

Hausarbeit-Zelluläre Automaten -

I143PraxisderSoftwareentwicklung



GITLAB: HAUSARBEIT_I143_GAMRADT_CC

von Christian Apsel(9090), Fabian Forthmann(9104), Georg Mezlaw(8881), Rane
Petersen(9132)

Inhaltsverzeichnis

1.	Einleitung.....	2
	Parity Modell	2
	Game of Life Modell	2
2.	UML – OO Struktur	3
	GridArray und GridPoints	5
	GridFactory	5
	Logger	5
	CopyGrid.....	6
	Parity	6
	GameOfLife.....	6
	GameTypeFactory	6
	ConfigReader	6
	ConfigValidator.....	6
	Main.....	6
	Game	6
3.	Wart-, Wiederverwend- und Erweiterbarkeit.....	7
4.	Installations- und Nutzungsanleitung.....	8
5.	Selbstreflexion der Gruppenmitglieder.....	9
i.	Christian Apsel.....	9
ii.	Fabian Forthmann	9
iii.	Georg Mezlaw.....	9
iv.	Rane Petersen	9
6.	Quellen und Hilfsmittel	10
7.	Abbildungs- und Tabellenverzeichnis.....	10

1. Einleitung

Diese Dokumentation wurde im Rahmen für das Modul I143 Praxis der Softwareentwicklung erstellt. Es geht um das Thema der zellulären Automaten. Dabei haben wir die festgelegten Regelsätze aus der Aufgabenstellung umgesetzt:

Parity Modell

„Eine Zelle ist genau dann lebendig, wenn in ihrer von-Neumann Nachbarschaft eine ungerade Anzahl lebender Zellen existiert, andernfalls ist sie tot.“

- Startkonfiguration 1: Grid-Size: 400x400, alle Zellen werden mit "tot" initialisiert. Lebendig sind lediglich 4 Zellen in der Mitte.
- Startkonfiguration 2: Grid-Size: 100x100, gerade Zeilen werden abwechselnd mit "tot" "lebendig" initialisiert. Ungerade Zeilen werden abwechselnd mit "lebendig" "tot" initialisiert.

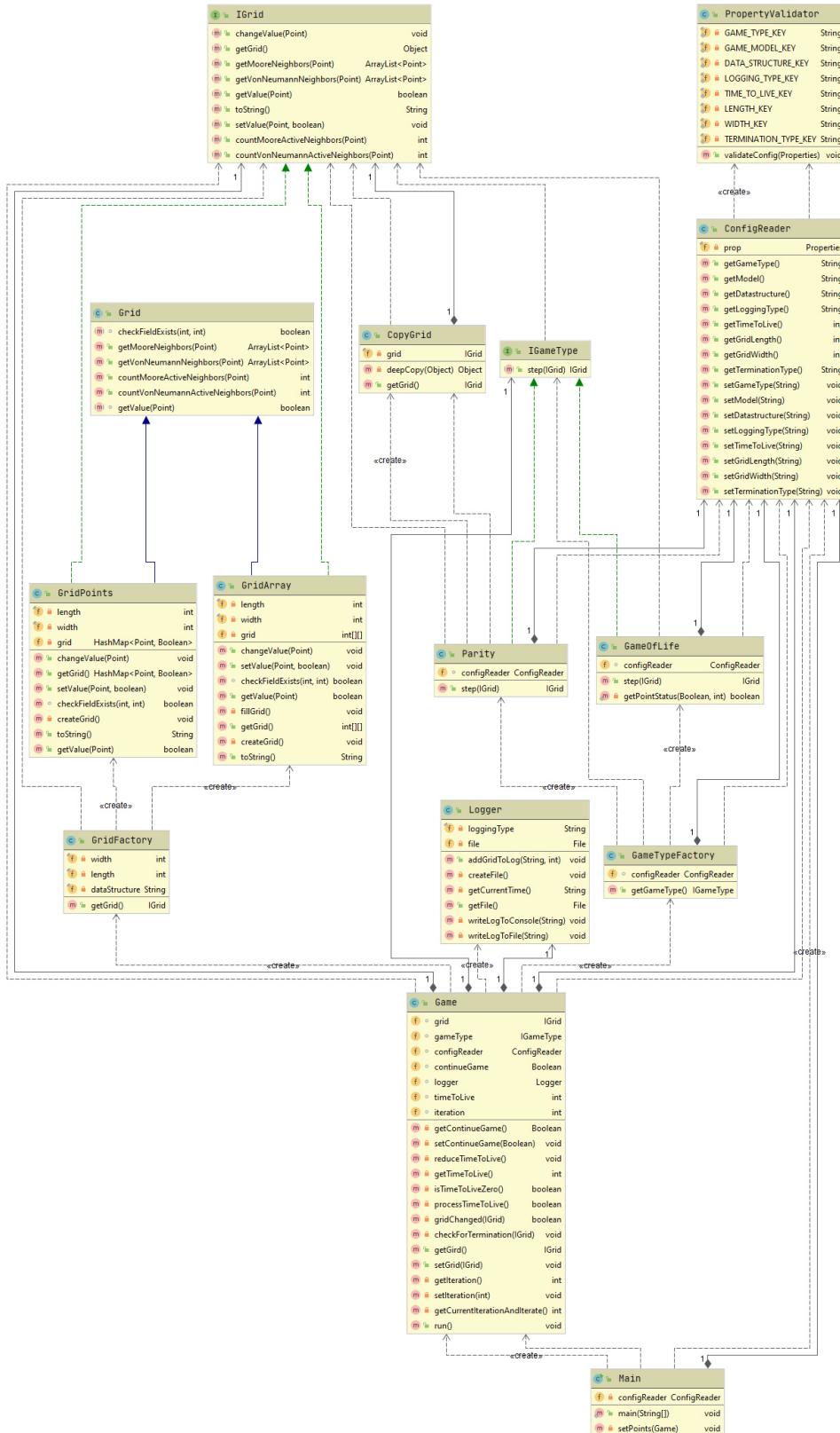
Game of Life Modell

„Eine lebendige Zelle bleibt lebendig, wenn Sie genau 2 oder 3 lebendige Nachbarn besitzt. Andernfalls stirbt sie. Eine tote Zelle bleibt tot bis sie exakt 3 lebendige Nachbarn hat“.

- Startkonfiguration 1: Grid-Size: 40x41, Nachbarschaft: Moore, Mittig platziert
- Startkonfiguration 2: Grid-Size: 100x100, gerade Zeilen werden abwechselnd mit "tot" "lebendig" initialisiert. Ungerade Zeilen werden abwechselnd mit "lebendig" "tot" initialisiert. Nachbarschaft: Moore
- Startkonfiguration 3: Grid-Size: 300x300, alle Zellen sind "tot", Nachbarschaft: Moore

Entwickelt wurde mit der IntelliJ IDE. Zur Versionsverwaltung wurde das Gitlab der Nordakademie genutzt. Während des Gitlab-Ausfalls, waren wir gezwungen auf ein privates Github-Repository auszuweichen. Alle Commits des Repos wurden durch das Wechseln des Remotes eins zu eins in das Gitlab übernommen (24.10-25.10). Auf Anfrage wird selbstverständlich ein Zugang zu dem privaten Repository gewährt. Es wurde überwiegend mit der Arbeitstechnik des Pair Programming gearbeitet. Zweier Gruppen haben im Wechsel zusammen am Quellcode gearbeitet. Dabei hat eine Person programmiert, während die Zweite über die Problemstellung nachgedacht, den geschriebenen Code kontrolliert und Auffälligkeiten angesprochen hat. Es wurde eine Grobe Aufteilung der Programmierung vorgenommen. Herr Forthmann und Herr Petersen haben überwiegend an der Spielelogik (Paket *gamelogic*), an dem *ConfigReader* und an dem *PropertyValidator* gearbeitet. Herr Apsel und Herr Mezlaw haben überwiegend an dem Spielfeld (Paket *grid*), dem *Logger* und der *Main* gearbeitet. Gegenseitige Unterstützung war dabei unerlässlich. Es wurde stets Test-First entwickelt. Um die individuellen Leistungen kenntlich zu machen, wurden die Verantwortlichen mithilfe der JavaDocs kenntlich gemacht. Die Dokumentation wurde von Georg Mezlaw verfasst.

2. UML – OO Struktur



Powered by yFiles

Abbildung 1 - UML ohne Experimente

Aus Gründen der Übersichtlichkeit wurden in dieser Abbildung die Experimente weggelassen. Das vollständige UML ist im Anhang.

Nachfolgend wird unsere Struktur mit den einzelnen Objekten näher beschrieben.

Unsere OO-Struktur setzt sich aus zwei [Interfaces](#) zusammen. Das [IGrid-Interface](#) enthält wichtige Informationen für das Spielfeld (Grid) und enthält folgende Methoden:

Tabelle 1 IGrid Methoden

<u>Name der Methode</u>	<u>Beschreibung</u>
changeValue	Setzt den Wert an einem Punkt – tauscht den Status, ob die Zelle lebt oder nicht, an einem Punkt (true oder false)
getGrid	Gibt das Spielfeld zurück
getMooreNeighbors	Gibt die Nachbarn an einem Punkt nach dem Moore-Schema zurück
getVonNeumannNeighbors	Gibt die Nachbarn an einem Punkt nach dem Von-Neumann-Schema zurück
getValue	Gibt den Wert eines Punkts wieder Lebt die Zelle oder nicht (true, false)
toString:	Umwandlung in einen String
setValuePoint	Setzt die Werte für einen Punkt
countMooreActiveNeighbors	Zählt die Anzahl der Nachbarn laut dem Moore-Schema
countNeumannActiveNeighbors	Zählt die Anzahl der Nachbarn laut dem Von-Neumann-Schema

Das [IGameType-Interface](#) enthält die Methode *step*. Dieses geht über das Spielfeld und startet die nächste Spieliteration. Es überprüft für jede Zelle wie diese sich verändert und schreibt das Ergebnis in ein neues Spielfeld

Tabelle 2 - IGameType Methoden

<u>Name der Methode</u>	<u>Beschreibung</u>
Step	Startet nächste Spieliteration

Außerdem wurde die abstrakte Superklasse *Grid* erstellt. Diese enthält ebenfalls die Methoden *countMooreActiveNeighbors*, *countNeumannActiveNeighbors*, *getMooreNeighbors*,

`getVonNeumannNeighbors` und `getValue` wie aus dem Interface `IGrid`. Zusätzlich enthält es die Methode `checkFieldExists`, die überprüft, ob ein Feld existiert.

Tabelle 3 - Grid Methoden

Name der Methode	Beschreibung
<code>countMooreActiveNeighbors</code>	Zählt die Anzahl der Nachbarn laut dem Moore-Schema
<code>countNeumannActiveNeighbors</code>	Zählt die Anzahl der Nachbarn laut dem Von-Neumann-Schema
<code>getMooreNeighbors</code>	Gibt die Nachbarn an einem Punkt nach dem Moore-Schema zurück
<code>getVonNeumannNeighbors</code>	Gibt die Nachbarn an einem Punkt nach dem Von-Neumann-Schema zurück
<code>getValue</code>	Gibt den Wert eines Punkts wieder Lebt die Zelle oder nicht (true, false)
<code>checkFieldExists</code>	Überprüft, ob ein Feld bereits existiert

Im Folgenden werden die entwickelten Klassen näher erläutert.

GridArray und GridPoints

Enthalten die `Grid`-Klasse als Oberklasse und das `IGrid` als Interface. Zusätzlich sind folgende Methoden implementiert:

Tabelle 4 - GridArray und GridPoints Methoden

Name der Methode	Beschreibung
<code>createGrid</code>	Erstellt ein Spielfeld
<code>fillGrid</code>	Füllt das Spielfeld initial mit Nullen

GridFactory

Diese Factory wird verwendet, um mehrere Objektinstanzen erzeugen zu können, erbt aus dem Interface `IGrid` und nutzt die Methode `getGrid` um das aktuelle Spielfeld zu bekommen.

Logger

Die Ausgabe der Simulation soll in eine Log-Datei erfolgen. Dafür wird diese Klasse mit folgenden Methoden verwendet:

Tabelle 5 - Logger Methoden

Name der Methode	Beschreibung
<code>addGridToLog</code>	Fügt einen String hinzu und ruft dann <code>writeLogToFile</code> oder <code>writeLogToConsole</code> auf

createFile	Erstellt eine Log-Datei
getCurrentTime	Holt sich die aktuelle Zeit für den Namen der Logfile
getFile	Gibt die File zurück
writeLogToConsole	Schreibt den Log-Inhalt in die Konsole
writeLogToFile	Schreibt den Log-Inhalt in eine Datei

CopyGrid

Hiermit wird eine Kopie eines Spielfeldes erstellt. Dies ist notwendig, da sonst nur Pointer existieren würden

Parity

Diese Klasse überprüft mithilfe der Methode *step*, ob ein Punkt "tot" oder "lebendig" ist.

GameOfLife

Enthält die Spiellogik für den Spielmodus des Game of Life, hat wie *Parity* die *step* Methode und zusätzlich wegen weiteren Spielregeln eine erweiterte Logik in der Methode *getPointStatus*.

GameTypeFactory

Eine Fabrik, die eine Methode *getGameType* enthält, mit dessen Hilfe Instanzen von *Parity* und *GameOfLife* erstellt werden.

ConfigReader

Liest die Konfigurationsdatei aus und ist für die Anpassung von Eigenschaften zuständig. Mit den unterschiedlichen GET-Methoden können die Eigenschaften einzelner Werte zurückgegeben werden.

ConfigValidator

Diese Klasse besitzt die Methode *validateConfig*, die ein Property-Objekt von dem *ConfigReader* bekommt und guckt, ob alle Felder da sind und wenn ja, ob sie auch mit validen Daten gesetzt sind.

Main

Diese Klasse wird verwendet, um das Programm initial zu starten. Diese enthält die Methode *getPoints*, die ein Userinterface zur Eingabe von aktiven Punkten erzeugt.

Game

Diese Klasse ist das Kernstück des Spiels. Diese fasst die Spiellogik zusammen und erstellt das Spiel.

Tabelle 6 - Game Methoden

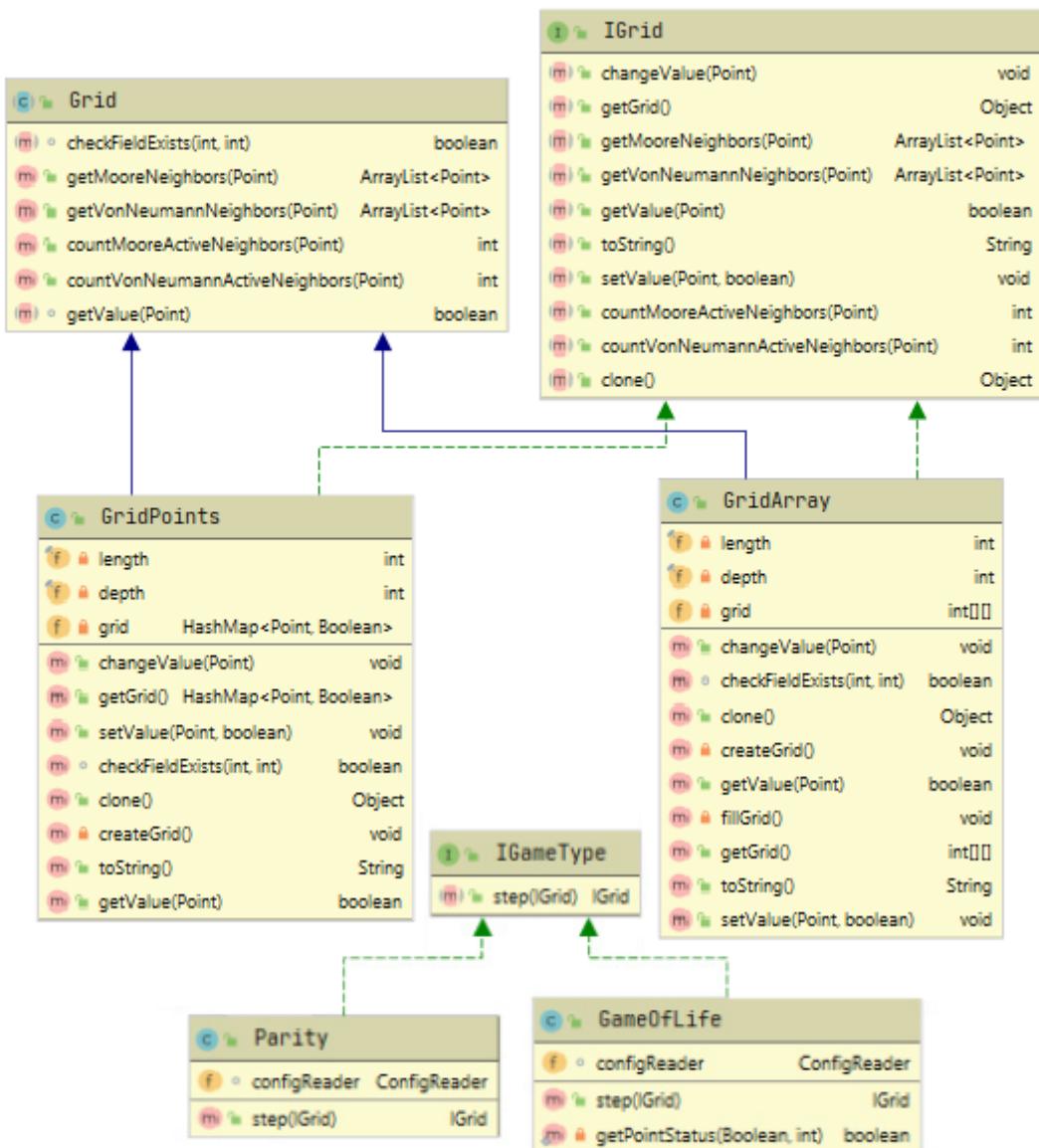
Name der Methode	Beschreibung
getContinueGame	Gibt zurück, ob das Spiel vorbei ist (true, false)
setContinueGame	Legt fest, ob das Spiel vorbei ist
checkForTermination	Liest aus dem ConfigReader welche Bedingungen erfüllt sein müssen und nach diesen Bedingungen wird ein Abbruch des Spiels ausgelöst
reduceTimeToLive	Reduziert die Lebenszeit, zählt herunter

processTimeToLive	Ruft die Methoden <code>reduceTimeToLive</code> und <code>isTimeToLiveZero</code> auf
isTimeToLiveZero	Gibt zurück, ob die Lebenszeit den Wert Null entspricht
gridChange	Überprüft, ob das Spielfeld verändert wurde
setGrid	Überschreibt das Spielfeld mit aktuellen Werten
run	Startet das Spiel. Startet den GameOfLife oder Parity Spielmodus mit dem Game-Objekt und dem Spielfeld

3. Wart-, Wiederverwend- und Erweiterbarkeit

Interfaces dienen als Schnittstelle, um festzulegen über welche Methoden die einzelnen Klassen enthalten müssen. Daher enthält ein Interface nur Konstanten und Funktionsköpfe. Dies ist sinnvoll um die Wart-, Wiederverwend- und Erweiterbarkeit zu gewährleisten. Dafür nutzen wir die Interfaces `IGrid` und `IGameType`. Diese beinhalten die gewollten Methoden und werden an die gewünschten Klassen `GridArray`, `GridPoints`, `GameOfLife`, `Parity`, `GameTypeFactory` und `GridFactory` vererbt. Dazu nutzen wir Factories wie `GameTypeFactory` und `GridFactory`. Wenn man mehrere Instanzen eines Objekts schnell erstellen möchte, verwendet man eine Factory. Eine Factory ist eine Funktion, die ein Objekt zurückgibt und wiederverwendet werden kann, um viele Objektinstanzen zu erzeugen. Mit der Hilfe von Interfaces und Factories gewährleisten wir eine Wartbarkeit, Wiederverwendbarkeit und Erweiterbarkeit. Durch die Verwendung dieser Interfaces und Factories ist es möglich neue GameTypes oder Datenstrukturen hinzuzufügen ohne das Programm grundlegend zu verändern.

