N - 1$ .

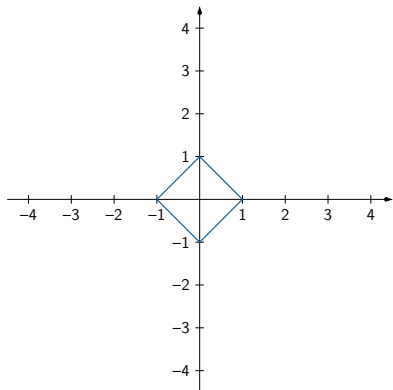
- Wir haben eine Menge zu der alle Punkte zählen für die gilt:  $|x| + |y| = N - 1$ .
- $N - 1$  rührt daher, da wir ja für  $N = 1$  sonst auch in  $y$ -Richtung je 1 hätten.



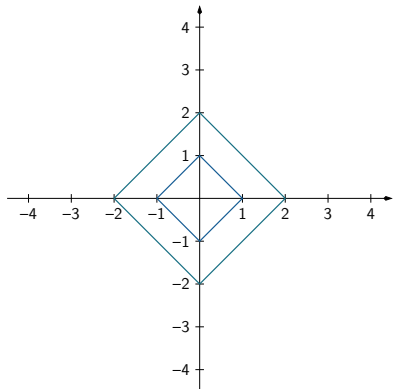
- Wir haben eine Menge zu der alle Punkte zählen für die gilt:  $|x| + |y| = N - 1$ .
- $N - 1$  rührt daher, da wir ja für  $N = 1$  sonst auch in  $y$ -Richtung je 1 hätten.



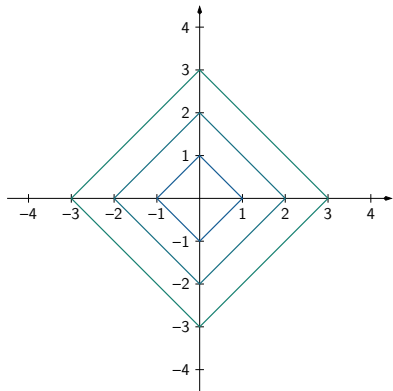
- Wir haben eine Menge zu der alle Punkte zählen für die gilt:  $|x| + |y| = N - 1$ .
- $N - 1$  rührt daher, da wir ja für  $N = 1$  sonst auch in  $y$ -Richtung je 1 hätten.



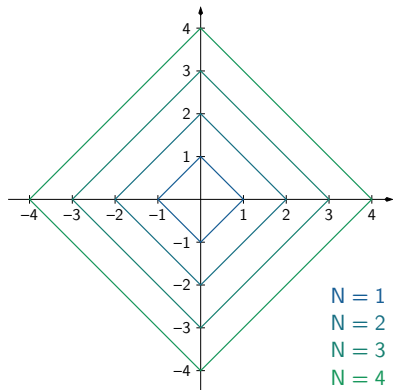
- Wir haben eine Menge zu der alle Punkte zählen für die gilt:  $|x| + |y| = N - 1$ .
- $N - 1$  rührt daher, da wir ja für  $N = 1$  sonst auch in  $y$ -Richtung je 1 hätten.



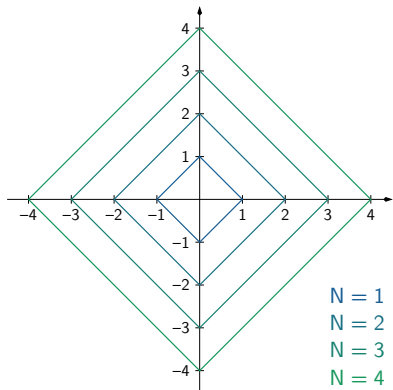
- Wir haben eine Menge zu der alle Punkte zählen für die gilt:  $|x| + |y| = N - 1$ .
- $N - 1$  rührt daher, da wir ja für  $N = 1$  sonst auch in  $y$ -Richtung je 1 hätten.



- Wir haben eine Menge zu der alle Punkte zählen für die gilt:  $|x| + |y| = N - 1$ .
- $N - 1$  rührt daher, da wir ja für  $N = 1$  sonst auch in  $y$ -Richtung je 1 hätten.

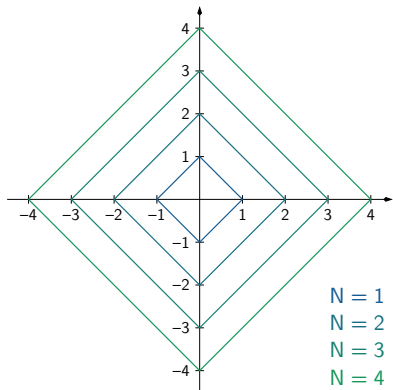


- Wir haben eine Menge zu der alle Punkte zählen für die gilt:  $|x| + |y| = N - 1$ .
- $N - 1$  rührt daher, da wir ja für  $N = 1$  sonst auch in  $y$ -Richtung je 1 hätten.



```
if(args.length != 1) System.exit(1);  
int N = Integer.parseInt(args[0]) - 1;
```

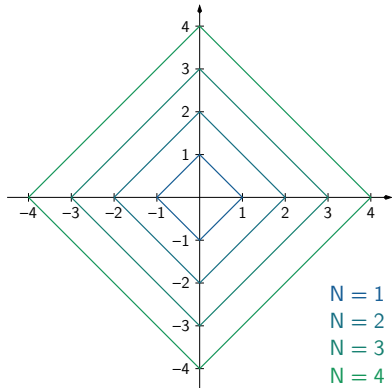
- Wir haben eine Menge zu der alle Punkte zählen für die gilt:  $|x| + |y| = N - 1$ .
- $N - 1$  rührt daher, da wir ja für  $N = 1$  sonst auch in  $y$ -Richtung je 1 hätten.



```
if(args.length != 1) System.exit(1);  
int N = Integer.parseInt(args[0]) - 1;  
// Diskretisieren  
for (int y = N; y >= -N; y--) {
```

```
}
```

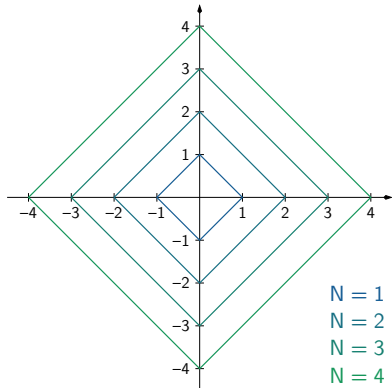
- Wir haben eine Menge zu der alle Punkte zählen für die gilt:  $|x| + |y| = N - 1$ .
- $N - 1$  rührt daher, da wir ja für  $N = 1$  sonst auch in  $y$ -Richtung je 1 hätten.



```
if(args.length != 1) System.exit(1);
int N = Integer.parseInt(args[0]) - 1;
// Diskretisieren
for (int y = N; y >= -N; y--) {
    for (int x = -N; x <= N; x++) {
        }
    }
}
```

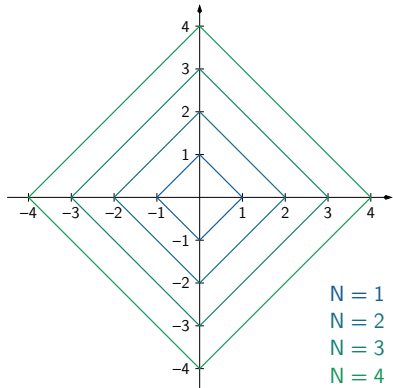


- Wir haben eine Menge zu der alle Punkte zählen für die gilt:  $|x| + |y| = N - 1$ .
- $N - 1$  rührt daher, da wir ja für  $N = 1$  sonst auch in  $y$ -Richtung je 1 hätten.



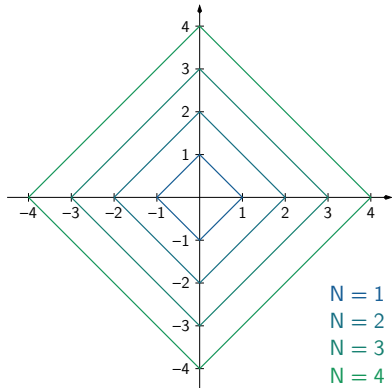
```
if(args.length != 1) System.exit(1);
int N = Integer.parseInt(args[0]) - 1;
// Diskretisieren
for (int y = N; y >= -N; y--) {
    for (int x = -N; x <= N; x++) {
        if (Math.abs(y) + Math.abs(x) == N)
            System.out.print("*");
    }
}
```

- Wir haben eine Menge zu der alle Punkte zählen für die gilt:  $|x| + |y| = N - 1$ .
- $N - 1$  rührt daher, da wir ja für  $N = 1$  sonst auch in  $y$ -Richtung je 1 hätten.



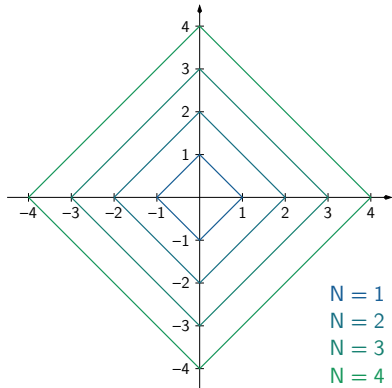
```
if(args.length != 1) System.exit(1);
int N = Integer.parseInt(args[0]) - 1;
// Diskretisieren
for (int y = N; y >= -N; y--) {
    for (int x = -N; x <= N; x++) {
        if (Math.abs(y) + Math.abs(x) == N)
            System.out.print("*");
        else System.out.print("_");
    }
}
```

- Wir haben eine Menge zu der alle Punkte zählen für die gilt:  $|x| + |y| = N - 1$ .
- $N - 1$  rührt daher, da wir ja für  $N = 1$  sonst auch in  $y$ -Richtung je 1 hätten.



```
if(args.length != 1) System.exit(1);
int N = Integer.parseInt(args[0]) - 1;
// Diskretisieren
for (int y = N; y >= -N; y--) {
    for (int x = -N; x <= N; x++) {
        if (Math.abs(y) + Math.abs(x) == N)
            System.out.print("*");
        else System.out.print("_");
    }
    System.out.println();
}
```

- Wir haben eine Menge zu der alle Punkte zählen für die gilt:  $|x| + |y| = N - 1$ .
- $N - 1$  rührt daher, da wir ja für  $N = 1$  sonst auch in  $y$ -Richtung je 1 hätten.



```

if(args.length != 1) System.exit(1);
int N = Integer.parseInt(args[0]) - 1;
// Diskretisieren
for (int y = N; y >= -N; y--) {
    for (int x = -N; x <= N; x++) {
        if (Math.abs(y) + Math.abs(x) == N)
            System.out.print("*");
        else System.out.print("_");
    }
    System.out.println();
}
    
```

RhombusElegant.java

# Übungsblatt 5 - Zusatzaufgabe b)

## Übungsblatt 5 - Zusatzaufgabe b)

- Die Musterlösung ([NestedRhombus.java](#)).

# Übungsblatt 5 - Zusatzaufgabe b)

- Die Musterlösung ([NestedRhombus.java](#)).

```
for(int i = 1; i <= N; i++) {
```

```
}
```

# Übungsblatt 5 - Zusatzaufgabe b)

- Die Musterlösung ([NestedRhombus.java](#)).

```
for(int i = 1; i <= N; i++) {  
    for(int j = 1; j <= N-i; j++) System.out.print("_");
```

```
}
```





# Übungsblatt 5 - Zusatzaufgabe b)

- Die Musterlösung ([NestedRhombus.java](#)).

```
for(int i = 1; i <= N; i++) {  
    for(int j = 1; j <= N-i; j++) System.out.print("_");  
    if(i % 2 == 1) {  
        for (int j = 1; j <= i; j++) {  
  
        }  
    } else {  
  
  
  
    }  
}
```

## Übungsblatt 5 - Zusatzaufgabe b)

- Die Musterlösung ([NestedRhombus.java](#)).

```
for(int i = 1; i <= N; i++) {
    for(int j = 1; j <= N-i; j++) System.out.print("_");
    if(i % 2 == 1) {
        for (int j = 1; j <= i; j++) {
            if (j % 2 == 1) System.out.print("*");
            else System.out.print("_");
        }
    } else {

```

# Übungsblatt 5 - Zusatzaufgabe b)

- Die Musterlösung (NestedRhombus.java).

```
for(int i = 1; i <= N; i++) {  
    for(int j = 1; j <= N-i; j++) System.out.print("_");  
    if(i % 2 == 1) {  
        for (int j = 1; j <= i; j++) {  
            if (j % 2 == 1) System.out.print("*_");  
            else System.out.print("_");  
        }  
    } else {  
        for (int j = 1; j <= i / 2; j++) { // Linke Hälfte  
  
        }  
  
    }  
  
}
```

# Übungsblatt 5 - Zusatzaufgabe b)

- Die Musterlösung ([NestedRhombus.java](#)).

```
for(int i = 1; i <= N; i++) {  
    for(int j = 1; j <= N-i; j++) System.out.print("_");  
    if(i % 2 == 1) {  
        for (int j = 1; j <= i; j++) {  
            if (j % 2 == 1) System.out.print("*");  
            else System.out.print("_");  
        }  
    } else {  
        for (int j = 1; j <= i / 2; j++) { // Linke Hälfte  
            if (j % 2 == 1) System.out.print("*");  
            else System.out.print("_");  
        }  
    }  
}
```

# Übungsblatt 5 - Zusatzaufgabe b)

- Die Musterlösung (NestedRhombus.java).

```
for(int i = 1; i <= N; i++) {  
    for(int j = 1; j <= N-i; j++) System.out.print("_");  
    if(i % 2 == 1) {  
        for (int j = 1; j <= i; j++) {  
            if (j % 2 == 1) System.out.print("*");  
            else System.out.print("_");  
        }  
    } else {  
        for (int j = 1; j <= i / 2; j++) { // Linke Hälfte  
            if (j % 2 == 1) System.out.print("*");  
            else System.out.print("_");  
        }  
        for (int j = i/2; j >= 1; j--) { // Rechte Hälfte  
            if (j % 2 == 1) System.out.print("*");  
            else System.out.print("_");  
        }  
    }  
}
```

# Übungsblatt 5 - Zusatzaufgabe b)

## ■ Die Musterlösung (NestedRhombus.java).

```
for(int i = 1; i <= N; i++) {  
    for(int j = 1; j <= N-i; j++) System.out.print("_");  
    if(i % 2 == 1) {  
        for (int j = 1; j <= i; j++) {  
            if (j % 2 == 1) System.out.print("*");  
            else System.out.print("_");  
        }  
    } else {  
        for (int j = 1; j <= i / 2; j++) { // Linke Hälfte  
            if (j % 2 == 1) System.out.print("*");  
            else System.out.print("_");  
        }  
        for (int j = i/2; j >= 1; j--) { // Rechte Hälfte  
            if (j % 2 == 1) System.out.print("*");  
            else System.out.print("_");  
        }  
    }  
}
```

# Übungsblatt 5 - Zusatzaufgabe b)

## ■ Die Musterlösung (NestedRhombus.java).

```
for(int i = 1; i <= N; i++) {  
    for(int j = 1; j <= N-i; j++) System.out.print("_");  
    if(i % 2 == 1) {  
        for (int j = 1; j <= i; j++) {  
            if (j % 2 == 1) System.out.print("*_");  
            else System.out.print("_");  
        }  
    } else {  
        for (int j = 1; j <= i / 2; j++) { // Linke Hälfte  
            if (j % 2 == 1) System.out.print("*_");  
            else System.out.print("_");  
        }  
        for (int j = i/2; j >= 1; j--) { // Rechte Hälfte  
            if (j % 2 == 1) System.out.print("*_");  
            else System.out.print("_");  
        }  
    }  
    System.out.println("");  
}
```



# Übungsblatt 5 - Zusatzaufgabe b)

# Übungsblatt 5 - Zusatzaufgabe b)

- Die Musterlösung der unteren Hälfte:

# Übungsblatt 5 - Zusatzaufgabe b)

- Die Musterlösung der unteren Hälfte:

```
for(int i = N-1; i >= 1; i--) {
```

```
}
```

# Übungsblatt 5 - Zusatzaufgabe b)

- Die Musterlösung der unteren Hälfte:

```
for(int i = N-1; i >= 1; i--) {  
    for(int j = i; j < N; j++) System.out.print("_");
```

```
}
```

# Übungsblatt 5 - Zusatzaufgabe b)

- Die Musterlösung der unteren Hälfte:

```
for(int i = N-1; i >= 1; i--) {  
    for(int j = i; j < N; j++) System.out.print("_");  
    if(i % 2 == 1) {  
  
        } else {  
  
        }  
}
```

# Übungsblatt 5 - Zusatzaufgabe b)

- Die Musterlösung der unteren Hälfte:

```
for(int i = N-1; i >= 1; i--) {  
    for(int j = i; j < N; j++) System.out.print("_");  
    if(i % 2 == 1) {  
        for (int j = 1; j <= i; j++) {  
  
        }  
    } else {  
  
  
    }  
}
```

# Übungsblatt 5 - Zusatzaufgabe b)

- Die Musterlösung der unteren Hälfte:

```
for(int i = N-1; i >= 1; i--) {  
    for(int j = i; j < N; j++) System.out.print("_");  
    if(i % 2 == 1) {  
        for (int j = 1; j <= i; j++) {  
            if (j % 2 == 1) System.out.print("*_");  
            else System.out.print("_");  
        }  
    } else {  
  
    }  
}
```

# Übungsblatt 5 - Zusatzaufgabe b)

- Die Musterlösung der unteren Hälfte:

```
for(int i = N-1; i >= 1; i--) {  
    for(int j = i; j < N; j++) System.out.print("_");  
    if(i % 2 == 1) {  
        for (int j = 1; j <= i; j++) {  
            if (j % 2 == 1) System.out.print("*");  
            else System.out.print("_");  
        }  
    } else {  
        for (int j = 1; j <= i / 2; j++) { // Linke Hälfte  
  
        }  
  
    }  
  
}
```



# Übungsblatt 5 - Zusatzaufgabe b)

- Die Musterlösung der unteren Hälfte:

```
for(int i = N-1; i >= 1; i--) {  
    for(int j = i; j < N; j++) System.out.print("_");  
    if(i % 2 == 1) {  
        for (int j = 1; j <= i; j++) {  
            if (j % 2 == 1) System.out.print("*_");  
            else System.out.print("_");  
        }  
    } else {  
        for (int j = 1; j <= i / 2; j++) { // Linke Hälfte  
            if (j % 2 == 1) System.out.print("*_");  
            else System.out.print("_");  
        }  
    }  
}
```

# Übungsblatt 5 - Zusatzaufgabe b)

## ■ Die Musterlösung der unteren Hälfte:

```
for(int i = N-1; i >= 1; i--) {  
    for(int j = i; j < N; j++) System.out.print("_");  
    if(i % 2 == 1) {  
        for (int j = 1; j <= i; j++) {  
            if (j % 2 == 1) System.out.print("*_");  
            else System.out.print("_");  
        }  
    } else {  
        for (int j = 1; j <= i / 2; j++) { // Linke Hälfte  
            if (j % 2 == 1) System.out.print("*_");  
            else System.out.print("_");  
        }  
        for (int j = i/2; j >= 1; j--) { // Rechte Hälfte  
            if (j % 2 == 1) System.out.print("*_");  
            else System.out.print("_");  
        }  
    }  
}
```

# Übungsblatt 5 - Zusatzaufgabe b)

## ■ Die Musterlösung der unteren Hälfte:

```
for(int i = N-1; i >= 1; i--) {  
    for(int j = i; j < N; j++) System.out.print("_");  
    if(i % 2 == 1) {  
        for (int j = 1; j <= i; j++) {  
            if (j % 2 == 1) System.out.print("*_");  
            else System.out.print("_");  
        }  
    } else {  
        for (int j = 1; j <= i / 2; j++) { // Linke Hälfte  
            if (j % 2 == 1) System.out.print("*_");  
            else System.out.print("_");  
        }  
        for (int j = i/2; j >= 1; j--) { // Rechte Hälfte  
            if (j % 2 == 1) System.out.print("*_");  
            else System.out.print("_");  
        }  
    }  
}
```

# Übungsblatt 5 - Zusatzaufgabe b)

## ■ Die Musterlösung der unteren Hälfte:

```
for(int i = N-1; i >= 1; i--) {  
    for(int j = i; j < N; j++) System.out.print("_");  
    if(i % 2 == 1) {  
        for (int j = 1; j <= i; j++) {  
            if (j % 2 == 1) System.out.print("*_");  
            else System.out.print("_");  
        }  
    } else {  
        for (int j = 1; j <= i / 2; j++) { // Linke Hälfte  
            if (j % 2 == 1) System.out.print("*_");  
            else System.out.print("_");  
        }  
        for (int j = i/2; j >= 1; j--) { // Rechte Hälfte  
            if (j % 2 == 1) System.out.print("*_");  
            else System.out.print("_");  
        }  
    }  
    System.out.println("");  
}
```



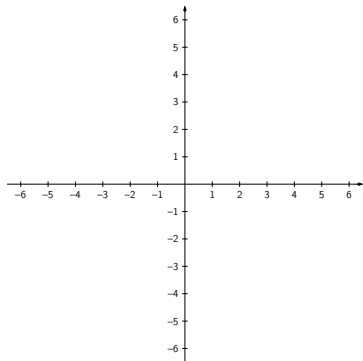
- Wir möchten nun alle Rauten durch  $|x| + |y| \leq N - 1$ .

