Aufgabenblatt 3

Kompetenzstufe 1 & Kompetenzstufe 2

Allgemeine Informationen zum Aufgabenblatt:

- Die Abgabe erfolgt in TUWEL. Bitte laden Sie Ihr IntelliJ-Projekt bis spätestens **Donnerstag, 01.12.2022 20:00 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. Zusätzlich soll über einen Parameter die Breite der Ausgabe steuerbar sein.

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

Aufgabe 2 (1 Punkt)

Implementieren Sie folgende Aufgabenstellung:

- Implementieren Sie eine Methode `addSign`:

  ```
  void addSign(String text, char sign)
  ```

  Diese Methode gibt einen String `text` formatiert aus. Es wird zwischen den ersten zwei Zeichen von `text` das Zeichen `sign` einmal eingefügt, zwischen zweiten und dritten Zeichen von `text` zweimal, usw. Geben Sie den veränderten String in einer Zeile auf der Konsole aus. Vorbedingung: `text != null`.

  Beispiel(e):
  `addSign("", '+')` liefert keine Ausgabe
  `addSign("IT", '-')` liefert `I-T`
  `addSign("Hello!", '#')` liefert `H#e##l###l####o#####!`
  `addSign("+EP1+", '&')` liefert `+&E&&P&&&1&&&&+`
  `addSign("INT", '*')` liefert `I*N**T`

- Implementieren Sie eine Methode `addSign`:

  ```
  void addSign(int number, char sign)
  ```

  Diese Methode gibt eine Zahl `number` formatiert aus. Es wird zwischen den ersten zwei Ziffern von `number` einmal das Zeichen `sign` eingefügt, zwischen der zweiten und dritten Ziffer von `number` zweimal, usw. 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):
  `addSign(1, '$')` liefert `1`
  `addSign(42, '%')` liefert `4%2`
  `addSign(183, '.')` liefert `1.8..3`
  `addSign(4096, ':')` liefert `4:0::9:::6`
  `addSign(65536, ']')` liefert `6]5]]5]]]3]]]]6`

- Implementieren Sie eine Methode `addSign`:

  ```
  void addSign(String text, String signs)
  ```

  Diese Methode gibt einen String `text` unterschiedlich formatiert aus. Es wird jeweils zwischen ersten und zweiten Zeichen von `text` einmal das erste Zeichen vom String `signs` eingefügt, zwischen zweiten und dritten Zeichen von `text` zweimal das erste Zeichen vom String `signs`, usw. Der resultierende String wird in einer Zeile auf der Konsole ausgegeben. Dies soll für jedes Zeichen aus dem String `signs` 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 `signs != null`.

  Beispiel(e):
  `addSign("IT", "/X/")` liefert
  ```
  I/T
  IXT
  I/T
  ```
  `addSign("Hello!", "(#?§")` liefert
  ```
  H(e((l(((l((((o(((((!
  H#e##l###l####o#####!
  H?e??l???l????o?????!
  H§e§§l§§§l§§§§o§§§§§!
  ```

- Implementieren Sie eine Methode `addSign`:

  ```
  void addSign(String text)
  ```

  Diese Methode gibt einen String `text` formatiert aus. Es wird zwischen den ersten zwei Zeichen von `text` das Zeichen `=` einmal eingefügt, zwischen zweiten und dritten Zeichen von `text` zweimal, usw. 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):
  `addSign("")` liefert keine Ausgabe
  `addSign("IT")` liefert `I=T`
  `addSign("Hello!")` liefert `H=e==l===l====o=====!`

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 `printEvenNumbersAscending`:

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

  Diese Methode gibt alle geraden Zahlen im Intervall von [`start`, `end`] **aufsteigend** aus.
  Vorbedingung: `start` $\leq$ `end`.

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

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

  Diese Methode gibt alle ungeraden Zahlen im Intervall von [`start`, `end`] **absteigend** aus.
  Vorbedingung: `start` $\leq$ `end`.

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

  ```
  int sumSquaredDigits(int number)
  ```

  Diese Methode quadriert alle Ziffern einer Zahl `number`, summiert diese quadrierten Ziffern auf und gibt die Summe zurück.
  Vorbedingung: `number` > 0.