In der folgenden Abbildung sind die Beziehungen zwischen Interfaces und Klassen verdeutlicht:



4. Installations- und Nutzungsanleitung

Für die Nutzung der Anwendung wird Java 8 benötigt. Zusätzlich wird für die Ausführung JetBrains IntelliJ IDEA (Version 2020.2.3) benötigt. Um die Anwendung auszuführen wird diese in IntelliJ importiert (**New->Project from version control**) und mit der Projekt-URL (https://gitlab2.nordakademie.de/ChristianApsel-A18/hausarbeit_i143_gamradt_cc.git) importiert. Anschließend wird über Maven der Befehl `mvn clean install` ausgeführt. Dadurch wird sichergestellt, dass die nötigen Abhängigkeiten vorhanden sind. Danach können beliebige Experimente (`src\main\java\de\nordakademie\pdse\experiments`) ausgeführt werden oder (`src\main\java>Main.java`) mit beliebigen Einstellungen gestartet werden. Die Einstellungen werden in der `config.properties` festgelegt. Nun wird die Main gestartet. Die Zellenwerte sind initial auf „tot“ gesetzt und können über den die Abfrage aus dem Programm verändert werden. Mit dem Befehl „run“ wird das Spiel gestartet. Die Ausgabe wird bei entsprechender Konfiguration in dem Stammverzeichnis erzeugt.

5. Selbstreflexion der Gruppenmitglieder

i. Christian Apsel

Die Arbeit über Distanz lief zumindest aus meiner Sicht ohne Probleme. Es gab mit dem Pair-Programming soweit mir bekannt auch keine Probleme, auch wenn es schwieriger ist, wirklich Test-first zu arbeiten, gerade wenn eine andere Person eine Methode benötigt, welche zu diesem Zeitpunkt noch nicht implementiert wurde. Die Aufgabenstellungen finde ich an der ein oder anderen Stelle etwas schwach formuliert, bzw. formuliert. Ein weiteres Problem war, dass Gitlab über 24 Stunden offline war, was bei manchen im Zeitmanagement zu Problemen geführt hat.

ii. Fabian Forthmann

Meiner Meinung nach lief die Hausarbeit im Großen und Ganzen sehr gut, obwohl wir uns kein einziges Mal in Person, sondern immer nur digital, getroffen haben. Besonders sinnvoll fand ich unsere ausgiebige Planungsphase am Anfang des Projektes, wo wir uns die Architektur und die Arbeitsverteilung für unterschiedliche Funktionsbereiche des Programms überlegt haben. Positiv ist mir außerdem unsere Entscheidung aufgefallen in Zweierpaaren zu programmieren. Herr Petersen und ich hatten uns nach kurzer Zeit als Team eingespielt und ich hatte das Gefühl, dass wir zusammen effektiver besseren Quellcode programmiert haben, als wir es vielleicht allein gemacht hätten. Auch die Aufgabenstellung war aus meiner Sicht nach etwas überlegen gut schaffbar. Bei meiner nächsten Hausarbeit würde ich die Planungsphase noch ausführlicher gestalten und so beispielsweise bereits definieren, welche Methoden genau in eine Klasse gehören. Außerdem könnte es aus meiner Sicht sinnvoll sein, Deadlines für bestimmte Meilensteine zu setzen, um das Warten auf benötigte Codestellen zu minimieren und es könnte sinnvoll sein, bereits im Vorhinein Standards für die Ausdrucksweise im Code zu besprechen.

iii. Georg Mezlaw

Die Gruppenarbeit lief meiner Meinung nach gut. Trotz der örtlichen Distanz konnten wir uns virtuell verabreden. Das Pair-Programming lief ebenfalls gut. Wir haben uns geholfen und unterstützt. Die Aufteilung war meiner Meinung nach, auch im Nachhinein sinnvoll, da so gezielt ohne Konfrontationen gearbeitet werden konnte. Trotz der Aufteilungen konnte man Anpassungen am gesamten Quellcode vornehmen. Zwischenzeitlich war das Gitlab der Nordakademie ausgefallen, was uns zuerst gehindert hat weiter zu arbeiten. Das Problem haben wir mit einem privaten Github-Repo gelöst, bis das GitLab wieder erreichbar war. Nächstes Mal würde ich direkt die Premium-Version von IntelliJ nutzen anstatt die Community-Edition, wie zu beginn. Ich würde mir für das nächste Mal ein besser formatiertes PDF als Aufgabenstellung wünschen, da dieses Dokument nicht durchsuchbar ist bzw. so formatiert, dass Worte und Sätze zusammengerutscht sind.

iv. Rane Petersen

Die Gruppenarbeit als auch der Erstellungsprozess war ein Erfolg. Dennoch gibt es Punkte, die sich beikommenden Projekten durchaus verbessern lassen. Dazu zählen u. A. die Ausarbeitung der Architektur. Die Idee für die einzelnen Klassen wurde schnell gefunden hatte aber nicht ausreichend tiefe, was sich in den nächsten Schritten als problematisch erwiesen hat. Das fehlende Design der einzelnen Methoden sorgte für die mehrfache Anpassung der Klassen, welche sich durch eine

für Anpassungen gegen Ende des Projekts, welche sich hätten vermeiden lassen können. Zusätzlich sind aufgrund meines Auslandssemester und Arbeiten an der Uni vor Ort mehrere verschiedene Mailadressen zum Committen benutzt worden. Dies ist etwas, was sich mit höherer Aufmerksamkeit hätte vermeiden lassen können.

6. Quellen und Hilfsmittel

<https://www.baeldung.com/java-testing-system-out-println> (23.10.2020)

<https://howtodoinjava.com/junit/junit-creating-temporary-filefolder-using-temporaryfolder-rule/> (24.10.2020)

[https://alvinalexander.com/java/java_file_exists-directory-exists/](https://alvinalexander.com/java/java-file-exists-directory-exists/) (30.10.2020)

https://www.w3schools.com/java/java_files_delete.asp (30.10.2020)

<https://stackoverflow.com/questions/1459656/how-to-get-the-current-time-in-yyyy-mm-dd-hhmisec-millisecond-format-in-java> (30.10.2020)

<https://stackoverflow.com/questions/19628157/switch-ignore-case-in-java-7> (30.10.2020)

7. Abbildungs- und Tabellenverzeichnis

Abbildung 1 - UML ohne Experimente 3

Tabelle 1 - IGrid Methoden	4
Tabelle 2 - IGameType Methoden.....	4
Tabelle 3 - Grid Methoden	5
Tabelle 4 - GridArray und GridPoints Methoden	5
Tabelle 5 - Logger Methoden	5
Tabelle 6 - Game Methoden.....	6

8. Anhang (Quellcode, Eidesstattliche Erklärungen, vollständiges UML)

Eidesstattliche Erklärung Fabian Forthmann

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form weder von mir noch von jemand anderem als Prüfungsleistung vorgelegt.

Datum: 29.10.2020

Unterschrift:



F. Forthmann



Eidesstattliche Erklärung des Studenten / der Studentin

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form weder von mir noch von jemand anderem als Prüfungsleistung vorgelegt.

Datum: 02.11.2020

Unterschrift: G. Mezlaw



Eidesstattliche Erklärung des Studenten / der Studentin

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form weder von mir noch von jemand anderem als Prüfungsleistung vorgelegt.

Datum: 01.11.2020

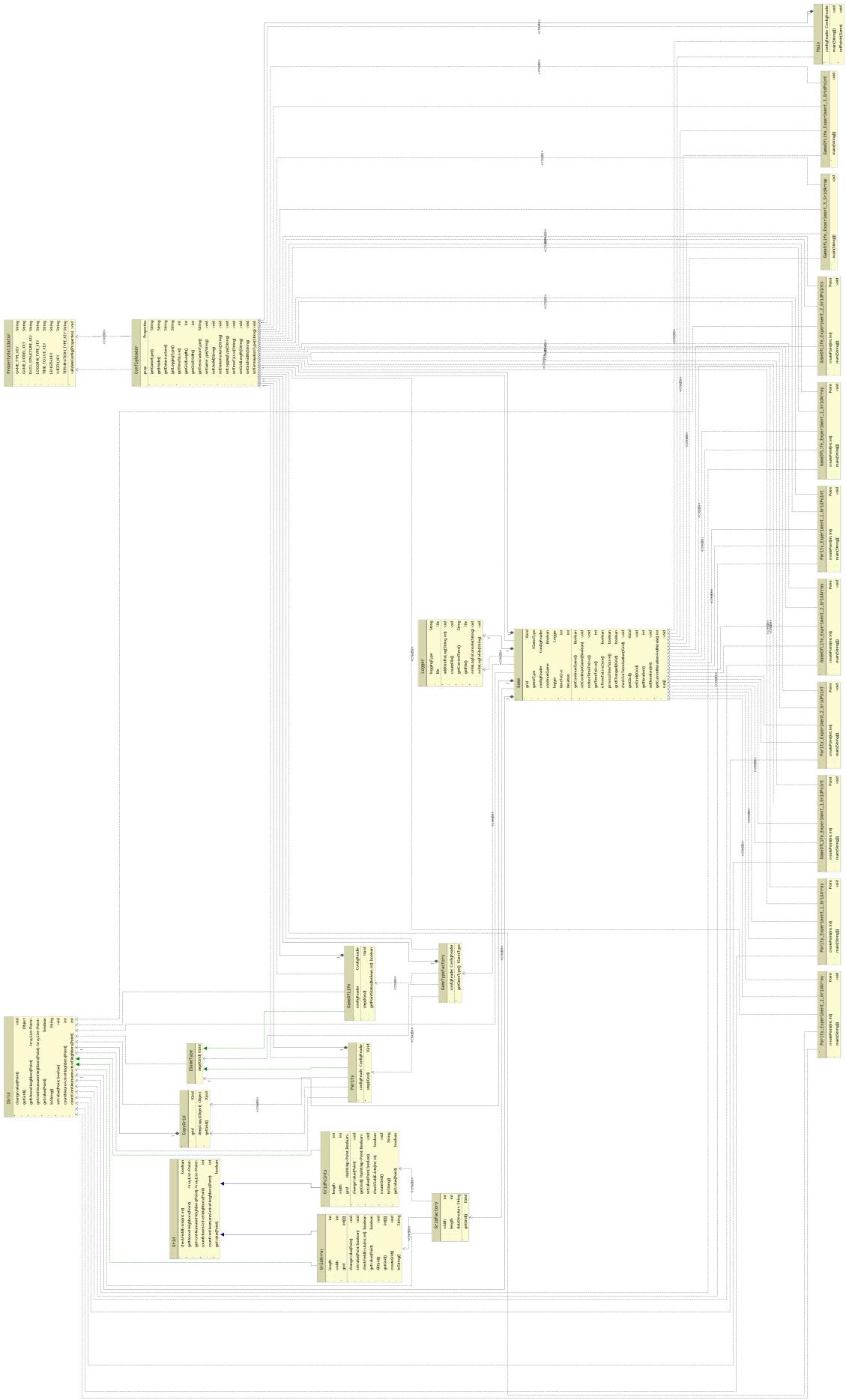
Unterschrift: A. Petersen

Eidesstattliche Erklärung des Studenten / der Studentin

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form weder von mir noch von jemand anderem als Prüfungsleistung vorgelegt.

Datum: 01.11.2020

Unterschrift: C. Apose



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.
 apache.org/xsd/maven-4.0.0.xsd">
5   <modelVersion>4.0.0</modelVersion>
6
7   <groupId>org.nordakademie</groupId>
8   <artifactId>hausarbeit_i143_gamradt_cc</artifactId>
9   <version>1.0-SNAPSHOT</version>
10  <build>
11    <plugins>
12      <plugin>
13          <groupId>org.apache.maven.plugins</groupId>
14          <artifactId>maven-compiler-plugin</artifactId>
15          <configuration>
16              <source>8</source>
17              <target>8</target>
18          </configuration>
19      </plugin>
20    </plugins>
21  </build>
22  <dependencies>
23      <dependency>
24          <groupId>junit</groupId>
25          <artifactId>junit</artifactId>
26          <version>4.12</version>
27          <scope>test</scope>
28      </dependency>
29      <dependency>
30          <groupId>org.mockito</groupId>
31          <artifactId>mockito-core</artifactId>
32          <version>3.5.13</version>
33          <scope>test</scope>
34      </dependency>
35      <dependency>
36          <groupId>commons-io</groupId>
37          <artifactId>commons-io</artifactId>
38          <version>2.6</version>
39      </dependency>
40  </dependencies>
41 </project>
```

1 .idea
2 *.iml
3 target
4

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <module org.jetbrains.idea.maven.project.MavenProjectsManager.isMavenModule="true" type="JAVA_MODULE" version="4">
3   <component name="NewModuleRootManager" LANGUAGE_LEVEL="JDK_1_8">
4     <output url="file://$MODULE_DIR$/target/classes" />
5     <output-test url="file://$MODULE_DIR$/target/test-classes" />
6     <content url="file://$MODULE_DIR$">
7       <sourceFolder url="file://$MODULE_DIR$/src/main/java" isTestSource="false" />
8       <sourceFolder url="file://$MODULE_DIR$/src/main/resources" type="java-resource" />
9       <sourceFolder url="file://$MODULE_DIR$/src/test/java" isTestSource="true" />
10      <excludeFolder url="file://$MODULE_DIR$/target" />
11    </content>
12    <orderEntry type="inheritedJdk" />
13    <orderEntry type="sourceFolder" forTests="false" />
14    <orderEntry type="library" scope="TEST" name="Maven: junit:junit:4.12" level="project" />
15    <orderEntry type="library" scope="TEST" name="Maven: org.hamcrest:hamcrest-core:1.3" level="project" />
16    <orderEntry type="library" scope="TEST" name="Maven: org.mockito:mockito-core:3.5.13" level="project" />
17    <orderEntry type="library" scope="TEST" name="Maven: net.bytebuddy:bytebuddy:1.10.15" level="project" />
18    <orderEntry type="library" scope="TEST" name="Maven: net.bytebuddy:bytebuddy-agent:1.10.15" level="project" />
19    <orderEntry type="library" scope="TEST" name="Maven: org.objenesis:objenesis:3.1" level="project" />
20    <orderEntry type="library" name="Maven: commons-io:commons-io:2.6" level="project" />
21  </component>
22 </module>
```

```

1 import de.nordakademie.pdse.config.ConfigReader;
2 import de.nordakademie.pdse.gamelogic.Game;
3
4 import java.awt.*;
5 import java.io.File;
6 import java.util.Scanner;
7
8 /**
9  * This class is used to start the program initially. It contains the method
10 * getPoints, which creates a user interface for the input of active points.
11 * @author Christian Apsel, Rane Petersen
12 */
13 public class Main {
14     private ConfigReader configReader;
15
16     public static void main(String[] args) throws Exception {
17
18         Main main = new Main();
19         File file;
20
21         if (args.length != 0) {
22             file = new File(args[0]);
23         } else {
24             file = new File("src/main/resources/config.properties");
25         }
26
27         main.configReader = new ConfigReader(file);
28         Game game = new Game(main.configReader);
29         main.setPoints(game);
30         game.run();
31     }
32
33
34     private void setPoints(Game game) {
35         String run = "";
36         Scanner scanner = new Scanner(System.in);
37         while (!run.equalsIgnoreCase("run")) {
38             System.out.println("Bitte geben Sie aktive Punkte ein: X,Y / Bitte
39             geben Sie run zum starten des Programms ein");
40             String temp = scanner.next();
41             if (temp.contains(",")) {
42                 String[] s = temp.split(",");
43                 if (s.length == 2 && Integer.parseInt(s[0]) < configReader.
44                     getGridLength() && Integer.parseInt(s[1]) < configReader.getGridWidth()) {
45                     game.getGird().setValue(new Point(Integer.parseInt(s[0]),
46                         Integer.parseInt(s[1])), true);
47                 } else {
48                     System.out.println("falsche Eingabe");
49                 }
50             } else if (temp.equalsIgnoreCase("run")) {
51                 run = "run";
52             } else {
53                 System.out.println("falsche Eingabe");
54             }
55         }
56     }
57
58 }
```

```

1 package de.nordakademie.pdse.grid;
2
3 import java.awt.*;
4 import java.util.ArrayList;
5
6 /**
7 * abstract superclass, contains the logic for the playing field, can count
8 * neighbors and find out if a field exists
9 * @author Georg Mezlaw
10 * @since 26.10.2020
11 */
12 public abstract class Grid {
13
14     abstract boolean checkFieldExists(int x, int y);
15
16     public ArrayList<Point> getMooreNeighbors(Point position) {
17         ArrayList<Point> neighbors = getVonNeumannNeighbors(position);
18         if (checkFieldExists(position.x - 1, position.y - 1)) {
19             neighbors.add(new Point(position.x - 1, position.y - 1));
20         }
21         if (checkFieldExists(position.x - 1, position.y + 1)) {
22             neighbors.add(new Point(position.x - 1, position.y + 1));
23         }
24         if (checkFieldExists(position.x + 1, position.y - 1)) {
25             neighbors.add(new Point(position.x + 1, position.y - 1));
26         }
27         if (checkFieldExists(position.x + 1, position.y + 1)) {
28             neighbors.add(new Point(position.x + 1, position.y + 1));
29         }
30         return neighbors;
31     }
32
33     public ArrayList<Point> getVonNeumannNeighbors(Point position) {
34         ArrayList<Point> neighbors = new ArrayList<>();
35         if (checkFieldExists(position.x + 1, position.y)) {
36             neighbors.add(new Point(position.x + 1, position.y));
37         }
38         if (checkFieldExists(position.x - 1, position.y)) {
39             neighbors.add(new Point(position.x - 1, position.y));
40         }
41         if (checkFieldExists(position.x, position.y + 1)) {
42             neighbors.add(new Point(position.x, position.y + 1));
43         }
44         if (checkFieldExists(position.x, position.y - 1)) {
45             neighbors.add(new Point(position.x, position.y - 1));
46         }
47         return neighbors;
48     }
49
50     public int countMooreActiveNeighbors(Point position) {
51         return (int) getMooreNeighbors(position).stream().filter(this::getValue)
52             .count();
53     }
54
55     public int countVonNeumannActiveNeighbors(Point position) {
56         return (int) getVonNeumannNeighbors(position).stream().filter(this::
57             getValue).count();
58     }
59     abstract boolean getValue(Point position);
60 }
```

```
1 package de.nordakademie.pdse.grid;
2
3 import java.awt.*;
4 import java.io.Serializable;
5 import java.util.ArrayList;
6
7 /**
8 * contains important information for the playing field. Can determine if a
9 * cell is alive or dead, return the playing field, return neighbors and count the
10 * neighbors
11 *
12 */
13 public interface IGrid extends Serializable {
14     void changeValue(Point position);
15
16     Object getGrid();
17
18     ArrayList<Point> getMooreNeighbors(Point point);
19
20     ArrayList<Point> getVonNeumannNeighbors(Point point);
21
22     boolean getValue(Point position);
23
24     String toString();
25
26     void setValue(Point position, boolean value);
27
28     public int countMooreActiveNeighbors(Point position);
29
30     public int countVonNeumannActiveNeighbors(Point position);
31
32 }
33
```

```
1 package de.nordakademie.pdse.grid;
2
3 import java.io.ByteArrayInputStream;
4 import java.io.ByteArrayOutputStream;
5 import java.io.ObjectInputStream;
6 import java.io.ObjectOutputStream;
7
8 /**
9  * This is used to create a copy of a playing field. This is necessary, because
10 * otherwise only pointers would exist
11 * @author Christian Apsel
12 * @since 26.10.2020
13 */
14 public class CopyGrid {
15     private final IGrid grid;
16
17     public CopyGrid(IGrid grid) {
18         this.grid = (IGrid) deepCopy(grid);
19     }
20
21     private Object deepCopy(Object object) {
22         try {
23             ByteArrayOutputStream byteArrayOutputStream = new
24                 ByteArrayOutputStream();
25             ObjectOutputStream objectOutputStream = new ObjectOutputStream(
26                 byteArrayOutputStream);
27             objectOutputStream.writeObject(object);
28             byteArrayInputStream byteArrayInputStream = new
29                 ByteArrayInputStream(byteArrayOutputStream.toByteArray());
30             ObjectInputStream objectInputStream = new ObjectInputStream(
31                 byteArrayInputStream);
32             return objectInputStream.readObject();
33         } catch (Exception e) {
34             e.printStackTrace();
35             return null;
36         }
37     }
38 }
39
```

```

1 package de.nordakademie.pdse.grid;
2
3 import java.awt.*;
4 import java.util.Arrays;
5
6 /**
7 * Contains the grid class as superclass and the IGrid as interface. Can create
8 * a playing field.
9 *
10 * @author Christian Apsel
11 * @since 26.10.2020
12 */
13 public class GridArray extends Grid implements IGrid {
14     private final int length;
15     private final int width;
16     private int[][] grid;
17
18     public GridArray(int length, int width) {
19         this.length = length;
20         this.width = width;
21         createGrid();
22     }
23
24     @Override
25     public void changeValue(Point position) {
26         int width = position.x;
27         int length = position.y;
28
29         if (getGrid()[width][length] == 0) {
30             getGrid()[width][length] = 1;
31         } else {
32             getGrid()[width][length] = 0;
33         }
34     }
35
36     @Override
37     public void setValue(Point position, boolean value) {
38         int width = position.x;
39         int length = position.y;
40         grid[width][length] = value ? 1 : 0;
41     }
42
43     @Override
44     boolean checkFieldExists(int x, int y) {
45         return x >= 0 && x <= width - 1 && y >= 0 && y <= length - 1;
46     }
47
48     @Override
49     public boolean getValue(Point position) {
50         int width = position.x;
51         int length = position.y;
52         return getGrid()[width][length] != 0;
53     }
54
55     private void fillGrid() {
56         for (int[] length : grid) {
57             Arrays.fill(length, 0);
58         }
59     }
60
61     @Override
62     public int[][] getGrid() {
63         return grid;