- Wir möchten nun alle Rauten durch  $|x| + |y| \leq N - 1$ .
- Da wir aber nur je die mit Längen  $-4$  haben wollen:  $|x| + |y| \equiv N - 1 \pmod{4}$ .

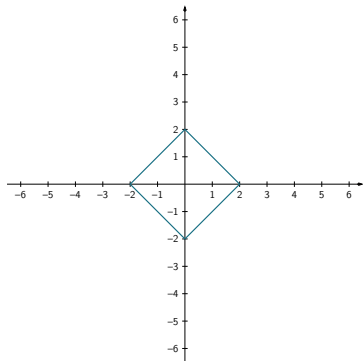
- Wir möchten nun alle Rauten durch  $|x| + |y| \leq N - 1$ .
- Da wir aber nur je die mit Längen  $-4$  haben wollen:  $|x| + |y| \equiv N - 1 \pmod{4}$ . Mathe ist Liebe.



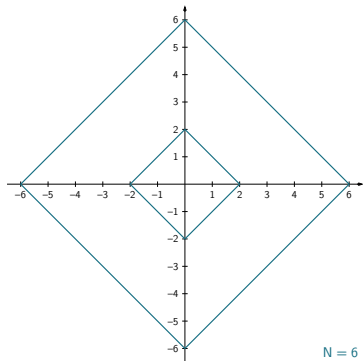
- Wir möchten nun alle Rauten durch  $|x| + |y| \leq N - 1$ .
- Da wir aber nur je die mit Längen  $-4$  haben wollen:  $|x| + |y| \equiv N - 1 \pmod{4}$ . Mathe ist Liebe.



- Wir möchten nun alle Rauten durch  $|x| + |y| \leq N - 1$ .
- Da wir aber nur je die mit Längen  $-4$  haben wollen:  $|x| + |y| \equiv N - 1 \pmod{4}$ . Mathe ist Liebe.

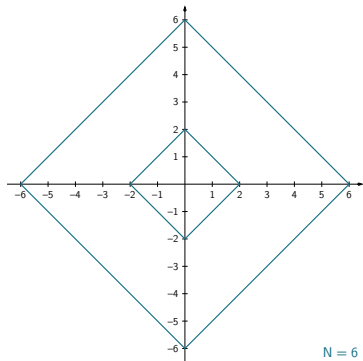


- Wir möchten nun alle Rauten durch  $|x| + |y| \leq N - 1$ .
- Da wir aber nur je die mit Längen  $-4$  haben wollen:  $|x| + |y| \equiv N - 1 \pmod{4}$ . Mathe ist Liebe.



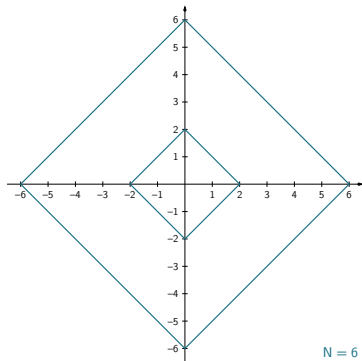
$N = 6$

- Wir möchten nun alle Rauten durch  $|x| + |y| \leq N - 1$ .
- Da wir aber nur je die mit Längen  $-4$  haben wollen:  $|x| + |y| \equiv N - 1 \pmod{4}$ . Mathe ist Liebe.



```
if (args.length != 1) System.exit(1);  
int N = Integer.parseInt(args[0]) - 1;
```

- Wir möchten nun alle Rauten durch  $|x| + |y| \leq N - 1$ .
- Da wir aber nur je die mit Längen  $-4$  haben wollen:  $|x| + |y| \equiv N - 1 \pmod{4}$ . Mathe ist Liebe.

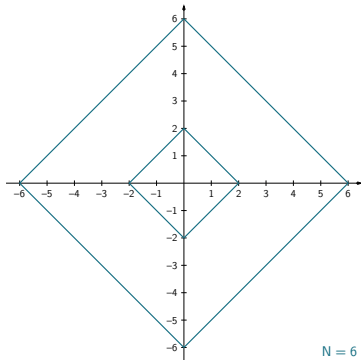


```
if (args.length != 1) System.exit(1);  
int N = Integer.parseInt(args[0]) - 1;
```

```
for (int y = N; y >= -N; y--) {
```

```
}
```

- Wir möchten nun alle Rauten durch  $|x| + |y| \leq N - 1$ .
- Da wir aber nur je die mit Längen  $-4$  haben wollen:  $|x| + |y| \equiv N - 1 \pmod{4}$ . Mathe ist Liebe.



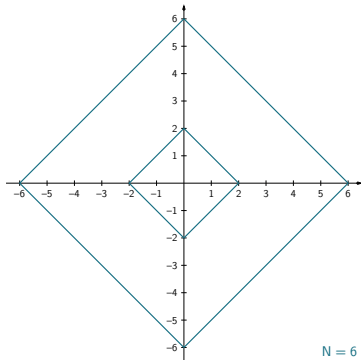
```
if (args.length != 1) System.exit(1);  
int N = Integer.parseInt(args[0]) - 1;
```

```
for (int y = N; y >= -N; y--) {  
    for (int x = -N; x <= N; x++) {
```

```
}
```

```
}
```

- Wir möchten nun alle Rauten durch  $|x| + |y| \leq N - 1$ .
- Da wir aber nur je die mit Längen  $-4$  haben wollen:  $|x| + |y| \equiv N - 1 \pmod{4}$ . Mathe ist Liebe.

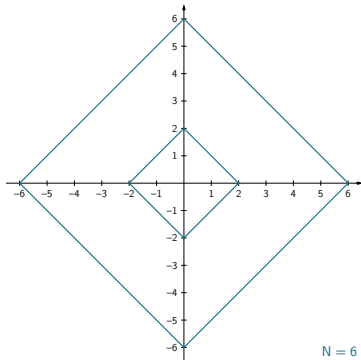


```
if (args.length != 1) System.exit(1);
int N = Integer.parseInt(args[0]) - 1;

for (int y = N; y >= -N; y--) {
    for (int x = -N; x <= N; x++) {
        int radius = Math.abs(y) + Math.abs(x);

    }
}
```

- Wir möchten nun alle Rauten durch  $|x| + |y| \leq N - 1$ .
- Da wir aber nur je die mit Längen  $-4$  haben wollen:  $|x| + |y| \equiv N - 1 \pmod{4}$ . Mathe ist Liebe.

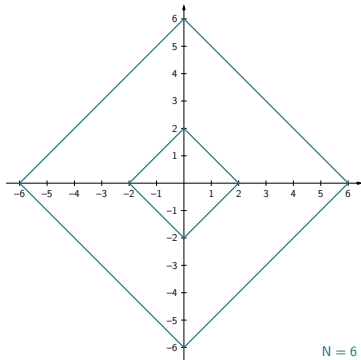


```
if (args.length != 1) System.exit(1);
int N = Integer.parseInt(args[0]) - 1;

for (int y = N; y >= -N; y--) {
    for (int x = -N; x <= N; x++) {
        int radius = Math.abs(y) + Math.abs(x);
        if (radius <= N && radius % 4 == N % 4)
            System.out.print("*");
    }
}
```



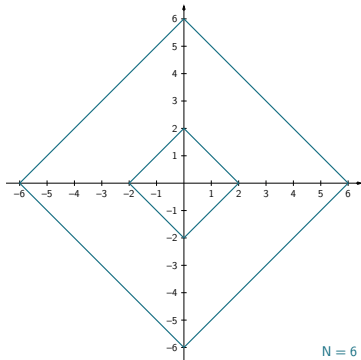
- Wir möchten nun alle Rauten durch  $|x| + |y| \leq N - 1$ .
- Da wir aber nur je die mit Längen  $-4$  haben wollen:  $|x| + |y| \equiv N - 1 \pmod{4}$ . Mathe ist Liebe.



```
if (args.length != 1) System.exit(1);
int N = Integer.parseInt(args[0]) - 1;

for (int y = N; y >= -N; y--) {
    for (int x = -N; x <= N; x++) {
        int radius = Math.abs(y) + Math.abs(x);
        if (radius <= N && radius % 4 == N % 4)
            System.out.print("*");
        else System.out.print("_");
    }
}
```

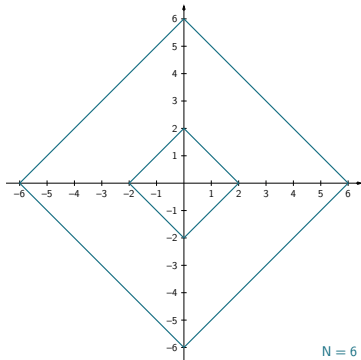
- Wir möchten nun alle Rauten durch  $|x| + |y| \leq N - 1$ .
- Da wir aber nur je die mit Längen  $-4$  haben wollen:  $|x| + |y| \equiv N - 1 \pmod{4}$ . Mathe ist Liebe.



```
if (args.length != 1) System.exit(1);
int N = Integer.parseInt(args[0]) - 1;

for (int y = N; y >= -N; y--) {
    for (int x = -N; x <= N; x++) {
        int radius = Math.abs(y) + Math.abs(x);
        if (radius <= N && radius % 4 == N % 4)
            System.out.print("*");
        else System.out.print("_");
    }
    System.out.println();
}
```

- Wir möchten nun alle Rauten durch  $|x| + |y| \leq N - 1$ .
- Da wir aber nur je die mit Längen  $-4$  haben wollen:  $|x| + |y| \equiv N - 1 \pmod{4}$ . Mathe ist Liebe.



```
if (args.length != 1) System.exit(1);
int N = Integer.parseInt(args[0]) - 1;

for (int y = N; y >= -N; y--) {
    for (int x = -N; x <= N; x++) {
        int radius = Math.abs(y) + Math.abs(x);
        if (radius <= N && radius % 4 == N % 4)
            System.out.print("*");
        else System.out.print("_");
    }
    System.out.println();
}
```

NestedRhombusElegant.java

# Can we do it in one line?

# Can we do it in one line?

- **Nicht klausurrelevant**, aber durch Anfrage ([NestedRhombusOneLine.java](#)):

# Can we do it in one line?

- **Nicht klausurrelevant**, aber durch Anfrage ([NestedRhombusOneLine.java](#)):

```
System.out.println(args.length != 1 ? "invalid_argument" :
```

```
);
```

# Can we do it in one line?

- **Nicht klausurrelevant**, aber durch Anfrage ([NestedRhombusOneLine.java](#)):

```
System.out.println(args.length != 1 ? "invalid_argument" :  
    IntStream.rangeClosed(-Integer.parseInt(args[0]) + 1, Integer.parseInt(args[0]) - 1)  
  
    );
```

# Can we do it in one line?

- **Nicht klausurrelevant**, aber durch Anfrage ([NestedRhombusOneLine.java](#)):

```
System.out.println(args.length != 1 ? "invalid_argument" :  
    IntStream.rangeClosed(-Integer.parseInt(args[0]) + 1, Integer.parseInt(args[0]) - 1)  
        .boxed()  
  
    );
```



# Can we do it in one line?

- **Nicht klausurrelevant**, aber durch Anfrage ([NestedRhombusOneLine.java](#)):

```
System.out.println(args.length != 1 ? "invalid_argument" :  
    IntStream.rangeClosed(-Integer.parseInt(args[0]) + 1, Integer.parseInt(args[0]) - 1)  
        .boxed()  
        .map(s -> Map.entry(s,  
  
                                ))  
  
    );
```

# Can we do it in one line?

- **Nicht klausurrelevant**, aber durch Anfrage ([NestedRhombusOneLine.java](#)):

```
System.out.println(args.length != 1 ? "invalid_argument" :  
    IntStream.rangeClosed(-Integer.parseInt(args[0]) + 1, Integer.parseInt(args[0]) - 1)  
        .boxed()  
        .map(s -> Map.entry(s, IntStream.rangeClosed(  
            -Integer.parseInt(args[0]) + 1,  
            Integer.parseInt(args[0]) - 1).boxed()))  
        .map(s -> s.getValue())  
  
    )  
    );
```

# Can we do it in one line?

- **Nicht klausurrelevant**, aber durch Anfrage ([NestedRhombusOneLine.java](#)):

```
System.out.println(args.length != 1 ? "invalid_argument" :  
    IntStream.rangeClosed(-Integer.parseInt(args[0]) + 1, Integer.parseInt(args[0]) - 1)  
        .boxed()  
        .map(s -> Map.entry(s, IntStream.rangeClosed(  
            -Integer.parseInt(args[0]) + 1,  
            Integer.parseInt(args[0]) - 1).boxed()))  
        .map(s -> s.getValue())  
        .map(x ->  
  
        )  
    )  
);
```

# Can we do it in one line?

- **Nicht klausurrelevant**, aber durch Anfrage ([NestedRhombusOneLine.java](#)):

```
System.out.println(args.length != 1 ? "invalid_argument" :  
    IntStream.rangeClosed(-Integer.parseInt(args[0]) + 1, Integer.parseInt(args[0]) - 1)  
        .boxed()  
        .map(s -> Map.entry(s, IntStream.rangeClosed(  
            -Integer.parseInt(args[0]) + 1,  
            Integer.parseInt(args[0]) - 1).boxed()))  
        .map(s -> s.getValue()  
            .map(x -> Math.abs(s.getKey()) + Math.abs(x) <= (Integer.parseInt(args[0]) - 1) &&  
                (Math.abs(s.getKey()) + Math.abs(x)) % 4 == (Integer.parseInt(args[0]) - 1) % 4  
                ? "*" : "_")  
            )  
        )  
    );
```

# Can we do it in one line?

- **Nicht klausurrelevant, aber durch Anfrage ([NestedRhombusOneLine.java](#)):**

```
System.out.println(args.length != 1 ? "invalid_argument" :  
    IntStream.rangeClosed(-Integer.parseInt(args[0]) + 1, Integer.parseInt(args[0]) - 1)  
        .boxed()  
        .map(s -> Map.entry(s, IntStream.rangeClosed(  
            -Integer.parseInt(args[0]) + 1,  
            Integer.parseInt(args[0]) - 1).boxed()))  
        .map(s -> s.getValue()  
            .map(x -> Math.abs(s.getKey()) + Math.abs(x) <= (Integer.parseInt(args[0]) - 1) &&  
                (Math.abs(s.getKey()) + Math.abs(x)) % 4 == (Integer.parseInt(args[0]) - 1) % 4  
                ? "*" : "_")  
            ).reduce("", (a, b) -> a + b)  
        )  
    );
```

# Can we do it in one line?

- **Nicht klausurrelevant, aber durch Anfrage (`NestedRhombusOneLine.java`):**

```
System.out.println(args.length != 1 ? "invalid_argument" :
    IntStream.rangeClosed(-Integer.parseInt(args[0]) + 1, Integer.parseInt(args[0]) - 1)
        .boxed()
        .map(s -> Map.entry(s, IntStream.rangeClosed(
            -Integer.parseInt(args[0]) + 1,
            Integer.parseInt(args[0]) - 1).boxed()))
        .map(s -> s.getValue()
            .map(x -> Math.abs(s.getKey()) + Math.abs(x) <= (Integer.parseInt(args[0]) - 1) &&
                (Math.abs(s.getKey()) + Math.abs(x)) % 4 == (Integer.parseInt(args[0]) - 1) % 4
                ? "*" : "_")
            ).reduce("", (a, b) -> a + b)
        )
    .collect(Collectors.joining("\n"))
);
```



- Man kann das Problem auch leicht rekursiv lösen. Das war so aber nicht gedacht.



- **Man kann das Problem auch leicht rekursiv lösen. Das war so aber nicht gedacht.**  
(Man kann jede Zeile auf die Basisfälle „Nichts“, „\*“, „\* \*“, ... reduzieren und somit rekursiv bauen. Auch ne nette one-liner Idee... Was sind aber die Nachteile der Rekursion?)

