
Dokumentation der Praktischen Arbeit
zur Prüfung zum
Mathematisch-technischen Softwareentwickler

29. April 2020

Lukas Dahlberg

Prüfungs-Nummer:

Programmiersprache: JAVA 8

Inhaltsverzeichnis

1. Aufgabenanalyse	1
1.1. Analyse	1
1.2. Eingabeformat	1
1.3. Ausgabeformat	2
1.4. Anforderung an das Programm	2
1.5. Sonderfälle	2
1.6. Fehlerfälle	2
2. Verfahrensbeschreibung	3
2.1. Vorgehensweise	3
2.2. Gesamtsystem	3
2.2.1. Main-Funktion	3
2.2.2. Model-Klassen	3
2.2.3. IO-Klassen	3
2.2.4. Controller-Klassen	4
2.3. Datenfluss	4
3. Programmbeschreibung	5
3.1. Pakete	5
3.1.1. Model	6
3.1.2. IO	6
3.1.3. Controller	7
3.2. Schnittstellen	7
3.2.1. Program	7
3.2.2. Spielstein	7
3.2.3. Spielfeld	8
3.2.4. DateiBehandlung	9
3.2.5. PuzzleLoeser	9
3.3. Präzisierung	12
3.3.1. Präzisierung SetSpielstein	12
3.3.2. Präzisierung GetKombination	13
3.3.3. Präzisierung GetNachbarSteine	13
3.3.4. Präzisierung LoesePuzzle	13
3.3.5. Präzisierung Drehen	13
3.4. Sequenzdiagramm	14

4. Testdokumentation	15
4.1. Systemtests	15
4.1.1. Normalfall	15
4.1.2. Sonderfall	16
4.1.3. Fehlerfall	16
4.2. Ausführliches Beispiel	17
5. Zusammenfassung und Ausblick	19
5.1. Zusammenfassung	19
5.2. Ausblick	20
A. Abweichungen und Ergänzungen zum Vorentwurf	21
A.1. Abweichungen	21
A.2. Ergänzungen	21
B. Benutzeranleitung	23
B.1. Verzeichnisstruktur der Abgabe	23
B.2. Vorbereitung des Systems	23
B.2.1. Systemvoraussetzungen	23
B.2.2. Installation	24
B.3. Kompilieren	25
B.4. Programmaufruf	26
B.5. Testen der Beispiele	26
B.6. Fehlerfälle	26
C. Entwicklungsumgebung	29
D. Verwendete Hilfsmittel	31
E. Erklärung	33
F. Aufgabenstellung	35
G. Quellcode	37
G.1. Source Dateien	37
G.1.1. Default Package	37
G.1.2. io	38
G.2. Skript Dateien	46
G.2.1. Windows	46
G.2.2. MacOS und Linux	50
H. In- und Output der Testdokumentation	53
H.1. Normalfall	53
H.1.1. IHK-Beispiel 1	53
H.2. Sonderfall	54

H.3. Fehlerfall	55
---------------------------	----

1. Aufgabenanalyse

1.1. Analyse

1.2. Eingabeformat

1.3. Ausgabeformat

1.4. Anforderung an das Programm

1.5. Sonderfälle

1.6. Fehlerfälle

Die auftretenden Fehler kann man in drei verschiedene Fehlerarten aufteilen. Zu diesen gehören technische, syntaktische und semantische Fehlerfälle.

Technische Fehler liegen vor, wenn die angegebene Datei nicht vorhanden ist, durch fehlende Zugriffsrechte nicht gelesen werden kann, oder die Ausgabedatei durch fehlende Schreibrechte nicht erstellt werden kann. Syntaktische Fehler treten auf, wenn die Eingabedatei nicht dem vorgegebenen Format entspricht, bei semantischen Fehlern sind die Daten in der Eingabedatei fehlerhaft.

Durch die Analyse der Aufgabenstellung und des Eingabeformats ergeben sich folgende syntaktische Fehlerfälle:

- Mögliche Fehlerfälle auflisten

Durch die Analyse der Aufgabenstellung und des Eingabeformats ergeben sich folgende semantische Fehlerfälle:

- Mögliche Fehlerfälle auflisten

Die Behandlung der Fehlerfälle wird in der Verfahrensbeschreibung angegeben.

2. Verfahrensbeschreibung

2.1. Vorgehensweise

Sonderfälle

Die oben genannten Sonderfälle werden wie folgt behandelt:

Sonderfall Bezeichnung Sonderfall Beschreibung und Behandlung

Fehlerfälle

Die oben genannten Fehlerfälle werden wie folgt behandelt:

Fehlerfall Bezeichnung Fehlerfall Beschreibung und Behandlung

In den Fällen, wo nach einem Fehler weitergearbeitet wird, wird auch weiter auf andere Fehler geprüft. Sollten Fehler entdeckt werden und die Verarbeitung fortgesetzt, so wird in der Ausgabe eine Hinweismeldung erscheinen, dass trotz eines Fehlers die Verarbeitung fortgeführt wurde.

2.2. Gesamtsystem

2.2.1. Main-Funktion

2.2.2. Model-Klassen

2.2.3. IO-Klassen

2.2.4. Controller-Klassen

2.3. Datenfluss

3. Programmbeschreibung

3.1. Pakete

Das Programm ist in drei Module unterteilt, welche nach dem MVC-Model entwickelt wurden. Für jedes dieser Module wurde zunächst ein Interface deklariert, damit das Programm leicht wart- und erweiterbar bleibt. Die Namen der Interfaces beginnen mit einem „I“, wie es etwaige Namenskonventionen vorgeben.