```

```
64     }
65
66     private void createGrid() {
67         grid = new int[width][length];
68         fillGrid();
69     }
70
71     //TODO Duplicate Code entfernen
72     @Override
73     public String toString() {
74         StringBuilder stringBuilder = new StringBuilder();
75         for (int i = 0; i < width; i++) {
76
77             for (int j = 0; j < length; j++) {
78
79                 if (getValue(new Point(i, j))) {
80                     stringBuilder.append(1);
81                 } else {
82                     stringBuilder.append(0);
83                 }
84             }
85             stringBuilder.append("\n");
86         }
87         return stringBuilder.toString();
88     }
89 }
90
```

```

1 package de.nordakademie.pdse.grid;
2
3 import java.awt.*;
4 import java.util.HashMap;
5 /**
6  * Contains the grid class as superclass and the IGrid as interface. Can create
7  * a playing field.
8  *
9  * @author Christian Apsel
10 * @since 26.10.2020
11 */
12 public class GridPoints extends Grid implements IGrid {
13     private final int length;
14     private final int width;
15     private HashMap<Point, Boolean> grid;
16
17     public GridPoints(int length, int width) {
18         this.length = length;
19         this.width = width;
20         createGrid();
21     }
22
23     @Override
24     public void changeValue(Point position) {
25         if (getGrid().get(position).equals(false)) {
26             getGrid().replace(position, true);
27         } else {
28             getGrid().replace(position, false);
29         }
30     }
31
32     @Override
33     public HashMap<Point, Boolean> getGrid() {
34         return grid;
35     }
36
37     @Override
38     public void setValue(Point position, boolean value) {
39         getGrid().replace(position, value);
40     }
41
42     @Override
43     boolean checkFieldExists(int x, int y) {
44         return x >= 0 && x <= width - 1 && y >= 0 && y <= length - 1;
45     }
46
47     private void createGrid() {
48         grid = new HashMap<>();
49         for (int i = 0; i < width; i++) {
50             for (int j = 0; j < length; j++) {
51                 grid.put(new Point(i, j), false);
52             }
53         }
54     }
55
56     //TODO Duplicate Code entfernen
57     @Override
58     public String toString() {
59         StringBuilder stringBuilder = new StringBuilder();
60         for (int i = 0; i < width; i++) {
61             for (int j = 0; j < length; j++) {
62
63                 if (getValue(new Point(i, j))) {

```

```
64             } else {
65                 stringBuilder.append(0);
66             }
67         }
68         stringBuilder.append("\n");
69     }
70     return stringBuilder.toString();
71 }
72
73 @Override
74 public boolean getValue(Point position) {
75     return getGrid().get(position);
76 }
77 }
78 }
```

```
1 package de.nordakademie.pdse.grid;
2
3 /**
4  * This factory is used to create multiple object instances, inherits from the
5  * interface IGrid and uses the method getGrid to get the current grid.
6  *
7  * @author Christian Apsel
8  * @since 26.10.2020
9 */
10 public class GridFactory {
11     private final int width;
12     private final int length;
13     private final String dataStructure;
14
15     public GridFactory(int length, int width, String dataStructure) {
16         this.width = width;
17         this.length = length;
18         this.dataStructure = dataStructure;
19     }
20
21     public IGrid getGrid() {
22         if ("GridArray".equalsIgnoreCase(dataStructure)) {
23             return new GridArray(length, width);
24         } else {
25             return new GridPoints(length, width);
26         }
27     }
28 }
```

```

1 package de.nordakademie.pdse.config;
2
3 import java.io.File;
4 import java.io.FileInputStream;
5 import java.util.Properties;
6
7 /**
8  * Reads the configuration file and is responsible for the adjustment of
9  * properties.
10 * With the different GET methods the properties of single values can be
11 * returned.
12 */
13 public class ConfigReader {
14
15     private final Properties prop = new Properties();
16
17     public ConfigReader(File propertyFile) {
18         try {
19             FileInputStream inputStream = new FileInputStream(propertyFile);
20             prop.load(inputStream);
21
22             PropertyValidator propertyValidator = new PropertyValidator();
23             propertyValidator.validateConfig(prop);
24         } catch (Exception e) {
25             throw new RuntimeException("The properties file could not be found
26 or loaded.");
27         }
28
29         public String getGameType() {
30             return prop.getProperty("game.type");
31         }
32
33         public String getModel() {
34             return prop.getProperty("game.model");
35         }
36
37         public String getDatastructure() {
38             return prop.getProperty("game.dataStructure");
39         }
40
41         public String getLoggingType() {
42             return prop.getProperty("game.loggingType");
43         }
44
45         public int getTimeToLive() {
46             return Integer.parseInt(prop.getProperty("game.timeToLive"));
47         }
48
49         public int getGridLength() {
50             return Integer.parseInt(prop.getProperty("game.grid.length"));
51         }
52
53         public int getGridWidth() {
54             return Integer.parseInt(prop.getProperty("game.grid.width"));
55         }
56
57         public String getTerminationType() {
58             return prop.getProperty("game.terminationType");
59         }
60
61         public void setGameType(String gameType) {

```

```
62     prop.setProperty("game.type", gameType);
63 }
64
65 public void setModel(String gameModel) {
66     prop.setProperty("game.model", gameModel);
67 }
68
69 public void setDatastructure(String datastructure) {
70     prop.setProperty("game.dataStructure", datastructure);
71 }
72
73 public void setLoggingType(String loggingType) {
74     prop.setProperty("game.loggingType", loggingType);
75 }
76
77 public void setTimeToLive(String timeToLive) {
78     prop.setProperty("game.timeToLive", timeToLive);
79 }
80
81 public void setGridLength(String gridLength) {
82     prop.setProperty("game.grid.length", gridLength);
83 }
84
85 public void setGridWidth(String gridWidth) {
86     prop.setProperty("game.grid.width", gridWidth);
87 }
88
89 public void setTerminationType(String terminationType) {
90     prop.setProperty("game.terminationType", terminationType);
91 }
92 }
93
```

```

1 package de.nordakademie.pdse.config;
2
3 import java.util.Properties;
4
5 /**
6  * This class gets a property object from the ConfigReader and checks if all
7  * fields are there and if yes, if they are set with valid data.
8  *
9  * @author Fabian Forthmann
10 */
11 public class PropertyValidator {
12     private static final String GAME_TYPE_KEY = "game.type";
13     private static final String GAME_MODEL_KEY = "game.model";
14     private static final String DATA_STRUCTURE_KEY = "game.dataStructure";
15     private static final String LOGGING_TYPE_KEY = "game.loggingType";
16     private static final String TIME_TO_LIVE_KEY = "game.timeToLive";
17     private static final String LENGTH_KEY = "game.grid.length";
18     private static final String WIDTH_KEY = "game.grid.width";
19     private static final String TERMINATION_TYPE_KEY = "game.terminationType";
20
21     public void validateConfig(Properties prop) {
22         boolean validDataInFile = true;
23         try {
24             if ("ttlOrNoChange".equalsIgnoreCase(prop.getProperty(
25                 TERMINATION_TYPE_KEY)) || "ttl".equalsIgnoreCase(prop.getProperty(
26                 TERMINATION_TYPE_KEY))) {
27                 if (Integer.parseInt(prop.getProperty(TIME_TO_LIVE_KEY)) < 0) {
28                     validDataInFile = false;
29                 }
30             } else if (!"noChange".equalsIgnoreCase(prop.getProperty(
31                 TERMINATION_TYPE_KEY))) {
32                 validDataInFile = false;
33             }
34             if (!"Parity".equalsIgnoreCase(prop.getProperty(GAME_TYPE_KEY))
35                 ) && !"GameOfLife".equalsIgnoreCase(prop.getProperty(GAME_TYPE_KEY)) {
36                 validDataInFile = false;
37             }
38             if (!"Moore".equalsIgnoreCase(prop.getProperty(GAME_MODEL_KEY))
39                 ) && !"vonNeumann".equalsIgnoreCase(prop.getProperty(GAME_MODEL_KEY)) {
40                 validDataInFile = false;
41             }
42             if (!"GridPoints".equalsIgnoreCase(prop.getProperty(
43                 DATA_STRUCTURE_KEY)) && !"GridArray".equalsIgnoreCase(prop.getProperty(
44                 DATA_STRUCTURE_KEY))) {
45                 validDataInFile = false;
46             }
47             if (!"console".equalsIgnoreCase(prop.getProperty(LOGGING_TYPE_KEY))
48                 ) && !"file".equalsIgnoreCase(prop.getProperty(LOGGING_TYPE_KEY)) && !"
49                 consoleAndFile".equalsIgnoreCase(prop.getProperty(LOGGING_TYPE_KEY)) && !"
50                 disable".equalsIgnoreCase(prop.getProperty(LOGGING_TYPE_KEY)) {
51                 validDataInFile = false;
52             }
53         } catch (RuntimeException e) {
54             throw new RuntimeException("The given property file is incomplete."
55             , e);
56         }
57         if (!validDataInFile) {

```

```
53         throw new RuntimeException("The given property file includes  
54             invalid data.");  
55     }  
56 }  
57
```

```

1 package de.nordakademie.pdse.logging;
2
3 import java.io.File;
4 import java.io.FileWriter;
5 import java.io.IOException;
6 import java.nio.file.Files;
7 import java.nio.file.Path;
8 import java.nio.file.Paths;
9 import java.time.LocalDateTime;
10 import java.time.format.DateTimeFormatter;
11
12 /**
13 * Returns the output of the simulation to a log file or the console
14 *
15 * @author Georg Mezlaw
16 * @since 26.10.2020
17 */
18 public class Logger {
19     private final String loggingType;
20     private File file;
21     private String fileName;
22
23     public Logger(String loggingType, String fileName) {
24         this.loggingType = loggingType;
25         if (!"console".equalsIgnoreCase(this.loggingType)) {
26             this.fileName = fileName + ".log";
27             createFile();
28         }
29     }
30
31     public Logger(String loggingType) {
32         this.loggingType = loggingType;
33         if (!"console".equalsIgnoreCase(this.loggingType)) {
34             this.fileName = "GridLog" + getCurrentTime() + ".log";
35             createFile();
36         }
37     }
38
39     public void addGridToLog(String grid, int iteration) {
40         StringBuilder stringBuilder = new StringBuilder();
41         stringBuilder.append("###" + iteration + "\n");
42         stringBuilder.append(grid);
43         switch (loggingType.toLowerCase()) {
44             case "consoleandfile":
45                 writeLogToFile(stringBuilder.toString());
46                 writeLogToConsole(stringBuilder.toString());
47                 break;
48             case "file":
49                 writeLogToFile(stringBuilder.toString());
50                 break;
51             case "console":
52                 writeLogToConsole(stringBuilder.toString());
53                 break;
54             case "disable":
55                 break;
56         }
57     }
58 }
59
60 private void createFile() {
61     Path path = Paths.get(fileName);
62     try {
63         Files.createFile(path);
64         file = new File(String.valueOf(path));

```

```
65             } catch (IOException ignored) {
66                 file = new File(String.valueOf(path));
67             }
68         }
69     }
70
71     private String getCurrentTime() {
72         DateTimeFormatter dateTimeFormatter = DateTimeFormatter.ofPattern("yyyy_MM_dd_HH-mm-ss-SSS");
73         LocalDateTime localDateTime = LocalDateTime.now();
74         return dateTimeFormatter.format(localDateTime);
75     }
76
77     public File getFile() {
78         return file;
79     }
80
81     private void writeLogToConsole(String input) {
82         System.out.println(input);
83     }
84
85     private void writeLogToFile(String input) {
86         try {
87             FileWriter fileWriter = new FileWriter(getFile(), true);
88             fileWriter.append(input);
89             fileWriter.append(System.lineSeparator());
90             fileWriter.close();
91         } catch (IOException ignored) {
92         }
93     }
94 }
95
```

```

1 package de.nordakademie.pdse.gamelogic;
2
3 import de.nordakademie.pdse.config.ConfigReader;
4 import de.nordakademie.pdse.grid.CopyGrid;
5 import de.nordakademie.pdse.grid.GridFactory;
6 import de.nordakademie.pdse.grid.IGrid;
7 import de.nordakademie.pdse.logging.Logger;
8
9 /**
10 * It summarizes the game logic and creates the game.
11 *
12 * @author Rane Petersen, Fabian Forthmann
13 * @since 24.10.2020
14 */
15 public class Game {
16
17     IGrid grid;
18     IGameType gameType;
19     ConfigReader configReader;
20     Boolean continueGame;
21     Logger logger;
22     int timeToLive;
23     int iteration;
24
25
26     public Game(ConfigReader configReader) {
27         this.configReader = configReader;
28         this.timeToLive = configReader.getTimeToLive();
29         this.grid = new GridFactory(configReader.getGridLength(), configReader.
    getGridWidth(), configReader.getDatastructure()).getGrid();
30         this.gameType = new GameTypeFactory(configReader).getGameType();
31         this.logger = new Logger(configReader.getLoggingType());
32         this.iteration = 0;
33         if (configReader.getTerminationType().equalsIgnoreCase("ttl")) {
34             this.continueGame = timeToLive > 0;
35         } else {
36             this.continueGame = true;
37         }
38     }
39
40     public Game(ConfigReader configReader, String logFileName) {
41         this.configReader = configReader;
42         this.timeToLive = configReader.getTimeToLive();
43         this.grid = new GridFactory(configReader.getGridLength(), configReader.
    getGridWidth(), configReader.getDatastructure()).getGrid();
44         this.gameType = new GameTypeFactory(configReader).getGameType();
45         this.logger = new Logger(configReader.getLoggingType(), logFileName);
46         this.iteration = 0;
47         if (configReader.getTerminationType().equals("ttl")) {
48             this.continueGame = timeToLive > 0;
49         } else {
50             this.continueGame = true;
51         }
52     }
53
54     private Boolean getContinueGame() {
55         return continueGame;
56     }
57
58     private void setContinueGame(Boolean continueGame) {
59         this.continueGame = continueGame;
60     }
61
62     private void reduceTimeToLive() {

