1. Test –Einführung in JAVA - BITEC-01

01.02.2024 – Cammerlander

- Sie haben 120 Minuten Zeit die Aufgaben zu lösen
- Sie können maximal 20 Punkte erreichen
- Es sind zur Prüfung zugelassen:
    - Taschenrechner (wenn erwünscht)
    - Transparente Wasserflasche
    - Geodreieck, Stifte, usw.
    - **am Computer sind alle Unterlagen sowie die Nutzung des Internets erlaubt**.
    - Die Nutzung des Internets umfasst nicht
        - Chatbots
        - Veröffentlichung der Lösungen
        - sonstige Kommunikation mit anderen Usern
    - Die Nutzung von allen anderen Dingen, muss vorher mit mir abgesprochen werden (z.B. Nutzung von Ohropax), ansonsten wird dies als schummeln gewertet.
    - Die Folge des Schummeln ist eine Bewertung mit 0 Punkten.
- Die Abgabe des Programmcodes erfolgt über Moodle (https://bbrz.rehasued.e-bfi-ooe.at)
- Viel Erfolg! :)

Notenschlüssel:

[0-50): N5; [50-62.5%): G4; [62.5-75%): B3; [75-87.5%): G2; [87.5-100%]: S1., (Schulnotensystem)

Name: _____

Bewertung (vom LV-Leiter auszufüllen):

Aufgabe 1.1	__ / 18
Aufgabe 2	__ / 32
Summe:	__ / 50

Aufgabe 1.1 - (18 Punkte)

Programmverständnis:

Erstelle ein Textfile, kopiere den unten gegebenen Code und füge diesen in das neue Textfile ein. Lösen Sie dort die folgenden Aufgaben:

```
System.out.print("Wie hoch soll das Objekt sein? ");
// TODO
Integer hoehe = Integer.parseInt(scanner.nextLine());

// TODO
String[][] basisfeld = new String[hoehe][hoehe];
String[][] basisfeldGespiegelt = new String[hoehe][hoehe];
String[][] basisfeldTransponiert = new String[hoehe][hoehe];
String[][] basisfeldTransponiertGespiegelt = new String[hoehe][hoehe];

// TODO
for (int zeile = 0; zeile < hoehe; zeile++) {
    // TODO
    for (int spalte = 0; spalte < hoehe; spalte++) {
        // TODO
        basisfeld[zeile][spalte] = " ";
    }
}

// TODO
for (int zeile = 0; zeile < hoehe; zeile++) {
    // TODO
    for (int spalte = 0; spalte < hoehe; spalte++) {
        if (spalte <= zeile) {
            // TODO
            basisfeld[zeile][spalte] = "#";
        }
    }
}

// TODO
for (int zeile = 0; zeile < hoehe; zeile++) {
    // TODO
    for (int spalte = 0; spalte < hoehe; spalte++) {
        // TODO
        basisfeldGespiegelt[zeile][spalte] = basisfeld[hoehe-zeile-1][spalte];
    }
}

// TODO
for (int zeile = 0; zeile < hoehe; zeile++) {
    // TODO
    for (int spalte = 0; spalte < hoehe; spalte++) {
        // TODO
        basisfeldTransponiert[zeile][spalte] = basisfeld[spalte][zeile];
    }
}

// TODO
for (int zeile = 0; zeile < hoehe; zeile++) {
    // TODO
    for (int spalte = 0; spalte < hoehe; spalte++) {
        // TODO
        basisfeldTransponiertGespiegelt[zeile][spalte] = basisfeldTransponiert[hoehe-zeile-1][spalte];
    }
}
```

a) Schreibe zu jedem „// TODO:“ entsprechende Kommentare. Diese sollen die Bedeutung des jeweiligen Codes widerspiegeln **sowie** die Art wie programmiert wird. Schreiben Sie also **nicht nur** was direkt im code passiert, sondern versuchen Sie **auch** zu beschreiben was die Idee hinter den geschriebenen Code ist.
b) Gehe auf den Ausdruck „*Integer dimension = Integer.parseInt(scanner.nextLine());*“ ein.
    i) Was ist der Unterschied zu „*Integer dimension = scanner.nextInt();*“?
    ii) Wie muss der Code angepasst werden wenn „*Integer dimension = scanner.nextInt();*“ anstatt „*Integer dimension = Integer.parseInt(scanner.nextLine());*“ verwendet wird, um das gleiche Verhalten im oben gezeigten Code zu erlangen?