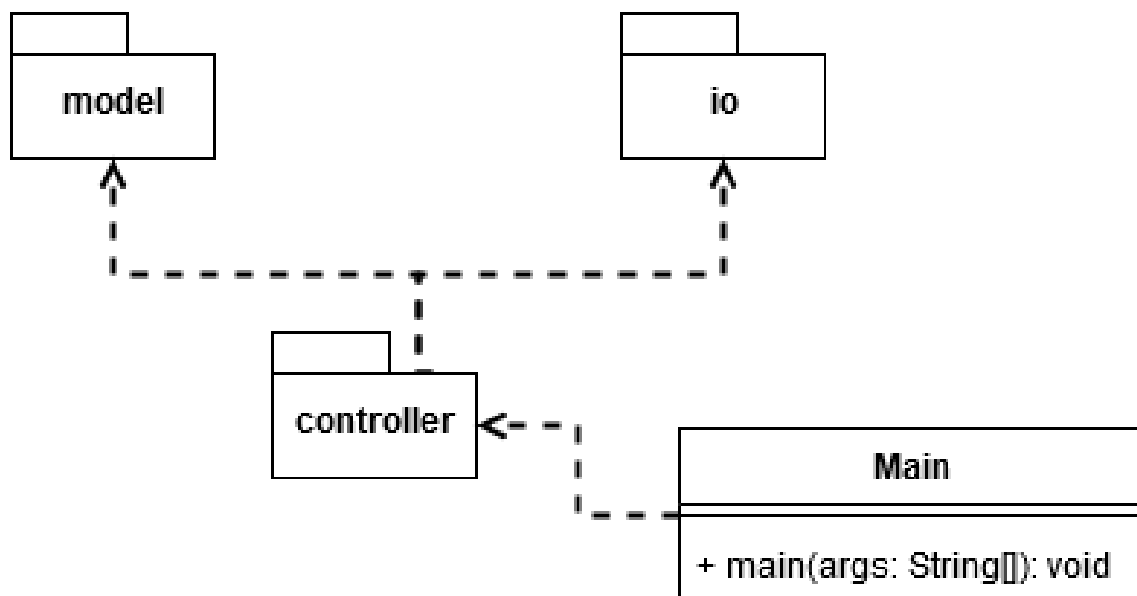


Abbildung 3.1.: Übersicht zu den Paketen

3.1.1. Model

IModel Schnittstelle zum Speichern der Daten: Spielsteine, Spielfeld, Matrix der Nachbarschaft für Felder

Model Implementierung von IModel.

3.1.2. IO

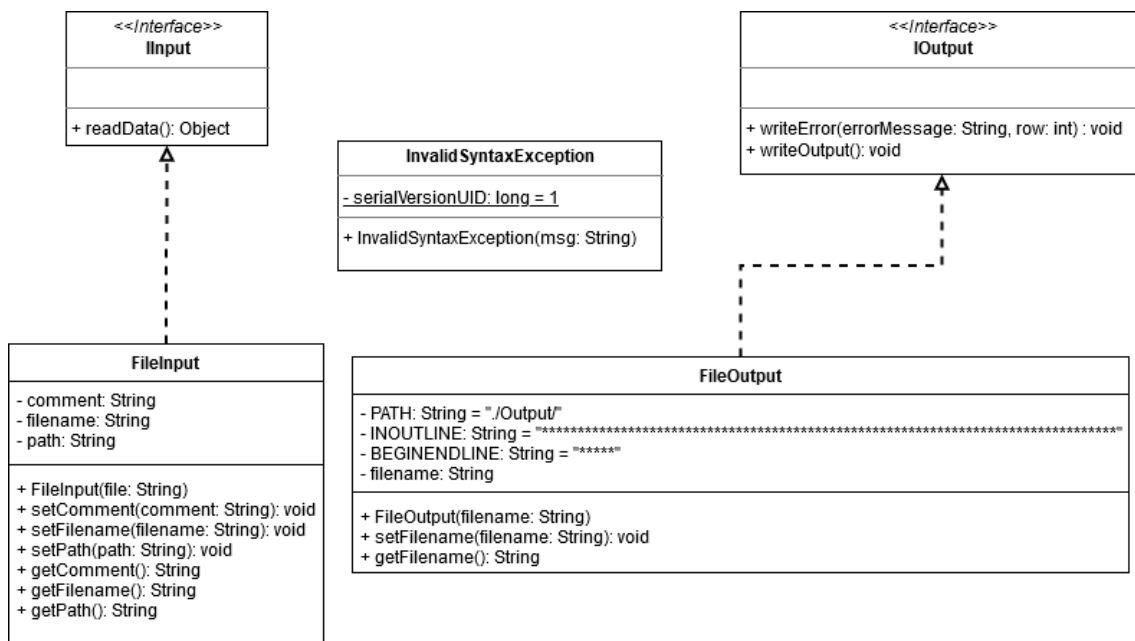


Abbildung 3.2.: Klassendiagramme aus dem Paket IO

IInput Schnittstelle zur Eingabe der Daten.

FileInput Implementierung von IInput. Die Klasse verfügt über eine Methode zum einlesen der Dateien, und eine weitere zur syntaktischen Überprüfung dieser.

IOutput Schnittstelle zur Ausgabe der Daten.

FileOutput Implementierung von IOutput. Die Klasse verfügt über zwei Methoden, eine zum schreiben einer einfachen Ausgabedatei, und eine zweite zum schreiben einer Fehlerdatei.

InvalidSyntaxException Exception Klasse zum signalisieren, dass ein syntaktischer Fehler in der Eingabedatei vorliegt. Wird in der Klasse **FileInput** verwendet.

3.1.3. Controller

PuzzleLoeserFactory Schnittstelle zum Starten des Controllers, damit die Verarbeitung gesteuert werden kann.

PuzzleLoeser Abstrakte Klasse als Schnittstelle zum Starten. **PuzzleLoeserEinsetzen** ↪. Beinhaltet die Implementierungen und den Lösungsalgorithmus. ↪

3.2. Schnittstellen

3.2.1. Program

Main

```
public static void main(string [] args)
```

Eingabeparameter: Array mit den Kommandozeilen-Argumenten

Rückgabeparameter: keine

3.2.2. Spielstein

drehen

```
public Spielstein drehen();
```

Eingabeparameter: keine vorhanden

Rückgabeparameter: der gedrehte Spielstein wird zurückgegeben.

getFeld

```
public int getFeld();
```

Eingabeparameter: keine vorhanden

Rückgabeparameter: das Feld, auf dem der Spielstein liegt, ist der Spielstein nicht gelegt, wird 0 zurückgegeben.

setFeld

```
public void setFeld(int feld);
```

Eingabeparameter: Die Feldnummer, des Feldes, in die der Stein gesetzt wurde.

Rückgabeparameter: keiner

kannKombi

```
public boolean kannKombi(List<Integer> kombination);
```

Eingabeparameter: Eine Liste mit den Kantenziffern, der Nachbarfelder, beginnend mit der Grundkante.