- Man kann das Problem auch leicht rekursiv lösen. Das war so aber nicht gedacht.  
(Man kann jede Zeile auf die Basisfälle „Nichts“, „\*“, „\* \*“, ... reduzieren und somit rekursiv bauen. Auch ne nette one-liner Idee... Was sind aber die Nachteile der Rekursion?)
- Was würde sich ändern, wenn wir andere Formen wie eine Ellipse möchten oder gar mehrere Formen kombinieren?

- Man kann das Problem auch leicht rekursiv lösen. Das war so aber nicht gedacht.  
(Man kann jede Zeile auf die Basisfälle „Nichts“, „\*“, „\* \*“, ... reduzieren und somit rekursiv bauen. Auch ne nette one-liner Idee... Was sind aber die Nachteile der Rekursion?)
- Was würde sich ändern, wenn wir andere Formen wie eine Ellipse möchten oder gar mehrere Formen kombinieren?
- Ich kann Zusatzaufgaben wie diese nur wärmstens empfehlen. Versucht es!

- Man kann das Problem auch leicht rekursiv lösen. Das war so aber nicht gedacht.  
(Man kann jede Zeile auf die Basisfälle „Nichts“, „\*“, „\* \*“, ... reduzieren und somit rekursiv bauen. Auch ne nette one-liner Idee... Was sind aber die Nachteile der Rekursion?)
- Was würde sich ändern, wenn wir andere Formen wie eine Ellipse möchten oder gar mehrere Formen kombinieren?
- Ich kann Zusatzaufgaben wie diese nur wärmstens empfehlen. Versucht es!
- Wer sich unterfordert fühlt, kann immer noch nach Handicaps oder weiteren Aufgaben fragen.

- Man kann das Problem auch leicht rekursiv lösen. Das war so aber nicht gedacht.  
(Man kann jede Zeile auf die Basisfälle „Nichts“, „\*“, „\* \*“, ... reduzieren und somit rekursiv bauen. Auch ne nette one-liner Idee... Was sind aber die Nachteile der Rekursion?)
- Was würde sich ändern, wenn wir andere Formen wie eine Ellipse möchten oder gar mehrere Formen kombinieren?
- Ich kann Zusatzaufgaben wie diese nur wärmstens empfehlen. Versucht es!
- Wer sich unterfordert fühlt, kann immer noch nach Handicaps oder weiteren Aufgaben fragen.
- Code-Kommentare sollten nicht redundanterweise erklären, was die jeweilige Zeile macht, sondern maximal was die Idee/das Konzept dahinter war. Kommentare sind an sich auch Code! Nur für menschliche „Compiler“. (Oder sowas wie GPT-3.)

# Präsenzaufgabe, zum Auflockern

2

Was tun Sie?!

# Präsenzaufgabe, zum Auflockern

2

Was tun Sie?!

Was produziert der rechts abgebildete Code, wobei:

# Präsenzaufgabe, zum Auflockern

2

Was tun Sie?!

Was produziert der rechts abgebildete Code, wobei:

```
static int a = 5;  
static int b = 10;  
static boolean c = true;
```



# Präsenzaufgabe, zum Auflockern

2

Was tun Sie?!

Was produziert der rechts abgebildete Code, wobei:

```
static int a = 5;  
static int b = 10;  
static boolean c = true;
```

```
public static void main(String[] args) {  
    methode1(b, a);  
    methode2(a, c);  
}  
  
static void methode1(int a, int b) {  
    boolean c = false;  
    b = b * a;  
    System.out.println(a + "␣" + b + "␣" + c);  
}  
  
static void methode2(int b, boolean d) {  
    b = a - b;  
    System.out.println(a + "␣" + b + "␣" +  
                        c + "␣" + d);  
}
```

# Präsenzaufgabe - Lösung

# Präsenzaufgabe - Lösung

- Die Ausgabe lautet:

# Präsenzaufgabe - Lösung

- Die Ausgabe lautet:

```
10 50 false  
5 0 true true
```

# Präsenzaufgabe - Lösung

- Die Ausgabe lautet:

```
10 50 false  
5 0 true true
```

- Warum?

# Präsenzaufgabe - Lösung

- Die Ausgabe lautet:

```
10 50 false  
5 0 true true
```

- Warum? *false* kommt über das lokale *c*,

# Präsenzaufgabe - Lösung

- Die Ausgabe lautet:

```
10 50 false  
5 0 true true
```

- Warum? *false* kommt über das lokale *c*, die *10* über das lokale *a*, welches als Parameter den Wert des globalen *b* erhält.

# Präsenzaufgabe - Lösung

- Die Ausgabe lautet:

```
10 50 false  
5 0 true true
```

- Warum? *false* kommt über das lokale *c*, die *10* über das lokale *a*, welches als Parameter den Wert des globalen *b* erhält. Die *50* über  $b * a$ , wobei hier die globalen Werte verwendet werden



# Präsenzaufgabe - Lösung

- Die Ausgabe lautet:

```
10 50 false  
5 0 true true
```

- Warum? *false* kommt über das lokale *c*, die *10* über das lokale *a*, welches als Parameter den Wert des globalen *b* erhält. Die *50* über *b \* a*, wobei hier die globalen Werte verwendet werden (allerdings heißt *a* jetzt *b* und *b* jetzt *a*).

# Präsenzaufgabe - Lösung

- Die Ausgabe lautet:

```
10 50 false  
5 0 true true
```

- Warum? *false* kommt über das lokale *c*, die *10* über das lokale *a*, welches als Parameter den Wert des globalen *b* erhält. Die *50* über *b \* a*, wobei hier die globalen Werte verwendet werden (allerdings heißt *a* jetzt *b* und *b* jetzt *a*).
- Warum?

# Präsenzaufgabe - Lösung

- Die Ausgabe lautet:

```
10 50 false  
5 0 true true
```

- Warum? *false* kommt über das lokale *c*, die *10* über das lokale *a*, welches als Parameter den Wert des globalen *b* erhält. Die *50* über *b \* a*, wobei hier die globalen Werte verwendet werden (allerdings heißt *a* jetzt *b* und *b* jetzt *a*).
- Warum? Die *5* kommt durch das globale *a*, da es kein lokales Pendant besitzt.

# Präsenzaufgabe - Lösung

- Die Ausgabe lautet:

```
10 50 false  
5 0 true true
```

- Warum? *false* kommt über das lokale *c*, die *10* über das lokale *a*, welches als Parameter den Wert des globalen *b* erhält. Die *50* über *b \* a*, wobei hier die globalen Werte verwendet werden (allerdings heißt *a* jetzt *b* und *b* jetzt *a*).
- Warum? Die *5* kommt durch das globale *a*, da es kein lokales Pendant besitzt. Die *0* daher,

# Präsenzaufgabe - Lösung

- Die Ausgabe lautet:

```
10 50 false  
5 0 true true
```

- Warum? *false* kommt über das lokale *c*, die *10* über das lokale *a*, welches als Parameter den Wert des globalen *b* erhält. Die *50* über *b \* a*, wobei hier die globalen Werte verwendet werden (allerdings heißt *a* jetzt *b* und *b* jetzt *a*).
- Warum? Die *5* kommt durch das globale *a*, da es kein lokales Pendant besitzt. Die *0* daher, dass das *lokale* *b* über die Parameter den Wert des globalen *a* trägt

# Präsenzaufgabe - Lösung

- Die Ausgabe lautet:

```
10 50 false
5 0 true true
```

- Warum? *false* kommt über das lokale *c*, die *10* über das lokale *a*, welches als Parameter den Wert des globalen *b* erhält. Die *50* über *b \* a*, wobei hier die globalen Werte verwendet werden (allerdings heißt *a* jetzt *b* und *b* jetzt *a*).
- Warum? Die *5* kommt durch das globale *a*, da es kein lokales Pendant besitzt. Die *0* daher, dass das lokale *b* über die Parameter den Wert des globalen *a* trägt und somit (informal)  $b = a - a$  gerechnet wird.

# Präsenzaufgabe - Lösung

- Die Ausgabe lautet:

```
10 50 false
5 0 true true
```

- Warum? *false* kommt über das lokale *c*, die *10* über das lokale *a*, welches als Parameter den Wert des globalen *b* erhält. Die *50* über *b \* a*, wobei hier die globalen Werte verwendet werden (allerdings heißt *a* jetzt *b* und *b* jetzt *a*).
- Warum? Die *5* kommt durch das globale *a*, da es kein lokales Pendant besitzt. Die *0* daher, dass das lokale *b* über die Parameter den Wert des globalen *a* trägt und somit (informal)  $b = a - a$  gerechnet wird. Das *d* entspricht dem Wert des globalen *c*,

# Präsenzaufgabe - Lösung

- Die Ausgabe lautet:

```
10 50 false  
5 0 true true
```

- Warum? *false* kommt über das lokale *c*, die *10* über das lokale *a*, welches als Parameter den Wert des globalen *b* erhält. Die *50* über  $b * a$ , wobei hier die globalen Werte verwendet werden (allerdings heißt *a* jetzt *b* und *b* jetzt *a*).
- Warum? Die *5* kommt durch das globale *a*, da es kein lokales Pendant besitzt. Die *0* daher, dass das lokale *b* über die Parameter den Wert des globalen *a* trägt und somit (informal)  $b = a - a$  gerechnet wird. Das *d* entspricht dem Wert des globalen *c*, welches in *methode1* nicht überschrieben sondern überlagert wurde.



# Präzise Antworten: Überlagerung

# Präzise Antworten: Überlagerung

- In Java gibt es verschiedene Sichtbarkeiten.

# Präzise Antworten: Überlagerung

- In Java gibt es verschiedene Sichtbarkeiten. Wir werden uns hier ansehen, was bei gleichnamigen Variablen geschieht.

# Präzise Antworten: Überlagerung

- In Java gibt es verschiedene Sichtbarkeiten. Wir werden uns hier ansehen, was bei gleichnamigen Variablen geschieht.
- Wir werden in diesem Kontext *nicht* auf den Stack eingehen.

# Präzise Antworten: Überlagerung

- In Java gibt es verschiedene Sichtbarkeiten. Wir werden uns hier ansehen, was bei gleichnamigen Variablen geschieht.
- Wir werden in diesem Kontext *nicht* auf den Stack eingehen.
- Grundslegend sprechen wir von *globaler* und *lokaler* Sichtbarkeit einer Variable:

# Präzise Antworten: Überlagerung

- In Java gibt es verschiedene Sichtbarkeiten. Wir werden uns hier ansehen, was bei gleichnamigen Variablen geschieht.
- Wir werden in diesem Kontext *nicht* auf den Stack eingehen.
- Grundsätzlich sprechen wir von *globaler* und *lokaler* Sichtbarkeit einer Variable:

// Global:

# Präzise Antworten: Überlagerung

- In Java gibt es verschiedene Sichtbarkeiten. Wir werden uns hier ansehen, was bei gleichnamigen Variablen geschieht.
- Wir werden in diesem Kontext *nicht* auf den Stack eingehen.
- Grundslegend sprechen wir von *globaler* und *lokaler* Sichtbarkeit einer Variable:

// Global:

```
public static int globaleZahl; // ← Überall wo Klasse sichtbar
```

# Präzise Antworten: Überlagerung

- In Java gibt es verschiedene Sichtbarkeiten. Wir werden uns hier ansehen, was bei gleichnamigen Variablen geschieht.
- Wir werden in diesem Kontext *nicht* auf den Stack eingehen.
- Grundslegend sprechen wir von *globaler* und *lokaler* Sichtbarkeit einer Variable:

// Global:

**public static** int globaleZahl; // ← Überall wo Klasse sichtbar

**private static** int klasseZahl; // ← Überall innerhalb der Klasse



# Präzise Antworten: Überlagerung

- In Java gibt es verschiedene Sichtbarkeiten. Wir werden uns hier ansehen, was bei gleichnamigen Variablen geschieht.
- Wir werden in diesem Kontext *nicht* auf den Stack eingehen.
- Grundslegend sprechen wir von *globaler* und *lokaler* Sichtbarkeit einer Variable:

// Global:

```
public static int globaleZahl; // ← Überall wo Klasse sichtbar  
private static int klasseZahl; // ← Überall innerhalb der Klasse  
public static int main(String[] args){
```

# Präzise Antworten: Überlagerung

- In Java gibt es verschiedene Sichtbarkeiten. Wir werden uns hier ansehen, was bei gleichnamigen Variablen geschieht.
- Wir werden in diesem Kontext *nicht* auf den Stack eingehen.
- Grundslegend sprechen wir von *globaler* und *lokaler* Sichtbarkeit einer Variable:

```
// Global:  
public static int globaleZahl; // ← Überall wo Klasse sichtbar  
private static int klasseZahl; // ← Überall innerhalb der Klasse  
public static int main(String[] args){  
    // Lokal:  
    int lokaleZahl;
```

# Präzise Antworten: Überlagerung

- In Java gibt es verschiedene Sichtbarkeiten. Wir werden uns hier ansehen, was bei gleichnamigen Variablen geschieht.
- Wir werden in diesem Kontext *nicht* auf den Stack eingehen.
- Grundslegend sprechen wir von *globaler* und *lokaler* Sichtbarkeit einer Variable:

```
// Global:  
public static int globaleZahl; // ← Überall wo Klasse sichtbar  
private static int klasseZahl; // ← Überall innerhalb der Klasse  
public static int main(String[] args){  
    // Lokal:  
    int lokaleZahl; // ← Nur innerhalb von main  
}
```

# Präzise Antworten: Überlagerung

- In Java gibt es verschiedene Sichtbarkeiten. Wir werden uns hier ansehen, was bei gleichnamigen Variablen geschieht.
- Wir werden in diesem Kontext *nicht* auf den Stack eingehen.
- Grundslegend sprechen wir von *globaler* und *lokaler* Sichtbarkeit einer Variable:

```
// Global:  
public static int globaleZahl; // ← Überall wo Klasse sichtbar  
private static int klasseZahl; // ← Überall innerhalb der Klasse  
public static int main(String[] args){  
    // Lokal:  
    int lokaleZahl; // ← Nur innerhalb von main  
}
```

- Nun mal sehen, wie die Variablen überschrieben werden.

# Verdrängungs'v'reuden

# Verdrängungs'v'reuden

- Es können in Methoden nur globale Variablen überschrieben werden.

# Verdrängungs'v'reuden

- Es können in Methoden nur globale Variablen überschrieben werden.
- Überschrieben werden sie entweder durch die Erstellung einer gleichnamige Variable

# Verdrängungs'v'reuden

- Es können in Methoden nur globale Variablen überschrieben werden.
- Überschrieben werden sie entweder durch die Erstellung einer gleichnamige Variable oder einem Parameter mit gleichem Namen.



# Verdrängungs'v'reuden

- Es können in Methoden nur globale Variablen überschrieben werden.
- Überschrieben werden sie entweder durch die Erstellung einer gleichnamige Variable oder einem Parameter mit gleichem Namen.
- Der Typ spielt *keine* Rolle. So erzeugt:

# Verdrängungs'v'reuden

- Es können in Methoden nur globale Variablen überschrieben werden.
- Überschrieben werden sie entweder durch die Erstellung einer gleichnamige Variable oder einem Parameter mit gleichem Namen.
- Der Typ spielt *keine* Rolle. So erzeugt:

```
3 public static String test = "";
4 public static void main(String[] args){
5     int test = 42;
6     System.out.println(test);
7 }
```

# Verdrängungs'v'reuden

- Es können in Methoden nur globale Variablen überschrieben werden.
- Überschrieben werden sie entweder durch die Erstellung einer gleichnamige Variable oder einem Parameter mit gleichem Namen.
- Der Typ spielt *keine* Rolle. So erzeugt:

```
3 public static String test = "";
4 public static void main(String[] args){
5     int test = 42;
6     System.out.println(test);
7 }
```

die Ausgabe `java` Verdraengung :

42

# Super-Lokal mit Blöcken

# Super-Lokal mit Blöcken

- Wir können in Java (fast) überall geschwungene Klammerpaare setzen.

# Super-Lokal mit Blöcken

- Wir können in Java (fast) überall geschwungene Klammerpaare setzen. Sie fassen Anweisungen wieder zu einem lokalen Block zusammen.

# Super-Lokal mit Blöcken

- Wir können in Java (fast) überall geschwungene Klammerpaare setzen. Sie fassen Anweisungen wieder zu einem lokalen Block zusammen.
- In diesem Block existieren alle Variablen der äußeren Blöcke,

# Super-Lokal mit Blöcken

- Wir können in Java (fast) überall geschwungene Klammerpaare setzen. Sie fassen Anweisungen wieder zu einem lokalen Block zusammen.
- In diesem Block existieren alle Variablen der äußeren Blöcke, wir können also keine neuen erstellen. Allerdings überleben die Variablen nur den Block.



# Super-Lokal mit Blöcken

- Wir können in Java (fast) überall geschwungene Klammerpaare setzen. Sie fassen Anweisungen wieder zu einem lokalen Block zusammen.
- In diesem Block existieren alle Variablen der äußeren Blöcke, wir können also keine neuen erstellen. Allerdings überleben die Variablen nur den Block.
- Zuerstmal ein eher sinnfreies Beispiel...

# Super-Lokal mit Blöcken

# Super-Lokal mit Blöcken

- Der Code

# Super-Lokal mit Blöcken

- Der Code

```
static int test = 0;
```

# Super-Lokal mit Blöcken

- Der Code

```
static int test = 0;  
public static void main(String[] args) {
```

# Super-Lokal mit Blöcken

- Der Code

```
static int test = 0;  
public static void main(String[] args) {  
    System.out.print(test + " ");
```

# Super-Lokal mit Blöcken

## ■ Der Code

```
static int test = 0;
public static void main(String[] args) {
    System.out.print(test + "_");
    {
        int test = 15; // Überlagere lokal
        System.out.print(test + "_");
    }
}
```

# Super-Lokal mit Blöcken

## ■ Der Code

```
static int test = 0;
public static void main(String[] args) {
    System.out.print(test + "_");
    {
        int test = 15; // Überlagere lokal
        System.out.print(test + "_");
    }
    System.out.print(test);
}
```



# Super-Lokal mit Blöcken

## ■ Der Code

```
static int test = 0;
public static void main(String[] args) {
    System.out.print(test + "_");
    {
        int test = 15; // Überlagere lokal
        System.out.print(test + "_");
    }
    System.out.print(test);
}
```

erzeugt die Ausgabe `java` Block:

# Super-Lokal mit Blöcken

## ■ Der Code

```
static int test = 0;
public static void main(String[] args) {
    System.out.print(test + "_");
    {
        int test = 15; // Überlagere lokal
        System.out.print(test + "_");
    }
    System.out.print(test);
}
```

erzeugt die Ausgabe **java** Block:

```
0 15 0
```

# Super-Lokal mit Blöcken

- Der Code

```
static int test = 0;  
public static void main(String[] args) {  
    System.out.print(test + "_");  
    {  
        int test = 15; // Überlagere lokal  
        System.out.print(test + "_");  
    }  
    System.out.print(test);  
}
```

erzeugt die Ausgabe `java` Block:

```
0 15 0
```

- Wann ist dies von Interesse?

