P. Thiemann, L. Fennell Sommersemester 2017

**Programmieren in Java**

http://proglang.informatik.uni-freiburg.de/teaching/java/2017/

# block-world

*Eine Welt der fallenden Blöcke*

Woche 08 Aufgabe 3/3

Herausgabe: 2017-06-19
Abgabe: 2017-06-30

**Achtung:** beachten Sie unbedingt die allgemeinen Hinweise zur Abgabe auf der Homepage.

Project `block-world`
Package `blockworld`
Klassen

| Block |
| --- |
| `public Block(int x, int y, int velocity, char shape)` |

| BlockWorld |
| --- |
| `public BlockWorld(int width, int height, List<Block> blocks, char empty)`<br>`public int getWidth();`<br>`public int getHeight();`<br>`public char[][] observe()`<br>`public void step()`<br>`public boolean isDead()` |

**Achtung:** Dies ist im Prinzip eine Wiederholung/Verlängerung der Aufgabe `w06/block-world`. Wir wiederholen die Aufgabe, da die Probleme mit Jenkins bei dieser Aufgabe eine wichtige „Lektion" verdorben haben, die wir jetzt nachholen wollen. Die Jenkins-Tests dieser neuaufgelegten Aufgabe sind in zwei Teile aufgeteilt. Für jeden Teil gibt es 2P.

1. Die reparierten Testfälle von `w06`. Die Jenkins Fehlermeldungen sollten nun verständlich und nachvollziehbar sein. Falls Sie in `w06` funktionierenden Code hatten, den Jenkins unverständlicherweise abgewiesen hat, sollten dieser hier jetzt einfach funktionieren.

2. Einige Testfälle, die problematische Situation von `w06` widerspiegeln: Jenkins hat nämlich für die verschiedenen Tests nicht jedes Mal neue `Block`-Listen erstellt, sondern alte wiederverwendet. Dies ist problematisch, wenn die `BlockWorld` den Inhalt der Eingabeliste verändert.

   Das folgende Beispiel illustriert die Situation:

```
1  List<Block> blocks = Arrays.asList(new Block(0, 0, 1, 'x')),
2                                     new Block(1, 0, 1, 'o'));
3  BlockWorld w1 = new BlockWorld(2, 3, blocks, '.');
4  w1.step(); w1.step(); w1.step();
```

```
5       ...
6       BlockWorld w2 = new BlockWorld(2, 3, blocks, '.');
7       assertEquals('x', w2.observe()[0][0])
```

Das letzte `assertEquals` schlägt fehl, wenn die Blöcke der Liste `blocks` schon von den `step`s von `w1` bewegt wurden.

Sichern Sie ihren Code also gegen solche Aufrufe mit gemeinsamen Eingabelisten ab (siehe dazu auch die Vorlesung vom 19.6.). Dann wird auch der zweite Satz Testfälle durchlaufen.

**Text der alten Aufgabe:** In dieser Aufgabe werden Welten mit fallenden Blöcken, kurz „Blockwelten", im Computer geschaffen. Blockwelten sind zweidimensional, endlich und diskret.

Eine Blockwelt der Höhe $h$ und Breite $w$ besteht aus einem Gitter von $w \times h$ Positionen. Die obere, linke Ecke des Gitters hat die Koordinaten $(0, 0)$, die untere, rechte Ecke die Koordinaten $(w - 1, h - 1)$. Ferner enthält eine Blockwelt natürlich Blöcke. Jeder Block nimmt eine Position auf dem Gitter ein. Jede Position kann von mehreren Blöcken besetzt werden [1]

Blockwelten ändern sich in diskreten Schritten. Blöcke haben eine Geschwindigkeit, mit der sie in der Welt nach unten fallen. Blockgeschwindigkeiten sind positive (d.h. $\geq 0$), ganzahlige Werte und geben die Anzahl der Felder an, die ein Block pro Schritt nach unten fällt. Ist ein Block unten auf dem Boden angelangt, bleibt der dort. Das bedeutet auch, dass Blockwelten nach endlichen vielen Schritten „tot" sind; irgendwann bewegen sich in ihnen keine Blöcke mehr.

