\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Dokumentation der praktischen Arbeit  
zur Prüfung zur  
mathematisch-technischen Softwareentwicklerin

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

21. April 2022

**Johanna Schindler**

Prüfungs-Nummer: 142 - 18741

Programmiersprache: Java

Inhaltsverzeichnis // TODO aktualisieren

[1 Aufgabenanalyse 4](#_Toc101427341)

[1.1 Allgemeines 4](#_Toc101427342)

[1.2 Eingabe 4](#_Toc101427343)

[1.3 Problemstellung 4](#_Toc101427344)

[1.4 Ausgabe 2](#_Toc101427345)

[1.5 Grenz- und Fehlerfälle 2](#_Toc101427346)

[1.5.1 Grenzfälle 2](#_Toc101427347)

[1.5.2 Fehlerfälle 2](#_Toc101427348)

[1.6 Anforderungsliste 3](#_Toc101427349)

[2 Verfahrensbeschreibung 3](#_Toc101427350)

[2.1 Eingabe und Initialisierung 3](#_Toc101427351)

[2.2 Berechnung der … 4](#_Toc101427352)

[2.3 Ausgabe 4](#_Toc101427353)

[3 Programmbeschreibung 4](#_Toc101427354)

[3.1 Grobentwurf 5](#_Toc101427355)

[3.1.1 Module 5](#_Toc101427356)

[3.1.2 Ein- und Ausgabe 5](#_Toc101427357)

[3.1.3 Model 6](#_Toc101427358)

[3.1.4 Controller 6](#_Toc101427359)

[3.2 Ablauf 6](#_Toc101427360)

[3.2.1 Gesamtablauf 6](#_Toc101427361)

[3.2.2 Initialisierung des Programms 6](#_Toc101427362)

[3.2.3 Strategie 7](#_Toc101427363)

[3.3 Algorithmen // TODO wenn Plural unterteilen 7](#_Toc101427364)

[3.4 Ein- und Ausgabekonvertierung 7](#_Toc101427365)

[3.4.1 Eingabekonvertierung 7](#_Toc101427366)

[3.4.2 Ausgabekonvertierung 7](#_Toc101427367)

[4 Testdokumentation 7](#_Toc101427368)

[4.1 Begründung und Diskussion der Testfälle 7](#_Toc101427369)

[4.1.1 Grenzfalltests 7](#_Toc101427370)

[4.1.2 Fehlertests 8](#_Toc101427371)

[4.1.3 Technische Fehler 8](#_Toc101427372)

[4.1.4 Normalfälle 9](#_Toc101427373)

[4.2 Ausführliches Beispiel 9](#_Toc101427374)

[4.2.1 Eingabe 9](#_Toc101427375)

[4.2.2 Algorithmen // TODO wenn mehrere unterteilen 9](#_Toc101427376)

[4.2.3 Ausgabe 9](#_Toc101427377)

[5 Zusammenfassung und Ausblick 10](#_Toc101427378)

[5.1 Zusammenfassung 10](#_Toc101427379)

[5.2 Ausblick 10](#_Toc101427380)

[5.2.1 Ein- und Ausgabe 10](#_Toc101427381)

[5.2.2 Strategien TODO 10](#_Toc101427382)

[6 Abbildungsverzeichnis 10](#_Toc101427383)

[A Abweichungen und Ergänzungen zum Vorentwurf 11](#_Toc101427384)

[B Benutzeranleitung 11](#_Toc101427385)

[a. Verzeichnisstruktur 11](#_Toc101427386)

[b. Systemvorraussetzungen 11](#_Toc101427387)

[c. Installation 11](#_Toc101427388)

[d. Kompilieren 11](#_Toc101427389)

[e. Programmausführung 12](#_Toc101427390)

[f. Testausführung 12](#_Toc101427391)

[C Entwicklerdokumentation 12](#_Toc101427392)

[a. Verfügbare Klassen und Schnittstellen 12](#_Toc101427393)

[b. Nutzungshinweise 12](#_Toc101427394)

[c. Fehlerbehandlung 12](#_Toc101427395)

[D Entwicklungsumgebung 12](#_Toc101427396)

[E Verwendete Hilfsmittel 13](#_Toc101427397)

[F Erklärung 13](#_Toc101427398)

**// TODO deleteme**

**Legende der Farben:**

// TODO vorher

// TODO in GroPro

# Aufgabenanalyse

## Allgemeines

// TODO

Im Rahmen dieser Arbeit soll ein Software-System entwickelt werden, welches

Das System nimmt… entgegen und bestimmt …

Das Software-System soll nach dieser Eingabe im Folgenden … ermitteln und … zurückgeben.

