Aufgabenblatt 3

Kompetenzstufe 1 & Kompetenzstufe 2

Allgemeine Informationen zum Aufgabenblatt:

- Die Abgabe erfolgt in TUWEL. Bitte laden Sie Ihr IntelliJ-Projekt bis spätestens **Mittwoch, 20.11.2024 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 in 1-2 Sätzen was der Code macht.

b) Bei der Erstellung wurde nicht auf eine sinnvolle Gliederung und Formatierung geachtet. Beschreiben Sie allgemein in 1-2 Sätzen was Sie am Code ändern würden und warum.

c) Identifizieren Sie Codebereiche, die in Methoden aufgeteilt werden können. Zum Beispiel Codeteile, die öfters vorhanden sind, können in Methoden ausgelagert werden. Auch Bereiche, die eine bessere Strukturierung ermöglichen, können in Methoden implementiert werden. Implementieren Sie entsprechende Methoden und rufen Sie diese in `main` auf, um die identische Ausgabe zu erhalten.

d) Der gegebene Code gibt ein Muster mit fester Breite aus. Berücksichtigen Sie bei Ihrer Implementierung mit Methoden eine Möglichkeit für die Änderung der Breite des Musters. Damit ändert sich automatisch auch die Höhe des Musters und der Ausgabe. Nehmen Sie an, dass die Breite nur positive gerade Werte größer gleich 4 annehmen kann.

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

[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 und machen Sie am Ende der Ausgabe einen Zeilenumbruch.
  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. Erst wird zwischen zwei Zeichen von `text` das erste Zeichen von `characters` abwechselnd zweimal bzw. einmal eingefügt und auf der Konsole in einer Zeile ausgegeben. Danach wird das zweite Zeichen von `characters` abwechselnd zweimal bzw. einmal zwischen den Zeichen von `text` eingefügt und auf der Konsole in einer weiteren Zeile ausgegeben, usw. 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", "!O(")` liefert
  `C!!W`
  `COOW`
  `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 und in einer Zeile auf der Konsole ausgegeben. Vermeiden Sie redundanten Code, indem Sie eine bereits implementierte Methode verwenden.
  Vorbedingung: `text != null`.

  Beispiel(e):
  `addChar("A")` liefert `A`
  `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** nebeneinander durch Leerzeichen getrennt auf der Konsole aus. Vorbedingung: `start` $\leq$ `end`.

  Beispiel:
  `printOddNumbersAscending(5, 14)` liefert 5 7 9 11 13

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

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

  Diese Methode gibt alle geraden Zahlen im Intervall von [0, `end`] **absteigend** nebeneinander durch Leerzeichen getrennt auf der Konsole aus.

  Vorbedingung: `end` ≥ 0.

  Beispiel:
  `printEvenNumbersDescending(11)` liefert 10 8 6 4 2 0

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

  ```
  int countCharChanges(String text)
  ```

  Diese Methode zählt alle Übergänge von unterschiedlichen Zeichen im String `text`, die direkt hintereinander vorkommen und gibt diese Anzahl zurück. Zum Beispiel hat der String `"ABBBC"` zwei Übergänge. Einen von `A` zu den drei `B`'s und dann von den `B`'s zu `C`.

  Vorbedingung: `text != null`

  Beispiele:
  `countCharChanges("A")` liefert 0
  `countCharChanges("AA")` liefert 0
  `countCharChanges("abBc")` liefert 3
  `countCharChanges("XYYYZZAAAB")` liefert 4
  `countCharChanges("satt")` liefert 2
  `countCharChanges("Schifffahrt")` liefert 8

Aufgabe 4 (1 Punkt)

Implementieren Sie folgende Aufgabenstellung:

(!) Sie dürfen bei dieser Methode keine Klassenvariablen oder zusätzliche eigene Hilfsmethoden verwenden. Der vorgegebene Methodenkopf darf nicht erweitert oder geändert werden. Für die Implementierung der Aufgabenstellung dürfen keine Schleifen oder Arrays verwendet werden. Sie dürfen für diese Aufgabe nur die Methoden `length()`, `isEmpty()`, `charAt()` und `substring()` der String-Klasse verwenden.

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

  ```
  boolean isStartAndEndSeq(String text, String sequence)
  ```

  Diese Methode gibt `true` zurück, falls der String `text` mit dem String `sequence` beginnt und endet (`sequence` muss mindestens zweimal in `text` vorkommen), ansonsten `false`. Zwischen der vorhandenen Sequenz am Beginn und am Ende können 0 bis beliebig viel Zeichen vorhanden sein.