Rückgabeparameter: Wenn der Spielstein in der aktuellen Orientierung die Kombination abdecken kann, dann wird true zurückgegeben, sonst false .

3.2.3. Spielfeld

GetInstance

```
public Spielfeld getInstance()
```

Eingabeparameter: keiner

Rückgabeparameter: eine Spielfeldinstanz.

GetKombination

```
public List<Integer> GetKombination(int feld)
```

Eingabeparameter: Die Feldnummer, für das Feld, deren Kombination gesucht ist.

Rückgabeparameter: Eine Liste mit Integer-Werten für die Kantenziffer, die an Position x vorhanden sein muss, beginnend von der Grundkante. Liegt an einer Kante noch kein weiterer Stein, wird anstelle der Kantenziffer ein null in der Liste zurückgegeben.

3.2.4. DateiBehandlung

SchreibeDatei

```
public void schreibeDatei(List<Spielstein> loesung)
```

Eingabeparameter: Eine Liste der Spielsteine, aufsteigend nach den Feldern, in die diese gelegt werden sortiert. Ist keine Lösung vorhanden, wird eine leere Liste übergeben.

Rückgabeparameter: keiner

SchreibeFehler

```
public void schreibeFehler(String text)
```

Eingabeparameter: Die Fehlermeldung die Ausgegeben werden soll, mit vorangestellter Zeile und Datei in der der Fehler festgestellt wurde, getrennt durch #

Rückgabeparameter: keiner

LeseDatei

```
public List<Spielstein> leseDatei()
```

Eingabeparameter: keine vorhanden

Rückgabeparameter: Eine Liste der in der Eingabedatei beschriebenen Spielsteine, sortiert nach der Einlesereihenfolge. Kommt es beim einlesen zu Fehlern, so wird null zurückgegeben.

3.2.5. PuzzleLoeser

loesePuzzle

```
public void loesePuzzle()
```

Eingabeparameter: keine vorhanden

Rückgabeparameter: keiner

GetLoesung

```
public List<Spielstein> getLoesung()
```

Eingabeparameter: keine vorhanden

Rückgabeparameter: Eine Liste der Spielsteine, aufsteigend nach den Feldern, in die diese gelegt werden sortiert. Ist keine Lösung vorhanden, wird eine leere Liste zurückgegeben.

setzeSpielstein

```
private boolean setzeTeilInFeld(Spielstein stein, int feld)
```

Eingabeparameter: Der zu setzende Spielstein und die Feldnummer des Feldes, in welcher der Spielstein zu setzen ist.

Rückgabeparameter: Wenn der Stein in das Feld gesetzt werden kann, wird `true` zurückgegeben, andernfalls `false`.

PuzzleLoesbarAnfang

```
private boolean puzzleLoesbarAnfang()
```

Eingabeparameter: keine vorhanden

Rückgabeparameter: Wenn das Puzzle lösbar scheint, wird `true` zurückgegeben, andernfalls `false`. D.h., wenn die Anzahl der einzelnen Kantenziffern gerade ist, wird angenommen, dass das Puzzle lösbar ist.

PuzzleLoesbarAlgorithmus

```
private boolean puzzleLoesbarAlgorithmus()
```

Eingabeparameter: keine vorhanden

Rückgabeparameter: Wenn noch setzbare Steine vorhanden sind, wo mindestens einer, noch gesetzt werden kann, wird `true`, andernfalls `false` zurückgegeben.

PuzzleGeloest

```
private boolean puzzleGeloest()
```

Eingabeparameter: keine vorhanden

Rückgabeparameter: gibt `true` zurück, wenn keine ungenutzten Steine mehr vorhanden sind, andernfalls wird `false` zurückgegeben.

3.3. Präzisierung

3.3.1. Präzisierung SetSpielstein

In der Methode `setSpielstein(Spielstein stein, int feld)` wird überprüft, ob der Spielstein in das Feld setzbar ist. Hierfür wird zunächst mit der Methode `kannKombi(List<Integer> kombination)` geschaut, ob der übergebene Spielstein, die Kombination für das übergebene Feld abdecken kann. Zur Bestimmung der Kombination wird hier die Methode `getKombination(int feld)` aufgerufen. Wenn gewährleistet ist, dass der Spielstein in das Feld setzbar ist, dann wird das Array `felder` angepasst. Der übergebene Spielstein wird in `felder[feld - 1]` gesetzt. In Folge dessen werden die Attribute des Steins angepasst, dass dieser als gelegt aufgefasst wird, in dem diesen die Informationen, des Feldes in dem der Stein liegt übergeben wird und das boolsche Attribut `gelegt` auf `true` gesetzt wird.

3.3.2. Präzisierung GetKombination

3.3.3. Präzisierung GetNachbarSteine

3.3.4. Präzisierung LoesePuzzle

3.3.5. Präzisierung Drehen

3.4. Sequenzdiagramm

4. Testdokumentation

Parallel zu dem Programm wurden Tests erstellt, um die direkte Funktionalität von hinzugefügtem Code zu überprüfen. Diese wurden als Systemtests ausgeführt und sind nach jeder Erweiterung des Programms komplett ausgeführt worden. Die Tests sind in Form von Eingabedateien in der Abgabe im Ordner Tests, die dazugehörigen Ausgaben im Ordner Output enthalten.



4.1. Systemtests

Alle Systemtests wurden durch Dateien getestet und die Ergebnisse manuell überprüft, weil eine automatische Verifizierung über Dateien für den gegebenen Zeitraum zu aufwendig ist und nicht gefordert war. Die Ein- und Ausgabedateien der Systemtests sind in der Abgabe enthalten.

4.1.1. Normalfall

Der Normalfall liegt vor, wenn der Inhalt der Eingabedatei keine Sonderfälle aus Abschnitt 1.5 auf Seite 2 beinhaltet und keiner der in Abschnitt 1.6 auf Seite 2 aufgelisteten Fehler eintritt.

Der Normalfall ist anhand des gegebenen IHK-Beispiels getestet worden. Anhand des ersten IHK-Beispiels wird in Abschnitt 4.2 auf Seite 17 der Algorithmus und die einzelnen Schritte genauer erläutert.

