Aufgabenblatt 3

Kompetenzstufe 1 & Kompetenzstufe 2

Allgemeine Informationen zum Aufgabenblatt:

- Die Abgabe erfolgt in TUWEL. Bitte laden Sie Ihr IntelliJ-Projekt bis spätestens **Donnerstag, 25.11.2021 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 und beschreiben Sie dessen Funktionsweise in wenigen Sätzen. Erklären Sie, was die einzelnen Methoden machen und schreiben Sie Ihre Erklärung als Kommentare dazu.

b) Bei der Erstellung wurde nicht auf eine sinnvolle Namensgebung bei den Methoden und Variablen geachtet. Ändern Sie dazu bitte Methoden- und Variablennamen so ab, dass diese die Funktionalität der jeweiligen Methoden bzw. Variablen widerspiegeln. Formatieren Sie zusätzlich 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 dazu die bereits vorhandene Methode `IhrMethodenname(...)` verwenden. Geben Sie der Methode einen Namen, der der Funktion des Codes entspricht.

Aufgabe 2 (1 Punkt)

Implementieren Sie folgende Aufgabenstellung:

- Implementieren Sie eine Methode `addSequence`:

  ```
  void addSequence(String text, char symbol)
  ```

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

  Beispiel(e):
  `addSequence("", '&')` liefert keine Ausgabe
  `addSequence("TU", '#')` liefert `T###U`
  `addSequence("Hello!", '-')` liefert `H---e---l---l---o---!`
  `addSequence("-EP1-", '$')` liefert `-$$$E$$$P$$$1$$$-`
  `addSequence("JAVA", '!')` liefert `J!!!A!!!V!!!A`

- Implementieren Sie eine Methode `addSequence`:

  ```
  void addSequence(int number, char symbol)
  ```

  Diese Methode gibt eine Zahl `number` formatiert aus. Es wird jeweils zwischen zwei Ziffern von `number` dreimal das Zeichen `symbol` 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):
  `addSequence(1, '%')` liefert `1`
  `addSequence(42, '*')` liefert `4***2`
  `addSequence(163, ':')` liefert `1:::6:::3`
  `addSequence(2048, '#')` liefert `2###0###4###8`
  `addSequence(42815, '(')` liefert `4(((2(((8(((1(((5`

- Implementieren Sie eine Methode `addSequence`:

  ```
  void addSequence(String text, String symbols)
  ```

  Diese Methode gibt einen String `text` unterschiedlich formatiert aus. Es wird jeweils zwischen jedem Zeichen von `text` dreimal ein Zeichen vom String `symbols` eingefügt. Der resultierende String wird in einer Zeile auf der Konsole ausgegeben. Dies soll für jedes Zeichen aus dem String `symbols` 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 `symbols != null`.

  Beispiel(e):
  `addSequence("TU", "(0)")` liefert
  ```
  T(((U
  T000U
  T)))U
  ```
  `addSequence("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 `addSequence`:

  ```
  void addSequence(String text)
  ```

  Diese Methode gibt einen String `text` formatiert aus. Es wird zwischen jedem Zeichen von `text` dreimal das Zeichen `+` 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):
  `addSequence("")` liefert keine Ausgabe
  `addSequence("TU")` liefert `T+++U`
  `addSequence("Hello!")` liefert `H+++e+++l+++l+++o+++!`

Aufgabe 3 (1 Punkt)

Implementieren Sie folgende 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.
  Vorbedingungen: `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.
  Vorbedingungen: `start` $\leq$ `end` und `divider` > 0.

- 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 `removeMultipleChars`:

  ```
  String removeMultipleChars(String text)
  ```

  Diese Methode entfernt Zeichen im String `text`. Kommt ein Zeichen mehrfach direkt hintereinander vor, dann wird diese Sequenz auf ein Zeichen reduziert. Das kann mehrmals im String vorkommen. Anschließend wird der neu entstandene String zurückgegeben.
  Vorbedingung: `text != null`.

  Beispiele:
  `removeMultipleChars("AA")` liefert `"A"`
  `removeMultipleChars("ABCCDE")` liefert `"ABCDE"`
  `removeMultipleChars("SEENDEEN")` liefert `"SENDEN"`
  `removeMultipleChars("ABCFFFE 14448")` liefert `"ABCFE 148"`

Aufgabe 4 (1 Punkt)

Implementieren Sie folgende 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 oder Arrays verwendet werden.

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

  ```
  boolean isStringPalindrome(String text)
  ```

  Diese Methode überprüft, ob es sich bei einem String `text` um ein *Palindrom* handelt. Das heißt, dass die Methode überprüft, ob von vorne und hinten beginnend bis zur Mitte hin die gleichen Zeichen im String `text` vorkommen.
  Vorbedingungen: `text != null` und `text.length() > 0`.

