# 1 Quelltextvorlage

Nehmen Sie als Quelltextvorlage für die folgenden Aufgaben die Übung *mehrdimensionale Felder*, in welcher Sie bereits die Klasse `MatrixTools` erstellt haben. Sollten Sie diese Übung nicht absolviert haben nutzen Sie bitte die zur Verfügung gestellte Musterlösung als Vorlage für diese Übung.

# 2 Erstellen der Projektstruktur

Erweitern Sie die Übung um folgende Packages und Klassen:

- Package `exception`
- Klasse `exception.MatrixNotInitializedException`
- Klasse `exception.InvalidMatrixException`
- Klasse `exception.MatricesNotMultipliableException`
- Klasse `exception.MatrixSpurNotAvailableException`
- Klasse `matrix.MatrixDimension`

Der Ordner `src` in Ihrem Eclipse-Projektverzeichnis sollte dann folgende Ordner- und Dateistruktur haben:

```
.
|-- exception
|   |-- InvalidMatrixException.java
|   |-- MatricesNotMultipliableException.java
|   |-- MatrixNotInitializedException.java
|   `-- MatrixSpurNotAvailableException.java
|-- main
|   `-- Main.java
|-- matrix
|   |-- MatrixDimension.java
|   `-- MatrixTools.java
```

Abbildung 1: Ordner- und Dateistruktur im src-Verzeichnis

# 3 Implementierung der Klasse `matrix.MatrixDimension` 4P.

Erstellen und vervollständigen Sie die Klasse `matrix.MatrixDimension` um die Attribute und Methoden, welche in den folgenden Unterabschnitten beschrieben sind!

### 3.1 Konstruktor `MatrixDimension()` 3P.

Der Konstruktor `public MatrixDimension(int[][] matrix)` soll den Inhalt der übergebenen Variable `matrix` auf eine mathematisch korrekte Matrix überprüfen. Stellt die Prüfung keine Fehler fest sollen die Dimensionen Breite (Anzahl der Spalten) und Höhe (Anzahl der Zeilen) der Matrix in den Variablen `width` und `height` gespeichert werden.

Die erste Dimension des `int[][]-Arrays` stellt die Anzahl der Zeilen und die zweite Dimension die Anzahl der Spalten dar. Beispiel: `int[][] matrix = new int[3][5]` → 3 Zeilen á 5 Spalten. **1P.**

Die Prüfung der math. Korrektheit soll im Einzelnen folgende mögliche Fehlerfälle erkennen und diese durch Auslösen/Werfen der zugeordneten `Exception` anzeigen:

1. `MatrixNotInitializedException`: Die Matrix selbst oder eine Zeile ist nicht initialisiert (`null`). **1P.**
2. `InvalidMatrixException`: Eine Zeile hat keine Spalten, die Matrix ist nicht rechteckig, d. h. nicht alle Zeilen enthalten die gleiche Anzahl Spalten, oder die Matrix besitzt keine Zeilen. **1P.**

Fügen Sie beim Auslösen einer `Exception` Details zur Ursache in den Text der angezeigten Fehlermeldung mit ein, z. B.

```
throw new MatrixNotInitializedException("{}Zeile "{} + i + "{} ist null"{});
```

### 3.2 Variablen und Getter-Methoden `getWidth()` und `getHeight()` 1P.

Die Klasse soll die Attribute `private int width` und `private int height` enthalten. Der lesende Zugriff auf die Variablen durch andere Klassen erfolgt über die öffentlichen Getter-Methoden `int getWidth()` und `int getHeight()`.

## 4 Implementierung eigener Ausnahme-Klassen 4P.

Implementieren Sie die bereits in Abschnitt **??** genannten Exception-Klassen:

- `exception.MatrixNotInitializedException` **1P.**
- `exception.InvalidMatrixException` **1P.**
- `exception.MatricesNotMultipliableException` **1P.**
- `exception.MatrixSpurNotAvailableException` **1P.**

Leiten Sie diese alle von der Oberklasse `Exception` ab. Implementieren Sie in jeder Klasse einen (zusätzlichen) Konstruktor, welcher einen Parameter vom Typ `String` deklariert und die Übergabe einer benutzerdefinierten Fehlermeldung ermöglicht.

Listing **??** zeigt die prinzipielle Struktur der zu implementierenden Ausnahmeklassen am Beispiel.

