

- Oberstufe -



Besondere Lernleistung Informatik

„CaRP-Assigner“

Course and Research Paper Assigner

Verfasser:	Fabius Mettner
Lehrer:	Dipl.-Inform. Christian Wolf
Stufe:	Q2
Abgabe:	16.03.2020

Inhaltsverzeichnis

1. Einleitung.....	2
1.1. Motivation	2
1.2. Vorgehensweise	5
2. Der Entwicklungsprozess.....	7
2.1. Die Neustrukturierung	7
2.2. Die Ausprogrammierung	7
2.3. Die größten Fehler	9
2.3.1. Das Verschwinden der Projektkurs-Schüler.....	9
2.3.2. Fehler beim Sortieren der Berechnungen nach der Güte.....	10
3. Manual.....	12
3.1. Importieren.....	12
3.1.1. Kriterien, die eine Eingabedatei erfüllen muss	13
3.2. Der Verteilungsprozess.....	16
3.3. Das Exportieren der Daten.....	16
3.4. Das Bearbeiten der Eingabedaten im CaRP-Assigner	17
3.5. Das Bearbeiten der Verteilung im CaRP-Assigner	18
3.6. Die Einstellungen	19
4. Fazit.....	20
4.1. Optimierungsmöglichkeiten	20
4.2. Zusammenfassung	20
5. Anhang	21
5.1. Konfiguration.....	21
5.2. Dateimanagment (In- und Export)	22
5.3. Verteilungsprozess	26
5.4. Graphische Oberfläche	33
5.5. Utils.....	45
6. Quellen- und Literaturverzeichnis.....	53
6.1. Internet	53
6.2. Bilder	53
6.3. Java-Abhängigkeiten	54
7. Abschlusserklärung	55

1. Einleitung

Der „Course and Research Paper Assinger“ (Kurs und Facharbeit Zuweiser = KuFa-Zuweiser) ist ein Programm zum Zuweisen von Schülern nach Wunschkursen, z.B. Sportkurse oder Facharbeiten, die nach einer Priorität geordnet sind. Dabei wird versucht eine Zuweisung zu erstellen, welche einer Verteilung mit der bestmöglichen Priorität entspricht und dabei das Limit eines jeden Kurses nicht überschreitet.

1.1. Motivation

Betrachtet man die alte Version des KuFA-Zuweisers (Abb. 1), so erkennt man eine im Java „Swing“-Format angelegte Benutzeroberfläche. Beim Ausführen des Berechnungsprozesses, friert diese graphische Oberfläche (GUI) ein,

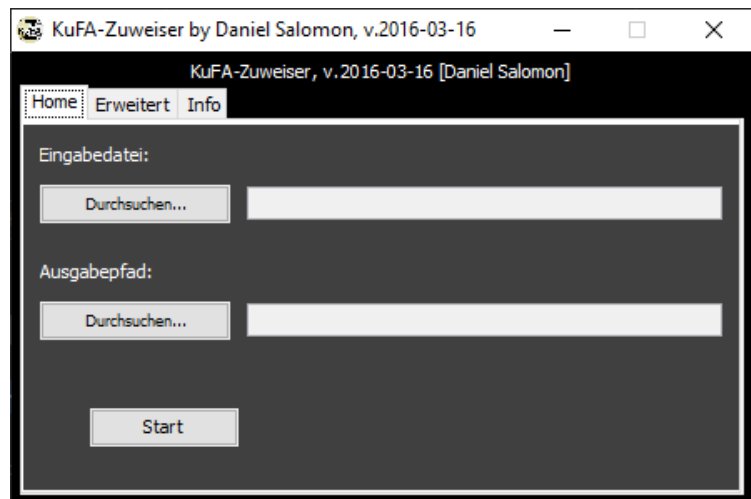


Abb. 1: Alte KuFA-Anwendung

wodurch ein Beobachten des Fortschrittes nicht mehr möglich ist. Auch lässt sich die Größe des Fensters nicht verändern. Beides hat die Grundfunktionen der Applikation nicht beeinträchtigt, konnte aber beim Anwenden der Applikation als störend empfunden werden. Aus diesem Grund lässt sich zunächst eine vollständige Überarbeitung der graphischen Oberfläche als Ziel setzen. Diese muss vor allem anwenderfreundlicher, variabler und moderner werden.

Bei der ersten Betrachtung des dahinterstehenden Quellcodes, ist bereits der nächste Verbesserungspunkt zu erkennen. Der alte KuFA-Zuweiser verfügt lediglich über wenige besonders lange Klassen, die die einzelnen Funktionen grob unterteilen. Um eine übersichtliche Struktur in das Projekt zu bringen, wurde nun mit sogenannten „packets“ (Paketen) gearbeitet, die das Projekt

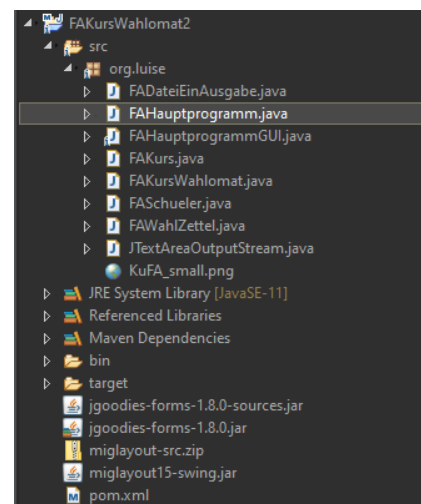


Abb. 2: Klassenstruktur des alten KuFA-Zuweisers

hierarchisch strukturieren sollten. Über diesen Weg werden die einzelnen Funktionen leichter unterscheidbar gemacht. Gleichzeitig bietet die in Abb. 3 zu erkennende Struktur einen weiteren Vorteil: Jeder Bereich kann einfacher getrennt werden und so in einem anderen Programm einfach wiederverwertet werden. Es erleichtert auch das Testen der einzelnen Programmabschnitte, durch den Fokus auf die unterschiedlichen Funktionsbereiche.

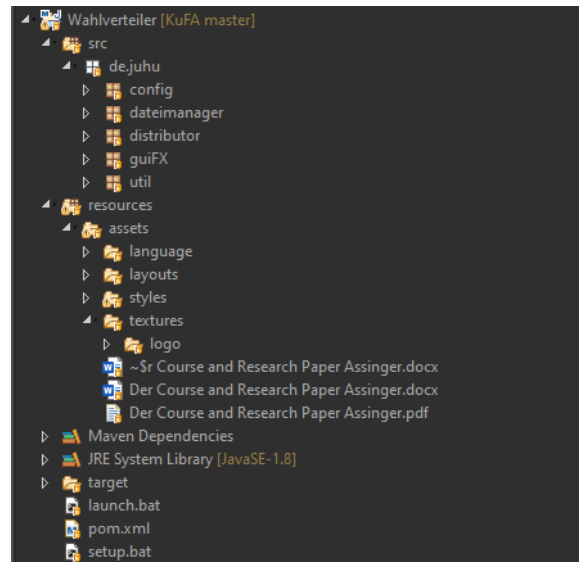


Abb. 3: Klassenstruktur des CaRP-Assigners

In Folge dieser Neustrukturierung ist der gesamte vorangegangene Quellcode aus mehreren Gründen verworfen worden:

1. Zum einen war es nicht möglich, mit ihm die einzelnen Funktionen entsprechend zu trennen.
2. Teilweise erschien der Quellcode sehr unübersichtlich und das Einarbeiten in diesen hätte ähnlich viel Zeit in Anspruch genommen, wie das strukturierte Neuerstellen des Quellcodes.
3. Durch diese komplette Neuauflage des Projektes wird es zudem möglich, einfacher neue Funktionen zu integrieren.

4

Im Verlauf des Erstellens der besonderen Lernleistung kommt zudem ein weiterer Punkt hinzu:

Eine Voransicht der Daten im Programm sowohl vor dem Zuweisungsprozess als auch nach diesem, erleichtert dem Endnutzer das Bedienen des Programms. So wird eine Vorschau für die Eingabe- und für die Ausgabedaten eingefügt. Um die flexible Nutzung des Programms hierbei zu steigern, wird im Anschluss noch die Möglichkeit der Bearbeitung der Daten in den Voransichten hinzugefügt.

1.2. Vorgehensweise

Mit diesen Verbesserungspunkten beginne ich die Erstellung eines komplett neuen Kurs- und Facharbeit Zuweisers. Zunächst einmal müssen die bereits vorhandenen Funktionen des alten Programms gesichtet und neu ausprogrammiert werden, um im Folgenden dann nach neuen Ansätzen zur Überarbeitung des Programmes und neuen Funktionen suchen zu können. Nach einiger Überlegungszeit und vielem Ausprobieren entsteht ein komplett neues Java-Projekt in Eclipse, einer Programmierumgebung für Java.

Hierbei werden schon die ersten großen Änderungen im Grundaufbau vorgenommen. Anstatt der flachen Hierarchie des Klassenbaumes wird nun auf einen hierarchisch durchstrukturierten Klassenbaumaufbau, wie er auch in Abb. 3 zu sehen ist, gesetzt. Dies ermöglicht nicht nur einen einfacheren Überblick über die einzelnen Bereiche, sondern erweist sich auch als sinnvoll, wenn man nur Teile des Programms verändern oder in eine neue Umgebung (z.B. in ein neues Projekt) überführen will.

Des Weiteren wird nun auch die Einbindung von Libraries erneuert. Statt die einzelnen benötigten Libraries manuell hinzuzufügen wird auf das Build Management Tool „Maven“ gesetzt. Neben einer vereinfachten Möglichkeit zum Exportieren des Projektes in das „.exe“ und „.jar“-Format ist es so auch möglich, einfacher neue Referenzen, sogenannte „Dependencies“, im Programm einzufügen. Diese Referenzen werden von Maven selbstständig verwaltet und es ist nicht mehr nötig, manuell Libraries in das Projekt zu importieren.

