Aufgabenblatt 2

Kompetenzstufe 1

Allgemeine Informationen zum Aufgabenblatt:

- Die Abgabe erfolgt in TUWEL. Bitte laden Sie Ihr IntelliJ-Projekt bis spätestens **Freitag, 15.11.2019 13:00 Uhr** in TUWEL hoch.
- Zusätzlich müssen Sie in TUWEL ankreuzen, welche Aufgaben Sie gelöst haben und während der Übung präsentieren können.
- Ihre Programme müssen kompilierbar und ausführbar sein.
- Ändern Sie bitte **nicht** die **Dateinamen** und die **vorhandene Ordnerstruktur**.
- Bei manchen Aufgaben finden Sie Zusatzfragen. Diese Zusatzfragen beziehen sich thematisch auf das erstellte Programm. Sie müssen diese Zusatzfragen für gekreuzte Aufgaben in der Übung beantworten können. Sie können die Antworten dazu als Java-Kommentare in die Dateien schreiben.
- Verwenden Sie, falls nicht anders angegeben, für alle Ausgaben `System.out.println()` bzw. `System.out.print()`.
- Verwenden Sie für die Lösung der Aufgaben keine Aufrufe (Klassen) aus der Java-API, außer diese sind ausdrücklich erlaubt.
- Erlaubt sind die Klassen `String`, `Math`, `StdDraw` und `Scanner` oder Klassen, die in den Hinweisen zu den einzelnen Aufgaben aufscheinen.

In diesem Aufgabenblatt werden folgende Themen behandelt:

- for-Schleife, while-Schleife
- Verschachtelung von Schleifen
- Zeichnen mit `StdDraw` unter Verwendung von Schleifen und Verzweigungen
- Implementieren und Verwenden von Methoden
- Umgang mit der Klasse `Scanner`

Aufgabe 1

Erweitern Sie die Methode `main`:

- Implementieren Sie die in Abbildung 1 gezeigte optische Täuschung, basierend auf der *Ouchi-Illusion*[1].
- Setzen Sie die Fenstergröße auf 512×512 Pixel. Die einzelnen (weißen und schwarzen) Rechtecke haben eine Abmessung von 16×4 Pixel.
- Die vier inneren Quadrate haben eine Größe von 128×128 Pixel. Die Rechtecke in den inneren Quadraten sind um 90° gedreht und haben eine Abmessung von 4×16 Pixel. Die Mittelpunkte der vier inneren Quadrate haben die folgenden Koordinaten in Leserichtung von links oben nach rechts unten: $(128, 384)$, $(384, 384)$, $(128, 128)$ und $(384, 128)$.