# Super-Lokal mit Blöcken

- Der Code

```
static int test = 0;  
public static void main(String[] args) {  
    System.out.print(test + " ");  
    {  
        int test = 15; // Überlagere lokal  
        System.out.print(test + " ");  
    }  
    System.out.print(test);  
}
```

erzeugt die Ausgabe `java` Block:

```
0 15 0
```

- Wann ist dies von Interesse? Bei `if` zum Beispiel ...

# Wie, wir kannten Blöcke schon?

# Wie, wir kannten Blöcke schon?

- Wenn wir nach einem `if/while/...` Kopf eine geschwungene Klammer setzen,

# Wie, wir kannten Blöcke schon?

- Wenn wir nach einem `if/while/...` Kopf eine geschwungene Klammer setzen, starten wir einen neuen Block.

# Wie, wir kannten Blöcke schon?

- Wenn wir nach einem `if/while/...` Kopf eine geschwungene Klammer setzen, starten wir einen neuen Block.
- Hinweis: Bei Methoden, Klassen, Enums, ... gehören die Klammern dazu (und sind deswegen anders).



# Wie, wir kannten Blöcke schon?

- Wenn wir nach einem `if/while/...` Kopf eine geschwungene Klammer setzen, starten wir einen neuen Block.
- Hinweis: Bei Methoden, Klassen, Enums, ... gehören die Klammern dazu (und sind deswegen anders).
- Allgemein noch:

# Wie, wir kannten Blöcke schon?

- Wenn wir nach einem `if/while/...` Kopf eine geschwungene Klammer setzen, starten wir einen neuen Block.
- Hinweis: Bei Methoden, Klassen, Enums, ... gehören die Klammern dazu (und sind deswegen anders).
- Allgemein noch: *achtet auf eine präzise Beschreibung!*

# Wie, wir kannten Blöcke schon?

- Wenn wir nach einem `if/while/...` Kopf eine geschwungene Klammer setzen, starten wir einen neuen Block.
- Hinweis: Bei Methoden, Klassen, Enums, ... gehören die Klammern dazu (und sind deswegen anders).
- Allgemein noch: *achtet auf eine präzise Beschreibung!* „Java weiß dann nicht was es tun soll“ klingt zwar süß,

# Wie, wir kannten Blöcke schon?

- Wenn wir nach einem `if/while/...` Kopf eine geschwungene Klammer setzen, starten wir einen neuen Block.
- Hinweis: Bei Methoden, Klassen, Enums, ... gehören die Klammern dazu (und sind deswegen anders).
- Allgemein noch: *achtet auf eine präzise Beschreibung!* „Java weiß dann nicht was es tun soll“ klingt zwar süß, aber erklärt nicht warum.

# Präsenzaufgabe, nun Vektoriell

# Präsenzaufgabe, nun Vektoriell

- Im Folgenden soll eine Vektor Klasse konstruiert werden.

# Präsenzaufgabe, nun Vektoriell

- Im Folgenden soll eine Vektor Klasse konstruiert werden.
- Zu unterstützen sind  $n$  Dimensionen  $\mathbb{R}^n$  ( $n \in \mathbb{N}$ ,  $n > 0$ ).

# Präsenzaufgabe, nun Vektoriell

- Im Folgenden soll eine Vektor Klasse konstruiert werden.
- Zu unterstützen sind  $n$  Dimensionen  $\mathbb{R}^n$  ( $n \in \mathbb{N}$ ,  $n > 0$ ).
- Ein Objekt der Vektor-Klasse hat die folgenden Informationen zu halten:



# Präsenzaufgabe, nun Vektoriell

- Im Folgenden soll eine Vektor Klasse konstruiert werden.
- Zu unterstützen sind  $n$  Dimensionen  $\mathbb{R}^n$  ( $n \in \mathbb{N}$ ,  $n > 0$ ).
- Ein Objekt der Vektor-Klasse hat die folgenden Informationen zu halten:  
Vektorkomponenten: Eine Folge von  $n$  Werten, die nach der Instanziierung alle *nicht mehr* verändert werden dürfen.

# Präsenzaufgabe, nun Vektoriell

- Im Folgenden soll eine Vektor Klasse konstruiert werden.
- Zu unterstützen sind  $n$  Dimensionen  $\mathbb{R}^n$  ( $n \in \mathbb{N}$ ,  $n > 0$ ).
- Ein Objekt der Vektor-Klasse hat die folgenden Informationen zu halten:  
Vektorkomponenten: Eine Folge von  $n$  Werten, die nach der Instanziierung alle *nicht mehr* verändert werden dürfen.  
Betrag des Vektors: Der Betrag des Vektors (ebenfalls unveränderbar).

# Präsenzaufgabe, nun Vektoriell

- Im Folgenden soll eine Vektor Klasse konstruiert werden.
- Zu unterstützen sind  $n$  Dimensionen  $\mathbb{R}^n$  ( $n \in \mathbb{N}$ ,  $n > 0$ ).
- Ein Objekt der Vektor-Klasse hat die folgenden Informationen zu halten:
  - Vektorkomponenten: Eine Folge von  $n$  Werten, die nach der Instanziierung alle *nicht mehr* verändert werden dürfen.
  - Betrag des Vektors: Der Betrag des Vektors (ebenfalls unveränderbar).
- Um die Präsenzaufgabe übersichtlicher zu gestalten, teile ich sie in mehrere auf.

# Vectorias, the great one

3

We please the mighty

Für einen unveränderbaren Vektor  $\mathbb{R}^n$  ( $n \in \mathbb{N}$ ,  $n > 0$ ):

# Vectorias, the great one

3

We please the mighty

Für einen unveränderbaren Vektor  $\mathbb{R}^n$  ( $n \in \mathbb{N}$ ,  $n > 0$ ):

1. Schreiben Sie eine Klasse `Vector`, welche alle notwendigen Attribute (unveränderbare Vektorkomponenten, unveränderbarer Betrag), enthält.

# Vectorias, the great one

3

We please the mighty

Für einen unveränderbaren Vektor  $\mathbb{R}^n$  ( $n \in \mathbb{N}$ ,  $n > 0$ ):

1. Schreiben Sie eine Klasse `Vector`, welche alle notwendigen Attribute (unveränderbare Vektorkomponenten, unveränderbarer Betrag), enthält.
2. Schreiben Sie eine `public String toString()`-Methode. Die nach Java-Standard alle Attribute ausgibt.

# Vectorias, the great one

3

We please the mighty

Für einen unveränderbaren Vektor  $\mathbb{R}^n$  ( $n \in \mathbb{N}$ ,  $n > 0$ ):

1. Schreiben Sie eine Klasse `Vector`, welche alle notwendigen Attribute (unveränderbare Vektorkomponenten, unveränderbarer Betrag), enthält.
2. Schreiben Sie eine `public String toString()`-Methode. Die nach Java-Standard alle Attribute ausgibt. Hier eine beispielhafte Ausgabe. Das genau Format ist Ihnen überlassen:

```
{ magnitude: 2.0, values: [0.0, 2.0, 0.0, 0.0, 0.0] }
```

# Vectors, the great second

4

We choose the mighty

Für einen unveränderbaren Vektor  $\mathbb{R}^n$  ( $n \in \mathbb{N}$ ,  $n > 0$ ):



# Vectorias, the great second

4

We choose the mighty

Für einen unveränderbaren Vektor  $\mathbb{R}^n$  ( $n \in \mathbb{N}$ ,  $n > 0$ ):

3. Implementieren Sie `public static Vector create(double... values)`, welche die Einschränkungen prüft. Sind Sie erfüllt, soll ein neuer unveränderbarer Vektor und sonst `null` zurückgegeben werden.

# Vectors, the great second

4

We choose the mighty

Für einen unveränderbaren Vektor  $\mathbb{R}^n$  ( $n \in \mathbb{N}$ ,  $n > 0$ ):

- Implementieren Sie `public static Vector create(double... values)`, welche die Einschränkungen prüft. Sind Sie erfüllt, soll ein neuer unveränderbarer Vektor und sonst `null` zurückgegeben werden. Beispiele:

```
Vector a = Vector.create(0, 2, 0, 0, 0);  
Vector b = Vector.create(13, 2.5, 3.2);  
System.out.println(a); // → Ausgabe von toString()
```

# Vectorias, the great third

5

We freeze the mighty

# Vectorias, the great third

5

We freeze the mighty

Für einen unveränderbaren Vektor  $\mathbb{R}^n$  ( $n \in \mathbb{N}, n > 0$ ):

# Vectorias, the great third

5

We freeze the mighty

Für einen unveränderbaren Vektor  $\mathbb{R}^n$  ( $n \in \mathbb{N}, n > 0$ ):

4. Implementieren Sie eine Methode `public double[] getValues()`, welche die übergebenen Werte zurückliefert, aber die Unveränderbarkeit sicher stellt.

# Vectorias, the great third

5

We freeze the mighty

Für einen unveränderbaren Vektor  $\mathbb{R}^n$  ( $n \in \mathbb{N}$ ,  $n > 0$ ):

4. Implementieren Sie eine Methode `public double[] getValues()`, welche die übergebenen Werte zurückliefert, aber die Unveränderbarkeit sicher stellt.

```
Vector a = Vector.create(0, 2, 0);  
double[] v = a.getValues();  
v[0] = 3;  
System.out.println(Arrays.toString(v)); // → [3.0, 2.0, 0.0]  
System.out.println(a); // → original vector
```

# Vectorias, the great fourth

6

We freeze the brightly

Für einen unveränderbaren Vektor  $\mathbb{R}^n$  ( $n \in \mathbb{N}, n > 0$ ):

# Vectorias, the great fourth

6

We freeze the brightly

Für einen unveränderbaren Vektor  $\mathbb{R}^n$  ( $n \in \mathbb{N}, n > 0$ ):

5. Implementieren Sie eine Methode `public Vector normalize()`, welche einen neuen normalisierten Vektor erzeugt.



# Vectorias, the great fourth

6

We freeze the brighty

Für einen unveränderbaren Vektor  $\mathbb{R}^n$  ( $n \in \mathbb{N}$ ,  $n > 0$ ):

5. Implementieren Sie eine Methode `public Vector normalize()`, welche einen neuen normalisierten Vektor erzeugt. Dieser hat die selbe Richtung, wie das Original, aber einen Betrag von 1. Die Normalisierung kann durch die folgende Formel erfolgen:

$$\vec{n} = \frac{\vec{v}}{||\vec{v}||}$$

# Vectori-bras, the great solver

# Vectori-bras, the great solver

- Hier die Klasse an sich: `Vector.java`

# Vectori-bras, the great solver

- Hier die Klasse an sich: [Vector.java](#)
- Zunächst die unveränderlichen Attribute:

# Vectori-bras, the great solver

- Hier die Klasse an sich: [Vector.java](#)
- Zunächst die unveränderlichen Attribute:

```
public class Vector {  
  
  
  
  
  
  
  
  
  
}
```

# Vectori-bras, the great solver

- Hier die Klasse an sich: [Vector.java](#)
- Zunächst die unveränderlichen Attribute:

```
public class Vector {  
    private final double[] values; // darf nicht public sein  
    public final double magnitude; // oder mit getter  
}
```

# Vectori-bras, the great solver

- Hier die Klasse an sich: [Vector.java](#)
- Zunächst die unveränderlichen Attribute:

```
public class Vector {  
    private final double[] values; // darf nicht public sein  
    public final double magnitude; // oder mit getter  
}
```

- Die toString()-Methode:





# Vectori-bras, the great solver

- Hier die Klasse an sich: [Vector.java](#)
- Zunächst die unveränderlichen Attribute:

```
public class Vector {  
    private final double[] values; // darf nicht public sein  
    public final double magnitude; // oder mit getter  
}
```

- Die toString()-Methode:

```
public String toString() {  
    return "{_magnitude:_ " + magnitude  
        + ",_values:_ " + Arrays.toString(values)  
        + "_}";  
}
```

# Vectori-bras, the late solver

# Vectori-bras, the late solver

- Die `create(double...)`-Methode:



# Vectori-bras, the late solver

- Die create(double...)-Methode:

```
public static Vector create(double... values) {  
    if(values.length <= 0)  
        return null;  
  
}
```

# Vectori-bras, the late solver

- Die `create(double...)`-Methode:

```
public static Vector create(double... values) {  
    if(values.length <= 0)  
        return null;  
    return new Vector(values);  
}
```

# Vectori-bras, the late solver

- Die create(double...)-Methode:

```
// expects: valid vector (auch möglich: Exceptions)
```

```
private Vector(double[] values) {
```

```
}
```

```
public static Vector create(double... values) {
```

```
    if(values.length <= 0)
```

```
        return null;
```

```
    return new Vector(values);
```

```
}
```

# Vectori-bras, the late solver

- Die create(double...)-Methode:

```
// expects: valid vector (auch möglich: Exceptions)
private Vector(double[] values) {
    this.values = values; // to be completely sure: another copy
    this.magnitude = calculateMagnitude(values);
}

public static Vector create(double... values) {
    if(values.length <= 0)
        return null;
    return new Vector(values);
}
```



# Vectori-bras, the bait resolver

# Vectori-bras, the bait resolver

- Hier noch die Hilfsmethode für den Betrag:

## Vectori-bras, the bait resolver

- Hier noch die Hilfsmethode für den Betrag:

```
private static double calculateMagnitude(double[] values) {
```

}

# Vectori-bras, the bait resolver

- Hier noch die Hilfsmethode für den Betrag:

```
private static double calculateMagnitude(double[] values) {  
    double sumOfSquares = 0;  
  
}
```

# Vectori-bras, the bait resolver

- Hier noch die Hilfsmethode für den Betrag:

```
private static double calculateMagnitude(double[] values) {  
    double sumOfSquares = 0;  
    for (double value : values) {  
  
    }  
  
}
```

# Vectori-bras, the bait resolver

- Hier noch die Hilfsmethode für den Betrag:

```
private static double calculateMagnitude(double[] values) {  
    double sumOfSquares = 0;  
    for (double value : values) {  
        sumOfSquares += Math.pow(value, 2);  
    }  
  
}
```

# Vectori-bras, the bait resolver

- Hier noch die Hilfsmethode für den Betrag:

```
private static double calculateMagnitude(double[] values) {  
    double sumOfSquares = 0;  
    for (double value : values) {  
        sumOfSquares += Math.pow(value, 2);  
    }  
    return Math.sqrt(sumOfSquares);  
}
```

# Vectori-was? The quest revolver



# Vectori-was? The quest revolver

- Zum Erhalt der Werte erzeugen wir eine Kopie:

# Vectori-was? The quest revolver

- Zum Erhalt der Werte erzeugen wir eine Kopie:

```
public double[] getValues() {  
  
}
```

# Vectori-was? The quest revolver

- Zum Erhalt der Werte erzeugen wir eine Kopie:

```
public double[] getValues() {  
    return copyArray(this.values);  
}
```

# Vectori-was? The quest revolver

- Zum Erhalt der Werte erzeugen wir eine Kopie:

```
public double[] getValues() {  
    return copyArray(this.values);  
}
```

```
private double[] copyArray(double[] arr) {
```

```
}
```

# Vectori-was? The quest revolver

- Zum Erhalt der Werte erzeugen wir eine Kopie:

```
public double[] getValues() {  
    return copyArray(this.values);  
}
```

```
private double[] copyArray(double[] arr) {  
    double[] copy = new double[arr.length];  
  
}
```

# Vectori-was? The quest revolver

- Zum Erhalt der Werte erzeugen wir eine Kopie:

```
public double[] getValues() {  
    return copyArray(this.values);  
}
```

```
private double[] copyArray(double[] arr) {  
    double[] copy = new double[arr.length];  
    for (int i = 0; i < copy.length; i++) {  
  
    }  
  
}
```

# Vectori-was? The quest revolver

- Zum Erhalt der Werte erzeugen wir eine Kopie:

```
public double[] getValues() {  
    return copyArray(this.values);  
}  
  
private double[] copyArray(double[] arr) {  
    double[] copy = new double[arr.length];  
    for (int i = 0; i < copy.length; i++) {  
        copy[i] = arr[i];  
    }  
}
```

# Vectori-was? The quest revolver

- Zum Erhalt der Werte erzeugen wir eine Kopie:

```
public double[] getValues() {  
    return copyArray(this.values);  
}
```

```
private double[] copyArray(double[] arr) {  
    double[] copy = new double[arr.length];  
    for (int i = 0; i < copy.length; i++) {  
        copy[i] = arr[i];  
    }  
    return copy;  
}
```



# Vectori-das, the final wolf

# Vectori-das, the final wolf

- Nun noch die Normalisierung:

# Vectori-das, the final wolf

- Nun noch die Normalisierung:

```
public Vector normalize() {
```

```
}
```

# Vectori-das, the final wolf

- Nun noch die Normalisierung:

```
public Vector normalize() {  
    double[] normalizedValues = new double[this.values.length];  
  
}
```

# Vectori-das, the final wolf

- Nun noch die Normalisierung:

```
public Vector normalize() {  
    double[] normalizedValues = new double[this.values.length];  
    for (int i = 0; i < normalizedValues.length; i++) {  
  
    }  
  
}
```

# Vectori-das, the final wolf

- Nun noch die Normalisierung:

```
public Vector normalize() {  
    double[] normalizedValues = new double[this.values.length];  
    for (int i = 0; i < normalizedValues.length; i++) {  
        normalizedValues[i] = this.values[i] / this.magnitude;  
    }  
  
}
```

# Vectori-das, the final wolf

- Nun noch die Normalisierung:

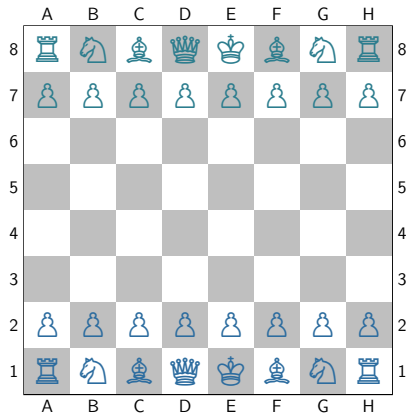
```
public Vector normalize() {  
    double[] normalizedValues = new double[this.values.length];  
    for (int i = 0; i < normalizedValues.length; i++) {  
        normalizedValues[i] = this.values[i] / this.magnitude;  
    }  
    return Vector.create(normalizedValues);  
}
```

# Alte Zusatzaufgabe



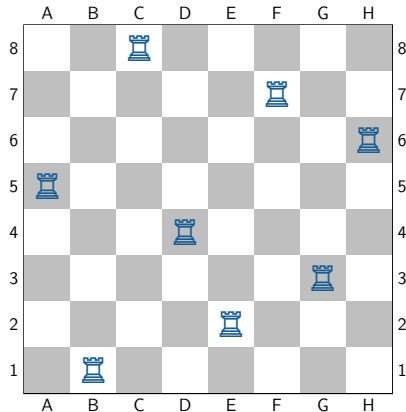
# Alte Zusatzaufgabe

- Schach!



