Aufgabe 2: Texthopsen

Team-ID: 00028
Name: Dennis Bauer
6. September 2024
Programmiersprache: Java (Version 22)

Inhaltsverzeichnis

Problembeschreibung

Bella und Amira haben das Spiel „Texthopsen“ erfunden. Dabei starten sie von einer festgelegten Position und hüpfen durch den Text. Die Anzahl der Hopser hängt vom dem Buchstaben ab, auf dem sie grade stehen: a = 1, b = 2, c = 3 usw. Nachdem sie gehopst sind, setzen sie von ihrer neuen Position aus das Hopsen fort, basierend auf dem Buchstaben, auf dem sie nun stehen. Dies wiederholt sich, bis sie den Text verlassen haben. Am Ende soll das Programm, das dieses Spiel simuliert, den Gewinner ausgeben.

Lösungsidee

Das erste Problem, das mir aufgefallen ist, besteht darin, dass Leerzeichen und alle anderen Zeichen, die keine Buchstaben sind, übersprungen werden. Um dieses Problem nun Lösen zu können, muss ich eine Methode finden, die aus dem Text alle nicht alphabetischen Zeichen löscht, sodass nur eine Folge von Buchstaben übrig bleibt, durch die man „durchhüpfen“ kann. Dazu werde ich die String Methode *.replaceAll()* nutzen, da ich dort ein *Regex* (einen regulären Ausdruck) angeben kann, um alle nicht alphabetischen Zeichen durch nichts zu ersetzten. Nun habe ich eine Zeichenfolge, mit der ich arbeiten kann.

Dabei stoße ich jedoch auf das nächste Problem: Wie wandle ich einen Buchstaben in eine Zahl um? Mein erster Gedanke war die ASCII-Tabelle (siehe Seite 8). Mithilfe dieser Tabelle kann ich jeden Buchstaben in seine zugehörige Zahl umwandeln. Da Groß- und Kleinbuchstaben in der ASCII-Tabelle unterschiedliche Zahlenwerte haben, muss ich zunächst die gesamte Zeichenfolge in Kleinbuchstaben umwandeln. Um beispielsweisen nun ein „a“ in eine 1 oder ein „b“ in eine 2 zu verwandeln, muss ich den ASCII-Wert des jeweiligen Buchstabens ermitteln und diesen Wert so anpassen, dass die gewünschte Zahl entsteht. Da die Buchstaben „a“ bis „z“ in der ASCII-Tabelle fortlaufend angeordnet sind, kann ich von jedem ASCII-Wert einfach denselben Wert subtrahieren. Das kleine „a“ hat den ASCII-Wert 97, daher muss ich von jedem Buchstabenwert 96 subtrahieren, um die gewünschte Zahl zu erhalten.

Das nächste Problem werden die Sonderbuchstaben (ä, ö, ü, ß) sein. Da diese Buchstaben nicht fortlaufend nach dem „z“ kommen, muss es für sie eine separate Behandlung geben. Dies ist jedoch nicht allzu kompliziert, da ich einfach von jedem dieser Buchstaben eine bestimmte, festgelegte Zahl subtrahieren muss, um die gewünschte Zahl zu erhalten.

Jetzt habe ich auf einfache Weise jedem Buchstaben eine Zahl zugeordnet. Der nächste Schritt besteht darin, programmatisch durch den Text zu „hopsen“. Dazu verwende ich eine einfache Schleife, die so lange durchläuft, wie der Schleifenindex i kleiner als die Länge des Textes ist. In jedem Durchgang der Schleife addiere ich die Zahl, die dem aktuellen Buchstaben an Position i entspricht zum Schleifenindex i und starte die Schleife neu. Bei jedem Schleifendurchgang zähle ich eine Variable hoch. Sobald die Schleife beendet ist, habe ich die Anzahl der benötigten Züge ermittelt, um durch den Text zu hüpfen. Die Person mit den wenigsten Durchgängen hat gewonnen.

So baut sich meine Lösungsidee auf. Der Grundgedanke ist im Wesentlichen im letzten Abschnitt zusammengefasst. Sollte es zu einem Unentschieden kommen, gewinnt immer die Person, die begonnen hat. Da dies stets Bella ist, gewinnt Bella im Falle eines Unentschiedens immer.

