Quellcode Serie 05 - Programmieren 1

Jara Zihlmann(20-117-032)
Vithusan Ramalingam (21-105-515)
Jan Ellenberger (21-103-643)

Klasse VierGewinnt:

```
/*
Jara Zihlmann(20-117-032)
Vithusan Ramalingam (21-105-515)
Jan Ellenberger (21-103-643)
*/
package kap5;

/* ********************************************************************** *
\
*                Programmierung 1 HS 2020 - Serie 5-
1                          *
\* ********************************************************************** *
/

import java.util.Arrays;
import java.util.Scanner;

import javax.swing.text.html.HTMLDocument.RunElement;


public class VierGewinnt
{

    public static final int COLS = 7;
    public static final int ROWS = 6;

    private static Token[][] board = new Token[ COLS ][ ROWS ]; // 7 columns w
ith 6 fields each
    private IPlayer[] players = new IPlayer[ 2 ]; // two players

    /** initialize board and players and start the game */
    public void play()
    {
        // initialize the board
        for ( Token[] column : this.board ) {
            Arrays.fill( column, Token.empty );
        }

        /* initialize players */
        players[ 0 ] = new HumanPlayer();
        System.out.print( "Play against a human opponent? (y / n) " );
```

```
        String opponent = new Scanner( System.in ).nextLine().toLowerCase();
        while ( ( 1 != opponent.length() ) || ( -
1 == ( "yn".indexOf ( opponent ) ) ) ) {
            System.out.print( "Can't understand your answer. Play against a hu
man opponent? (y / n) " );
            opponent = new Scanner( System.in ).nextLine().toLowerCase();
        }
        if ( opponent.equals( "y" ) ) {
            players[ 1 ] = new HumanPlayer();
        } else {
            players[ 1 ] = new ComputerPlayer();
        }
        players[ 0 ].setToken( Token.player1 );
        players[ 1 ].setToken( Token.player2 );

        /* play... */
        boolean solved = false;
        int currentPlayer = ( new java.util.Random() ).nextInt( 2 );  //choose
 randomly who begins
        System.out.println( "current player: " + currentPlayer );
        int insertCol, insertRow; // starting from 0
        while ( !solved && !this.isBoardFull() ) {
            // get player's next "move"
            // note that we pass only a copy of the board as an argument,
            // otherwise the player would be able to manipulate the board and
cheat!
            insertCol = players[ currentPlayer ].getNextColumn( getCopyOfBoard
() );
            // insert the token and get the row where it landed
            insertRow = this.insertToken( insertCol, players[ currentPlayer ].
getToken() );
            // check if the game is over
            solved = this.checkVierGewinnt( insertCol, insertRow );
            //switch to other player
            if ( !solved )
                currentPlayer = ( currentPlayer + 1 ) % 2;
        }
        System.out.println( displayBoard( this.board ) );
        if ( solved )
            System.out.println( "Player " + players[ currentPlayer ].getToken(
) + " wins!" );
        else
            System.out.println( "Draw! Game over." );
    }


    /**
     * Inserts the token at the specified column (if possible)
     * @param column the column to insert the token
     * @param token the players token
```

```
     * @return the row where the token landed
     */
    private int insertToken( int column, Token tok )
    {
        //TODO: Your code goes here

        //falls die Reihe ausserhalb des Boards ist
        //bricht das Programm ab
        if(column >= board.length || column < 0
        || board[column][board[0].length - 1] != Token.empty) {
            System.exit(1);
        }

        //token in Reihe einfügen solange es im Feld ist
        int row = -1;
        for (int i = 0; i < board[column].length; i++) {
            if (board[column][i].equals(Token.empty)) {
                row = i;
                break;
            }
        }
        board[column][row] = tok;
        return  row;

    }


    /**
     * Checks if every position is occupied
     * @returns true, iff the board is full.
     */
    private boolean isBoardFull()
    {
        //TODO: Your code goes here

        //überprüfen ob noch eine Position frei ist, sonst
        //isBoardFull = true setzen
        for (int i = 0; i < COLS; i++) {
            if (board[i][ROWS - 1].equals(Token.empty)) {
               return false;
            }
        }
        return true;
    }


    private boolean checkFourInColumn( int col, int row ){

        Token tokenToCheck = this.board[col][row];
```

```
        int fourInColumn = 0;
        boolean win = false;

        //jedes mal prüfen dass nicht über den Rand gezählt wird
        //solange nicht eine der Gewinnchancen erfüllt ist
        for (int i = -3;
             i < 4 && fourInColumn < 4;
             i++) {
            //  x
            //  x nach oben und unten prüfen
            //  x
            fourInColumn = ((row + i) < board[col].length && (row + i) >= 0