## Alte Zusatzaufgabe

- Schach!



# Alte Zusatzaufgabe

# Alte Zusatzaufgabe

- Für Türme lässt sich die Aufgabe leicht umformulieren ([Zusatz.java](#)):  
„In jeder Zeile und jeder Spalte darf es maximal einen Turm geben“

# Alte Zusatzaufgabe

- Für Türme lässt sich die Aufgabe leicht umformulieren ([Zusatz.java](#)):  
„In jeder Zeile und jeder Spalte darf es maximal einen Turm geben“
- Das Grundgerüst:

# Alte Zusatzaufgabe

- Für Türme lässt sich die Aufgabe leicht umformulieren ([Zusatz.java](#)):  
„In jeder Zeile und jeder Spalte darf es maximal einen Turm geben“
- Das Grundgerüst:

```
boolean[][] field = { /* ... */ }
```

# Alte Zusatzaufgabe

- Für Türme lässt sich die Aufgabe leicht umformulieren ([Zusatz.java](#)): „In jeder Zeile und jeder Spalte darf es maximal einen Turm geben“
- Das Grundgerüst:

```
boolean[][] field = { /* ... */ }  
boolean validField = true;
```

# Alte Zusatzaufgabe

- Für Türme lässt sich die Aufgabe leicht umformulieren ([Zusatz.java](#)): „In jeder Zeile und jeder Spalte darf es maximal einen Turm geben“
- Das Grundgerüst:

```
boolean[][] field = { /* ... */ }  
boolean validField = true;  
for (int i = 0; i < field.length; i++) {
```



# Alte Zusatzaufgabe

- Für Türme lässt sich die Aufgabe leicht umformulieren ([Zusatz.java](#)): „In jeder Zeile und jeder Spalte darf es maximal einen Turm geben“
- Das Grundgerüst:

```
boolean[][] field = { /* ... */ }  
boolean validField = true;  
for (int i = 0; i < field.length; i++) {  
    // ...  
}
```

# Alte Zusatzaufgabe

- Für Türme lässt sich die Aufgabe leicht umformulieren ([Zusatz.java](#)): „In jeder Zeile und jeder Spalte darf es maximal einen Turm geben“
- Das Grundgerüst:

```
boolean[][] field = { /* ... */ }  
boolean validField = true;  
for (int i = 0; i < field.length; i++) {  
    // ...  
}
```

- Kein „line“, da wir damit Zeilen und Spalten abgehen werden...

# Alte Zusatzaufgabe

- Für Türme lässt sich die Aufgabe leicht umformulieren ([Zusatz.java](#)): „In jeder Zeile und jeder Spalte darf es maximal einen Turm geben“
- Das Grundgerüst:

```
boolean[][] field = { /* ... */ }  
boolean validField = true;  
for (int i = 0; i < field.length; i++) {  
    // ...  
}
```

- Kein „line“, da wir damit Zeilen und Spalten abgehen werden...
- Hinweis: Wenn wir auch prüfen möchten, ob es wirklich  $n$  Türme sind, dann können wir die Aussage ersetzen durch „... genau einen Turm“,

# Alte Zusatzaufgabe

- Für Türme lässt sich die Aufgabe leicht umformulieren ([Zusatz.java](#)): „In jeder Zeile und jeder Spalte darf es maximal einen Turm geben“
- Das Grundgerüst:

```
boolean[][] field = { /* ... */ }  
boolean validField = true;  
for (int i = 0; i < field.length; i++) {  
    // ...  
}
```

- Kein „line“, da wir damit Zeilen und Spalten abgehen werden...
- Hinweis: Wenn wir auch prüfen möchten, ob es wirklich  $n$  Türme sind, dann können wir die Aussage ersetzen durch „... genau einen Turm“, und die Türme beispielsweise zählen.

# Alte Zusatzaufgabe

# Alte Zusatzaufgabe

// für jedes i

# Alte Zusatzaufgabe

```
// für jedes i  
int rooksHorizontal = 0;
```

# Alte Zusatzaufgabe

```
// für jedes i  
int rooksHorizontal = 0;  
int rooksVertical = 0;
```



# Alte Zusatzaufgabe

```
// für jedes i
int rooksHorizontal = 0;
int rooksVertical = 0;
for (int j = 0; j < field.length; j++) { // da w = h
```

# Alte Zusatzaufgabe

```
// für jedes i
int rooksHorizontal = 0;
int rooksVertical = 0;
for (int j = 0; j < field.length; j++) { // da w = h
    if(field[i][j])
```

# Alte Zusatzaufgabe

```
// für jedes i
int rooksHorizontal = 0;
int rooksVertical = 0;
for (int j = 0; j < field.length; j++) { // da w = h
    if(field[i][j])
        rooksHorizontal += 1;
```

# Alte Zusatzaufgabe

```
// für jedes i
int rooksHorizontal = 0;
int rooksVertical = 0;
for (int j = 0; j < field.length; j++) { // da w = h
    if(field[i][j])
        rooksHorizontal += 1;
    if(field[j][i])
```

# Alte Zusatzaufgabe

```
// für jedes i
int rooksHorizontal = 0;
int rooksVertical = 0;
for (int j = 0; j < field.length; j++) { // da w = h
    if(field[i][j])
        rooksHorizontal += 1;
    if(field[j][i])
        rooksVertical += 1;
}
```

# Alte Zusatzaufgabe

```
// für jedes i
int rooksHorizontal = 0;
int rooksVertical = 0;
for (int j = 0; j < field.length; j++) { // da w = h
    if(field[i][j])
        rooksHorizontal += 1;
    if(field[j][i])
        rooksVertical += 1;
}
```

# Alte Zusatzaufgabe

```
// für jedes i
int rooksHorizontal = 0;
int rooksVertical = 0;
for (int j = 0; j < field.length; j++) { // da w = h
    if(field[i][j])
        rooksHorizontal += 1;
    if(field[j][i])
        rooksVertical += 1;
}
if(rooksHorizontal > 1 || rooksVertical > 1) {
```

# Alte Zusatzaufgabe

```
// für jedes i
int rooksHorizontal = 0;
int rooksVertical = 0;
for (int j = 0; j < field.length; j++) { // da w = h
    if(field[i][j])
        rooksHorizontal += 1;
    if(field[j][i])
        rooksVertical += 1;
}
if(rooksHorizontal > 1 || rooksVertical > 1) {
    validField = false;
}
```



# Alte Zusatzaufgabe

```
// für jedes i
int rooksHorizontal = 0;
int rooksVertical = 0;
for (int j = 0; j < field.length; j++) { // da w = h
    if(field[i][j])
        rooksHorizontal += 1;
    if(field[j][i])
        rooksVertical += 1;
}
if(rooksHorizontal > 1 || rooksVertical > 1) {
    validField = false;
    break;
}
```

# Alte Zusatzaufgabe

```
// für jedes i
int rooksHorizontal = 0;
int rooksVertical = 0;
for (int j = 0; j < field.length; j++) { // da w = h
    if(field[i][j])
        rooksHorizontal += 1;
    if(field[j][i])
        rooksVertical += 1;
}
if(rooksHorizontal > 1 || rooksVertical > 1) {
    validField = false;
    break;
}
```

# Alte Zusatzaufgabe

# Alte Zusatzaufgabe

- Zuletzt noch die Auswertung:

# Alte Zusatzaufgabe

- Zuletzt noch die Auswertung:

`if(validField)`

# Alte Zusatzaufgabe

- Zuletzt noch die Auswertung:

```
if(validField)  
    System.out.println("Feld_korrekt");
```

# Alte Zusatzaufgabe

- Zuletzt noch die Auswertung:

```
if(validField)
    System.out.println("Feld_korrekt");
else
```

# Alte Zusatzaufgabe

- Zuletzt noch die Auswertung:

```
if(validField)
    System.out.println("Feld_korrekt");
else
    System.out.println("Feld_inkorrekt");
```



# Alte Zusatzaufgabe, Alternative

# Alte Zusatzaufgabe, Alternative

- Betrachten wir eine Prüfroutine für ein gegebenes Feld ([ZusatzB.java](#))



# Alte Zusatzaufgabe, Alternative

- Betrachten wir eine Prüfroutine für ein gegebenes Feld ([ZusatzB.java](#))

```
boolean isRookPositionValid(int x, int y, boolean[][] field) {  
    for (int i = 0; i < field.length; i++) {  
  
    }  
  
}
```

# Alte Zusatzaufgabe, Alternative

- Betrachten wir eine Prüfroutine für ein gegebenes Feld ([ZusatzB.java](#))

```
boolean isRookPositionValid(int x, int y, boolean[][] field) {  
    for (int i = 0; i < field.length; i++) {  
        if(i != x && field[y][i]) return false;  
    }  
}
```

# Alte Zusatzaufgabe, Alternative

- Betrachten wir eine Prüfroutine für ein gegebenes Feld ([ZusatzB.java](#))

```
boolean isRookPositionValid(int x, int y, boolean[][] field) {  
    for (int i = 0; i < field.length; i++) {  
        if(i != x && field[y][i]) return false;  
        if(i != y && field[i][x]) return false;  
    }  
  
}
```

# Alte Zusatzaufgabe, Alternative

- Betrachten wir eine Prüfroutine für ein gegebenes Feld ([ZusatzB.java](#))

```
boolean isRookPositionValid(int x, int y, boolean[][] field) {  
    for (int i = 0; i < field.length; i++) {  
        if(i != x && field[y][i]) return false;  
        if(i != y && field[i][x]) return false;  
    }  
    return true;  
}
```

# Alte Zusatzaufgabe, Alternative



# Alte Zusatzaufgabe, Alternative

- Nun können wir über das Feld iterieren und alle Turmpositionen prüfen:

# Alte Zusatzaufgabe, Alternative

- Nun können wir über das Feld iterieren und alle Turmpositionen prüfen:

```
public static boolean checkField(boolean[][] field) {
```

```
}
```

# Alte Zusatzaufgabe, Alternative

- Nun können wir über das Feld iterieren und alle Turmpositionen prüfen:

```
public static boolean checkField(boolean[][] field) {  
    for(int y = 0; y < field.length; y++) {  
  
  
  
  
  
  
    }  
  
}
```

# Alte Zusatzaufgabe, Alternative

- Nun können wir über das Feld iterieren und alle Turmpositionen prüfen:

```
public static boolean checkField(boolean[][] field) {  
    for(int y = 0; y < field.length; y++) {  
        for(int x = 0; x < field.length; x++) {  
  
        }  
    }  
  
}
```

# Alte Zusatzaufgabe, Alternative

- Nun können wir über das Feld iterieren und alle Turmpositionen prüfen:

```
public static boolean checkField(boolean[][] field) {  
    for(int y = 0; y < field.length; y++) {  
        for(int x = 0; x < field.length; x++) {  
            if(field[y][x] && !isRookPositionValid(x, y, field))  
                return false;  
        }  
    }  
}
```

# Alte Zusatzaufgabe, Alternative

- Nun können wir über das Feld iterieren und alle Turmpositionen prüfen:

```
public static boolean checkField(boolean[][] field) {  
    for(int y = 0; y < field.length; y++) {  
        for(int x = 0; x < field.length; x++) {  
            if(field[y][x] && !isRookPositionValid(x, y, field))  
                return false;  
        }  
    }  
    return true;  
}
```

# Alte Zusatzaufgabe, Alternative

- Nun können wir über das Feld iterieren und alle Turmpositionen prüfen:

```
public static boolean checkField(boolean[][] field) {  
    for(int y = 0; y < field.length; y++) {  
        for(int x = 0; x < field.length; x++) {  
            if(field[y][x] && !isRookPositionValid(x, y, field))  
                return false;  
        }  
    }  
    return true;  
}
```

- Auch hier lässt sich die Prüfung beispielsweise durch einen Zähler erweitern.

Ich bin eine Biene.



# Ich bin eine Biene.

- Klassen an sich schon bekannt.

# Ich bin eine Biene.

- Klassen an sich schon bekannt. Keywords: `class` so zum Beispiel bei `Hamster.java`:

# Ich bin eine Biene.

- Klassen an sich schon bekannt. Keywords: **class** so zum Beispiel bei Hamster . java:

```
1 class Hamster {  
2     // ...  
3 }
```

# Ich bin eine Biene.

- Klassen an sich schon bekannt. Keywords: **class** so zum Beispiel bei `Hamster.java`:

```
1 class Hamster {  
2     // ...  
3 }
```

- Mittels **new** lässt sich ein neues Objekt der Klasse erstellen.

# Ich bin eine Biene.

- Klassen an sich schon bekannt. Keywords: **class** so zum Beispiel bei `Hamster.java`:

```
1 class Hamster {  
2     // ...  
3 }
```

- Mittels **new** lässt sich ein neues Objekt der Klasse erstellen.
- Beispiel: `Hamster günther = new Hamster();`.

# Ich bin eine Biene.

- Klassen an sich schon bekannt. Keywords: **class** so zum Beispiel bei `Hamster.java`:

```
1 class Hamster {  
2     // ...  
3 }
```

- Mittels **new** lässt sich ein neues Objekt der Klasse erstellen.
- Beispiel: `Hamster günther = new Hamster();`. Erzeugt ein *Objekt* der *Klasse* `Hamster` mit dem Namen `günther`.

# Ich bin eine Biene.

- Klassen an sich schon bekannt. Keywords: `class` so zum Beispiel bei `Hamster.java`:

```
1 class Hamster {  
2     // ...  
3 }
```

- Mittels `new` lässt sich ein neues Objekt der Klasse erstellen.
- Beispiel: `Hamster gүнther = new Hamster();`. Erzeugt ein *Objekt* der *Klasse* `Hamster` mit dem Namen `gүнther`. (Die Deklaration der Variable verläuft analog zu `int Hamster = ...`)

# Objektorientierung - Attribute



# Objektorientierung - Attribute

- Dieser Klasse können wir Variablen zuweisen.

# Objektorientierung - Attribute

- Dieser Klasse können wir Variablen zuweisen. Diese besitzt dann auch jedes Objekt für sich.

# Objektorientierung - Attribute

- Dieser Klasse können wir Variablen zuweisen. Diese besitzt dann auch jedes Objekt für sich.
- Um auf die Variablen von außen zugreifen zu können, deklarieren wir sie mit `public`:

# Objektorientierung - Attribute

- Dieser Klasse können wir Variablen zuweisen. Diese besitzt dann auch jedes Objekt für sich.
- Um auf die Variablen von außen zugreifen zu können, deklarieren wir sie mit **public**:

```
1 class Hamster {  
2     public String name;  
3     public int alter;  
4 }
```

# Objektorientierung - Attribute

- Dieser Klasse können wir Variablen zuweisen. Diese besitzt dann auch jedes Objekt für sich.
- Um auf die Variablen von außen zugreifen zu können, deklarieren wir sie mit **public**:

```
1 class Hamster {  
2     public String name;  
3     public int alter;  
4 }
```

- Mittels **static** deklarieren wir eine Variable, die für alle Hamster identisch ist.

# Objektorientierung - Attribute

- Dieser Klasse können wir Variablen zuweisen. Diese besitzt dann auch jedes Objekt für sich.
- Um auf die Variablen von außen zugreifen zu können, deklarieren wir sie mit **public**:

```
1 class Hamster {  
2     public String name;  
3     public int alter;  
4 }
```

- Mittels **static** deklarieren wir eine Variable, die für alle Hamster identisch ist.  
(Deswegen auch **public static void main(...)**.)

# Objektorientierung - Attribute

- Dieser Klasse können wir Variablen zuweisen. Diese besitzt dann auch jedes Objekt für sich.
- Um auf die Variablen von außen zugreifen zu können, deklarieren wir sie mit **public**:

```
1 class Hamster {  
2     public String name;  
3     public int alter;  
4 }
```

- Mittels **static** deklarieren wir eine Variable, die für alle Hamster identisch ist.

(Deswegen auch **public static void** main(...).)

```
public static boolean flauschig = true;
```

# Objektorientierung - Methoden



# Objektorientierung - Methoden

- Analog zu *main* können wir Methoden deklarieren.

# Objektorientierung - Methoden

- Analog zu *main* können wir Methoden deklarieren.
- Nebst den Zugriffsspezifikatoren (**public**, ...)

# Objektorientierung - Methoden

- Analog zu *main* können wir Methoden deklarieren.
- Nebst den Zugriffsspezifikatoren (**public**, ...) und dem Rückgabetyt (**void**  $\hat{=}$  nichts)

# Objektorientierung - Methoden

- Analog zu *main* können wir Methoden deklarieren.
- Nebst den Zugriffsspezifikatoren (**public**, ...) und dem Rückgabetyt (**void**  $\hat{=}$  nichts) schreiben wir noch den Namen (*main*),

# Objektorientierung - Methoden

- Analog zu *main* können wir Methoden deklarieren.
- Nebst den Zugriffsspezifikatoren (**public**, ...) und dem Rückgabetyt (**void**  $\hat{=}$  nichts) schreiben wir noch den Namen (*main*), so wie in Klammern, welche Argumente die Methode erhält (**String**[] args).

# Objektorientierung - Methoden

- Analog zu *main* können wir Methoden deklarieren.
- Nebst den Zugriffsspezifikatoren (`public`, ...) und dem Rückgabetyt (`void`  $\hat{=}$  nichts) schreiben wir noch den Namen (`main`), so wie in Klammern, welche Argumente die Methode erhält (`String[] args`).
- Beispiel:

# Objektorientierung - Methoden

- Analog zu *main* können wir Methoden deklarieren.
- Nebst den Zugriffsspezifikatoren (**public**, ...) und dem Rückgabetyt (**void**  $\hat{=}$  nichts) schreiben wir noch den Namen (*main*), so wie in Klammern, welche Argumente die Methode erhält (**String[]** *args*).
- Beispiel:

```
public int add(int a, int b){ /* ... */ }
```

Diese Methode erhält zwei Integer und liefert einen zurück. Beispiel:

# Objektorientierung - Methoden

- Analog zu *main* können wir Methoden deklarieren.
- Nebst den Zugriffsspezifikatoren (**public**, ...) und dem Rückgabetyp (**void**  $\hat{=}$  nichts) schreiben wir noch den Namen (*main*), so wie in Klammern, welche Argumente die Methode erhält (**String[]** *args*).
- Beispiel:

```
public int add(int a, int b){ /* ... */ }
```

Diese Methode erhält zwei Integer und liefert einen zurück. Beispiel:

```
public int add(int a, int b){  
    return a + b;  
}
```



# Objektorientierung - Methoden

- Analog zu *main* können wir Methoden deklarieren.
- Nebst den Zugriffsspezifikatoren (**public**, ...) und dem Rückgabetyt (**void**  $\hat{=}$  nichts) schreiben wir noch den Namen (*main*), so wie in Klammern, welche Argumente die Methode erhält (**String**[] args).
- Beispiel:

```
public int add(int a, int b){ /* ... */ }
```

Diese Methode erhält zwei Integer und liefert einen zurück. Beispiel:

```
public int add(int a, int b){  
    return a + b;  
}
```

Der Aufruf `add(2,3)` liefert uns also 5 zurück.