## Eingabe

// TODO Form und Format

Dazu müssen zunächst die Eingabeparameter eingelesen werden. Dies soll, da in der Aufgabenstellung nicht explizit angegeben, in Textdateien im Textformat stattfinden, welche die Dateiendung .txt aufweisen müssen. Alle Zeilen, welche mit einem Semikolon beginnen, werden als Kommentar betrachtet und bis auf die Beschreibung des Testbeispiels in den ersten drei Zeilen der Datei ignoriert. Auch Leerzeilen werden vom Programm übersprungen.

Eine gültige Eingabedatei („ihk\_1.txt“) könnte wie folgt aussehen: // TODO

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
; Testbeispiel von Abbildung 1  
; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
; Groesse des Gebietes  
5 4  
; Hoehen  
4 5 3 2 1  
3 4 2 0 4  
2 2 4 3 2  
4 2 1 2 1

Abbildung 1: Beispiel einer gültigen Eingabedatei

// TODO

Zum Initialisieren des Programms wird die Eingabedatei zeilenweise eingelesen und die vorher beschriebenen Werte aus den Zeilen ausgelesen, geprüft und in die vorgesehene Datenstruktur überführt. Im Anschluss werden mithilfe eines … Algorithmus … bestimmt.

## Problemstellung

// TODO Was macht der Algorithmus?

## Ausgabe

// TODO

Nach Beendigung des Algorithmus soll eine Ausgabe des Ergebnisses auf zwei verschiedene Weisen erzeugt werden.

1. Ausgabe auf dem Monitor

Die Ausgabe von dem Beispiel in Abbildung 1 könnte dabei so aussehen:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Testbeispiel von Abbildung 1  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Ausdehnung in X: 5  
Ausdehnung in Y: 4  
Eingelesene Hoehen:  
4 5 3 2 1  
3 4 2 0 4  
2 2 4 3 2  
4 2 1 2 1  
Benoetigte Antennen: 1  
Antenne 1: 2 2

Abbildung 2: Beispiel einer Ausgabe in der Konsole

1. Ausgabe in eine Textdatei

Der Inhalt der Ausgabe in eine Textdatei ist identisch mit dem in a) beschriebenen und in Abbildung 2 beispielhaft aufgeführten. Die zu erstellende Ausgabetextdatei erhält den Namen der Eingabedatei mit dem Suffix „\_Ausgabe.txt“.

## Grenz- und Fehlerfälle

Während des Programmablaufs, das heißt beim Einlesen, Initialisieren, Durchführen und Ausgeben können verschiedene Grenz- und Fehlerfälle auftreten, die im Folgenden beschrieben werden.

// TODO

### Grenzfälle

1. Das kleinstmögliche (sinnvolle) Feld der Größe 2x2 wird eingegeben.
2. Das größtmögliche Feld der Größe // TODO wird eingegeben

### Fehlerfälle

Die auftretenden Fehler können üblicherweise in drei Fehlerarten aufgeteilt werden. Man unterscheidet zwischen technischen, syntaktischen und semantischen Fehlern.

#### Technische Fehler

Technische Fehler können beim Zugriff auf die Ein- und Ausgabedateien auftreten. Dabei können folgende Fehler auftreten:

1. Es wurde keine Eingabedatei angegeben.
2. Die Eingabedatei existiert nicht.
3. Die Eingabedatei ist nicht lesbar.
4. Die Ausgabedatei ist nicht beschreibbar

Da das Anlegen einer Ausgabedatei bei technischen Fehlern unter Umständen nicht möglich ist, werden diese auf der Standardfehlerausgabe des Systems ausgegeben.