  Beispiele:
  `sumSquaredDigits(1)` liefert 1
  `sumSquaredDigits(102)` liefert 5
  `sumSquaredDigits(1234)` liefert 30
  `sumSquaredDigits(10000)` liefert 1
  `sumSquaredDigits(93842)` liefert 174
  `sumSquaredDigits(875943789)` liefert 438

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

  `String removeSpaces(String text)`

  Diese Methode entfernt alle Leerzeichen im String `text`. Anschließend wird der neu entstandene String zurückgegeben.
  Vorbedingung: `text != null`.

  Beispiele:
  `removeSpaces(" ")` liefert `""`
  `removeSpaces("+ +")` liefert `"++"`
  `removeSpaces(" 12 3 45 ")` liefert `"12345"`
  `removeSpaces("a  b   c    d"")` liefert `"abcd"`

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 `countNOrderedPairs`:

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

  Diese Methode überprüft, wie viele Paare von aufeinander folgenden Kleinbuchstaben in `text` vorkommen, die richtig geordnet sind. Ein Paar gilt jeweils als richtig geordnet, wenn der erste Eintrag kleiner gleich dem zweiten Eintrag ist. Die Paare im String dürfen sich auch überlappen (z.B. enthält ein String `"adm"` zwei geordnete Paare, weil `a <= d` und `d <= m` ist). Die Zählung beginnt bei `index` (inklusive).
  Vorbedingungen: `text != null`, `text.length() > 0` und `index` beschreibt einen gültigen Index von `text`.

  Beispiele:
  `countNOrderedPairs("bhhebegcmoqast", 0)` liefert 9
  `countNOrderedPairs("bhhebegcmoqast", 2)` liefert 7
  `countNOrderedPairs("bhhebegcmoqast", 5)` liefert 6
  `countNOrderedPairs("bhhebegcmoqast", 6)` liefert 5
  `countNOrderedPairs("bhhebegcmoqast", 8)` liefert 4

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

  ```
  String shiftAllSignsRight(String text, char sign)
  ```

  Diese Methode verschiebt alle Zeichen `sign` innerhalb des Strings `text` an die letzte Position im String. Alle Zeichen hinter dem entnommenen und an die letzte Stelle verschobenen Zeichen rücken eine Indexposition vor, um die so entstandene Lücke zu schließen (siehe Beispiele). Das Ergebnis wird als neuer String zurückgeliefert.
  Vorbedingungen: `text != null`.

  Beispiele:
  `shiftAllSignsRight("az3kj", 'z')` liefert `"a3kjz"`
  `shiftAllSignsRight("kjdn{nd8xngs+d#k", 'n')` liefert `"kjd{d8xgs+d#knnn"`
  `shiftAllSignsRight("", 'e')` liefert `""`
  `shiftAllSignsRight("4", '4')` liefert `"4"`
  `shiftAllSignsRight("ji)o3ie6pk(2i",'i')` liefert `"j)o3e6pk(2iii"`
  `shiftAllSignsRight("nothing", 'x')` liefert `"nothing"`

Aufgabe 5 (2 Punkte)