```

```

63     timeToLive--;
64 }
65
66     private int getTimeToLive() {
67         return timeToLive;
68     }
69
70     private boolean isTimeToLiveZero() {
71         if (getTimeToLive() > 0) {
72             return false;
73         } else {
74             return true;
75         }
76     }
77
78     private boolean processTimeToLive() {
79         reduceTimeToLive();
80         return isTimeToLiveZero();
81     }
82
83     private boolean gridChanged(IGrid grid) {
84         if (this.grid.toString().equals(grid.toString())) {
85             return false;
86         } else {
87             return true;
88         }
89     }
90
91     private void checkForTermination(IGrid newGrid) {
92         switch (configReader.getTerminationType().toLowerCase()) {
93             case "ttl":
94                 this.setContinueGame(!processTimeToLive());
95                 break;
96             case "nochange":
97                 this.setContinueGame(gridChanged(newGrid));
98                 break;
99             case "ttlornochange":
100                this.setContinueGame(!processTimeToLive() && this.gridChanged(
newGrid));
101                break;
102            }
103        }
104
105     public IGrid getGird() {
106         return grid;
107     }
108
109     public void setGrid(IGrid grid) {
110         this.grid = grid;
111     }
112
113     private int getIteration() {
114         return iteration;
115     }
116
117     private void setIteration(int iteration) {
118         this.iteration = iteration;
119     }
120
121     private int getCurrentIterationAndIterate() {
122         int i = this.getIteration();
123         i++;
124         this.setIteration(i);
125         i--;

```

```
126     return i;
127 }
128
129     public void run() throws Exception {
130         logger.addGridToLog(getGird().toString(),
131         getCurrentIterationAndIterate());
132         while (this.getContinueGame()) {
133             CopyGrid copyGrid = new CopyGrid(getGird());
134             IGrid newGrid = gameType.step(copyGrid.getGrid());
135             checkForTermination(newGrid);
136             if (gridChanged(newGrid) && !this.configReader.getTerminationType
137             ().equalsIgnoreCase("ttl")) {
138                 this.setGrid(newGrid);
139                 logger.addGridToLog(getGird().toString(),
140                 getCurrentIterationAndIterate());
141             }
142         }
143     }
144 }
145
```

```
1 package de.nordakademie.pdse.gamelogic;
2
3 import de.nordakademie.pdse.config.ConfigReader;
4 import de.nordakademie.pdse.grid.CopyGrid;
5 import de.nordakademie.pdse.grid.IGrid;
6
7 import java.awt.*;
8
9 /**
10 * This class uses the step method to check whether a point is "dead" or "alive".
11 *
12 * @author Rane Petersen
13 */
14 public class Parity implements IGameType {
15     ConfigReader configReader;
16
17     public Parity(ConfigReader configReader) {
18         this.configReader = configReader;
19     }
20
21     public IGrid step(IGrid oldGrid) {
22         CopyGrid copyGrid = new CopyGrid(oldGrid);
23         IGrid newGrid = oldGrid;
24         if ("vonNeumann".equalsIgnoreCase(configReader.getModel())) {
25             Point point;
26             for (int length = 0; length < configReader.getGridLength(); length
27                ++) {
28                 for (int width = 0; width < configReader.getGridWidth(); width
29                    ++) {
30                     point = new Point(length, width);
31                     newGrid.setValue(point, copyGrid.getGrid().
32                         countVonNeumannActiveNeighbors(point) % 2 == 1);
33                 }
34             }
35         }
36     }
37 }
```

```
1 package de.nordakademie.pdse.gamelogic;
2
3 import de.nordakademie.pdse.grid.IGrid;
4
5 /**
6  * It goes over the playing field and starts the next iteration of the game. It
7  * checks for each cell how it changes and writes the result into a new game
8  * iteration
9  */
10 public interface IGameType {
11
12     public IGrid step(IGrid oldGrid) throws Exception;
13 }
14
```

```

1 package de.nordakademie.pdse.gamelogic;
2
3 import de.nordakademie.pdse.config.ConfigReader;
4 import de.nordakademie.pdse.grid.CopyGrid;
5 import de.nordakademie.pdse.grid.IGrid;
6
7 import java.awt.*;
8
9 /**
10 * Contains the game logic for the game mode of the Game of Life, has like
11 * Parity the step method and additionally because of further game rules an
12 * extended logic in the method getPointStatus
13 */
14 public class GameOfLife implements IGameType {
15     ConfigReader configReader;
16
17     public GameOfLife(ConfigReader configReader) {
18         this.configReader = configReader;
19     }
20
21     @Override
22     public IGrid step(IGrid oldGrid) {
23         CopyGrid copyGrid = new CopyGrid(oldGrid);
24         IGrid newGrid = oldGrid;
25         Point point;
26         if ("Moore".equalsIgnoreCase(configReader.getModel())) {
27             for (int width = 0; width < configReader.getGridLength(); width
28++) {
29                 for (int length = 0; length < configReader.getGridWidth();
length++) {
30                     point = new Point(length, width);
31                     newGrid.setValue(point, getPointStatus(copyGrid.getGrid().get
32         value(point), copyGrid.getGrid().countMooreActiveNeighbors(point)));
33                 }
34             } else if ("vonNeumann".equalsIgnoreCase(configReader.getModel())) {
35                 for (int width = 0; width < configReader.getGridLength(); width
36++) {
37                     for (int length = 0; length < configReader.getGridWidth();
length++) {
38                         point = new Point(length, width);
39                         newGrid.setValue(point, getPointStatus(copyGrid.getGrid().get
40         value(point), copyGrid.getGrid().countVonNeumannActiveNeighbors(po
38
39
40
41         }
42     }
43
44     private static boolean getPointStatus(Boolean point, int aliveNeighbors) {
45
46         if (point.equals(true)) {
47             point = aliveNeighbors == 2 || aliveNeighbors == 3;
48         } else {
49             if (aliveNeighbors == 3) {
50                 point = true;
51             }
52         }
53     }
54 }
55 }
56

```

```
1 package de.nordakademie.pdse.gamelogic;
2
3 import de.nordakademie.pdse.config.ConfigReader;
4
5 /**
6 * A factory that contains a getGameType method that is used to create
7 * instances of Parity and GameOfLife.
8 * @author Rane Petersen
9 */
10 public class GameTypeFactory {
11     ConfigReader configReader;
12
13     public GameTypeFactory(ConfigReader configReader) {
14         this.configReader = configReader;
15     }
16
17     public IGameType getGameType() {
18         if (configReader.getGameType().equalsIgnoreCase("Parity")) {
19             return new Parity(configReader);
20         } else {
21             return new GameOfLife(configReader);
22         }
23     }
24 }
25
```

```
1 package de.nordakademie.pdse.experiments.parity;
2
3 import de.nordakademie.pdse.config.ConfigReader;
4 import de.nordakademie.pdse.gamelogic.Game;
5 import de.nordakademie.pdse.grid.IGrid;
6
7 import java.awt.*;
8 import java.io.File;
9
10 /**
11  * @author Georg Mezlaw, Rane Petersen
12 */
13 public class Parity_Experiment_1_GridArray {
14
15     private static Point createPoint(int x, int y) {
16         return new Point(x, y);
17     }
18
19     public static void main(String[] args) throws Exception {
20         ConfigReader configReader = new ConfigReader(new File("src/main/
resources/config.properties"));
21         configReader.setGridLength("400");
22         configReader.setGridWidth("400");
23         configReader.setModel("vonNeumann");
24         configReader.setGameType("Parity");
25         configReader.setTerminationType("ttl");
26         configReader.setTimeToLive("100");
27         configReader.setLoggingType("file");
28         configReader.setDatastructure("GridArray");
29         Game game = new Game(configReader, "Parity_Experiment_1_GridArray");
30         IGrid grid = game.getGird();
31         grid.setValue(createPoint(200, 200), true);
32         grid.setValue(createPoint(200, 201), true);
33         grid.setValue(createPoint(201, 200), true);
34         grid.setValue(createPoint(201, 201), true);
35         game.run();
36     }
37 }
38
```

```
1 package de.nordakademie.pdse.experiments.parity;
2
3 import de.nordakademie.pdse.config.ConfigReader;
4 import de.nordakademie.pdse.gamelogic.Game;
5 import de.nordakademie.pdse.grid.IGrid;
6
7 import java.awt.*;
8 import java.io.File;
9
10 /**
11  * @author Georg Mezlaw, Rane Petersen
12 */
13 public class Parity_Experiment_1_GridPoint {
14
15     private static Point createPoint(int x, int y) {
16         return new Point(x, y);
17     }
18
19     public static void main(String[] args) throws Exception {
20         ConfigReader configReader = new ConfigReader(new File("src/main/
resources/config.properties"));
21         configReader.setGridLength("400");
22         configReader.setGridWidth("400");
23         configReader.setModel("vonNeumann");
24         configReader.setGameType("Parity");
25         configReader.setTerminationType("ttl");
26         configReader.setTimeToLive("100");
27         configReader.setLoggingType("file");
28         configReader.setDatastructure("GridPoint");
29         Game game = new Game(configReader, "Parity_Experiment_1_GridPoint");
30         IGrid grid = game.getGird();
31         grid.setValue(createPoint(200, 200), true);
32         grid.setValue(createPoint(200, 201), true);
33         grid.setValue(createPoint(201, 200), true);
34         grid.setValue(createPoint(201, 201), true);
35         game.run();
36     }
37 }
38
```

```

1 package de.nordakademie.pdse.experiments.parity;
2
3 import de.nordakademie.pdse.config.ConfigReader;
4 import de.nordakademie.pdse.gamelogic.Game;
5 import de.nordakademie.pdse.grid.IGrid;
6
7 import java.awt.*;
8 import java.io.File;
9
10 /**
11  * @author Georg Mezlaw, Rane Petersen
12 */
13 public class Parity_Experiment_2_GridArray {
14     private static Point createPoint(int x, int y) {
15         return new Point(x, y);
16     }
17
18     public static void main(String[] args) throws Exception {
19         ConfigReader configReader = new ConfigReader(new File("src/main/
resources/config.properties"));
20         configReader.setGridLength("100");
21         configReader.setGridWidth("100");
22         configReader.setModel("vonNeumann");
23         configReader.setGameType("Parity");
24         configReader.setTerminationType("ttl");
25         configReader.setTimeToLive("100");
26         configReader.setLoggingType("file");
27         configReader.setDatastructure("GridArray");
28         Game game = new Game(configReader, "Parity_Experiment_2_GridArray");
29         IGrid grid = game.getGird();
30         for (int length = 0; length < configReader.getGridLength(); length++) {
31             for (int width = 0; width < configReader.getGridWidth(); width++) {
32                 if (width % 2 == 0) {
33                     if ((length % 2 == 1)) {
34                         grid.setValue(createPoint(length, width), false);
35                     } else {
36                         grid.setValue(createPoint(length, width), true);
37                     }
38                 } else {
39                     if ((length % 2 == 1)) {
40                         grid.setValue(createPoint(length, width), true);
41                     } else {
42                         grid.setValue(createPoint(length, width), false);
43                     }
44                 }
45             }
46         }
47         game.setGrid(grid);
48         game.run();
49     }
50 }
51

```

```

1 package de.nordakademie.pdse.experiments.parity;
2
3 import de.nordakademie.pdse.config.ConfigReader;
4 import de.nordakademie.pdse.gamelogic.Game;
5 import de.nordakademie.pdse.grid.IGrid;
6
7 import java.awt.*;
8 import java.io.File;
9
10 /**
11  * @author Georg Mezlaw, Rane Petersen
12 */
13 public class Parity_Experiment_2_GridPoint {
14     private static Point createPoint(int x, int y) {
15         return new Point(x, y);
16     }
17
18     public static void main(String[] args) throws Exception {
19         ConfigReader configReader = new ConfigReader(new File("src/main/
resources/config.properties"));
20         configReader.setGridLength("100");
21         configReader.setGridWidth("100");
22         configReader.setModel("vonNeumann");
23         configReader.setGameType("Parity");
24         configReader.setTerminationType("ttl");
25         configReader.setTimeToLive("100");
26         configReader.setLoggingType("file");
27         configReader.setDatastructure("GridPoints");
28         Game game = new Game(configReader, "Parity_Experiment_2_GridPoint");
29         IGrid grid = game.getGird();
30         for (int length = 0; length < configReader.getGridLength(); length++) {
31             for (int width = 0; width < configReader.getGridWidth(); width++) {
32                 if (width % 2 == 0) {
33                     if ((length % 2 == 1)) {
34                         grid.setValue(createPoint(length, width), false);
35                     } else {
36                         grid.setValue(createPoint(length, width), true);
37                     }
38                 } else {
39                     if ((length % 2 == 1)) {
40                         grid.setValue(createPoint(length, width), true);
41                     } else {
42                         grid.setValue(createPoint(length, width), false);
43                     }
44                 }
45             }
46         }
47         game.setGrid(grid);
48         game.run();
49     }
50 }
51

```

```

1 package de.nordakademie.pdse.experiments.gameoflife;
2
3 import de.nordakademie.pdse.config.ConfigReader;
4 import de.nordakademie.pdse.gamelogic.Game;
5 import de.nordakademie.pdse.grid.IGrid;
6
7 import java.awt.*;
8 import java.io.File;
9
10 /**
11  * @author Georg Mezlaw, Rane Petersen
12 */
13 public class GameOfLife_Experiment_1_GridArray {
14
15     private static Point createPoint(int x, int y) {
16         return new Point(x, y);
17     }
18
19     public static void main(String[] args) throws Exception {
20         ConfigReader configReader = new ConfigReader(new File("src/main/
resources/config.properties"));
21         configReader.setGridLength("41");
22         configReader.setGridWidth("40");
23         configReader.setModel("Moore");
24         configReader.setGameType("GameOfLife");
25         configReader.setTerminationType("ttl");
26         configReader.setTimeToLive("100");
27         configReader.setLoggingType("file");
28         configReader.setDatastructure("GridArray");
29         Game game = new Game(configReader, "GameOfLife_Experiment_1_GridArray");
30         IGrid grid = game.getGird();
31
32         grid.setValue(createPoint(18, 18), true);
33         grid.setValue(createPoint(18, 19), true);
34         grid.setValue(createPoint(18, 21), true);
35         grid.setValue(createPoint(18, 22), true);
36
37         grid.setValue(createPoint(19, 18), true);
38         grid.setValue(createPoint(19, 19), true);
39         grid.setValue(createPoint(19, 21), true);
40         grid.setValue(createPoint(19, 22), true);
41
42         grid.setValue(createPoint(20, 19), true);
43         grid.setValue(createPoint(20, 21), true);
44
45         grid.setValue(createPoint(21, 17), true);
46         grid.setValue(createPoint(21, 19), true);
47         grid.setValue(createPoint(21, 21), true);
48         grid.setValue(createPoint(21, 23), true);
49
50         grid.setValue(createPoint(22, 18), true);
51         grid.setValue(createPoint(22, 19), true);
52         grid.setValue(createPoint(22, 21), true);
53         grid.setValue(createPoint(22, 23), true);
54
55         grid.setValue(createPoint(23, 17), true);
56         grid.setValue(createPoint(23, 18), true);
57         grid.setValue(createPoint(23, 22), true);
58         grid.setValue(createPoint(23, 23), true);
59
60         game.setGrid(grid);
61         game.run();
62     }
63 }
```

```

1 package de.nordakademie.pdse.experiments.gameoflife;
2
3 import de.nordakademie.pdse.config.ConfigReader;
4 import de.nordakademie.pdse.gamelogic.Game;
5 import de.nordakademie.pdse.grid.IGrid;
6
7 import java.awt.*;
8 import java.io.File;
9
10 /**
11  * @author Georg Mezlaw, Rane Petersen
12 */
13 public class GameOfLife_Experiment_1_GridPoint {
14
15     private static Point createPoint(int x, int y) {
16         return new Point(x, y);
17     }
18
19     public static void main(String[] args) throws Exception {
20         ConfigReader configReader = new ConfigReader(new File("src/main/
21 resources/config.properties"));
22         configReader.setGridLength("41");
23         configReader.setGridWidth("40");
24         configReader.setModel("Moore");
25         configReader.setGameType("GameOfLife");
26         configReader.setTerminationType("ttl");
27         configReader.setTimeToLive("100");
28         configReader.setLoggingType("file");
29         configReader.setDatastructure("GridPoints");
30         Game game = new Game(configReader, "GameOfLife_Experiment_1_GridPoint");
31         IGrid grid = game.getGird();
32         grid.setValue(createPoint(18, 18), true);
33         grid.setValue(createPoint(18, 19), true);
34         grid.setValue(createPoint(18, 21), true);
35         grid.setValue(createPoint(18, 22), true);
36
37         grid.setValue(createPoint(19, 18), true);
38         grid.setValue(createPoint(19, 19), true);
39         grid.setValue(createPoint(19, 21), true);
40         grid.setValue(createPoint(19, 22), true);
41
42         grid.setValue(createPoint(20, 19), true);
43         grid.setValue(createPoint(20, 21), true);
44
45         grid.setValue(createPoint(21, 17), true);
46         grid.setValue(createPoint(21, 19), true);
47         grid.setValue(createPoint(21, 21), true);
48         grid.setValue(createPoint(21, 23), true);
49
50         grid.setValue(createPoint(22, 18), true);
51         grid.setValue(createPoint(22, 19), true);
52         grid.setValue(createPoint(22, 21), true);
53         grid.setValue(createPoint(22, 23), true);
54
55         grid.setValue(createPoint(23, 17), true);
56         grid.setValue(createPoint(23, 18), true);
57         grid.setValue(createPoint(23, 22), true);
58         grid.setValue(createPoint(23, 23), true);
59
60         game.setGrid(grid);
61         game.run();
62     }
63 }
```

```

1 package de.nordakademie.pdse.experiments.gameoflife;
2
3 import de.nordakademie.pdse.config.ConfigReader;
4 import de.nordakademie.pdse.gamelogic.Game;
5 import de.nordakademie.pdse.grid.IGrid;
6
7 import java.awt.*;
8 import java.io.File;
9
10 /**
11  * @author Georg Mezlaw, Rane Petersen
12 */
13 public class GameOfLife_Experiment_2_GridArray {
14
15     private static Point createPoint(int x, int y) {
16         return new Point(x, y);
17     }
18
19     public static void main(String[] args) throws Exception {
20         ConfigReader configReader = new ConfigReader(new File("src/main/
21 resources/config.properties"));
22         configReader.setGridLength("100");
23         configReader.setGridWidth("100");
24         configReader.setModel("Moore");
25         configReader.setGameType("GameOfLife");
26         configReader.setTerminationType("ttl");
27         configReader.setTimeToLive("100");
28         configReader.setLoggingType("file");
29         configReader.setDatastructure("GridArray");
30         Game game = new Game(configReader, "GameOfLife_Experiment_2_GridArray");
31         IGrid grid = game.getGird();
32         for (int length = 0; length < configReader.getGridLength(); length++) {
33             for (int width = 0; width < configReader.getGridWidth(); width++) {
34                 if (width % 2 == 0) {
35                     if ((length % 2 == 1)) {
36                         grid.setValue(createPoint(length, width), false);
37                     } else {
38                         grid.setValue(createPoint(length, width), true);
39                     }
40                 } else {
41                     if ((length % 2 == 1)) {
42                         grid.setValue(createPoint(length, width), true);
43                     } else {
44                         grid.setValue(createPoint(length, width), false);
45                     }
46                 }
47             }
48         game.setGrid(grid);
49         game.run();
50     }
51 }