#### Syntaktische Fehler

// TODO

Syntaktische Fehler liegen vor, wenn die Eingabedatei nicht das vorgegebene Format und in Abschnitt 1.2 beschriebene hat. (Die Angaben der Zeilennummern beziehen sich immer auf eine Betrachtung der Eingabe inklusive der Kommentarzeilen, abgezogen jeglicher auftretender Leerzeilen.)

1. Ein Kommentar ist nicht durch vorangestelltes „;“ gekennzeichnet.
2. Die Eingabedatei ist leer.

Syntaktische Fehler erzeugen eine Fehlermeldung, welche anstatt der eigentlichen Programmausgabe in die Konsole und in die Ausgabedatei geschrieben wird.

#### Semantische Fehler

1. // TODO

Semantische Fehler erzeugen ebenfalls eine Fehlermeldung, welche anstatt der eigentlichen Programmausgabe in die Konsole und in die Ausgabedatei geschrieben wird.

## Anforderungsliste

// TODO

// TODO Vorbereitung

# Verfahrensbeschreibung

// TODO Verweis auf Anforderungsliste

Da die Aufgabe nun detailliert beschrieben und analysiert wurde, wird im Folgenden der Ablauf des Programms näher erläutert, wobei technische Details verfeinert und Abläufe konkretisiert werden.

## Eingabe und Initialisierung

// TODO Welche **logischen** Datenstrukturen (sprachenunabh.) und warum? zB Feld fester Größe oder verkettete Liste

Nach dem Start des Programms überprüft dieses das Vorhandensein der angegebenen Eingabedatei. Ist diese vorhanden, wird sie zeilenweise eingelesen. Sollten technische Fehler beim Einlesen auftreten, wird das Programm abgebrochen und eine Fehlermeldung ausgegeben. Die Eingabedatei wird als Parameter dem Programmaufruf mitgegeben.

Die Datei wird dann zeilenweise eingelesen, wobei Kommentar- und Leerzeilen übersprungen werden.

Wurden also keine syntaktischen oder semantischen Fehler wie in Abschnitt 1.5.2 beschrieben festgestellt, so wurden das Feld mit folgenden Werten initialisiert:

## Berechnung der …

// TODO evtl unterteilung in versch Strategien

Nach Beendigung der Eingabe und der Initialisierung der Dateistrukturen wird das erzeugte … an … übergeben, der … übernimmt.

// grober Ablauf -> TODO warum rekursiv/iterativ?

// dann genaue Beschreibung der Rekursion o.Ä.

// mathematische Formeln

// Skizzen

Die genaue Vorgehensweise beim … ist wie folgt:

## Ausgabe

// TODO Konvertierung und Ausgabe in gewünschte Formate

Anschließend an den Algorithmus kann das Ergebnis auf die zwei gewünschten Arten ausgegeben werden. Das Verfahren ist simpel:

# Programmbeschreibung

// TODO Verweis auf Anforderungsliste

// TODO Diagramme soweit es geht vorbereiten

**Anmerkung zu deutschen und englischen Begriffen**

Eigene Bezeichner, insbesondere für inhaltliche Elemente, wurden deutsch benannt. Vorkommende englischsprachige Bezeichner wurden verwendet, wenn dies geläufige Begriffe in der Softwareentwicklung sind, da so die Aufgabe der entsprechenden Komponente allgemein ersichtlich ist, beispielsweise bei Reader, Converter oder get/set/is.

## Grobentwurf

Das Programm soll, nach Ausklammerung von Ein- und Ausgabe durch eine Model-View-Controller Architektur umgesetzt werden, welche im folgenden Paketdiagramm grob skizziert wird.

### Module

// TODO MVC abgrenzen

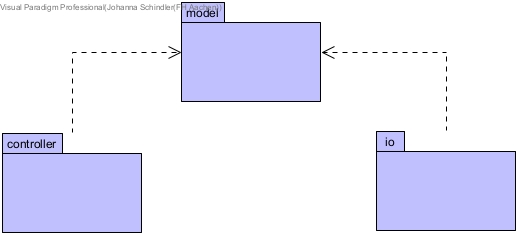


Abbildung : Module im Paketdiagramm

Das Model beinhaltet dabei die benötigten Daten in strukturierter Form, also …

Der Controller beinhaltet den Algorithmus, welcher auf den Daten des Models operiert. Eine klassische View existiert in diesem Programm nicht, da keine visuelle Anzeige des Programms in der Aufgabenstellung gefordert war. Stattdessen gibt es das Modul IO, welches für das Ein- und Auslesen sowie das Konvertieren der Daten in die Model-Klassen zuständig ist. Das Model wird also von der Eingabe erzeugt.

