Aufgabenblatt 3

Kompetenzstufe 1 & Kompetenzstufe 2

Allgemeine Informationen zum Aufgabenblatt:

- Die Abgabe erfolgt in TUWEL. Bitte laden Sie Ihr IntelliJ-Projekt bis spätestens **Freitag, 29.11.2019 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:

- Codeanalyse und Implementierungsstil
- Implementieren von Methoden
- Überladen von Methoden
- Aufrufen von Methoden
- Rekursion

Aufgabe 1

Aufgabenstellung:

a) Analysieren Sie den gegebenen Code und beschreiben Sie dessen Funktionsweise. Erklären Sie was die einzelnen Methoden machen.

b) Bei der Erstellung wurde nicht auf eine sinnvolle Namensgebung bei den Methoden geachtet. Ändern Sie dazu bitte Methoden- und Variablennamen so ab, dass diese die Funktionalität der jeweiligen Methode widerspiegeln. Zusätzlich formatieren Sie den Code so, dass dieser besser leserlich wird.

c) In einem letzten Schritt kürzen Sie den Programmcode. Dazu werden alle Methoden zu einer einzigen Methode verschmolzen, die in `main` aufgerufen wird und das gleiche Ergebnis generiert. Sie können die anderen Methoden auskommentieren.

Aufgabe 2

Aufgabenstellung:

- Implementieren Sie eine Methode `addTags`:

  ```
  void addTags(String text, char tag)
  ```

  Diese Methode gibt einen String `text` formatiert aus. Es wird am Anfang, am Ende und in der Mitte von `text` das Zeichen `tag` eingefügt. Zum Schluss geben Sie den veränderten String mittels `System.out.println()` auf der Konsole aus. Sollte der übergebene String `text` eine ungerade Länge haben, dann wird das Zeichen in der Mitte so eingefügt, dass der erste Teil des geteilten Strings um ein Zeichen länger ist, als der zweite Teil.
  Vorbedingung: `text != null`.

  Beispiel(e):
  `addTags("A", '?')` liefert `?A??`
  `addTags("AB", ',')` liefert `,A,B,`
  `addTags("Hello World!", ':')` liefert `:Hello :World!:`
  `addTags("Java-Programmierung", '+')` liefert `+Java-Progr+ammierung+`
  `addTags("Das ist ein Test", '-')` liefert `-Das ist -ein Test-`

- Implementieren Sie eine Methode `addTags`:

  ```
  void addTags(int number, char tag)
  ```

  Diese Methode gibt eine Zahl `number` formatiert aus. Es wird am Anfang, am Ende und in der Mitte von `number` das Zeichen `tag` eingefügt und der resultierende String mittels `System.out.println()` auf der Konsole ausgegeben. Sollte die übergebene Zahl `number` eine ungerade Anzahl an Stellen haben, dann wird das Zeichen in der Mitte so eingefügt, dass der erste Teil der geteilten Zahl um eine Ziffer länger ist, als der zweite Teil. Für die Realisierung der Methode darf die Zahl `number` **nicht** in einen String umgewandelt werden!
  Vorbedingung: `number` $\geq$ `0`.

  Beispiel(e):
  `addTags(1, '$')` liefert `$1$$`
  `addTags(35, '*')` liefert `*3*5*`
  `addTags(657, ':')` liefert `:65:7:`
  `addTags(2048, '#')` liefert `#20#48#`
  `addTags(26348, '+')` liefert `+263+48+`

- Implementieren Sie eine Methode `addTags`:

  ```
  void addTags(String text, String tags)
  ```

  Diese Methode gibt einen String `text` formatiert aus. Es wird am Anfang, am Ende und in der Mitte von `text` jeweils ein Zeichen von `tags` eingefügt und der resultierende String mittels `System.out.println()` auf der Konsole ausgegeben. Sollte der übergebene String

`text` eine ungerade Länge haben, dann wird das Zeichen in der Mitte so eingefügt, dass der erste Teil des geteilten Strings um ein Zeichen länger ist, als der zweite Teil. Die Variable `tags` ist ein String. Daher wird der übergebene String `text` mit verschiedenen Zeichen hintereinander ausgegeben. Verwenden Sie dazu bereits vorhandene Methoden und vermeiden Sie die Duplizierung von Code. Vorbedingungen: `text != null` und `tag != null`.