```

```
1 package de.nordakademie.pdse.experiments.gameoflife;
2
3 import de.nordakademie.pdse.config.ConfigReader;
4 import de.nordakademie.pdse.gamelogic.Game;
5
6 import java.io.File;
7
8 /**
9  * @author Georg Mezlaw, Rane Petersen
10 */
11 public class GameOfLife_Experiment_3_GridArray {
12
13     public static void main(String[] args) throws Exception {
14         ConfigReader configReader = new ConfigReader(new File("src/main/
resources/config.properties"));
15         configReader.setGridLength("300");
16         configReader.setGridWidth("300");
17         configReader.setModel("Moore");
18         configReader.setGameType("GameOfLife");
19         configReader.setTerminationType("ttl");
20         configReader.setTimeToLive("100");
21         configReader.setLoggingType("file");
22         configReader.setDatastructure("GridArray");
23         Game game = new Game(configReader, "GameOfLife_Experiment_3_GridArray");
24         game.run();
25     }
26
27 }
28
```

```
1 package de.nordakademie.pdse.experiments.gameoflife;
2
3 import de.nordakademie.pdse.config.ConfigReader;
4 import de.nordakademie.pdse.gamelogic.Game;
5
6 import java.io.File;
7
8 /**
9  * @author Georg Mezlaw, Rane Petersen
10 */
11 public class GameOfLife_Experiment_3_GridPoint {
12
13     public static void main(String[] args) throws Exception {
14         ConfigReader configReader = new ConfigReader(new File("src/main/
resources/config.properties"));
15         configReader.setGridLength("300");
16         configReader.setGridWidth("300");
17         configReader.setModel("Moore");
18         configReader.setGameType("GameOfLife");
19         configReader.setTerminationType("ttl");
20         configReader.setTimeToLive("100");
21         configReader.setLoggingType("file");
22         configReader.setDatastructure("GridPoints");
23         Game game = new Game(configReader, "GameOfLife_Experiment_3_GridPoint");
24         game.run();
25     }
26
27 }
28
```

```

1 package de.nordakademie.pdse.experiments.gameoflife;
2
3 import de.nordakademie.pdse.config.ConfigReader;
4 import de.nordakademie.pdse.gamelogic.Game;
5 import de.nordakademie.pdse.grid.IGrid;
6
7 import java.awt.*;
8 import java.io.File;
9
10 /**
11  * @author Georg Mezlaw, Rane Petersen
12 */
13 public class GameOfLife_Experiment_2_GridPoints {
14
15     private static Point createPoint(int x, int y) {
16         return new Point(x, y);
17     }
18
19     public static void main(String[] args) throws Exception {
20         ConfigReader configReader = new ConfigReader(new File("src/main/
21 resources/config.properties"));
22         configReader.setGridLength("100");
23         configReader.setGridWidth("100");
24         configReader.setModel("Moore");
25         configReader.setGameType("GameOfLife");
26         configReader.setTerminationType("ttl");
27         configReader.setTimeToLive("100");
28         configReader.setLoggingType("file");
29         configReader.setDatastructure("GridPoint");
30         Game game = new Game(configReader, "GameOfLife_Experiment_2_GridPoints"
31 );
32         IGrid grid = game.getGird();
33         for (int length = 0; length < configReader.getGridLength(); length++) {
34             for (int width = 0; width < configReader.getGridWidth(); width++) {
35                 if (width % 2 == 0) {
36                     if ((length % 2 == 1)) {
37                         grid.setValue(createPoint(length, width), false);
38                     } else {
39                         grid.setValue(createPoint(length, width), true);
40                     }
41                 } else {
42                     if ((length % 2 == 1)) {
43                         grid.setValue(createPoint(length, width), true);
44                     } else {
45                         grid.setValue(createPoint(length, width), false);
46                     }
47                 }
48             }
49             game.setGrid(grid);
50         }
51     }

```

```
1 game.type=Parity
2 #game.type= "Parity", "GameOfLife"#
3 game.model=vonNeumann
4 #game.model= "Moore", "vonNeumann"#
5 game.dataStructure=GridPoints
6 #game.dataStructure= "GridPoints", "GridArray"#
7 game.loggingType=console
8 #game.loggingType= "console", "file", "consoleAndFile", "disable"#
9 game.timeToLive=10
10 game.grid.length=10
11 game.grid.width=10
12 game.terminationType=ttl
13 #game.terminationType= "ttl", "noChange", "ttlOrNoChange"
14
```

```
1 package de.nordakademie.pdse.grid;
2
3 import org.junit.Test;
4
5 import java.awt.*;
6
7 import static org.junit.Assert.assertEquals;
8
9 public class GridTest {
10
11     @Test
12     public void countMooreActiveNeighbors() {
13         GridPoints currentGrid = new GridPoints(4, 4);
14         currentGrid.changeValue(new Point(0, 1));
15         currentGrid.changeValue(new Point(1, 1));
16         assertEquals(2, currentGrid.countMooreActiveNeighbors(new Point(0, 0
    )));
17     }
18
19     @Test
20     public void countVonNeumannActiveNeighbors() {
21         GridPoints currentGrid = new GridPoints(4, 4);
22         currentGrid.changeValue(new Point(0, 1));
23         currentGrid.changeValue(new Point(1, 1));
24         assertEquals(1, currentGrid.countVonNeumannActiveNeighbors(new Point(0
    , 0)));
25     }
26 }
27
```

```
1 package de.nordakademie.pdse.grid;
2
3 import org.junit.Test;
4
5 import java.awt.*;
6
7 import static org.junit.Assert.assertNotEquals;
8
9 public class CopyGridTest {
10     @Test
11     public void testGetGrid() {
12         IGrid originalGrid = new GridPoints(5, 5);
13         CopyGrid copyGrid = new CopyGrid(originalGrid);
14         originalGrid.setValue(new Point(0, 0), true);
15         assertNotEquals(originalGrid.getValue(new Point(0, 0)), copyGrid.
    getGrid().getValue(new Point(0, 0)));
16     }
17 }
18
```

```

1 package de.nordakademie.pdse.grid;
2
3 import org.junit.Test;
4
5 import java.awt.*;
6 import java.util.ArrayList;
7 import java.util.Arrays;
8
9 import static org.junit.Assert.*;
10
11 public class GridArrayTest {
12
13     @Test
14     public void testCreateGrid() {
15         int[][] testGrid = {{0}};
16         assertEquals(Arrays.deepToString(testGrid), Arrays.deepToString(new
17             GridArray(1, 1).getGrid()));
18     }
19
19     @Test
20     public void testChangeValue() {
21         int[][] testGrid = {{1}};
22         GridArray currentGrid = new GridArray(1, 1);
23         currentGrid.changeValue(new Point(0, 0));
24         assertEquals(testGrid, currentGrid.getGrid());
25     }
26
27
28
29     @Test
30     public void testGetVonNeumannNeighbors() {
31         ArrayList<Point> expectedValues = new ArrayList<>();
32         expectedValues.add(new Point(0, 0));
33         expectedValues.add(new Point(1, 1));
34         expectedValues.add(new Point(2, 0));
35         GridArray currentGrid = new GridArray(4, 4);
36
37         assertTrue(expectedValues.containsAll(currentGrid.
38             getVonNeumannNeighbors(new Point(1, 0)))
39             && currentGrid.getVonNeumannNeighbors(new Point(1, 0)).size
40             () == expectedValues.size());
41     }
42
43     @Test
44     public void testGetMooreNeighbors() {
45         ArrayList<Point> expectedValues = new ArrayList<>();
46         expectedValues.add(new Point(0, 0));
47         expectedValues.add(new Point(0, 1));
48         expectedValues.add(new Point(1, 1));
49         expectedValues.add(new Point(2, 1));
50         expectedValues.add(new Point(2, 0));
51         GridArray currentGrid = new GridArray(4, 4);
52
53         assertTrue(expectedValues.containsAll(currentGrid.getMooreNeighbors(new
54             Point(1, 0)))
55             && currentGrid.getMooreNeighbors(new Point(1, 0)).size() ==
56             expectedValues.size());
57     }
58
59     @Test
60     public void testToString() {
61         GridArray currentGrid = new GridArray(4, 4);
62         String testString;

```

```
60     testString = "0000" + "\n" + "0000" + "\n" + "0000" + "\n" + "0000" +
61     "\n";
62     assertEquals(testString, currentGrid.toString());
63 }
64
64 @Test
65 public void testSetValue(){
66     ArrayList<Point> expectedValues = new ArrayList<>();
67     GridArray currentGrid = new GridArray(4, 4);
68     assertFalse(currentGrid.getValue(new Point(0,0)));
69     currentGrid.setValue(new Point(0,0), true);
70     assertTrue(currentGrid.getValue(new Point(0,0)));
71 }
72
73
74 }
75
```

```

1 package de.nordakademie.pdse.grid;
2
3 import org.junit.Test;
4
5 import java.awt.*;
6 import java.util.ArrayList;
7 import java.util.HashMap;
8
9 import static org.junit.Assert.*;
10
11 public class GridPointsTest {
12     @Test
13     public void testCreateGrid() {
14         HashMap<Point, Boolean> testGrid = new HashMap<>();
15         testGrid.put(new Point(0, 0), false);
16         assertEquals(testGrid, new GridPoints(1, 1).getGrid());
17     }
18
19     @Test
20     public void testChangeValue() {
21         HashMap<Point, Boolean> testGrid = new HashMap<>();
22         testGrid.put(new Point(0, 0), true);
23         GridPoints currentGrid = new GridPoints(1, 1);
24         currentGrid.changeValue(new Point(0, 0));
25         assertEquals(testGrid, currentGrid.getGrid());
26     }
27
28     @Test
29     public void testGetVonNeumannNeighbors() {
30         ArrayList<Point> expectedValues = new ArrayList<>();
31         expectedValues.add(new Point(0, 0));
32         expectedValues.add(new Point(1, 1));
33         expectedValues.add(new Point(2, 0));
34         GridPoints currentGrid = new GridPoints(4, 4);
35
36         assertTrue(expectedValues.containsAll(currentGrid.
37             getVonNeumannNeighbors(new Point(1, 0)))
38             && currentGrid.getVonNeumannNeighbors(new Point(1, 0)).size
39             () == expectedValues.size());
40     }
41
42     @Test
43     public void testGetMooreNeighbors() {
44         ArrayList<Point> expectedValues = new ArrayList<>();
45         expectedValues.add(new Point(0, 0));
46         expectedValues.add(new Point(0, 1));
47         expectedValues.add(new Point(1, 1));
48         expectedValues.add(new Point(2, 1));
49         expectedValues.add(new Point(2, 0));
50         GridPoints currentGrid = new GridPoints(4, 4);
51
52         assertTrue(expectedValues.containsAll(currentGrid.getMooreNeighbors(new
53             Point(1, 0)))
54             && currentGrid.getMooreNeighbors(new Point(1, 0)).size() ==
55             expectedValues.size());
56     }
57
58     @Test
59     public void testSetValue(){
60         ArrayList<Point> expectedValues = new ArrayList<>();
61         GridPoints currentGrid = new GridPoints(4, 4);
62         assertFalse(currentGrid.getValue(new Point(0,0)));
63         currentGrid.setValue(new Point(0,0), true);

```

```
61     assertTrue(currentGrid.getValue(new Point(0,0)));
62 }
63
64 @Test
65 public void testToString() {
66     GridPoints currentGrid = new GridPoints(4, 4);
67     String testString;
68     testString = "0000" + "\n" + "0000" + "\n" + "0000" + "\n" + "0000" +
69     "\n";
70     assertEquals(testString, currentGrid.toString());
71 }
72 }
73 }
```

```
1 package de.nordakademie.pdse.grid;
2
3 import de.nordakademie.pdse.config.ConfigReader;
4 import org.junit.Test;
5 import org.mockito.internal.matchers.apachecommons.ReflectionEquals;
6
7 import static org.junit.Assert.assertTrue;
8 import static org.mockito.Mockito.mock;
9 import static org.mockito.Mockito.when;
10
11 public class GridFactoryTest {
12
13     @Test
14     public void testGetGridAsGridArray() {
15         ConfigReader configReader = mock(ConfigReader.class);
16         when(configReader.getDatastructure()).thenReturn("GridArray");
17         GridFactory gridFactory = new GridFactory(4, 4, (configReader.
18             getDatastructure()));
18         IGrid currentGrid = new GridArray(4, 4);
19         assertTrue(new ReflectionEquals(currentGrid).matches(gridFactory.
20             getGrid()));
21     }
22
23     @Test
24     public void testGetGridAsGridPoints() {
25         ConfigReader configReader = mock(ConfigReader.class);
26         when(configReader.getDatastructure()).thenReturn("GridPoints");
27         GridFactory gridFactory = new GridFactory(4, 4, configReader.
28             getDatastructure());
28         IGrid currentGrid = new GridPoints(4, 4);
29         assertTrue(new ReflectionEquals(currentGrid).matches(gridFactory.
30             getGrid()));
31     }
31 }
```

```

1 package de.nordakademie.pdse.config;
2
3 import org.apache.commons.io.FileUtils;
4
5 import java.io.File;
6 import java.io.IOException;
7 import java.util.Properties;
8
9 import org.junit.Rule;
10 import org.junit.Test;
11 import org.junit.rules.TemporaryFolder;
12
13 import static org.junit.Assert.assertEquals;
14
15 /**
16 * Unit test vor getters and setters in the config class.
17 *
18 * @author Fabian Forthmann
19 */
20 public class ConfigReaderTest {
21
22     private final static String GAME_TYPE = "Parity";
23     private final static String MODEL = "Moore";
24     private final static String DATA_STRUCTURE = "GridPoints";
25     private final static String LOGGING_TYPE = "console";
26     private final static String TIME_TO_LIVE = "500";
27     private final static String GRID_LENGTH = "40";
28     private final static String GRID_WIDTH = "67";
29     private final static String TERMINATION_TYPE = "ttl";
30
31     private final static String ALTERNATIVE_GAME_TYPE = "GameOfLife";
32     private final static String ALTERNATIVE_MODEL = "vonNeumann";
33     private final static String ALTERNATIVE_DATA_STRUCTURE = "GridArray";
34     private final static String ALTERNATIVE_LOGGING_TYPE = "file";
35     private final static String ALTERNATIVE_TIME_TO_LIVE = "30";
36     private final static String ALTERNATIVE_GRID_LENGTH = "3";
37     private final static String ALTERNATIVE_GRID_WIDTH = "98";
38     private final static String ALTERNATIVE_TERMINATION_TYPE = "noChange";
39
40     @Rule
41     public TemporaryFolder tempFolder = new TemporaryFolder();
42
43     @Test
44     public void readExistingPropertiesFromFile() throws IOException {
45         Properties properties = initializeValidPropertiesFile();
46
47         final File tempFile = tempFolder.newFile("tempFile.properties");
48         // Writing String into the temporary file and formatting it
49         FileUtils.writeStringToFile(tempFile, properties.toString().replaceAll(
50             ",", "\n").replaceFirst("\\{", "").replaceAll("}", "").replaceAll(" ", ""));
50         ConfigReader configReader = new ConfigReader(tempFile);
51
52         assertEquals(GAME_TYPE, configReader.getGameType());
53         assertEquals(MODEL, configReader.getModel());
54         assertEquals(DATA_STRUCTURE, configReader.getDatastructure());
55         assertEquals(LOGGING_TYPE, configReader.getLoggingType());
56         assertEquals(Integer.parseInt(TIME_TO_LIVE), configReader.getTimeToLive
57             ());
57         assertEquals(Integer.parseInt(GRID_LENGTH), configReader.getGridLength
58             ());
58         assertEquals(Integer.parseInt(GRID_WIDTH), configReader.getGridWidth
59             ());
59         assertEquals(TERMINATION_TYPE, configReader.getTerminationType());
60     }

```

```

61
62     @Test
63     public void setPropertyValues() throws IOException {
64         Properties properties = initializeValidPropertiesFile();
65
66         final File tempFile = tempFolder.newFile("tempFile.properties");
67         // Writing String into the temporary file and formatting it
68         FileUtils.writeStringToFile(tempFile, properties.toString().replaceAll
69             (",", "\n").replaceFirst("\\{", "").replaceAll("}", "").replaceAll(" ", ""));
70         ConfigReader configReader = new ConfigReader(tempFile);
71
72         configReader.setGameType(ALTERNATIVE_GAME_TYPE);
73         configReader.setModel(ALTERNATIVE_MODEL);
74         configReader.setDatastructure(ALTERNATIVE_DATA_STRUCTURE);
75         configReader.setLoggingType(ALTERNATIVE_LOGGING_TYPE);
76         configReader.setTimeToLive(ALTERNATIVE_TIME_TO_LIVE);
77         configReader.setGridLength(ALTERNATIVE_GRID_LENGTH);
78         configReader.setGridWidth(ALTERNATIVE_GRID_WIDTH);
79         configReader.setTerminationType(ALTERNATIVE_TERMINATION_TYPE);
80
81         assertEquals(ALTERNATIVE_GAME_TYPE, configReader.getGameType());
82         assertEquals(ALTERNATIVE_MODEL, configReader.getModel());
83         assertEquals(ALTERNATIVE_DATA_STRUCTURE, configReader.getDatastructure
84             ());
85         assertEquals(ALTERNATIVE_LOGGING_TYPE, configReader.getLoggingType());
86         assertEquals(Integer.parseInt(ALTERNATIVE_TIME_TO_LIVE), configReader.
87             getTimeToLive());
87         assertEquals(Integer.parseInt(ALTERNATIVE_GRID_LENGTH), configReader.
88             getGridLength());
89         assertEquals(Integer.parseInt(ALTERNATIVE_GRID_WIDTH), configReader.
90             getGridWidth());
91         assertEquals(ALTERNATIVE_TERMINATION_TYPE, configReader.
92             getTerminationType());
93     }
94
95     @Test
96     public void changeExistingPropertyValues() throws IOException {
97         Properties properties = initializeValidPropertiesFile();
98
99         final File tempFile = tempFolder.newFile("tempFile.properties");
100        // Writing String into the temporary file and formatting it
101        FileUtils.writeStringToFile(tempFile, properties.toString().replaceAll
102            (",", "\n").replaceFirst("\\{", "").replaceAll("}", "").replaceAll(" ", ""));
103        ConfigReader configReader = new ConfigReader(tempFile);
104
105        configReader.setGameType(ALTERNATIVE_GAME_TYPE);
106        configReader.setModel(ALTERNATIVE_MODEL);
107        configReader.setDatastructure(ALTERNATIVE_DATA_STRUCTURE);
108        configReader.setLoggingType(ALTERNATIVE_LOGGING_TYPE);
109        configReader.setTimeToLive(ALTERNATIVE_TIME_TO_LIVE);
110        configReader.setGridLength(ALTERNATIVE_GRID_LENGTH);
111        configReader.setGridWidth(ALTERNATIVE_GRID_WIDTH);
112        configReader.setTerminationType(ALTERNATIVE_TERMINATION_TYPE);
113
114        assertEquals(ALTERNATIVE_GAME_TYPE, configReader.getGameType());
115        assertEquals(ALTERNATIVE_MODEL, configReader.getModel());
116        assertEquals(ALTERNATIVE_DATA_STRUCTURE, configReader.getDatastructure
117             ());
118        assertEquals(ALTERNATIVE_LOGGING_TYPE, configReader.getLoggingType());
119        assertEquals(Integer.parseInt(ALTERNATIVE_TIME_TO_LIVE), configReader.
120             getTimeToLive());
121        assertEquals(Integer.parseInt(ALTERNATIVE_GRID_LENGTH), configReader.
122             getGridLength());
123        assertEquals(Integer.parseInt(ALTERNATIVE_GRID_WIDTH), configReader.

