# Objektorientierte Programmierung — Blatt 2

Institut für Softwaretechnik und Programmiersprachen | Sommersemester 2024
Matthias Tichy, Raphael Straub und Florian Sihler

Abgabe (git) bis
12. Mai 2024

## Array Essentials

6 ★ 3 🟦 2 🟢

- Ein- und mehrdimensionale Arrays in Java verstehen und einsetzen
- Schleifen im Kontext von Arrays anwenden

Willkommen zum zweiten Übungsblatt! Diese Woche beschäftigen wir uns primär mit Arrays. Diese (unangenehm statische) Datenstruktur glänzt vor allem dann mit Effizienz, wenn sich ihre Größe nicht ändern soll. Es steht Ihnen frei, bei jeder Aufgabe beliebig viele Hilfsfunktionen zu definieren. Das Einbinden einer Library oder das Verwenden der Collections API ist allerdings nicht gestattet.

### **Aufgabe 1:** Into the Matrix

**a**) Simple Determinants ★

Zunächst soll eine Funktion `int simpleDeterminant(int[][] matrix)` geschrieben werden, welche die Determinante einer $2 \times 2$-Matrix berechnet. Stellen Sie zu Beginn sicher, dass es sich tatsächlich um eine Matrix der entsprechenden Größe handelt. Falls nicht, werfen Sie eine `RuntimeException` mittels `throw new RuntimeException()`.

Die Determinante einer $2 \times 2$-Matrix $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ ist definiert als $a \cdot d - b \cdot c$.

**Beispiele:**

```
int[][] matrix = {{1, 2}, {3, 4}};
simpleDeterminant(matrix); //  ▸  -2 (1*4 - 2*3)
```

---

```
int[][] matrix = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
simpleDeterminant(matrix); //  ▸  RuntimeException
```

**b**) Matrix Multiplication ★★

Nun sollen Matrizen, wie in der Schule oder der Mathe-Vorlesung gelernt, miteinander multipliziert werden. Für zwei Matrizen $A$ und $B$ mit den Dimensionen $m \times n$ und $n \times p$ ist das Ergebnis $C = A \cdot B$ eine Matrix der Dimension $m \times p$. Jede Zelle $C_{ij}$ wird berechnet als $\sum_{k=1}^{n} A_{ik} \cdot B_{kj}$ (wir multiplizieren und addieren also die $i$-te Zeile von $A$ mit der $j$-ten Spalte von $B$). Schreiben Sie hierzu eine Funktion `int[][] multiply(int[][] left, int[][] right)`.

In dieser Aufgabe wollen wir Matrizen in beliebiger Größe erlauben. Verifizieren Sie dennoch, dass die Größen kompatibel sind und werfen Sie gegebenenfalls eine Exception.

**Beispiele:**

```
int[][] left = {{1, 2}, {3, 4}};
int[][] right = {{2, 0}, {1, 2}};
multiply(left, right); //  ▸  {{4, 4}, {10, 8}}
```

---

```
int[][] left = {{1, 2, 3, 4}, {5, 6, 7, 8}};
int[][] right = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}, {10, 11, 12}};
multiply(left, right); //  ▸  {{70, 80, 90}, {158, 184, 210}}
```

**c**) Matrix Transposition ★★

Schreiben Sie jetzt eine Funktion `int[][] transpose(int[][] matrix)` welche eine transponierte Version der Matrix zurückgibt (also die Zeilen und Spalten vertauscht).

**Beispiele:**

```
int[][] matrix = {{1, 2, 3}, {4, 5, 6}};
transpose(matrix); //  ▸  {{1, 4}, {2, 5}, {3, 6}}
```

---

```
int[][] matrix = {{1, 2, 3, 4}, {5, 6, 7, 8}};
transpose(matrix); //  ▸  {{1, 5}, {2, 6}, {3, 7}, {4, 8}}
```

**d**) Better Determinants ★★●●

Nun sollen Sie eine Funktion `int determinant(int[][] matrix)` schreiben, die es erlaubt Determinanten von $N \times N$-Matrizen zu berechnen (für beliebige $N > 1$). Wenden Sie hierfür eine Laplace Expansion an.