Für den Normalfall sind noch vier weitere Testbeispiele entwickelt worden. Das erste Beispiel `LOESBAR_Beispiel1.txt` soll einen Normalfall mit vielen gleichen Werten testen, das zweite Beispiel `LOESBAR_Beispiel2.txt` soll die Anforderung testen, dass zwischen den Kantenziffern beliebig viele Leerzeichen sein können. Die dritte Beispieldatei `LOESBAR_Beispiel3`  `.txt` soll einen Normalfall mit nur unterschiedlichen Teilen testen, die auch bei Verdrehung nicht mit den anderen übereinstimmen. Das vierte Beispiel `NICHT_LOESBAR_Beispiel1`  `.txt` testet die Anforderungen für den Fall, dass ein Puzzle übergeben wurde, welches nicht dem Sonderfall entspricht, da die Anzahl jeder Kantenziffer gerade ist, jedoch auch zu keiner Lösung führt.

4.1.2. Sonderfall

Für den Sonderfall, der in der Aufgabenanalyse im Abschnitt Sonderfälle (Abschnitt 1.5 auf Seite 2) beschrieben wurde, wurden insgesamt drei Testfälle erstellt, wobei die drei Testfälle eine ungerade Anzahl an Kantenziffern testet. Es waren mehr Testfälle als aufgelistete Sonderfälle nötig, weil die Beschreibung eines Sonderfalls allgemein ist, der Sonderfall aber horizontal und vertikal überprüft werden muss.

4.1.3. Fehlerfall

Es sind für die sieben Fehlerfälle, die in der Aufgabenanalyse im Abschnitt Fehlerfälle (Abschnitt 1.6 auf Seite 2) aufgelistet sind, sieben Tests entwickelt worden. Wie auch schon bei den Sonderfällen sind einige in der Beschreibung zusammengefasst, aber einzeln getestet worden. In allen Fällen wird eine Fehlermeldung in die Ausgabedatei geschrieben und das Programm korrekt beendet, ohne die Logik zum Lösen des Puzzles zu starten.

Zusätzlich wurde noch getestet, wie das Programm auf nicht vorhandene Eingabedateien und nicht schreibbare Ausgabedateien reagiert. In beiden Fällen wurde die Fehlermeldung in einer Ausgabedatei im Root-Ordner ausgegeben und das Programm korrekt beendet.

4.2. Ausführliches Beispiel

An dieser Stelle wird das erste IHK-Beispiel herangezogen, also ein gültiges und lösbares Puzzle. Da das Beispiel keinerlei Fehler enthält, wird auf die Prüfung dieser in dem Beispiel verzichtet.

Die Eingabedatei von diesem Beispiel sieht wie folgt aus:

Trotz der Null-Indizierung in Java, werden weiterhin die Indizes 1 bis 12 verwendet. Im Programmcode werden auch diese Werte weitergegeben und beim auslesen oder schreiben von Arrays und Lists entsprechend angepasst.

Zunächst wird bestimmt, ob das Puzzle überhaupt lösbar ist mit der Methode `puzzleLoesbarAnfang` $\rightarrow ()$.

Die Rechenvariablen `anzKZ0`, `anzKZ1` und `anzKZ2` werden mit 0 initialisiert. In Folge dessen werden alle Kanten jedes Puzzlesteils einmal aufgerufen, und die entsprechende Rechenvariable um 1 erhöht, wenn eine 0, 1 oder 2 eingelesen wurde. Im Anschluss wird das Ergebnis der boolschen Berechnung des Ergebnis, in dem überprüft wird, ob `anzKZ0 \% 2 == 0` $\rightarrow \&\& \text{anzKZ1} \% 2 == 0 \ \&\& \text{anzKZ2} \% 2 == 0$ ist, zurückgegeben.

Anschließend wird der Backtracking Algorithmus gestartet. Hier wird bis das Puzzle gelöst ist probiert, den ungenutzten Spielstein mit der geringsten Kartenummer (Position in der Eingabedatei) auf das nächste freie Feld zu setzen. Begonnen wird mit dem Feld 1 und Stein 1, welcher aufgrund dessen, dass keinerlei Bedingungen an diesen gestellt werden, gelegt werden kann.

Nun wird die Tiefe des Algorithmus um 1 erhöht, und es wird fortgefahren mit dem einsetzen des 2. Steins in Feld 2. Dieser passt jedoch in der Grundorientierung nicht an das besetzte Feld 1 dran. Aus diesem Grund wird der Stein solange gedreht, bis dieser passt. Nach maximal 5 Drehungen, gilt der Stein als nicht legbar und der Algorithmus würde wieder eine Stufe nach oben gehen. Da der Stein allerdings nach einer Drehung an Feld 1 passt, wird mit dem dritten Spielstein für Feld drei fortgefahren. Nachdem der Algorithmus fertig ist, wird diese Ausgabedatei erstellt:

5. Zusammenfassung und Ausblick

5.1. Zusammenfassung

Durch die funktionale Trennung nach dem Model-View-Controller-Modell, kann die Ausgabe leicht geändert werden, ohne den Lösungsalgorithmus oder die Datenhaltung zu beeinflussen. Es können allerdings auch andere Lösungsalgorithmen entwickelt werden und der Controller, welcher diesen beinhaltet, einfach ausgetauscht werden. Damit ist das Programm offen für Erweiterung aber geschlossen gegenüber Änderungen. Ein Manko ist, dass die Eingabe nicht in einer separaten Klasse erstellt wurde, sondern gemeinsam mit der Ausgabe. Damit muss bei Änderungen in der Eingabe auch die Ausgabe neu erstellt werden. Da dieses Problem aber durch Vererbung umgangen werden kann, indem die Eingabe ein weiteres Interface implementiert, ist an dieser Stelle nicht von dem Konzept abgewichen wurden.

Das abspeichern der Spielsteine in einer Liste spart Zeit, weil bei der Werteabfrage bzw. bei Verwendung dieser nicht jedes mal der Typ konvertiert werden muss. Aufgrund dessen, dass nur eine mögliche Lösung gesucht werden muss, spart dies Zeit, da der Algorithmus mit finden der ersten möglichen Lösung endet.

Die Schwäche des Algorithmus liegt allerdings in sehr unterschiedlichen Steinen, wo der nächste passende Stein der letzte ungenutzte ist. Dort greifen die vorhandenen Abbruchbedingungen nicht bzw. nicht wirksam genug und die Berechnungsdauer steigt rapide an. An diese Stelle müssten bei weiterer Entwicklung noch Optimierungen vorgenommen werden.

