Technische Universität Braunschweig
Institut für Anwendungssicherheit

Prof. Dr. Martin Johns
Marius Musch
Maximilian von Unwerth

Programmieren I

02. Übungsblatt

Melden Sie sich bitte für die Veranstaltung bei Stud.IP (http://studip.tu-bs.de) an und tragen sich in einer der kleinen Übungen ein. (`https://studip.tu-braunschweig.de/dispatch.php/course/details?sem_id=d08ea3cd66c59967be1218a540bdefa7`)

Verspätete, nicht hochgeladene Abgaben, Abgaben, die per Mail getätigt werden, sowie Plagiate führen zur Bewertung der Aufgabe mit 0 Punkten.

*Insgesamt können Sie für das Lösen der Pflichtaufgabe **15 Punkte** erhalten. Berücksichtigt werden ausschließlich Ergebnisse, die bis zum **29.11.2020 um 23:59** in Ihr Git Repository auf `https://programming.ias.cs.tu-bs.de/` gepusht wurden.*

Ihre Lösung muss in den bereits für dieses Übungsblatt angelegten Ordner (blatt2) im Git Repository gespeichert werden, damit diese gewertet wird.

Nur compilierende Programme gelten als Lösung! Ein Programm das Teillösungen beinhaltet aber nicht compiliert oder aus anderen Gründen nicht ausführbar ist erhält 0 Punkte.

Aufgabe 1: Programmieren Sie ein Programm, welches als Eingabe nun keine Zahl, sondern einen String, also eine Zeichenkette, erhält und jeweils das semantische Gegenteil von einem vordefiniertem Set davon ausgibt. Zum Beispiel:

$$hoch <-> tief$$
$$klein <-> groß$$
$$stark <-> schwach$$

Nutzen Sie dafür die `switch-case`-Anweisung.

Aufgabe 2: Werten Sie – jeweils unter Annahme der Deklaration `int x = 7;` – die folgenden Ausdrücke aus. In welchen Fällen wird der Wert von `x` durch die Auswertung verändert? Überlegen Sie zuerst und überprüfen Sie Ihre Antwort anschließend am Rechner.

a) `x <= 2`

b) `x -= 2`

c) `x % 2`

d) `x / 2`

e) `x & 2`

f) `x << 2`

g) `x | 2`

i) `x >> 2`

k) `x =+ 2`

h) `x ^ 2`

j) `x += 2`

l) `x =~ 2`

Aufgabe 3: Was geben die folgenden Programmstücke aus? Überlegen Sie zuerst und überprüfen Sie Ihre Antwort anschließend am Rechner. (Die Formatierung hält sich in diesem Fall bewusst nicht an die Formatierungsrichtlinien.)

a)

```
1         boolean b = false;
2         if (b = true) {
3             System.out.println(true);
4         } else {
5             System.out.println(false);
6         }
```

listings/Aufgabe22a.java

b)

```
1         int i = 0;
2         if (i < 1)
3             i = i + 1;
4             if (i < 1)
5                 i = i + 1;
6         else
7             i = i - 1;
8         System.out.println(i);
```

listings/Aufgabe22b.java

c)

```
1         int i = 0;
2         if (i > 0)
3             if (i < 1) i = i + 1;
4         else
5             i = i - 1;
6         System.out.println(i);
```

listings/Aufgabe22c.java

Aufgabe 4: Die nachfolgenden Programmfragmente weisen Fehler auf. Finden Sie diese. Begründen Sie Ihre Antwort.

```
1         boolean println, true;
2         char ab c, de, f^g;
3         int a = 0xf7, b = 0xg7, c = 12e3, d = 017, 2e;
4         double s_, t$u, v_$w;
5         System.out.println("a: " , a);
```

listings/Aufgabe12.java

Pflichtaufgabe 5: FizzBuzz - (2 Punkte) Bei dem sogenannten Fizz-Buzz-Spiel zählen die Spieler eine Zahl hoch. Sie starten bei eins und beginnen einfach zu zählen. Allerdings muss jede Zahl, die durch **3** teilbar ist durch ein **Fizz** und jede Zahl, die durch **5** teilbar ist durch ein **Buzz** ersetzt werden. Ist die Zahl sogar durch **3** und durch **5** teilbar, muss diese durch **Fizz Buzz** ersetzt werden.
Eine typische Auflistung der ersten 20 Fizz-Buzz-Zahlen ergibt also:

