Übung 09: JUnit

Abgabetermin: 2. 6. 2016, 10:00

Name: _____ **Matrikelnummer**: _____

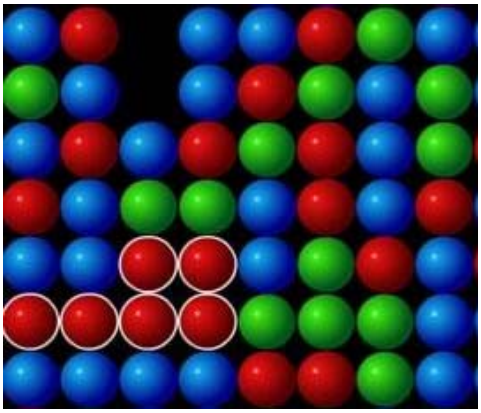
Informatik: ☐ G1 (Grimmer) ☐ G2 (Prähofer) ☐ G3 (Prähofer)

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

Aufgabe	Punkte	abzugeben elektronisch		korr.	Punkte
Übung 9	24	Java-Programm, Ausgabe eines Testlaufs			

MagicMarbles Spiel (24 Points)

Magic Marbles ist ein einfaches Computerspiel, bei dem ein Spieler horizontal und vertikal zusammenhängende gleichfarbige Murmeln auswählen kann und diese dann im nächsten Spielschritt verschwinden. Ziel des Spiels ist es, das gesamte Spielfeld zu leeren, wobei möglichst viele Murmeln auf einmal entfernt werden sollen damit man eine hohe Punktzahl erreicht. Übrig gebliebene einzelne Murmeln bedeuten am Spielende Punkteabzüge.



Ziel dieser Übung ist es, eine einfache Version von Magic Marbles zu entwickeln und mit JUnit ausführlich zu testen. In der folgenden Übung 10 wollen wir dann dazu eine graphische Oberfläche bauen.

A) Implementierung des Spiels (14 Punkte)

Spielablauf:

Zu Beginn legt der Spieler die Spielfeldgröße fest, d.h., Ihre Version von *Magic Marbles* soll Spielfelder mit beliebiger Spielfeldgröße unterstützen. In weiterer Folge wird ein entsprechendes Spielfeld mit einer zufälligen Belegung von roten, grünen und blauen Murmeln erzeugt. Ab diesem Zeitpunkt wird (i) das Spielfeld auf der Konsole ausgegeben, (ii) die aktuellen Punkte werden ausgegeben und (iii) es wird nach einem gültigen Zug gefragt. Dies wird wiederholt, bis keine weiteren gültigen Züge mehr getätigt werden können. Das Spiel ist beendet, wenn entweder das Spielfeld vollständig geleert wurde oder kein gültiger Zug mehr möglich ist (dies ist der Fall falls nur noch einzelne unzusammenhängende Murmeln übrig sind). Wurde das Ende erreicht, wird schließlich noch der erreichte Punktestand ausgegeben.

Punkteberechnung:

Die Anzahl der erreichten Punkte errechnet sich als Summe der bisher erreichten Punkte plus der Anzahl der auf einmal entfernten Murmeln zum Quadrat. Das heißt, werden in einem Zug 5 zusammenhängende Murmeln entfernt, so erhöht sich der Punktestand um 25 Punkte. Jede am Spielende übrig gebliebene einzelne Murmel erniedrigt den Punktestand um 10 Punkte, d.h., bleiben 2 Murmeln übrig, so werden 20 Punkte subtrahiert.

Exemplarischer Spielablauf:

```
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!
```

Implementieren Sie das Spiel. Arbeiten Sie dabei auch mit Java-Assertions, um das Programm gegen bestimmte Bedingungen abzusichern.

Um Ihnen diese Aufgabe zu erleichtern, ist ein Klassengerüst vorgegeben (siehe `UE09_Vorgabe.zip`), welches folgende Pakete sowie Klassen enthält:

Paket `magicmarbles.model` (Umsetzung des Spiels):

- *Interface MM`Field`*: Gibt die Methoden vor, die eine Implementierung von Magic Marbles anbieten muss.
- *Klasse MM`FieldImpl`*: In dieser Klasse soll das Spiel implementiert werden. In der Vorgabe sind hier die leeren Methodenrümpfe des Interfaces `Field.java` enthalten.
- *Klasse MMException*: Exception-Klasse für Exceptions bei der Auswahl von Positionen im Spiel.

- *Enumeration MM*`FieldState.java`: Gibt mögliche Werte vor, die ein Platz des Spielfeldes annehmen kann. Ein Platz kann entweder mit einer roten, grünen oder blauen Murmel besetzt sein oder kann leer sein.
- *Enumeration* `MMState.java`: Gibt mögliche Zustände vor, die das Spiel annehmen kann. Das Spiel kann entweder im Gange („running“) oder beendet („end“) sein.

Paket `magicmarbles.textui` (Konsolen-basierte Schnittstelle):

- *Klasse* `Main.java`: Implementiert eine Konsolen-basierte Anwendung, um mit dem Spiel zu interagieren.

Hinweise:

- Beachten Sie das automatische „Nachrutschen“ der Murmeln bei auftretenden Löchern zwischen den Murmeln. Hierbei rutschen die Murmeln in vertikaler Richtung nach (d.h., nach unten). Treten überdies leere Spalten zwischen den Murmeln auf, so werden die Murmeln in horizontaler Richtung verschoben (d.h., nach rechts). Das heißt, die Murmeln bewegen sich bei auftretenden Löchern nach rechts unten (siehe Spielablauf).
- Beachten Sie weiters, dass nach jedem ausgeführten Zug überprüft werden muss, ob noch weitere gültige Züge möglich sind, d.h., ob das Ende des Spiels erreicht wurde.
- Beachten Sie schließlich, dass die kommenden Übungen 10 auf dieser Übung aufbauen werden – d.h., mit einer sauberen Ausarbeitung dieser Übung schaffen Sie sich eine gute Ausgangsbasis für die beiden folgenden Übungen!

B) JUnit-Tests (10 Punkte)

Schreiben Sie nun einen umfassenden JUnit-Test für die Spieleapplikation. Schreiben Sie mehrere Tests, wobei jeder Test folgend durchgeführt wird:

1. Beginnen Sie jeden Test mit einem wohl definierten Anfangszustand.
2. Führen Sie einen Zug durch.
3. Testen Sie mit assert-Anweisungen des Junit-Frameworks den erwarteten Spielstand: Wo sind welche Murmeln am Brett? Wie ist der Punktestand? Ist das Spielende erreicht?
4. Wiederholen Sie Punkte 2 und 3 beliebig oft (vielleicht bis Spielende erreicht).

Wichtig beim JUnit-Test ist, dass möglichst alle relevanten Fälle getestet worden sind. Dazu gehören auch Fälle, die eigentlich eine ungültige Verwendung darstellen, z.B., die Selektion einer leeren Position. Auch hier muss das System richtig reagieren. Testen Sie daher, ob die Exceptions richtig geworfen werden.

Der JUnit-Test soll im Package

`magicmarbles.model.test`

implementiert werden.