  Vorbedingungen: `text != null`, `sequence != null`, `sequence.length() > 0`,

  Beispiele:
  `isStartAndEndSeq("", "1")` liefert `false`
  `isStartAndEndSeq("AA", "A")` liefert `true`
  `isStartAndEndSeq("ABBAB", "AB")` liefert `true`
  `isStartAndEndSeq("ABBBA", "AB")` liefert `false`
  `isStartAndEndSeq("ottootto", "otto")` liefert `true`
  `isStartAndEndSeq("otto", "otto")` liefert `false`
  `isStartAndEndSeq("ottotto", "otto")` liefert `false`
  `isStartAndEndSeq("ottoottt", "otto")` liefert `false`
  `isStartAndEndSeq("test1234test", "test")` liefert `true`
  `isStartAndEndSeq("NEN", "NEEN")` liefert `false`

Aufgabe 5 (1 Punkt)

Implementieren Sie folgende Aufgabenstellung:

(!) Sie dürfen bei dieser Methode keine Klassenvariablen oder zusätzliche eigene Hilfsmethoden verwenden. Der vorgegebene Methodenkopf darf nicht erweitert oder geändert werden. Für die Implementierung der Aufgabenstellung dürfen keine Schleifen oder Arrays verwendet werden. Sie dürfen für diese Aufgabe nur die Methoden `length()`, `isEmpty()`, `charAt()` und `substring()` der String-Klasse verwenden.

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

  ```
  String orderCharGroups(String text)
  ```

  Diese Methode erstellt einen neuen String, indem alle Zeichen des Strings `text` geordnet eingefügt werden. Es sind in `text` immer nur zwei unterschiedliche Zeichen vom Typ `char` vorhanden. Der neue String hat alle Vorkommen des einen Zeichens links und die Vorkommen des anderen Zeichens rechts.

  Sie können frei wählen, welches Zeichen in Ihrer Implementierung auf der linken Seite steht und welches auf der rechten Seite steht.

  Hinweis: Es gibt keine Ordnung der Zeichen. Sie können frei wählen, welches Zeichen in Ihrer Implementierung auf der linken Seite und welches Zeichen auf der rechten Seite steht. Das heißt, dass z.B. nicht `a` links und `b` rechts, oder `1` links und `2` rechts stehen muss (siehe Beispiele).

  Vorbedingung: `text != null` und `text` besteht nur aus maximal zwei unterschiedlichen Zeichen vom Typ `char`.

  Beispiele:
  `orderCharGroups("")` liefert `""`
  `orderCharGroups("1")` liefert `1`
  `orderCharGroups("12")` liefert `12`
  `orderCharGroups("1212")` liefert `1122`
  `orderCharGroups("abbaaababbaa")` liefert `aaaaaaabbbbb`
  `orderCharGroups("ABBA")` liefert `BBAA`
  `orderCharGroups("11221122")` liefert `22221111`
  `orderCharGroups("AAAAAA")` liefert `AAAAAA`

Aufgabe 6 (1 Punkt)

Implementieren Sie folgende Aufgabenstellung:

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

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

  ```
  void drawCirclesRec(CodeDraw myDrawObj, int y, int radius, int num)
  ```

  Diese Methode zeichnet ein rekursives Kreismuster. Die Methode hat den Parameter `y`, welcher der y-Koordinate des Kreismittelpunktes entspricht. Die x-Koordinate des Kreismittelpunktes ist konstant auf die Hälfte der Fensterbreite gesetzt. Mit dem dritten Parameter `radius` wird der Radius der Kreise in der aktuellen Rekursionsstufe angegeben. Zusätzlich gibt es noch den Parameter `num`, der die Rekursionsstufe (Tiefe) angibt. Dieser Parameter wird bei jedem Rekursionsschritt um 1 dekrementiert.

  Der Aufruf von `drawCirclesRec(myDrawObj, pixelHeight / 2, pixelWidth / 2, 5)` erzeugt durch Selbstaufrufe der Methode `drawCirclesRec` ein Muster, wie in Abbildung 1 dargestellt. Die Fenstergröße wurde für dieses Beispiel auf 512×512 Pixel gesetzt. Bei jedem rekursiven Aufruf nach oben und unten wird der Mittelpunkt des nächsten Kreises um die Länge `radius/2` in y-Richtung verschoben. Zusätzlich wird bei jedem Rekursionsschritt der Radius `radius` der Kreise halbiert und die Variable `num` um 1 dekrementiert. Die Rekursion wird fortgeführt, solange `num > 0` gegeben ist. Der Aufruf `myDrawObj.setColor(new Color(255 - num * 30, 255 - num * 30 , 0));` ändert die Farbgebung in jeder Rekursionsstufe.