Beispiel(e):
`addTags("Hello World!", "+#$")` liefert
```
+Hello +World!+
#Hello #World!#
$Hello $World!$
```
`addTags("Java-Programmierung", ":*&!)")` liefert
```
:Java-Progr:ammierung:
*Java-Progr*ammierung*
&Java-Progr&ammierung&
!Java-Progr!ammierung!
```

Zusatzfrage(n):

1. Wie kann bei der Methode `addTags` ein voreingestelltes Standardtrennzeichen verwendet werden, sodass bei Aufruf von `addTags("Hello World!")` die Ausgabe `XHello XWorld!X` erfolgt?

Aufgabe 3

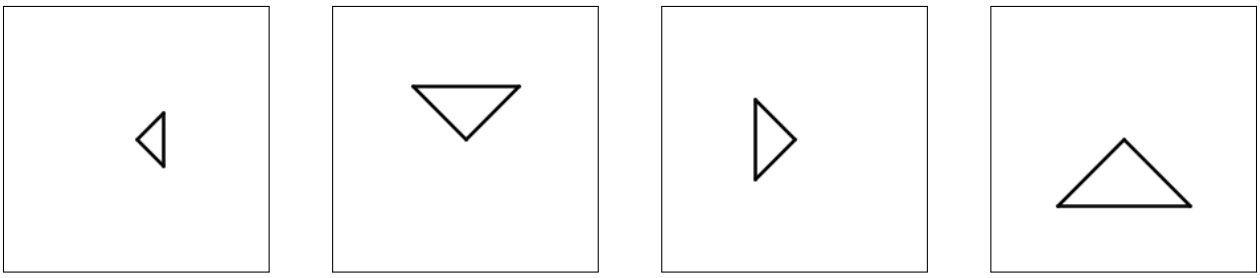
Aufgabenstellung:

(!) Wir haben bei dieser Aufgabe auf Vorbedingungen verzichtet. Ihre Implementierung muss zumindest für die Testbeispiele in der Projektdatei funktionieren. Sie dürfen aber auch eigene Testfälle implementieren.

- Implementieren Sie die Methode `drawTriangle`:

```
void drawTriangle(double centerX, double centerY,
                  double height, int direction)
```

  Diese Methode hat vier Parameter und zeichnet ein rechtwinkeliges gleichschenkeliges Dreieck, bestehend aus drei Linien (Abbildung 1). Die Parameter `centerX` und `centerY` beschreiben den Punkt der Dreiecksspitze und der Parameter `height` die Höhe des Dreiecks. Die Länge der Basis (Hypotenuse) des Dreiecks entspricht zweimal der Höhe `height`. Der Parameter `direction` beschreibt die Richtung (ausgehend von der Dreiecksspitze zu den zwei anderen Eckpunkten), in die das Dreieck gezeichnet wird. Der Parameter `direction` kann die Werte 0 für Richtung Osten, 1 für Richtung Norden, 2 für Richtung Westen und 3 für Richtung Süden annehmen.



(a) (b) (c) (d)

Abbildung 1: Rechtwinkelige gleichschenkelige Dreiecke mit verschiedenen Höhen, gezeichnet in verschiedene Richtungen.