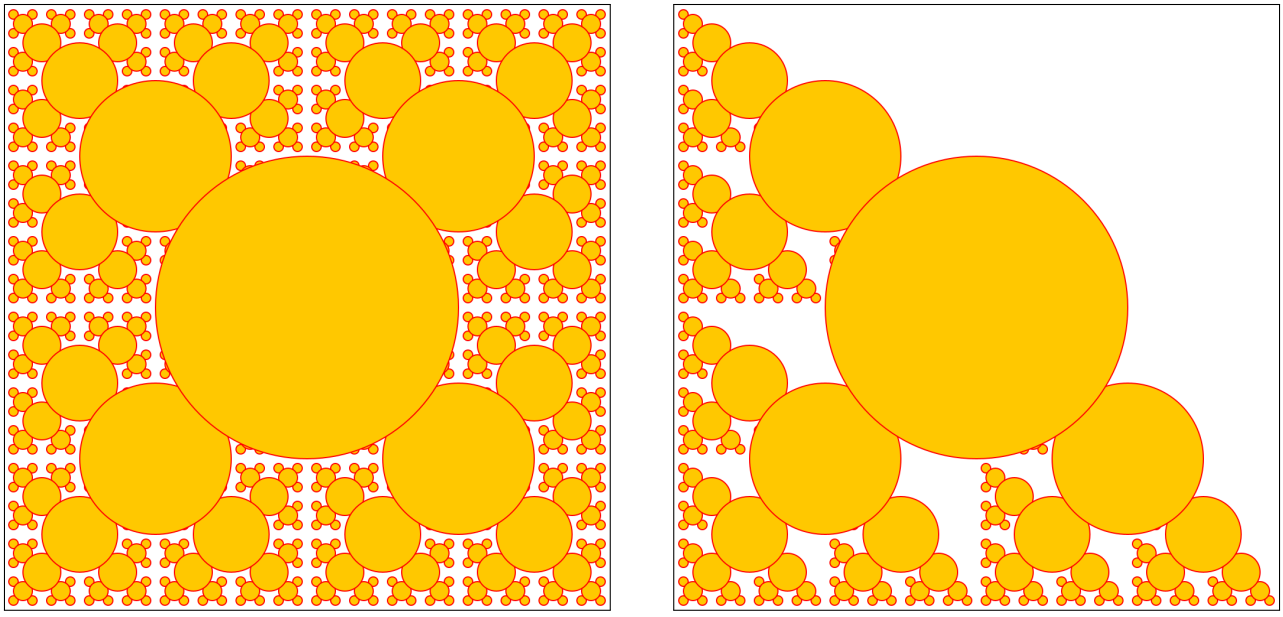
Implementieren Sie folgende Aufgabenstellung:

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

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

```
void drawCirclePatternRecursively(CodeDraw myDrawObj, int x, int y, int r)
```

Diese Methode zeichnet orange Kreise mit einer roten Umrandung. Der Methode werden die Koordinaten `x` und `y` der Kreismittelpunkte übergeben. Zusätzlich wird mit dem Parameter `r` der Radius des Kreises festgelegt. Mit diesen Parametern wird ein Kreis gezeichnet. Der Aufruf von `drawCirclePatternRecursively(myDrawObjR, 256, 256, 128)` erzeugt durch Selbstaufrufe der Methode `drawCirclePatternRecursively` ein Kreismuster, wie in Abbildung 1a dargestellt. Bei jedem rekursiven Aufruf wird der Mittelpunkt des nächsten Kreises um die Länge `r` in x- und y-Richtung verschoben (in jede der vier Diagonalrichtungen). Der Radius des Kreises halbiert sich bei jedem Rekursionsschritt. Die Rekursion wird fortgeführt, solange der Radius `r > 2` ist.



(a) (b)

Abbildung 1: a) Rekursives Kreismuster bestehend aus orangen Kreisen mit rotem Rand. b) Abgeänderte Variante des Kreismusters.

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

  ```
  void drawCirclePatternIteratively(CodeDraw myDrawObj, int maxRadius)
  ```

  Diese Methode zeichnet ebenfalls orange Kreise mit einer roten Umrandung wie zuvor für die Methode `drawCirclePatternRecursively` beschrieben. Der Unterschied ist, dass die Methode iterativ implementiert werden muss (**keine rekursiven Aufrufe**). Die Methode hat einen Parameter `maxRadius`, der den Radius des größten Kreises beschreibt. Durch die Angabe von `maxRadius` und die Bedingung, dass der Radius für die kleinsten Kreise größer 2 sein muss, ergeben sich die verschiedenen Radien der Kreise, die gezeichnet werden müssen. Alle anderen Angaben können aus der vorherigen Beschreibung übernommen werden. Der Aufruf `drawCirclePatternIteratively(myDrawObjI, 128)` führt zur Ausgabe in Abbildung 1a.

- Hinweis: Setzen Sie die Fenstergröße auf 512×512 Pixel, um mit dem kleinsten Radius von 4 Pixel das Muster in Abbildung 1a zu erhalten.

Zusatzfrage(n):

1. Wie oft wird die Methode `drawCirclePatternRecursively` aufgerufen, wenn die Rekursion bis zu einem Radius `r > 2` aufgerufen wird?

2. Wie viele Kreise werden auf der letzten Rekursionsstufe (die kleinsten Kreise) gezeichnet?

3. Wie müssen Sie Ihre rekursive Implementierung abändern, um das Muster in Abbildung 1b zu erzeugen?