            && tokenToCheck == this.board[col][row + i])
            ? ++fourInColumn: 0;
             }
        if (fourInColumn == 4){
            win = true;
        }
        else{
            win = false;
        }
        return win;
    }


    private boolean checkFourInRow( int col, int row ){

        Token tokenToCheck = this.board[col][row];

        int fourInRow = 0;
        boolean win = false;
        //jedes mal prüfen dass nicht über den Rand gezählt wird
        //solange nicht eine der Gewinnchancen erfüllt ist
        for (int i = -3;
             i < 4 && fourInRow < 4;
             i++) {
        //  x x x   nach links und rechts testen
        fourInRow = ((col + i) < board.length && (col + i) >= 0
        && tokenToCheck == this.board[col + i][row])
        ? ++fourInRow: 0;
             }
        if (fourInRow == 4){

            win = true;
        }
        else{
            win = false;
        }

        return win;
```

```
    }



    private boolean checkDiagonalLeftRight( int col, int row ){

        Token tokenToCheck = this.board[col][row];

        int diagonalLeftRight = 0;
        boolean win = false;
        //jedes mal prüfen dass nicht über den Rand gezählt wird
        //solange nicht eine der Gewinnchancen erfüllt ist
        for (int i = -3;
             i < 4 && diagonalLeftRight < 4;
             i++) {
            //          x
            //      x   diagonal von links nach rechts oben prüfen
            //  x
            diagonalLeftRight = ((col + i) >= 0 && (row + i) >= 0
                    && (col + i) < board.length && (row + i) < board[col].leng
th
                    && tokenToCheck == this.board[col + i][row + i])
                    ? ++diagonalLeftRight: 0;
             }
        if (diagonalLeftRight == 4){
            win = true;
        }
        else{
            win = false;
        }
        return win;
    }


    private boolean checkDiagonalRightLeft( int col, int row ){

        Token tokenToCheck = this.board[col][row];

        int diagonalRightLeft = 0;
        boolean win = false;
        //jedes mal prüfen dass nicht über den Rand gezählt wird
        //solange nicht eine der Gewinnchancen erfüllt ist
        for (int i = -3;
             i < 4 && diagonalRightLeft < 4;
             i++) {
            //  x
            //      x   diagonal von recht nach links oben prüfen
            //          x
            diagonalRightLeft = ((col - i) >= 0 && (row + i) >= 0
```

```
                    && (col - i) < board.length && (row + i) < board[col].leng
th
                    && tokenToCheck == this.board[col - i][row + i])
                    ? ++diagonalRightLeft: 0;
             }
        if (diagonalRightLeft == 4){
            win = true;
        }
        else{
            win = false;
        }
        return win;
    }



    /**
     * Checks for at least four equal tokens in a row in
     * either direction, starting from the given position.
     */
    private boolean checkVierGewinnt( int col, int row )
    {
        //TODO: Your code goes here

        //falls eine der 4 Hilsmethoden true ist
        //return true um solved auf true zu setzen
        //und das Spiel als gewonnen anzugeben
        if (checkFourInColumn(col, row) == true
        || checkFourInRow(col, row) == true
        || checkDiagonalLeftRight(col, row) == true
        || checkDiagonalRightLeft(col, row) == true )
        {
            return true;
        }

        else
        {
            return false;
        }

    }


    /** Returns a (deep) copy of the board array */
    private Token[][] getCopyOfBoard()
    {
        Token[][] copiedBoard = new Token[ COLS ][ ROWS ];
        for ( int i = 0; i < copiedBoard.length; i++ ) {
            for ( int j = 0; j < copiedBoard[ i ].length; j++ ) {
                copiedBoard[ i ][ j ] = this.board[ i ][ j ];
```

```
            }
        }
        return copiedBoard;
    }


    /** returns a graphical representation of the board */
    public static String displayBoard( Token[][] myBoard )
    {
        String rowDelimiter = "+";
        String rowNumbering = " ";
        for ( int col = 0; col < myBoard.length; col++ ) {
            rowDelimiter += "---+";
            rowNumbering += " " + ( col + 1 ) + "  ";
        }
        rowDelimiter += "\n";

        String rowStr;
        String presentation = rowDelimiter;
        for ( int row = myBoard[ 0 ].length - 1; row >= 0; row-- ) {
            rowStr = "| ";
            for ( int col = 0; col < myBoard.length; col++ ) {
                rowStr += myBoard[ col ][ row ].toString() + " | ";
            }
            presentation += rowStr + "\n" + rowDelimiter;
        }
        presentation += rowNumbering;
        return presentation;
    }



    /** main method, starts the program */
    public static void main( String args[] )
    {
        VierGewinnt game = new VierGewinnt();
        game.play();
    }
}
```