Die Beziehungen zwischen den Paketen wurden der Übersicht halber nicht in den UML-Klassendiagrammen aufgeführt.

### Ein- und Ausgabe

// TODO IOConverter Trennung zu Controller, Strategy Pattern

Um auch andere Ein- und Ausgabewege als die in der Aufgabenstellung Gewünschten zu ermöglichen, werden Ein- und Ausgabe sowie die Konvertierer durch jeweils ein Interface abstrahiert.

Der FeldReader liest beliebige Eingabedateien mit der Endung „.txt“ ein, wenn keine technischen Fehler auftreten und lässt dabei Leerzeilen aus. Hier findet noch keine Syntaxüberprüfung statt. Er gibt den gesamten Inhalt der Datei zurück. Der InputConverter konvertiert diesen Inhalt in ein Feld unter Berücksichtigung der Syntaxregeln. Da von dieser Klasse nicht zwingend ein Objekt erstellt werden soll, sondern eine statische Methode im Folgenden für alle möglichen Textdateiinhalte die Konvertierung übernimmt, wird sie als static gekennzeichnet.

Der FeldWriter delegiert das Result an den OutputConverter, der je nach gewünschtem Ausgabetyp die Konsolenausgabe, die Datei oder die Dateien erstellt. Auch dieser ist als statische Klasse implementiert, sodass das Konvertieren in jegliche Form über die Klasse selbst und ohne Konvertierer-Objekt geschehen kann. Die Form der Ausgabe ist entkoppelt von dem FeldWriter und im OutputConverter gekapselt. Die unterschiedliche Herangehensweise bei FeldReader und -Writer ist dadurch begründet, dass die Eingabe immer im Textdateiformat geschieht, die Ausgabe jedoch in verschiedene Medien und in verschiedenen Formaten geschehen soll. Das Ausgabeformat bleibt dadurch austauschbar, solange der Ausgabetyp bekannt ist. Die verschiedenen Ausgabetypen werden durch die Enumeration KnzOutputTyp benannt, da es sich hierbei um eine feste Anzahl disjunkter Ausgabetypen handelt.

### Model

Das Model beinhaltet …

// wofür werden die gebraucht

### Controller

// TODO Strategy Pattern für Alg

Das Interface IAlgorithmus abstrahiert die Schnittstelle, welche die Klasse BacktrackingAlgorithmus und alle weiteren in der Zukunft denkbaren Algorithmen zur … bereitstellen müssen. Dabei handelt es sich um einen Konstruktor, der das zu bearbeitende Feld erhält und die Ausführ-Methode, die im Falle des Backtrackings eine private, rekursive Methode aufruft und dann das Result zurückgibt. BacktrackingAlgorithmus hat dabei außerdem weitere Hilfsmethoden, die zur Berechnung … benötigt werden. Die Klasse Programm übernimmt den kompletten Programmablauf, das heißt sie steuert nach Aufruf der Methode starteProgramm() die Eingabe, das Starten des Algorithmus und die Ausgabe.

## Ablauf

### Gesamtablauf

Der Ablauf des Programms lässt sich wie folgt als UML-Sequenzdiagramm beschreiben. Die Main übernimmt dabei die Aufgabe, das eigentliche Programm mit dem Eingabedateinamen zu starten und ist an sich trivial, sodass auf ein Klassendiagramm verzichtet wurde.

Zunächst wird der Gesamtablauf der Programmausführung vereinfacht dargestellt. Vor allem die Initialisierung und die Abläufe innerhalb des Backtracking-Algorithmus sind komplexer als hier dargestellt. Diese werden daher anschließend noch mittels Aktivitätsdiagrammen präzisiert, um die Sequenzdiagramme möglichst übersichtlich und verständlich zu halten.

