Aufgabenblatt 3

Kompetenzstufe 1 & Kompetenzstufe 2

Allgemeine Informationen zum Aufgabenblatt:

- Die Abgabe erfolgt in TUWEL. Bitte laden Sie Ihr IntelliJ-Projekt bis spätestens **Freitag, 27.11.2020 15:00 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 `StdDraw` und oder Klassen, die in den Hinweisen zu den einzelnen Aufgaben aufscheinen.
- 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 StdDraw
- Vergleich von rekursiver und iterativer Implementierung

Aufgabe 1 (1 Punkt)

Aufgabenstellung:

a) Analysieren Sie den gegebenen Code und beschreiben Sie dessen Funktionsweise. 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 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 die anderen Methoden auskommentieren.

Aufgabe 2 (1 Punkt)

Implementieren Sie folgende Aufgabenstellung:

- Implementieren Sie eine Methode `addSeparator`:

  ```
  void addSeparator(String text, char separator)
  ```

  Diese Methode gibt einen String `text` formatiert aus. Es wird zwischen zwei Zeichen von `text` das Zeichen `separator` eingefügt. Zum Schluss geben Sie den veränderten String mittels `System.out.println()` auf der Konsole aus.
  Vorbedingung: `text != null`.

  Beispiel(e):
  `addSeparator("A", '?')` liefert `A`
  `addSeparator("AB", ',')` liefert `A,B`
  `addSeparator("Hello!", ':')` liefert `H:e:l:l:o:!`
  `addSeparator("-Java-", '-')` liefert `--J-a-v-a--`
  `addSeparator(" TEST ", '+')` liefert `+T+E+S+T+`

- Implementieren Sie eine Methode `addSeparator`:

  ```
  void addSeparator(int number, char separator)
  ```

  Diese Methode gibt eine Zahl `number` formatiert aus. Es wird zwischen zwei Ziffern von `number` das Zeichen `separator` eingefügt. Der resultierende String wird mittels `System.out.println()` auf der Konsole ausgegeben. Für die Realisierung der Methode darf die Zahl in einen String umgewandelt werden (`Integer.toString(...)`) und soll danach die bereits implementierte Methode verwenden.
  Vorbedingung: `number` $>$ `0`.

  Beispiel(e):
  `addSeparator(1, '$')` liefert `1`
  `addSeparator(35, '*')` liefert `3*5`
  `addSeparator(657, ':')` liefert `6:5:7`
  `addSeparator(2048, '#')` liefert `2#0#4#8`
  `addSeparator(26348, '+')` liefert `2+6+3+4+8`

- Implementieren Sie eine Methode `addSeparator`:

  ```
  void addSeparator(String text, String separators)
  ```

  Diese Methode gibt einen String `text` unterschiedlich formatiert aus. Es wird zwischen jedem Zeichen von `text` ein Zeichen vom String `separators` eingefügt. Der resultierende String wird mittels `System.out.println()` auf der Konsole ausgegeben. Dies soll für jedes Zeichen aus dem String `separators` geschehen, d.h. der String `text` wird mit verschiedenen Zeichen formatiert mehrmals ausgegeben. Verwenden Sie dazu bereits vorhandene Methoden und

vermeiden Sie die Duplizierung von Code.
Vorbedingungen: `text != null` und `separators != null`.

Beispiel(e):
`addSeparator("AB", "+#$")` liefert
`A+B`
`A#B`
`A$B`
`addSeparator("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 `addSeparator`:

  `void addSeparator(String text)`

  Diese Methode gibt einen String `text` formatiert aus. Es wird zwischen jedem Zeichen von `text` das Zeichen `$` eingefügt. Zum Schluss geben Sie den veränderten String mittels `System.out.println()` auf der Konsole aus. Verwenden Sie dazu die bereits implementierten Methoden.
  Vorbedingung: `text != null`.

  Beispiel(e):
  `addSeparator("A")` liefert `A`
  `addSeparator("AB")` liefert `A$B`
  `addSeparator("Hello!")` liefert `H$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 `calcCrossSum`:

  ```
  int calcCrossSum(int number)
  ```

  Diese Methode berechnet die Summe aller Ziffern der Zahl `number` und gibt diese zurück.
  Vorbedingung: `number` > 0.

  Beispiele:
  `calcCrossSum(1)` liefert 1
  `calcCrossSum(102)` liefert 3
  `calcCrossSum(1234)` liefert 10
  `calcCrossSum(10000)` liefert 1
  `calcCrossSum(93842)` liefert 26
  `calcCrossSum(875943789)` liefert 60

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

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

  Diese Methode dupliziert alle Vorkommen von `letter` im String `text`. Der neu entstandene String wird zurückgegeben.
  Vorbedingung: `text != null`.