```

```
114 getGridWidth());
115     assertEquals(ALTERNATIVE_TERMINATION_TYPE, configReader.
116         getTerminationType());
117 }
118 @Test(expected = RuntimeException.class)
119 public void readNotExistingConfigFile() {
120     File configPropertyFile = new File("fileThatDoesNotExist.properties");
121     ConfigReader configReader = new ConfigReader(configPropertyFile);
122     configReader.getGameType();
123 }
124
125 private Properties initializeValidPropertiesFile() {
126     Properties properties = new Properties();
127
128     properties.setProperty("game.type", GAME_TYPE);
129     properties.setProperty("game.model", MODEL);
130     properties.setProperty("game.dataStructure", DATA_STRUCTURE);
131     properties.setProperty("game.loggingType", LOGGING_TYPE);
132     properties.setProperty("game.timeToLive", TIME_TO_LIVE);
133     properties.setProperty("game.grid.length", GRID_LENGTH);
134     properties.setProperty("game.grid.width", GRID_WIDTH);
135     properties.setProperty("game.terminationType", TERMINATION_TYPE);
136
137     return properties;
138 }
139 }
140 }
```

```

1 package de.nordakademie.pdse.config;
2
3 import org.junit.Test;
4
5 import java.util.Properties;
6
7 /**
8 * Unit tests for the validation of the given property parameters.
9 *
10 * @author Fabian Forthmann
11 */
12 public class ConfigValidatorTest {
13
14     private final static String GAME_TYPE = "game.type";
15     private final static String GAME_MODEL = "game.model";
16     private final static String DATA_STRUCTURE = "game.dataStructure";
17     private final static String LOGGING_TYPE = "game.loggingType";
18     private final static String TIME_TO_LIVE = "game.timeToLive";
19     private final static String LENGTH = "game.grid.length";
20     private final static String WIDTH = "game.grid.width";
21     private final static String TERMINATION_TYPE = "game.terminationType";
22
23     @Test
24     public void validateValidPropertyConfig() {
25         Properties prop = new Properties();
26         prop.setProperty(GAME_TYPE, "GameOfLife");
27         prop.setProperty(GAME_MODEL, "vonNeumann");
28         prop.setProperty(DATA_STRUCTURE, "GridArray");
29         prop.setProperty(LOGGING_TYPE, "disable");
30         prop.setProperty(LENGTH, "10");
31         prop.setProperty(WIDTH, "10");
32         prop.setProperty(TIME_TO_LIVE, "589");
33         prop.setProperty(TERMINATION_TYPE, "ttlOrNoChange");
34
35         PropertyValidator propertyValidator = new PropertyValidator();
36         propertyValidator.validateConfig(prop);
37     }
38
39     @Test
40     public void validateValidPropertyConfigWithTtl() {
41         Properties prop = new Properties();
42         prop.setProperty(GAME_TYPE, "GameOfLife");
43         prop.setProperty(GAME_MODEL, "vonNeumann");
44         prop.setProperty(DATA_STRUCTURE, "GridArray");
45         prop.setProperty(LOGGING_TYPE, "disable");
46         prop.setProperty(LENGTH, "10");
47         prop.setProperty(WIDTH, "10");
48         prop.setProperty(TERMINATION_TYPE, "ttl");
49         prop.setProperty(TIME_TO_LIVE, "99");
50
51         PropertyValidator propertyValidator = new PropertyValidator();
52         propertyValidator.validateConfig(prop);
53     }
54
55     @Test(expected = RuntimeException.class)
56     public void validateInvalidPropertyConfigWithFalseStrings() {
57         Properties prop = new Properties();
58         prop.setProperty(GAME_TYPE, "GameOfLifeA");
59         prop.setProperty(GAME_MODEL, "vonNeumannB");
60         prop.setProperty(DATA_STRUCTURE, "GridArrayC");
61         prop.setProperty(LOGGING_TYPE, "disableD");
62         prop.setProperty(LENGTH, "10");
63         prop.setProperty(WIDTH, "10");
64         prop.setProperty(TIME_TO_LIVE, "92");

```

```

65     prop.setProperty(TERMINATION_TYPE, "ttlOrNoChangeE");
66
67     PropertyValidator propertyValidator = new PropertyValidator();
68     propertyValidator.validateConfig(prop);
69 }
70
71 @Test(expected = RuntimeException.class)
72 public void validateInvalidPropertyConfigWithFalseIntegers() {
73     Properties prop = new Properties();
74     prop.setProperty(GAME_TYPE, "GameOfLife");
75     prop.setProperty(GAME_MODEL, "vonNeumann");
76     prop.setProperty(DATA_STRUCTURE, "GridArray");
77     prop.setProperty(LOGGING_TYPE, "disable");
78     prop.setProperty(LENGTH, "0");
79     prop.setProperty(WIDTH, "-10");
80     prop.setProperty(TIME_TO_LIVE, "-987");
81     prop.setProperty(TERMINATION_TYPE, "ttlOrNoChange");
82
83     PropertyValidator propertyValidator = new PropertyValidator();
84     propertyValidator.validateConfig(prop);
85 }
86
87 @Test(expected = RuntimeException.class)
88 public void validateInvalidPropertyConfigWithInvalidTtl() {
89     Properties prop = new Properties();
90     prop.setProperty(GAME_TYPE, "GameOfLife");
91     prop.setProperty(GAME_MODEL, "vonNeumann");
92     prop.setProperty(DATA_STRUCTURE, "GridArray");
93     prop.setProperty(LOGGING_TYPE, "disable");
94     prop.setProperty(LENGTH, "10");
95     prop.setProperty(WIDTH, "10");
96     prop.setProperty(TERMINATION_TYPE, "ttl");
97     prop.setProperty(TIME_TO_LIVE, "-54");
98
99     PropertyValidator propertyValidator = new PropertyValidator();
100    propertyValidator.validateConfig(prop);
101 }
102
103 @Test(expected = RuntimeException.class)
104 public void validateInvalidPropertyConfigWithMissingFields() {
105     Properties prop = new Properties();
106     prop.setProperty(GAME_TYPE, "GameOfLife");
107     prop.setProperty(DATA_STRUCTURE, "GridArray");
108     prop.setProperty(LOGGING_TYPE, "disable");
109     prop.setProperty(WIDTH, "10");
110     prop.setProperty(TERMINATION_TYPE, "noChange");
111
112     PropertyValidator propertyValidator = new PropertyValidator();
113     propertyValidator.validateConfig(prop);
114 }
115 }
116

```

```

1 package de.nordakademie.pdse.logging;
2
3 import de.nordakademie.pdse.grid.GridFactory;
4 import org.junit.After;
5 import org.junit.Before;
6 import org.junit.Test;
7
8 import java.io.*;
9
10 import static org.junit.Assert.assertArrayEquals;
11 import static org.junit.Assert.assertTrue;
12
13 public class LoggerTest {
14
15     private final PrintStream standardOut = System.out;
16     private final ByteArrayOutputStream outputStreamCaptor = new
17     ByteArrayOutputStream();
18
19     @Before
20     public void setUp() {
21         System.setOut(new PrintStream(outputStreamCaptor));
22     }
23
24     @After
25     public void after() {
26         outputStreamCaptor.reset();
27     }
28
29     @Test
30     public void testAddGridToFile() {
31         String expectedOutput = "###2\n" +
32             "0000\n" +
33             "0000\n" +
34             "0000\n" +
35             "0000\n" + "\n\n";
36         Logger log = new Logger("file");
37         log.addGridToLog(new GridFactory(4, 4, "GridArray").getGrid().toString
38             (), 2);
39         StringBuilder current = new StringBuilder();
40         String line;
41         try {
42             BufferedReader reader = new BufferedReader(new FileReader(log.
43                 getFile()));
44             while ((line = reader.readLine()) != null) {
45                 current.append(line).append("\n");
46             }
47         } catch (IOException e) {
48             e.printStackTrace();
49         }
50         current.append("\n");
51         assertEquals(expectedOutput.toCharArray(), current.toString().
52             toCharArray());
53     }
54
55     @Test
56     public void testAddGridToLogConsole() {
57         String expectedOutput = "###2\n" +
58             "0000\n" +
59             "0000\n" +
60             "0000\n" + "\r\n";
61         System.setOut(new PrintStream(outputStreamCaptor));

```

```
61     Logger log = new Logger("console");
62     log.addGridToLog(new GridFactory(4, 4, "GridArray").getGrid().toString
63     (), 2);
64     String current = outputStreamCaptor.toString();
65     assertEquals(expectedOutput.toCharArray(), current.toCharArray());
66 }
67
68 @Test
69 public void testCustomConstructor() {
70     Logger log = new Logger("file", "experiment1");
71     assertTrue(log.getFile().exists());
72 }
73
74 }
```

```

1 package de.nordakademie.pdse.logging;
2
3 import de.nordakademie.pdse.grid.GridFactory;
4 import org.junit.After;
5 import org.junit.Before;
6 import org.junit.Test;
7
8 import java.io.*;
9
10 import static org.junit.Assert.assertArrayEquals;
11
12 public class TestConsoleAndFileOutput {
13
14     private final PrintStream standardOut = System.out;
15     private final ByteArrayOutputStream outputStreamCaptor = new
16     ByteArrayOutputStream();
17
18     @After
19     public void after() {
20         outputStreamCaptor.reset();
21     }
22
23     @Before
24     public void setUp() {
25         System.setOut(new PrintStream(outputStreamCaptor));
26     }
27
28     @Test
29     public void testAddGridToLogToConsoleAndToFile() {
30         String expectedConsoleOutput = "###2\n" +
31             "0000\n" +
32             "0000\n" +
33             "0000\n" +
34             "0000\n" + "\r\n";
35         String expectedFileOutput = "###2\n" +
36             "0000\n" +
37             "0000\n" +
38             "0000\n" +
39             "0000\n" + "\n\n";
40         System.setOut(new PrintStream(outputStreamCaptor));
41         Logger log = new Logger("consoleAndFile");
42         log.addGridToLog(new GridFactory(4, 4, "GridArray").getGrid().toString
43             (), 2);
44         String currentConsole = outputStreamCaptor.toString();
45         StringBuilder currentFile = new StringBuilder();
46         String line;
47         try {
48             BufferedReader reader = new BufferedReader(new FileReader(log.
49             getFile()));
50             while ((line = reader.readLine()) != null) {
51                 currentFile.append(line).append("\n");
52             }
53         } catch (IOException e) {
54             e.printStackTrace();
55         }
56         currentFile.append("\n");
57         assertArrayEquals(expectedFileOutput.toCharArray(), currentFile.
58             toString().toCharArray());
59         assertArrayEquals(expectedConsoleOutput.toCharArray(), currentConsole.
60             toCharArray());
61     }
62 }

```

```

1 package de.nordakademie.pdse.gamelogic;
2
3 import de.nordakademie.pdse.config.ConfigReader;
4 import de.nordakademie.pdse.grid.GridFactory;
5 import de.nordakademie.pdse.grid.IGrid;
6 import org.junit.After;
7 import org.junit.Before;
8 import org.junit.Test;
9 import org.mockito.Mock;
10
11 import java.awt.*;
12 import java.io.ByteArrayOutputStream;
13 import java.io.File;
14 import java.io.PrintStream;
15
16 import static org.junit.Assert.*;
17 import static org.mockito.Mockito.mock;
18 import static org.mockito.Mockito.when;
19
20 /**
21 * @author Rane Petersen
22 */
23
24 public class GameTest {
25
26     private final ByteArrayOutputStream outputStreamCaptor = new
27     ByteArrayOutputStream();
28
29     @After
30     public void after() {
31         outputStreamCaptor.reset();
32     }
33
34     @Before
35     public void setUp() {
36         System.setOut(new PrintStream(outputStreamCaptor));
37     }
38
39     @Mock
40     ConfigReader configReader = mock(ConfigReader.class);
41
42     @Mock
43     IGrid grid = mock(IGrid.class);
44
45     @Mock
46     IGrid grid2 = mock(IGrid.class);
47
48     @Test
49     public void getGird() {
50         when(configReader.getTerminationType()).thenReturn("noChange");
51         when(configReader.getLoggingType()).thenReturn("console");
52         when(configReader.getDatastructure()).thenReturn("GridArray");
53         when(configReader.getGameType()).thenReturn("Parity");
54         when(configReader.getTimeToLive()).thenReturn(2);
55         GridFactory gridFactory = new GridFactory(configReader.getGridLength
56             (), configReader.getGridWidth(), configReader.getDatastructure());
57         grid = gridFactory.getGrid();
58         Game game = new Game(configReader);
59         game.setGrid(grid);
60         assertEquals(grid, game.getGrid());
61     }
62     @Test
63     public void setGrid() {

```

```

63     when(configReader.getTerminationType()).thenReturn("noChange");
64     when(configReader.getLoggingType()).thenReturn("console");
65     when(configReader.getDatastructure()).thenReturn("GridArray");
66     when(configReader.getGameType()).thenReturn("Parity");
67     when(configReader.getTimeToLive()).thenReturn(2);
68     GridFactory gridFactory = new GridFactory(configReader.getGridLength
69         (), configReader.getGridWidth(), configReader.getDatastructure());
70     grid = gridFactory.getGrid();
71     Game game = new Game(configReader);
72     game.setGrid(grid);
73     assertEquals(grid, game.getGird());
74     game.setGrid(grid2);
75     assertEquals(grid2, game.getGird());
76     assertNotEquals(grid, game.getGird());
77 }
78
79 @Test
80 public void run() throws Exception {
81     System.setOut(new PrintStream(outputStreamCaptor));
82     GridFactory gridFactory = new GridFactory(3, 3, "GridArray");
83     IGrid grid = gridFactory.getGrid();
84     when(configReader.getLoggingType()).thenReturn("console");
85     when(configReader.getDatastructure()).thenReturn("GridArray");
86     when(configReader.getGameType()).thenReturn("Parity");
87     when(configReader.getModel()).thenReturn("vonNeumann");
88     when(configReader.getTerminationType()).thenReturn("ttl");
89     when(configReader.getGridLength()).thenReturn(3);
90     when(configReader.getGridWidth()).thenReturn(3);
91     when(configReader.getTimeToLive()).thenReturn(2);
92     grid.setValue(new Point(1, 1), true);
93     Game game = new Game(configReader);
94     game.setGrid(grid);
95     game.run();
96     IGrid checkGrid = game.getGird();
97     for (int width = 0; width < 3; width++) {
98         for (int length = 0; length < 3; length++) {
99             assertFalse(checkGrid.getValue(new Point(length, width)));
100     }
101     String expectedOutput = "###0\n" +
102         "000\n" +
103         "010\n" +
104         "000\n" +
105         "\r\n" +
106         "###1\n" +
107         "010\n" +
108         "101\n" +
109         "010\n" +
110         "\r\n" +
111         "##2\n" +
112         "000\n" +
113         "000\n" +
114         "000\n" +
115         "\r\n";
116     assertArrayEquals(expectedOutput.toCharArray(), outputStreamCaptor.
117         toString().toCharArray());
118 }
119
120 @Test
121 public void secondaryConstructor() throws Exception {
122     GridFactory gridFactory = new GridFactory(3, 3, "GridArray");
123     IGrid grid = gridFactory.getGrid();
124     String fileName = "testSecondaryConstructor";
125     assertFalse(new File(fileName + ".log").exists());

```

```
125     when(configReader.getLoggingType()).thenReturn("file");
126     when(configReader.getDatastructure()).thenReturn("GridArray");
127     when(configReader.getGameType()).thenReturn("Parity");
128     when(configReader.getModel()).thenReturn("vonNeumann");
129     when(configReader.getTerminationType()).thenReturn("ttl");
130     when(configReader.getGridLength()).thenReturn(3);
131     when(configReader.getGridWidth()).thenReturn(3);
132     when(configReader.getTimeToLive()).thenReturn(0);
133     grid.setValue(new Point(1, 1), true);
134     Game game = new Game(configReader, fileName);
135     assertTrue(new File(fileName + ".log").exists());
136     game.setGrid(grid);
137     game.run();
138     assertTrue(new File(fileName + ".log").exists());
139     new File(fileName + ".log").delete();
140     assertFalse(new File(fileName + ".log").exists());
141 }
142 }
143
```

```

1 package de.nordakademie.pdse.gamelogic;
2
3 import de.nordakademie.pdse.config.ConfigReader;
4 import de.nordakademie.pdse.grid.GridFactory;
5 import de.nordakademie.pdse.grid.IGrid;
6 import org.junit.Test;
7 import org.mockito.Mockito;
8
9 import java.awt.*;
10
11 import static org.junit.Assert.assertFalse;
12 import static org.junit.Assert.assertTrue;
13 import static org.mockito.Mockito.mock;
14 import static org.mockito.Mockito.when;
15
16 /**
17 * @author Rane Petersen
18 */
19
20 public class ParityTestSetAlive {
21
22     GridFactory gridFactory = new GridFactory(3, 3, "GridArray");
23     IGrid grid = gridFactory.getGrid();
24
25     @Mock
26     ConfigReader configReader = mock(ConfigReader.class);
27
28     private static Point createPoint(int x, int y) {
29         return new Point(x, y);
30     }
31
32     @Test
33     public void trueValueToFalseThroughStepsInGrid() throws Exception {
34         grid.setValue(createPoint(1, 1), true);
35         when(configReader.getGridLength()).thenReturn(3);
36         when(configReader.getGridWidth()).thenReturn(3);
37         when(configReader.getModel()).thenReturn("vonNeumann");
38         Parity parity = new Parity(configReader);
39         IGrid stepGrid = parity.step(grid);
40         stepGrid.getValue(createPoint(0, 2));
41         assertTrue(stepGrid.getValue(createPoint(0, 1)));
42         assertFalse(stepGrid.getValue(createPoint(0, 2)));
43         assertTrue(stepGrid.getValue(createPoint(1, 0)));
44         assertFalse(stepGrid.getValue(createPoint(1, 1)));
45         assertTrue(stepGrid.getValue(createPoint(1, 2)));
46         assertFalse(stepGrid.getValue(createPoint(2, 0)));
47         assertTrue(stepGrid.getValue(createPoint(2, 1)));
48         assertFalse(stepGrid.getValue(createPoint(2, 2)));
49     }
50 }
51