Aufgabe 2.1 – (32 Punkte)

Schleifen, **Verzweigungen** und **User Input** und **Arrays** über Konsole:

Das folgende Programm erzeugt Passwörter welche zufällig ergzeugt werden.

Beschreiben Sie in Kommentaren was Ihr Code macht! z.B. Warum Sie eine gewisse Bedingung für eine While Schleife verwendet haben.

(Lesen Sie zuerst die Puntke 1.-3. der Aufgabe durch)

1. **Schlechte Passwort generieren und Userinput** (5 + 5 Punkte)
   a. Hier sollen zufällig Passwörter erzeugt werden welche folgendermaßen aufgebaut sind.
   Es soll zuerstaus **9 Buchstaben** bestehen. Die ersten 3 Buchstaben sind Kleinbuchstaben (z.B. **aoe**). Danach 3 Großbuchstaben (z.B. **OIU**) und danach wieder 3 Kleinbuchstabben (z.B.**oio**). Die Buchstaben sollen zufällig kombiniert werden. Die Wörter sollen aus eiem Pool von **mindestens a bis z** ausgewählt werden.
   b. Fragen Sie am Schluss den User ob er mit diesem generierten Passwort zufrieden ist, wenn ja beendet das Programm, wenn nein, wird neu generiert.
2. **bessere Passwörter generieren** (10 Punkte)
   Zusätzlich soll der User nun die Anzahl der Passwörter bestimmen können. Diese sind folgendermaßen aufgebaut. 3 Kleinbuchstaben, dann 3 Großbruchstaben und dann 3 Kleinbuchstaben. Wenn das Wort Länger als 9 Buchstaben ist, fülle dazwischen mit Zahlen oder Sonderzeichen auf. Eine Zahl wenn ein Kleinbuchstabe folgt und ein Sonderzeichen wenn ein Großbuchstabe folgt. Je nachdem welche Länge der User eingibt soll dieses Muster beibehalten werden.
   z.B.

Länge	9	10	11
Wörter	3 Klein, 3 Groß, 3 Klein	1 Zahl, 3 Klein, 3 Groß, 3 Klein ODER **(zufällig gewälht)** 3 Klein, 3 Groß, 1 Zahl, 3 Klein ODER **(zufällig gewälht)** 3 Klein, 1 Sonder, 3 Groß, 3 Klein	2 Zahl, 3 Klein, 3 Groß, 3 Klein ODER **(zufällig gewälht)** 1 Zahl, 3 Klein, 3 Groß, 1 Zahl, 3 Klein ODER **(zufällig gewälht)** 3 Klein, 3 Groß, 2 Zahl, 3 Klein ODER **(zufällig gewälht)** 3 Klein, 2 Sonder, 3 Groß, 3 Klein ODER **(zufällig gewälht)** 1 Zahl, 3 Klein, 1 Sonder, 3 Groß, 3 Klein ODER **(zufällig gewälht)** 3 Klein, 1 Sonder, 3 Groß, 1 Zahl, 3 Klein

Beschränken Sie sich auf Länge 9-11. **Bonus: für beliebige Länge lösen.**

3. **Gute Passwörter generieren**

   Vernachlässigen Sie nun das Muster und generieren Passwörter in beliebiger Länge, jedoch länger als 12 Zeichen, mit beliebigen Symbolen (Klein, Groß, Sonderzeichen und Zahlen).