Umsetzung

Ich programmiere ein konsolenbasiertes Java-Programm. Da ich nur mit der Konsole und keinen anderen grafischen Funktionen arbeiten kann, habe ich mich entschieden, eine selbst programmierte Klasse in mein Projekt zu importieren, mit der ich die Farbe und den Stil des Textes ändern kann. Mein erster Schritt war es, die Klasse in mein Programm zu importieren.

Der zweite Schritt besteht darin, dass das Programm den Text, mit dem gearbeitet werden soll, speichern muss. Dieser Text soll in einem erstellten *String* namens *Text* gespeichert werden. Um den Text in das Programm zu importieren, habe ich mich dazu entschieden, Dateien zu verwenden, die das Programm einliest. Um eine solche Datei auszulesen, habe ich eine Funktion geschrieben, die dies übernimmt, den Text richtig formatiert und am Ende in einem einzelnen String zurückgibt, *getTextInput(String fileName)*. Diese Funktion besitzt jedoch einen Parameter, in dem der Dateiname übergeben werden muss, unter dem der Text gespeichert ist. Da der Benutzer diesen eingeben muss, wird als Parameter eine Funktion namens *getFileName()* verwendet, die den Dateinamen vom Benutzer abfragt und zurückgibt. Diese Funktion funktioniert wie folgt:

Zuerst wird ein *Scanner*-Objekt erstellt, das die Benutzereingabe aus der Konsole einliest. Anschließend startet eine Schleife, die den Benutzer mithilfe dieses *Scanner*-Objekts fragt, ob er einen Beispieltext verwenden möchte oder einen eigenen. Die Abfrage erfolgt, indem der Benutzer entweder eine Eins oder eine Zwei in die Konsole eingibt. Die Antwort wird dann in einer Variablen namens *answer* gespeichert. Der Methodenaufruf vom *Scanner*-Object, der die Eingabe aus der Konsole liest und zurückgibt, befindet sich in einem *Try-Catch-*Block, um fehlerhafte Eingaben zu verhindern. Falls die Eingabe ungültig ist oder die Zahl nicht Eins oder Zwei entspricht, wird der Benutzer darauf hingewiesen und die Schleife beginnt von Neuem. Dies geschieht, indem die *do-while-Schleife* überprüft, ob die Variable *answer* weder Eins noch Zwei ist. Solange *answer* also nicht Eins oder Zwei entspricht, wird die Schleife wiederholt und der Benutzer erneut gefragt (siehe **Eigenes-Beispiel-2**). Nun wird eine Variable erstellt die den Dateinamen speichert (*fileName*). Dieser Namen wird entweder durch die Abfrage, welches Beispiel verwendet werden soll, oder indem der Name der Datei mit dem eigenen Text erfragt wird, erstellt. Dies basiert auf der Antwort des Benutzers. Die Auswahl des Beispiels erfolgt ähnlich wie bei der ersten Abfrage: Der Benutzer wird aufgefordert, eine Zahl im Bereich von 1 bis 5 einzugeben. Diese Abfrage erfolgt erneut in einem *try-*

catch-Block, um fehlerhafte Eingaben abzufangen. Dabei wird überprüft, ob der eingegebene Wert im gültigen Bereich (1-5) liegt und ob die Eingabe eine Zahl ist. Ist die Eingabe ungültig, wird der Benutzer darauf hingewiesen und die Abfrage wird durch eine *do-while-Schleife* wiederholt (siehe **Eigenes-Beispiel-3**). Die Schleife läuft so lange, bis eine gültige Zahl eingegeben wird. Wenn die Eingabe gültig ist, wird der Dateiname erstellt. Da die Beispiele alle den Namen „HopsenX“ tragen, wobei X für die ausgewählte Beispielnummer steht, wird der Dateiname wie folgt aufgebaut: Der Text „Hopsen“ wird um die vom Benutzer eingegebene Zahl ergänzt. Dieser entstandene Dateiname wird dann zurück gegeben. Wenn der Benutzer sich entschieden hat, eine eigene Datei zu verwenden, wird er aufgefordert, den Dateinamen einzugeben. Da prinzipiell jede Zeichenfolge als Dateiname zulässig ist, ist hier keine wiederholende Schleife oder ein *try-catch*-Block erforderlich, da keine fehlerhaften Eingaben zu erwarten sind. Bevor jedoch der eingegebene Dateiname verwendet wird, wird überprüft, ob der Benutzer das Dateiformat angegeben hat. Falls ja, wird dieses Format mit Hilfe der Methode *.subString()* aus dem Dateinamen entfernt, da es automatisch in der Funktion, die für das Auslesen der Datei zuständig ist, hinzugefügt wird.

