Aufgabenblatt 3

Kompetenzstufe 1 & Kompetenzstufe 2

Allgemeine Informationen zum Aufgabenblatt:

- Die Abgabe erfolgt in TUWEL. Bitte laden Sie Ihr IntelliJ-Projekt bis spätestens **Mittwoch, 29.11.2023 23:55 Uhr** in TUWEL hoch.
- Zusätzlich müssen Sie in TUWEL ankreuzen, welche Aufgaben Sie gelöst haben.
- 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` und `CodeDraw`, es sei denn in den Hinweisen zu den einzelnen Aufgaben ist etwas anderes angegeben.
- Bitte beachten Sie die Vorbedingungen! Sie dürfen sich darauf verlassen, dass alle Aufrufe die genannten Vorbedingungen erfüllen. Sie müssen diese nicht in den Methoden überprüfen.

In diesem Aufgabenblatt werden folgende Themen behandelt:

- Codeanalyse und Implementierungsstil
- Implementieren von Methoden
- Überladen von Methoden
- Rekursion
- Rekursion und CodeDraw
- Vergleich von rekursiver und iterativer Implementierung

Aufgabe 1 (1 Punkt)

Aufgabenstellung:

a) Analysieren Sie den gegebenen Code (Spaghetticode[1]) und beschreiben Sie dessen Funktionsweise in wenigen Sätzen.

b) Bei der Erstellung wurde nicht auf eine sinnvolle Gliederung und Formatierung geachtet. Beschreiben Sie Teile des Codes, die Sie ändern würden und warum.

c) Identifizieren Sie Codebereiche, die in Methoden aufgeteilt werden können. Schreiben Sie entsprechende Methoden und rufen Sie diese in `main` auf, um die identische Ausgabe zu erhalten. Der gegebene Code gibt ein Muster mit fester Breite aus. Ändern Sie den Code so um, dass dieser über den Aufruf der Methoden verschieden breite Muster generieren kann. Nehmen Sie an, dass die Breite nur positive gerade Werte annehmen kann.

d) Nach Umbau müssen die Aufrufe in `main` die gleiche Ausgabe produzieren wie der Spaghetticode.

[1] `https://de.wikipedia.org/wiki/Spaghetticode`

Aufgabe 2 (1 Punkt)

Implementieren Sie folgende Aufgabenstellung:

- Implementieren Sie eine Methode `addChar`:

  ```
  void addChar(String text, char character)
  ```

  Diese Methode gibt einen String `text` formatiert aus. Es wird zwischen zwei Zeichen von `text` das Zeichen `character` abwechselnd zweimal bzw. einmal eingefügt. Geben Sie den veränderten String in einer Zeile auf der Konsole aus.
  Vorbedingung: `text != null`.

  Beispiel(e):
  `addChar("", '&')` liefert keine Ausgabe
  `addChar("A", '+')` liefert `A`
  `addChar("CW", '*')` liefert `C**W`
  `addChar("EP1", '-')` liefert `E--P-1`
  `addChar("Index", '#')` liefert `I##n#d##e#x`

- Implementieren Sie eine Methode `addChar`:

  ```
  void addChar(int number, char character)
  ```

  Diese Methode gibt eine Zahl `number` formatiert aus. Es wird zwischen zwei Ziffern von `number` das Zeichen `character` abwechselnd zweimal bzw. einmal eingefügt. Der resultierende String wird in einer Zeile auf der Konsole ausgegeben. Für die Realisierung der Methode darf die Zahl in einen String umgewandelt werden (`Integer.toString(...)`). Vermeiden Sie redundanten Code, indem Sie eine bereits implementierte Methode verwenden.
  Vorbedingung: `number` $\geq$ `0`.

  Beispiel(e):
  `addChar(1, '.')` liefert `1`
  `addChar(42, ':')` liefert `4::2`
  `addChar(148, '$')` liefert `1$$4$8`
  `addChar(2048, ')')` liefert `2))0)4))8`
  `addChar(131719, '%')` liefert `1%%3%1%%7%1%%9`

- Implementieren Sie eine Methode `addChar`:

  ```
  void addChar(String text, String characters)
  ```

  Diese Methode gibt einen String `text` unterschiedlich formatiert aus. Es wird zwischen zwei Zeichen von `text` das erste Zeichen von `characters` abwechselnd zweimal bzw. einmal eingefügt. Danach wird das zweite Zeichen von `characters` abwechselnd zweimal bzw. einmal zwischen den Zeichen von `text` eingefügt, usw. Der resultierende String wird in einer Zeile auf der Konsole ausgegeben. Dies soll für jedes Zeichen aus dem String `characters` geschehen, d.h. der String `text` wird mit verschiedenen Zeichen formatiert mehrmals ausgegeben. Vermeiden Sie redundanten Code, indem Sie eine bereits implementierte Methode verwenden.
  Vorbedingungen: `text != null` und `characters != null`.