4. **Passwort und Programm verbinden** (3 + 4 + 3 Punkte)
   a. Verwenden Sie die Aufgabe 1, 2 oder 3 um ein Passwort zu generieren. Fragen Sie den User danach um einen beliebigen Text welcher eingegeben wird. Dieser beschreibt für welches Programm das Passwort ist. Fragen Sie am Schluss noch ob der User zufrieden ist, ansonsten erlauben Sie den User eine Neue Eingabe eines Programmes (das Passwort wird aber gespeichert und nicht neu eingegeben!).
   b. Um auch eine neues Passwort eingeben zu können, nachdem diese bereits generiert und bestätigt wurde, erlauben Sie dem User folgenden Befehlt „!!BEFEHL – neues Programm“. Dieser Befehlt wird als erstes eingegeben, anstatt einen Text zu schreiben.
   c. Fügen Sie an zufälligen Stellen des in b) erzeugten **Passwortes** menschlich lesbare Wörter ein. Legen Sie dazu ein Array an welches diese enthält (z.B. „warum“). Lassen Sie das Programm zufällig wählen ob dieses menschlich lesbares Wort groß oder kleingeschrieben wird.

Aufgabe 2.2 - (BONUS)

Schleifen, **Verzweigungen** und **User Input** und **Arrays** über Konsole:

Versuchen Sie die Aufgabe 7 und 8 des Probetests. Hier nochmal die gesamte Angabe.

Generiere Muster, welche vom User gewählt werden, sowie die benötigten Parameter der Muster.

Zuerst soll der User gefragt werden welches Muster zu generieren wünscht. Die Eingabe des Users soll der Name des Musters sein (z.B. **Dreieck, Pyramide, Raute**). Weitere Kategorien der Muster sind nach Eingabe des „Hauptmusters“ zu erfolgen
(z.B. **Pyramide** dann **spitze rechts**, usw.). Falls danach weitere Usereingaben nötig sind, sind diese danach abzufragen (z.B. **Pyramide** dann **spitze rechts**, dann **~** und **%**).

1. **Dreieck** (5 Punkte)
   Hier soll der User die Höhe des Dreieicks steuern können. Die Höhe ist hier die **Anzahl der Zeilen.** Lesen Sie dazu den benötigten Userinput ein.

```
//          #
//          ##
//          ###
//          ####
//          #####
//          ######
```

2. **Pyramide - spitze rechts** (5 Punkte)
   Hier soll der User die Höhe der Pyramide steuern können. Die Höhe ist hier die **Anzahl der Spalten.** Lesen Sie dazu den benötigten Userinput ein.

```
//          #
//          ##
//          ###
//          ####
//          #####
//          ######
//          #####
//          ####
//          ###
//          ##
//          #
```

3. **Pyramide - spitze oben** (6 Punkte)
   Hier soll der User die Höhe der Pyramide steuern können. Die Höhe ist hier die **Anzahl der Zeilen.** Lesen Sie dazu den benötigten Userinput ein.

```
//               #
//              ###
//             #####
//            #######
//           #########
//          ###########
```

4. **Pyramide - spitze links** (6 Punkte)
   Hier soll der User die Höhe der Pyramide steuern können. Die Höhe ist hier die **Anzahl der Spalten.** Lesen Sie dazu den benötigten Userinput ein.

5. **Raute** (6 Punkte)
   Hier soll der User die **Länge einer der Diagonalen** der Raute steuern können. Lesen Sie dazu den benötigten Userinput ein.

6. **Muster** (6 Punkte)
   Hier soll der User die **Länge einer der Diagonalen** der Raute steuern können. Lesen Sie dazu den benötigten Userinput ein.
7. **Füllung** (6 Punkte)
   Es soll dem User möglich sein, die „Füllung“ der Aufgaben 1.-6. bestimmen zu können (bis jetzt wars #). Zudem soll es dem User möglich sein jede n’te Zeile mit einer gewählten „Füllung“. Hier bedeutet z.B. „n’te = 3“, dass der User 3 eingibt, und dadurch jede 3. Zeile mit einem anderen Symbol zu befüllen ist.

