Ausarbeitung Programiersprachen II: Autonomous Instantdocument System

Jonas Schwind und Martin Boschmann

Technische Hochschule Ostwestfahlen-Lippe

PGR2 8512: Programmiersprachen II

Prof. Dr. rer. nat. Stefan Wolf

08.09.2023

Ausarbeitung Programiersprachen II: Autonomous Instantdocument System

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Analyse

Anforderungen

Erforderlich

- Das Programm soll aus vorgefertigten (anpassbaren) Schnipseln ein LaTeX-dokument erstellen.
- Die Schnipssel werden in einer MySQL Datenbank verwaltet, diese soll sich zu einer CSV datei exportieren lassen.
- Andere Einstellungen sollen in einer JSON verwaltet werden.
- Die Konfigurationsdateien (Schnipsel eigeschlossen) sollen an dem Ort gespeichert werden, der vom Betriebssystem für diesen Zweck vogesehen ist.
- Es soll eine Grafische Oberfläche für den gemeinen Nutzer, und eine CLI-Schnittstelle geben. Letztere emöglicht z.b. eine Automatisierung via cron.
- Es soll eine Detailierte Anwenderdokumentation geben, welche auf github.com und in Form einer manpage (oder funktional vergleichbares) einsehbar ist.

Optional

- Das Programm sollte universell einsetzbar sein. Das beduetet dass das Programm nicht zwingend für Klausuren eingesetzt werden muss, und auch für andere Dokumente verwendet werden kann.
- ~~Generierung neuer Schnippsel durch ChatGPT~~ (verworfen)

Architektur

Die Software ist modular aufgebaut. Das Projekt besteht aus 5 Modulen, davon sind 4 funktional und dienen der Umsetzung von Projektteilaspekten. Das letzte Modul (welches hier nur als Modul bezeichnet wird, weil es von Maven als solches behandelt wird), dient als Sammelstelle für Dateien, die für eine Installation notwendig sind.

Der Zweck des modularen Aufbaus ist darin begründet, dass so jedem Teammitglied Module nach Bedarf zugewiesen werden können, und so die Verantwortlichkeiten geklärt werden. Das in Abbildung 1 gezeigte UML-Diagramm, zeigt die Softwarearchitektur des Projekts, die getter uns setter wurden hier und im folgenden jedoch nicht aufgeführt.

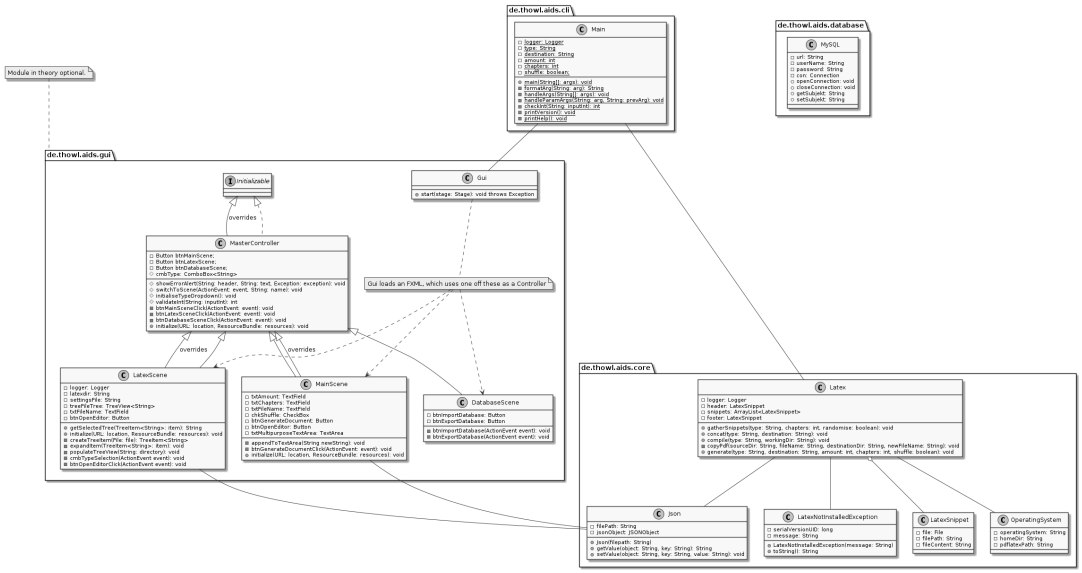

Abbildung 1

Gesammtarchitektur des Programms

Werkzeuge

Im Folgenden wird eine Liste von Werkzeugen aufgeführt, die während der Entwicklung eingesetzt wurden.