# Ein höflicher Hamster

# Ein höflicher Hamster

- Lassen wir den Hamster sich vorstellen:

# Ein höflicher Hamster

- Lassen wir den Hamster sich vorstellen:

```
1 class Hamster {  
2     public String name;  
3     public int alter;  
4     public static boolean flauschig = true;  
5  
6     public void vorstellen(){  
7         System.out.println("Ich_bin_" + name + "_und_" + alter + "_←  
            Jahre_alt");  
8     }  
9 }
```

# Objektorientierung - Konstruktoren

# Objektorientierung - Konstruktoren

- Beim Erzeugen einer Klasse durch `new` wird der Konstruktor aufgerufen.

# Objektorientierung - Konstruktoren

- Beim Erzeugen einer Klasse durch `new` wird der Konstruktor aufgerufen. (An sich handelt es sich um eine Methode ohne Rückgabetyt).

# Objektorientierung - Konstruktoren

- Beim Erzeugen einer Klasse durch `new` wird der Konstruktor aufgerufen. (An sich handelt es sich um eine Methode ohne Rückgabetyt).
- So kann der Hamster bei der Erstellung einen Namen und ein Alter erhalten:



# Objektorientierung - Konstruktoren

- Beim Erzeugen einer Klasse durch **new** wird der Konstruktor aufgerufen. (An sich handelt es sich um eine Methode ohne Rückgabetyt).
- So kann der Hamster bei der Erstellung einen Namen und ein Alter erhalten:

```
1 // ...  
2 public Hamster(String _name, int _alter) {  
3     name = _name;  
4     alter = _alter;  
5 }
```

# Objektorientierung - Konstruktoren

- Beim Erzeugen einer Klasse durch **new** wird der Konstruktor aufgerufen. (An sich handelt es sich um eine Methode ohne Rückgabotyp).
- So kann der Hamster bei der Erstellung einen Namen und ein Alter erhalten:

```
1 // ...  
2 public Hamster(String _name, int _alter) {  
3     name = _name;  
4     alter = _alter;  
5 }
```

- Nennen wir die Variablen gleich, so werden die des Hamsters über **this** erreicht,

# Objektorientierung - Konstruktoren

- Beim Erzeugen einer Klasse durch **new** wird der Konstruktor aufgerufen. (An sich handelt es sich um eine Methode ohne Rückgabetyt).
- So kann der Hamster bei der Erstellung einen Namen und ein Alter erhalten:

```
1 // ...  
2 public Hamster(String _name, int _alter) {  
3     name = _name;  
4     alter = _alter;  
5 }
```

- Nennen wir die Variablen gleich, so werden die des Hamsters über **this** erreicht, also: **this.name = name;**

# Die Hamster nutzen 😊

# Die Hamster nutzen 😊

- Erzeugen wir einmal zwei Hamster in der main-Methode:

# Die Hamster nutzen 😊

- Erzeugen wir einmal zwei Hamster in der main-Methode:

```
1 Hamster dieter = new Hamster("Dieter", 15);  
2 Hamster jenny = new Hamster("Jennifer", 7);
```

# Die Hamster nutzen 😊

- Erzeugen wir einmal zwei Hamster in der main-Methode:

```
1 Hamster dieter = new Hamster("Dieter", 15);  
2 Hamster jenny = new Hamster("Jennifer", 7);
```

Nun können wir Sie sich vorstellen lassen:

# Die Hamster nutzen 😊

- Erzeugen wir einmal zwei Hamster in der main-Methode:

```
1 Hamster dieter = new Hamster("Dieter", 15);  
2 Hamster jenny = new Hamster("Jennifer", 7);
```

Nun können wir Sie sich vorstellen lassen:

```
1 dieter.vorstellen();  
2 jenny.vorstellen();
```



# Die Hamster nutzen 😊

- Erzeugen wir einmal zwei Hamster in der main-Methode:

```
1 Hamster dieter = new Hamster("Dieter", 15);  
2 Hamster jenny = new Hamster("Jennifer", 7);
```

Nun können wir Sie sich vorstellen lassen:

```
1 dieter.vorstellen();  
2 jenny.vorstellen();
```

Dies liefert jeweils:

# Die Hamster nutzen 😊

- Erzeugen wir einmal zwei Hamster in der main-Methode:

```
1 Hamster dieter = new Hamster("Dieter", 15);  
2 Hamster jenny = new Hamster("Jennifer", 7);
```

Nun können wir Sie sich vorstellen lassen:

```
1 dieter.vorstellen();  
2 jenny.vorstellen();
```

Dies liefert jeweils:

Ich bin Dieter und 15 Jahre alt

Ich bin Jennifer und 7 Jahre alt

# Allgemeines zu Klassen

# Allgemeines zu Klassen

- Jede Klasse gehört in eine eigene .java-Datei mit entsprechendem Namen.

# Allgemeines zu Klassen

- Jede Klasse gehört in eine eigene .java-Datei mit entsprechendem Namen.
- Beim Kompilieren einfach alle .java-Dateien angeben.

# Allgemeines zu Klassen

- Jede Klasse gehört in eine eigene .java-Datei mit entsprechendem Namen.
- Beim Kompilieren einfach alle .java-Dateien angeben.
- Beim Ausführen nur die Auswählen, die die *main*-Methode enthält/starten soll.

# Allgemeines zu Klassen

- Jede Klasse gehört in eine eigene . java-Datei mit entsprechendem Namen.
- Beim Kompilieren einfach alle . java-Dateien angeben.
- Beim Ausführen nur die Auswählen, die die *main*-Methode enthält/starten soll.
- Andere Klassen binden wir mittels `import` ein.

# Allgemeines zu Klassen

- Jede Klasse gehört in eine eigene .java-Datei mit entsprechendem Namen.
- Beim Kompilieren einfach alle .java-Dateien angeben.
- Beim Ausführen nur die Auswählen, die die *main*-Methode enthält/starten soll.
- Andere Klassen binden wir mittels **import** ein. Beispiel:



# Allgemeines zu Klassen

- Jede Klasse gehört in eine eigene . java-Datei mit entsprechendem Namen.
- Beim Kompilieren einfach alle . java-Dateien angeben.
- Beim Ausführen nur die Auswählen, die die *main*-Methode enthält/starten soll.
- Andere Klassen binden wir mittels **import** ein. Beispiel:

```
import java.util.Scanner;
```

# Allgemeines zu Klassen

- Jede Klasse gehört in eine eigene . java-Datei mit entsprechendem Namen.
- Beim Kompilieren einfach alle . java-Dateien angeben.
- Beim Ausführen nur die Auswählen, die die *main*-Methode enthält/starten soll.
- Andere Klassen binden wir mittels **import** ein. Beispiel:

```
import java.util.Scanner;
```

- Ab jetzt steht `Scanner` zur Verfügung.

# Allgemeines zu Klassen

- Jede Klasse gehört in eine eigene .java-Datei mit entsprechendem Namen.
- Beim Kompilieren einfach alle .java-Dateien angeben.
- Beim Ausführen nur die Auswählen, die die *main*-Methode enthält/starten soll.
- Andere Klassen binden wir mittels **import** ein. Beispiel:

```
import java.util.Scanner;
```

- Ab jetzt steht `Scanner` zur Verfügung.
- So können wir auch unsere eigene Klassen einbinden.

# Bonus: Rückgabewerte

# Bonus: Rückgabewerte

- Programme liefern einen Statuswert zurück.

# Bonus: Rückgabewerte

- Programme liefern einen Statuswert zurück.
- Interpretation ist proprietär. Einzig wichtig:

# Bonus: Rückgabewerte

- Programme liefern einen Statuswert zurück.
- Interpretation ist proprietär. Einzig wichtig:  
0: Kein Fehler.

# Bonus: Rückgabewerte

- Programme liefern einen Statuswert zurück.
- Interpretation ist proprietär. Einzig wichtig:
  - 0: Kein Fehler.
  - sonst: Zahl kodiert den Fehler.



# Bonus: Rückgabewerte

- Programme liefern einen Statuswert zurück.
- Interpretation ist proprietär. Einzig wichtig:
  - 0: Kein Fehler.
  - sonst: Zahl kodiert den Fehler.
- Deswegen: Im Fehlerfall `System.exit(1)`,

## Bonus: Rückgabewerte

- Programme liefern einen Statuswert zurück.
- Interpretation ist proprietär. Einzig wichtig:
  - 0: Kein Fehler.
  - sonst: Zahl kodiert den Fehler.
- Deswegen: Im Fehlerfall `System.exit(1)`, oder `System.exit(42)`,

## Bonus: Rückgabewerte

- Programme liefern einen Statuswert zurück.
- Interpretation ist proprietär. Einzig wichtig:
  - 0: Kein Fehler.
  - sonst: Zahl kodiert den Fehler.
- Deswegen: Im Fehlerfall `System.exit(1)`, oder `System.exit(42)`, oder ...

# Bonus: Kommunikation

# Bonus: Kommunikation

- „System.“ liefert drei Streams

# Bonus: Kommunikation

- „System.“ liefert drei Streams über die kommuniziert werden kann:

# Bonus: Kommunikation

- „System.“ liefert drei Streams über die kommuniziert werden kann:
  - `System.out`: für die normale Ausgabe.

# Bonus: Kommunikation

- „System.“ liefert drei Streams über die kommuniziert werden kann:
  - `System.out`: für die normale Ausgabe.
  - `System.err`: für Fehler.



# Bonus: Kommunikation

- „System.“ liefert drei Streams über die kommuniziert werden kann:
  - `System.out`: für die normale Ausgabe.
  - `System.err`: für Fehler.
  - `System.in`: Lesen von Nutzereingaben.

# Bonus: Kommunikation

- „System.“ liefert drei Streams über die kommuniziert werden kann:
  - `System.out`: für die normale Ausgabe.
  - `System.err`: für Fehler.
  - `System.in`: Lesen von Nutzereingaben.
- `System.out` und `System.err` funktionieren identisch.

# Hilfe! Ich bin Überladen

# Hilfe! Ich bin Überladen

- Methoden,

# Hilfe! Ich bin Überladen

- Methoden, kennen wir:

# Hilfe! Ich bin Überladen

- Methoden, kennen wir:

```
public int add (int a, int b) { /* ... */ }
```

# Hilfe! Ich bin Überladen

- Methoden, kennen wir:

```
public int add (int a, int b) { /* ... */ }
```

- Diese Methode besitzt eine *Signatur*:

# Hilfe! Ich bin Überladen

- Methoden, kennen wir:

```
public int add (int a, int b) { /* ... */ }
```

- Diese Methode besitzt eine *Signatur*: add(int, int). Der Rückgabotyp ist *nicht* Teil!



# Hilfe! Ich bin Überladen

- Methoden, kennen wir:

```
public int add (int a, int b) { /* ... */ }
```

- Diese Methode besitzt eine *Signatur*: `add(int, int)`. Der Rückgabetyp ist *nicht* Teil!
- Wir können mehrere Methoden mit verschiedener Signatur erstellen.

# Hilfe! Ich bin Überladen

- Methoden, kennen wir:

```
public int add (int a, int b) { /* ... */ }
```

- Diese Methode besitzt eine *Signatur*: `add(int, int)`. Der Rückgabetyp ist *nicht* Teil!
- Wir können mehrere Methoden mit verschiedener Signatur erstellen.
- Bisher bekannt: anderer Name.

# Hilfe! Ich bin Überladen

- Methoden, kennen wir:

```
public int add (int a, int b) { /* ... */ }
```

- Diese Methode besitzt eine *Signatur*: `add(int, int)`. Der Rückgabetyp ist *nicht* Teil!
- Wir können mehrere Methoden mit verschiedener Signatur erstellen.
- Bisher bekannt: anderer Name.
- Aber auch: gleicher Name, andere Argumente:

# Hilfe! Ich bin Überladen

- Methoden, kennen wir:

```
public int add (int a, int b) { /* ... */ }
```

- Diese Methode besitzt eine *Signatur*: `add(int, int)`. Der Rückgabotyp ist *nicht* Teil!
- Wir können mehrere Methoden mit verschiedener Signatur erstellen.
- Bisher bekannt: anderer Name.
- Aber auch: gleicher Name, andere Argumente:

```
public int add (int a, int b) { /* ... */ }
```

# Hilfe! Ich bin Überladen

- Methoden, kennen wir:

```
public int add (int a, int b) { /* ... */ }
```

- Diese Methode besitzt eine *Signatur*: add(int, int). Der Rückgabebetyp ist *nicht* Teil!
- Wir können mehrere Methoden mit verschiedener Signatur erstellen.
- Bisher bekannt: anderer Name.
- Aber auch: gleicher Name, andere Argumente:

```
public int add (int a, int b) { /* ... */ }  
public int add (int a, float b) { /* ... */ }
```

# Hilfe! Ich bin Überladen

- Methoden, kennen wir:

```
public int add (int a, int b) { /* ... */ }
```

- Diese Methode besitzt eine *Signatur*: add(int, int). Der Rückgabebetyp ist *nicht* Teil!
- Wir können mehrere Methoden mit verschiedener Signatur erstellen.
- Bisher bekannt: anderer Name.
- Aber auch: gleicher Name, andere Argumente:

```
public int add (int a, int b) { /* ... */ }  
public int add (int a, float b) { /* ... */ }  
public int add (int a) { /* ... */ }
```

# Hilfe! Ich bin Überladen

- Methoden, kennen wir:

```
public int add (int a, int b) { /* ... */ }
```

- Diese Methode besitzt eine *Signatur*: add(int, int). Der Rückgabotyp ist *nicht* Teil!
- Wir können mehrere Methoden mit verschiedener Signatur erstellen.
- Bisher bekannt: anderer Name.
- Aber auch: gleicher Name, andere Argumente:

```
public int add (int a, int b) { /* ... */ }  
public int add (int a, float b) { /* ... */ }  
public int add (int a) { /* ... */ }  
public int add () { /* ... */ }
```

# Prinzip: Überladen



# Prinzip: Überladen

- Wenn wir eine Methode mit gleichem Namen definieren,

# Prinzip: Überladen

- Wenn wir eine Methode mit gleichem Namen definieren, aber einer verschiedenen Signatur,

# Prinzip: Überladen

- Wenn wir eine Methode mit gleichem Namen definieren, aber einer verschiedenen Signatur, so heißt dies: *Überladen*.

# Prinzip: Überladen

- Wenn wir eine Methode mit gleichem Namen definieren, aber einer verschiedenen Signatur, so heißt dies: *Überladen*.
- Java ruft die (seiner Meinung nach!) passende Methode auf.

# Beispiele zur Überladung

- Es sei (die volle Datei: [Ueberladen.java](#)):

# Beispiele zur Überladung

- Es sei (die volle Datei: [Ueberladen.java](#)):

# Beispiele zur Überladung

- Es sei (die volle Datei: [Ueberladen.java](#)):

```
1 public static void print(short a) {  
2     System.out.println("Short:_" + a);  
3 }
```

- Und analog:

# Beispiele zur Überladung

- Es sei (die volle Datei: [Ueberladen.java](#)):

```
1 public static void print(short a) {  
2     System.out.println("Short:_" + a);  
3 }
```

- Und analog:



# Beispiele zur Überladung

- Es sei (die volle Datei: [Ueberladen.java](#)):

```
1 public static void print(short a) {  
2     System.out.println("Short:␣" + a);  
3 }
```

- Und analog:

```
1 public static void print(int a) {  
2     System.out.println("Int:␣" + a);  
3 }
```

- Sowie für `float` und `double`.

# Beispiele zur Überladung

# Beispiele zur Überladung

- Was ergibt:

# Beispiele zur Überladung

- Was ergibt:

```
print(2); print('a'); print(0.25); print(0.1f);  
print((byte) 15);
```

# Beispiele zur Überladung

- Was ergibt:

```
print(2); print('a'); print(0.25); print(0.1f);  
print((byte) 15);
```

- `java` Ueberladen:

# Beispiele zur Überladung

- Was ergibt:

```
print(2); print('a'); print(0.25); print(0.1f);  
print((byte) 15);
```

- **java** Ueberladen:

```
Int: 2  
Int: 97  
Double: 0.25  
Float: 0.1  
Short: 15
```

# Wie wählt Java?

# Wie wählt Java?

- Die erste Ausgabe (`print(2)` → `Int: 2`) ist logisch.



# Wie wählt Java?

- Die erste Ausgabe (`print(2)` → `Int: 2`) ist logisch. (Eine Konstante ist eine Zahl, in Java ein `int`.)

# Wie wählt Java?

- Die erste Ausgabe (`print(2)` → Int: `2`) ist logisch. (Eine Konstante ist eine Zahl, in Java ein `int`.)
- Java kann bei „`print('a')` → Int: `97`“ den `char`

# Wie wählt Java?

- Die erste Ausgabe (`print(2)` → Int: `2`) ist logisch. (Eine Konstante ist eine Zahl, in Java ein `int`.)
- Java kann bei „`print('a')` → Int: `97`“ den `char` durch implizite Umwandlung zu einem `int` konvertieren.

# Wie wählt Java?

- Die erste Ausgabe (`print(2)` → `Int: 2`) ist logisch. (Eine Konstante ist eine Zahl, in Java ein `int`.)
- Java kann bei „`print('a')` → `Int: 97`“ den `char` durch implizite Umwandlung zu einem `int` konvertieren.
- „`print(0.25)` → `Double: 0.25`“

# Wie wählt Java?

- Die erste Ausgabe (`print(2)` → `Int: 2`) ist logisch. (Eine Konstante ist eine Zahl, in Java ein `int`.)
- Java kann bei „`print('a')` → `Int: 97`“ den `char` durch implizite Umwandlung zu einem `int` konvertieren.
- „`print(0.25)` → `Double: 0.25`“ da Fließkommazahlen in Java standardmäßig `double` sind.