  Beispiel(e):
  `addChar("CW", "!0(")` liefert
  `C!!W`
  `C00W`
  `C((W`
  `addChar("Index", "T1#+")` liefert
  `ITTnTdTTeTx`
  `I11n1d11e1x`
  `I##n#d##e#x`
  `I++n+d++e+x`

- Implementieren Sie eine Methode `addChar`:

  ```
  void addChar(String text)
  ```

  Diese Methode gibt einen String `text` formatiert aus. Es wird zwischen zwei Zeichen von `text` das Zeichen `=` abwechselnd zweimal bzw. einmal eingefügt. Geben Sie zum Schluss den veränderten String in einer Zeile auf der Konsole aus. Vermeiden Sie redundanten Code, indem Sie eine bereits implementierte Methode verwenden.
  Vorbedingung: `text != null`.

  Beispiel(e):
  `addChar("")` liefert keine Ausgabe
  `addChar("CW")` liefert `C==W`
  `addChar("EP1")` liefert `E==P=1`

Aufgabe 3 (1 Punkt)

Implementieren Sie folgende Aufgabenstellung:

(!) Gilt für alle zu implementierenden Methoden: Sie dürfen keine Klassenvariablen 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 `printOddNumbersAscending`:

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

  Diese Methode gibt alle ungeraden Zahlen im Intervall von [`start`, `end`] **aufsteigend** in der Konsole aus. Ausgabe der einzelnen Werte kann untereinander oder nebeneinander erfolgen.
  Vorbedingung: `start` $\leq$ `end`.

  Beispiel:
  `printOddNumbersAscending(3, 15)` liefert 3 5 7 9 11 13 15

- Implementieren Sie eine **rekursive** Methode `printEvenNumbersDescending`:

  ```
  void printEvenNumbersDescending(int end)
  ```

  Diese Methode gibt alle geraden Zahlen im Intervall von [0, `end`] **absteigend** in der Konsole aus. Ausgabe der einzelnen Werte kann untereinander oder nebeneinander erfolgen.
  Vorbedingung: `end` > 0.

  Beispiel:
  `printEvenNumbersDescending(14)` liefert 14 12 10 8 6 4 2 0

- Implementieren Sie eine **rekursive** Methode `countDigitsAboveFive`:

  ```
  int countDigitsAboveFive(int number)
  ```

  Diese Methode zählt alle Ziffern der Zahl `number`, die größer 5 sind und gibt diese Anzahl zurück.
  Vorbedingung: `number` > 0.

  Beispiele:
  `countDigitsAboveFive(6)` liefert 1
  `countDigitsAboveFive(4)` liefert 0
  `countDigitsAboveFive(456)` liefert 1
  `countDigitsAboveFive(1234)` liefert 0
  `countDigitsAboveFive(61238)` liefert 2
  `countDigitsAboveFive(93862)` liefert 3
  `countDigitsAboveFive(518279463)` liefert 4

- Implementieren Sie eine **rekursive** Methode `checkIfStopAvailable`:

  ```
  boolean checkIfStopAvailable(String text)
  ```

  Diese Methode überprüft, ob die Zeichenkette `"STOP"` im String `text` mindestens einmal vorkommt. Es soll die Zeichenkette gefunden werden, egal ob die Buchstaben groß oder klein geschrieben sind. Wird die Zeichenkette gefunden, dann wird `true` zurückgegeben. Ist die Länge von `text` kleiner vier Zeichen, oder es wird die Zeichenkette nicht gefunden, dann wird `false` retourniert. Bei dieser Methode sind ausschließlich die Methoden `length`, `charAt`, `toLowerCase` und `substring` der Klasse `String` erlaubt!
  Vorbedingung: `text != null`.