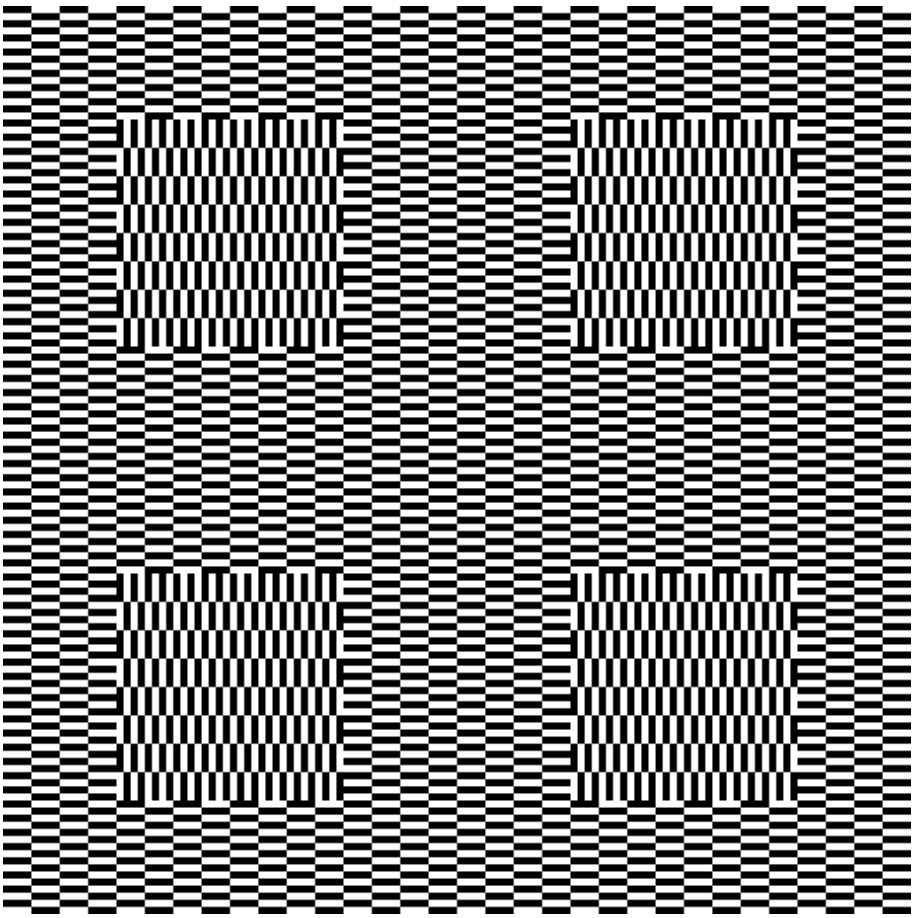

Abbildung 1: Ergebnis der Ouchi-Illusion laut den entsprechenden Vorgaben.

Zusatzfrage(n):

1. Wenn Sie while-Schleifen verwendet haben, überlegen Sie: Ist es sinnvoll und möglich, diese durch for-Schleifen zu ersetzen?

[1] Die *Ouchi-Illusion* wurde nach seinem Entdecker, dem japanischen Künstler Hajime Ouchi, benannt.

Aufgabe 2

Erweitern Sie die Methode `main`:

- Implementieren Sie ein Programm, welches die unten gezeigten Rautenfiguren erzeugt. Die Höhe der Figuren wird mit der Variablen `int figHeight` gesteuert. Die Höhe gibt die Anzahl der Zeilen an und darf nur positive ungerade Werte annehmen. Sie können den Wert für `figHeight` gerne mittels Scanner einlesen.
  Beispiele:

```
+++++++/\+++++++
++++++/00\++++++
+++++/1001\+++++
++++/010010\++++
+++/10100101\+++    +++++/\+++++
++/0101001010\++    ++++/00\++++
+/101010010101\+    +++/1001\+++
|01010100101010|    ++/010010\++
+\101010010101/+    +/10100101\+    +++/\+++
++\0101001010/++    |0101001010|    ++/00\++
+++\10100101/+++    +\10100101/+    +/1001\+
++++\010010/++++    ++\010010/++    |010010|
+++++\1001/+++++    +++\1001/+++    +\1001/+
++++++\00/++++++    ++++\00/++++    ++\00/++
+++++++\/+++++++    +++++\/+++++    +++\/+++    ||
```

Ausgaben (von links nach rechts) mit `figHeight` 15, 11, 7 und 1.

(!) Die Zeichen innerhalb der Raute entsprechen den Ziffern '0' und '1'. Damit der Backslash ausgegeben werden kann, müssen Sie '\\' schreiben. Es handelt sich dabei um eine sogenannte *Escape Sequence*.

(!) Für die Realisierung des Beispiels dürfen keinerlei Strings oder Arrays verwendet werden. Auch Methoden aus diesen Klassen dürfen nicht zum Einsatz kommen.

Zusatzfrage(n):

1. Kann das Beispiel mit einer einzelnen Schleife gelöst werden? Wenn ja, wie würden Sie bei der Implementierung vorgehen?