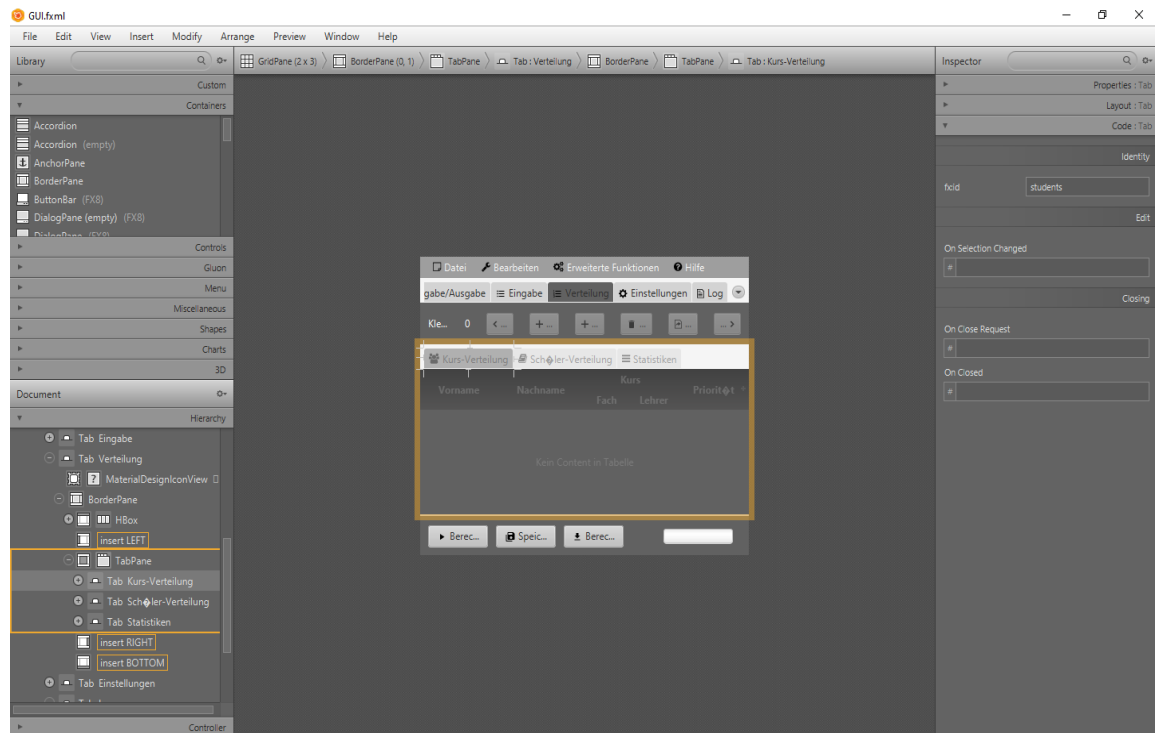


Abb. 5: Ansicht des GUIs im „Scene Builder“

Ebenso hat sich die graphische Oberfläche grundlegend verändert. Die genutzte „Swing“ Oberfläche wird verworfen und durch eine neue, auf „JavaFX“ basierende Benutzeroberfläche, ersetzt. Diese gibt der Anwendung nicht nur einen „modernerer Look“, sondern ermöglichte auch das erleichterte Darstellen neuer Funktionen in das graphische Gesamtbild. Um eine passende Oberfläche zu schaffen, arbeitet man mit dem für diesen Zweck entwickelten „Scene Builder“, welcher die Erstellung neuer Benutzeroberflächen durch eine bearbeitbare Voransicht weitgehend erleichtert.

Im weiteren Entwicklungsprozess werden neben neuen Features auch weitere Referenzen und Designs eingefügt. So ist das Wechseln zwischen einem hellen und einem dunklen Oberflächendesign, sowie das Wechseln der Sprache zwischen Deutsch und Englisch möglich geworden. Dies kann durch das Einbinden von „css“, zum Verändern des Oberflächendesigns, sowie das Einbinden von Sprachdateien über sogenannte „Internationalized Strings“ realisiert werden.

2. Der Entwicklungsprozess

Nachdem die grundlegenden Ideen strukturiert und der Aufbau des Projekts geklärt ist, kann nun der eigentliche Entwicklungsprozess gestartet werden. Hierbei wird viel Zeit zum Testen und Verbessern benötigt und in die Implementierung der einzelnen Ideen mithilfe der

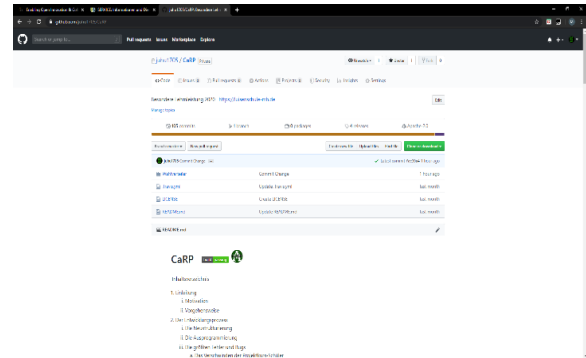


Abb. 6: „Github Repository“ des CaRP-Assigners

vorangegangenen Vorgehensweisen investiert. Um einen Datenverlust oder einen andersgearteten Rückschlag zu vermeiden, wird nun ein „GitHub-Repository“ (Abb. 6) angelegt, auf dem die letzte funktionierende Version stets hinterlegt ist.

2.1. Die Neustrukturierung

Zuerst ist durch die neu aufgesetzte Struktur auch eine vollständige Erneuerung des alten „KuFA-Zuweisers“ entstanden. Dabei werden folgende Baumpfade erstellt: Unter dem Pfad „distributor“ ist alles für den Zuweisungsprozess Relevante eingearbeitet, während das Einlesen und Schreiben von Dateien in den Reiter „dateimanager“ eingefügt wird. Der Pfad „config“ enthält alle relevanten Dateien zum Erstellen der Konfiguration und unter „guiFX“ werden die Klassen der graphischen Oberfläche gesammelt. Alle weiteren Klassen fasst man schließlich im Reiter „util“ zusammen.

Um eine weitreichende Struktur zu wahren, wird zudem ein eigener Ordner für alle nicht Java-Dateien erstellt. Hier finden sich neben der „Hilfe“ Datei auch die „Stylesheets“ (css Dateien zum Bearbeiten der Ansicht der graphischen Oberfläche) und die Sprachdateien, sowie die „fxml“ Dateien, die die Struktur der einzelnen GUIs beinhalten.

2.2. Die Ausprogrammierung

Begonnen hat die Ausprogrammierung zunächst mit dem Im- und Exporter für Dateien. Im gleichen Zug entsteht auch die Klasse „References“ mit einem den Umständen angepassten Logger. Um die Exportdatei graphisch aufzuwerten,

wird die Klasse „CellStyles“ erstellt, die eine kleine Auswahl an unterschiedlichen Ansichten für die Tabellenzellen von Excel bereitstellt (siehe Abb.7).

Der Berechnungsprozess und alle zugehörigen Klassen werden nachfolgend in einem Schritt eingefügt. Um diese Funktionen zu testen, wird zunächst auf ein einfaches GUI auf „Swing“-Basis zurückgegriffen, bevor schließlich im nächsten Schritt unter Zuhilfenahme des „SceneBuilders“ ein JavaFX GUI aufgesetzt wird.

	A	B	C	D	E
1	Name	Vorname	Kurs	Lehrer	Priorität
2					
3	Mustermann	Max	C	D	2
4	Mustermann	Max	C	D	2
5	Mustermann	Max	@PJK G		1
6	Mustermann	Max	C	D	2
7	Mustermann	Max	@PJK G		1
8	Mustermann	Max	C	D	2
9	Mustermann	Max	C	D	2
10	Mustermann	Max	C	D	2
11	Mustermann	Max	C	D	2
12	Mustermann	Max	@PJK G		1
13	Mustermann	Max	C	D	2
14	Mustermann	Max	C	D	2
15	Mustermann	Max	C	D	2

Abb. 7: Exporttabelle der Kursverteilung mit Beispieldaten

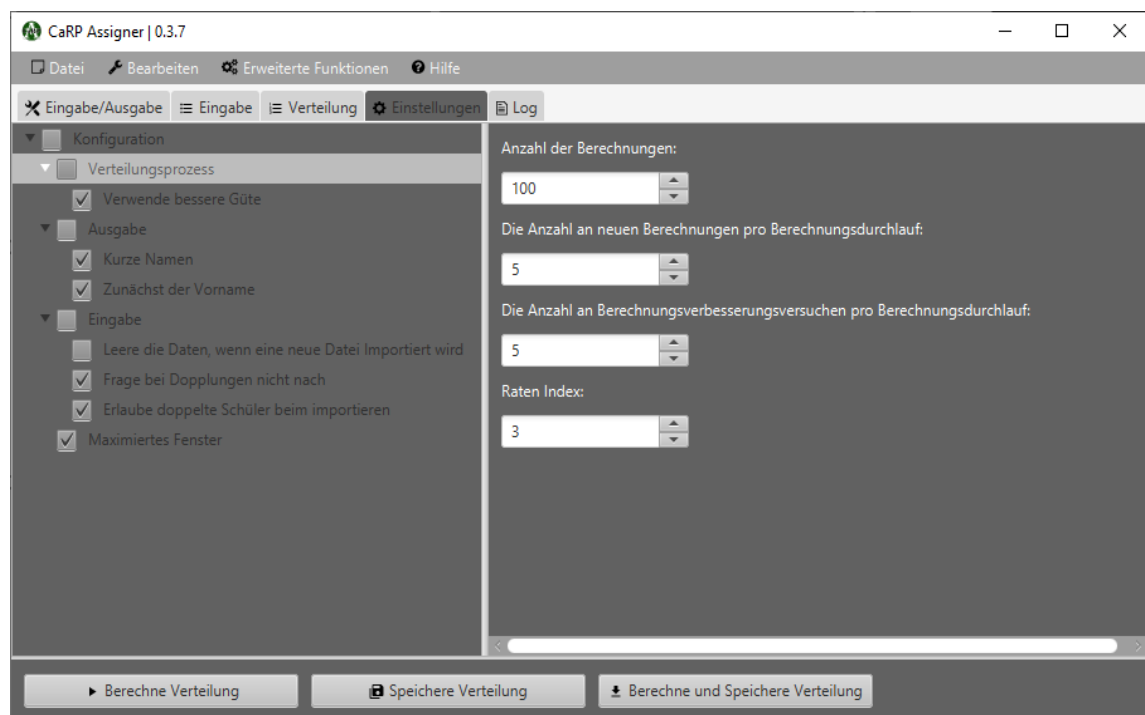


Abb. 8: Überarbeitete Einstellungsansicht

Nach Erstellen des neuen GUIs kommen nun vor allem in dem Bereich der Berechnung und in der graphischen Darstellung immer wieder kleinere und größere Neuerungen hinzu. So wird die im Benutzerverzeichnis hinterlegte, speicherbare Konfiguration, der speicherbare Log und die Möglichkeit, eine einlesbare „carp“-Datei zu exportieren, entwickelt. Über einen längeren experimentellen Zeitraum mit vielen Änderungen im graphischen Design,

entstehen die neuen Vorschauen für die Voransicht der Verteilung und der einzulesenden Daten.

Schließlich werden die Shortcuts, sowie die Icons in das Programm eingeführt und das „Über“ Fenster optimiert. Zuletzt wird die in Abb. 8 zu erkennende Darstellung der Einstellungen überarbeitet und damit endet der eigentliche Programmierprozess. Zum Schluss müssen noch bestehende Bugs (Fehler) behoben und Klassen ausdokumentiert werden.