Listing 1: Struktur eigener Ausnahmeklassen

```
1  class IchMagDichNichtException extends Exception
2  {
3      private static final String errMsg = "Ich mag dich nicht";
4
5      public IchMagDichNichtException()
6      {
7          super(errMsg);
8      }
9
10     public IchMagDichNichtException(String reason)
11     {
12         super(errMsg + ", weil: " + reason);
13     }
14 }
```

## 5 Anpassung der Methode `MatrixTools.matrixMul()` 2P.

Erweitern Sie die Methode `matrixMul()` der Klasse `MatrixTools` so, dass vor Beginn der Multiplikation die Dimension beider Matrizen mit Hilfe der Klasse `MatrixDimension` ermittelt wird. Die dabei möglicherweise ausgelösten Ausnahmen sollen innerhalb der Methode nicht behandelt (abgefangen) werden, sondern weitergeleitet werden.

Prüfen Sie im nächsten Schritt die Dimensionen beider Matrizen auf Kompatibilität und lösen Sie im Fehlerfall eine `MatricesNotMultipliableException` aus.

## 6 Anpassung der Methode `MatrixTools.matrixSpur()` 2P.

Passen Sie die Methode `matrixSpur()` gem. Listing **??** an.

Listing 2: Methode `matrixSpur`

```
1  public static int matrixSpur(int[][] matrix) throws
       ➥InvalidMatrixException, MatrixNotInitializedException,
       ➥MatrixSpurNotAvailableException
```

**Parameter:** zweidimensionales *Array* (Matrix)

**Rückgabe:** berechnete Spur der Parameter-Matrix

**Beschreibung:** Die Methode soll im ersten Schritt, vor Berechnung der Spur, die Dimension der übergebenen Matrix mit Hilfe der Klasse `MatrixDimension` ermitteln.

Im nächsten Schritt soll die Dimension geprüft werden und im Fall einer nicht quadratischen Matrix eine `MatrixSpurNotAvailableException` mittels `throw` ausgelöst werden.

# 7 Zusatzaufgabe Einlesen einer Matrix aus Textdatei 4P.

Erweitern Sie die Klasse `MatrixTools` um die Methode `getMatrixFromFile()` gem. Listing ??.

Listing 3: Methode `getMatrixFromFile`

```
1 public static int[][] getMatrixFromFile(String filename) throws
    ➥IOException{
2 }
```

**Parameter:** Textdatei, welche eine Matrix beinhaltet

**Beschreibung:** Die Methode soll eine `int[][]`-Matrix aus einer einfachen Textdatei einlesen. Die Spaltenwerte sind jeweils durch Leerzeichen, die einzelnen Zeilen durch einen Zeilenumbruch (Newline) voneinander getrennt.

**Empfehlung:** Für das Parsen von Zeilen und Spaltenwerten eignet sich beispielsweise die Klasse `java.util.Scanner`. Um eine unbekannte Anzahl von Werten während des Einlesevorgangs zu speichern empfiehlt sich die Nutzung von Container-Klassen, wie z. B. `java.util.ArrayList`, da diese im Gegensatz zu Arrays nicht mit fester Dimensionen instanziiert werden müssen.

Für das Testen Ihrer Methode können Sie sich eine Textdatei im Projektverzeichnis von Eclipse anlegen. (Achtung nicht im `src`-Verzeichnis). Es genügt dann die Verwendung des Dateinamens ohne Verzeichnis-/Pfadangabe damit diese beim Programmstart im Eclipse von den Klassen der Java-API gefunden wird.

Für das korrekte Einlesen von rechteckigen Matrizen erhalten Sie **2P.** Für das richtige Einlesen von „Flattermatrizen“ erhalten Sie zusätzlich **2P.** Ein Beispiel für eine „Flattermatrix“ sei wie folgt gegeben:

$$\mathbf{A} = \begin{pmatrix} 1 & 5 & 9 & 12 & \\ 3 & 6 & 2 & & \\ 7 & 4 & & & \\ 2 & 8 & 1 & 11 & 3 \end{pmatrix}$$

# 8 Bewertung

| Aufgabe | Vorraus-setzung | Punkte |
|---|---|---|
| Programm compilierbar | X | - |
| Klassen, Packages, Methoden, Signaturen richtig benannt/umgesetzt | X | - |
| Klasse `matrix.MatrixDimension` | | 4P. |
| Klasse `exception.MatrixNotInitializedException` | | 1P. |
| Klasse `exception.InvalidMatrixException` | | 1P. |
| Klasse `exception.MatricesNotMultipliableException` | | 1P. |
| Klasse `exception.MatrixSpurNotAvailableException` | | 1P. |
| Anpassung Methode `MatrixTools.matrixMul()` | | 2P. |
| Anpassung Methode `MatrixTools.matrixSpur()` | | 2P. |
| Zusatzaufgabe `getMatrixFromFile()` | | 4P. |
| **Gesamtpunkte 100%:** | | 12P. |
| **maximale Gesamtpunkte 133%:** | | 16P. |