**Beispiel:**

```
int[][] matrix = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12},
                  {13, 14, 15, 16}};
determinant(matrix); //  ▸  0
```

## **Aufgabe 2:** Game of Life

In dieser Aufgabe implementieren wir das berühmte „Conway's Game of Life".

Die „Welt" in diesem Spiel besteht aus einem zwei-dimensionalen Gitter (für uns fester Größe), das aus quadratischen Zellen besteht. Jede Zelle befindet sich in einem der beiden Zustände: *Dead* oder *Alive*. Machen Sie sich zunächst selbstständig mit Conway's Game of Life und dem Konzept eines zellulären Automaten bekannt, falls Sie die Konzepte noch nicht kennen.

**a**) Command Line Arguments ■

Wir wollen die Möglichkeit die Größe des Spielfelds beim Start des Programms festzulegen. Erweitern Sie hierzu die `void main(String[] args)` Methode so, dass sie zwei Ganzzahlen (integer) als Argumente entgegennimmt, die die Breite und Höhe des Spielfelds definieren. Diese sollen dann beim Start des Programms gesetzt werden können.

**Beispiel:**
```
java Aufgabe2 160 90
```

**Achtung:** Als Vorlage liefern wir ein gradle Projekt, welches über das run plugin gestartet werden sollte. Damit die main-Methode ihrer Klasse ausgeführt wird und nicht die von uns gelieferte main-Klasse, müssen sie den Wert von `'mainClassName'` in der `build.gradle` Datei anpassen. Achten Sie darauf, dass sie den richtigen Paket-Namen verwenden. Um Argumente zu übergeben, verwenden sie den folgenden Befehl:

**Beispiel:**
```
./gradlew run --args="160 90"
```

Für diese Aufgabe sind keine Tests vorgesehen. Daher ist es nicht notwendig die `@Exercise`-Annotation zu verwenden.

**b**) Drawing the Grid ★

In jeder Iteration wollen wir das Spielfeld in der Kommandozeile darstellen. Schreiben Sie daher eine Funktion

- `void drawGrid(boolean[][] grid)` *oder*
- `void drawGrid(boolean[] grid, int width)`,

welche das Spielfeld zeichnet. Entscheiden Sie selbstständig, welche Repräsentation des Grids verwendet werden soll (beide funktionieren).

Verifizieren Sie vor dem Zeichnen, dass das Grid wirklich rechteckig ist und werfen Sie eine `RuntimeException` falls nicht.

**Beispiel:**
Für ein zweidimensionales Array gilt:

```
boolean[][] grid = {{true, false, false}, {false, false, true},
                    {true, true, true}};
drawGrid(grid); //   ▸

+
    +
+ + +
```

---

Für ein eindimensionales Array erwarten wir die selbe Ausgabe:

```
boolean[] grid = {true, false, false, false, false, true,
                  true, true, true};
drawGrid(grid, 3);
```

**c**) Update Rules ★ 🟦 🟢

In dieser Aufgabe wollen wir eine Methode schreiben, die eine Iteration bzw. einen Schritt von Conway's Game of Life umsetzt. Schreiben Sie hierfür eine Funktion

- `boolean[][] stepGOL(boolean[][] grid)` *oder*

- `boolean[] stepGOL(boolean[] grid, int width)`

entsprechend Ihrer Wahl in Teilaufgabe 2b. Achten Sie darauf, dass ein neues Array mit den aktualisierten Werten erzeugt wird und das übergebene Array *nicht* verändert wird.

Die Update-Regeln in jeder Generation/jedem Schritt lauten für die Zellen wie folgt:

1. Jede „live“ Zelle mit weniger als zwei „live“ Nachbarn stirbt an Einsamkeit.
2. Jede „live“ Zelle mit zwei oder drei „live“ Nachbarn bleibt im nächsten Schritt am Leben.
3. Jede „live“ Zelle mit mehr als drei „live“ Nachbarn stirbt an Überbevölkerung.
4. Jede „dead“ Zelle mit genau drei „live“ Nachbarn wird im nächsten Schritt lebendig.

