

# Programmieren II (Java)

## 3. Praktikum: Arrays und Strings

Sommersemester 2022

Christopher Auer, Tobias Lehner



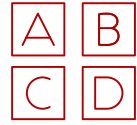
### Abgabetermine

### Lernziele

- ▶ Arrays: Erstellung, Zugriff und Literale
- ▶ [↗](#) Strings: arbeiten mit [↗](#) Strings und [↗](#) StringBuilder

### Hinweise

- ▶ Sie dürfen die Aufgaben *alleine* oder zu *zweit* bearbeiten und abgeben
- ▶ Sie müssen *4 der 5* Praktika bestehen
- ▶ *Kommentieren* Sie Ihren Code
  - ▶ Jede *Methode* (wenn nicht vorgegeben)
  - ▶ *Wichtige* Anweisungen/Code-Blöcke
  - ▶ *Nicht kommentierter* Code führt zu *Nichtbestehen*
- ▶ Bestehen Sie eine Abgabe *nicht* haben Sie einen *zweiten Versuch*, in dem Sie Ihre Abgabe *verbessern müssen*
- ▶ *Wichtig*
  - ▶ Sie sind einer *Praktikumsgruppe* zugewiesen, *nur* in dieser werden Ihre Abgaben *akzeptiert*
  - ▶ Beachten Sie dazu die Hinweise auf der [↗](#) Moodle-Kursseite



## Aufgabe 1: Wordle (Teil 2) ★★

Rufen Sie sich Aufgabe aus dem **1. Praktikum** in Erinnerung, in der wir eine Konsolenversion des Spiels Wordle implementiert haben. In dieser ersten Version haben wir das Lösungswort im Quellcode *fest angegeben*.

In dieser Aufgabe bestimmen wir das Lösungswort zufällig aus einer Menge von möglichen Wörtern, die wir aus einer Datei einlesen. Betrachten Sie sich die Datei `words.txt`. Diese stammt, in modifizierter Form, von der Webseite zum Buch *Word Frequencies in Written and Spoken English* von Geoffrey Leech, Paul Rayson, Andrew Wilson (Longman, London). Die Datei enthält häufige Worte der englischen Sprache (erste Spalte) mit Worttyp (zweite Spalte) und Häufigkeit in einem untersuchten Corpus an Texten (normalisiert 1 bis 100). Die einzelnen Spalten sind durch ein *Tabulatorzeichen* (`\t`) getrennt. Wir werden aus dieser Datei das Menge der Worte erstellen, aus dem Wordle sein Lösungswort wählt.

Gehen Sie bei Ihrer Lösung von dem bereitgestellten Java-Programm `Wordle.java` aus. Dieses beinhaltet eine leicht modifizierte Lösung zur Aufgabe aus dem 1. Praktikum. Für Sie wichtig ist die Methode:

```
String[] getWords(int minFreq)
```

die Sie im Folgenden erweitern müssen. Die Methode liest zunächst *kompletten Inhalt der Datei* `words.txt` in einen `String`. Aus diesem `String` soll `getWords` einen `String-Array` erzeugen und zurückgeben, der die Menge der möglichen Worte enthält.

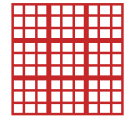
Erweitern Sie `getWords` wie folgt:

- ✓ Trennen Sie den `String words` an den Zeilenumbrüchen (`\n`) in einen `String-Array` auf, um die einzelnen Einträge bearbeiten zu können. Nutzen Sie dazu `String.split`.
- ✓ Wir sammeln die einzelnen Wörter in einem `StringBuilder` auf. Erstellen Sie eine entsprechende *lokale Variable*.
- ✓ Bearbeiten Sie in einer Schleife jede einzelne Zeile wie folgt:
  - ▶ Trennen Sie die Zeile am Tabulatorzeichen (`\t`) in ihre Einträge auf. Sie können dazu wieder `String.split` verwenden
  - ▶ Wir wollen nur *Hauptwörter* betrachten. Prüfen Sie ob in der *zweiten Spalte* "NoC" steht; wenn nicht, *verwerfen Sie den Eintrag*.
  - ▶ Manche Wörter haben ungültige Zeichen (\*, %). Prüfen Sie jedes einzelne Zeichen des Wortes darauf, ob es ein *Buchstabe* ist; *verwerfen Sie den Eintrag*, sollte das nicht der Fall sein. Sie können zur Überprüfung `Character.isLetter` verwenden.
  - ▶ `getWords` bekommt als Parameter `int minFreq`. Verwerfen Sie alle Einträge, deren Häufigkeit *echt kleiner als* `minFreq` sind. Die Häufigkeit finden Sie in der letzten Spalte. Zur Erinnerung: Verwenden Sie `Integer.parseInt` um aus einem `String` einen `Integer` zu parsen.
  - ▶ Das offizielle Wordle verwendet keine Worte in Mehrzahlform. Dies allgemein zu ermitteln ist schwierig. Wir verwenden daher eine einfache *Heuristik*: Verwerfen Sie alle Worte, die mit einem 's' enden.<sup>1</sup>
  - ▶ Zu guter Letzt müssen alle Worte verworfen werden, die *nicht genau aus fünf Zeichen* bestehen.