Aufgabe 3

Aufgabenstellung:

- Implementieren Sie eine Methode `isHappyNumber`:

  ```
  boolean isHappyNumber(int number)
  ```

  Diese Methode überprüft, ob es sich bei einer gegebenen positiven ganzen Zahl `number` um eine *fröhliche* oder eine *traurige* Zahl handelt. Es handelt sich dann um eine fröhliche Zahl, wenn folgende Rechenvorschrift mit einer endlichen Anzahl an Iterationsschritten zur Zahl 1 führt:

  - Es wird jede Ziffer einer Zahl quadriert.
  - Die Quadrate werden summiert.

  Auf die Summe der Quadrate wird erneut diese Rechenvorschrift angewendet. Wird die Zahl 1 erreicht, dann ist die Berechnung am Ende angekommen und es handelt sich um eine fröhliche Zahl.

  Folgende Beispiele repräsentieren fröhliche Zahlen:
  23 $\rightarrow 2^2 + 3^2 = 13 \rightarrow 1^2 + 3^2 = 10 \rightarrow 1^2 + 0^2 = \mathbf{1}$
  97 $\rightarrow 9^2 + 7^2 = 130 \rightarrow 1^2 + 3^2 + 0^2 = 10 \rightarrow 1^2 + 0^2 = \mathbf{1}$
  7 $\rightarrow 7^2 = 49 \rightarrow 4^2 + 9^2 = 97 \rightarrow 9^2 + 7^2 = 130 \rightarrow 1^2 + 3^2 + 0^2 = 10 \rightarrow 1^2 + 0^2 = \mathbf{1}$

  Sollte die oben beschriebene Rechenvorschrift (Ziffern quadrieren und summieren) mit einer endlichen Anzahl an Iterationsschritten zur Zahl 4 führen und nicht zur Zahl 1, dann handelt es sich um eine **traurige Zahl**. Nachdem die Zahl 4 erreicht wurde, beginnt ein Zyklus, der immer wieder zur Zahl 4 führt.

  Folgende Beispiele repräsentieren traurige Zahlen:
  58 $\rightarrow 5^2 + 8^2 = 89 \rightarrow 8^2 + 9^2 = 145 \rightarrow 1^2 + 4^2 + 5^2 = 42 \rightarrow 4^2 + 2^2 = 20 \rightarrow 2^2 + 0^2 = \mathbf{4}$
  40 $\rightarrow 4^2 + 0^2 = 16 \rightarrow 1^2 + 6^2 = 37 \rightarrow 3^2 + 7^2 = 58 \rightarrow 89 \rightarrow 145 \rightarrow 42 \rightarrow 20 \rightarrow \mathbf{4}$
  5 $\rightarrow 5^2 = 25 \rightarrow 2^2 + 5^2 = 29 \rightarrow 2^2 + 9^2 = 85 \rightarrow 8^2 + 5^2 = 89 \rightarrow 145 \rightarrow 42 \rightarrow 20 \rightarrow \mathbf{4}$

  Vorbedingung: `number` > 0.

(!) Für die Realisierung des Beispiels dürfen keinerlei Strings oder Arrays verwendet werden. Auch Methoden aus diesen Klassen dürfen nicht zum Einsatz kommen.

- Implementieren Sie eine Methode `countHappyNumbers`:

  ```
  int countHappyNumbers(int start, int end)
  ```

  Diese Methode zählt, wie viele fröhliche Zahlen im Intervall [`start`, `end`] vorkommen, und gibt diese Anzahl zurück.
  Vorbedingung: $((\texttt{start} \wedge \texttt{end}) > 0) \wedge (\texttt{start} \leq \texttt{end})$

- Implementieren Sie eine Methode `printHappyNumbers`:

  ```
  void printHappyNumbers(int start, int end)
  ```

  Diese Methode gibt alle fröhlichen Zahlen im Intervall [`start`, `end`] mittels `System.out.println()` auf der Konsole aus.
  Vorbedingung: $((\texttt{start} \wedge \texttt{end}) > 0) \wedge (\texttt{start} \leq \texttt{end})$

Zusatzfrage(n)

1. Wie viele Parameter können einer Methode übergeben werden?
2. Kann eine Methode mehr als einen Rückgabewert haben?

Aufgabe 4

Aufgabenstellung:

- Implementieren Sie ein *Guessing Game* in Anlehnung an den Programmcode aus der Vorlesung, bei dem ein zufällig gewählter Buchstabe erraten werden soll. Zusätzlich wird das Programm in Methoden unterteilt und um grafische Komponenten ergänzt.