Entwicklungsumgebungen

1. DOOM Emacs

   DOOM Emacs ist eine Distribution von Emacs. Es bietet einige vorkonfigurierte Pakete und Shortcuts, die das Programmieren erleichtern.

2. VS Code

   VS Code ist ein Open-Source Code-Editor von Microsoft, welcher mit einem Plugin auch die Programmierung in Java ermöglicht.

git

git ist ein gängiges Versionskontrollsystem. Es ermöglicht die Zusammenarbeit von mehreren Entwicklern am selben Projekt.

JSON

Javascript Objekt Notation (JSON) ist ein Datenformat welches wir dazu verwenden, Einstellungen des Programms persistent zu halten. Die Einstellungen werden in einer Datei gespeichert und bei Bedarf geladen.

Maven

Maven ist ein Build-Management-Tool für Java. Es automatisiert das Verwalten von externen Libraries, das Compilieren des Codes, und das Testen mit Unittests.

plantuml

plantuml ist ein textbasiertes Werkzeug zur Erstellung von UML-Diagrammen. Die Diagramme werden mit einer Syntax definiert, die sich leicht in Quellcode übertragen lässt.

Benutzerschnittstellen

CLI

Es wurden folgende Argumente für die CLI-Schnittstelle festgelegt.

Tabelle 1

CLI-Argumente

Argument	Funktion
t, –type <type>	Legt den Dokumenttyp fest.
-c, –chapters <chapters>	Legt die Kaptielanzahl pro Dokument fest.
-a, –amount <amount>	Legt die Anzahl der DokumentInstanzen fest.
-s, –shuffle	Schaltet den Shuffle modus an.
-h, –help	Zeigt eine Hilfe an.
-v, –version	Zeigt die versionsnummer an.

Die in Tabelle 1 aufgeführten Argumente können auch in der Anwenderdokumentation nachgelesen werden.

GUI

Der Aufbau der Benutzeroberfläche wurde von zwei Anwendungen inspiriert. Das Farbschema, das in unserer Anwendung verwendet wurde, wurde von Opera GX (Abbildung 2) übernommen. Der Launcher des GAMMA Modpacks für das Spiel STALKER Anomaly (Abbildung 3) inspirierte das Layout.

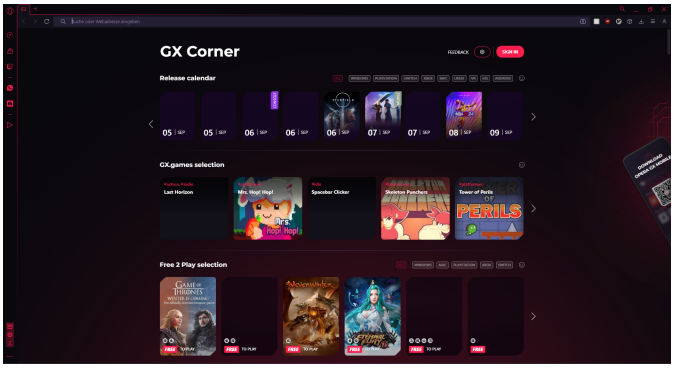

Abbildung 2

Userinterface von Opera GX

Abbildung 3

Userinterface des GAMMA Launchers

Entwurf

de.thowl.aids.cli

Das Modul `cli` ist der Startpunkt des Programms, es ist der CLI-Client oder startet die grafische Oberfläche. (s. Abbildung 4)