//TODO SD

### Initialisierung des Programms

//TODO SD

### Strategie

//TODO SD

## Algorithmen // TODO wenn Plural unterteilen

//TODO AD

## Ein- und Ausgabekonvertierung

//TODO ADs

### Eingabekonvertierung

### Ausgabekonvertierung

# Testdokumentation

In diesem Kapitel wird das Programm an einem ausgesuchten Beispiel ausführlich getestet, sowie die weiteren Testbeispiele erklärt und begründet.

## Begründung und Diskussion der Testfälle

Die Testfälle lassen sich anhand des Präfix des Dateinamens kategorisieren.

|  |  |
| --- | --- |
| **n\_** | Tests für einfache Normalfälle, bei denen … |
| **g\_** | Tests für in Unterabschnitt 1.5.1 beschriebene Grenzfälle |
| **syn\_** | Tests für in Unterabschnitt 1.5.21.5.1 beschriebene syntaktische Fehlerfälle |
| **sem\_** | Tests für in Unterabschnitt 1.5.21.5.1 beschriebene semantische Fehlerfälle |
| **ihk\_** | vorgegebene Testbeispiele der Aufgabenstellung, beinhalten Normal- und Sonderfälle |

Die Nummerierung der Grenz- und Fehlerfälle entspricht der in Abschnitt 1.5 verwendeten.

### Grenzfalltests

Die in Abschnitt 1.5.1 beschriebenen Grenzfälle werden durch die folgenden Tests geprüft.

|  |  |
| --- | --- |
| **Grenzfall 1** | wird durch die Datei g1\_minimal.txt getestet.  Die Eingabe des kleinstmöglichen Feldes soll als gültig erkannt werden und …. (trivial) |
| **Grenzfall 2** | wird durch die Datei g2\_maximal.txt getestet.  Die Eingabe des größtmöglichen Feldes soll als gültig erkannt werden…. |

### Fehlertests

Die in Abschnitt 1.5.2 beschriebenen syntaktischen und semantischen Fehlerfälle werden durch die folgenden Tests geprüft.

// TODO Beschreibung „Der Test soll zeigen dass“

|  |  |
| --- | --- |
| **Syntaktischer Fehler 1** | wird durch die Datei syn1\_kommentarFehlt.txt getestet. |
| **Syntaktischer Fehler 2** | wird durch die Datei syn2\_kommentarOhneSemikolon.txt getestet. |

### Technische Fehler

Die in Abschnitt 1.5.2.1 beschriebenen technischen Fehlerfälle lassen sich aus technischen Gründen nicht als Ein- und Ausgabedateien der Abgabe beifügen. Aus diesem Grund werden hier die entsprechenden Aufrufe und die resultierende Ausgabe aufgeführt.

// TODO konkrete Fehlermeldungen benennen

|  |  |
| --- | --- |
| **Technischer Fehler 1** | Der Test soll zeigen, dass das Programm das Fehlen einer Eingabedatei erkennt und den Benutzer mit einer passenden Fehlermeldung informiert.  java -jar ./run/Programm.jar |
| **Technischer Fehler 2** | Der Test soll zeigen, dass das Programm erkennt, dass die angegebene Eingabedatei nicht existiert und den Benutzer mit einer passenden Fehlermeldung informiert.  java -jar ./run/Programm.jar existiertNicht.txt |
| **Technischer Fehler 3** | Der Test soll zeigen, dass das Programm erkennt, dass die Leserechte für die angegebene Eingabedatei fehlen und den Benutzer mit einer passenden Fehlermeldung informiert.  // TODO |
| **Technischer Fehler 4** | Der Test soll zeigen, dass das Programm erkennt, dass die Schreiberechte zur passenden Ausgabedatei zur Eingabedatei fehlen und die Ausgabe nicht geschrieben werden kann. Darüber soll der Benutzer mit einer passenden Fehlermeldung informiert werden.  // TODO |

### Normalfälle

Normalfälle sind zum einen durch die Beispiele aus der Aufgabenstellung gegeben, welche durch die Eingabedateien „ihk\_1.txt“, „ihk\_2.txt“, „ihk\_3.txt“ getestet wurden. Weitere Normalfälle sind //TODO.

