Assignment 09: JUnit

Due Date: 2. 6. 2016, 10:00

Name: _____ **Student number**: _____

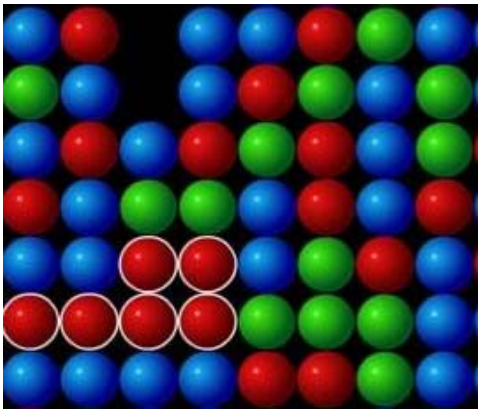
CS: ☐ G1 (Grimmer) ☐ G2 (Prähofer) ☐ G3 (Prähofer)

WIN: ☐ G1 (Khalil) ☐ G2 (Khalil) ☐ G3 (Hummel)

Task	Points	Hand-in			
UE 09	24	Java program, Output of a testrun			

Magic Marbles Game (24 Points)

Magic Marbles is a simple computer game with differently colored marbles that are horizontally and vertically aligned. A player may select connected (adjacent) marbles that will then disappear in the next run. The aim of the game is to clear the entire board, whereby as many marbles should be removed at once to be in order to reach a high score. If only single marbles are left over, this is the end of the game. Single leftover marbles further reduce the score in the end.



This exercise aims to develop a simple version of magic marbles and extensively test the game with JUnit. In the following exercise 10, we want to build a graphical interface for the game.

A) Implementation of the game (14 points)

Gameplay:

At the beginning, the player defines the size of the playing field, i.e., in your version of *Magic Marbles* playing fields with any board size should be supported. Subsequently, a game box is created with a random assignment of red, green, and blue marbles. Then, (i) the playing field is printed on the console, (ii) the current score is printed, and (iii) the user is asked for a valid move. This is repeated until no more valid moves can be made. The game is over when either the playing field has been completely emptied, or no valid move is possible any more (this is the case if only individual, not connected marbles of similar color are left). In the end, the achieved score is calculated and printed out.

Calculation of the Score:

The number of points is calculated as the sum of the previously achieved points plus the points of the removed marbles to the power of two. For example, if 5 adjacent marbles are removed at once, the score is increased by 25 points. Each individual marble remaining at the end of the game decreases the score by 10 points, i.e., in case 2 marbles remain, 20 points are subtracted.

Example Game:

```
Height: 5
Width: 4

    0  0  0  0
    1  2  3  4
01  b  g  g  r
02  b  g  r  b
03  g  g  g  g
04  r  r  r  r
05  g  r  r  r
********************
Current Score: 0
********************
Please select a non-empty field
Row: 5
Column: 4

    0  0  0  0
    1  2  3  4
01
02  b
03  b  g  g  r
04  g  g  r  b
05  g  g  g  g
********************
Current Score: 49
********************
Please select a non-empty field
Row: 5
Column: 3

    0  0  0  0
    1  2  3  4
01
02
03
04     b     r
05     b  r  b
********************
Current Score: 113
********************
Please select a non-empty field
Row: 5
Column: 2

    0  0  0  0
    1  2  3  4
01
02
03
04           r
05        r  b

********************
Final Score: 87
********************
Game over!
```

Implement the game. Work with Java assertions to make the program robust against specific conditions.

To facilitate this task, a class hierarchy is provided (see `UE09_Vorgabe.zip`), which includes the following packages and classes:

Package `magicmarbles.model` (implementation of the game):

- *Interface MM* ***`Field`***: declares the methods that must be implemented by the Magic Marbles game.
- *Class MM* ***`FieldImpl`***: The game is implemented in this class. In "UE09_Vorgabe", empty templates of methods of the interface `Field.java` are included.
- *Class MMException*: Exception class for exceptions related to selection of positions in the game.
- *Enumeration MM* ***`FieldState.java`***: gives possible values of a position on the play field. A position can be filled with a red, green, or blue marble or it can be empty.
- *Enumeration* ***`MMState.java`***: defines possible states of the game. The game can be either in progress ("running") or terminated ("end").

Package `magicmarbles.textui` (console-based interface):

- *Class* `Main.java`: implements a console-based application to interact with the game logic.

Notes:

- Note that automatic "slipping" of marbles into holes should be implemented. Marbles may slip in the vertical direction (i.e., downward). Further, in case empty columns occur between marbles, the marbles should be moved in the horizontal direction to the right. This means that marbles “move” to the right bottom corner (see gameplay).
- Note also that after each move it has to be checked whether other valid moves are possible, i.e., whether the game has reached the end or not.
- Finally, note that the upcoming exercise 10 builds on this exercise. With a clean preparation of this exercise you are already well prepared for the next assignment.

(B) JUnit tests (10 points)

Now, write a comprehensive JUnit test for the game. Write several tests, with each test to be performed as follows:

1. start each test, with a well-defined initial state.
2. perform a game move.
3. test the expected score, using assert statements of the JUnit framework: where are the marbles on the board? what's the score? is the end of the game reached?
4. repeat points 2 and 3 as arbitrary times (perhaps until the end of the game is reached).

It is important that all relevant cases have been tested with JUnit tests. Include also test cases of wrong, invalid use. Include for example the case of selecting an empty position. Here, too, the system must respond correctly. Therefore, test whether the exceptions are properly thrown.

The JUnit test should be implemented in the package

`magicmarbles.model.test` .