- Es sollen folgende vier Methoden implementiert werden:

  1. Eine Methode für die Generierung eines zufälligen Buchstabens. Der generierte Buchstabe liegt im Intervall von [a, z].
  2. Eine Methode für das Einlesen der Eingaben. Verwenden Sie, wie in der Vorlesung vorgestellt, den Scanner, um die Daten von der Konsole einzulesen. Geben Sie den eingelesenen Buchstaben zurück. Falsche Eingaben werden innerhalb der Methode behandelt.
  3. Eine Methode, die eine Nachricht (`String`) für die spielende Person auf der Konsole ausgeben kann.
  4. Eine Methode, die in jeder Spielrunde die grafischen Komponenten zeichnet und in einem StdDraw-Fenster ausgibt.

  Sie können weitere Methoden implementieren, falls dadurch das Programm übersichtlicher wird.

- Nachfolgend wird der Spielablauf beschrieben, wie mit Sonderfällen umgegangen werden muss und wie die grafischen Komponenten des Spiels aussehen sollen.

  – Spielablauf: Eine spielende Person gibt einen Buchstaben im Intervall von [a, z] ein (Rateversuch). Danach wird auf der Konsole ausgegeben, ob der gesuchte Buchstabe größer oder kleiner als der Rateversuch ist. Hat die spielende Person den gesuchten Buchstaben erraten, ist das Spiel beendet und das Spiel wurde gewonnen. Wurde der gesuchte Buchstabe nach 5 Rateversuchen nicht erraten, ist das Spiel beendet und die spielende Person hat verloren.

  – Sonderfälle: Falsche Eingaben, wie Zahlen, Großbuchstaben oder Zeichenketten, werden ignoriert und es geht kein Rateversuch verloren. Die spielende Person wird immer mit einer Meldung auf der Konsole informiert.

  – Grafische Ausgabe: Für die grafische Ausgabe legen Sie ein Fenster der Größe 600×300 Pixel an. Wie in Abbildung 2a gezeigt, werden die verbleibenden Versuche in textueller Form (blauer Text) ausgegeben. Zusätzlich wird im Ausgabefenster ein graues Rechteck der Größe 540×120 Pixel gezeichnet. In diesem Rechteck wird die Distanz (Dezimalwert der Absolutdifferenz) zwischen gesuchten und eingegebenen Buchstaben visualisiert (Abbildung 2b-2f). Dazu wird ein 500×50 Pixel großer oranger Balken verwendet. Je weiter der Balken rechts bei der roten Linie ist (Abbildung 2c), desto weiter entfernt ist der eingegebene Buchstabe vom gesuchten Buchstaben. Umso kleiner der Balken wird und somit näher an die grüne Linie herankommt (Abbildung 2d), desto geringer ist die Entfernung zwischen dem gesuchten und eingegebenen Buchstaben. Die Abbildung 2e zeigt keinen orangen Balken, da der Buchstabe erraten wurde (Zustand *Gewonnen*) und die Distanz zwischen den Buchstaben 0 ist. Über dem Balkenbereich befindet sich ein

schwarzer Text "Distanz zwischen den Buchstaben:". Zusätzlich soll die Information, ob gewonnen wurde, mit `Gewonnen!`, wie in Abbildung 2e gezeigt, realisiert werden. Andernfalls soll der Schriftzug `Verloren!` (Abbildung 2f) angezeigt werden, wenn das Spiel verloren wurde.

Abbildung 2: Beispielausgaben von verschiedenen Spielzuständen.

Zusatzfrage(n):

1. Wie können die eingegebenen Daten (bzw. deren Datentypen) mit Hilfe des Scanners unterschieden werden?