2.3. Die größten Fehler

Während der Programmierung haben sich in den Programmcode immer wieder kleinere und größere Fehler eingeschlichen. Einige von ihnen sind sofort aufgefallen, bei anderen ist ein genaues Verfolgen der vorgehenden Prozesse nötig gewesen, um sie im Code selbst ausfindig zu machen und dort zu beheben. Dafür ist zunächst ein kleinschrittiges Auffinden der Fehlerquellen in kleinsten Programmabschnitten erforderlich. Mithilfe von hunderten „Print-Messages“ und unter Nutzung des in Eclipse verfügbaren Debug-Modus, können so viele der Fehlerquellen identifiziert werden. Nach dem Auffinden der Fehlerquelle ist das eigentliche Fehlerbeheben nicht mehr problematisch, da die meisten Fehler lediglich kleinen logischen Aussetzern (des Programmierers 😊) oder dem Übersehen von Zusammenhängen zu verdanken sind. Bei komplexeren Fehlern ist es sinnvoll, testweise die Wirkung einer Veränderung einzelner Programmabschnitte auszuprobieren, um so schließlich doch die Fehlerquelle zu entdecken. Im Folgenden sind einige dieser Fehler beispielhaft aufgeführt.

2.3.1. Das Verschwinden der Projektkurs-Schüler

Die Behebung des ersten Fehlers, die ungewollte Verminderung der Anzahl der Projektkursschüler während des Berechnungsprozesses, hat besonders viel Zeit in Anspruch genommen. Jedoch ist die Behebung des Problems nach Entdecken des Programmfehlers relativ simpel gewesen. Durch Betrachten unterschiedlicher Verteilungsvorgänge, kann der Fehler schließlich in folgender Schleifenkonstruktion lokalisiert werden:

```

For (int i = 0; i < this.allStudents.size(); i++) {
    If ((this.allStudents.get(i).getActiveCourse() != null &&
        this.allStudents.get(i)
            .getActiveCourse().equals(this.ignore())) ||
        this.allStudents.get(i).getCoursesAsList()
            .contains(this.ignore()))
        this.ignoredStudents.add(this.allStudents.remove(i));
}

```

Wie zu sehen ist, wird jedes Mal beim Durchlaufen der Schleife der Iterator *i* eine Position weitergezählt. Wird nun ein Schüler aus der Liste „allStudents“ entfernt, rücken die nachfolgenden Schüler eine Position auf. Stehen nun zwei Projektkursschüler hintereinander in der Liste, so wird der erste erkannt und richtig verschoben, der dahinterstehende Schüler rückt dadurch einen Listenplatz auf, d.h., er nimmt nun den gerade behandelten Index des Iterator an. Da dieser nun aber eine Position weitergezählt wird, kann dieser Schüler nicht mehr behandelt werden und nicht mehr in die Liste „ignoredStudents“ eingefügt werden. Somit verschwindet er aus dem Verteilungsprozess und ist letztendlich gelöscht.

Beheben kann man diesen Fehler, in dem man den Index nur erhöht, wenn die if-Bedingung nicht zutrifft. Also ist die Lösung dieses Fehlers folgende:

```

For (int i = 0; i < this.allStudents.size(); i++) {
    If ((this.allStudents.get(i).getActiveCourse() != null &&
        this.allStudents.get(i).getActiveCourse()
            .equals(this.ignore())) ||
        this.allStudents.get(i).getCoursesAsList()
            .contains(this.ignore()))
        this.ignoredStudents.add(this.allStudents.remove(i));
    else
        i++;
}

```

2.3.2. Fehler beim Sortieren der Berechnungen nach der Güte

Dieser zweite Fehler, der erläutert wird, entsteht durch das falsche Interpretieren von den Rückgabewerten der „compareTo(Object o)“-Methode. Nur durch Ausprobieren wird diesen Fehler schließlich behoben. Da der Ursprung des Fehlers bekannt war, nämlich die selbst geschriebene „Save.compareTo(Save s)“ Methode, kann durch Ausprobieren der Reihung und Verrechnung der verglichenen Werte stets schnell eine Lösung gefunden werden. Hierzu wird teilweise das Vorzeichen des zurückgegebenen Wertes vertauscht oder auch if-

Bedingungen für Randfälle eingeführt, um so die Speicherung in der richtigen Reihenfolge anordnen zu können. Dabei bewirkt der entsprechende Vorzeichenwechsel entweder, dass die beste oder dass die schlechteste Berechnung vorne in der Liste der Speicher steht und zuerst angezeigt wird. Dieser Fehler ist hier vor allem aus dem Grund aufgeführt, weil er durch zwischenzeitliches Umstellen von den Berechnungen der Güte und anderen zu berücksichtigenden Werten immer wieder aufgetreten ist.

3. Manual

Um den Einstieg in die Nutzung des CaRP-Assigners zu erleichtern, werden im Folgenden die wichtigsten Funktionen erklärt und Schritt für Schritt der Umgang mit dem Programm erläutert.

3.1. Importieren

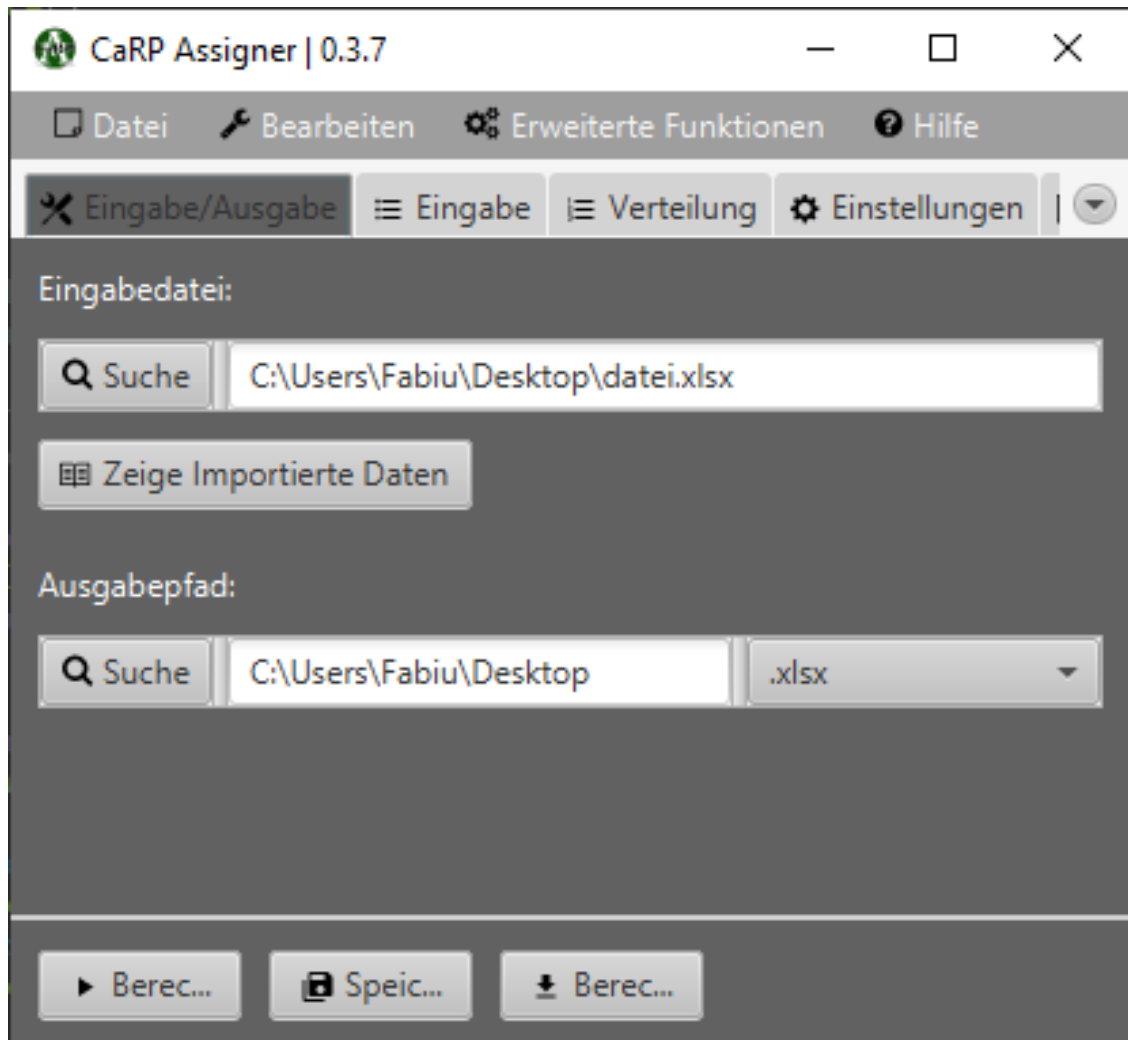


Abb. 9: Der Reiter „Eingabe/Ausgabe“

Gestartet wird zunächst mit dem Importieren (Einfügen) einer Datei in den CaRP-Assigner. Der CaRP-Assigner unterstützt sowohl .csv, .xls und .xlsx Dateien. Um eine Datei zu laden, begeben Sie sich in den Reiter „Eingabe/Ausgabe“. Das Importieren einer Datei kann nun über drei unterschiedliche Wege geschehen:

1. Über Drag and Drop: Nehmen Sie Ihre Datei und ziehen Sie diese über das weiße Textfeld neben dem oberen „Suche“ Knopf. Wenn Sie die Datei nun loslassen, wird sie automatisch in den Assigner geladen.

2. Über die „Suche“ Funktion: Betätigen Sie den oberen „Suche“ Knopf und wählen Sie Ihre Datei aus dem Dateisystem aus. Mit „Öffnen“ können Sie die Datei in das Programm laden.
3. Über das Menü: Sie können auch über den Menü-Reiter „Datei“ und dann über die Funktion „Suche Eingabedatei“ den Dateexplorer zum Importieren der gewünschten Tabellendatei aufrufen.

Nachdem Sie Ihre Datei importiert haben, können Sie sich über den Knopf „Zeige Importierte Daten“, die von dem Programm eingelesenen Daten anzeigen lassen.

Hinweis: Es ist möglich, Daten zu bereits importierten Daten hinzuzufügen. Sollten sich bei diesem Vorgang Daten doppeln, wird Ihnen eine Warnung angezeigt. Die Optionen in diesem Warnfenster können auch in der Konfiguration geändert werden. Wenn sie diese Warnung ausstellen, wird sie Ihnen jedoch nach jedem Neustart wieder angezeigt werden!

3.1.1. Kriterien, die eine Eingabedatei erfüllen muss

Damit eine Datei von dem neuen CaRP-Assigner eingelesen werden kann, muss sie folgende Kriterien erfüllen:

1. Zum Einfügen eines Schülers
 - a. Die Datei muss die unter den Konfigurationseinstellungen zu findende Config „Schüler Kennzeichnung“ beinhalten, um einen Schüler hinzuzufügen. Diese „Schüler Kennzeichnung“ kann entweder in der Spalte „A“ vor den Angaben zum Schüler stehen, sodass dessen Werte erst in der darauffolgenden Spalte „B“ beginnen, oder sie kann als Tabellenbenennung angegeben werden. Dann steht der Nachname des Schülers schon in der Spalte „A“.
 - b. Dabei ist die maximale Kursanzahl zu beachten. Der Schüler wird wie folgt in der Tabelle aufgeschrieben: {Nachname, Vorname, Kurs 1: Fach, Kurs 1: Lehrer, Kurs 2: Fach, Kurs 2: Lehrer, ..., Kurs („maximale Kursanzahl“): Fach, Kurs („maximale Kursanzahl“): Lehrer}
 - c. Wenn mehr Fächer angegeben werden als die „maximale Kursanzahl“, werden auch sie eingelesen, aber sie sind in der Berechnung nicht verfügbar, solange die „maximale Kursanzahl“ kleiner ist als die Anzahl der Fächer. Daher ist es zu empfehlen, dass die „maximale

Kursanzahl“ auf die Kursanzahl des Schülers mit den meisten Kursangaben gesetzt wird!