5.2. Ausblick

Der Algorithmus kann noch verbessert werden, indem nicht für jeden ungenutzten Stein, jedes Feld ausprobiert wird, sondern nur für die ungenutzten, die die von den Nachbarfeldern geforderte Kombination enthalten. Dass heißt, dass nur die Steine probiert werden, deren Kantenziffern eine Folge beinhaltet, dass diese mit den Kantenziffern an den Nachbarfeldern übereinstimmen. Somit muss nicht mehr für jeden nicht legbaren Stein fünf Legeversuche durchgeführt werden, sondern nur noch für die Steine, die gelegt werden können. Dadurch kann selbst bei großen Abständen der Kartennummern für passende Karten der Algorithmus diese schnelle legen.

Als mögliche Erweiterungen für das Programm ist folgendes denkbar

- Eine grafische Oberfläche
 - Ausgabe des Puzzles in Fünfecken.
 - Ausgabe aller möglichen Lösungen des Puzzles.
 - Eingabe der Puzzleteile über die grafische Oberfläche.
 - Verwendung anderer geometrischer Figuren als Puzzleteile, wie z.B. Sechsecke. Hierfür müsste auch das Spielfeld entsprechend angepasst werden.
 - Die Anzahl verschiedener Kantenziffern kann erhöht bzw. verringert werden, um das Puzzle entsprechend schwieriger bzw. leichter zu gestalten.
 - Andere Algorithmen, mit denen Regeländerungen verbunden sein könnten, wie z.B. dass die Summe der aneinanderliegenden Kanten 3 ergeben muss. (Dieses Beispiel ist nur sinnvoll, wenn weitere Kantenziffern erlaubt sind.)
-

A. Abweichungen und Ergänzungen zum Vorentwurf

A.1. Abweichungen

Auf dem Programmkonzept wurden am Hauptalgorithmus die folgenden Modifikationen vorgenommen:

In der Implementierung sind die folgenden Modifikationen bei Datenstrukturen vorgenommen worden:

Bei der Behandlung der Sonder- und Fehlerfälle wurden keine Modifizierungen vorgenommen.

A.2. Ergänzungen

B. Benutzeranleitung

B.1. Verzeichnisstruktur der Abgabe

Im Prüfungsprodukt sind folgende Dateien und Verzeichnisse vorhanden:

Wurzelverzeichnis

- vorkompilierte Version des Programms
- Skript zum automatischen Ausführen des Programms mit mehreren Eingabedateien
- Skript zum Kompilieren des Programms
- Skript zum Löschen aller erstellten Ausgabedateien

Output Enthält die Ausgabedateien des Programms.

src Enthält den Quellcode des Programms.

Tests Enthält beispielhafte Eingabedateien.

Doku Enthält die Dokumentation.

B.2. Vorbereitung des Systems

B.2.1. Systemvoraussetzungen

Um das Programm verwenden zu können, wird ein Microsoft Betriebssystem in der Version 8 oder höher benötigt. Auf dem Betriebssystem muss die Windows PowerShell sowie eine Java Runtime Environment (JRE) der Version 8 oder höher installiert sein.

Da die PowerShell nach Angaben von Microsoft standardmäßig auf jedem Windows Rechner mit einem Betriebssystem von Windows 8 oder höher installiert ist, werden hierfür keine weiteren Schritte notwendig. Für die Anwenderinnen und Anwender, welche auf einem Unix basiertem System arbeiten, stehen entsprechende Shell-Skripts zur Verfügung.

Sollte die JRE nicht oder in einer veralteten Version installiert sein, kann dies unter der

folgenden Adresse heruntergeladen werden:

<https://www.java.com/de/download/>

PowerShell einrichten

Die PowerShell untersagt das Ausführen von fremden Skripten in den Standardeinstellungen. Damit dies geändert werden kann, muss die PowerShell als Administrator gestartet werden.

Um die PowerShell als Administrator zu starten, geht man in den Startmenüeintrag der PowerShell. Diesen finden Sie durch das Öffnen des Startmenüs und anschließender Eingabe von „PowerShell“. Dort machen Sie einen Rechtsklick auf „Windows PowerShell“, und klicken in dem aufkommenden Menü auf den Punkt „Als Administrator ausführen“. Je nach Sicherheitseinstellungen des Betriebssystems erscheint ein Fenster mit der Nachfrage, ob Änderungen durch das Programm zugelassen werden sollen, dies wird mit „Ja“ bestätigt. Gegebenenfalls müssen Sie auch Ihre Administratorrechte durch Angabe der Administrator-Benutzerdaten verifizieren, um fortfahren zu können. Anschließend öffnet sich die Windows PowerShell mit Administratorrechten.

Als erstes sollten die aktuellen Einstellungen mittels `Get-ExecutionPolicy` ausgelesen und notiert werden, damit die Einstellungen später wieder zurückgesetzt werden können. Im nächsten Schritt werden die Richtlinien geändert. Dies geschieht über den Befehl `Set-ExecutionPolicy RemoteSigned`. `RemoteSigned` bedeutet, dass Skripte, die aus dem Internet heruntergeladen wurden, signiert sein müssen, um ausgeführt zu werden. Lokal erstellte Skripte werden immer ausgeführt.

Falls die Skripte nicht ausgeführt werden sollten, müssen die Richtlinien weiter herabgesetzt werden, dies geschieht über `Set-ExecutionPolicy Unrestricted`. `Unrestricted` bedeutet, dass alle Skripte ausgeführt werden, jedoch wird für unsignierte Skripte, die aus dem Internet stammen, eine Warnung ausgegeben.

Die Richtlinien der PowerShell können in den ursprünglichen Zustand mittels des Befehls `Set-ExecutionPolicy <POLICY>` zurückgesetzt werden. Dabei ist `<POLICY>` der im ersten Schritt ausgelesene Wert. Somit könnten die Voreinstellungen beispielsweise wie folgt wieder aktiviert werden: `Set-ExecutionPolicy Restricted`

B.2.2. Installation

Der gesamte Inhalt der Abgabe ist in ein beliebiges und beschreibbares Verzeichnis zu kopieren. Danach ist das Programm betriebsbereit.