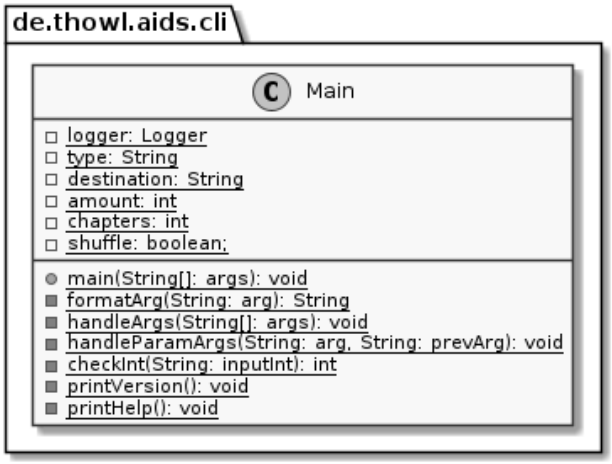

Abbildung 4

Architektur des cli Moduls

de.thowl.aids.cli.Main

Die `main` Methode verwaltet die CLI-Argumente, sofern welche übergeben wurden. Wenn keine Argumente übergeben wurden, startet das Programm mit einer grafischen Oberfläche.

Sollten Argumente übergeben worden sein, werden diese zunächst formatiert (s. Abbildung6). Im Anschluss werden die Argumente mit der `handleArgs` Methode abgearbeitet, sofern diese der Methode bekannt sind. (s. Abbildung 7) Wenn `handleArgs` mit diesem Argument nichts anfangen kann, wird davon ausgegangen, dass es sich bei diesem Argument um ein *parametrisiertes Argument* handelt, bspw. `-amount 10`. Diese werden von der `handleParamArgs` Methode verarbeitet (s. Abbildung8).

de.thowl.aids.core

Im Modul `Core` sind alle essenziellen Klassen und Methoden vorhanden. Es ist das Herzstück des Programms.

de.thowl.aids.core.Json

Die `json` Klasse kümmert sich, wie der Name vermuten lässt, um die Datenhaltung/Verwaltung via JSON.