// TODO detaillierte Diskussion

**IHK-Beispiel 1**

**IHK-Beispiel 2**.

**IHK-Beispiel 3**

Die weiteren Beispiele // TODO Hier soll getestet werden, ob eine einfache Lösung auch bei größeren Feldern schnell gefunden werden kann. // TODO Ergebnis?

## Ausführliches Beispiel

Als ausführliches Beispiel wird die Eingabedatei … verwendet, da ...

### Eingabe

// TODO

### Algorithmen // TODO wenn mehrere unterteilen

// TODO

### Ausgabe

// TODO

# Zusammenfassung und Ausblick

## Zusammenfassung

Das Programm erfüllt die in der Aufgabenstellung definierten Anforderungen und wurde in Bezug auf Funktionalität und Fehlererkennung ausführlich getestet. Durch die hohe Laufzeitkomplexität des Backtracking-Algorithmus ist die Bearbeitung großer Felder jedoch sehr rechenaufwendig, wodurch das Testen des Grenzfalls der maximalen Fläche lediglich der Größe // TODO möglich war.

## Ausblick

Für das Programm gibt es diverse Erweiterungsmöglichkeiten in verschiedenen Bereichen.

### Ein- und Ausgabe

// TODO IO & Controller Trennung, Strategy Pattern

Die bisherige Ein- und Ausgabe über Textdateien im vorgegebenen Format lässt sich leicht um weitere Formate erweitern. Dies ist durch zusätzliche Implementierungen der Interfaces IReader, IWriter und IInputConverter und IOutputConverter möglich.

Denkbar wären

|  |  |
| --- | --- |
| **Ausgabe als XML** | Dazu müsste die Implementierung von IOutputConverter einen String mit XML erzeugen und in eine „.xml“ Datei schreiben. |
| **Ausgabe in eine Datenbank** | Dazu müsste die Implementierung von IOutputConverter ein SQL-Query erzeugen, welches die Implementierung von IWriter dann an die Datenbank absetzt. |

### Strategien TODO

// TODO Strategy Pattern

Neben der verwendeten Backtracking-Strategie wäre noch eine Vielzahl weiterer möglich. Diese müssten ebenfalls das Interface IStrategie implementieren.

Zwar ermittelt der Backtracking-Algorithmus in jedem Fall das bestmögliche Ergebnis, jedoch ist er sehr rechenaufwendig. Gerade bei größeren Flächen kann es daher sinnvoll sein, auf andere Verfahren auszuweichen, welche zu Lasten der bestmöglichen Lösung schneller ein Ergebnis erzeugen.

# Abbildungsverzeichnis // TODO aktualisieren

[Abbildung 1: Beispiel einer gültigen Eingabedatei 1](file:///C:\Users\Johanna\Documents\FH\6_Semester\IHK\GroPro\GroPro_Dokumentation_Vorlage.docx#_Toc101270722)

[Abbildung 2: Beispiel einer Ausgabe in der Konsole 2](file:///C:\Users\Johanna\Documents\FH\6_Semester\IHK\GroPro\GroPro_Dokumentation_Vorlage.docx#_Toc101270723)

[Abbildung 6: Module im Paketdiagramm 4](#_Toc101270724)

1. Abweichungen und Ergänzungen zum Vorentwurf
2. Benutzeranleitung
3. Verzeichnisstruktur

Im angegebenen Archiv befinden sich folgende Verzeichnisse:

|  |  |
| --- | --- |
| **run** | beinhaltet Skripte zum Kompilieren des Programms und zum Ausführen der Testfälle, sowie das kompilierte Programm |
| **src** | enthält den Quellcode des Programms |
| **doc** | enthält diese Dokumentation, sowie die Java API-Dokumentation, auf die in Anhang C verwiesen wird |
| **testfaelle** | beinhaltet die ausgeführten Testbeispiele als Ein- und Ausgabedateien |

1. Systemvorraussetzungen

Um das Programm verwenden zu können, wird ein Microsoft Betriebssystem in der Version 10 oder höher, sowie eine Java Laufzeitumgebung (JRE) in der Version 16 benötigt. Für das Kompilieren des Programms muss zusätzlich das Java Development Kit (JDK) in der Version 16 oder höher installiert sein.