  Beispiele:
  `duplicateLetterInString("hallo", 'a')` liefert `haallo`
  `duplicate...String("Es ist die Erde", 'e')` liefert `Es ist diee Erdee`
  `duplicateLetterInString("3HALLO4", 'L')` liefert `3HALLLLO4`
  `duplicateLetterInString("a1b2c3d4e5", 'g')` liefert `a1b2c3d4e5`

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 `countCharsSmaller`:

  ```
  int countCharsSmaller(String text, char value)
  ```

  Diese Methode zählt alle Zeichen des Strings `text`, dessen ASCII-Wert als Dezimalzahl kleiner als der Wert von `value` ist. Die Anzahl wird als Ergebnis zurückgeliefert.
  Vorbedingungen: `text != null` und `text.length() > 0`.

  Beispiele:
  `countCharsSmaller("DAS (ist) ]ein] Test!", (char)100)` liefert 12
  `countCharsSmaller("a!", (char)200)` liefert 2
  `countCharsSmaller("Ein Test", (char)100)` liefert 3

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

  ```
  String removeCharsInString(String text, char start, char end)
  ```

  Diese Methode entfernt alle Zeichen im String `text`, die einen ASCII-Wert größer `start` und kleiner `end` haben und gibt anschließend den dadurch neu entstandenen String zurück.
  Vorbedingungen: `text != null`, `text.length() > 0` und `start` $\leq$ `end`.

  Beispiele:
  `removeCharsInString("testtrompete", 'd', 'n')` liefert `tsttropt`
  `removeCharsInString("test", 's', 'u')` liefert `es`
  `removeCharsInString("t", 't', 't')` liefert `t`
  `removeCharsInString("angabe", (char)97, (char)122))` liefert `aa`

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

  ```
  String shiftDigitRight(String text)
  ```

  Diese Methode verschiebt eine Ziffer innerhalb des Strings `text` an die letzte Stelle. Alle Zeichen von der ursprünglichen Position der Ziffer bis zum letzten Index werden dabei um eine Position in Richtung kleinerem Index verschoben. Das Ergebnis wird als neuer String zurückgeliefert.
  Vorbedingungen: `text != null` und es kommt im gesamten String höchstens eine Ziffer vor.

  Beispiele:
  `shiftDigitRight("az3kj")` liefert `azkj3`
  `shiftDigitRight("kjdn{nd8xngs+d#k")` liefert `kjdn{ndxngs+d#k8`
  `shiftDigitRight("")` liefert `""`
  `shiftDigitRight("4")` liefert `4`
  `shiftDigitRight("ji)oiepk(2")` liefert `ji)oiepk(2`
  `shiftDigitRight("ohneziffer")` liefert `ohneziffer`

Aufgabe 5 (2 Punkte)

Implementieren Sie folgende Aufgabenstellung:

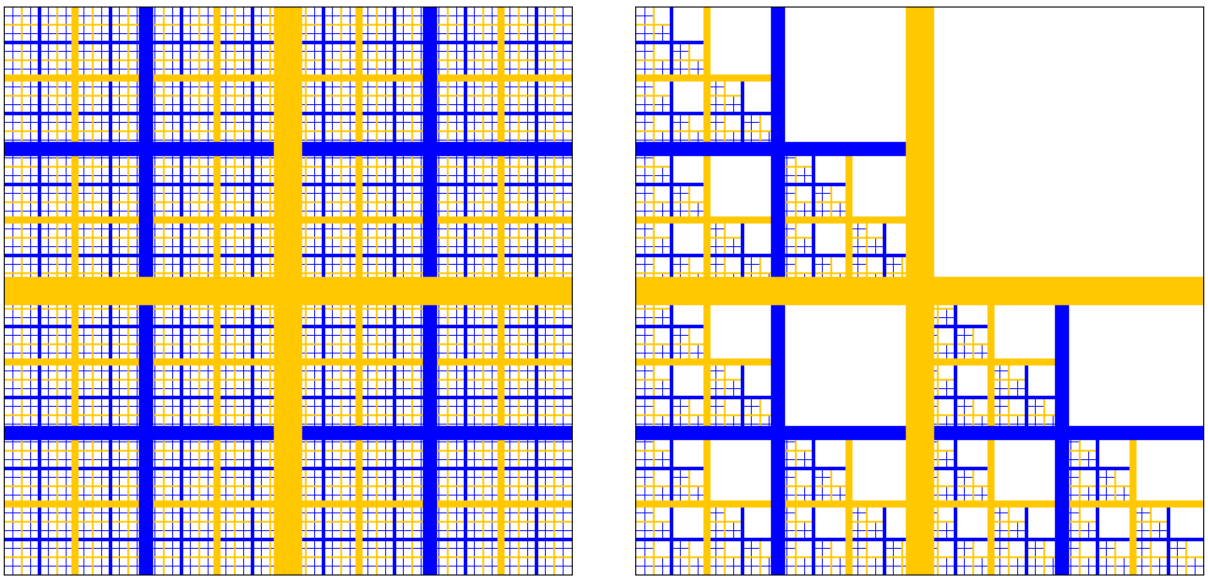
- Implementieren Sie die **rekursive** Methode `drawPatternRecursive`:

```
void drawPatternRecursive(int x, int y, int l, boolean c)
```