  Beispiele:
  `checkIfStopAvailable("")` liefert `false`
  `checkIfStopAvailable("sto")` liefert `false`
  `checkIfStopAvailable("STOP")` liefert `true`
  `checkIfStopAvailable("stOkP3tOp")` liefert `false`
  `checkIfStopAvailable("dasStOpistda")` liefert `true`
  `checkIfStopAvailable("ASTOP")` liefert `true`
  `checkIfStopAvailable("asTsTopstoPb")` liefert `true`

Aufgabe 4 (1 Punkt)

Implementieren Sie folgende Aufgabenstellung:

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

- Implementieren Sie eine **rekursive** Methode `countNOrderedLetters`:

  ```
  int countNOrderedLetters(String text, int index)
  ```

  Diese Methode überprüft, wie viele Buchstabenpaare von aufeinander folgenden Buchstaben in `text` vorkommen, die richtig geordnet sind. Ein Paar gilt jeweils als richtig geordnet, wenn die Buchstaben alphabetisch geordnet vorkommen. Zwei gleiche Buchstaben nebeneinander gelten ebenfalls als geordnet. Die Paare im String dürfen sich auch überlappen (z.B. enthält ein String `"cabmhi"` drei geordnete Paare, weil `a <= b`, `b <= m` und `h <= i` ist). Die Zählung beginnt bei `index` (inklusive).
  Vorbedingungen: `text != null`, `text.length() > 0` und `index` beschreibt einen gültigen Index von `text`.

  Beispiele:
  `countNOrderedLetters("bCaa12fgHIdE32zYxYz", 0)` liefert 8
  `countNOrderedLetters("bCaa12fgHIdE32zYxYz", 1)` liefert 7
  `countNOrderedLetters("bCaa12fgHIdE32zYxYz", 6)` liefert 6
  `countNOrderedLetters("bCaa12fgHIdE32zYxYz", 13)` liefert 2
  `countNOrderedLetters("bCaa12fgHIdE32zYxYz", 17)` liefert 1

- Implementieren Sie eine **rekursive** Methode `duplicateSelectedChar`:

  ```
  String duplicateSelectedChar(String text, char character)
  ```

  Diese Methode fügt vor allen Buchstaben (egal ob groß oder klein geschrieben) des Strings `text`, die dem Buchstaben `character` entsprechen den Buchstaben `character` nochmals ein. Der Parameter `character` kann groß oder klein geschrieben vorhanden sein. Zusätzlich wird ganz am Ende des Strings die Anzahl der eingefügten Buchstaben als Zahl angehängt, falls dieser mindestens einmal vorkommt. Das Ergebnis wird als neuer String zurückgeliefert.
  Vorbedingungen: `text != null`, im String `text` kommt keine Ziffer vor und der gesuchte `character` kommt im String `text` maximal 9 mal vor.

  Beispiele:
  `duplicateSelectedChar("", 'o')` liefert `""`
  `duplicateSelectedChar("K", 'k')` liefert `"KK1"`
  `duplicateSelectedChar("abcAijk", 'a')` liefert `"aabcAAijk2"`
  `duplicateSelectedChar("abcAijk", 'A')` liefert `"aabcAAijk2"`
  `duplicateSelectedChar("AbNcdnNopqnUvWN", 'n')` liefert `"AbNNcdnnNNopqnnUvWNN5"`
  `duplicateSelectedChar("AbNcdnNopqnUvWN", 'm')` liefert `"AbNcdnNopqnUvWN"`
  `duplicateSelectedChar("nothing", 'Z')` liefert `"nothing"`

Aufgabe 5 (2 Punkte)

Implementieren Sie folgende Aufgabenstellung:

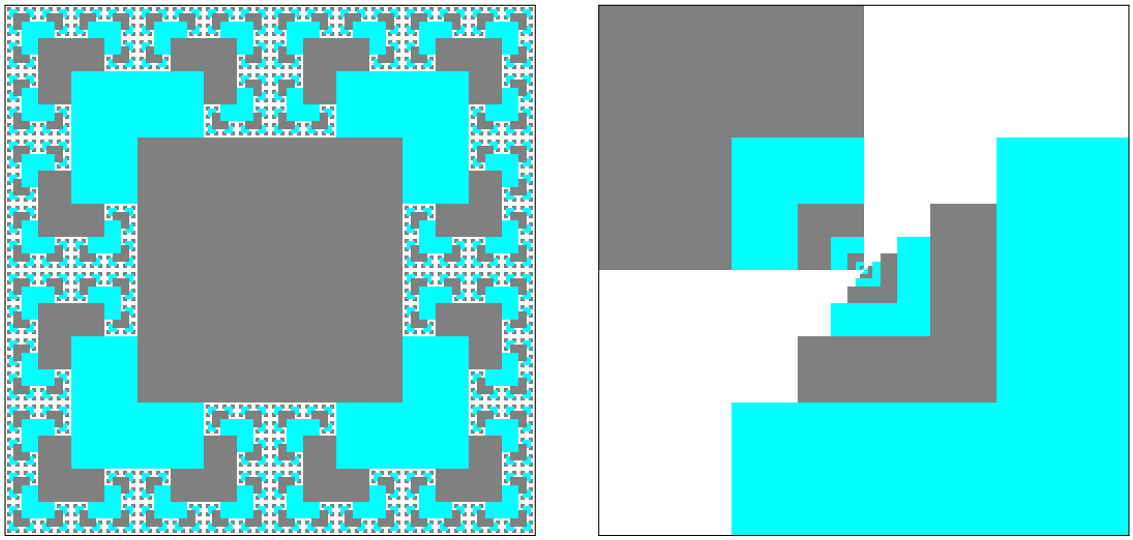
(!) Sie dürfen für die folgende rekursive Methode `drawPatternRecursively` keine globalen Variablen oder zusätzliche eigene Hilfsmethoden verwenden. Der vorgegebene Methodenkopf darf nicht erweitert oder geändert werden. Für die Implementierung der Methode `drawPatternRecursively` darf keine Schleife verwendet werden.

- Implementieren Sie die **rekursive** Methode `drawPatternRecursively`:

```
void drawPatternR...(CodeDraw myDrawObj, int x, int y, int s, boolean c)
```

Diese Methode zeichnet überlappende Quadrate. Die Methode hat die Parameter `x` und `y`, welche den Koordinaten der Quadratmittelpunkte entsprechen. Zusätzlich wird mit dem Parameter `s` die Seitenlänge der Quadrate angegeben. Mit diesen Parametern wird ein gefülltes Quadrat gezeichnet. Die Methode hat als vierten Parameter einen boolean-Wert `c`, der zur Farbsteuerung verwendet wird. Ist der Parameter `c == true`, dann wird das Quadrat *Gray* gezeichnet, bei `false` *Cyan*. Bei jeder Rekursionsstufe ändert sich die Farbe. Geben Sie acht, dass die größeren Quadrate die kleineren Quadrate überdecken und nicht umgekehrt.

Der Aufruf von `drawPatternRecursively(myDrawObj, 256, 256, 256, true)` erzeugt durch Selbstaufrufe der Methode `drawPatternRecursively` ein Quadratmuster, wie in Abbildung 1a dargestellt. Bei jedem rekursiven Aufruf wird der Mittelpunkt des nächsten Quadrates um die Länge `s/2` in x- und y-Richtung verschoben (in jede der vier Diagonalrichtungen). Die Seitenlänge `s` des Quadrats halbiert sich bei jedem Rekursionsschritt. Die Rekursion wird solange fortgeführt, solange die Seitenlänge `s` $\geq$ `4` ist.



(a) (b)

Abbildung 1: a) Rekursives Quadratmuster bestehend aus grauen und cyanfarbenen Quadraten. b) Abgeänderte Variante des Quadratmusters.

- Die zweite Methode ist eine **iterative** Methode mit dem Namen `drawPatternIteratively`:

```
void drawPatternIteratively(CodeDraw myDrawObj, int width)
```

und wurde unter Verwendung von ChatGPT 3.5[1][2] (**G**enerative **P**re-trained **T**ransformer[3]) erzeugt. Es wurde die Beschreibung der rekursiven Methode genommen und ChatGPT damit gefragt eine iterative Variante zu erstellen unter Angabe des Methodenrumpfes und Erklärung des Parameters `width`. Zusätzlich wurde der Hinweis angegeben, dass die Implementierung in JAVA erfolgen soll. Es wurde jener Code erzeugt, den Sie in der Angabe sehen. Es wurden nur die Zeichenbefehle gegen jene von CodeDraw ausgetauscht. Mit dem vorhandenen Code wird die Abbildung 1b erzeugt.

Beantworten Sie folgende Frage zur iterativen Version:

Wie muss der Code umgebaut werden, dass das richtige Ergebnis herauskommt? Implementieren Sie dazu eine korrekte iterative Variante in einer eigenen Methode.

[1] `https://openai.com/blog/chatgpt`
[2] `https://en.wikipedia.org/wiki/ChatGPT`
[3] `https://en.wikipedia.org/wiki/Generative_pre-trained_transformer`