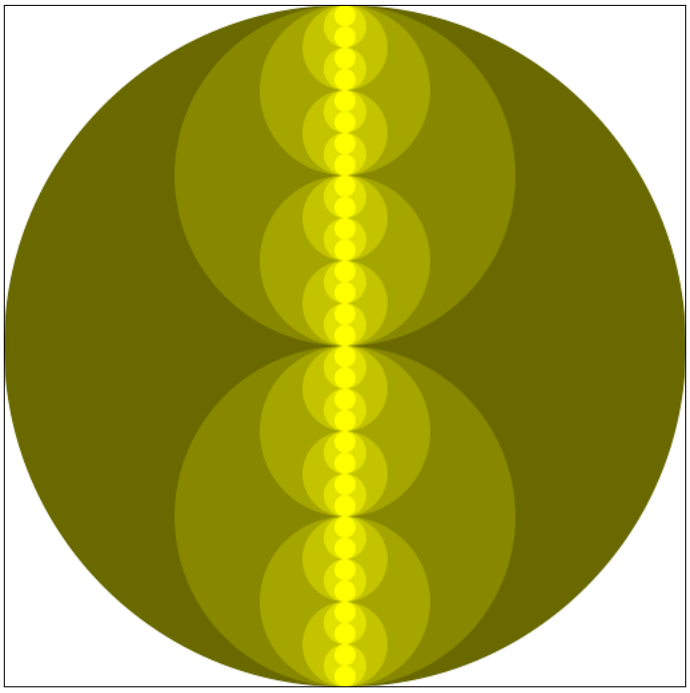


Abbildung 1: Rekursives Kreismuster.

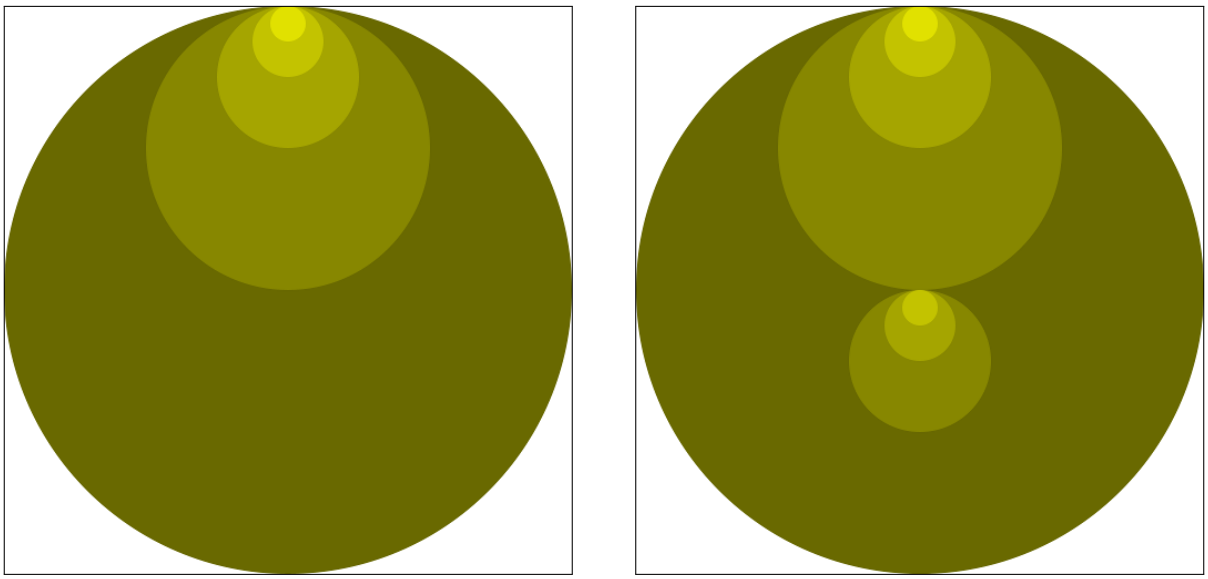
Aufgabe 7 (1 Punkt)

Implementieren Sie folgende Aufgabenstellung:

- Implementieren Sie die **iterative** Methode `drawCirclesIterHand`:

  ```
  void drawCirclesIterHand(CodeDraw myDrawObj, int y, int radius, int num)
  ```

  Diese Methode soll das gleiche Ergebnis liefern wie die rekursive Methode in Aufgabe 6. Wir haben unter Verwendung von ChatGPT [1][2] (**G**enerative **P**re-trained **T**ransformer[3]) eine iterative Variante erzeugen lassen. In Abbildung 2a ist das Ergebnis bei Verwendung von Version 3.5 zu sehen und in Abbildung 2b das Ergebnis bei Verwendung der Version 4.0. Es wurde beide Male die Beschreibung der rekursiven Methode genommen und *ChatGPT* damit gefragt, eine iterative Variante zu erstellen. Zusätzlich wurde der Hinweis angegeben, dass die Implementierung in JAVA, unter der Zuhilfenahme der Bibliothek *CodeDraw*, erfolgen soll. Anschließend wurde noch ergänzt, dass die Methodenköpfe nicht verändert werden dürfen und auch die Verwendung von Arrays nicht erlaubt ist.



(a) (b)

Abbildung 2: Iteratives Kreismuster mit a) *ChatGPT* 3.5 und b) mit *ChatGPT* 4.0 generiert.

Analysieren Sie die von **ChatGPT 3.5** erstellte Version (`drawCirclesIter35(...)`) und überlegen Sie, was Sie umbauen müssen, um eine korrekte Version zu erhalten. Schreiben Sie eine richtige und korrekte iterative Methode (`drawCirclesIterHand()`), die auf einer ChatGPT Version basiert und das gezeigte Ergebnis in Abbildung 1 liefert.

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