2. Zum Einfügen eines Kurses:

- a. Die bei den Schülern angegebenen Kurse werden automatisch mit dem Schülermaximum aus der Config Datei initialisiert.
- b. Die Datei muss die unter den Konfigurationseinstellungen zu findende Config „Kurs Kennzeichnung“ beinhalten, um einen Kurs hinzuzufügen. Diese „Kurs Kennzeichnung“ kann entweder in der Spalte „A“ vor den Angaben zum Kurs stehen, sodass dessen Werte erst in der darauffolgenden Spalte „B“ beginnen, oder sie kann als Tabellen Benennung angegeben werden. Dann beginnen die Angaben zum Kurs in Spalte „A“. Der Kurs muss wie folgt in der Tabelle stehen:
{Fach, Lehrer, Maximale Schülerzahl}

3. Um einen Kommentar zu kennzeichnen, muss in der Spalte „A“ als erstes das Zeichen aus der Konfiguration, welches unter „Kommentar Kennzeichnung“ zu finden ist, angegeben werden.

Hinweis: Alle Einstellungen, die hier erwähnt werden, sind unter „Konfiguration>Eingabe“ zu finden.



Abb. 10: Vorschau der Statistik einer Verteilung im light-Theme

3.2. Der Verteilungsprozess

Um eine Verteilung zu berechnen, müssen bereits vorher Daten eingelesen sein. Sollte das der Fall sein, reicht ein Einfaches betätigen des Knopfes „Berechne Verteilung“ aus, um den Verteilungsprozess zu starten. Den Knopf „Berechne Verteilung“ finden Sie unten links auf der graphischen Oberfläche (siehe Abb. 10). Wenn unten rechts eine Prozessanzeige erscheint, welche Ihnen anzeigt, wie weit das Programm mit dem Verteilungsprozess ist, haben Sie den Zuweisungsprozess erfolgreich gestartet.

Sobald die Zuweisung abgeschlossen ist, wird eine Vorschau der zugewiesenen Daten angezeigt. Diese gliedert sich in drei Unteransichten: Die „Kurs-Verteilung“, die die Schüler anzeigt mit dem ihnen zugeordneten Kurs. Die „Schüler-Verteilung“, die die Kurse mit den entsprechenden Schülern, die ihnen zugewiesen wurden, anzeigt und die in Abb. 10 zu sehenden „Statistiken“, welche weitere relevante Daten aufführen.

3.3. Das Exportieren der Daten

Das Exportieren ist wieder sowohl in das .xls, .xlsx als auch .csv Format möglich. Um den Ort auszuwählen, an den Sie Ihre Daten exportieren möchten, gehen Sie wieder in den Reiter „Eingabe/Ausgabe“ aus Abb. 9. Hier betätigen Sie den unteren „Suchen“ Knopf, um einen Ausgabepfad auszuwählen. Wählen Sie nun den gewünschten Ausgabeordner und bestätigen Sie mit „Ordner Auswählen“. Um das Dateiformat festzulegen, in welches die Daten exportiert werden sollen, kann der rechts in der gleichen Reihe zu findende Knopf verwendet werden, mit dem zwischen den drei Formaten gewählt werden kann. Die ausgewählte Möglichkeit wird dann als Text des Knopfes angezeigt.

Nun kann die Verteilung über den Knopf „Speichere Verteilung“ gespeichert werden. In dem gewünschten Ordner finden Sie nun zum einen eine „.log“-Datei, in der sich der Log befindet, dann eine „.carp“-Datei, die Sie mit dem KuFA-Zuweiser wieder öffnen können, und zuletzt die exportierte Datei im gewünschten Format.

3.4. Das Bearbeiten der Eingabedaten im CaRP-Assigner

Abb. 11: Bearbeitungsfenster für einen Schüler mit Daten

Unter dem Reiter „Eingabe“ finden Sie eine Vorschau aller der von Ihnen importierten Daten. Um diese Daten zu bearbeiten, wählen Sie den Eintrag in der Liste der Schüler oder der Liste der Kurse aus, den Sie gerne bearbeiten möchten. Mit einem Rechtsklick werden Ihnen nun die Bearbeitungsfunktionen angezeigt. Alternativ finden Sie im Menü unter dem Reiter „Bearbeiten“ ebenfalls für den ausgewählten Kurs oder Schüler die folgenden Bearbeitungsfunktionen:

1. „Füge hinzu“: Mit dieser Funktion kann ein neuer Schüler/Kurs in das Programm eingefügt werden. Dabei erscheint eine grafische Oberfläche, in die Sie die gewünschten Daten eintragen können. Wenn Sie fertig sind, betätigen Sie den Knopf „Speichern“ und der neue Schüler/Kurs wird automatisch eingefügt.
2. „Bearbeite“: Mit dieser Funktion ist es möglich, einen Schüler/Kurs zu bearbeiten. Hier erscheint dasselbe GUI wie beim Einfügen des gewünschten

Schülers/Kurses. Die Felder sind nun dem Kurs oder Schüler entsprechend ausgefüllt und können daher beliebig verändert werden. Indem Sie „Speichern“ drücken, werden die Daten überschrieben.

3. „Entferne“: Mit dieser Funktion ist es möglich, den aktuell ausgewählten Kurs/Schüler zu löschen.

Hinweis: Groß und Kleinschreibung wird bei den Kursen nicht berücksichtigt.

3.5. Das Bearbeiten der Verteilung im CaRP-Assigner

Fach	Lehrer	Schüler 1		Schüler 2		Schüler 3	
		Vorname	Nachname	Vorname	Nachname	Vorname	Nachname
B	A	Max	Musterman...	Max	Musterman...	Max	Musterman...
D	C	Max	Musterman...	Max	Musterman...	Max	Musterman...
F	E	Max	Musterman...	Max	Musterman...	Max	Musterman...

Abb. 12: Fenster zum Bearbeiten eines Schülers in einer Verteilung

Auch an den verteilten Daten können noch Veränderungen vorgenommen werden. Dazu kann der Schüler/Kurs, der bearbeitet werden soll, ausgewählt werden und dann die gewünschte Aktion, wie auch beim Bearbeiten der Eingabedaten, ausgeführt werden:

1. „Bearbeite“: Ermöglicht es bei Schülern, diesen in einen anderen seiner gewählten Kurse zu setzen oder seinen Namen zu verändern, und bei Kursen, deren Namen zu verändern. Dazu erscheint, wie auch beim Bearbeiten der Eingabedaten, eine graphische Oberfläche, an der die Veränderungen vorgenommen werden können.

2. „Entferne“: Entfernt den Schüler oder Kurs aus der Berechnung. Dabei können aus Sicherheitsgründen lediglich leere Kurse entfernt werden!

Um einen Kurs oder Schüler hinzuzufügen, können die über den Tabellen zu findenden Knöpfe „Füge Schüler hinzu“ und „Füge Kurs hinzu“ genutzt werden.

Da bei einem Verteilungsprozess mehrere Verteilungen erstellt werden, ist es mit Hilfe der Knöpfe „Vorherige“ und „Nächste“ auch möglich, zwischen den besten fünf Verteilungen zu wechseln.

Um nur die Kurse mit Schülern in der Verteilung zu haben, können auch direkt alle leeren Kurse über den Knopf „Entferne leere Kurse“ gelöscht werden.

Wenn Sie eine Vorschau der Datei in Excel sehen möchten, ist dies über den Knopf „Öffne Tabellenkalkulation“ möglich.

3.6. Die Einstellungen

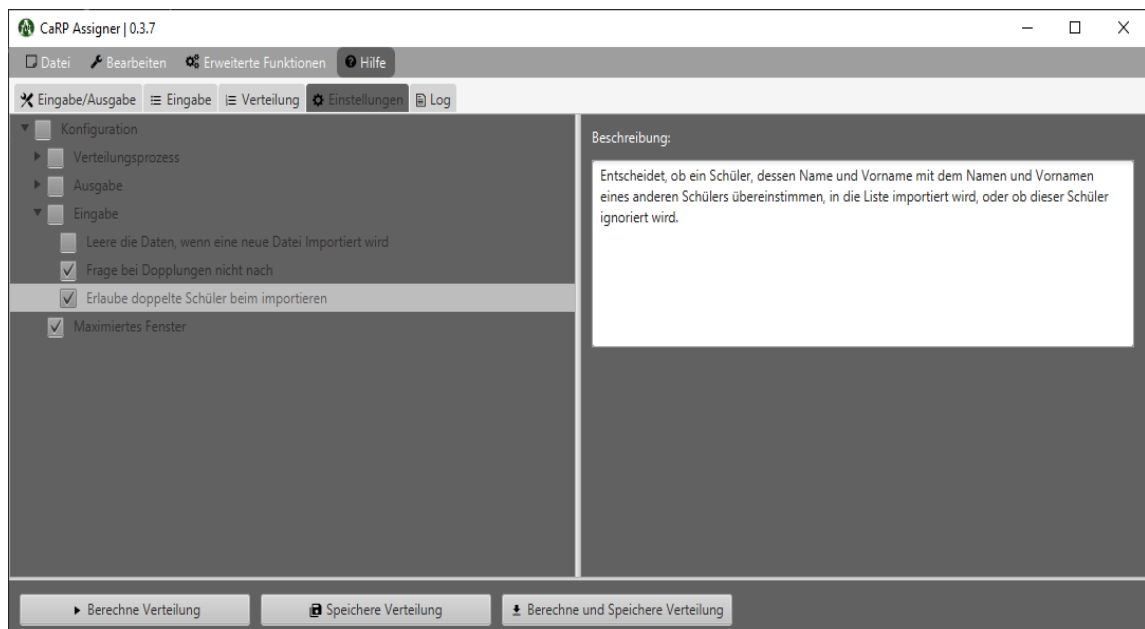


Abb.13: Konfigurationsfenster mit rechts eingeblendetem Beschreibungstext für die links ausgewählte Einstellung

Abschließend wird nun der Reiter „Einstellungen“ erklärt. Hier befinden sich alle Konfigurationsmöglichkeiten. Dabei können im Baum (links), die gewünschten Pfade ausgewählt werden, wodurch auf der rechten Hälfte die zugeordneten Konfigurationsmöglichkeiten angezeigt werden. Einstellungen, die nur an- und ausgeschaltet werden können, sind im Baum mitaufgeführt. Werden diese ausgewählt, erscheint rechts eine Beschreibung über deren Funktion. Allen anderen Einstellungen sind Hovertexte, also Texte, die beim Darüberfahren mit der Maus erscheinen, beigelegt, die ebenfalls deren Funktion erläutern.