# Wie wählt Java?

- Die erste Ausgabe (`print(2)` → `Int: 2`) ist logisch. (Eine Konstante ist eine Zahl, in Java ein `int`.)
- Java kann bei „`print('a')` → `Int: 97`“ den `char` durch implizite Umwandlung zu einem `int` konvertieren.
- „`print(0.25)` → `Double: 0.25`“ da Fließkommazahlen in Java standardmäßig `double` sind.
- „`print(0.1f)` → `Float: 0.1`“

# Wie wählt Java?

- Die erste Ausgabe (`print(2)` → `Int: 2`) ist logisch. (Eine Konstante ist eine Zahl, in Java ein `int`.)
- Java kann bei „`print('a')` → `Int: 97`“ den `char` durch implizite Umwandlung zu einem `int` konvertieren.
- „`print(0.25)` → `Double: 0.25`“ da Fließkommazahlen in Java standardmäßig `double` sind.
- „`print(0.1f)` → `Float: 0.1`“ hier konvertieren wir die Zahl explizit!

# Wie wählt Java?

- Die erste Ausgabe (`print(2)` → `Int: 2`) ist logisch. (Eine Konstante ist eine Zahl, in Java ein `int`.)
- Java kann bei „`print('a')` → `Int: 97`“ den `char` durch implizite Umwandlung zu einem `int` konvertieren.
- „`print(0.25)` → `Double: 0.25`“ da Fließkommazahlen in Java standardmäßig `double` sind.
- „`print(0.1f)` → `Float: 0.1`“ hier konvertieren wir die Zahl explizit!
- Bei „`print((byte) 15)` → `Short: 15`“



# Wie wählt Java?

- Die erste Ausgabe (`print(2)` → `Int: 2`) ist logisch. (Eine Konstante ist eine Zahl, in Java ein `int`.)
- Java kann bei „`print('a')` → `Int: 97`“ den `char` durch implizite Umwandlung zu einem `int` konvertieren.
- „`print(0.25)` → `Double: 0.25`“ da Fließkommazahlen in Java standardmäßig `double` sind.
- „`print(0.1f)` → `Float: 0.1`“ hier konvertieren wir die Zahl explizit!
- Bei „`print((byte) 15)` → `Short: 15`“ findet Java kein `print(byte)`,

# Wie wählt Java?

- Die erste Ausgabe (`print(2)` → `Int: 2`) ist logisch. (Eine Konstante ist eine Zahl, in Java ein `int`.)
- Java kann bei „`print('a')` → `Int: 97`“ den `char` durch implizite Umwandlung zu einem `int` konvertieren.
- „`print(0.25)` → `Double: 0.25`“ da Fließkommazahlen in Java standardmäßig `double` sind.
- „`print(0.1f)` → `Float: 0.1`“ hier konvertieren wir die Zahl explizit!
- Bei „`print((byte) 15)` → `Short: 15`“ findet Java kein `print(byte)`, konvertiert aber nicht zu `print(int)` sondern sucht das „passendste“ `print(short)`.

# Exkurs: Matrizen in Java

# Exkurs: Matrizen in Java

- Wir kennen:

# Exkurs: Matrizen in Java

- Wir kennen: Arrays!

# Exkurs: Matrizen in Java

- Wir kennen: Arrays!

```
int[] arr = {3, 5, 7};
```

# Exkurs: Matrizen in Java

- Wir kennen: Arrays!

```
int[] arr = {3, 5, 7};
```

# Exkurs: Matrizen in Java

- Wir kennen: Arrays!

Array  
`int[] arr = {3, 5, 7};`



# Exkurs: Matrizen in Java

- Wir kennen: Arrays!

Array  
`int[] arr = {3, 5, 7};`  
Integer

# Exkurs: Matrizen in Java

- Wir kennen: Arrays!

Array  
int[] arr = {3, 5, 7};  
Integer  
arr[0]

The diagram illustrates the declaration and access of a Java array. The code `int[] arr = {3, 5, 7};` is shown. A bracket above the `[]` is labeled "Array". A bracket below the `int` is labeled "Integer". An arrow points from `arr[0]` to the first element of the array, `3`, in the initialization set.

# Exkurs: Matrizen in Java

- Wir kennen: Arrays!

Diagram illustrating the declaration and initialization of an array in Java:

```
int[] arr = {3, 5, 7};
```

The diagram includes the following annotations:

- Array**: A bracket above the `[]` indicating the array type.
- Integer**: A bracket below the `int` type.
- arr[0]**: An arrow points from the first element `3` to the index `0`.
- arr[1]**: An arrow points from the second element `5` to the index `1`.

# Exkurs: Matrizen in Java

- Wir kennen: Arrays!

Diagram illustrating an array declaration and access in Java:

```
int[] arr = {3, 5, 7};
```

The diagram shows the following components:

- int**: Labeled as **Integer** (underneath).
- []**: Labeled as **Array** (above).
- arr**: The array variable name.
- {3, 5, 7}**: The array elements.
- arr[0]**, **arr[1]**, and **arr[2]**: Array indices pointing to the elements 3, 5, and 7 respectively.

# Exkurs: Matrizen in Java

- Wir kennen: Arrays!

Diagram illustrating an array declaration and its elements:

```
int[] arr = {3, 5, 7};
```

Annotations:

- `int` is labeled as **Integer**.
- `[]` is labeled as **Array**.
- `arr.length` is labeled as `→ 3`.
- The elements `3`, `5`, and `7` are mapped to `arr[0]`, `arr[1]`, and `arr[2]` respectively.

# Exkurs: Matrizen in Java

- Wir kennen: Arrays!

Diagram illustrating an array declaration and initialization in Java:

```
int[] arr = {3, 5, 7};
```

The diagram includes the following annotations:

- Array**: A bracket above the `[]` part of the declaration.
- Integer**: A bracket below the `int` type.
- arr.length → 3**: A bracket above the curly braces in the initialization, indicating the array's length.
- arr[0]**, **arr[1]**, and **arr[2]**: Arrows pointing from these labels to the elements 3, 5, and 7 respectively in the initialization.

- Wir wollen: Matrizen!

# Exkurs: Matrizen in Java

- Wir kennen: Arrays!

Diagram illustrating an array declaration and initialization in Java:

```
int[] arr = {3, 5, 7};
```

The diagram includes annotations:

- `int` is labeled "Integer".
- `[]` is labeled "Array".
- `arr.length` is labeled `arr.length → 3`.
- Arrows point from the values `3`, `5`, and `7` to their respective indices: `arr[0]`, `arr[1]`, and `arr[2]`.

- Wir wollen: Matrizen!

Diagram illustrating a 2D array (matrix) structure:

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

# Exkurs: Matrizen in Java

- Wir kennen: Arrays!

Diagram illustrating an array declaration and initialization in Java:

```
int[] arr = {3, 5, 7};
```

The diagram includes annotations:

- `int` is labeled "Integer".
- `[]` is labeled "Array".
- `arr` is the variable name.
- `{3, 5, 7}` is the array content.
- `arr.length` is labeled `arr.length → 3`.
- Arrows point from `arr[0]`, `arr[1]`, and `arr[2]` to the elements `3`, `5`, and `7` respectively.

- Wir wollen: Matrizen!

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

Doch wie in Java?



# Exkurs: Matrizen in Java

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

# Exkurs: Matrizen in Java

- Recht leicht:

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

# Exkurs: Matrizen in Java

- Recht leicht: Array von Arrays!

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

# Exkurs: Matrizen in Java

- Recht leicht: Array von Arrays!

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

```
int[][] arr = {{1,2,3}, {4,5,6}};
```

# Exkurs: Matrizen in Java

- Recht leicht: Array von Arrays!

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

```
int[][] arr = {{1,2,3}, {4,5,6}};
```

# Exkurs: Matrizen in Java

- Recht leicht: Array von Arrays!

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

Array  
`int[][] arr = {{1,2,3}, {4,5,6}};`

# Exkurs: Matrizen in Java

- Recht leicht: Array von Arrays!

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

$\underbrace{\text{int}[]}_{\text{Arrays}} \underbrace{[]}_{\text{Array}} \text{ arr} = \{ \{1, 2, 3\}, \{4, 5, 6\} \};$

# Exkurs: Matrizen in Java

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

- Recht leicht: Array von Arrays!

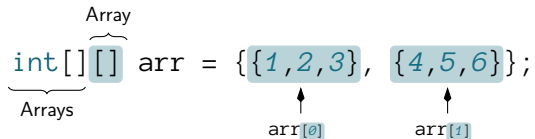
`int[] [] arr = {{1, 2, 3}, {4, 5, 6}};`



# Exkurs: Matrizen in Java

- Recht leicht: Array von Arrays!

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

  
The diagram shows the Java code `int[][] arr = {{1,2,3}, {4,5,6}};` with several annotations. A bracket under `int` is labeled "Arrays". A bracket over the second pair of brackets `[]` is labeled "Array". The first inner array `{1,2,3}` is highlighted in light blue, with an arrow pointing to it from the label `arr[0]`. The second inner array `{4,5,6}` is also highlighted in light blue, with an arrow pointing to it from the label `arr[1]`.

```
int[][] arr = {{1,2,3}, {4,5,6}};
```

# Exkurs: Matrizen in Java

- Recht leicht: Array von Arrays!

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

The diagram shows the declaration and initialization of a 2D array. The declaration is `int[][] arr`. Above the second set of brackets is a bracket labeled "Array". Below the first set of brackets is a bracket labeled "Arrays". The initialization is `{ {1, 2, 3}, {4, 5, 6} }`. Above this is a bracket labeled `arr.length → 2`. Below the first inner array `{1, 2, 3}` is an arrow pointing to `arr[0]`. Below the second inner array `{4, 5, 6}` is an arrow pointing to `arr[1]`. The numbers 1, 2, 3 and 4, 5, 6 are highlighted in light blue.

```
int[][] arr = { {1, 2, 3}, {4, 5, 6} };
```

# Exkurs: Matrizen in Java

- Recht leicht: Array von Arrays!

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

`int[][] arr = {{1, 2, 3}, {4, 5, 6}};`

Diagram illustrating the structure of a 2D array (Matrix) in Java:

- `int[][]` is annotated with "Arrays" under the first `[]` and "Array" above the second `[]`.
- `arr` is the variable name.
- `arr.length` is annotated with `→ 2`, indicating the number of sub-arrays.
- The first sub-array `{1, 2, 3}` is pointed to by `arr[0]`.
- The second sub-array `{4, 5, 6}` is pointed to by `arr[1]`, which is annotated with "Wieder ein Array" (Again an array).

# Exkurs: Matrizen in Java

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

- Recht leicht: Array von Arrays!

`int[][] arr = { {1, 2, 3}, {4, 5, 6} };`

Diagram illustrating the structure of a 2D array (Matrix) in Java:

- `int`: Data type (integers).
- `int[]`: Array of integers (labeled "Arrays").
- `int[][]`: Array of arrays (labeled "Array").
- `arr.length`: Points to 2, indicating the number of rows.
- `arr[0]`: Points to the first row `{1, 2, 3}`.
- `arr[1]`: Points to the second row `{4, 5, 6}`, labeled "Wieder ein Array" (Again an array).

- So gilt:

# Exkurs: Matrizen in Java

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

- Recht leicht: Array von Arrays!

```
int[] Arrays Array arr = { {1, 2, 3}, {4, 5, 6} };
                        arr.length -> 2
                        ^           ^
                        |           |
                    arr[0]       arr[1] ← Wieder ein Array
```

- So gilt: `arr[0]`  $\rightarrow$  {1, 2, 3}

# Exkurs: Matrizen in Java

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

- Recht leicht: Array von Arrays!

`int[][] arr = {{1, 2, 3}, {4, 5, 6}};`

- So gilt: `arr[0]`  $\rightarrow$  `{1, 2, 3}`
- Mehrdimensionale Arrays sind also

# Exkurs: Matrizen in Java

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

- Recht leicht: Array von Arrays!

```
int[] [] arr = {{1, 2, 3}, {4, 5, 6}};
```

Annotations:   
- `int`: Arrays   
- `[] []`: Array   
- `arr.length`: `2`   
- `arr[0]`: points to `{1, 2, 3}`   
- `arr[1]`: points to `{4, 5, 6}` (labeled "Wieder ein Array")

- So gilt: `arr[0]`  $\rightarrow$  `{1, 2, 3}`
- Mehrdimensionale Arrays sind also „ein Array aus Arrays von Integern“.

# Exkurs: Matrizen in Java

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

- Recht leicht: Array von Arrays!

```
int[][] arr = {{1, 2, 3}, {4, 5, 6}};
```

Annotations:   
- `int[]` is labeled "Arrays"   
- `[]` is labeled "Array"   
- `arr.length` is labeled `arr.length → 2`   
- `arr[0]` points to the first inner array `{1, 2, 3}`   
- `arr[1]` points to the second inner array `{4, 5, 6}`, which is also labeled "Wieder ein Array" (Again an array).

- So gilt: `arr[0] → {1, 2, 3}`
- Mehrdimensionale Arrays sind also „ein Array aus Arrays von Integern“.
- Kompliziert?



# Exkurs: Matrizen in Java

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

- Recht leicht: Array von Arrays!

```
int[] [] arr = { {1, 2, 3}, {4, 5, 6} };
```

arr.length  $\rightarrow$  2

arr[0]      arr[1] ← Wieder ein Array

- So gilt:  $\text{arr}[0] \rightarrow \{1, 2, 3\}$
- Mehrdimensionale Arrays sind also „ein Array aus Arrays von Integern“.
- Kompliziert? Das Muster lässt sich durch Übung leicht einprägen.

# Exkurs: Matrizen in Java

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

- Recht leicht: Array von Arrays!

```
int[] [] arr = { {1, 2, 3}, {4, 5, 6} };
```

arr.length → 2

arr[0]      arr[1] ← Wieder ein Array

- So gilt:  $\text{arr}[0] \rightarrow \{1, 2, 3\}$
- Mehrdimensionale Arrays sind also „ein Array aus Arrays von Integern“.
- Kompliziert? Das Muster lässt sich durch Übung leicht einprägen.
- Vorerst:

# Exkurs: Matrizen in Java

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

- Recht leicht: Array von Arrays!

`int[][] arr = {{1, 2, 3}, {4, 5, 6}};`  
Annotations: `int[]` (Arrays), `arr` (Array), `arr.length` (2), `arr[0]` points to `{1, 2, 3}`, `arr[1]` points to `{4, 5, 6}` (Wieder ein Array).

- So gilt: `arr[0] → {1, 2, 3}`
- Mehrdimensionale Arrays sind also „ein Array aus Arrays von Integern“.
- Kompliziert? Das Muster lässt sich durch Übung leicht einprägen.
- Vorerst: `arr.length` = Anzahl Zeilen

# Exkurs: Matrizen in Java

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

- Recht leicht: Array von Arrays!

```
int[] [] arr = { {1, 2, 3}, {4, 5, 6} };
```

arr.length  $\rightarrow$  2

arr[0]      arr[1] ← Wieder ein Array

- So gilt:  $\text{arr}[0] \rightarrow \{1, 2, 3\}$
- Mehrdimensionale Arrays sind also „ein Array aus Arrays von Integern“.
- Kompliziert? Das Muster lässt sich durch Übung leicht einprägen.
- Vorerst: `arr.length` = Anzahl Zeilen und

# Exkurs: Matrizen in Java

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

- Recht leicht: Array von Arrays!

```
int[] [] arr = { {1, 2, 3}, {4, 5, 6} };
```

arr.length → 2

arr[0]      arr[1] ← Wieder ein Array

- So gilt:  $\text{arr}[0] \rightarrow \{1, 2, 3\}$
- Mehrdimensionale Arrays sind also „ein Array aus Arrays von Integern“.
- Kompliziert? Das Muster lässt sich durch Übung leicht einprägen.
- Vorerst:  $\text{arr.length}$  = Anzahl Zeilen und  $\text{arr}[0].length$  = Anzahl Spalten.

# Exkurs: Matrizen multiplizieren

# Exkurs: Matrizen multiplizieren

- Sollte eigentlich aus Mathematikveranstaltungen bekannt sein.

# Exkurs: Matrizen multiplizieren

- Sollte eigentlich aus Mathematikveranstaltungen bekannt sein.
- Seien  $A, B$  Matrizen.



# Exkurs: Matrizen multiplizieren

- Sollte eigentlich aus Mathematikveranstaltungen bekannt sein.
- Seien  $A, B$  Matrizen. Genauer:  $A \in \mathbb{R}^{a,k}$  und  $B \in \mathbb{R}^{k,b}$  mit  $a, k, b \in \mathbb{N}$ .

# Exkurs: Matrizen multiplizieren

- Sollte eigentlich aus Mathematikveranstaltungen bekannt sein.
- Seien  $A, B$  Matrizen. Genauer:  $A \in \mathbb{R}^{a,k}$  und  $B \in \mathbb{R}^{k,b}$  mit  $a, k, b \in \mathbb{N}$ . Dann gilt für  $C = A \cdot B$ , dass  $C \in \mathbb{R}^{a,b}$ .

# Exkurs: Matrizen multiplizieren

- Sollte eigentlich aus Mathematikveranstaltungen bekannt sein.
- Seien  $A, B$  Matrizen. Genauer:  $A \in \mathbb{R}^{a,k}$  und  $B \in \mathbb{R}^{k,b}$  mit  $a, k, b \in \mathbb{N}$ . Dann gilt für  $C = A \cdot B$ , dass  $C \in \mathbb{R}^{a,b}$ .
- Kurz:

# Exkurs: Matrizen multiplizieren

- Sollte eigentlich aus Mathematikveranstaltungen bekannt sein.
- Seien  $A, B$  Matrizen. Genauer:  $A \in \mathbb{R}^{a,k}$  und  $B \in \mathbb{R}^{k,b}$  mit  $a, k, b \in \mathbb{N}$ . Dann gilt für  $C = A \cdot B$ , dass  $C \in \mathbb{R}^{a,b}$ .
- Kurz: „Spalten von  $A$  = Zeilen von  $B$ “.

# Exkurs: Matrizen multiplizieren

- Sollte eigentlich aus Mathematikveranstaltungen bekannt sein.
- Seien  $A, B$  Matrizen. Genauer:  $A \in \mathbb{R}^{a,k}$  und  $B \in \mathbb{R}^{k,b}$  mit  $a, k, b \in \mathbb{N}$ . Dann gilt für  $C = A \cdot B$ , dass  $C \in \mathbb{R}^{a,b}$ .
- Kurz: „Spalten von  $A$  = Zeilen von  $B$ “. (Java: `a[0].length == b.length`).