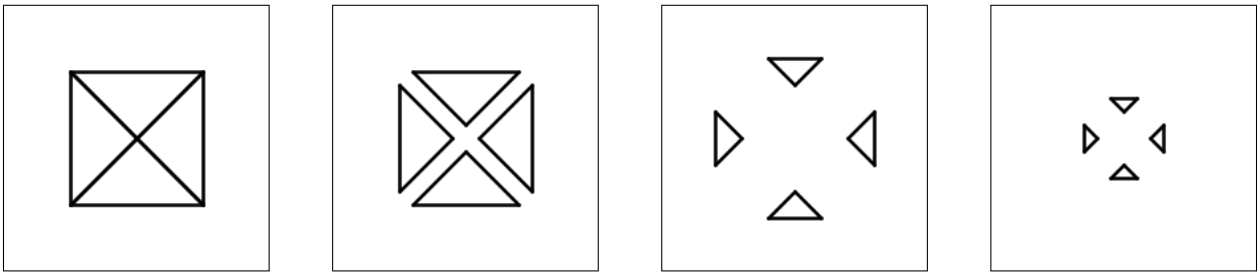
  Für die hier dargestellten Dreiecksmuster wurden folgende Aufrufe verwendet:
  für Abbildung 1a → `drawTriangle(100, 100, 20, 0)`
  für Abbildung 1b → `drawTriangle(100, 100, 40, 1)`
  für Abbildung 1c → `drawTriangle(100, 100, 30, 2)`
  für Abbildung 1d → `drawTriangle(100, 100, 50, 3)`

- Implementieren Sie die Methode `drawTrianglePattern`:

```
void drawTrianglePattern(double centerX, double centerY,
                         int height, int centerShift)
```

  Diese Methode zeichnet vier Dreiecke wie in den Abbildungen 2a-2d dargestellt. Die Parameter `centerX` und `centerY` geben den Mittelpunkt des Dreiecksmusters an. Mit dem Parameter

`height` wird die Höhe der einzelnen Dreiecke angegeben. Der Parameter `centerShift` gibt an, wie weit jede der Dreiecksspitzen vom Zentrum weg verschoben ist. Für Abbildung 2a wurde ein `centerShift` 0 verwendet, wodurch sich alle vier Dreiecksspitzen im Zentrum treffen. Für Abbildung 2b wurde ein `centerShift` von 10 gewählt, wodurch jede Dreiecksspitze um 10 Pixel in die jeweilige Richtung verschoben ist.



(a) (b) (c) (d)

Abbildung 2: Dreiecksmuster, bestehend aus jeweils 4 Dreiecken, die um einen `centerShift` vom Zentrum entfernt gezeichnet werden können.