4. Fazit

Abschließend soll noch einmal der CaRP-Assigner betrachtet werden und einige weitere Verbesserungsmöglichkeiten aufgeführt werden sowie die gesammelten Erfahrungen herausgestellt werden.

4.1. Optimierungsmöglichkeiten

Viele Verbesserungen wurden in dieser Version des CaRP-Assigners bereits umgesetzt. Dennoch lassen sich immer wieder weitere Verbesserungen und Vereinfachungen finden.

Zuerst will ich hier noch einen weiteren möglichen Weg zum Einlesen der Daten erwähnen. So könnte man das Importieren von Daten aus einer Datenbank ermöglichen. Dies würde später auch die Kurswahlen der Schüler mithilfe einer Web-Applikation erlauben.

Auch bei Betrachtung der graphischen Oberfläche können noch kleine Änderungen zum Verbessern vorgenommen werden, wie beispielsweise das Hinzufügen weiterer Designarten, um das graphische Aussehen auf jeden Nutzer anzupassen. Zudem wäre eine Erweiterung der zur Auswahl stehenden Sprachen möglich und eine Überarbeitung der englischen Sprachversion.

4.2. Zusammenfassung

Abschließend lässt sich die Arbeit als große Bereicherung meiner Fähigkeiten in Java sehen. Neben dem Erlernen vom Arbeiten mit „css-Styling“ und Maven konnte ich auch meine Programmierkenntnisse, unter anderem durch das Erlernen der Funktion von Lambda Ausdrücken weitreichend verbessern. Auch im Umgang mit GitHub konnte ich so meine Kenntnisse weiter schulen. In diesem Sinne ist festzuhalten, dass diese besondere Lernleistungen mich inner- als auch außerunterrichtlich stark gefordert, sowie gefördert hat.

5. Anhang

In den folgenden Unterpunkten befindet sich eine Übersicht über die Klassen des CaRP-Assigners. Für nähere Kommentare, sowie den Quellcode liegt der Ausarbeitung eine CD mit den Daten, sowie der Version 0.3.8 des CaRP-Assigners bei. Alternativ kann das Projekt auch auf GitHub (<https://github.com/juhu1705/CaRP>) eingesehen werden.

5.1. Konfiguration

Name

@Interface ConfigElement - Diese Annotation verifiziert ein Attribut, als Konfigurationsattribut.

Synopsis

```
public @interface ConfigElement {  
    public String defaultValue();  
    public Class elementClass();  
    public String description();  
    public String name();  
    public String location();  
}
```

Name

Class ConfigManager – Diese Klasse dient zur Verwaltung aller Konfigurationselemente.

Synopsis

```
public class ConfigManager {  
    private static ConfigManager instance;  
    private ArrayList<Field> fields;  
  
    public static ConfigManager getInstance();  
    public Field getField(String name);  
    public void register(Field configElement) throws IOException;  
    public void register(Class c) throws IOException;  
    public void loadDefault();  
    public void load(String input) throws SAXException, IOException;
```

```
    public void save(File output) throws IOException;
    public void createMenuTree(TreeView<String> tree, VBox configurations);
    public void onConfigChanged();
}
```

Name

Class FieldHandler – Organisiert das Einlesen der Konfigurationsdatei.

Synopsis

```
public class FieldHandler implements ContentHandler {
    private String value, defaultValue, type, name, currentValue;

    public void characters(char[] arg0, int arg1, int arg2) throws
        SAXException;

    public void endDocument() throws SAXException;

    public void endElement(String arg0, String arg1, String arg2) throws
        SAXException;

    public void endPrefixMapping(String arg0) throws SAXException;

    public void ignorableWhitespace(char[] arg0, int arg1, int arg2) throws
        SAXException;

    public void processingInstruction(String arg0, String arg1) throws
        SAXException;

    public void setDocumentLocator(Locator arg0);

    public void skippedEntity(String arg0) throws SAXException;

    public void startDocument() throws SAXException;

    public void startElement(String arg0, String arg1, String arg2, Attributes
        arg3) throws SAXException;

    public void startPrefixMapping(String arg0, String arg1) throws
        SAXException;
}
```

5.2. Dateimanagment (In- und Export)

Name

Class CSVExporter – Verwaltet das Exportieren eines WriteableContents nach CSV

Synopsis

```
public class CSVExporter {
    protected CSVExporter();
}
```

```
    public static void writeCSV(String pathfile, WriteableContent toWrite)
        throws IOException;

    public static void writeCSV(String pathfile, WriteableContent... toWrite)
        throws IOException;
}
```

Name

Class CSVImporter – Verwaltet den Import einer csv Datei in einen WriteableContent

Synopsis

```
public class CSVImporter {
    protected CSVImporter();

    public static WriteableContent readCSV(String pathfile) throws
        IOException, URISyntaxException;

    private static InputStream getInput(String name) throws
        URISyntaxException, FileNotFoundException;
}
```

Name

Class ExcelExporter - Verwaltet das Exportieren eines WriteableContents in ein Excel Format

Synopsis

```
public class ExcelExporter {
    public static void writeXLS(String pathfile, WriteableContent toWrite);
    public static void writeXLSX(String pathfile, WriteableContent toWrite);
    public static void writeXLS(String pathfile, WriteableContent...
        toWrite);

    public static void writeXLS(String pathfile, List<WriteableContent>
        toWrite) throws IOException;

    public static void writeXLSX(String pathfile, WriteableContent...
        toWrite) throws IOException;

    public static void writeXLSX(String pathfile, List<WriteableContent>
        toWrite) throws IOException;
}
```

Name

ExcelImporter - Importiert eine Excel Datei aus dem mitgegebenen Pfad.

Synopsis

```
public class ExcelImporter {  
    public static WriteableContent readXLS(String pathfile) throws  
        IOException, URISyntaxException;  
  
    public static WriteableContent readXLSX(String pathfile) throws  
        IOException, URISyntaxException;  
  
    public static List<WriteableContent> readXLSImproved(String pathfile)  
        throws IOException, URISyntaxException;  
  
    public static List<WriteableContent> readXLSXImproved(String pathfile)  
        throws IOException, URISyntaxException;  
  
    private static InputStream getInput(String name);  
}
```

Name

Class LogWriter – Verwaltet das Exportieren der LogDatei.

Synopsis

```
public class LogWriter {  
    public static void writeLog(String pathfile);  
}
```

Name

Vec2i - Repräsentiert einen zweidimensionalen Vektor.

Synopsis

```
public class Vec2i implements Serializable {  
    public int x, y;  
    //Konstruktoren  
    public Vec2i();  
    public Vec2i(int f);  
    public Vec2i(int x, int y);  
    public Vec2i(Vec2i Vec2i);  
    // Methoden  
    public int getLength();  
    public Vec2i normalize();  
    public Vec2i mul(Vec2i Vec2i);  
    public Vec2i mul(int f);  
}
```

```

    public Vec2i div(Vec2i Vec2i);
    public Vec2i div(int f);
    public Vec2i sub(Vec2i Vec2i);
    public Vec2i sub(int f);
    public Vec2i add(Vec2i Vec2i);
    public Vec2i add(int f);
    public Vec2i mod(Vec2i Vec2i);
    public Vec2i mod(int f);
    public Vec2i set(int x, int y);

    @Override
    public boolean equals(Object obj);

    @Override
    protected Vec2i clone() throws CloneNotSupportedException;

    @Override
    public String toString();
}

```

Name

Class WriteableContent - Diese Klasse dient als Schnittstelle zum Im- / Export. Sie speichert die Importierten Daten für das Einlesen zwischen.

Synopsis

```

public class WriteableContent {
    private HashMap<Vec2i, String> lines;
    private String name;

    // Konstruktoren
    public WriteableContent();
    public WriteableContent(String name);

    // Methoden
    public String getName();
    public WriteableContent setName(String name);
    public WriteableContent addCell(Vec2i position, String content);
    public WriteableContent removeCell(Vec2i position)
    public WriteableContent addLine(Vec2i startPosition, String[] contents);
    public WriteableContent addGrid(Vec2i startPosition, String[][] grid);
    public WriteableContent addLine(Vec2i startPosition, List<String>
    contents);
}

```

```

    public WriteableContent addListGrid(Vec2i startPosition,
    List<List<String>> contents);

    public WriteableContent removeLine(Vec2i startPosition, int length);

    public String[][] getReverseGrid();

    public String[][] getGrid();

    public String getStringAt(Vec2i position);

    public void writeXLS(HSSFWorkbook workbook, HSSFSheet sheet, int
    startingLineY);

    public void writeCSV(BufferedWriter writer);

    public void writeXLSX(XSSFWorkbook workbook, XSSFSheet sheet, int
    startingLineY);

    private Vec2i getMaxLength();
}

```

5.3. Verteilungsprozess

Name

Class Course - Diese Klasse bildet einen Kurs ab und beinhaltet alle diesbezüglich wichtigen Informationen.

Synopsis

```

public class Course implements Comparable<Course>, Serializable {
    private ArrayList<Student> students;
    private String teacher;
    private String subject;
    private int maxStudents;

    // Konstruktoren
    public Course(String subject, String teacher, int maxStudents);
    public Course(String subject, String teacher);
    public Course(String[] split) throws NullPointerException;

    // Methoden
    public String getTeacher();
    public String getSubject();

    @Override
    public boolean equals(Object obj);

    public boolean isFull();

    @Override
    public String toString();
}

```

```
public ArrayList<Student> getStudents();
public boolean contains(Student student);
public boolean addStudent(Student student);
public Student removeStudent(Student student);
public Student getStudent(int i);
public int size();
public String studentsToString();
@Override
public int compareTo(Course c);
@Override
protected Object clone() throws CloneNotSupportedException;
public int getMaxStudentCount();
public void setTeacher(String teacher);
public void setSubject(String subject);
public void setStudentMax(int maxStudents);
}
```

Name

Class Distributor – Der Distributor verwaltet den Zuweisungsprozess, sowie das Zwischenspeichern der berechneten Verteilungen.