<sup>1</sup>Die Heuristik *akzeptiert* Worte wie „women“ und *verwirft* Worte wie „chaos“.

- ▶ Hängen Sie das ermittelte Wort an den `StringBuilder` an. Dabei sollen die einzelnen Worte durch einen *Doppelpunkt* getrennt im `StringBuilder` stehen.
- ✓ Nach Beendigung der Schleife, wandeln Sie den `StringBuilder` mit `toString` in einen `String` um und trennen sie den `String` an den *Doppelpunkten* auf um den gewünschten `String-Array` zu erzeugen und zurückzugeben.
- ✓ Testen Sie Ihr Programm in dem Sie ein paar Runden Wordle spielen!
- ▶ *Hinweise*
  - ▶ Beachten Sie, dass die Worte in `words.txt` *kleingeschrieben* sind, während der Vergleich in der Methode `playWordle` in *Großbuchstaben* erfolgt. Wandeln Sie die Worte nach dem Einlesen also in *Großbuchstaben* um.
  - ▶ Legen Sie die Datei `words.txt` in das Verzeichnis, in dem Ihr Java-Programm ausgeführt wird. In Eclipse ist dies das Projektverzeichnis. Sollte das Programm mit einer Fehlermeldung (`FileNotFoundException`) abbrechen, geben Sie in der Methode `readAllLinesFromDict` den *absoluten Pfad zur Datei an*, z.B.:

```
C:\\Users\\auer\\...\\...\\words.txt  
/Users/auer/.../.../words.txt
```



## Aufgabe 2: Sudoku ☆ bis ☆

In dieser Aufgabe beschäftigen wir uns mit  Sudoku-Rätsel. Zur Erinnerung: Ein Sudoku-Rätsel muss so befüllt werden, dass in

- ▶ jeder *Zeile*
- ▶ jeder *Spalte*
- ▶ und jedem der 9 *Unterquadrate*

jede Ziffer von *1 bis 9* jeweils *genau einmal* vorkommt. Hier die Ausgangssituation für ein Sudoku-Rätsel (links) und dessen Lösung (rechts)<sup>2</sup>

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Wir modellieren ein Sudoku-Rätsel als einen *zweidimensionalen int-Array* der Größe  $9 \times 9$ . Der erste Index gibt dabei die Zeile, der zweite Index die Spalte an. Zum Beispiel hat der markierte Punkt die Array-Indizes `[4][7]`:

	0	1	2	3	4	5	6	7	8
0									
1									
2									
3									
4								•	
5									
6									
7									
8									

Ein leerer Eintrag wird durch den `int`-Wert 0 dargestellt.

### Array-Literal und Ausgabe

- ✓ Deklarieren Sie eine Klasse `Sudoku` mit einer *öffentlichen statischen Methode* `getExample()`, die obiges Sudoku-Rätsel (linkes Bild oben) als *Array-Literal* (nochmals: **Array-Literal!**<sup>3</sup>) deklariert und zurückgibt.

<sup>2</sup>Quelle: <https://github.com/aweinstein/sudoku2tikz>

<sup>3</sup>Es ist ein **Array-Literal** verlangt! Wenn Sie nicht wissen, was ein Array-Literal ist, konsultieren Sie die Vorlesungsfolien!

- ✓ Erstellen Sie eine *öffentliche statische Methode* `printSudoku` die die aktuelle Belegung eines Sudoku-Rätsels ausgibt. Die Ausgabe soll dabei den Darstellungen oben ähneln und die Zahlen und Spalten mit den Indizes beschriften. Verwenden Sie für die Ausgabe *Schleifen*!
- Beispiel:

```

      0 1 2   3 4 5   6 7 8
+-----+-----+-----+
0 | 5 3   |   7   |       |
1 | 6     | 1 9 5 |       |
2 |   9 8 |       |   6   |
+-----+-----+-----+
3 | 8     |   6   |       3 |
4 | 4     |   8   3 |       1 |
5 | 7     |   2   |       6 |
+-----+-----+-----+
6 |   6   |       |   2 8   |
7 |       | 4 1 9 |       5 |
8 |       |   8   |   7 9   |
+-----+-----+-----+

```

- ✓ Erstellen Sie eine `main`-Methode, in der Sie das durch `getExample` erzeugte Sudoku-Beispiel ausgeben.