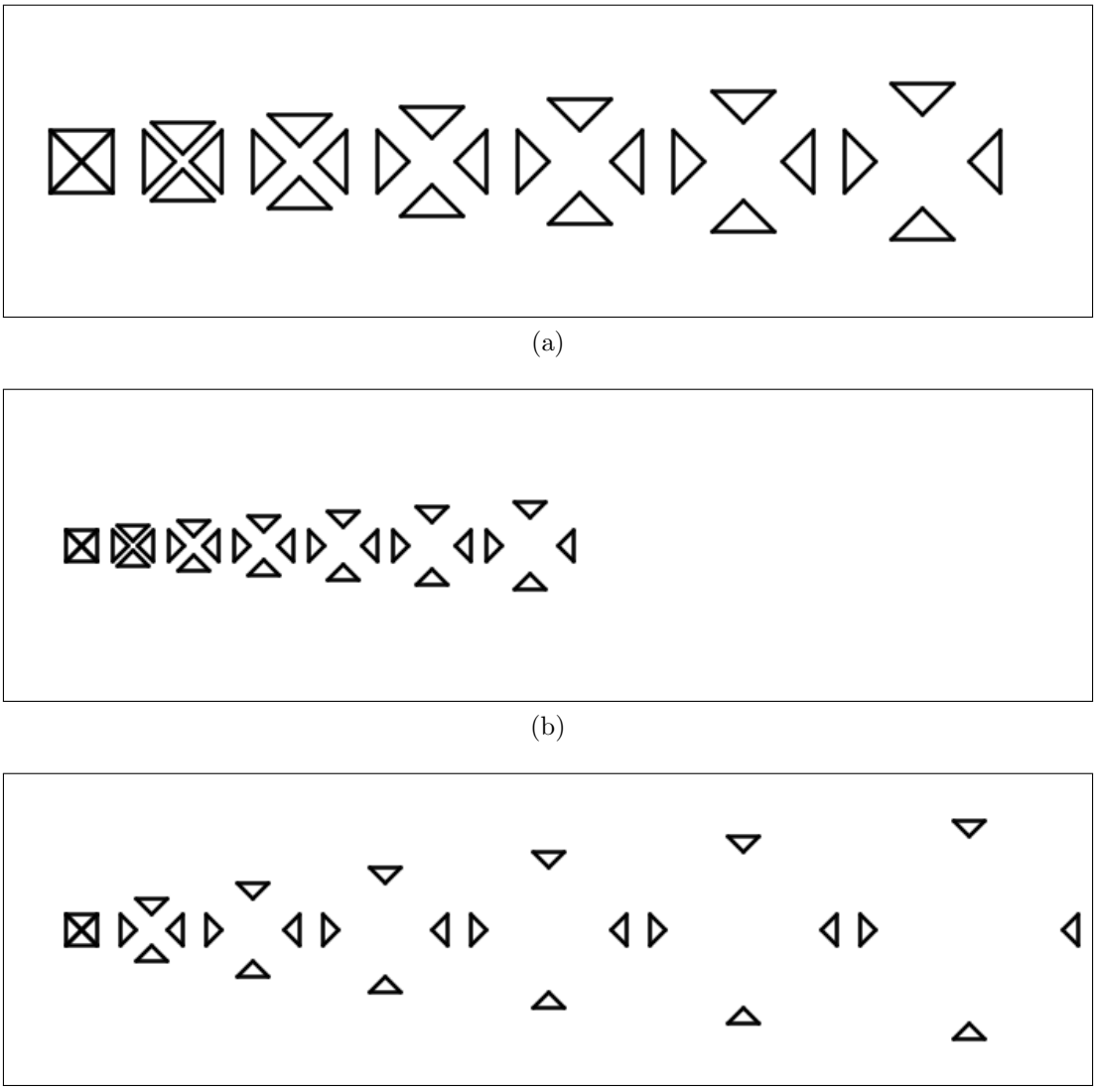
Für die hier dargestellten Dreiecksmuster wurden folgende Aufrufe verwendet:
für Abbildung 2a → `drawTrianglePattern(100, 100, 50, 0)`
für Abbildung 2b → `drawTrianglePattern(100, 100, 40, 10)`
für Abbildung 2c → `drawTrianglePattern(100, 100, 20, 40)`
für Abbildung 2d → `drawTrianglePattern(100, 100, 10, 20)`

- Implementieren Sie die Methode `drawTrianglePatternLine`:

```
void drawTrianglePatternLine(double centerX, double centerY,
            int height, int growth, int distance)
```

Diese Methode hat fünf Parameter und zeichnet eine Reihe mit sieben Dreiecksmustern (siehe Abbildungen 3a-3c), bestehend aus jeweils vier Dreiecken, die mit der Methode `drawTrianglePattern` erzeugt werden können. Die Parameter `centerX` und `centerY` geben den Mittelpunkt des ersten Dreiecksmusters an. Mit dem Parameter `height` wird die Höhe der einzelnen Dreiecke angegeben. Der Parameter `growth` bestimmt dabei, um wie viele Pixel bei jedem Dreiecksmuster die Dreiecke sich vom Zentrum entfernen (entspricht `centerShift` bei Methode `drawTrianglePattern`). Mit dem letzten Parameter `distance` kann noch angegeben werden, wie groß der Abstand zwischen zwei benachbarten Dreiecksmustern sein soll.

Für die hier dargestellten Dreiecksmuster wurden folgende Aufrufe verwendet:
für Abbildung 3a → `drawTrianglePatternLine(50,100,20,5,20)`
für Abbildung 3b → `drawTrianglePatternLine(50,100,10,3,10)`
für Abbildung 3c → `drawTrianglePatternLine(50,100,10,10,15)`



(a)

(b)

(c)

Abbildung 3: Verschiedene Reihen mit Dreiecksmustern. Jede Reihe besteht aus sieben Dreiecksmustern, die sich durch die Parameter `height`, `growth` und `distance` in ihrem Erscheinungsbild konfigurieren lassen.