Synopsis

```
public class Distributor implements Runnable {
    public static PriorityQueue<Save> calculated;
    public static boolean calculate;
    private static int nextID;
    private static Distributor instance;
    ArrayList<Student> loadedallStudents;
    ArrayList<Course> loadedallCourses;
    ArrayList<Student> ignoredStudents;
    Course ignoredCourse;
    ArrayList<Student> allStudents;
    ArrayList<Course> allCourses;
    private ArrayList<Reader> readers;
    // Konstruktoren
    protected Distributor();
}
```

```

public Distributor(String filename);
public Distributor(Save actual, Save... readObject);
// Methoden
public static Distributor getInstance();
public static int getStudentID();
@Override
public void run();
private void loadDataFromSave(Save save);
public void printRate();
public int getStudentsWithRate(int rate);
public int rate();
public void assign();
private boolean isAnyCourseFull();
public    ArrayList[]    copyData(ArrayList<Student>    oldStudents,
ArrayList<Course> oldCourses, Course ignoredCourse2);
public void save();
private ArrayList<Student> getStudentsWithPriority(int priority);
private ArrayList<Student> getUnallocatedStudents();
private int[] getPriorities();
private int countPriority(int priority);
public int getHighestPriorityWhithoutIntegerMax();
public int getHighestPriority();
private void print();
private boolean doesCourseExist(String name);
public ArrayList getCalcStudents();
public ArrayList getIgnoreStudents();
public Course getOrCreateCourseByName(String name);
public Course getCourseByName(String name);
public ArrayList<Course> getCourses();
public Course ignore();
public void addStudent(Student s);
private void readFile(String path);
private    List<WriteableContent>    importFile(String    path)    throws
IOException, URISyntaxException;
private void readGrid(String gridName, String[][] grid, String
filename);

```

```
private boolean isReaderKey(String input);
public boolean addReader(Reader reader);
private void loadReaders();
public void updateStandartReaders();
public Student getStudentByID(int studentID);
public void addCourse(Course c);
public void reset();
public void clear();
public void removeStudent(Student student);
public void removeCourse(Course course);
public void setIgnoreMark(String ignoreStudent);
}
```

Name

Class InformationSave - Diese Klasse speichert alle sonstigen im Fenster Statistiken abgebildeten Informationen und ist eindeutig einem Save zugeordnet.

Synopsis

```
public class InformationSave implements Serializable {
    Save parent;
    private int highestPriority;
    private int rate;
    private double guete;
    private int studentCount;
    private ArrayList<Student> unallocatedStudents;
    private ArrayList<Student> badPriorityStudents;
    private int[] studentPriorities;
    // Konstruktoren
    InformationSave(Save parent);
    // Methoden
    public WriteableContent write();
    public int getHighestPriority();
    public int getRate();
    public int[] getStudentPriorities();
    public ArrayList<Student> getBStudents();
}
```

```
    public ArrayList<Student> getUStudents();  
    public int getStudentCount();  
    public void update();  
    private void updateGuete();  
    private int translatePriority(int priority);  
    public double getGuete();  
}
```

Name

Class Reader – Abstract – Stellt einen Leser da, der eine bestimmte Zeile beim Einlesen verwaltet.

Synopsis

```
public abstract class Reader {  
    String key;  
    // Konstruktor  
    public Reader(String key);  
    // Methoden  
    public boolean isKey(String key);  
    public abstract void read(String[] line, int lineNumber);  
}
```

Name

Class Save - Diese Klasse dient zur Speicherung berechneter Zuweisungen der Klasse Distributor im Arbeitsspeicher und stellt zudem noch Methoden zur Bearbeitung der gespeicherten Daten sowie zum Exportieren der Daten bereit.

Synopsis

```
public class Save implements Comparable<Save>, Serializable {  
    private InformationSave informations;  
    private List<Student> allStudents;  
    private List<Course> allCourses;  
    // Konstruktor  
    public Save(List<Student> editedStudents, List<Student> ignoredStudents,  
        List<Course> allCourses);  
    // Methoden  
    public int getHighestPriority();  
}
```

```

    public int[] getStudentPriorities();
    public int rate(int highestPriority);
    public Course getCourseByName(String name);
    public int getHighestPriorityWhithoutIntegerMax();
    public ArrayList<Student> getStudentsWithPriority(int priority);
    private static List<Course> sortCourse(List<Course> courseToSort);
    public static List<Student> sortStudents(List<Student> studentsToSort);
    public List<Student> getAllStudents();
    public List<Course> getAllCourses();
    public Course[] getAllCoursesAsArray();
    public InformationSave getInformation();
    public List<WriteableContent> writeInformation();
    public WriteableContent writeStudentInformation();
    public WriteableContent writeCourseInformation();

    @Override
    public int compareTo(Save s);

    private boolean sameCalculation(Save s);
    public int compareTo(int guete);
    public void addCourse(Course c);
    public Student getStudentByID(int studentID);
    public boolean addStudent(Student student);
    public void removeStudent(Student student);
    public void removeCourse(Course course);
}

```

Name

Class Student – Diese Klasse bildet einen Schüler ab und beinhaltet alle wichtigen Informationen des Schülers.

Synopsis

```

public class Student implements Comparable<Student>, Serializable {
    private ArrayList<Course> courses;
    private Course activeCourse;
    protected int priority;
    private int id;
    private String name;
}

```



```
private String prename;
private boolean mark;
// Konstruktoren
protected Student();
public Student(String name, String prename, Course... courses);
private Student(String name, String prename, int id);
// Methoden
private void generateID();
public boolean next();
public boolean onlyNext();
public void setName(String name);
public String getName();
public void setPrenome(String prename);
public String getPrenome();
public void addCourse(Course course);
public void addCourse(int index, Course course) throws
IndexOutOfBoundsException;
public Course[] getCourses();
private Course setNextCourse();
private Course setCourse(Course course);
public Course getNextCourse();
public Course getNextCourse(int iterator);
public Course getNextCourse(Course course, int iterator);
public int getCourseAmount(Course course);
public Course getActiveCourse();
public void setActiveCourse(Course activeCourse);
public void refreshPriority();
private int calculatePriority();
public int getPriority();
public int getRate();
public int getRate(int highestPriority);
public void mark();
public void unmark();
public boolean isMarked();
```

```

    @Override
    public int compareTo(Student s);

    @Override
    public boolean equals(Object obj);

    @Override
    public String toString();

    @Override
    protected Object clone() throws CloneNotSupportedException;

    public int getPosition(Course c);

    public boolean idequals(int studentID);

    public int getID();

    public void setCourses(Course... c);

    public List<Course> getCoursesAsList();

    public void checkMarkt(int highestPriority);
}

```

5.4. Graphische Oberfläche

Name

Class CourseView - Behandelt die Tabelle zur Ansicht der importierten Kursliste.

Synopsis

```

public class CourseView {
    TableView inputTable;

    // Konstruktor

    public CourseView(TableView<Course> inputTable);

    // Methode

    public void fill();
}

```

Name

Class InputView - Behandelt die Tabelle zur Ansicht der importierten Schülerliste.

Synopsis

```

public class InputView {
    TableView inputTable;
}

```

```
// Konstruktor
public InputView(tableView<Student> inputTable);

// Methode
public void fill();
}
```

Name

Class OutputCourseView - Behandelt die Tabelle zur Ansicht der verteilten Kurse.

Synopsis

```
public class OutputCourseView implements Runnable {
    TableView tv;
    ArrayList<TableColumn<Course, String>> students;
    // Konstruktor
    public OutputCourseView(tableView<Course> inputTable);
    // Methoden
    public void fill();
    private void loadMaxStudents();
    @Override
    public void run()
}
```

Name

Class OutputInformationView - Behandelt alle unter Statistiken angezeigten Tabellen.

Synopsis

```
public class OutputInformationView implements Runnable {
    public ArrayList<TableColumn<Student, String>> scourses;
    // Methoden
    public void fill();
    @Override
    public void run();
}
```

Name

Class OutputStudentsView - Behandelt die Tabelle zur Ansicht der verteilten Schüler.

Synopsis

```
public class OutputStudentsView implements Runnable {  
    TableView tv;  
    // Konstruktor  
    public OutputStudentsView(TableView<Student> inputTable);  
    // Methoden  
    public void fill();  
    @Override  
    public void run();  
}
```

Name

Class SwitchCourseView - Behandelt die Tabelle zum Bearbeiten / Hinzufügen von Schülern zu einer Verteilung.

Synopsis

```
public class SwitchCourseView implements Runnable {  
    TableView tv;  
    ArrayList<TableColumn<Course, String>> students;  
    Student student;  
    // Konstruktor  
    public SwitchCourseView(TableView<Course> inputTable, Student student);  
    // Methoden  
    public void fill();  
    @Override  
    public void run();  
}
```

Name

Class AboutManager - Behandelt alle Aktionen des About Fensters.

Synopsis

```
public class AboutManager {  
    @FXML  
    public TextField weg;  
  
    // Methoden  
  
    public void openLink(ActionEvent event);  
    public void startMail(ActionEvent event);  
    public void openLSPage(MouseEvent event);  
    public void onHelpSearch(ActionEvent event);  
}
```

Name

Class AddCourseManager - Behandelt alle Aktionen des Fensters zum Hinzufügen und bearbeiten von Kursen zu den importierten Daten.

Synopsis

```
public class AddCourseManager implements Initializable {  
    public static String s, t;  
    public static int mS;  
  
    @FXML  
    private TextField subject;  
  
    @FXML  
    private TextField teacher;  
  
    @FXML  
    private Spinner<Integer> maxStudents;  
  
    // Methoden  
  
    public void onAdd(ActionEvent event);  
  
    @Override  
    public void initialize(URL location, ResourceBundle resources);  
}
```

Name

Class AddCourseToSaveManager - Behandelt alle Aktionen des Fensters zum Hinzufügen von Kursen zu einer bestehenden Verteilung.

Synopsis

```
public class AddCourseToSaveManager implements Initializable {  
    public static String s, t;
```

```

    public static int mS = -2;

    @FXML
    private TextField subject;

    @FXML
    private TextField teacher;

    @FXML
    private Spinner<Integer> maxStudents;

    // Methoden

    public void onAdd(ActionEvent event)

    @Override
    public void initialize(URL location, ResourceBundle resources)

}

```

Name

Class AddStudentManager - Behandelt alle Aktionen des Fensters zum Hinzufügen und Bearbeiten von Schülern zu den importierten Daten.

Synopsis

```

public class AddStudentManager implements Initializable {

    public ArrayList<TextField> courses;

    public static int studentID;

    @FXML
    public TextField prename, name, c1f, c1t;

    @FXML
    public VBox vBox;

    public TextField last;

    public int i;

    // Methoden

    public void onAdd(ActionEvent event);

    @Override
    public void initialize(URL location, ResourceBundle resources);

}

```

Name

Class AddStudentToSaveManager - Behandelt alle Aktionen des Fensters zum Hinzufügen und Bearbeiten von Schülern zur Verteilung.

Synopsis

```

public class AddStudentToSaveManager implements Initializable {

    public static Student student;

}

```

```
@FXML
private TableView<Course> courses;

@FXML
private TextField prename;

@FXML
private TextField name;

@FXML
private TableColumn<Course, String> teacher, subject;

private SwitchCourseView scw;

// Methoden

public void onSetActive(ActionEvent event);

public void onFinished(ActionEvent event);

@Override
public void initialize(URL location, ResourceBundle resources);
}
```

Name

Class ErrorGuiController - Behandelt alle Aktionen des Fehler Fensters.