Die Funktion *getTextInput(String fileName)*, die den Dateinamen als Parameter erhält, funktioniert nun wie folgt:

Zuerst wird der *String* erstellt, in dem der Text später gespeichert und zurückgegeben werden soll. Als nächstes wird ein *Scanner*-Objekt namens *fileInput* erstellt, das eine Datei ausliest, die ihm als Parameter übergeben wird. Die Datei wird aus dem Ressourcen-Ordner des Projekts anhand der übergebenen Variablen *fileName* importiert. Falls diese Datei nicht existiert, wird der Fehler in einem *catch*-Block abgefangen, und der Benutzer wird darüber informiert, dass die Datei nicht vorhanden ist (siehe **Eigenes-Beispiel-4**). Nachdem der Benutzer benachrichtigt wurde, wird das Programm mit *System.exit(1)* beendet. Wird die Datei jedoch gefunden, wird das Programm fortgesetzt. Da die Datei zeilenweise ausgelesen wird, beginnt eine *While*-Schleife, die überprüft, ob eine weitere Zeile existiert. Falls dies der Fall ist, wird ein *String* erstellt, der den Inhalt dieser Zeile speichert. Anschließend wird überprüft, ob dieser *String* Text enthält, da eine Zeile auch leer sein kann. Ist der *String* nicht leer, wird zunächst jeder Buchstabe im *String* in Kleinbuchstaben umgewandelt. Dies geschieht mit der *String*-Methode *.toLowerCase()*. Danach werden alle Zeichen, die nicht a-z, ä, ö, ü oder ß sind, durch nichts ersetzt. Dafür wird die *.replaceAll()*-Methode

verwendet. Diese Methode erhält als ersten Parameter einen *Regex* (einen regulären Ausdruck), der die zu ersetzenden Zeichen angibt und als zweiten Parameter das Zeichen durch das ersetzt werden soll. Der *Regex* sieht wie folgt aus: *[^a-zäöüß]*. Das ^-Zeichen gibt an, dass alle Zeichen gemeint sind, die nicht in der folgenden Liste enthalten sind. *a-z* bedeutet, dass alle Kleinbuchstaben von a bis z nicht ersetzt werden sollen. Die Zeichen *ä, ö, ü* und *ß* sind explizit angegeben, da sie nicht in das a-z-Spektrum fallen. Alle Zeichen, die nicht aufgezählt wurden, werden nun durch ein leeres Zeichen ("") ersetzt. Nachdem alle unerwünschten Zeichen aus der Zeile entfernt wurden, wird diese an die Variable *text* angehängt. Dieser Vorgang wiederholt sich fürjede Zeile in der Datei.

Beschrieben Schleife im Code umgesetzt

```
// Liest die verschiedenen Zeilen aus der Datei und fügt sie an den String text an.
while (fileInput.hasNext()) {
    String line = fileInput.nextLine();
    if (!line.isEmpty()) {
        // Wandelt alle Buchstaben in der Zeile in Kleinbuchstaben um.
        line = line.toLowerCase();
        // Entfernt alle Zeichen, die nicht im Unicode-Bereich liegen.
        line = line.replaceAll( regex: "[^a-zäöüß]",  replacement: "");
        text.append(line);
    }
}
```

Sobald die Schleife abgeschlossen ist, enthält der *text* den gesamten Text, bestehend aus Kleinbuchstaben sowie den Zeichen ä, ö, ü und ß und ohne andere Zeichen. Dieser *String* wird dann zurückgegeben und ist nun in einer Variablen gespeichert mit dem gleichen Namen, *text*.