1,2,Fizz,4,Buzz,Fizz,7,8,Fizz,Buzz,11,Fizz,13,14,Fizz-Buzz,16,17,Fizz,19,Buzz

Lesen Sie die Zahl n bis wohin ihr Programm zählen soll als Kommandozeilenparameter ein. Erzeugen Sie ein Array, dass zuerst eine Liste aller Zahlen von 1 bis n beinhaltet (1 Punkt). Ergänzen Sie dann ihr Programm so, dass an den richtigen Stellen **Fizz** und **Buzz** in das Array eingesetzt werden und geben Sie am Ende ihr Array aus (1 Punkt).

Pflichtaufgabe 6: COVID19 Datenverarbeitung - (3 Punkte) Die Website `https://npgeo-corona-npgeo-de.hub.arcgis.com/` stellt aktuellen vom Robert-Koch-Institut zur Verfügung gestellten Daten zu den COVID-19 Fällen in Deutschland zum Download bereit. In dieser Aufgabe wollen wir uns damit beschäftigen, wie man aus diesen Daten Werte wie zum Beispiel die Anzahl der Genesenen ermitteln kann. Laden Sie dazu zunächst die ebenfalls im StudIP zur Verfügung gestellte *RKI_COVID19.csv* herunter. Speichern Sie die csv-Datei in dem Ordner, indem sich auch die Java-Datei, die Sie für diese Aufgabe erstellen befindet. Diese enthält nur die Personen mit Wohnsitz in Niedersachsen und Meldedatum im November (Stand 16.11), um die Datei klein zu halten. Bitte fügen Sie die *RKI_COVID19.csv* trotzdem Ihrer *.gitignore* hinzu, um unseren Gitea-Server nicht unnötig zu belasten. *CSV* steht für *Comma Seperated Values* und eine solche Datei ist ähnlich wie eine Tabelle aufgebaut. Die erste Zeile enthält die Namen der einzelnen Tabellenspalten. Jeder Name ist mit einem Komma von dem nächsten getrennt. Am Ende der Zeile steht ein Zeilenumbruch der den nächsten Datensatz anzeigt. Jeder Wert in jeder darauffolgenden Zeile ist wieder durch ein Komma von den anderen getrennt. Auf der Website finden Sie ebenfalls eine Dokumentation zu allen Spalten der Tabelle.

Lesen Sie die Datei mit dem folgenden Codeschnipsel ein und speichern Sie die Daten in einem geeignet großen Array. Ermitteln Sie dafür zuerst wie viele Zeilen und wie viele Spalten Sie für Ihr Array benötigen 1 Punkt. Sie können dafür die csv-Datei einlesen und alle Zeilen und Spalten zählen.

```java
File file = new File("RKI_COVID19.csv");
Scanner fileReader = null;
try {
    fileReader = new Scanner(file);
} catch (FileNotFoundException e) {
    e.printStackTrace();
}
```

listings/File.java

Wie auch bei dem Scanner, den Sie bereits kennen, können Sie mit diesem Scanner eine Datei Zeile für Zeile einlesen (anstatt Eingaben aus dem Terminal). Mit Hilfe von `fileReader.hasNextLine()` können Sie überprüfen, ob die Datei noch eine weitere Zeile enthält. Mit `String text = fileReader.nextLine()` lesen Sie eine Zeile der Datei ein und gehen in der Datei eine Zeile weiter. Das heißt: Wenn Sie nun erneut die gleiche Methode aufrufen, erhalten Sie den Text der nächsten Zeile und so weiter.

Hinweis: Mit Hilfe der Methode `.split()` können Sie Strings bei einem bestimmten Zeichen in eine Liste von Strings aufteilen.

Nachdem Sie das Array erstellt haben, speichern Sie alle Daten der CSV-Datei in dem Array 1 Punkt. Danach suchen Sie in der Datei nach der Spalte **Anzahl Genesen**. Addieren Sie alle Werte, die in jeder Zeile zu finden sind zu einem Gesamtwert auf. Haben Sie alles richtig gemacht haben Sie die Anzahl der genesenen Personen in Niedersachsen mit Meldedatum im November ermittelt 1 Punkt.