Synopsis

```
public class ErrorGuiController implements Initializable {

    public static String headline, information;

    @FXML
    public TextArea tinformation, theadline;

    // Methoden

    public void onOK(ActionEvent event);

    @Override
    public void initialize(URL location, ResourceBundle resources);
}
```

Name

Class FullProgress - Behandelt alle Aktionen der Prozessleiste.

Synopsis

```
public class FullProgress {

    private static FullProgress instance;

    private DoubleProperty progress;

    // Methoden
```

```
    public static FullProgress getInstance();  
    public final double getProgress();  
    public final void setProgress(double progress);  
    public final DoubleProperty progressProperty();  
}
```

Name

Class GUIDoubleStudentManager - Behandelt alle Aktionen des Fensters, dass erscheint, wenn ein doppelter Schüler beim Importieren einer Datei auftritt.

Synopsis

```
public class GUIDoubleStudentManager implements Initializable {  
    public static String sName = "", sPrenome = "";  
    public static boolean finished = false;  
  
    @FXML  
    public TextField name, prename;  
  
    @FXML  
    public CheckBox shouldMemorice;  
  
    // Methoden  
  
    public void skip(ActionEvent event);  
    public void add(ActionEvent event);  
  
    @Override  
    public void initialize(URL location, ResourceBundle resources);  
}
```

Name

Class GUILoader - Diese Klasse stellt die Mainklasse des CaRP-Assigners da. Von hier werden die Startprozesse eingeleitet und die Sprachdateien geladen.

Synopsis

```
public class GUILoader extends Application {  
    private static Stage primaryStage;  
    public static Stage secondaryStage;  
    public static Scene scene;  
    private static File toLoad;  
  
    // Methoden
```



```

    public static void main(String[] args);

    @Override
    public void start(Stage primaryStage) throws Exception;

    public void starting2() throws IOException;

    public static Stage getPrimaryStage();

    private void loadLanguage();
}

```

Name

Class GUIManager - Diese Klasse verwaltet alle Aktionen des Haupt-GUIs.

Synopsis

```

public class GUIManager implements Initializable {

    private static GUIManager instance;

    public static GUIManager getInstance();

    public static Save actual;

    @FXML
    public ImageView i0;

    @FXML
    public ProgressBar p0;

    @FXML
    public TextArea ta1;

    @FXML
    public TreeView<String> configurationTree;

    @FXML
    public Label counter;

    @FXML
    public TextField t1, t2;

    @FXML
    public Button r1, r2, r3, r4, r5, b1, b2, b3, b4, b5, b6;

    @FXML
    public ComboBox<Level> cb1;

    @FXML
    public ComboBox<String> cb2;

    @FXML
    public ComboBox<PrintFormat> cb0;

    @FXML
    public TableView<Student> tv0, tv1, unallocatedStudents, bStudents;
}

```

```

@FXML
public TableView<Entry<String, Double>> rates;

@FXML
public TableView<Entry<Integer, Integer>> priorities;

@FXML
public TableView<Course> tv1, tv2;

@FXML
public TableColumn<Student, String> vtc, ntc, k1tc, k1stc, k1ttc, cvtc,
cntc, cptc, ckstc, ckttc, unallocatedName, unallocatedPrenome, bName,
bPrenome, bSubject, bTeacher, bPriority;

@FXML
public TableColumn<Course, String> subject, teacher, oSubject, oTeacher,
maxStudentCount;

@FXML
public TableColumn<Entry<String, Double>, String> rate, rateV;

@FXML
public TableColumn<Entry<Integer, Integer>, String> priority, swpriority,
percentualPriorities;

@FXML
public Tab students, teachers, statistics, tabStudents, tabCourses,
tabInput, tabOutput;

@FXML
public TabPane masterTabPane;

@FXML
public Menu menuStudent, menuCourse;

public ArrayList<TableColumn<Student, String>> atci;

public InputView inputView;

public CourseView cView;

public OutputStudentsView outputSView;

public OutputCourseView outputCView;

public OutputInformationView outputIView;

@FXML
public VBox config;

@FXML
public ListView<String> lv0;

@FXML
public CheckMenuItem mb0, mb1;

private CheckMenuItem last, lastSwitch;

@FXML
public AnchorPane ap0;

FadeTransition ft;

TranslateTransition tt;

```

```
ScaleTransition st;

// Methoden

public void onDragOver(DragEvent event);
public void onFullScreen(ActionEvent event);
public void onDeleteStudentFromActualSave(ActionEvent event);
public void onRemoveUnusedCourses(ActionEvent event);
public void onDeleteCourseFromActualSave(ActionEvent event);
public void onAddStudentToActualSave(ActionEvent event);
public void onEditCourseActualSave(ActionEvent event);
public void onAddCourseToActualSave(ActionEvent event);
public void onNextSave(ActionEvent event);
public void onPreviousSave(ActionEvent event);
public void addCourse(ActionEvent event);
public void onClearCalculations(ActionEvent event);
public void onThemeChange(ActionEvent event);
public void moveBadStudentToFirst(MouseEvent event);
public void moveUnallocatedStudentToFirst(MouseEvent event);
public void onDeleteCourse(ActionEvent event);
public void onEditCourse(ActionEvent event);
public void onCourseChangeRequest(MouseEvent event);
public void onDeleteStudent(ActionEvent event);
public void onSwitchCourse(ActionEvent event);
public void onSwitchCourseUnallocatedStudent(ActionEvent event);
public void onEditStudent(ActionEvent event);
public void onStudentChangeRequest(MouseEvent event);
public void onDragDroppedInput(DragEvent event);
public void addStudent(ActionEvent event);
public void onFileTypeChanged(ActionEvent event);
public void onTabIn(Event event);
public void onShowImportedData(ActionEvent event);
public void onSelectionChangedStudent(Event event);
public void searchActionInput(ActionEvent event);
public void onAbout(ActionEvent event);
public void printFormatChanged(ActionEvent event);
public void levelChanges(ActionEvent event);
```

```
public void searchActionOutput(ActionEvent event);
public void onDragDroppedOutput(DragEvent event);
public void runAction(ActionEvent event);
public void startErrorFrame(String headline, String message);
public void addCourseToActual(ActionEvent event);
public void onClearDistributor(ActionEvent event);
public void saveAction(ActionEvent event);
public void saveActualAction(ActionEvent event);
public void runAndSaveAction(ActionEvent event);
public void onSaveLog(ActionEvent event);
public void getInformation(ActionEvent event);
public void onSaveConfig(ActionEvent event);
public void close(ActionEvent event);
public void onHelp(ActionEvent event);
public void onShowInExcel(ActionEvent event);
public void updateInputView();

@Override
public void initialize(URL location, ResourceBundle resources);
public void startPicture();
public void save(String location);
public void onNew(ActionEvent event);
public void onLoad(ActionEvent event);
public void load(String location);
public void onSetEnglish(ActionEvent event);
public void onSetGerman(ActionEvent event);
}
```

Name

Class PartProgress - Behandelt alle Prozesse der versteckten Prozessleiste, die nie angezeigt wird. (Kann später noch eingefügt werden, ist aber nicht notwendig)

Synopsis

```
public class PartProgress {
    private static PartProgress instance;
    public static PartProgress getInstance();
}
```

```
private DoubleProperty progress;

// Methoden

public final double getProgress()

public final void setProgress(double progress);

public final DoubleProperty progressProperty();

}
```

Name

Class ProgressIndicator - Verwaltet die sichtbare Fortschrittsanzeige des Prozesses.

Synopsis

```
public class ProgressIndicator {

    private static ProgressIndicator instance;

    public static ProgressIndicator getInstance();

    private int fProgressMax, fProgressValue, aProgressMax, aProgressValue;

    // Konstruktor

    protected ProgressIndicator();

    // Methoden

    public ProgressIndicator setfProgressMax(int value);

    public ProgressIndicator setfProgressValue(int value);

    public ProgressIndicator addfProgressValue(int value);

    public ProgressIndicator setaProgressMax(int value);

    public ProgressIndicator setaProgressValue(int value);

    public ProgressIndicator addaProgressValue(int value);

    public int getfProgressMax();

    public int getfProgressValue();

    public int getaProgressMax();

    public int getaProgressValue();

}
```

Name

Class SwitchCourseManager - Behandelt alle Aktionen des Fensters zum Bearbeiten von Kursen in der Verteilung.

Synopsis

```
public class SwitchCourseManager implements Initializable {  
    public static Student student;  
  
    @FXML  
    private TableView<Course> courses;  
  
    @FXML  
    private TextField prename;  
  
    @FXML  
    private TextField name;  
  
    @FXML  
    private TableColumn<Course, String> teacher, subject;  
  
    private SwitchCourseView scw;  
  
    // Methoden  
  
    public void onSetActive(ActionEvent event);  
    public void onFinished(ActionEvent event);  
  
    @Override  
    public void initialize(URL location, ResourceBundle resources);  
}
```

5.5. Utils

Name

Class CellStyles - Beinhaltet alle Zell-Formatierungsformen, die für die Excel Zellformatierung benutzt werden.

Synopsis

```
public class CellStyles {  
    // Methods  
  
    public static HSSFCellStyle title(HSSFWorkbook workbook);  
    public static HSSFCellStyle normal2(HSSFWorkbook workbook);  
    public static HSSFCellStyle normal1(HSSFWorkbook workbook);  
    public static HSSFCellStyle up(HSSFWorkbook workbook);  
  
    public static XSSFCellStyle title(XSSFWorkbook workbook);  
    public static XSSFCellStyle normal2(XSSFWorkbook workbook);  
    public static XSSFCellStyle normal1(XSSFWorkbook workbook);  
    public static XSSFCellStyle up(XSSFWorkbook workbook);  
}
```