Als nächster Schritt speichert das Programm zwei Variablen, die die Anzahl der Schritte für je Bella und Amira festhalten, die sie benötigt haben, um das Spiel durchzulaufen. Diese Anzahl wird in der Funktion *runThroughGame(String text, int startPosition)* berechnet. Die Funktion arbeitet wie folgt:

Zunächst werden zwei *Integer*-Variablen erstellt. Die erste Variable, *pos*, speichert die aktuelle Position im Text des Spielers (Programmes) und wird zu Beginn auf die Startposition des Spielers gesetzt, die als Parameter übergeben wurde (*startPosition*). Die zweite Variable, *runs*, speichert die Anzahl der „Sprünge“ (Durchläufe des Programms), indem sie zählt, wie oft die Schleife,

die einen Sprung simuliert, durchläuft. Als nächstes beginnt die *while*-Schleife, die so lange läuft, wie *pos* kleiner oder gleich der Länge des Textes ist. In der Schleife wird zunächst die Variable *runs* um eins erhöht, da nun ein Zug erfolgt. Anschließend wird der Buchstabe, auf dem der Spieler gerade steht, in eine Zahl umgewandelt. Dies geschieht, indem der *Charakter* (*char*) im Text an der Position des Spielers mit der Methode *.charAt()* ermittelt wird. Dieser Charakter wird dann in einen *int* umgewandelt, indem er einfach in einen Integer *gecastet* wird. Die resultierende Zahl entspricht dem ASCII-Wert des Buchstabens. Um den gewünschten Wert zu erhalten, wird 96 von dieser Zahl subtrahiert, da „a“ den ASCII-Wert 97 hat und die Buchstaben bis „z“ fortlaufend nummeriert sind. So wird ein „a“ zu einer Eins, ein „b“ zu einer Zwei und so weiter. Diese Zahl wird dann in der Variablen *letterNumber* gespeichert. Als nächstes wird überprüft, ob die Zahl in *letterNumber* größer oder kleiner als 26 ist. Dies ist der Fall, wenn der Spieler (das Programm) auf einem Buchstaben steht, der nicht von a bis z reicht. Ist dies der Fall, wird die Zahl in *letterNumber* mit einem *Switch-Case*-Block auf verschiedene Möglichkeiten überprüft. Insgesamt gibt es vier mögliche Zahlen, die in *letterNumber* stehen können: 132, 150, 156 und 127.

Die Zahl 132 ergibt sich, da der ASCII-Wert für „ä“ 228 beträgt und 228 minus 96 gleich 132 ist. Diese Logik wird für alle vier möglichen Buchstaben angewendet. Die entsprechenden Möglichkeiten sind:

- 132 für ä,
- 150 für ö,
- 156 für ü,
- 127 für ß.

Wenn eine dieser Zahlen zutrifft, wird *letterNumber* auf die zugehörige Zahl des Buchstabens gesetzt. Beispielsweise wird *letterNumber* auf 27 gesetzt, wenn es 132 (ä) ist, auf 28 gesetzt, wenn es 150 (ö) ist, und so weiter. Es sollten keine anderen Möglichkeiten vorkommen, da alle anderen Zeichen ersetzt wurden. Falls dennoch eine andere Zahl erkannt wird, wird das Programm beendet. Als letztes wird dann noch die Postion mit *letterNumber* addiert. Der Spieler „hüpft“ also die entsprechende Länge, die in *letterNumber* gespeichert ist.

Die grade beschrieben Schleife, die das spiel simuliert

```
// Die Schleife läuft, bis die Position über das Textende hinausgeht.
while (pos + 1 <= text.length()) {
    // Jeder Schleifendurchlauf entspricht einem Spielzug, daher wird die Variable erhöht.
    runs++;

    // Wandelt den Buchstaben an der aktuellen Position im Text in eine Zahl um (ASCII).
    int letterNumber = ((int) text.charAt(pos)) - 96;

    // Wandelt Zeichen, die keine Buchstaben des lateinischen Alphabets sind, in die entsprechende Zahl um.
    if (letterNumber > 26 || letterNumber < 1) {
        switch (letterNumber) {
            case 132 -> letterNumber = 27; //ä
            case 150 -> letterNumber = 28; //ö
            case 156 -> letterNumber = 29; //ü
            case 127 -> letterNumber = 30; //ß
            default -> System.exit( status: 1); //Alles andere (Nicht möglich)
        }
    }

    // Erhöht die Position des Programmes um die Anzahl an "Sprünge" die der Buchstabe wieder spiegelt auf
    // dem das Programm sich grade befindet
    pos += letterNumber;
}
```