**Beispiel:**
Für ein zwei-dimensionales Array gilt damit:

```
boolean[][] grid = {{true, true, true}, {false, true, true},
                    {true, true, false}};
stepGOL(grid); //  ▸
    {{true, false, true}, {false, false, false}, {true, true, true}}
```

Für ein ein-dimensionales Array gilt:

```
boolean[] grid = {true, true, true, false, true, true,
                  true, true, false};
stepGOL(grid, 3); //  ▸
    {true, false, true, false, false, false, true, true, true}
```

**d**) Initial State

Schreiben Sie nun eine weitere main-Methode (in einer neuen Datei und Klasse um Konflikte mit der existierenden zu vermeiden) die nicht nur die Feld-Höhe und -Breite, sondern auch eine beliebig lange Liste an Integer bekommen kann. Sie können davon ausgehen, dass immer eine gerade Anzahl an Integer übergeben wird. Die Integer-Paare repräsentieren die *x* und *y* Koordinaten der Zellen, die initial am Leben sind.

Das bedeutet: Basierend auf diesen initialen Werten müssen sie die das initiale Grid generieren.

**Beispiel:**

```
java Aufgabe2 5 5 0 0 1 1 2 2 4 4 0 4 3 4  ▸
+
  +
    +

+      ++
```

**e**) The Great Loop

Vereinen Sie jetzt die Ergebnisse der letzten Teilaufgaben. Eine Schleife soll das Spielfeld updaten und jede neue Version Ausgeben. Falls das Spiel einen Haltezustand („Fixpunkt“) erreicht, sich also in einem Schritt nicht mehr verändert, dann soll die Schleife beendet werden. Damit sie die Tests

(die selben für Teilaufgabe 2d) verwenden können, muss ihr Programm die Anzahl der benötigten Iterationen als letzte Ausgabe auf der Kommandozeile (in einer eigenen Zeile) erzeugen.

## **Aufgabe 3:** TicTacToe

Ein Klassiker unter den Programmieraufgaben für Anfänger: TicTacToe.
Wir haben für Sie eine Funktion `static int readInput()` vorbereitet, die den Nutzer nach einer Zahl zwischen 1 und 9 fragt und somit zur Erfassung der Spielerzüge dient. Eine Zahl repräsentiert hierbei die folgenden Zellen:

```
1 | 2 | 3
---------
4 | 5 | 6
---------
7 | 8 | 9
```

Die Nutzereingabe wird dabei so lange wiederholt, bis ein gültiger Wert (also eine Zahl zwischen 1 und 9) eingegeben wird. Beachten Sie jedoch, dass die Funktion nicht überprüft, ob eine gewählte Zelle bereits belegt ist.

**a**) Draw 

Schreiben Sie eine `draw` Methode, die das Spielfeld in der Konsole ausgibt. Als Parameter soll ein ein-dimensionales Ganzzahl-Array übergeben werden, welches die Zellen repräsentiert. Repräsentieren Sie ein leeres Feld mit dem Wert 0, Spieler 1 mit dem Wert 1 und Spieler 2 mit dem Wert 2.

**b**) Victory 

Schreiben Sie eine `checkVictory` Methode, welche überprüft, ob einer der beiden Spieler gewonnen. Als parameter soll ein ein-dimensionales Ganzzahl-Array übergeben werden, welches die Zellen repräsentiert. Zusätzlich soll ein int zurückgegeben werden, welcher den Spieler repräsentiert, dessen Sieg überprüft werden soll. Zurückgegeben werden soll ein Boolescher Wert.

**c**) The Game Loop 

Nutzten sie die bereitgestellte Funktion `static int readInput()` in Kombination mit den von ihnen implementieren Methoden, um den Game-Loop zu vervollständigen. Stellen Sie sicher, dass die Game-Loop Methode keine Parameter erwartet. Nach des Spiels sollte das Ergebnis ausgegeben werden:

```
Winner: Player 1!
```

Falls weder Spieler 1 noch Spieler 2 gewonnen haben, sollten sie dies auch ausgeben:

```
Tie!
```

**Achtung:** Es sollen zwei menschliche Spieler abwechselnd nach Ihren Zügen gefragt werden. Die Implementation einer KI ist nicht erforderlich.

**d**) Artificial Intelligence 

Ohne eine Implementation zu erstellen, beschreiben Sie, wie Sie eine künstliche Intelligenz für das Spiel TicTacToe implementieren könnten. Geben Sie Ihre Antwort hierzu als Textdatei (oder PDF ihrer Handschriftlichen Notizen) im Hauptverzeichnis des Repositories ab.