```

```
1 package de.nordakademie.pdse.gamelogic;
2
3 import de.nordakademie.pdse.config.ConfigReader;
4 import org.junit.Test;
5 import org.mockito.Mock;
6
7 import static org.junit.Assert.assertEquals;
8 import static org.mockito.Mockito.mock;
9 import static org.mockito.Mockito.when;
10
11 /**
12 * @author Rane Petersen
13 */
14
15 public class GameTypeFactoryTest {
16
17     @Mock
18     ConfigReader configReader = mock(ConfigReader.class);
19
20     @Test
21     public void returnGameOfLife() {
22         when(configReader.getGameType()).thenReturn("GameOfLife");
23         IGameType gameType = new GameTypeFactory(configReader).getGameType();
24         assertEquals(gameType.getClass(), GameOfLife.class);
25     }
26
27     @Test
28     public void returnParity() {
29         when(configReader.getGameType()).thenReturn("Parity");
30         IGameType gameType = new GameTypeFactory(configReader).getGameType();
31         assertEquals(gameType.getClass(), Parity.class);
32     }
33 }
34 }
35 }
```

```

1 package de.nordakademie.pdse.gamelogic;
2
3 import de.nordakademie.pdse.config.ConfigReader;
4 import de.nordakademie.pdse.grid.GridFactory;
5 import de.nordakademie.pdse.grid.IGrid;
6 import org.junit.After;
7 import org.junit.Before;
8 import org.junit.Test;
9 import org.mockito.Mock;
10
11 import java.awt.*;
12 import java.io.ByteArrayOutputStream;
13 import java.io.PrintStream;
14
15 import static org.junit.Assert.*;
16 import static org.mockito.Mockito.mock;
17 import static org.mockito.Mockito.when;
18
19 public class GameTestLoggerAndRun {
20
21     private final ByteArrayOutputStream outputStreamCaptor = new
22     ByteArrayOutputStream();
23
24     IGrid grid;
25
26     @Mock
27     ConfigReader configReader = mock(ConfigReader.class);
28
29     @After
30     public void after() {
31         outputStreamCaptor.reset();
32     }
33
34     @Before
35     public void setUp() {
36         System.setOut(new PrintStream(outputStreamCaptor));
37     }
38
39     @Test
40     public void noChange() throws Exception {
41         System.setOut(new PrintStream(outputStreamCaptor));
42         when(configReader.getTerminationType()).thenReturn("noChange");
43         when(configReader.getGridLength()).thenReturn(3);
44         when(configReader.getGridWidth()).thenReturn(3);
45         when(configReader.getLoggingType()).thenReturn("console");
46         when(configReader.getDatastructure()).thenReturn("GridArray");
47         when(configReader.getGameType()).thenReturn("Parity");
48         when(configReader.getModel()).thenReturn("vonNeumann");
49         when(configReader.getTimeToLive()).thenReturn(10);
50         GridFactory gridFactory = new GridFactory(configReader.getGridLength
51             (), configReader.getGridWidth(), configReader.getDatastructure());
52         grid = gridFactory.getGrid();
53         grid.setValue(new Point(1, 1), true);
54         Game game = new Game(configReader);
55         game.setGrid(grid);
56         game.run();
57         String expectedOutput = "###0\n" +
58             "000\n" +
59             "010\n" +
60             "000\n" +
61             "\r\n" +
62             "###1\n" +
63             "010\n" +
64             "101\n" +

```

```

63             "010\n" +
64             "\r\n" +
65             "###2\n" +
66             "000\n" +
67             "000\n" +
68             "000\n" +
69             "\r\n";
70         assertEquals(expectedOutput.toCharArray(), outputStreamCaptor.
71             toString().toCharArray());
72     }
73
74     @Test
75     public void ttlZero() throws Exception {
76         System.setOut(new PrintStream(outputStreamCaptor));
77         when(configReader.getTerminationType()).thenReturn("ttl");
78         when(configReader.getGridLength()).thenReturn(3);
79         when(configReader.getGridWidth()).thenReturn(3);
80         when(configReader.getLoggingType()).thenReturn("console");
81         when(configReader.getDatastructure()).thenReturn("GridArray");
82         when(configReader.getGameType()).thenReturn("Parity");
83         when(configReader.getModel()).thenReturn("vonNeumann");
84         when(configReader.getTimeToLive()).thenReturn(0);
85         GridFactory gridFactory = new GridFactory(configReader.getGridLength
86             (), configReader.getGridWidth(), configReader.getDatastructure());
86         grid = gridFactory.getGrid();
87         grid.setValue(new Point(1, 1), true);
88         Game game = new Game(configReader);
89         game.setGrid(grid);
90         game.run();
91         assertTrue(game.getGird().getValue(new Point(1, 1)));
92         String expectedOutput = "###0\n" +
93             "000\n" +
94             "010\n" +
95             "000\n" +
96             "\r\n";
97         assertEquals(expectedOutput.toCharArray(), outputStreamCaptor.
98             toString().toCharArray());
99     }
100
101    @Test
102    public void ttlOrNoChangeBreakttl() throws Exception {
103        when(configReader.getTerminationType()).thenReturn("ttlOrNoChange");
104        when(configReader.getGridLength()).thenReturn(3);
105        when(configReader.getGridWidth()).thenReturn(3);
106        when(configReader.getLoggingType()).thenReturn("console");
107        when(configReader.getDatastructure()).thenReturn("GridArray");
108        when(configReader.getGameType()).thenReturn("Parity");
109        when(configReader.getModel()).thenReturn("vonNeumann");
110        when(configReader.getTimeToLive()).thenReturn(1);
111        GridFactory gridFactory = new GridFactory(configReader.getGridLength
112             (), configReader.getGridWidth(), configReader.getDatastructure());
113        grid = gridFactory.getGrid();
114        grid.setValue(new Point(1, 1), true);
115        Game game = new Game(configReader);
116        game.setGrid(grid);
117        game.run();
118    }
119
120    @Test
121    public void ttlOrNoChangeBreakNoChange() throws Exception {
122        System.setOut(new PrintStream(outputStreamCaptor));
123        when(configReader.getTerminationType()).thenReturn("ttlOrNoChange");
124        when(configReader.getGridLength()).thenReturn(3);
125        when(configReader.getGridWidth()).thenReturn(3);

```

```
123     when(configReader.getLoggingType()).thenReturn("console");
124     when(configReader.getDatastructure()).thenReturn("GridArray");
125     when(configReader.getGameType()).thenReturn("Parity");
126     when(configReader.getModel()).thenReturn("vonNeumann");
127     when(configReader.getTimeToLive()).thenReturn(10);
128     GridFactory gridFactory = new GridFactory(configReader.getGridLength
129         (), configReader.getGridWidth(), configReader.getDatastructure());
130     grid = gridFactory.getGrid();
131     grid.setValue(new Point(1, 1), true);
132     Game game = new Game(configReader);
133     game.setGrid(grid);
134     game.run();
135     String expectedOutput = "###0\n" +
136         "000\n" +
137         "010\n" +
138         "000\n" +
139         "\r\n" +
140         "##1\n" +
141         "010\n" +
142         "101\n" +
143         "010\n" +
144         "\r\n" +
145         "##2\n" +
146         "000\n" +
147         "000\n" +
148         "\r\n";
149     assertEquals(expectedOutput.toCharArray(), outputStreamCaptor.
150         toString().toCharArray());
151 }
152 }
```

```

1 package de.nordakademie.pdse.gamelogic;
2
3 import de.nordakademie.pdse.config.ConfigReader;
4 import de.nordakademie.pdse.grid.GridFactory;
5 import de.nordakademie.pdse.grid.IGrid;
6 import org.junit.Test;
7 import org.mockito.Mock;
8
9 import java.awt.*;
10
11 import static org.junit.Assert.assertFalse;
12 import static org.junit.Assert.assertTrue;
13 import static org.mockito.Mockito.mock;
14 import static org.mockito.Mockito.when;
15
16 /**
17 * @author Rane Petersen
18 */
19
20 public class GameOfLifeTestSetAlive {
21
22     GridFactory gridFactory = new GridFactory(3, 3, "GridArray");
23     IGrid grid = gridFactory.getGrid();
24
25     @Mock
26     ConfigReader configReader = mock(ConfigReader.class);
27
28     @Test
29     public void trueValueToFalseThroughStep(){
30         grid.setValue(new Point(1, 1), true);
31         when(configReader.getGridLength()).thenReturn(3);
32         when(configReader.getGridWidth()).thenReturn(3);
33         when(configReader.getModel()).thenReturn("Moore");
34
35         GameOfLife gameOfLife = new GameOfLife(configReader);
36         IGrid stepGrid = gameOfLife.step(grid);
37         assertFalse(stepGrid.getValue(new Point(1,1)));
38     }
39
40     @Test
41     public void falseValueToTrueThroughStep() {
42         grid.setValue(new Point(0, 0), true);
43         grid.setValue(new Point(0, 1), true);
44         grid.setValue(new Point(0, 2), true);
45         when(configReader.getGridLength()).thenReturn(3);
46         when(configReader.getGridWidth()).thenReturn(3);
47         when(configReader.getModel()).thenReturn("Moore");
48
49         GameOfLife gameOfLife = new GameOfLife(configReader);
50         IGrid stepGrid = gameOfLife.step(grid);
51         assertTrue(stepGrid.getValue(new Point(0,1)));
52         assertTrue(stepGrid.getValue(new Point(1,1)));
53         assertFalse(stepGrid.getValue(new Point(0,0)));
54         assertFalse(stepGrid.getValue(new Point(0,2)));
55         assertFalse(stepGrid.getValue(new Point(1,0)));
56         assertFalse(stepGrid.getValue(new Point(1,2)));
57     }
58 }
59

```

```

1 package de.nordakademie.pdse.gamelogic;
2
3 import de.nordakademie.pdse.config.ConfigReader;
4 import de.nordakademie.pdse.grid.GridFactory;
5 import de.nordakademie.pdse.grid.IGrid;
6 import org.junit.Test;
7 import org.mockito.Mock;
8
9 import java.awt.*;
10
11 import static org.junit.Assert.assertFalse;
12 import static org.junit.Assert.assertTrue;
13 import static org.mockito.Mockito.mock;
14 import static org.mockito.Mockito.when;
15
16 /**
17 * @author Rane Petersen, Fabian Forthmann
18 */
19
20 public class ParityTestOneByOneGrid {
21
22     GridFactory gridFactory = new GridFactory(1, 1, "GridArray");
23     IGrid grid = gridFactory.getGrid();
24     Point point = new Point(0, 0);
25
26     @Mock
27     ConfigReader configReader = mock(ConfigReader.class);
28
29     @Test
30     public void emptyParityGridVonNeumann() {
31         grid.setValue(point, false);
32         when(configReader.getGridLength()).thenReturn(1);
33         when(configReader.getGridWidth()).thenReturn(1);
34         when(configReader.getModel()).thenReturn("vonNeumann");
35         Parity parity = new Parity(configReader);
36         assertFalse(parity.step(grid).getValue(point));
37     }
38
39     @Test
40     public void trueParityGridVonNeumann() {
41         grid.setValue(point, true);
42         when(configReader.getGridLength()).thenReturn(1);
43         when(configReader.getGridWidth()).thenReturn(1);
44         when(configReader.getModel()).thenReturn("vonNeumann");
45         Parity parity = new Parity(configReader);
46         assertFalse(parity.step(grid).getValue(point));
47     }
48 }
49

```

```

1 package de.nordakademie.pdse.gamelogic;
2
3 import de.nordakademie.pdse.config.ConfigReader;
4 import de.nordakademie.pdse.grid.GridFactory;
5 import de.nordakademie.pdse.grid.IGrid;
6 import org.junit.Test;
7 import org.mockito.Mockito;
8
9 import java.awt.*;
10
11 import static org.junit.Assert.assertFalse;
12 import static org.mockito.Mockito.mock;
13 import static org.mockito.Mockito.when;
14
15 /**
16 * @author Rane Petersen, Fabian Forthmann
17 */
18
19 public class GameOfLifeTestOneByOneGrid {
20
21     GridFactory gridFactory = new GridFactory(1, 1, "GridArray");
22     IGrid grid = gridFactory.getGrid();
23     Point point = new Point(0, 0);
24
25     @Mock
26     ConfigReader configReader = mock(ConfigReader.class);
27
28     @Test
29     public void oneByOneGridWithFalseValueMoore(){
30         grid.setValue(point, false);
31         when(configReader.getGridLength()).thenReturn(1);
32         when(configReader.getGridWidth()).thenReturn(1);
33         when(configReader.getModel()).thenReturn("Moore");
34
35         GameOfLife gameOfLife = new GameOfLife(configReader);
36         assertFalse(gameOfLife.step(grid).getValue(point));
37     }
38
39     @Test
40     public void oneByOneGridWithTrueValueMoore(){
41         grid.setValue(point, true);
42         when(configReader.getGridLength()).thenReturn(1);
43         when(configReader.getGridWidth()).thenReturn(1);
44         when(configReader.getModel()).thenReturn("Moore");
45
46         GameOfLife gameOfLife = new GameOfLife(configReader);
47         assertFalse(gameOfLife.step(grid).getValue(point));
48     }
49
50     @Test
51     public void oneByOneGridWithFalseValueVonNeumann() {
52         grid.setValue(point, false);
53         when(configReader.getGridLength()).thenReturn(1);
54         when(configReader.getGridWidth()).thenReturn(1);
55         when(configReader.getModel()).thenReturn("vonNeumann");
56
57         GameOfLife gameOfLife = new GameOfLife(configReader);
58         assertFalse(gameOfLife.step(grid).getValue(point));
59     }
60
61     @Test
62     public void oneByOneGridWithTrueValueVonNeumann() {
63         grid.setValue(point, true);
64         when(configReader.getGridLength()).thenReturn(1);