Aufgabe 4

Aufgabenstellung:

(!) Gilt für alle zu implementierenden Methoden: Sie dürfen keine globalen Variablen oder zusätzliche eigene Hilfsmethoden verwenden. Die vorgegebenen Methodenköpfe dürfen nicht erweitert oder geändert werden. Für die Implementierung der Aufgabenstellung dürfen keine Schleifen verwendet werden.

- Implementieren Sie eine rekursive Methode `printNumbersAscending`:

  ```
  void printNumbersAscending(int start, int end, int divider)
  ```

  Diese Methode gibt alle Zahlen im Intervall von [`start`, `end`] **aufsteigend** aus, die sich durch die Zahl `divider` restlos teilen lassen. Vorbedingung: `start` $\leq$ `end` und `divider` > 0.

- Implementieren Sie eine rekursive Methode `printNumbersDescending`:

  ```
  void printNumbersDescending(int start, int end, int divider)
  ```

  Diese Methode gibt alle Zahlen im Intervall von [`start`, `end`] **absteigend** aus, die sich **nicht** durch die Zahl `divider` restlos teilen lassen. Vorbedingung: `start` $\leq$ `end` und `divider` > 0.

- Implementieren Sie eine rekursive Methode `calcDigitProduct`:

  ```
  int calcDigitProduct(int number)
  ```

  Diese Methode berechnet das Produkt aller Ziffern der Zahl `number` und gibt es zurück. Nullen innerhalb der Zahl werden bei der Multiplikation aber ignoriert. Vorbedingung: `number` > 0.

  Beispiele:
  `calcDigitProduct(1)` liefert 1
  `calcDigitProduct(102)` liefert 2
  `calcDigitProduct(1234)` liefert 24
  `calcDigitProduct(10000)` liefert 1
  `calcDigitProduct(93842)` liefert 1728
  `calcDigitProduct(875943789)` liefert 15240960

- Implementieren Sie eine rekursive Methode `filterNumbersInString`:

  ```
  String filterNumbersInString(String text)
  ```

  Diese Methode entfernt alle Vorkommen der Ziffern im Intervall von [0, 9] aus dem String `text`. Der neu entstandene String wird zurückgegeben. Vorbedingung: `text != null`.

  Beispiele:
  `filterNumbersInString("hallo")` liefert `hallo`
  `filterNumbersInString("Test 1 mit 45 Punkten!")` liefert `Test  mit  Punkten!`
  `filterNumbersInString("1A1234567890B0")` liefert `AB`

- Implementieren Sie eine rekursive Methode `reverseStringDoubleLetter`:

  ```
  String reverseStringDoubleLetter(String text, char letter)
  ```

  Diese Methode erzeugt einen neuen String, der alle Zeichen des Strings `text` in umgekehrter Reihenfolge beinhaltet. Zusätzlich wird in diesem neuen String bei jedem Vorkommen von `letter` das Zeichen `letter` ein zusätzliches Mal an dieser Stelle eingefügt.
  Vorbedingung: `text != null`.

  Beispiele:
  `reverseStringDoubleLetter("X", 'X')` liefert `XX`
  `reverseStringDoubleLetter("Hallo", 'l')` liefert `ollllaH`
  `reverseStringDoubleLetter("String umdrehen!", 'z')` liefert `!neherdmu gnirtS`

Zusatzfrage(n):

1. Was ist *Fundiertheit* im Zusammenhang mit der Rekursion?
2. Was ist *Fortschritt* im Zusammenhang mit der Rekursion?
3. Warum benötigen Sie bei einer Rekursion eine Abbruchbedingung?
4. Gibt es eine Limitierung für die Rekursionstiefe?