Klasse MatrixOperations:

```java
/*
Jara Zihlmann(20-117-032)
Vithusan Ramalingam (21-105-515)
Jan Ellenberger (21-103-643)
*/
package kap5;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.Scanner;



public class MatrixOperations {

    private final static String PATH_MATRICES = "src/kap5/res/";
    // throws FileNotFoundException falls kein file für die Matrix angelgt wur
de
    public static int[][] readMatrix(String fileName) throws FileNotFoundExcep
tion {
        //ArrayList erstellen  und
        //Lines der Matrizen in Arrayliste einfügen
        //um die grösse zu erhalten
        Scanner scan = new Scanner(new File(PATH_MATRICES + fileName));
        ArrayList<String> matrixLines = new ArrayList<>();
        int[][] matrix = null;
        while (scan.hasNextLine()) {
            matrixLines.add(scan.nextLine());
        }
        scan.close();

        for (int i = 0; i < matrixLines.size(); i++) {
            String[] elements = matrixLines.get(i).split(" ");

            //instanzieren und alle Zahlen dem Array hinzufügen
            if (matrix == null)
                matrix = new int[matrixLines.size()][elements.length];
            for (int j = 0; j < elements.length; j++) {
                matrix[i][j] = Integer.parseInt(elements[j]);
            }
        }
        //Array der Matrix zurückgeben
        return matrix;
    }
```

```java
// transposieren einer Matrix
// Matrix darf nicht leer sein
// Matrix muss quadratisch sein
public static int[][] transpose(int[][] tmatrix) {
    if(tmatrix == null || tmatrix.length < 1) {
        System.out.println("Die Matrix darf nicht leeer sein!");
        return tmatrix;

    }
    //überprüfen ob Matrix quadratisch ist
    else if (tmatrix[0].length != tmatrix.length){
        System.out.println("Diese Matrix kann nicht transponiert werden,"
        + "sie muss quadratisch sein");
        return null;
    }

    //matrix instanzieren mit neuer länge
    int[][] transponiert = new int[tmatrix.length][tmatrix[0].length];

    for (int i = 0; i < tmatrix.length; i++) {
        for (int j = 0; j < tmatrix[i].length; j++) {
            transponiert[j][i] = tmatrix[i][j];
        }
    }
    //transponierte Matrix zurückgeben
    return transponiert;
}

// Methode um Matrizen zu multiplizieren
public static int[][] product(int[][] matrixA, int[][] matrixB) {
    int rowsA = matrixA.length, columnA = matrixA[0].length;
    int rowsB = matrixB.length, columnB = matrixB[0].length;
    int[][] resultMatrix = new int[rowsA][columnB];

    if (columnA != rowsB) {
        System.out.println("Die beiden Matrizen sind nicht kompatibel");
        return null;
    }

    for (int i = 0; i < resultMatrix.length; i ++) {
        for (int j = 0; j < resultMatrix[i].length; j ++) {
            int sum = 0;

            for (int k = 0; k < columnA; k++) {
                sum += matrixA[i][k] * matrixB[k][j];
            }
            resultMatrix[i][j] = sum;
        }
```

```
        }
        return resultMatrix;
    }


    //Methode um Matrix in einen String umzuwandeln und zurückgeben
    public static String matrixToString(int[][] matrix) {
        String matrixString = "";
        if (matrix == null)
            return "";

        for (int[] line: matrix) {
            for (int entry: line) {
                matrixString += entry + " ";
            }
            matrixString += "\n";
        }
        return matrixString;
    }
}
```