B.3. Kompilieren

sec:Kompilieren] Ein vorkompiliertes und ausführbares Programm liegt in Form von `GroPro.jar` der Abgabe bei. Soll das Programm wegen Änderungen im Quelltext, oder aus anderen Gründen neu kompiliert werden, kann mit dem Skript `compile.cmd` bzw. `compile.sh` das Programm neu kompiliert werden.

Damit dies erfolgreich ausgeführt werden kann, darf die vorliegende Verzeichnisstruktur nicht verändert werden, weil das Skript speziell auf diese Struktur angepasst ist. Weiter ist erforderlich, dass ein Compiler für Java (JDK) der Version 8 in der Umgebungsvariable `PATH` eingebunden ist. Ist der Compiler nicht in der Umgebungsvariable funktioniert der Compilerbefehl nicht.

Um die `PATH`-Variable zu erweitern, geht man im Startmenü mit einem Rechtsklick auf „Computer“ → „Eigenschaften“. Dort wird „Erweiterte Systemeinstellungen“ am linken Rand ausgewählt. Auf der Registerkarte „Erweitert“ wird der unterste Knopf „Umgebungsvariablen...“ gewählt. Im nächsten Schritt wählt man die `PATH`-Variable aus, dabei ist es egal, ob nun die Benutzer- oder die Systemvariable geändert wird. Jedoch werden zur Änderung der Systemvariable Administratorrechte benötigt. Nach der Auswahl der `PATH`-Variable wird auf den Knopf „Bearbeiten...“ gedrückt. Im Feld für den Wert der Variable wird am Ende folgendes angehängt:

```
; <Pfad zum JDK bin Ordner>
```

Falls ein JDK einer höheren Version vorhanden ist, kann auch dieses verwendet werden. Das es aufgrund von Sprachlevelunterschieden zu Problemen bei der Ausführung und dem Kompilieren kommt, kann nicht ausgeschlossen werden. Verwenden Sie in dem Fall am besten ein `openJDK1.8.x`.



Beim anfügen in die `PATH`-Variable ist es wichtig auf das Semikolon zu achten, da dieses als Trennzeichen agiert.

Wenn diese Bedingungen erfüllt sind, kann das Batch-/Shell-Skript `compile.cmd` bzw. `compile.sh` ausgeführt werden. Dabei wird automatisch die Datei `GroPro.jar` neu erstellt.

B.4. Programmaufruf

Um einzelne Dateien zu verarbeiten, müssen diese dem Programm `GroPro.jar` über die Kommandozeile oder PowerShell als Parameter mitgegeben werden. In beiden Fällen erstellt das Programm eine Ausgabedatei, die den ursprünglichen Dateinamen um „out“ im Fehlerfall um „err“ erweitert.

B.5. Testen der Beispiele

Sollen mehrere Testbeispiele verarbeitet werden, müssen diese in einem Verzeichnis zusammen gefasst werden. Durch das PowerShell-Skript `start.ps1` bzw. dem Shell-Skript `start.`  `sh` werden diese dann nacheinander verarbeitet. Die Skripte haben als optionale Parameter `DIR` und `TYP`. Über `DIR` kann ein Verzeichnis mit den Testfällen angegeben werden und über `TYP` die Endung der Dateien. Die default-Werte sind für `DIR` `./Tests` und für `TYP` `.`  `txt`. Diese werden verwendet, wenn keine anderen Werte übergeben werden. Ein Beispiel für den Aufruf ist:

```
./start.ps1 TYP=.input
```

Sollten Leerzeichen in einem Dateipfad vorhanden sein, so muss die Pfadangabe in Anführungszeichen stehen.

B.6. Fehlerfälle

Bei der Ausführung der Skript-Dateien kann es zu diversen Fehlern kommen. Im folgenden finden Sie eine kurze Übersicht über mögliche Fehlerfälle und deren Behebung:

Unbekannter Parameter Überprüfen Sie, ob der angegebene Parameter keine Tippfehler enthält.

Dies kann auch auftreten, wenn bei der Verwendung von Leerzeichen in einem Value keine Anführungszeichen um diesen gesetzt worden sind.

Pfad nicht gefunden (Programmausführung) Überprüfen Sie, ob der angegebene Pfad vorhanden ist. Sollte dies nicht der Fall sein, führen Sie das Skript erneut aus mit einem neuen Pfad. Hierfür verwenden Sie bitte den Parameter `DIR`. Andernfalls überprüfen Sie, ob Sie über Schreibberechtigungen auf diesem Verzeichnis verfügen.

Pfad nicht gefunden (Kompilieren) Beim auftreten dieses Fehlers, befindet sich kein Ordner `src` im selben Verzeichnis wie das Skript oder der `src`-Ordner wurde gelöscht. Bitte setzen Sie das Projekt auf den Auslieferungszustand zurück, damit die Source-dateien wieder vorhanden sind, und führen das Skript dann erneut aus.

Pfad nicht gefunden (cleanup) Der Ordner `Output` auf Ebene des Skriptes existiert nicht. Bitte erstellen Sie den Ordner `Output` oder setzen Sie das Projekt auf den Auslieferungszustand zurück.

Datei `GroPro.jar` nicht gefunden Sollte dieser Fehler auftreten, müssen Sie vor dem ausführen des Skripts `start.ps1` bzw. `start.sh` das Skript zum kompilieren ausführen (`compile.cmd` bzw. `compile.sh`)

Befehl `javac` falsch geschrieben / nicht gefunden Das JDK zum kompilieren konnte nicht gefunden werden. Dies kann daran liegen, dass das JDK nicht in der `PATH`-Variable enthalten ist, oder gar nicht installiert. Bitte befolgen Sie hierfür die Anweisungen im Bereich Kompilieren.