8. **„Steigung“ der Aufgaben 1-6.** (BONUS)
   Hier soll durch die Eingabe der „Steigung“ gesteuert werden wie „spitz“ das generierte Muster ist. Vergessen Sie nicht, Sie können das Internet verwenden!

   z.B.
   Steigung 1 bedeutet dass $\frac{\Delta y}{\Delta x} = 1$, y bedeutet hier die vertikale (Zeilen) und x die horizontale (Spalten).
   - Lösen Sie zuerst das Problem mit Steigung kleiner als 1 und danach größer als 1.
   - Achtung! Da die Höhe (auf der y Achse) vom User fixiert ist, müss solange die Schritte in x gegangen werden, bis diese Höhe erreicht ist!
   - Wählen Sie frei ob sie wenn die Steigung nicht genau dargestellt werden kann, ob sie floor, ceiling oder round verwenden. Dies beeinflusst das generierte Muster, sie sind jedoch alle richtig. Es wird round empfohlen.
   - Achtung! Gehen Sie davon aus dass Δx und Δy eines Symbols das gleiche ist! Damit ist folgendes gemeint.
     z.B. die Eingabe „Steigung 1“ hat folgendes Bild zur folge, obwohl in echt, hier nicht die Steigung 1 vorliegt da ein Symbol höher als breit ist. Wir ignorieren das aber, da das nur ein Datstellungsproblem ist!

```
//              #
//             ##
//            ###
//           ####
//          #####
//         ######
```

Hier ein Beispiel zur Steigung $0.67 = \frac{\Delta y}{\Delta x}$ und ein möglicher Lösungsversuch:

Stellen Sie sich auf die linke ecke, dort ist dort ist die koordinate $x = 1$. Wir gehen hier nach rechts weiter. Wenn wir wissen wollen welches feld von der Linie berührt wird, sagt uns $y = \frac{\Delta y}{\Delta x} * x$ das Feld. $\frac{\Delta y}{\Delta x}$ kommt vom user als Eingabe, x und y ist die Position eines Arrays.
Also $0.67 * 1 = 0.67$. Bedeutet gerundet $x = 1$ und $y = 1$.
Für $x = 2$ $y = 0.67 * 2 = 1.33$ und gerundet $y = 1$. Wenn wir das weiter machen, haben wir

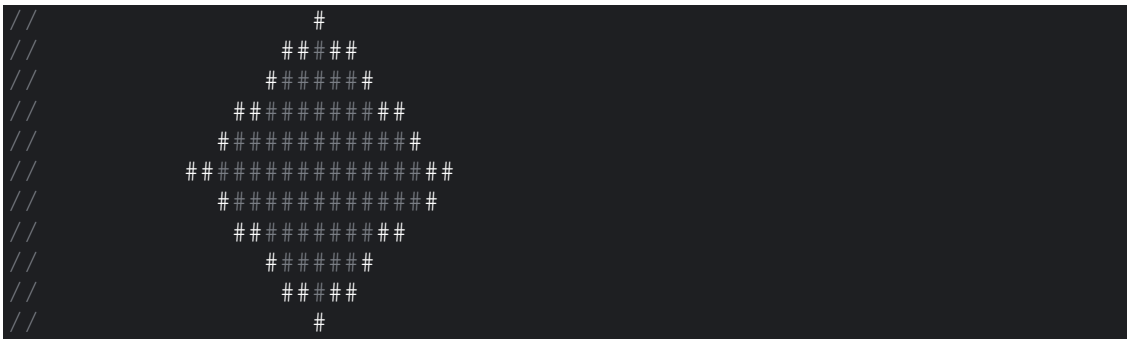
x	1	2	3	4	5	6	7	8	**9**
y	0.67	1.33	2	2.67	3.33	4	4.67	5.33	6
runden	1	1	2	3	3	4	5	5	**6**

Wir haben also in der letzten Reihe **9** Symbole und **6** Symbole als höhe, was wieder $\frac{\Delta y}{\Delta x} = \frac{\mathbf{6}}{\mathbf{9}} = 0.67$ ergibt. Die Zeile „runden“ in der Tabelle ergibt die weißen ‚#‘.

Verwende ein 2d-Array um diese Linie anzulegen. Danach suche das Symbol ‚#‘ in jeder Zeile und fülle nach rechts auf.

Für kompliziertere Muster, wie die Raute, teile das Problem in 4 kleinere Probleme (das wir und gerade angeschaut haben ist eines davon) und füge diese danach zusammen. Also

Ergibt



Beachte die Eingabe des Users (länge er Diagonale, nicht die Höhe).