Die Variablen *bella* und *amira* haben nun die Anzahl an Zügen gespeichert, die jede Spielerin benötigt hat. Diese Werte werden von dem Rückgabewert der Funktion bestimmt. Der Parameter für die Startposition ist dabei für Amira immer 0 und für Bella immer 1. Schließlich wird ausgegeben, wer weniger Züge benötigt hat. Dies erfolgt durch eine einfache *If*-Abfrage. Falls beide Spielerinnen die gleiche Anzahl an Zügen gebraucht haben, gewinnt immer Bella, da sie als erste begonnen hat.

Beispiele

Beispiel-1

```
-Jugendwettbewerb-2024-42-Runde-3- Junioraufgabe-2 Dennis Bauer

Möchten sie eine Beispieldatei nutzen (1) oder eine eigene Datei (2)?: 1
Bitte geben sie an, mit welchem Beispiel gespielt werden soll (1-5): 1

---Ergebnisse---------------------------
Bella hat 68 Runden gebraucht.
Amira hat 71 Runden gebraucht.
So hat Bella mit 3 Runden weniger gewonnen!
-----------------------------------------
```

Beispiel-2

```
-Jugendwettbewerb-2024-42-Runde-3- Junioraufgabe-2 Dennis Bauer

Möchten sie eine Beispieldatei nutzen (1) oder eine eigene Datei (2)?: 1
Bitte geben sie an, mit welchem Beispiel gespielt werden soll (1-5): 2

---Ergebnisse---------------------------
Bella hat 25 Runden gebraucht.
Amira hat 25 Runden gebraucht.
Bella und Amira brauchten gleich lange, aber da Bella  begonnen hat, hat sie gewonnen!
-----------------------------------------
```

Beispiel-3

```
-Jugendwettbewerb-2024-42-Runde-3- Junioraufgabe-2 Dennis Bauer

Möchten sie eine Beispieldatei nutzen (1) oder eine eigene Datei (2)?: 1
Bitte geben sie an, mit welchem Beispiel gespielt werden soll (1-5): 3

---Ergebnisse----------------------------
Bella hat 18 Runden gebraucht.
Amira hat 18 Runden gebraucht.
Bella und Amira brauchten gleich lange, aber da Bella  begonnen hat, hat sie gewonnen!
--------------------------------------
```

Beispiel-4

```
-Jugendwettbewerb-2024-42-Runde-3- Junioraufgabe-2 Dennis Bauer

Möchten sie eine Beispieldatei nutzen (1) oder eine eigene Datei (2)?: 1
Bitte geben sie an, mit welchem Beispiel gespielt werden soll (1-5): 4

---Ergebnisse----------------------------
Bella hat 35 Runden gebraucht.
Amira hat 32 Runden gebraucht.
So hat Amira mit 3 Runden weniger gewonnen!
--------------------------------------
```

Beispiel-5

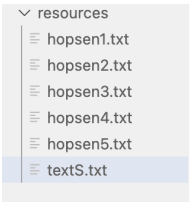
```
-Jugendwettbewerb-2024-42-Runde-3- Junioraufgabe-2 Dennis Bauer

Möchten sie eine Beispieldatei nutzen (1) oder eine eigene Datei (2)?: 1
Bitte geben sie an, mit welchem Beispiel gespielt werden soll (1-5): 5

---Ergebnisse----------------------------
Bella hat 923 Runden gebraucht.
Amira hat 930 Runden gebraucht.
So hat Bella mit 7 Runden weniger gewonnen!
--------------------------------------
```

Eigenes-Beispiel-1 (Text: aabbccddeeffgghhiijjkkllmmnnooppqqrrssttuuvvww)