### Prüfen der Einträge

- ✓ Erstellen Sie eine *öffentliche statische Methode* `checkValidSudokuMatrix(int [][] sudoku)`, die folgende Eigenschaften des Parameters `sudoku` prüft:
- ▶ `sudoku` darf *nicht null* sein
  - ▶ Die Anzahl der Zeilen muss **9** sein
  - ▶ Keine Zeile darf *null* sein
  - ▶ Die Anzahl der Spalten in jeder Zeile muss **9** sein
  - ▶ Alle Einträge in `sudoku` müssen einen Wert  $\geq 0$  und  $\leq 9$  haben
- Sollte eine Eigenschaft verletzt sein, erzeugen Sie eine `IllegalArgumentException` mit *aus-sagekräftiger und für den Fehlerfall spezifischen Fehlermeldung*!
- ✓ Erstellen Sie eine *öffentliche statische Methode* `isValidEntry` mit folgender Signatur:

```
boolean isValidEntry(int[][] sudoku, int row, int column, int entry)
```

Die Methode prüft, ob an die Stelle in Zeile `row` und Spalte `column` der Eintrag `entry` gesetzt werden kann. Gehen Sie dabei wie folgt vor:

- ▶ Rufen Sie `checkValidSudokuMatrix` auf `sudoku` auf um dessen Gültigkeit zu prüfen.
- ▶ Prüfen Sie dann die Gültigkeit von `row`, `column` und `entry`.
- ▶ Prüfen Sie, ob in der *Zeile* `row`, der *Spalte* `column` oder in dem entsprechenden Unterquadrat der Eintrag `entry` bereits vorhanden ist. *Wichtig*: Sie müssen dabei die *Zelle selbst* (an Stelle `[row][column]`) bei allen drei Tests *ignorieren*.
- ▶ Geben Sie `false` zurück, wenn `entry` bereits vorhanden ist; sonst `true`.

*Hinweise:*

- ▶ Versuchen Sie nicht *alles auf einmal zu lösen*! Fangen Sie mit der Prüfung der Zeile an und fahren Sie erst fort, wenn diese Prüfung korrekt ist. Testen Sie Ihre Implementierung mit *eigenen Beispielen über die main-Methode* und den beigefügten *JUnit-Tests*.
- ▶ Bei der Überprüfung der *Unterquadrate*, überlegen Sie sich zunächst anhand der obigen Bilder, wie Sie aus `row` und `column` die Indizes der Einträge des Unterquadrats ermitteln können. Die *Integer-Division* kann dabei hilfreich sein.

- ✓ Implementieren Sie eine *öffentliche statische Methode* `boolean isSolution(int[][] sudoku)`, die `true` liefert, wenn es sich bei sudoku um eine *vollständige Lösung* eines Sudoku-Rätsels handelt. D.h. alle Einträge müssen belegt sein ( $\neq 0$ ) und den oben beschriebenen Bedingungen genügen. Verwenden Sie für `isSolution` die Methoden `checkValidSudokuMatrix` und `isValidEntry`.
- Hinweis: Im beigefügten JUnit-Test in der Methode `testIsSolution` finden Sie einen Code-Ausschnitt, der ein gelöstes Sudoku erzeugt. Sie können diesen Code-Ausschnitt kopieren und abändern um Ihre Implementierung zu testen.

### Sudoku spielen (optional)

*Erweitern* Sie Ihre `main`-Methode wie folgt: Dem Nutzer soll es möglich sein, dass er durch *Eingaben auf der Konsole* das Sudoku löst. Gestartet wird mit der Rückgabe von `getExample`. Der Nutzer bekommt die aktuelle Sudoku-Belegung als *Konsolenausgabe*. Danach kann er über die Eingabe des *Zeilenindex, Spaltenindex und eines Wertes* einen Eintrag setzen. Das Programm prüft dabei:

- ▶ ob die Eingaben *gültig* sind
- ▶ bereits ein Eintrag an der Stelle *vorhanden* ist
- ▶ der Eintrag nach den *Regeln von Sudoku* an der Stelle überhaupt gesetzt werden kann

Sollte eine Bedingung *nicht erfüllt* sein, wird eine Fehlermeldung ausgegeben. Ansonsten wird der Eintrag gesetzt und der Nutzer kann nach der Ausgabe der modifizierten Belegung eine neue Eingabe tätigen.

Das Programm *beendet*, wenn das Sudoku *vollständig* ist.