Klasse MatrixTest:

```
/*
Jara Zihlmann(20-117-032)
Vithusan Ramalingam (21-105-515)
Jan Ellenberger (21-103-643)
*/
package kap5;

public class MatrixTest {

    public static String[] matrixDateien = {"matrix_a", "matrix_b",
    "matrix_c", "matrix_d", "matrix_e", "matrix_z",
    "matrix_product_a", "matrix_product_b"};

    public static void main(String[] args) {
        System.out.println("Gelesene und transponierte Matrizen:");
        //Test um gelesene Matrizen wenn möglich Transponiert ausgeben
        printTransponiertUndGeleseneMatrix();
        System.out.println("Test: Matrix Multipliziert");
        //Test multiplizierte Matrizen ausgeben
        printMultiplizierteMatrix();

    }

    //print der Multiplizierten Matrix
    public static void printMultiplizierteMatrix() {
        int[][] matrixA, matrixB, matrixProduct;

        //Matrizen multiplizieren und ausgeben
        try {
            matrixA = MatrixOperations.readMatrix("matrix_product_a");
            matrixB = MatrixOperations.readMatrix("matrix_product_b");
            matrixProduct = MatrixOperations.product(matrixA, matrixB);

            System.out.println(MatrixOperations.matrixToString(matrixA));
            System.out.println(MatrixOperations.matrixToString(matrixB));
            System.out.println(MatrixOperations.matrixToString(matrixProduct))
;
        }
        //sonst ausgabe des Fehlers
        catch (Exception e) {
            e.printStackTrace();
        }
    }

    //Matrizen testen
    //verschidene richtige und falsche Matrizen
```

```
    //auch throwsexeption testen mit matrix_z da diese nicht vorhanden ist
    public static void printTransponiertUndGeleseneMatrix() {
        int[][] matrix, transponierteMatrix;

        try {
            for (String fileName: matrixDateien) {
                matrix = MatrixOperations.readMatrix(fileName);
                transponierteMatrix = MatrixOperations.transpose(matrix);

                System.out.println(MatrixOperations.matrixToString(matrix));
                System.out.println(MatrixOperations.matrixToString(transponier
teMatrix));
            }
        }
        //allfällige Fehlerausgabe
        catch (Exception e) {
            System.out.println(e);
        }

    }
}
```

Die verschiedenen Matrizen:

Matrix_a: **1 3 5**
2 4 6

Matrix_b : **1 2 3 4**
6 7 8 9
1 2 3 4
6 7 8 9

Matrix_c : **2 7**
4 9

Matrix_d : (leer)

Matrix_e: **2 7 9**
7 8

Matrix_z : (existiert nicht)

Matrix_product_a : **1 2**
3 4
5 6

Matrix_product_b : **1 2 3**
4 5 6

Jara Zihlmann(20-117-032) Vithusan Ramalingam (21-105-515) Jan Ellenberger (21-103-643)