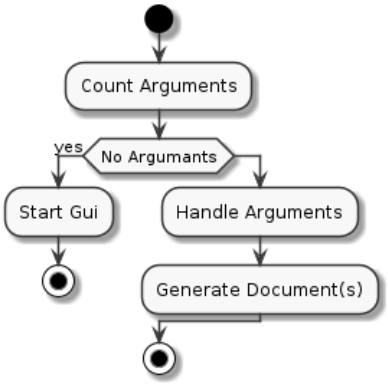

Abbildung 5

Ablaufdiagramm der main Methode

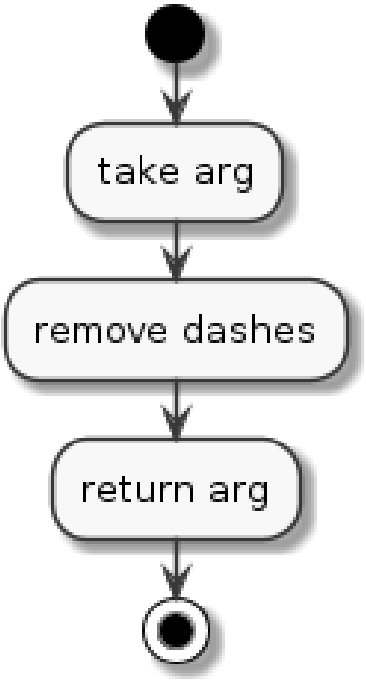

Abbildung 6

Ablaufdiagramm der formatArg Methode

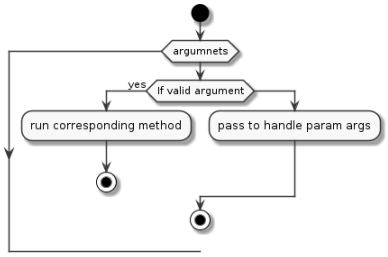

Abbildung 7

Ablaufdiagramm der handleArgs Methode

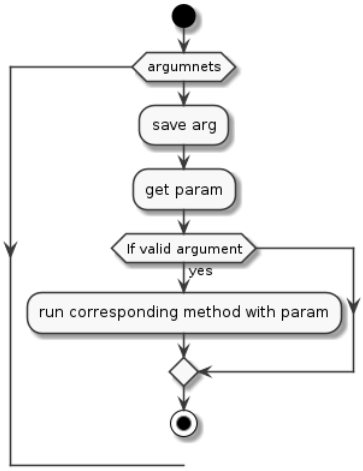

Abbildung 8

Ablaufdiagramm der handleParamArgs Methode

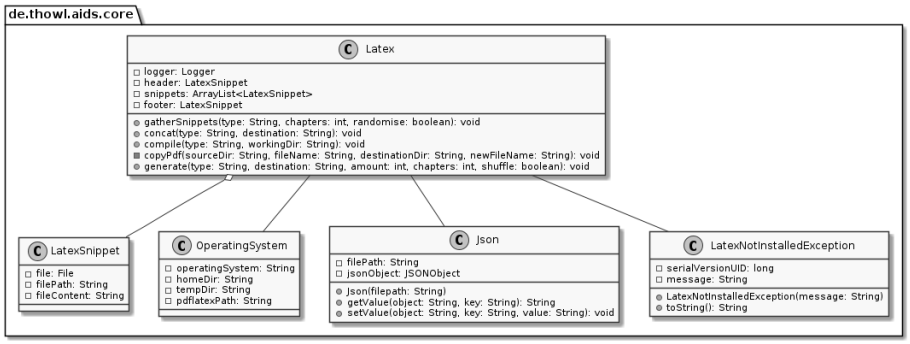

Abbildung 9

Architektur des core Moduls

Die Ablaufdiagramme in Abbildung 10 und Abbildung 11 sind trivial und sprechen für sich selbst.

de.thowl.aids.core.Latex

Die Klasse `Latex` dient zur Erstellung und Kompilierung von LaTeX Dokumenten.

Die Methode `gatherSnippets` sammelt alle Schnipsel, die für ein LaTeX-dokument benötigt werden. Der Dokumentheader und Footer werden in je einer separaten Variable gespeichert (s. Abbildung 12).

Alle anderen Schnipsel landen in einer ArrayList.

Die Methode `concat` verbindet alle Schnippel, die von `gatherSnippets` gesammelt wurden, zu einer `.tex`-datei. Der Inhalt des Headers und des Footers werden in die Datei kopiert.

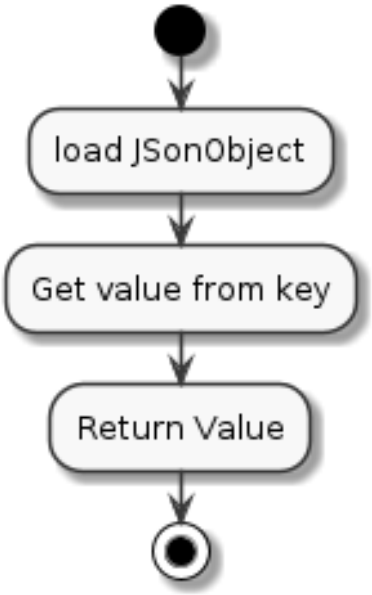

Abbildung 10

Ablaufdiagramm der getValue Methode

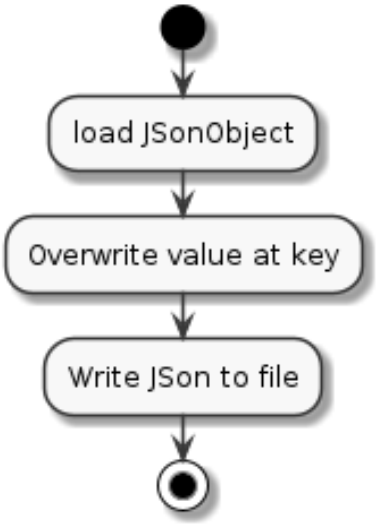

Abbildung 11

Ablaufdiagramm der setValue Methode

Bei allen anderen Schnipseln wird der Dateipfad zu einem `\inculde{}` formatiert und eingefügt (s. Abbildung 13).

Die `compile` Methode compiliert die von `concat` erstellte LaTeX-datei zu einem PDF-Dokument (s. Abbildung 13).