Sollten diese nicht oder in veralteter Version installiert sein, kann ein Download der JRE und JDK in gewünschter Version unter der folgenden Adresse geschehen:

<https://www.oracle.com/java/technologies/downloads/>

// TODO maven herunterladen und Version

1. Installation

Der gesamte Inhalt der vorliegenden .zip-Datei ist in ein beliebiges, beschreibbares Verzeichnis zu kopieren. Danach ist das Programm betriebsbereit.

1. Kompilieren

Um den Quellcode des Programms in eine ausführbare Datei zu kompilieren,

//TODO Maven Anleitung

1. Programmausführung

Um einzelne Testdateien zu verarbeiten, muss das Programm über die Kommandozeile (cmd) ausgeführt werden. Der Aufruf aus dem Wurzelverzeichnis des Projektes erfolgt über den Befehl:

java -jar ./run/Programm.jar <pfad/zur/inputdatei.txt>

1. Testausführung

Um alle in testfaelle/input vorliegenden Testdateien nacheinander zu verarbeiten, kann das Skript run/RunProgram.bat genutzt werden. Mittels Doppelklick auf die Datei im Windows Explorer startet die Ausführung.

// TODO „hier klicken?“

Zum Testen von bestimmten weiteren, bereits im Verzeichnis testfaelle/input hinterlegten Beispielen muss das Programm wie in Absatz B.V beschrieben ausgeführt werden.

// TODO falls sehr lange dauert Hinweis

1. Entwicklerdokumentation
2. Verfügbare Klassen und Schnittstellen

Die Dokumentation der verfügbaren Klassen und Schnittstellen wurde mit Hilfe des Tools javadoc aus dem Quellcode erzeugt und im mitgelieferten Archiv im Verzeichnis „doc/javadoc“ abgelegt. Zur Einsicht muss die Datei „doc/javadoc/index.html“ in einem Webbrowser geöffnet werden.

1. Nutzungshinweise

// TODO exemplarisch, wie Programm angepasst & erweitert

1. Fehlerbehandlung

Die Fehlerbehandlung beim Initialisieren des Programms ist Aufgabe der Implementierung von IReader im Falle von technischen Fehlern und von der Klasse InputConverter im Falle von syntaktischen oder semantischen Fehlern. Die Klassen werfen im Fehlerfall eine spezielle Exception: TechnicalException, InvalidSemantikException oder InvalidSemantikException, welche von IOException erben. Die Fehlermeldung, welche als String im ersten Parameter des Konstruktors in die Exception gegeben wird und über die Methode getMessage() ausgelesen werden kann, wird dann in die Ausgabedatei geschrieben. Diese Konvention muss von zusätzlichen Implementierungen des Interfaces IReader und des Interfaces IInputConverters beibehalten werden.

1. Entwicklungsumgebung

Programmiersprache : Java

Compiler : OpenJDK javac 16

Rechner : Dell Latitude 5500

Betriebssystem : Windows 11 Pro

// TODO maven?

1. Verwendete Hilfsmittel

* IntelliJ 2021.1

Entwicklungsumgebung für Java

<https://www.jetbrains.com/de-de/idea/>

* Visual Paradigm Professional

UML-Tool

<https://www.visual-paradigm.com/>

* Microsoft Word

Texteditor

<https://www.microsoft.com/de-de/download/office.aspx>

1. Erklärung

Erklärung des Prüfungsteilnehmers:

Ich erkläre verbindlich, dass das vorliegende Prüfprodukt von mir selbstständig erstellt wurde. Die als Arbeitshilfe genutzten Unterlagen sind in der Arbeit vollständig aufgeführt. Weder ganz noch in Teilen wurde die Arbeit bereits als Prüfungsleistung vorgelegt. Mir ist bewusst, dass jedes Zuwiderhandeln als Täuschungsversuch zu gelten hat, der die Anerkennung des Prüfprodukts als Prüfungsleistung ausschließt.

// TODO Unterschrift

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ort, Datum Unterschrift des Prüfungsteilnehmers