# Aufgabe 1: Wörter aufräumen

Team-ID: 00523

Team: Einstein One

Bearbeiter/-innen dieser Aufgabe:
Stefan Cames

30. Oktober 2020

## Inhaltsverzeichnis



### Lösungsidee

Die Grundidee beim „Aufräumen“ der Wörter besteht darin, die vorgebenden Lücken des Lücktextes mit den verfügbaren Wörtern zu füllen, sodass alle Wörter aufgehen und ein Text entsteht. Wichtigste Anhaltspunkte sind dabei die Länge der Lücken und somit die Länge aller verfügbaren Wörter, sowie einzelne Buchstaben, welche im Lückentext bereits ausgefüllt sind. Dieser zweite Filter ist wichtig, da es durchaus vorkommen kann, dass man auch gleichlange Wörter hat und mit Hilfe vorgebender Buchstaben entscheiden kann, welches der verfügbaren Wörter das richtige für die entsprechende Lücke ist. Ist ein und dasselbe Wort mehrmals in der Liste der verfügbaren Wörter, so wird einfach ein Wort genommen.

### Umsetzung

Setzt man die dargelegte Idee in Java um, so muss man als ersten Schritt die vorgegebene Textdatei erst einmal einlesen. Mittels eines FileReaders/BufferedReaders ist es möglich, beide Zeilen der Txt-Datei einzulesen, wobei sowohl Lückentext als auch die Wörter als String abgespeichert werden. Im Folgenden werden alle Lücken, sowie die Wörter jeweils in einem Array mit entfernten Leerzeichen gespeichert. Hierbei enthält das Array *wordlist* Word-Objekte, welche die eigentlichen Lücken als String speichern. Beim Erzeugen dieser Objekte werden Satzteichen vorsorglich entfernt und bereits geprüft, ob die Lücke einen vorgegebenen Buchstaben enthält.
In Form einer While-Schleife wird nun das eigentliche Programm ausgeführt, welche so lange läuft, bis alle Wörter in die Lücken eingesetzt wurden. Für jede Lücke, welche noch nicht gefüllt ist, wird nun die Methode *solveWort()* aufgerufen. Die Methode *solveWort()* filtert mitteln einer im weiteren Verlauf erklärten Methode *filterKeys()* alle noch verfügbaren Wörter und gibt nur noch die zurück, welche entsprechend der Länge und einem einzelnen Buchstaben in der Lücke passen könnten.

Ist dies nur noch ein einziges mögliches Wort, so wird es kurzerhand als Lösung für die Lücke benutzt und ist somit für andere Lücken nicht mehr verfügbar, wird also (in der Main-Klasse) aus Liste aller verfügbaren Wörter gelöscht. Die Lücke ist somit gelöst. Stehen mehrere Wörter zur Verfügung, so wird erst einmal mit den anderen Lücken weitergemacht, in der Erwartung andere Lücken schließen zu können und so für eine spätere Wiederholung eine kleinere Auswahl zu haben, bis irgendwann nur noch ein passendes Wort in Frage kommt.

Wirft man nun einen Blick auf die Methode *filterKeys()*, welche dazu dient, alle Keywords auf die jeweilige Lücke zu überprüfen und auszusortieren, so passiert dies mittels einer for-Schleife, welche alle verfügbaren Wörter durchläuft. Ist die Länge der Lücke eine andere als die Länge des aktuell anzuschauenden Wortes, so wird es direkt aussortiert. Stimmt die Länge jedoch, wird als folgender Schritt überprüft, ob bei der Lücke bereits ein Buchstabe vorgeben war. Ist dies der Fall, überprüft das Programm, ob sich im gerade anzuschauenden Wort genau an derselben Stelle der gleiche Buchstabe befindet, wie er auch in der Lücke vorkommt. Ist dies nicht der Fall, wird das Wort aussortiert. In einer finalen Überprüfung wird vom Programm noch festgestellt, ob in der verbleibenden Liste aller Wörter – wenn diese mehr als ein Wort beinhaltet – ein und dasselbe Wort mehrmals vorkommt und – ist dies der Fall – wird nur eins der Wörter behalten. Im Optimalfall gibt diese Methode nun bereits nur noch ein mögliches Lösungswort zurück, andernfalls wird sie eben später noch einmal auf eine kleinere Auswahl an verbleibenden Wörtern angewendet.

## Beispiele

### 1.

_h | oh
__ | je,
_a_ | was
__r | für
___e | eine
__b___ | arbeit!

oh je, was für eine arbeit!

### 2.

_m | Am
___a__ | Anfang
Warnung: Noch nicht eindeutig
____e |
Warnung: Noch nicht eindeutig
__s |
____e____ | Universum
__________ | erschaffen.
D__ | Das
_a____ | machte
_i___ | viele
__u__ | Leute
____ | sehr
___e__ | wütend
_n_ | und
_u___ | wurde
_____ | allenthalben
_l_ | als
__h____ | Schritt
__ | in
Warnung: Noch nicht eindeutig
___ |
_____h_ | falsche
_____ | Richtung
_____e___ | angesehen.
____e | wurde
__s | das
___ | die

Am Anfang wurde das Universum erschaffen. Das machte viele Leute sehr wütend und wurde allenthalben als Schritt in die falsche Richtung angesehen.

Im ersten Beispiel lässt sich der reibungslose Ablauf meines Programms für „normale“ Lückentexte feststellen, in welchen kein Wort mehrfach vorkommt. Im zweiten Beispiel ist das nicht ganz so eindeutig, hier wird vorerst weitergemacht und nach dem Ausschlussverfahren gearbeitet, sodass auch hier am Ende ein eigenständiger Satz rauskommt. Bei dem doppelten Wort „wurde“ wird einfach eins der beiden benutzt – somit funktioniert auch dieser Check. Satzzeichen werden zu Beginn entfernt und am Ende wieder an das entsprechende Wort drangehangen, mit diesen kann das Programm also auch umgehen und wird nicht gestört. Ein wichtiges Detail wäre hier allerdings, dass jede Lücke maximal einen Buchstaben gegeben haben darf, andernfalls müsste man die Lücke Zeichen für Zeichen durchlaufen und alle Buchstaben herausfiltern und vergleichen.

## Quellcode

Der Quellcode befindet sich im Anhang (Main.java / Word.java).