Dieses Beispiel zeigt, wie das Programm einen eigenen Text einließt




```
-Jugendwettbewerb-2024-42-Runde-3- Junioraufgabe-2 Dennis Bauer

Möchten sie eine Beispieldatei nutzen (1) oder eine eigene Datei (2)?: 2
Bitte geben sie den Namen der Datei an (beachten sie, dass sich die Datei im "resources" Ordner befinden muss): textS

---Ergebnisse----------------------------
Bella hat 9 Runden gebraucht.
Amira hat 8 Runden gebraucht.
So hat Amira mit 1 Runden weniger gewonnen!
--------------------------------------
```

Eigenes-Beispiel-2

Dieses Beispiel zeigt, wie das Programm mit fehlerhaften Eingaben in der ersten Abfrage umgeht.

```
-Jugendwettbewerb-2024-42-Runde-3- Junioraufgabe-2 Dennis Bauer

Möchten sie eine Beispieldatei nutzen (1) oder eine eigene Datei (2)?: 3
Bitte geben sie nur "1" für eine Beispieldatei oder "2" für eine eigene Datei an!
Möchten sie eine Beispieldatei nutzen (1) oder eine eigene Datei (2)?: s
Bitte geben sie nur Zahlen an!
Möchten sie eine Beispieldatei nutzen (1) oder eine eigene Datei (2)?: 0
Bitte geben sie nur "1" für eine Beispieldatei oder "2" für eine eigene Datei an!
Möchten sie eine Beispieldatei nutzen (1) oder eine eigene Datei (2)?: 1
Bitte geben sie an, mit welchem Beispiel gespielt werden soll (1-5): 1

---Ergebnisse----------------------------
Bella hat 68 Runden gebraucht.
Amira hat 71 Runden gebraucht.
So hat Bella mit 3 Runden weniger gewonnen!
--------------------------------------
```

Eigenes-Beispiel-3

Dieses Beispiel zeigt, wie das Programm mit fehlerhaften Eingaben in der zweiten Abfrage umgeht.

```
-Jugendwettbewerb-2024-42-Runde-3- Junioraufgabe-2 Dennis Bauer

Möchten sie eine Beispieldatei nutzen (1) oder eine eigene Datei (2)?: 1
Bitte geben sie an, mit welchem Beispiel gespielt werden soll (1-5): 7
Bitte geben sie nur Werte von 1 bis 5 an!
Bitte geben sie an, mit welchem Beispiel gespielt werden soll (1-5): 0
Bitte geben sie nur Werte von 1 bis 5 an!
Bitte geben sie an, mit welchem Beispiel gespielt werden soll (1-5): df
Bitte geben sie nur Zahlen an!
Bitte geben sie an, mit welchem Beispiel gespielt werden soll (1-5): 2

---Ergebnisse---------------------------
Bella hat 25 Runden gebraucht.
Amira hat 25 Runden gebraucht.
Bella und Amira brauchten gleich lange, aber da Bella  begonnen hat, hat sie gewonner
----------------------------------------
```

Eigenes-Beispiel-4

Dieses Beispiel zeigt, wie das Programm mit Dateinamen um geht, die nicht vorhanden sind.

```
-Jugendwettbewerb-2024-42-Runde-3- Junioraufgabe-2 Dennis Bauer

Möchten sie eine Beispieldatei nutzen (1) oder eine eigene Datei (2)?: 2
Bitte geben sie den Namen der Datei an (beachten sie, dass sich die Datei im "resources" Ordner befinden muss): remscheid
Fehler, Datei mit dem Namen remscheid.txt wurde nicht im normalen resources Ordner gefunden!
```

Die ASCII-Tabele

Ein Computer arbeitet binär, das heißt, er speichert und verarbeitet Informationen nur in Form von Nullen und Einsen. Mithilfe des binären Zahlensystems können aus diesen Nullen und Einsen Dezimalzahlen dargestellt werden. Buchstaben hingegen lassen sich nicht so direkt aus binären Werten ableiten. Dafür wurde die ASCII-Tabelle entwickelt. In dieser Tabelle ist jedem Zeichen eine eindeutige Nummer zugeordnet. So kann ein Computer Zahlen speichern und diese später in die entsprechenden Buchstaben oder Symbole umwandeln.