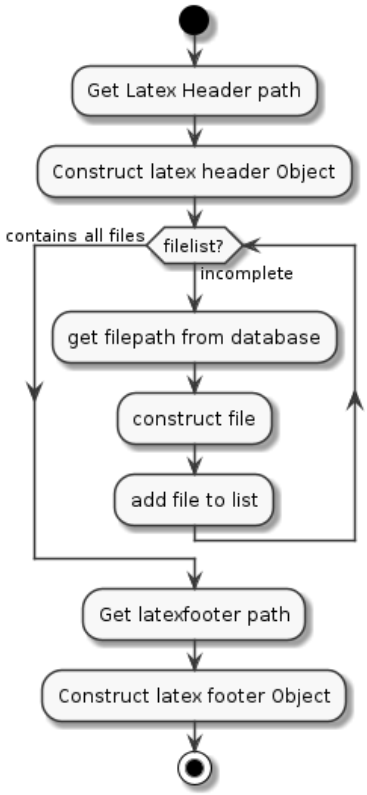

Abbildung 12

Ablaufdiagramm der gatherSnippets Methode

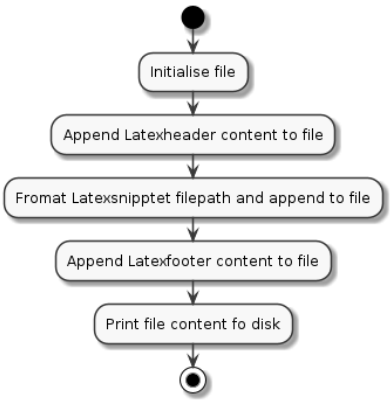

Abbildung 13

Ablaufdiagramm der concat Methode

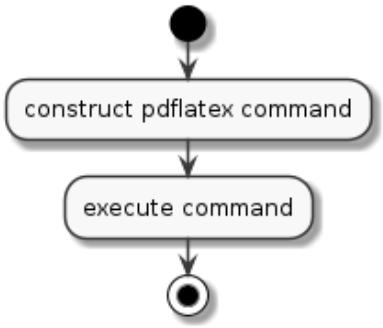

Abbildung 14

Ablaufdiagramm der compile Methode

de.thowl.aids.core.OperartingSystem

Der in Abbildung 15 visualisierte Konstruktor, legt die Objektattribute abhängig vom verwendetem Betriebssystem an.

Es soll vermieden werden, für das Programm wichtige Dateien sinnlos im Dokumente-Verzeichnis abzulegen. Daher werden die Dateipfade, die vom OS für diesen Zweck vorgesehen sind, verwendet.

Aus persönlichen Gründen kann die Verwendung unter macOS allerdings nicht getestet werden.

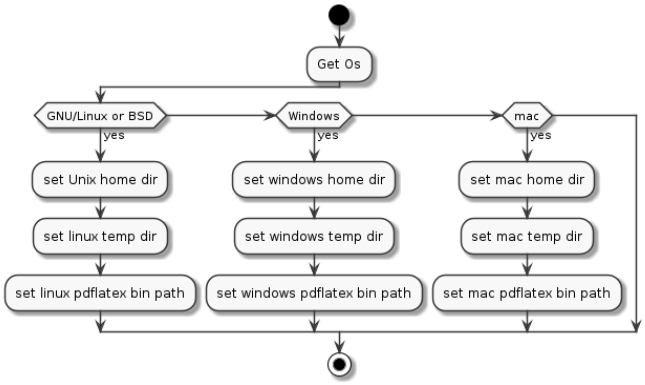

Abbildung 15

Ablaufdiagramm des Konstrucktors der Klasse OperartingSystem

de.thowl.aids.gui

Das `gui` Modul ist für die grafische Oberfläche zuständig. In diesem Abschnitt werebn Eventhandler nicht Betrachtet.

