P1 - Serie 05

Lukas Batschelet (16-499-733)

Implementationsaufgaben

(01) Vier Gewinnt

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):

```
1
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!
```

2. 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):
   1. `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)`.
   2. `isBoardFull`: gibt genau dann true zurück, wenn alle Felder durch einen Stein besetzt sind.
   3. `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.

```
public int insertToken(int column, Token tok) {
         // check if the column is valid
         if (column > VierGewinnt.COLS || column < 0) {
                        System.out.println("The entered column number does not exist");
                        System.exit(1);
         }
         // check if the chosen column is full
         if (board[column][VierGewinnt.ROWS - 1] != Token.empty) {
                        System.out.println("This column is already full");
                        System.exit(1);
         }
         // search first free row in column and place tok
         int row = 0;
         while (row < VierGewinnt.ROWS) {
                        if (board[column][row] == Token.empty) {
                                  break; // empty place found -> leave while loop
                        }
                        row++;
         }
         board[column][row] = tok;
         return row;
}
```

```
private boolean isBoardFull() {
        for(int col = 0; col < VierGewinnt.COLS; col++) {
                if(!isColumnFull(col))
                        return false;
        }
        return true;
}

private boolean isColumnFull(int column) {
        return this.board[column][ROWS - 1] != Token.empty;
}
```

```java
public boolean checkVierGewinnt(int col, int row) {
        Token tokToCheck = board[col][row];
        // check straight down
        if (row >= 3) {
                if (checkDown(col, row, tokToCheck)) {
                        return true;
                }
        }
        // check horizontal
        if (checkHorizontal(col, row, tokToCheck)) {
                return true;
        }
        // check diagonal left down to right up
        if (checkDiagonalLeftRight(col, row, tokToCheck)) {
                return true;
        }
        // check diagonal right down to left up
        if (checkDiagonalRightLeft(col, row, tokToCheck)) {
                return true;
        }
        // if none of the checks returned true, return false
        return false;
}
```

Rechnen mit Matrizen

(02) Rechnen mit Matrizen

1. 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.
2. Schreiben Sie in der Klasse `MatrixOperations` eine statische Methode `transpose` [1], 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.

3. Schreiben Sie in der Klasse `MatrixOperations` eine statische Methode `product` [2], welche eine $n \times m$ 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:

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

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.

Beispielechung:

$$Input: \ A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}, B = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} Output: C = \begin{pmatrix} 9 & 12 & 15 \\ 19 & 26 & 33 \\ 29 & 40 & 51 \end{pmatrix}$$

4. 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.

```java
public static int[][] readMatrix(String fileName) throws FileNotFoundException {

        ArrayList<String[]> lines = new ArrayList<String[]>();
        Scanner fileReader = new Scanner(new File(fileName));
        while (fileReader.hasNext()){
                String line = fileReader.nextLine();
                lines.add(line.split(" "));
        }

        int[][] matrix = new int[lines.size()][lines.get(0).length];

        for(int i = 0; i < matrix.length; i++) {
                for(int j = 0; j < matrix[0].length; j++) {
                        matrix[i][j] = Integer.parseInt(lines.get(i)[j]);
                }
```

```
        }
        return matrix;
}
```

```
public static int[][] transpose(int[][] matrix){
        if(matrix.length != matrix[0].length) {
                System.out.println("Die gegebene Matrix ist nicht Quadratisch");
                return null;
        }
        int[][] transpose = new int[matrix.length][matrix.length];
        for (int row = 0; row < matrix.length; row++) {
                for (int col = 0; col < matrix[0].length; col++) {
                        transpose[col][row] = matrix[row][col];
                }
        }
        return transpose;
}
```

```
public static int[][] product (int[][] a, int[][] b) {
                if (a[0].length != b.length) {
                        System.out.println("Dimensionen passen nicht!");
                        return null;
                }
                int[][] c = new int [a.length][b[0].length];
                int n = a[0].length;
                for (int i = 0; i < c.length; i++) {
                        for (int k = 0; k < c[0].length; k++) {
                                int sum = 0;
                                for (int j = 0; j < n; j++) {
                                        sum += a[i][j] * b[j][k];
                                }
                                c[i][k] = sum;
                        }
                }
                return c;
        }
}
```

---

1. https://de.wikipedia.org/wiki/Transponierte_Matrix ↩
2. https://de.wikipedia.org/wiki/Matrizenmultiplikation ↩