

# Programmieren 1

## Übungsserie 5

---

### Stoff

- Bis zu Kapitel 8
- Fokus: Arrays, Statische Methoden, Lesen aus Dateien

---

### Allgemeine Informationen zur Abgabe

- Die Abgabe erfolgt online auf ILIAS.
- Quellcode zu den Implementationsaufgaben muss als **\*.zip** Datei abgegeben werden. Exportieren Sie hierzu Ihr Projekt direkt aus Eclipse. Quellcode, den wir nicht kompilieren können, wird nicht akzeptiert.
- Lösungen zu den Theorieaufgaben muss als **\*.pdf** Datei abgegeben werden. Andere Formate werden nicht akzeptiert.
- Arbeit in Zweiergruppen: Geben Sie jeweils nur ein Exemplar der Lösung pro Gruppe ab. Geben Sie in jeder Quellcode-Datei die *Namen und Matrikelnummern* beider Gruppenmitglieder in den ersten beiden Zeilen als Kommentar an.
- Vorbesprechung: *03.11.2023*
- Abgabe: *17.11.2023 13:00 Uhr*

---

### Anmerkung

- Organisiert die Aufgaben in Packages. Sprich ein Package **game** für Aufgabe 1 und ein Package **matrizen** für Aufgabe 2.

### Implementationsaufgaben

1. Sie sollen ein “Vier gewinnt“ Spiel programmieren, bei dem man in der Konsole gegen eine zweite Person spielen kann.

Laden Sie von ILIAS die Dateien **VierGewinnt.java**, **Player.java** und **Token.java** herunter. Die Klasse **VierGewinnt** enthält bereits Methoden **play()** (definiert den Spielablauf), **main** (startet das Spiel) und **displayField()** (graphische Darstellung des Spielfelds):

```

Player X choose a column between 1 and 7: 2
+---+---+---+---+---+---+
| | | | | | | |
+---+---+---+---+---+---+
| | | | | | | |
+---+---+---+---+---+---+
| X | | | | | | |
+---+---+---+---+---+---+
| O | X | O | | | | |
+---+---+---+---+---+---+
| X | O | X | | | | X |
+---+---+---+---+---+---+
| O | O | O | X | X | | O |
+---+---+---+---+---+---+
  1  2  3  4  5  6  7
Player X wins!

```

Um das Spiel zum Laufen zu bekommen, müssen Sie in der Klasse **VierGewinnt** die folgenden Methoden implementieren (die anderen gegebenen Methoden dürfen Sie *nicht* verändern):

- insertToken**: Der übergebene Stein (**Token**-Objekt) soll in die gewählte Spalte (**column**) des Spielfelds (**Array[][] board**) gefüllt werden. Falls eine nicht existierende oder bereits bis oben gefüllte Spalte gewählt wurde, soll das Programm mit einer Fehlermeldung abbrechen. Verwenden Sie dazu **System.exit(1)**.
- isBoardFull**: gibt genau dann **true** zurück, wenn alle Felder durch einen Stein besetzt sind.
- checkVierGewinnt**: überprüft – ausgehend vom durch **col** und **row** gegebenen Feld – ob es in einer der vier Richtungen (d.h. **-**, **|**, **/**, **\**) mindestens vier gleiche Steine gibt. In diesem Fall wird **true** zurückgegeben, andernfalls **false**. Tipp: Schreiben Sie für jede der vier Richtungen eine Hilfsmethode.

## 2. Rechnen mit Matrizen

### Anmerkungen:

- Diese Aufgabe erfordert den Gebrauch statischer Methoden und der Klasse **Scanner** zum Einlesen von Daten aus einer Datei. Beides wird nächste Woche in der Vorlesung eingeführt. Wir empfehlen, die folgenden Teilaufgaben erst nach dem Studium von Kapitel 8 zu lösen!
  - In den folgenden Teilaufgaben werden die Matrizen *A*, *B*, *C* als 2-dimensionale Arrays dargestellt (z.B. `int[][] A = new int[m][n];`).
- Schreiben Sie in einer Klasse **MatrixOperations** eine statische Methode **readMatrix**, welche die Daten einer Matrix aus einer Datei einliest. Eine Matrix-Datei sei dabei folgendermassen formatiert:

```

1 2 3
4 5 6

```

Speichern Sie die eingelesenen Daten in einem 2-dimensionalen Array.

- (b) Schreiben Sie in der Klasse `MatrixOperations` eine statische Methode `transpose`<sup>1</sup>, welche eine  $n \times n$  Matrix  $A$  als Parameter erhält und  $A^T$  (die transponierte Matrix  $A$ ) zurückgibt. Falls die übergebene Matrix nicht quadratisch sein sollte, erzeugen Sie eine Fehlermeldung und geben `null` zurück. Beim Transponieren einer Matrix spiegelt man einen Matrixeintrag  $a_{ij}$  an der Diagonalen von  $A$ . Einfach gesagt, aus  $a_{ij}$  wird  $a_{ji}$ .

$$A = \begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix} \text{ und } A^T = \begin{pmatrix} a_{00} & a_{10} & a_{20} \\ a_{01} & a_{11} & a_{21} \\ a_{02} & a_{12} & a_{22} \end{pmatrix}$$

Ihre Methode sollte alle möglichen Werte von  $n$  berücksichtigen.

- (c) Schreiben Sie in der Klasse `MatrixOperations` eine statische Methode `product`<sup>2</sup>, welche ein  $n \times m$  Matrix  $A$  und eine  $m \times l$  Matrix  $B$  als Parameter entgegennimmt und das Produkt  $C = AB$  zurückgibt. Die Dimension von  $C$  ist  $n \times l$  und die Einträge  $c_{ij}$  berechnen sich wie folgt:

$$c_{ij} = \sum_{k=0}^{m-1} a_{ik} b_{kj}$$

Ihre Methode sollte alle möglichen Werte von  $m$ ,  $n$  und  $l$  berücksichtigen. Falls die Anzahl Spalten von  $A$  nicht mit der Anzahl Zeilen von  $B$  übereinstimmen sollte, erzeugen Sie eine Fehlermeldung und geben Sie `null` zurück.

**Beispielelung:**

$$\text{Input: } A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}, B = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

$$\text{Output: } C = \begin{pmatrix} 9 & 12 & 15 \\ 19 & 26 & 33 \\ 29 & 40 & 51 \end{pmatrix}$$

- (d) Erstellen Sie eine Klasse `MatrixTest.java` in der Sie die Methoden `readMatrix`, `transpose` und `product` mit geeigneten Beispielen testen. Achten Sie darauf bei der Methode `transpose` und `product` auch den Fehlerfall zu testen.

<sup>1</sup>[https://de.wikipedia.org/wiki/Transponierte\\_Matrix](https://de.wikipedia.org/wiki/Transponierte_Matrix)

<sup>2</sup><https://de.wikipedia.org/wiki/Matrizenmultiplikation>