Name

Class Config – Diese Klasse enthält alle Konfigurationsvariablen

Synopsis

```
public class Config {

    @ConfigElement(defaultValue = "OFF", elementClass = Level.class,
description = "loglevel.description", name = "loglevel.text", location =
"config")
    public static Level maxPrintLevel;

    @ConfigElement(defaultValue = "", elementClass = String.class,
description = "inputfile.description", name = "inputfile.text", location =
"config.import")
    public static String inputFile;

    @ConfigElement(defaultValue = ".xlsx", elementClass = String.class,
description = "outputfiletype.description", name = "outputfiletype.text",
location = "config.export")
    public static String outputFileType;

    @ConfigElement(defaultValue = "%localappdata%\\Local\\CaRP",
elementClass = String.class, description = "outputdirectory.description",
name = "outputdirectory.text", location = "config.export")
    public static String outputFile;

    @ConfigElement(defaultValue = "@PJK", elementClass = String.class,
description = "ignoremark.description", name = "ignoremark.text",
location = "config.import")
    public static String ignoreStudent;

    @ConfigElement(defaultValue = "@Course", elementClass = String.class,
description = "coursemark.description", name = "coursemark.text",
location = "config.import")
    public static String newCourse;

    @ConfigElement(defaultValue = "Student", elementClass = String.class,
description = "studentmark.description", name = "studentmark.text",
location = "config.import")
    public static String newStudent;

    @ConfigElement(defaultValue = "#", elementClass = String.class,
description = "commentmark.description", name = "commentmark.text",
location = "config.import")
    public static String commentLine;

    @ConfigElement(defaultValue = "ALL", elementClass = String.class,
description = "printformat.description", name = "printformat.text",
location = "config")
    public static String printFormat;

    @ConfigElement(defaultValue = "false", elementClass = Boolean.class,
description = "shouldMaximize.description", name = "shouldMaximize.text",
location = "config")
    public static boolean shouldMaximize;

    @ConfigElement(defaultValue = "false", elementClass = Boolean.class,
description = "shortnames.description", name = "shortnames.text",
```

```

        location = "config.export")
    public static boolean shortNames;

    @ConfigElement(defaultValue = "true", elementClass = Boolean.class,
        description = "firstprename.description", name = "firstprename.text",
        location = "config.export")
    public static boolean firstPrename;

    @ConfigElement(defaultValue = "true", elementClass = Boolean.class,
        description = "usenewgoodness.description", name = "usenewgoodness.text",
        location = "config.calculation")
    public static boolean useNewGoodness;

    @ConfigElement(defaultValue = "true", elementClass = Boolean.class,
        description = "clearcalculationdata.description", name =
        "clearcalculationdata.text", location = "config.import")
    public static boolean clear;

    @ConfigElement(defaultValue = "false", elementClass = Boolean.class,
        description = "dontask.description", name = "dontask.text", location =
        "config.import")
    public static boolean rememberDecision;

    @ConfigElement(defaultValue = "true", elementClass = Boolean.class,
        description = "allowDoubles.description", name = "allowDoubles.text",
        location = "config.import")
    public static boolean allowDoubleStudents;

    @ConfigElement(defaultValue = "100", elementClass = Integer.class,
        description = "runcount.description", name = "runcount.text", location =
        "config.calculation")
    public static int runs;

    @ConfigElement(defaultValue = "5", elementClass = Integer.class,
        description = "newcalculating.description", name = "newcalculating.text",
        location = "config.calculation")
    public static int newCalculating;

    @ConfigElement(defaultValue = "5", elementClass = Integer.class,
        description = "improvecalculation.description", name =
        "improvecalculation.text", location = "config.calculation")
    public static int improvingOfCalculation;

    @ConfigElement(defaultValue = "3", elementClass = Integer.class,
        description = "coosemaximum.description", name = "coosemaximum.text",
        location = "config.import")
    public static int maxChooses;

    @ConfigElement(defaultValue = "3", elementClass = Integer.class,
        description = "studentlimit.description", name = "studentlimit.text",
        location = "config.import")
    public static int normalStudentLimit;

    @ConfigElement(defaultValue = "3", elementClass = Integer.class,
        description = "rateindex.description", name = "rateindex.text", location =
        "config.calculation")
    public static int powValue;
}

```


Name

Class `LoggingFormatter` - Formatiert den Log entsprechend den Konfigurationseinstellungen.

Synopsis

```
public class LoggingFormatter extends Formatter {  
    private static final DateTimeFormatter timeformatter;  
    // Methode  
    @Override  
    public String format(LogRecord record);  
}
```

Name

Class `LoggingHandler` - Behandelt die Log Nachrichten und speichert sie zwischen.

Synopsis

```
public class LoggingHandler extends Handler implements Runnable {  
    private StringBuilder log;  
    private ArrayList<LogRecord> history;  
    private ArrayList<TextArea> logPrinter;  
    private boolean isRunning, changed;  
    // Methoden  
    public StringBuilder getLog();  
    public void clear();  
    public void updateLog();  
    public void bindTextArea(TextArea textArea);  
    @Override  
    public void run();  
    @Override  
    public void publish(LogRecord record);  
    @Override  
    public void flush();  
    @Override  
    public void close() throws SecurityException;  
}
```

Name

Class MergeSort - Ein einfacher Threadbasierter MergeSort-Algorithmus zum Sortieren von Comparable-Objekten.

Synopsis

```
public class MergeSort<T extends Comparable> implements Callable {  
    private ArrayList<T> input;  
    private ExecutorService pool;  
    // Konstruktor  
    public MergeSort(ArrayList<T> input, ExecutorService pool);  
    // Methoden  
    @Override  
    public ArrayList<T> call() throws Exception;  
    private ArrayList<T> mergeThreadless(ArrayList<T> input, ExecutorService  
    pool) throws Exception;  
}
```

Name

Enum PrintFormat - Das Print Format ist wichtig für die Logger Klasse und bestimmt die im Log mit angegebenen Zusatzinformationen.

Synopsis

```
public enum PrintFormat {  
    LOGGER(3), CLASS(4), ONLY_MESSAGE(0), TIME(1), LEVEL(2), ALL(100);  
    private int level;  
    PrintFormat(int level);  
    public int getLevel();  
}
```

Name

Class PriorityQueue - Eine kleine Listenklasse basierend auf der ArrayList, die zum sortierten Speichern der hinzugefügten Objekte dient.

Synopsis

```
public class PriorityQueue<T extends Comparable<T>> implements List<T> {  
    public ArrayList<T> list;  
    // Konstruktoren  
    public PriorityQueue()
```

```
public PriorityQueue(int i);

// Methoden

public T poll();

public T peek();

public T previous(T actual);

public T next(T actual);

public int indexOf(T actual);

@Override
public int size();

@Override
public boolean isEmpty();

@Override
public boolean contains(Object o);

@Override
public Iterator<T> iterator();

@Override
public Object[] toArray();

@Override
public <T> T[] toArray(T[] a);

@Override
public boolean add(T e);

@Override
public boolean remove(Object o);

@Override
public boolean containsAll(Collection<?> c);

@Override
public boolean addAll(Collection<? extends T> c);

@Override
public boolean addAll(int index, Collection<? extends T> c);

@Override
public boolean removeAll(Collection<?> c);

@Override
public boolean retainAll(Collection<?> c);

@Override
public void clear();

@Override
public T get(int index);

@Override
public T set(int index, T element);

@Override
public void add(int index, T element);
```

```

    @Override
    public T remove(int index);

    @Override
    public int indexOf(Object o);

    @Override
    public int lastIndexOf(Object o);

    @Override
    public ListIterator<T> listIterator();

    @Override
    public ListIterator<T> listIterator(int index);

    @Override
    public List<T> subList(int fromIndex, int toIndex);
}

```

Name

Class Util – Beinhaltet einige nützliche Methoden.

Synopsis

```

public class Util {
    // Methoden

    public static boolean isBlank(String input);

    public static boolean endsWith(String toCheck, String... strings);

    public static String[] removeFirst(String[] line);

    public static int maxStudentCount(List<Course> courses);

    public static boolean isIgnoreCourse(String... name);

    public static Stage openWindow(String resourceLocation, String title,
        Stage parent, boolean darkTheme);
}

```

Name

Class References - Beinhaltet alle Referenzobjekte.

Synopsis

```

public class References {
    public static StreamHandler sh;

    public static LoggingFormatter lf;

    public static ResourceBundle Language;

    public static final LoggingHandler LOGGING_HANDLER;
}

```

```
public static final String HOME_FOLDER;  
public static final Logger LOGGER;  
public static final String VERSION;  
public static final String PROJECT_NAME;  
public static final Random RAND_GEN;  
}
```

6. Quellen- und Literaturverzeichnis

Facharbeit des Schülers Daniel Salomon aus dem Schuljahr 2016 zur besonderen Lernleistung „KuFA-Zuweiser“

6.1. Internet

Stack Overflow: „community of programmer helping each other“,

<http://stackoverflow.com/questions/>

Maven Repository: „Search/Browse/Explore“,

<https://mvnrepository.com/>

GitHub: „The world’s leading software development platform“,

<https://github.com/>

Eclipse,

<https://www.eclipse.org/>

Scene Builder,

<https://gluonhq.com/products/scene-builder/>

6.2. Bilder

Abb. 1: Alte KuFA-Anwendung

Quelle: Selbstangefertigter Screenshot von der alten KuFA-Anwendung

Abb. 2: Klassenstruktur des alten KuFA-Zuweisers

Quelle: Selbstangefertigter Screenshot vom Klassenbaum des alten KuFA-Zuweisers

Abb. 3: Klassenstruktur des CaRP-Assigners

Quelle: Selbstangefertigter Screenshot vom Klassenbaum des CaRP-Assigners

Abb. 4: Eingabedatenvorschau des CaRP-Assigners

Quelle: Selbstangefertigter Screenshot vom lauffähigen CaRP-Assigner

Abb. 5: Ansicht des GUIs im „Scene Builder“

Quelle: Selbstangefertigter Screenshot von der GUI Vorschau im „Scene Builder“

Abb. 6: „Github Repository“ des CaRP-Assigners

Quelle: Selbstangefertigter Screenshot vom GitHub Repository des CaRP-Assigners (stand: 13.03.2020)

Abb. 7: Exporttabelle der Kurs-verteilung mit Beispieldaten

Quelle: Selbstangefertigter Screenshot von einer aus dem CaRP-Assigner exportierten Beispieldatei

Abb. 8: Überarbeitete Einstellungsansicht

Quelle: Selbstangefertigter Screenshot vom lauffähigen CaRP-Assigner

Abb. 9: Der Reiter „Eingabe/Ausgabe“

Quelle: Selbstangefertigter Screenshot vom lauffähigen CaRP-Assigner

Abb. 10: Vorschau der Statistik einer Verteilung im Light Theme

Quelle: Selbstangefertigter Screenshot vom lauffähigen CaRP-Assigner

Abb. 11: Bearbeitungsfenster für einen Schüler mit Daten

Quelle: Selbstangefertigter Screenshot vom lauffähigen CaRP-Assigner

Abb. 12: Fenster zum Bearbeiten eines Schülers in einer Verteilung

Quelle: Selbstangefertigter Screenshot vom lauffähigen CaRP-Assigner

Abb. 13: Konfigurationsfenster mit rechts eingeblendetem Beschreibungstext für die links ausgewählte Einstellung

Quelle: Selbstangefertigter Screenshot vom lauffähigen CaRP-Assigner

6.3. Java-Abhängigkeiten

POI, POI-OOXML [The Apache Software Foundation]

<https://mvnrepository.com/artifact/org.apache.poi/poi>

<https://mvnrepository.com/artifact/org.apache.poi/poi-ooxml>

XMLPULL [Stafan Haustein, Alexander Slominski]

<https://mvnrepository.com/artifact/xmlpull/xmlpull>

FontAwesomeFX [Dave Gandy, Jens Deters]

<https://bitbucket.org/Jerady/fontawesomefx/>

SAX [sax]

<https://mvnrepository.com/artifact/sax/sax/2.0.1>

maven-model [Apache]

<https://mvnrepository.com/artifact/org.apache.maven/maven-model>

launch4j-maven-plugin [Lukaszenart]

<https://github.com/lukaszlenart/launch4j-maven-plugin>

7. Abschlusserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe. Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken oder Internetquellen als solche kenntlich gemacht habe.

Ort, Datum

Unterschrift des Schülers