Game Of Life in Scala

Jens Schauder
LINEAS Informationstechnik GmbH
Braunschweig

Schlüsselworte
Scala, Kata, TDD, ScalaTest, Game of Life

Einleitung
Eine Einführung in eine ihnen noch weitgehend unbekannte Programmiersprache, ein dazu passendes Unit Test Framework und TDD in unter einer Stunde? Das geht wohl kaum. Aber demonstrieren kann man es mit einer Code Kata. Wenn Sie sich also für Scala, ScalaTest oder TDD interessieren, sind sie in diesem Vortrag richtig.

Übersicht

Ich werde ein Code Kata in Scala mit dem Testframework ScalaTest durchführen. Und dabei so viel Wissenswertes darüber vermitteln, wie nur möglich.

Ein Code Kata ist eine Programmierübung, bei der eine einfache Programmieraufgabe immer wieder gelöst wird, um den Lösungsweg zu verinnerlichen und zu üben. Die Übung kann sich dabei sowohl auf die Handhabung der IDE, die Programmiersprache oder das Lösungsdesign beziehen.

Game of Life
Conways Game of Life ist ein zellulärer Automat, der ein unendliches quadratisches Gitter mit Zellen auf jedem Feld repräsentiert. Zellen können leben oder tot sein. Dieser Zustand ändert sich gemäß den folgenden Regeln in jeder Generation:

- Eine tote Zelle mit genau drei Nachbarn lebt in der nächsten Runde
- Lebende Zellen mit weniger als zwei lebenden Nachbarn sterben in der nächsten Runde.
- Eine Lebende Zelle mit zwei oder drei lebenden Nachbarn bleibt am Leben.
- Eine Lebende Zelle mit mehr als drei Nachbarn ist in der folgenden Generation tot.

Die Lösung
… wird in etwa wie folgt aussehen:

```
package de.schauderhaft.kata.gol

import org.scalatest._
import org.scalatest.matchers._
import org.junit.runner._
import org.scalatest.junit._

/**
```

```
 * <ul>
 * <li>eine Zelle mit weniger als zwei Nachbarn stirbt.</li>
 * <li>eine Zelle mit zwei oder drei Nachbarn bleibt am Leben</li>
 * <li>eine tote Zelle mit drei Nachbarn wird geboren</li>
 * <li>eine Zelle mit mehr als drei Nachbarn stirbt</li>
 * </ul>
 *
 */

@RunWith(classOf[JUnitRunner])
class GameOfLife extends FunSuite with ShouldMatchers {

    val topLeft = (-1, 1)
    val topCenter = (0, 1)
    val topRight = (1, 1)

    val left = (-1, 0)
    val center = (0, 0)
    val right = (1, 0)

    val bottomLeft = (-1, -1)
    val bottomCenter = (0, -1)
    val bottomRight = (1, -1)

    test("an empty Universe stays empty") {
        new Universe().next.cells should be('empty)
    }

    test("a single cell dies") {
        new Universe(Set(center)).next.cells should be('empty)
    }

    test("a cell with two neighbors lives 1") {
        new Universe(Set(left, center, right)).next.cells should
contain(center)
    }

    test("a cell with two neighbors lives 2") {
        new Universe(Set(topLeft, topCenter, topRight)).next.cells
should contain(topCenter)
    }

    test("a cell with two neighbors lives, vertical") {
        new Universe(Set(topCenter, center,
bottomCenter)).next.cells should contain(center)
    }

    test("a cell with three neighbors starts to live") {
        new Universe(Set(left, center, right)).next.cells should
contain(topCenter)
    }
```

```
    test("a cell with four neighbors dies") {
        new Universe(Set(left, center, right, topRight,
topCenter)).next.cells should not contain (center)
    }

}

class Universe(val cells : Set[(Int, Int)]) {
    def this() = this(Set())
    def next() : Universe = new Universe(survivingCells ++
bornCells)

    private def bornCells : Set[(Int, Int)] =
        cells.flatMap(neighbors).filter(willBeBorn)

    private def willBeBorn(cell : (Int, Int)) : Boolean =
numberOfLivingNeighbors(cell) == 3

    private def survivingCells : Set[(Int, Int)] =
        cells flatMap (c => if (willLive(c)) Some(c) else None)

    private def numberOfLivingNeighbors(cell : (Int, Int)) : Int =
livingNeighbors(cell).size

    private def neighbors(cell : (Int, Int)) = for (
        x <- -1 to 1;
        y <- -1 to 1;
        if x != 0 || y != 0
    ) yield (cell._1 + x, cell._2 + y)

    private def livingNeighbors(cell : (Int, Int)) =
neighbors(cell).filter(cells.contains(_))

    private def willLive(c : (Int, Int)) : Boolean = {
        val livingNeighbors = numberOfLivingNeighbors(c)
        livingNeighbors >= 2 && livingNeighbors < 4
    }
}
```

Kontaktadresse:
Jens Schauder
LINEAS Informationstechnik GmbH
Theodor-Heuss-Str 2
D-38122 Braunschweig

Telefon:	+49 (0) 0531-8852-1228
Fax:	+49 (0) 0531-8852-1528
E-Mail	jens.schauder@lineas.de
Internet:	www.lineas.de