Diese Methode zeichnet Kreuze bestehend aus horizontalen und vertikalen Rechtecken. Die Methode hat die Parameter `x` und `y`, welche den Koordinaten der Rechteckmittelpunkte entsprechen. Zusätzlich wird mit dem Parameter `l` die Länge (längere Seite) eines Rechtecks festgelegt. Mit diesen Parametern werden zwei gefüllte Rechtecke gezeichnet, sodass ein Kreuz entsteht. Die Breite (kürzere Seite) eines Rechtecks entspricht immer 5% der Länge. Die Methode hat als vierten Parameter einen boolean-Wert `c`, der zur Farbsteuerung verwendet wird. Ist der Parameter `c == true`, dann wird das Rechteck (Kreuz) *orange* gezeichnet, bei `false` *blau*. Bei jeder Rekursionsstufe ändert sich die Farbe.

Der Aufruf von `drawPatternRecursive(0, 0, 512, true)` erzeugt durch Selbstaufrufe der Methode `drawPatternRecursive` ein Kreuzmuster, wie in Abbildung 1a dargestellt. Bei jedem rekursiven Aufruf wird der Mittelpunkt des nächsten Rechteckkreuzes um die Länge `l/4` in x- und y-Richtung verschoben (in jede der vier Diagonalrichtungen). Die Länge `l` des Rechtecks halbiert sich bei jedem Rekursionsschritt. Bei einer Auflösung von `l<16` Pixel soll das Zeichnen beendet werden.

Sie dürfen für die zu implementierende Methode keine globalen Variablen oder zusätzliche eigene Hilfsmethoden verwenden. Der vorgegebene Methodenkopf darf nicht erweitert oder geändert werden. Für die Implementierung der Methode darf keine Schleife verwendet werden.



(a) (b)

Abbildung 1: a) Rekursives Kreuzmuster bestehend aus orangen und blauen Rechtecken. b) Abgeänderte Variante des Kreuzmusters.

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

  ```
  void drawPatternIterative(int width)
  ```

  Diese Methode zeichnet ebenfalls Kreuze bestehend aus horizontalen und vertikalen Rechtecken wie zuvor für die Methode `drawPatternRecursive` beschrieben. Der Unterschied ist, dass die Methode iterativ implementiert werden muss (**keine rekursiven Aufrufe**). Die Methode hat einen Parameter `width`, der die Länge des größten Rechtecks in der Mitte angibt. Bei einer Rechtecklänge kleiner 16 Pixel soll das Zeichnen beendet werden. Alle anderen Angaben (wie sich die Rechtecke verkleinern) können aus der vorherigen Beschreibung übernommen werden. Der Aufruf `drawPatternIterative(512)` führt zur Ausgabe in Abbildung 1a.

(!) Setzen Sie die Fenstergröße auf 512×512 Pixel, um bei einer Auflösungsgrenze von `l<16` Pixel das Muster in Abbildung 1a zu erhalten. Durch Verwendung von `StdDraw.setXscale(...)` und `StdDraw.setYscale(...)` können Sie den Ursprung des StdDraw-Fensters z.B. in die Mitte der Zeichenfläche verschieben. Für eine schnellere Anzeige der Grafik verwenden Sie *DoubleBuffering*[1].

Zusatzfrage(n):

1. Wie oft wird die Methode `drawPatternRecursive` aufgerufen, wenn als Abbruchbedingung die Auflösungsgrenze von `l<16` gewählt wird?
2. Wie viele Kreuze werden auf der letzten Rekursionsstufe (die kleinsten Kreuze) gezeichnet?
3. Wie müssen Sie Ihre rekursive Implementierung abändern, um das Muster in Abbildung 1b zu erzeugen?

[1] Mehr Informationen zu *DoubleBuffering* finden Sie unter: `https://introcs.cs.princeton.edu/java/stdlib/javadoc/StdDraw.html`