# Exkurs: Matrizen multiplizieren

- Sollte eigentlich aus Mathematikveranstaltungen bekannt sein.
- Seien  $A, B$  Matrizen. Genauer:  $A \in \mathbb{R}^{a,k}$  und  $B \in \mathbb{R}^{k,b}$  mit  $a, k, b \in \mathbb{N}$ . Dann gilt für  $C = A \cdot B$ , dass  $C \in \mathbb{R}^{a,b}$ .
- Kurz: „Spalten von  $A$  = Zeilen von  $B$ “. (Java: `a[0].length == b.length`).
- Formal berechnet sich jeder Eintrag  $c_{z,s}$  (Zeile  $z$ , Spalte  $s$ ) über:

# Exkurs: Matrizen multiplizieren

- Sollte eigentlich aus Mathematikveranstaltungen bekannt sein.
- Seien  $A, B$  Matrizen. Genauer:  $A \in \mathbb{R}^{a,k}$  und  $B \in \mathbb{R}^{k,b}$  mit  $a, k, b \in \mathbb{N}$ . Dann gilt für  $C = A \cdot B$ , dass  $C \in \mathbb{R}^{a,b}$ .
- Kurz: „Spalten von  $A$  = Zeilen von  $B$ “. (Java: `a[0].length == b.length`).
- Formal berechnet sich jeder Eintrag  $c_{z,s}$  (Zeile  $z$ , Spalte  $s$ ) über:

$$c_{z,s} = \sum_{j=1}^k a_{z,j} \cdot b_{j,s}.$$

# Exkurs: Matrizen multiplizieren

- Sollte eigentlich aus Mathematikveranstaltungen bekannt sein.
- Seien  $A, B$  Matrizen. Genauer:  $A \in \mathbb{R}^{a,k}$  und  $B \in \mathbb{R}^{k,b}$  mit  $a, k, b \in \mathbb{N}$ . Dann gilt für  $C = A \cdot B$ , dass  $C \in \mathbb{R}^{a,b}$ .
- Kurz: „Spalten von  $A$  = Zeilen von  $B$ “. (Java: `a[0].length == b.length`).
- Formal berechnet sich jeder Eintrag  $c_{z,s}$  (Zeile  $z$ , Spalte  $s$ ) über:

$$c_{z,s} = \sum_{j=1}^k a_{z,j} \cdot b_{j,s}.$$

- Nicht anschaulich? Vermutlich.



# Exkurs: Matrizen multiplizieren – anschaulich

# Exkurs: Matrizen multiplizieren – anschaulich

- Seien A und B:

# Exkurs: Matrizen multiplizieren – anschaulich

- Seien A und B:

$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix},$$

# Exkurs: Matrizen multiplizieren – anschaulich

- Seien A und B:

$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

# Exkurs: Matrizen multiplizieren – anschaulich

- Seien A und B:

$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

- Anzahl Spalten von A entspricht Anzahl Zeilen von B

# Exkurs: Matrizen multiplizieren – anschaulich

- Seien A und B:

$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

- Anzahl Spalten von A entspricht Anzahl Zeilen von B  $\Rightarrow A \cdot B = C \in \mathbb{R}^{2,2}$ .

# Exkurs: Matrizen multiplizieren – anschaulich

- Seien A und B:

$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

- Anzahl Spalten von A entspricht Anzahl Zeilen von B  $\Rightarrow A \cdot B = C \in \mathbb{R}^{2,2}$ .
- Zur Anschaulichkeit schreiben wir sie um!

# Exkurs: Matrizen multiplizieren – anschaulich

$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$



# Exkurs: Matrizen multiplizieren – anschaulich

- Seien A und B formatiert als:

$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

# Exkurs: Matrizen multiplizieren – anschaulich

- Seien A und B formatiert als:

$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

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

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

# Exkurs: Matrizen multiplizieren – anschaulich

- Seien A und B formatiert als:

$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

$$\begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix} \quad B$$

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

A

# Exkurs: Matrizen multiplizieren – anschaulich

- Seien A und B formatiert als:

$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

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

A

# Exkurs: Matrizen multiplizieren – anschaulich

- Seien A und B formatiert als:

$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

$$\begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix} \quad B$$

$$\begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix} \quad \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

A

- Berechnen wir zunächst  $a \hat{=} c_{1,1}$

# Exkurs: Matrizen multiplizieren – anschaulich

- Seien A und B formatiert als:

$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

$$\begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix} \quad B$$

$$\begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix} \quad \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

A

- Berechnen wir zunächst  $a \hat{=} c_{1,1}$

(Die Multiplikation ist möglich, da die linke Matrix so „breit“, wie die obere „hoch“ ist.)

# Exkurs: Matrizen multiplizieren – anschaulich, a

■  $A \cdot B$ :

$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

## Exkurs: Matrizen multiplizieren – anschaulich, a

- $A \cdot B$ :

$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

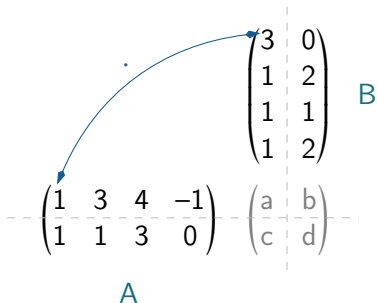
$$\begin{array}{c|c} \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix} & \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix} \\ \hline \text{A} & \text{B} \end{array}$$

Für a ergibt sich damit:  
a =



# Exkurs: Matrizen multiplizieren – anschaulich, a

■  $A \cdot B$ :

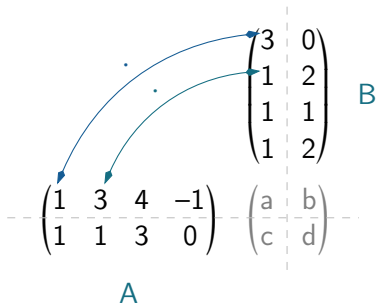


$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

Für a ergibt sich damit:  
 $a = 1 \cdot 3$

# Exkurs: Matrizen multiplizieren – anschaulich, a

■  $A \cdot B$ :



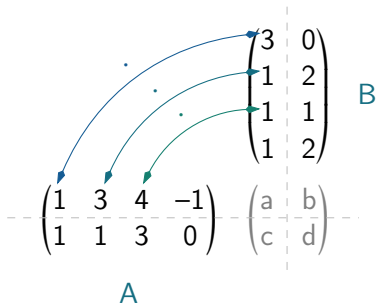
$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

Für a ergibt sich damit:

$$a = 1 \cdot 3 + 3 \cdot 1$$

# Exkurs: Matrizen multiplizieren – anschaulich, a

■  $A \cdot B$ :



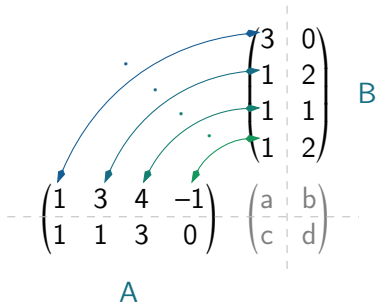
$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

Für a ergibt sich damit:

$$a = 1 \cdot 3 + 3 \cdot 1 + 4 \cdot 1$$

# Exkurs: Matrizen multiplizieren – anschaulich, a

■  $A \cdot B$ :



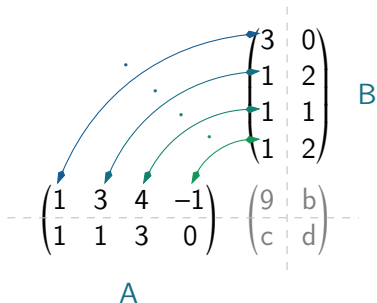
$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

Für  $a$  ergibt sich damit:

$$a = 1 \cdot 3 + 3 \cdot 1 + 4 \cdot 1 + (-1) \cdot 1$$

# Exkurs: Matrizen multiplizieren – anschaulich, a

■  $A \cdot B$ :



$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

Für  $a$  ergibt sich damit:

$$a = 1 \cdot 3 + 3 \cdot 1 + 4 \cdot 1 + (-1) \cdot 1 \\ = 9$$

# Exkurs: Matrizen multiplizieren – anschaulich, b

■  $A \cdot B$ :

$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

# Exkurs: Matrizen multiplizieren – anschaulich, b

■  $A \cdot B$ :

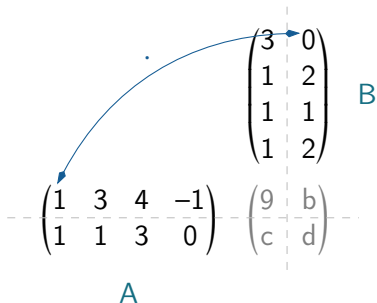
$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

$$\begin{array}{c} \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix} \end{array} \begin{array}{c} \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix} \end{array} \begin{array}{c} A \\ B \end{array}$$
$$\begin{array}{c} \begin{pmatrix} 9 & b \\ c & d \end{pmatrix} \end{array}$$

Für  $b$  ergibt sich damit:  
 $b =$

# Exkurs: Matrizen multiplizieren – anschaulich, b

■  $A \cdot B$ :



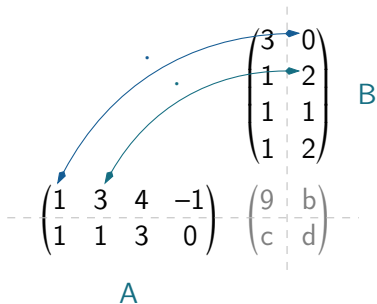
$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

Für b ergibt sich damit:  
 $b = 1 \cdot 0$



# Exkurs: Matrizen multiplizieren – anschaulich, b

■  $A \cdot B$ :



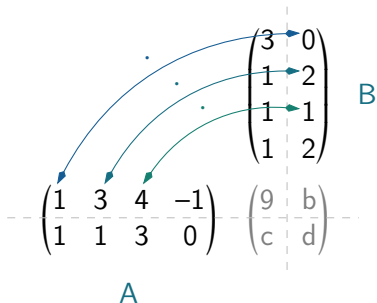
$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

Für  $b$  ergibt sich damit:

$$b = 1 \cdot 0 + 3 \cdot 2$$

# Exkurs: Matrizen multiplizieren – anschaulich, b

■  $A \cdot B$ :



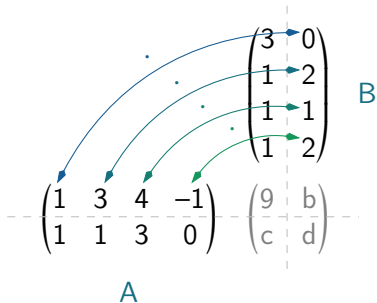
$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

Für b ergibt sich damit:

$$b = 1 \cdot 0 + 3 \cdot 2 + 4 \cdot 1$$

# Exkurs: Matrizen multiplizieren – anschaulich, b

■  $A \cdot B$ :



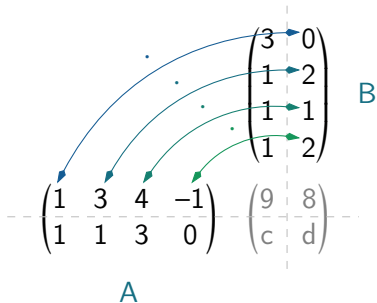
$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

Für  $b$  ergibt sich damit:

$$b = 1 \cdot 0 + 3 \cdot 2 + 4 \cdot 1 + (-1) \cdot 2$$

# Exkurs: Matrizen multiplizieren – anschaulich, b

■  $A \cdot B$ :



$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

Für  $b$  ergibt sich damit:

$$b = 1 \cdot 0 + 3 \cdot 2 + 4 \cdot 1 + (-1) \cdot 2 \\ = 8$$

# Exkurs: Matrizen multiplizieren – anschaulich, c

■  $A \cdot B$ :

$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

# Exkurs: Matrizen multiplizieren – anschaulich, c

■  $A \cdot B$ :

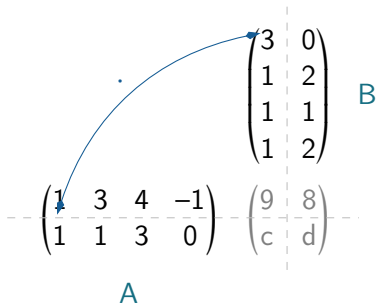
$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

$$\begin{array}{ccccc} & & & \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix} & B \\ \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix} & \begin{pmatrix} 9 & 8 \\ c & d \end{pmatrix} & & & \\ A & & & & \end{array}$$

Für  $c$  ergibt sich damit:  
 $c =$

# Exkurs: Matrizen multiplizieren – anschaulich, c

■  $A \cdot B$ :

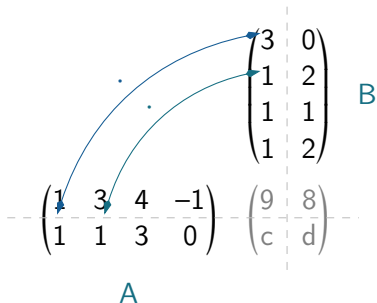


$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

Für c ergibt sich damit:  
 $c = 1 \cdot 3$

# Exkurs: Matrizen multiplizieren – anschaulich, c

■  $A \cdot B$ :



$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

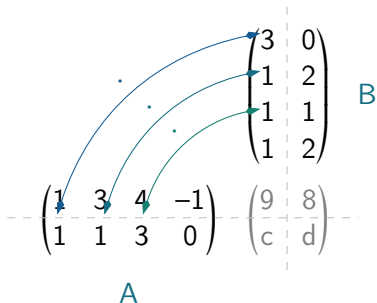
Für c ergibt sich damit:

$$c = 1 \cdot 3 + 1 \cdot 1$$



# Exkurs: Matrizen multiplizieren – anschaulich, c

■  $A \cdot B$ :



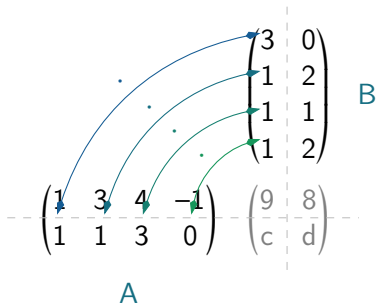
$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

Für c ergibt sich damit:

$$c = 1 \cdot 3 + 1 \cdot 1 + 3 \cdot 1$$

# Exkurs: Matrizen multiplizieren – anschaulich, c

■  $A \cdot B$ :



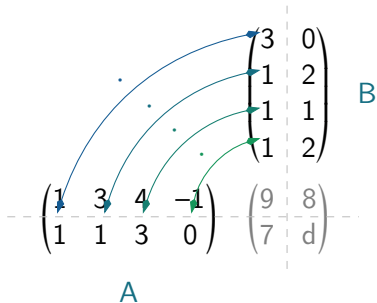
$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

Für  $c$  ergibt sich damit:

$$c = 1 \cdot 3 + 1 \cdot 1 + 3 \cdot 1 + 0 \cdot 1$$

# Exkurs: Matrizen multiplizieren – anschaulich, c

■  $A \cdot B$ :



$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

Für c ergibt sich damit:

$$c = 1 \cdot 3 + 1 \cdot 1 + 3 \cdot 1 + 0 \cdot 1 \\ = 7$$

# Exkurs: Matrizen multiplizieren – anschaulich, d

■  $A \cdot B$ :

$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

# Exkurs: Matrizen multiplizieren – anschaulich, d

■  $A \cdot B$ :

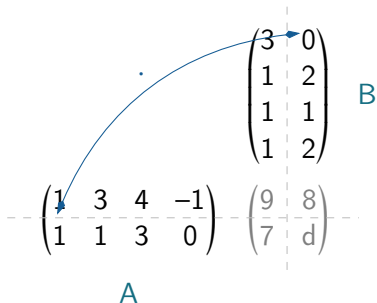
$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

$$\begin{array}{ccccc} & & & \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix} & B \\ \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix} & A & \cdot & \begin{pmatrix} 9 & 8 \\ 7 & d \end{pmatrix} & \end{array}$$

Für  $d$  ergibt sich damit:  
 $d =$

# Exkurs: Matrizen multiplizieren – anschaulich, d

■  $A \cdot B$ :

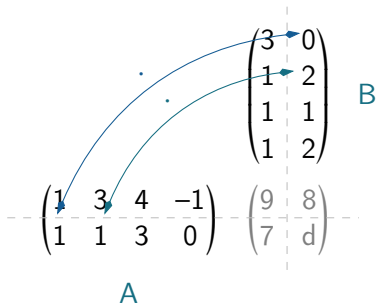


$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

Für d ergibt sich damit:  
 $d = 1 \cdot 0$

# Exkurs: Matrizen multiplizieren – anschaulich, d

■  $A \cdot B$ :

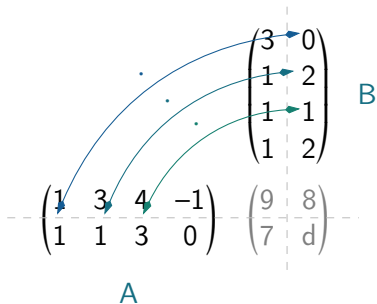


$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

Für d ergibt sich damit:  
 $d = 1 \cdot 0 + 1 \cdot 2$

# Exkurs: Matrizen multiplizieren – anschaulich, d

■  $A \cdot B$ :



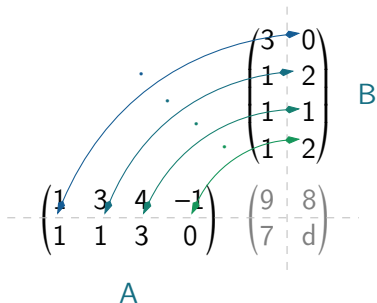
$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

Für d ergibt sich damit:  
 $d = 1 \cdot 0 + 1 \cdot 2 + 3 \cdot 1$



# Exkurs: Matrizen multiplizieren – anschaulich, d

■  $A \cdot B$ :



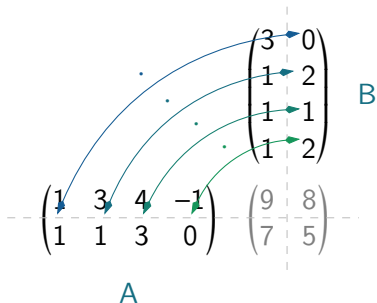
$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

Für d ergibt sich damit:

$$d = 1 \cdot 0 + 1 \cdot 2 + 3 \cdot 1 + 0 \cdot 2$$

# Exkurs: Matrizen multiplizieren – anschaulich, d

■  $A \cdot B$ :



$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

Für d ergibt sich damit:

$$d = 1 \cdot 0 + 1 \cdot 2 + 3 \cdot 1 + 0 \cdot 2 \\ = 5$$

# Exkurs: Matrizen multiplizieren – anschaulich, d

■  $A \cdot B$ :

$$A = \begin{pmatrix} 1 & 3 & 4 & -1 \\ 1 & 1 & 3 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 \\ 1 & 2 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$$

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

A

Für d ergibt sich damit:

$$d = 1 \cdot 0 + 1 \cdot 2 + 3 \cdot 1 + 0 \cdot 2 \\ = 5$$

■ Damit gilt:  $C = \begin{pmatrix} 9 & 8 \\ 7 & 5 \end{pmatrix}$ .