```

```
65     when(configReader.getGridWidth()).thenReturn(1);
66     when(configReader.getModel()).thenReturn("vonNeumann");
67
68     GameOfLife gameOfLife = new GameOfLife(configReader);
69     assertFalse(gameOfLife.step(grid).getValue(point));
70 }
71 }
72
```

```
1 game.type=Parity
2 #game.type= "Parity", "GameOfLife"#
3 game.model=Moore
4 #game.model= "Moore", "vonNeumann"#
5 game.dataStructure=GridPoints
6 #game.dataStructure= "GridPoints", "GridArray"#
7 game.loggingType=console
8 #game.loggingType= "console", "file", "consoleAndFile", "disable"#
9 game.timeToLive=10
10 game.grid.length=10
11 game.grid.width=10
12 game.terminationType=ttl
13 #game.terminationType= "ttl", "noChange", "ttlOrNoChange"
14
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.grid;
6
7 public abstract class Grid {
8     public Grid() { /* compiled code */ }
9
10    abstract boolean checkFieldExists(int i, int i1);
11
12    public java.util.ArrayList<java.awt.Point> getMooreNeighbors(java.awt.Point
position) { /* compiled code */ }
13
14    public java.util.ArrayList<java.awt.Point> getVonNeumannNeighbors(java.awt.
Point position) { /* compiled code */ }
15
16    public int countMooreActiveNeighbors(java.awt.Point position) { /* compiled
code */ }
17
18    public int countVonNeumannActiveNeighbors(java.awt.Point position) { /* *
compiled code */ }
19
20    abstract boolean getValue(java.awt.Point point);
21 }
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.grid;
6
7 public interface IGrid extends java.io.Serializable {
8     void changeValue(java.awt.Point point);
9
10    java.lang.Object getGrid();
11
12    java.util.ArrayList<java.awt.Point> getMooreNeighbors(java.awt.Point point
13 );
14    java.util.ArrayList<java.awt.Point> getVonNeumannNeighbors(java.awt.Point point
15 );
16    boolean getValue(java.awt.Point point);
17
18    java.lang.String toString();
19
20    void setValue(java.awt.Point point, boolean b);
21
22    int countMooreActiveNeighbors(java.awt.Point point);
23
24    int countVonNeumannActiveNeighbors(java.awt.Point point);
25
26    java.lang.Object clone() throws java.lang.CloneNotSupportedException;
27 }
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.grid;
6
7 public class CopyGrid {
8     private final de.nordakademie.pdse.grid.IGrid grid;
9
10    public CopyGrid(de.nordakademie.pdse.grid.IGrid grid) { /* compiled code */
11        }
12
13    private java.lang.Object deepCopy(java.lang.Object object) { /* compiled
14        code */ }
15    public de.nordakademie.pdse.grid.IGrid getGrid() { /* compiled code */ }
16 }
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.grid;
6
7 public class GridArray extends de.nordakademie.pdse.grid.Grid implements de.
nordakademie.pdse.grid.IGrid {
8     private final int length;
9     private final int depth;
10    private int[][] grid;
11
12    public GridArray(int length, int depth) { /* compiled code */ }
13
14    public void changeValue(java.awt.Point position) { /* compiled code */ }
15
16    boolean checkFieldExists(int x, int y) { /* compiled code */ }
17
18    public java.lang.Object clone() throws java.lang.CloneNotSupportedException
{ /* compiled code */ }
19
20    private void createGrid() { /* compiled code */ }
21
22    public boolean getValue(java.awt.Point position) { /* compiled code */ }
23
24    private void fillGrid() { /* compiled code */ }
25
26    public int[][] getGrid() { /* compiled code */ }
27
28    public java.lang.String toString() { /* compiled code */ }
29
30    public void setValue(java.awt.Point position, boolean value) { /* compiled
code */ }
31 }
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.grid;
6
7 public class GridPoints extends de.nordakademie.pdse.grid.Grid implements de.
nordakademie.pdse.grid.IGrid {
8     private final int length;
9     private final int depth;
10    private java.util.HashMap<java.awt.Point,java.lang.Boolean> grid;
11
12    public GridPoints(int length, int depth) { /* compiled code */ }
13
14    public void changeValue(java.awt.Point position) { /* compiled code */ }
15
16    public java.util.HashMap<java.awt.Point,java.lang.Boolean> getGrid() { /* 
compiled code */ }
17
18    public void setValue(java.awt.Point position, boolean value) { /* compiled 
code */ }
19
20    boolean checkFieldExists(int x, int y) { /* compiled code */ }
21
22    public java.lang.Object clone() throws java.lang.CloneNotSupportedException
{ /* compiled code */ }
23
24    private void createGrid() { /* compiled code */ }
25
26    public java.lang.String toString() { /* compiled code */ }
27
28    public boolean getValue(java.awt.Point position) { /* compiled code */ }
29 }
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.grid;
6
7 public class GridFactory {
8     private final int depth;
9     private final int length;
10    private final java.lang.String dataStructure;
11
12    public GridFactory(int length, int depth, java.lang.String dataStructure
13    ) { /* compiled code */ }
14
15    public de.nordakademie.pdse.grid.IGrid getGrid() { /* compiled code */ }
16 }
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.config;
6
7 public class ConfigReader {
8     private final java.util.Properties prop;
9
10    public ConfigReader(java.io.File propertyFile) { /* compiled code */ }
11
12    public java.lang.String getGameType() { /* compiled code */ }
13
14    public java.lang.String getModel() { /* compiled code */ }
15
16    public java.lang.String getDatastructure() { /* compiled code */ }
17
18    public java.lang.String getLoggingType() { /* compiled code */ }
19
20    public int getTimeToLive() { /* compiled code */ }
21
22    public int getGridLength() { /* compiled code */ }
23
24    public int getGridWidth() { /* compiled code */ }
25
26    public java.lang.String getTerminationType() { /* compiled code */ }
27
28    public void setGameType(java.lang.String gameType) { /* compiled code */ }
29
30    public void setModel(java.lang.String gameModel) { /* compiled code */ }
31
32    public void setDatastructure(java.lang.String datastructure) { /* compiled
   code */ }
33
34    public void setLoggingType(java.lang.String loggingType) { /* compiled code
   */ }
35
36    public void setTimeToLive(java.lang.String timeToLive) { /* compiled code
   */ }
37
38    public void setGridLength(java.lang.String gridLength) { /* compiled code
   */ }
39
40    public void setGridWidth(java.lang.String gridWidth) { /* compiled code */
   }
41
42    public void setTerminationType(java.lang.String terminationType) { /*
   compiled code */ }
43 }
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.logging;
6
7 public class Logger {
8     private final java.lang.String loggingType;
9     private java.io.File file;
10
11    public Logger(java.lang.String loggingType) { /* compiled code */ }
12
13    public void addGridToLog(java.lang.String grid, int iteration) { /* compiled code */ }
14
15    private void createFile() { /* compiled code */ }
16
17    private java.lang.String getCurrentTime() { /* compiled code */ }
18
19    private java.io.File getFile() { /* compiled code */ }
20
21    private void writeLogToConsole(java.lang.String input) { /* compiled code */ }
22
23    private void writeLogToFile(java.lang.String input) { /* compiled code */ }
24 }
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.gamelogic;
6
7 public class Game {
8     de.nordakademie.pdse.grid.IGrid grid;
9     de.nordakademie.pdse.gamelogic.IGameType gameType;
10    de.nordakademie.pdse.config.ConfigReader configReader;
11    java.lang.Boolean continueGame;
12    de.nordakademie.pdse.logging.Logger logger;
13    int timeToLive;
14    int iteration;
15
16    public Game(de.nordakademie.pdse.config.ConfigReader configReader) { /* compiled code */ }
17
18    private java.lang.Boolean getContinueGame() { /* compiled code */ }
19
20    private void setContinueGame(java.lang.Boolean continueGame) { /* compiled code */ }
21
22    private void reduceTimeToLive() { /* compiled code */ }
23
24    private int getTimeToLive() { /* compiled code */ }
25
26    private boolean isTimeToLiveZero() { /* compiled code */ }
27
28    private boolean processTimeToLive() { /* compiled code */ }
29
30    private boolean gridChanged(de.nordakademie.pdse.grid.IGrid grid) { /* compiled code */ }
31
32    private void checkForTermination(de.nordakademie.pdse.grid.IGrid newGrid)
33    ) { /* compiled code */ }
34
35    public de.nordakademie.pdse.grid.IGrid getGrid() { /* compiled code */ }
36
37    public void setGrid(de.nordakademie.pdse.grid.IGrid grid) { /* compiled code */ }
38
39    public int getIteration() { /* compiled code */ }
40
41    public void setIteration(int iteration) { /* compiled code */ }
42
43    private int getCurrentIterationAndIterate() { /* compiled code */ }
44
45    public void run() throws java.lang.Exception { /* compiled code */ }
45 }
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.gamelogic;
6
7 public class Parity implements de.nordakademie.pdse.gamelogic.IGameType {
8     de.nordakademie.pdse.config.ConfigReader configReader;
9
10    public Parity(de.nordakademie.pdse.config.ConfigReader configReader) { /*  
     compiled code */ }
11
12    public de.nordakademie.pdse.grid.IGrid step(de.nordakademie.pdse.grid.IGrid  
oldGrid) throws java.lang.Exception { /* compiled code */ }
13 }
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.gamelogic;
6
7 public interface IGameType {
8     de.nordakademie.pdse.grid.IGrid step(de.nordakademie.pdse.grid.IGrid iGrid
9 } throws java.lang.Exception;
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.gamelogic;
6
7 public class GameOfLife implements de.nordakademie.pdse.gamelogic.IGameType {
8     de.nordakademie.pdse.config.ConfigReader configReader;
9
10    public GameOfLife(de.nordakademie.pdse.config.ConfigReader configReader) {
11        /* compiled code */
12    }
13
14    public de.nordakademie.pdse.grid.IGrid step(de.nordakademie.pdse.grid.IGrid
15        oldGrid) { /* compiled code */ }
16
17    private static boolean getPointStatus(java.lang.Boolean point, int
18        aliveNeighbors) { /* compiled code */ }
19}
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.gamelogic;
6
7 public class GameTypeFactory {
8     de.nordakademie.pdse.config.ConfigReader configReader;
9
10    public GameTypeFactory(de.nordakademie.pdse.config.ConfigReader
11                           configReader) { /* compiled code */ }
12
13    public de.nordakademie.pdse.gamelogic.IGameType getGameType() { /* compiled
14        code */ }
15}
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.experiments.parity;
6
7 public class Parity_Experiment_1_GridArray {
8     public Parity_Experiment_1_GridArray() { /* compiled code */ }
9
10    private static java.awt.Point createPoint(int x, int y) { /* compiled code */
11        /* */
12    }
13 }
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.experiments.parity;
6
7 public class Parity_Experiment_1_GridPoint {
8     public Parity_Experiment_1_GridPoint() { /* compiled code */ }
9
10    private static java.awt.Point createPoint(int x, int y) { /* compiled code */
11        /* */
12    }
13 }
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.experiments.parity;
6
7 public class Parity_Experiment_2_GridArray {
8     public Parity_Experiment_2_GridArray() { /* compiled code */ }
9
10    private static java.awt.Point createPoint(int x, int y) { /* compiled code */
11        /* */
12    }
13 }
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.experiments.parity;
6
7 public class Parity_Experiment_2_GridPoint {
8     public Parity_Experiment_2_GridPoint() { /* compiled code */ }
9
10    private static java.awt.Point createPoint(int x, int y) { /* compiled code */
11        /* */
12    }
13 }
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.experiments.gameoflife;
6
7 public class GameOfLife_Experiment_1_GridArray {
8     public GameOfLife_Experiment_1_GridArray() { /* compiled code */ }
9
10    private static java.awt.Point createPoint(int x, int y) { /* compiled code */
11        /* */
12    }
13 }
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.experiments.gameoflife;
6
7 public class GameOfLife_Experiment_1_GridPoint {
8     public GameOfLife_Experiment_1_GridPoint() { /* compiled code */ }
9
10    private static java.awt.Point createPoint(int x, int y) { /* compiled code */
11        /* */
12    }
13 }
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.experiments.gameoflife;
6
7 public class GameOfLife_Experiment_2_GridArray {
8     public GameOfLife_Experiment_2_GridArray() { /* compiled code */ }
9
10    private static java.awt.Point createPoint(int x, int y) { /* compiled code */
11        /* */
12    }
13 }
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.experiments.gameoflife;
6
7 public class GameOfLife_Experiment_3_GridArray {
8     public GameOfLife_Experiment_3_GridArray() { /* compiled code */ }
9
10    public static void main(java.lang.String[] args) throws java.lang.Exception
11        { /* compiled code */ }
12 }
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.experiments.gameoflife;
6
7 public class GameOfLife_Experiment_3_GridPoint {
8     public GameOfLife_Experiment_3_GridPoint() { /* compiled code */ }
9
10    public static void main(java.lang.String[] args) throws java.lang.Exception
11        { /* compiled code */ }
12 }
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.experiments.gameoflife;
6
7 public class GameOfLife_Experiment_2_GridPoints {
8     public GameOfLife_Experiment_2_GridPoints() { /* compiled code */ }
9
10    private static java.awt.Point createPoint(int x, int y) { /* compiled code */
11        /* */
12    }
13 }
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.grid;
6
7 public class GridTest {
8     public GridTest() { /* compiled code */ }
9
10    @org.junit.Test
11    public void countMooreActiveNeighbors() { /* compiled code */ }
12
13    @org.junit.Test
14    public void countVonNeumannActiveNeighbors() { /* compiled code */ }
15 }
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.grid;
6
7 public class CopyGridTest {
8     public CopyGridTest() { /* compiled code */ }
9
10    @org.junit.Test
11    public void testGetGrid() { /* compiled code */ }
12 }
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.grid;
6
7 public class GridArrayTest {
8     public GridArrayTest() { /* compiled code */ }
9
10    @org.junit.Test
11    public void testCreateGrid() { /* compiled code */ }
12
13    @org.junit.Test
14    public void testChangeValue() { /* compiled code */ }
15
16    @org.junit.Test
17    public void testGetVonNeumannNeighbors() { /* compiled code */ }
18
19    @org.junit.Test
20    public void testGetMooreNeighbors() { /* compiled code */ }
21
22    @org.junit.Test
23    public void testToString() { /* compiled code */ }
24
25    @org.junit.Test
26    public void testSetValue() { /* compiled code */ }
27 }
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.grid;
6
7 public class GridPointsTest {
8     public GridPointsTest() { /* compiled code */ }
9
10    @org.junit.Test
11    public void testCreateGrid() { /* compiled code */ }
12
13    @org.junit.Test
14    public void testChangeValue() { /* compiled code */ }
15
16    @org.junit.Test
17    public void testGetVonNeumannNeighbors() { /* compiled code */ }
18
19    @org.junit.Test
20    public void testGetMooreNeighbors() { /* compiled code */ }
21
22    @org.junit.Test
23    public void testSetValue() { /* compiled code */ }
24
25    @org.junit.Test
26    public void testToString() { /* compiled code */ }
27 }
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.grid;
6
7 public class GridFactoryTest {
8     public GridFactoryTest() { /* compiled code */ }
9
10    @org.junit.Test
11    public void testGetGridAsGridArray() { /* compiled code */ }
12
13    @org.junit.Test
14    public void testGetGridAsGridPoints() { /* compiled code */ }
15 }
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.config;
6
7 public class ConfigReaderTest {
8     private static final java.lang.String GAME_TYPE = "1";
9     private static final java.lang.String MODEL = "2";
10    private static final java.lang.String DATA_STRUCTURE = "3";
11    private static final java.lang.String LOGGING_TYPE = "4";
12    private static final java.lang.String TIME_TO_LIVE = "5";
13    private static final java.lang.String GRID_LENGTH = "6";
14    private static final java.lang.String GRID_WIDTH = "7";
15    private static final java.lang.String TERMINATION_TYPE = "8";
16    private static final java.lang.String ALTERNATIVE_GAME_TYPE = "A";
17    private static final java.lang.String ALTERNATIVE_MODEL = "B";
18    private static final java.lang.String ALTERNATIVE_DATA_STRUCTURE = "C";
19    private static final java.lang.String ALTERNATIVE_LOGGING_TYPE = "D";
20    private static final java.lang.String ALTERNATIVE_TIME_TO_LIVE = "10";
21    private static final java.lang.String ALTERNATIVE_GRID_LENGTH = "20";
22    private static final java.lang.String ALTERNATIVE_GRID_WIDTH = "30";
23    private static final java.lang.String ALTERNATIVE_TERMINATION_TYPE = "E";
24    @org.junit.Rule
25    public org.junit.rules.TemporaryFolder tempFolder;
26
27    public ConfigReaderTest() { /* compiled code */ }
28
29    @org.junit.Test
30    public void readExistingPropertiesFromFile() throws java.io.IOException {
31        /* compiled code */
32    }
33    @org.junit.Test
34    public void setPropertyValues() throws java.io.IOException { /* compiled
35        code */
36    }
37    @org.junit.Test
38    @org.junit.Test(expected = java.lang.RuntimeException.class)
39    public void readNotExistingConfigFile() { /* compiled code */
40
41        private java.util.Properties initializeValidPropertiesFile() { /* compiled
42            code */
43    }
44}
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.logging;
6
7 public class LoggerTest {
8     public LoggerTest() { /* compiled code */ }
9
10    @org.junit.Test
11    public void addGridToLog() { /* compiled code */ }
12 }
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.gamelogic;
6
7 public class GameTest {
8     @org.mockito.Mock
9     de.nordakademie.pdse.config.ConfigReader configReader;
10    @org.mockito.Mock
11    de.nordakademie.pdse.grid.IGrid grid;
12    @org.mockito.Mock
13    de.nordakademie.pdse.grid.IGrid grid2;
14
15    public GameTest() { /* compiled code */ }
16
17    @org.junit.Test
18    public void getGird() { /* compiled code */ }
19
20    @org.junit.Test
21    public void setGrid() { /* compiled code */ }
22
23    @org.junit.Test
24    public void run() { /* compiled code */ }
25 }
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.gamelogic;
6
7 public class ParityTestSetAlive {
8     de.nordakademie.pdse.grid.GridFactory gridFactory;
9     de.nordakademie.pdse.grid.IGrid grid;
10    @org.mockito.Mock
11    de.nordakademie.pdse.config.ConfigReader configReader;
12
13    public ParityTestSetAlive() { /* compiled code */ }
14
15    private static java.awt.Point createPoint(int x, int y) { /* compiled code */
16        /*
17         * @org.junit.Test
18         * public void invertGrid() throws java.lang.Exception { /* compiled code */
19     }
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.gamelogic;
6
7 public class GameTypeFactoryTest {
8     @org.mockito.Mock
9     de.nordakademie.pdse.config.ConfigReader configReader;
10
11    public GameTypeFactoryTest() { /* compiled code */ }
12
13    @org.junit.Test
14    public void returnGameOfLife() { /* compiled code */ }
15
16    @org.junit.Test
17    public void returnParity() { /* compiled code */ }
18
19    @org.junit.Test(expected = java.lang.IllegalArgumentException.class)
20    public void invalidGameType() { /* compiled code */ }
21 }
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.gamelogic;
6
7 public class GameOfLifeTestSetAlive {
8     de.nordakademie.pdse.grid.GridFactory gridFactory;
9     de.nordakademie.pdse.grid.IGrid grid;
10    @org.mockito.Mock
11    de.nordakademie.pdse.config.ConfigReader configReader;
12
13    public GameOfLifeTestSetAlive() { /* compiled code */ }
14
15    private static java.awt.Point createPoint(int x, int y) { /* compiled code */
16        /*
17         * @org.junit.Test
18         * public void invertGrid() throws java.lang.Exception { /* compiled code */
19     }
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.gamelogic;
6
7 public class ParityTestOneByOneGrid {
8     de.nordakademie.pdse.grid.GridFactory gridFactory;
9     de.nordakademie.pdse.grid.IGrid grid;
10    java.awt.Point point;
11    @org.mockito.Mock
12    de.nordakademie.pdse.config.ConfigReader configReader;
13
14    public ParityTestOneByOneGrid() { /* compiled code */ }
15
16    @org.junit.Test(expected = java.lang.IllegalArgumentException.class)
17    public void emptyGridMoore() throws java.lang.Exception { /* compiled code */
18        /* */
19    }
20
21    @org.junit.Test(expected = java.lang.IllegalArgumentException.class)
22    public void trueGridMoore() throws java.lang.Exception { /* compiled code */
23        /* */
24    }
25
26    @org.junit.Test
27    public void emptyGridNeumann() throws java.lang.Exception { /* compiled code */
28        /* */
29    }
30
31    @org.junit.Test
32    public void trueGridNeumann() throws java.lang.Exception { /* compiled code */
33        /* */
34    }
35}
```

```
1 // IntelliJ API Decompiler stub source generated from a class file
2 // Implementation of methods is not available
3
4
5 package de.nordakademie.pdse.gamelogic;
6
7 public class GameOfLifeTestOneByOneGrid {
8     de.nordakademie.pdse.grid.GridFactory gridFactory;
9     de.nordakademie.pdse.grid.IGrid grid;
10    java.awt.Point point;
11    @org.mockito.Mock
12    de.nordakademie.pdse.config.ConfigReader configReader;
13
14    public GameOfLifeTestOneByOneGrid() { /* compiled code */ }
15
16    @org.junit.Test
17    public void gridValueFalseMoore() throws java.lang.Exception { /* compiled
   code */ }
18
19    @org.junit.Test
20    public void gridValueTrueMoore() throws java.lang.Exception { /* compiled
   code */ }
21
22    @org.junit.Test
23    public void gridValueFalseNeumann() throws java.lang.Exception { /*
   compiled code */ }
24
25    @org.junit.Test
26    public void gridValueTrueNeumann() throws java.lang.Exception { /* compiled
   code */ }
27
28    @org.junit.Test(expected = java.lang.IllegalArgumentException.class)
29    public void gridValueFalseInvalidModel() throws java.lang.Exception { /*
   compiled code */ }
30
31    @org.junit.Test(expected = java.lang.IllegalArgumentException.class)
32    public void gridValueTrueInvalidModel() throws java.lang.Exception { /*
   compiled code */ }
33 }
```