  Beispiele:
  `isStringPalindrome("lagerregal")` liefert `true`
  `isStringPalindrome("otto")` liefert `true`
  `isStringPalindrome("rentner")` liefert `true`
  `isStringPalindrome("abcba")` liefert `true`
  `isStringPalindrome("test")` liefert `false`
  `isStringPalindrome("teppichstaubsauger")` liefert `false`

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

  ```
  String shiftTs(String text)
  ```

  Diese Methode verschiebt alle Vorkommen des Buchstabens `'T'` innerhalb des Strings `text` an die erste Stelle. Das Ergebnis wird als neuer String zurückgeliefert.
  Vorbedingungen: `text != null`. Beispiele:
  `shiftTs("")` liefert `""`
  `shiftTs("T")` liefert `"T"`
  `shiftTs("BT")` liefert `"TB"`
  `shiftTs("TBT")` liefert `"TTB"`
  `shiftTs("TBCTDFTTGHTTT")` liefert `"TTTTTTTBCDFGH"`
  `shiftTs("TTHVVGWTTBSJTTT")` liefert `"TTTTTTTHVVGWBSJ"`
  `shiftTs("ZMNGFBTTTTTTTTT")` liefert `"TTTTTTTTTZMNGFB"`

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 drawPatternRecursively(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 *Orange* gezeichnet, bei `false` *Blue*. Bei jeder Rekursionsstufe ändert sich die Farbe. Geben Sie acht, dass die größeren Quadrate die kleineren Quadrate überdecken und nicht umgekehrt.

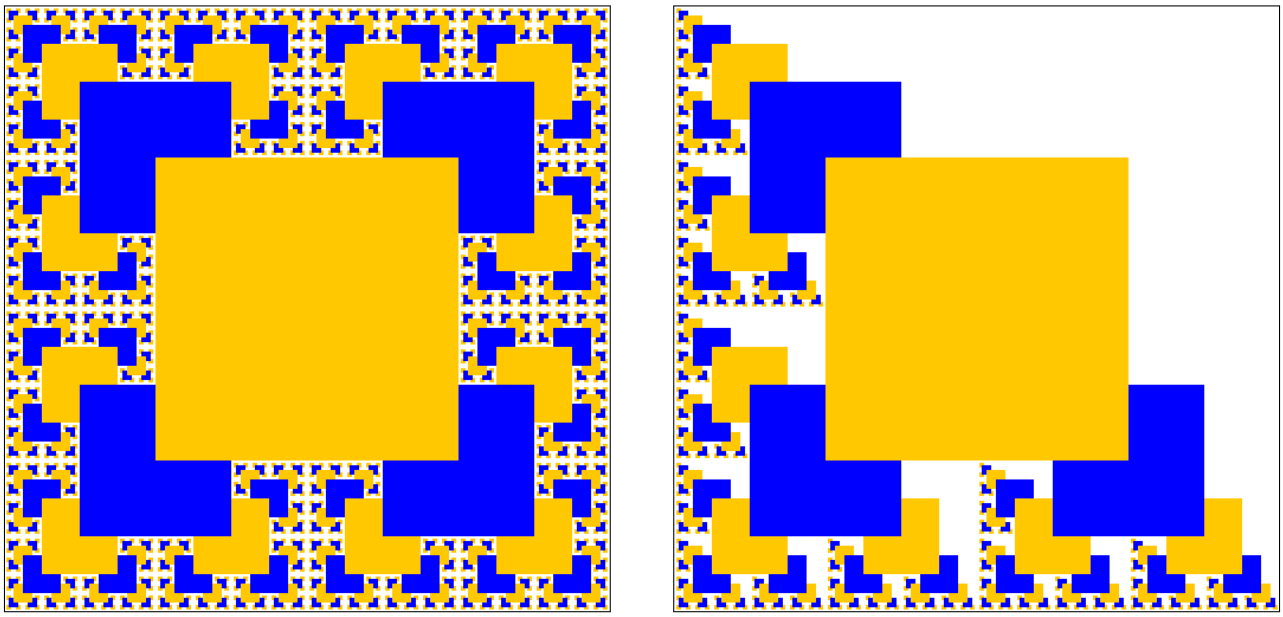
  Der Aufruf von `drawPatternRecursively(myDrawObj, 256, 256, 256, true)` erzeugt durch Selbstaufrufe der Methode `drawPatternRecursively` ein Quadratmuster, wie in Abbildung 1a dargestellt. Bei jedem rekursiven Aufruf wird der Mittelpunkt des nächsten Quadrates um die Länge `s/2` in x- und y-Richtung 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.

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

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

  Diese Methode zeichnet ebenfalls Quadrate wie zuvor für die Methode `drawPatternRecursively` beschrieben. Der Unterschied ist, dass die Methode iterativ implementiert werden muss (**keine rekursiven Aufrufe**). Die Methode hat einen Parameter `width`, der die Größe des Ausgabefensters beschreibt. Durch die Fenstergröße und die kleinste Seitenlänge der Quadrate von 4 ergeben sich die verschiedenen Größenstufen der Quadrate, die gezeichnet werden müssen. Alle anderen Angaben können aus der vorherigen Beschreibung übernommen werden. Der Aufruf `drawPatternIteratively(myDrawObj, 512)` führt zur Ausgabe in Abbildung 1a.

- Hinweis: Setzen Sie die Fenstergröße auf 512×512 Pixel, um bei einer minimalen Seitenlänge der Quadrate von 4 Pixel das Muster in Abbildung 1a zu erhalten.



(a) (b)

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

Zusatzfrage(n):

1. Wie oft wird die Methode `drawPatternRecursively` aufgerufen, wenn die Rekursion bis zu einer Quadratseitenlänge von 4 aufgerufen wird?

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

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