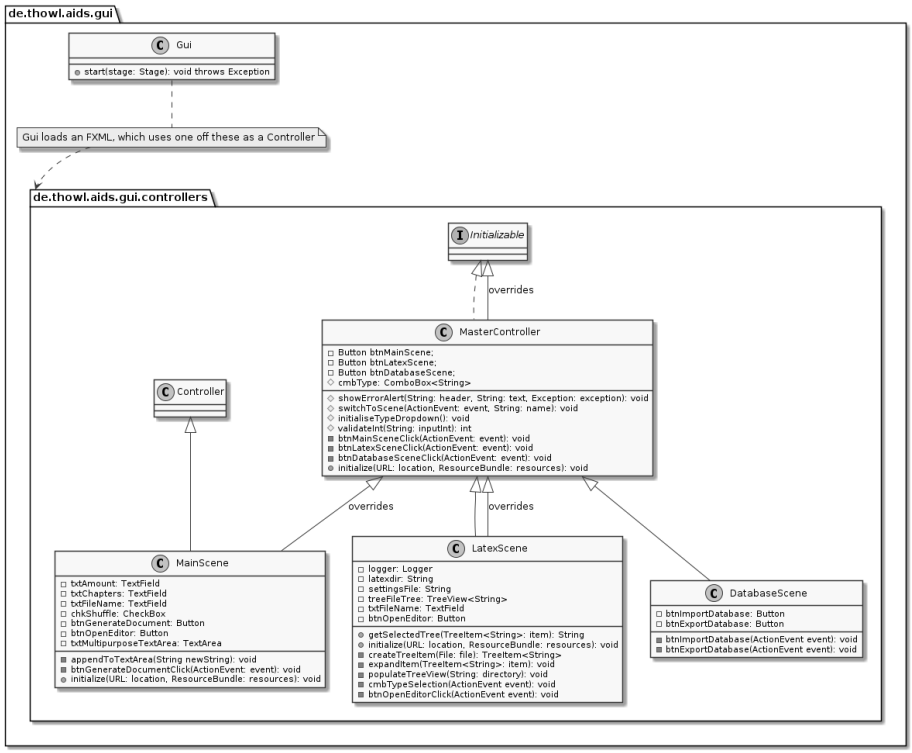

Abbildung 16

Architektur des gui Moduls

de.thowl.aids.database

Das `database` Modul ist für die Datenbank zuständig.

de.thowl.aids.gui.Controller

Die Klasse `Controller` ist eine abstrakte Klasse, von der alle GUI-Controller erben.

Die in Abbildung 17 dargestellte Methode füllt das TypeDropdown, welches dazu verwendet wird, den Dokumenttypen grafisch auszuwählen, mit Daten. Der einfachste Weg dies zum Zeitpunkt der Implementierung zu realisieren ist es, die Dateistruktur im config-verzeichnis nachzubilden (s. Abbildung 17).

Die `switchToScene` methode (Abbildung 18) wechselt die Szene in der GUI. Dazu werden zuerst die bestehenden Dimensionen (Fensterhöhe und Breite) zwischengespeichert. Im Anschluss wird die FMXL und das Stylesheet ersetzt. Daraufhin werden die gespeicherten Dimensionen wieder auf das Fenster angewendet, um Größenveränderungen zu verhindern.

de.thowl.aids.gui.MainScene

Die Methode `appendToTextArea` (Abbildung 19) fügt eine Textzeile in eine TextArea ein, welche als Statusindikator genutzt wird.

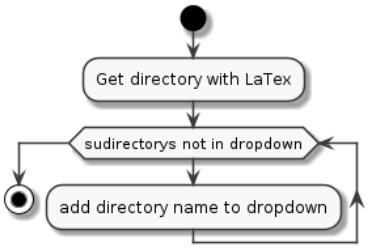

Abbildung 17

Ablaufdiagramm der initializeTypeDropdown Methode

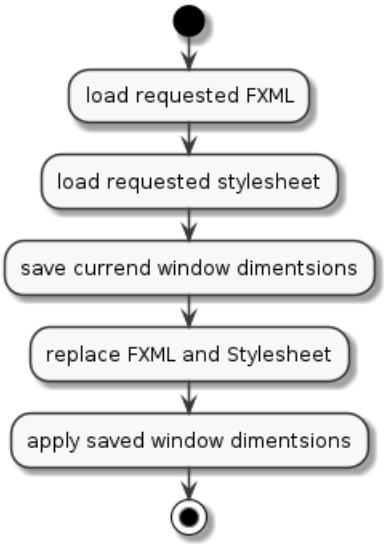

Abbildung 18

Ablaufdiagramm der switchToScene Methode

Dazu wird der bestehende Text kopiert, der neue String angefügt, und anschließend der alte durch den neuen Text ersetzt.