Pflichtaufgabe 7: Häufigkeitsanalyse - (10 Punkte) Auch in dieser Aufgabe soll es weiter um die Verarbeitung von Daten gehen. Die zum Knacken von vielen Geheimschriften beliebige Häufigkeitsanalyse beruht darauf, dass in jeder Sprache bestimmte Buchstaben deutlich häufiger vorkommen als Andere. Beispielsweise steht in der deutschen Sprache das **e** mit mehr als 17% an erster Stelle. Das Zählen der Buchstaben in einer abgefangenen Nachricht kann gerade bei längeren Texten jedoch sehr aufwändig sein. Um diese essentielle Vorarbeit zum Knacken einer geheimen Nachricht zu verarbeiten, soll in dieser Aufgabe ein Programm geschrieben werden, dass die verschiedenen Zeichen in einem Text zählt und am Ende eine Liste aller Symbole mit der Anzahl ihres Vorkommens ausgibt.

Erstellen Sie in dem Ordner, in dem Sie auch die *.java*-Datei für diese Aufgabe speichern eine *.txt*-Datei und füllen Sie diese mit einem beliebigen Text. Lesen Sie nun die Datei exakt wie in Pflichtaufgabe 6 mit einem **Scanner** ein. Erstellen Sie vor dem Einlesen ein zwei-dimensionales Array, dass in der ersten "Spalte" die verschiedenen Symbole und in der zweiten "Spalte" die Anzahl ihres Vorkommens speichert.

Hinweis: Sie können in **int**-Arrays auch *chars* speichern. `int[] test = {'A'};` initialisiert zum Beispiel das Array mit dem Zeichen **A**. Im Array wird dann der *ASCII*-Wert[1] des Zeichens gespeichert.

Überlegen Sie sich vor dem Einlesen der Datei, wie groß Ihr Array sein muss 1 Punkt. Beginnen Sie dann Ihre Datei Zeichen für Zeichen auszulesen 1 Punkt. Nutzen Sie dafür die Eigenschaft, dass Strings auch nur Zeichen*ketten* sind. Wenn Sie nun mit `.nextLine()` durch Ihre Datei gehen, sollen Sie zuerst überprüfen, ob das nächste betrachte Zeichen bereits in Ihrem Array enthalten ist. Erstellen Sie dafür eine neue Methode `symbolInArray()`,

[1] `https://en.wikipedia.org/wiki/ASCII`

die ihr Array und das Symbol übergeben bekommt, die prüft, ob ein Symbol bereits in Ihrem Array vorkommt oder noch nicht. Wählen Sie einen geeigneten Rückgabetypen und implementieren Sie ihre Funktion 1 Punkt.

Ist das Symbol noch nicht in dem Array, fügen Sie es hinzu 1 Punkt. Ist es bereits enthalten, erhöhen Sie bloß den Counter in Ihrer zweiten "Spalte" 1 Punkt. Implementieren Sie zur Überprüfung an welcher Stelle sich dann der Eintrag dieses Zeichens in Ihrem Array befindet eine Methode `findSymbol()`, die erneut das Array und das Symbol übergeben bekommt und den Index des Array, an der sich dieses Symbol befindet zurückgibt 1 Punkt.

Nachdem sich alle verschiedenen Zeichen in Ihrem Array befinden, programmieren Sie eine Methode `sortArray()`, die alle Zeilen Ihrer "'Daten-Tabelle" nach der Häufigkeit der Zeichen sortiert. 2 Punkte. Ist Ihr Array nun auch noch sortiert, müssen die Daten nur noch ausgegeben werden. Erzeugen Sie dafür eine letzte Methode `printArray()`, die die Daten des Arrays ausgibt 1 Punkt. Lassen Sie sich außerdem nur die Einträge ausgeben, die auch mindestens eine Häufigkeit von 1 haben 1 Punkt.

Nehmen Sie nun als Eingabe einen sehr langen Text, können Sie sehr gut erkenne, welche Buchstaben häufiger und welche deutlich weniger oft vorkommen.