Implementieren Sie die Klassen `BlockWorld` und `Block`. Analog zur obigen Beschreibung wird ein `BlockWorld` Objekt durch Angabe von Breite und Höhe und einer Liste von Blöcken initialisiert. Zusätzlich hat der Konstruktor noch eine `char`-Argument, das angibt, wie leere Positionen in der Welt beobachtet werden können (s.u.). Blöcke haben (mindestens) eine Position und eine Geschwindigkeit und eine Form als `char`. Positionen und Geschwindigkeiten müssen positiv sein, ansonsten wirft der Konstruktor von `Block` eine `IllegalArgumentException`. Durch Aufrufen der `step` Methode wird ein Schritt in einem `BlockWorld` Objekt ausgeführt. Die Methode `isDead` gibt `true` zurück genau dann wenn die `BlockWorld` kein Block mehr bewegt werden kann. Der Konstruktor von `BlockWorld` wirft eine `IllegalArgumentException`, wenn die ihm übergebenen Blöcke nicht in die Welt passen.

Die Methode `observe` gibt ein `char` Array zurück, der den aktuellen Zustand der Welt darstellt. Die Positionen des Arrays entsprechen dabei den Positionen der Welt. Ist eine Koordinate in der Welt leer, enthält das Array das Zeichen `empty`. Andernfalls enthält es das größte unter den Zeichen der Blöcke, die sich auf der Position befinden. (Das heißt, dass bei einer Welt mit den Blöcken `new Block(0, 0, 1, 'a')` und `new Block(0, 0, 1, 'b')` auf Position $(0, 0)$ das Zeichen `'b'` beobachtet wird)

Im Skelett zu dieser Aufgabe finden Sie die Klasse `BlockWorldExampleTests`, die einige JUnit-Beispieltestfälle enthält.

```
package blockworld;

import org.junit.Test;
```

[1] Die führenden Blockwelt-Wissenschaftler sind sich nicht einig, wie das in einer 2D Welt überhaupt möglich sein kann ... aber die experimentellen Beobachtungen lassen keinen anderen Schluss zu.

```
import java.util.Arrays;
import java.util.Collections;
import java.util.List;

import static org.junit.Assert.*;

public class BlockWorldExampleTest {

    @Test
    public void testMinimalWorld() {
        List<Block> bs = Collections.singletonList(new Block (0, 0, 1, 'x'));
        BlockWorld w = new BlockWorld(1, 2, bs, '.');
        assertFalse(w.isDead());
        assertArrayEquals(new char[][]{ new char[]{ 'x', '.'}}, w.observe());
        w.step();
        assertTrue(w.isDead());
        assertArrayEquals(new char[][]{ new char[]{ '.', 'x'}}, w.observe());
    }

    @Test(expected = IllegalArgumentException.class)
    public void testFailingBlockConstruction() {
        new Block(-1, 0, 1, 'x');
    }

}
```

**Hinweise:**

- Die oben beschriebenen Methoden reichen Ihnen zur Lösung der Aufgabe nicht aus. Definieren Sie weitere so wie es Ihnen sinnvoll erscheint. In der Vorlesung werden dann „best practices" zum Klassendesign besprochen.
- Die Funktion `java.util.Arrays.asList` erlaubt das Umwandeln eines Arrays als Liste. Außerdem ermöglicht die Funktion eine kompakte Schreibweise für Listen, z.B. für Testfälle.

```
1 List<Integer> somePrimes = Arrays.asList(2, 3, 5, 7, 11, 17);
```

- Im Skelett des Projekts befindet sich auch eine Klasse `Main`. Diese enthält eine `main`-Methode, die es erlaubt, Blockwelten in einem Fenster grafisch darzustellen und ablaufen zu sehen. Vielleicht inspiriert Sie das ja auch, sich etwas mit GUI-Programmierung in Java zu beschäftigen.

  `https://docs.oracle.com/javase/tutorial/uiswing/TOC.html`

  (In dieser Vorlesung werden wir aber nicht mehr dazu kommen)