C. Entwicklungsumgebung

Programmiersprache	:	JAVA 8
Compiler	:	openJDK javac Version 1.8.0 ₂₁₂ – 3 – <i>redhat</i>
Rechner	:	Intel(R) Core(TM) i5-6300U 2,50GHz 32GB Arbeitsspeicher
Betriebssystem	:	Windows 10 Enterprise 1809 64 Bit-Betriebssystem

D. Verwendete Hilfsmittel

- Eclipse Photon
Entwicklungsumgebung für Java und andere Programmiersprachen
<https://eclipse.org>
- Java Platform, Standard Edition 8
API Specification
Internetseite mit Erklärungen und Hilfen rund um die Programmiersprache
[https://https://docs.oracle.com/javase/8/docs/api/](https://docs.oracle.com/javase/8/docs/api/)
- Notepad++
Open-Source-Texteditor für die Windows Desktopumgebung
<http://notepad-plus-plus.org/>
- Overleaf
Plattform unabhängiger LaTeX-Editor
<https://overleaf.com>
- Papeeria
Plattform unabhängiger LaTeX-Editor
<https://papeeria.com>
- Structorizer
Plattform unabhängiges Programm zur Erzeugung von Nassi-Shneiderman-Diagrammen
<http://structorizer.fisch.lu/>
- diagrams.net
Online UML-Diagramm Editor
<https://draw.io>
- UMLet
UML-Diagramm Editor als Eclipse-Plugin
<https://www.umlet.com/changes.htm>

E. Erklärung

Erklärung des Prüfungsteilnehmers / der Prüfungsteilnehmerin:

Ich versichere durch meine Unterschrift, dass ich das Prüfungsprodukt selbstständig und ohne fremde Hilfe angefertigt und alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe. Die Arbeit hat in dieser Form keiner anderen Prüfungsinstitution vorgelegen.

Das abgegebene Prüfungsprodukt entspricht der gedruckten Version.

Köln, den 29. April 2020

Ort und Datum

Unterschrift des Prüfungsteilnehmers

F. Aufgabenstellung

G. Quellcode

G.1. Source Dateien

G.1.1. Default Package

Main.java

```
1  /**
   * Die Klasse {@code Main} dient dem Einstieg in das Programm.
   * Hier werden die notwendigen Klassen aufgerufen, die zur
   *                               ↪ Bearbeitung des Problems notwendig sind.
   *
   5  * @author Lukas Dahlberg
   * @version 1.0
   */
public class Main{

10     public static void main(String[] args){
        if(args.length > 2){
            for(int i = 2; i < args.length; ++i){
                // Aufruf des Controller Interface mit der bergabe der
                *                               ↪ Eingabedatei in args[i]
            }
        }
15     }
}

}
```

G.1.2. io

InvalidSyntaxException.java

```
1 package io;
2
3 /**
4  * {@code InvalidSyntaxException} ist eine dedizierte Exception,
5  * ↪ welche genutzt werden soll, ↪
6  * wenn es zu syntaktischen Fehlern in der Eingabe kommt.
7  *
8  * @author Lukas Dahlberg
9  * @version 1.0
10 */
11 public class InvalidSyntaxException extends RuntimeException {
12
13     private static final long serialVersionUID = 1L;
14
15     public InvalidSyntaxException(String msg) {
16         super(msg);
17     }
18 }
```

IInput.java

```
1 package io;  
2  
3 public interface IInput {  
4     public Object readData();  
5  
6 }  
7
```

FileInput.java

```
1 package io;
3 /**
 * Die Klasse {@code DateiEingabe} dient dem Einlesen der
 *      ↳ Eingabedatei.
 * Hier wird auf syntaktische Fehler geprüft, welche gem Konzept
 *      ↳ behandelt werden.
 *
 * @author Lukas Dahlberg
8  * @version 1.0
 */
public class FileInput implements IInput {

    private String comment;
13    private String filename;
    private String path;

    public FileInput(String file) {
        // TODO implement
18    }

    @Override
    public Object readData() {
        return null;
23    }

    private void setComment(String comment) {
        if(comment == null || comment.trim().length() == 0) {
            return; // TODO throw Exception
28        }
        this.comment = comment;
    }

    public String getComment() {
33        return comment;
    }

    private void setFilename(String filename) {
        if(filename == null || filename.trim().length() == 0) {
            return; // TODO throw Exception
38        }
        this.filename = filename;
    }

    public String getFilename() {
43        return filename;
    }
}
```

```
48     private void setPath(String path) {  
        if(path == null || path.trim().length() == 0) {  
            return; // TODO throw Exception  
        }  
        this.path = path;  
53     }  
  
    public String getPath() {  
        return path;  
    }  
58 }
```

IOutput.java

```
1 package io;
2
3 public interface IOutput {
4
5     /**
6      * Gibt eine Fehlermeldung aus, mit Bezug auf den Ort des
7      * ↪ Auftretts. ↪
8      *
9      * @param errorMessage Die auszugebende Fehlermeldung
10     * @param row Die Zeilen-Nr. der fehlerverursachenden Zeile
11     */
12     void writeError(String errorMessage, int row);
13
14     /**
15      * Gibt die Lösung des Problems aus.
16      */
17     void writeOutput();
18 }
```

FileOutput.java

```

1 package io;
2
3 import java.io.File;
4 import java.io.FileNotFoundException;
5 import java.io.PrintWriter;
6
7 /**
8  * Die Klasse {@code DateiAusgabe} dient dem ausgeben der Antwort
9  *   ↳ in Form einer Datei.
10  * Der Name der Ausgabedatei setzt sich zusammen aus dem Namen der
11  *   ↳ Eingabedatei
12  * und einer Dateiendung, welche bestimmt ist durch die
13  *   ↳ schreibende Methode {@link FileOutputStream#
14  *   ↳ writeOutput(String)}
15  * bzw. {@link FileOutputStream#writeError(String, String, int)}
16  *
17  * Die Ausgabedatei wird immer in das {@value #PATH} Verzeichnis
18  *   ↳ geschrieben.
19  *
20  * @author Lukas Dahlberg
21  * @version 1.0
22 */
23 public class FileOutput implements IOutput {
24
25     private static final String PATH = "./Output/";
26     private static final String INOUTLINE = "
27     ↳ *****"
28     ↳ ";
29
30     private static final String BEGINENDLINE = "*****";
31     private String filename;
32
33     public FileOutput(String filename) {
34         setFilename(filename);
35     }
36
37     public void setFilename(String filename) {
38         if(filename == null || filename.trim().length() == 0) {
39             throw new IllegalArgumentException("Kein Dateiname
40             ↳ bergeben worden.");
41         }
42         this.filename = filename;
43     }
44
45     public String getFilename() {
46         return filename;
47     }
48 }