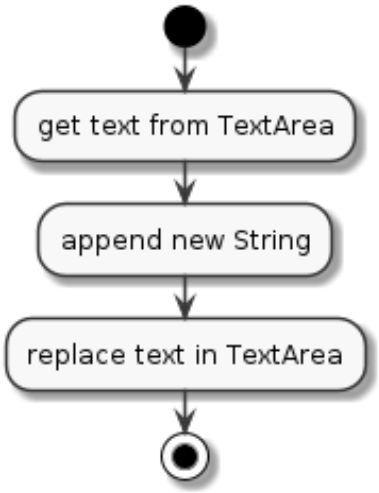

Abbildung 19

Ablaufdiagramm der sappendToTextArea Methode

Gesprächsprokolle

Im Folgenden werden nur die Gespräche mit den Dozenten aufgeführt, Alles Weitere wurde bei Bedarf auf WhatsApp geklärt.

1ste Besprechung

Es wurden unsere Ideen und Ansätze bezüglich des Projektes vorgestellt, und der Meilensteinplan präsentiert. Diese wurden alle gut aufgenommen. Im Abschluss wurde noch ein Themenwunsch für eine evtl. Vorlesung vorgeschlagen: JavaFx.

2te Besprechung

Es wurde der aktuelle Fortschritt bezüglich des Projektes, sowie Probleme vorgestellt. Probleme mit den Unit-Tests konnten nach einem Vorschlag ignoriert werden (diese werden daher auch nicht weiter erwähnt).

Implementierung (Probleme und Lösungen)

pdflatex

Beim Testen der `compile()` Methode passierte absolut gar nichts15Es wurden keine Fehler oder vergleichbares ausgegeben, und von einem PDF-Dokument fehlte jede Spur. Es gab also keinen Hinweis darauf, dass `pdflatex` ausgeführt wurde.

Um dem Ursprung auf den Grund zu gehen, sollte erstmal dafür gesorgt werden, dass der gewohnte Output von pdflatex von unserem Programm ausgegeben wird, um Fehler erkennen zu können. Bei der Umsetzung wurde festgestellt, dass der fehlende Output an sich die Ursache war, und eben dieser nicht blockiert werden darf. Dieses Problem hat sich mehr oder weniger von selbst gelöst.

Um `pdflatex` auszuführen, setzten wir den Befehl als String zusammensetzen und übergaben ihn an einen Processrunnner. Diese Art und Weise wurde plötzlich als deprecated markiert.

Nach einem Blick in die Dokumentation wurde klar, dass ein `String[]` vorgesehen ist. Unser Ansatz sollte durch häufigen Missbrauch (nicht näher beschrieben) aus Java entfernt werden. Der entsprechende Quellcode wurde stante pede angepasst.

CSS-stylesheet

Einige Elemente wie die Icons in der Seitenleiste und die Text-Area, wollten sich nicht den in der Größe anzeigen lassen, die ursprünglich vorgesehen war.

Etwas Nachforschung ergab, dass die Icons einige Extraregeln für die Skalierung benötigten. Da die Text-Area nach ewigen Trial-and-Error immer noch nicht die geplanten Dimensionen aufwies, und die optische Schönheit aus Zeitgründen nicht priorisiert wurde, wurde dies als “Designentscheidung” angesehen und so belassen.

ChatGPT

ChatGPT ist bei der Beantwortung von Anfragen immer übermäßig gesprächig, Eine typische ChatGPT-Antwort sieht in etwa so aus:

> [Beschreibung der eigenen Frage]
>
> [Tatsächliche Antwort]
>
> [Kurzes Fazit, oder ähnliches]

Da ChatGPT durch einen Bug bei zu vielen ähnlichen Fragen in einen Zustand wechselt,

indem die eigenen Rahmenbedingungen ignoriert werden, gibt es selbst mit Überredungsversuchen keine zuverlässige Möglichkeit, den für uns wichtigen Teil der Antwort zu isolieren. Diese Feature umzusetzen würde sich als sehr schwierig erweisen, und vermutlich in kryptischen, unwartbaren regex Masken ausarten.

Das Feature wurde verworfen, es war ohnehin nur als optionale Anforderung geplant.