```



```
82      * @see IOutput#writeOutput(String)
      *
      * @throws RuntimeException Wenn die Ausgabedatei nicht
      *       ↳ geschrieben werden kann.
      */
      @Override
      public void writeOutput() {
87          //TODO implement Ausgabe
          try(PrintWriter printer = new PrintWriter(new File(PATH +
      *       ↳ filename + ".out"))){
              printer.flush();
          } catch (FileNotFoundException e) {
              throw new RuntimeException("Fehler_↳beim_↳schreiben_↳der_↳
      *       ↳ Ausgabedatei");
          }
92      }
  }
}
```

G.2. Skript Dateien

G.2.1. Windows

Kompilieren

```
1  :: Skript zum kompilieren
   @echo off

   :: Environment-Variablen setzen
   CALL ./setEnvs.cmd

6  :: Standardwerte in lokalen Variablen setzen
   set prog="GroPro.jar"
   set srcdir="./src"

11 :: Ueberpruefen ob Ordner vorhanden ist und Dateien kompilieren
   if not exist %srcdir% (
       GOTO ERRORSECTION
   )
   dir /s /B *.java > sources.txt
16  javac @sources.txt -d ./bin
   jar -cvfe %prog% Main ./bin
   del sources.txt
   echo "Programm_kompiliert"
   GOTO SUCCESSFULL

21 :ERRORSECTION
   echo "_____FEHLER_____"
   echo "Pfad_fuer_Sourcecode-Dateien_wurde_nicht_gefunden!"
   echo "Bitte_ueberpruefen_Sie_ob_der_Pfad"
26  echo %srcdir%
   echo "existiert_und_fuehren_das_Skript_erneut_aus."

   :SUCCESSFULL
```

Output leeren

```
1 $outdir="./Output"
  $filetyp=".out"

#Ueberprueft ob Ordner mit Ausgaben vorhanden ist
  if(Test-Path "$outdir")
6   {
      #Nimmt alle Dokumente mit gewuenschter Endung aus dem
      ↳Ordner fuer die Ausgabe
      $files = Get-ChildItem $outdir | Where { $_.Extension -eq ↳
      ↳$filetyp }
      foreach($file in $files)
      {
11         if($file)
            {
                echo("Loesche:_" + ($file.FullName))
                rm $file.FullName
            }
16         else
            {
                echo "-----WARNUNG-----"
                echo "Ordner_fuer_die_Ausgabedateien_ist_leer!"
                echo "Bitte_ueberpruefen_Sie_ob_der_Ordner"
21                echo "$outdir"
                echo "Ausgabedateien_enthaelt_und_fuehren_das_↳
                ↳Skript_erneut_aus."
                echo "Die_Ausgabedateien_enden_alle_auf_.out!"

            }
        }
26        echo("Alle_Dateien_geloescht")
    }
    else
    {
31        echo "-----FEHLER-----"
        echo "Pfad_fuer_die_Ausgabedateien_wurde_nicht_gefunden!"
        echo "Bitte_ueberpruefen_Sie_ob_der_Pfad"
        echo "$outdir"
        echo "existiert_und_fuehren_das_Skript_erneut_aus."

    }
}
```

Programm ausführen

```

1 #Skript zum ausfuehren aller Testfaelle

#Standardwerte setzen
$prog = "GroPro.jar"
5 $dir = "./Tests"
$typ = ".txt"

#Ueberpruefen gesetzter Parameter
for($i=0; $i -lt $args.count; $i++)
10 {
    $a = ([string] $args[$i]).split("=")
    #Abfragen ob ein neues Verzeichnis für die Testfaelle
    #angegeben ist
    if($a[0].ToUpper() -eq "DIR")
    {
15         $dir=$a[1]
    }
    #Abfrage ob Testfaelle einem neuen Dateityp entsprechen
    elseif($a[0].ToUpper() -eq "TYP")
    {
20         $typ=$a[1]
    }
    else
    {
        echo "-----FEHLER-----"
25         echo "Unbekannter_Parameter:" $a[0].ToUpper()
        echo "Bitte_ueberpruefen_Sie_die_richtige_Schreibweise_des_
        #Parameters."
        echo "Erlaube_Parameter_sind:"
        echo "DIR_Angabe_eines_Verzeichnisses_in_welchem_
        #Eingabedateien_liegen_(Standardwert:./Tests)"
        echo "TYP_Angabe_der_Dateiendung_der_Eingabedateien_(
        #Standardwert:.txt)"
30         echo "Ausfuehrung_wird_beendet."
        exit
    }
}

35 #Ueberprueft ob Programm vorhanden ist
if(Test-Path "$prog")
{
    #Ueberprueft ob Ordner mit Testfaellen vorhanden ist
    if(Test-Path "$dir")
40 {
        #Nimmt alle Dokumente mit gewuenschter Endung aus dem
        #Ordner fuer Testfaelle

```

```

$files = Get-ChildItem $dir | Where { $_.Extension -eq ↵
↵$typ }
foreach($file in $files)
{
    45     if($file)
    {
        echo("—>" + ($file.FullName) + "<—")
        & .\ $prog $file.FullName
    }
    50     else
    {
        echo "—————WARNUNG—————"
        echo "Ordner_fuer_Testfeaele_ist_leer!"
        echo "Bitte_ueberpruefen_Sie_ob_der_Ordner"
        55     echo "$dir"
        echo "Testfaelle_enthaelt_und_fuehren_das_Skript_↵
↵erneut_aus."
        exit
    }
    }
    60     echo("Alle_Dateien_ausgefuehrt.")
}
else
{
    echo "—————FEHLER—————"
    65     echo "Pfad_fuer_Testfeaele_nicht_gefunden!"
    echo "Bitte_ueberpruefen_Sie_ob_der_Pfad"
    echo "$dir"
    echo "existiert_und_fuehren_das_Skript_erneut_aus."
}
70 }
else
{
    echo "—————FEHLER—————"
    echo "Datei_GroPro.jar_wurde_nicht_gefunden!"
    75     echo "Bitte_ueberpruefen_Sie_ob_das_Skript"
    echo "im_selben_Verzeichnis_ist_wie_die_Datei_GroPro.jar"
    echo "oder_kompilieren_Sie_das_Programm_erneut."
}

```

G.2.2. MacOS und Linux

Kompilieren

Output leeren

Programm ausführen

H. In- und Output der Testdokumentation

H.1. Normalfall

H.1.1. IHK-Beispiel 1

Eingabedatei:

Ausgabedatei:

H.2. Sonderfall

H.3. Fehlerfall

