Aufgabenblatt 5

Kompetenzstufe 1 & Kompetenzstufe 2

Allgemeine Informationen zum Aufgabenblatt:

- Die Abgabe erfolgt in TUWEL. Bitte laden Sie Ihr IntelliJ-Projekt bis spätestens **Freitag, 04.01.2019 13:00 Uhr** in TUWEL hoch.
- Zusätzlich müssen Sie in TUWEL ankreuzen, welche Aufgaben Sie gelöst haben und während der Übung präsentieren können.
- 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, wie z.B. die Klassen `StdDraw`, `String`, `Scanner` und `Math` oder Klassen, die in den Hinweisen zu den einzelnen Aufgaben aufscheinen.

In diesem Aufgabenblatt werden folgende Themen behandelt:

- Rekursion
- Rekursion in Verbindung mit Arrays und Strings
- Einfache und verzweigte Rekursion

Aufgabe 1

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.

- Implementieren Sie eine rekursive Methode `printAndCountNumbers`:

  ```
  int printAndCountNumbers(int x, int y)
  ```

  Diese Methode nimmt zwei int-Werte entgegen und gibt alle Zahlen **aufsteigend** zwischen `x` und `y` (beide inklusive) aus. Zusätzlich werden alle ausgegebenen Werte aufsummiert und als Ergebnis zurückgegeben. *Vorbedingung:* `x` $\leq$ `y`.

- Implementieren Sie eine rekursive Methode `printAndCountNumbersDes`:

  ```
  int printAndCountNumbersDes(int x, int y)
  ```

  Diese Methode nimmt zwei int-Werte entgegen und gibt alle Zahlen **absteigend** zwischen `y` und `x` (beide inklusive) aus. Zusätzlich werden alle ausgegebenen Werte aufsummiert und als Ergebnis zurückgegeben. *Vorbedingung:* `x` $\leq$ `y`.

- Implementieren Sie eine rekursive Methode `calcMaxSumTriple`:

  ```
  int calcMaxSumTriple(int[] array, int i)
  ```

  Diese Methode nimmt ein int-Array und einen int-Wert entgegen und berechnet die größte Summe von drei aufeinander folgenden Arrayeinträgen. Das heißt, dass die Summen von 3 Arrayeinträgen ab Index 0 bis zum Index `array.length - 3` gebildet werden. *Vorbedingungen:* `array != null`, `array.length > 2` und `i` ist ein gültiger Index von `array`.

  Beispiele:
  `calcMaxSumTriple(new int[]{1,4,8,3,7,3,8,2,7,4,3}, 0)` liefert `18`
  `calcMaxSumTriple(new int[]{1,4,8,3,7,1,8,7,3,4,3}, 0)` liefert `18`
  `calcMaxSumTriple(new int[]{7,5,3}, 0)` liefert `15`

- Implementieren Sie eine rekursive Methode `calcMaxSumTriple`:

  ```
  int calcMaxSumTriple(int[] array)
  ```

  Diese Methode nimmt ein int-Array entgegen und berechnet so wie die Methode zuvor die größte Summe von drei aufeinander folgenden Arrayeinträgen. Der Unterschied zur vorherigen Methode ist, dass es jetzt keinen int-Wert in der Parameterliste gibt und die Lösung nur mit Hilfe von Arrays erzeugt werden muss. Die davor präsentierte Methode `calcMaxSumTriple(int[] array, int i)` mit dem Parameter `i` darf an dieser Stelle **nicht** verwendet werden. Sie können aber die Befehle `System.arraycopy(...)` oder `Arrays.copyOfRange(...)` verwenden. *Vorbedingungen:* `array != null` und `array.length > 2`.

Beispiele:
`calcMaxSumTriple(new int[]{1,4,8,3,7,3,8,2,7,4,3})` liefert `18`
`calcMaxSumTriple(new int[]{1,4,8,3,7,1,8,7,3,4,3})` liefert `18`
`calcMaxSumTriple(new int[]{7,5,3})` liefert `15`

- Implementieren Sie eine rekursive Methode `findMaxDiff`:

```
int findMaxDiff(int[] array, int i)
```

Diese Methode nimmt ein int-Array und einen int-Wert entgegen und berechnet die maximale absolute Differenz zweier aufeinander folgender Elemente von `array` im Indexbereich 0 bis `i` (inklusive). *Vorbedingungen:* `array != null`, `array.length > 1` und `i` ist ein gültiger Index von `array`.

Beispiele:
`findMaxDiff(new int[]{5,50,7,1,20}, 4)` liefert `45`
`findMaxDiff(new int[]{5,8,7,1,20}, 2)` liefert `3`
`findMaxDiff(new int[]{5,14,5,1,2,1,20}, 6)` liefert `19`
`findMaxDiff(new int[]{1,1,1,1,1,1,1}, 6)` liefert `0`
`findMaxDiff(new int[]{2,4}, 1)` liefert `2`

Aufgabe 2

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.

- Implementieren Sie eine rekursive Methode `insertIndex`:

  ```
  String insertIndex(String s)
  ```

  Diese Methode nimmt einen String entgegen und fügt vor jedes Zeichen des Strings dessen Index im String ein. Das Ergebnis wird als neuer String zurückgeliefert. *Vorbedingung:* `s != null`.

  Beispiele:
  `insertIndex("Hallo")` liefert `"0H1a2l3l4o"`
  `insertIndex("Fahrkarten!")` liefert `"0F1a2h3r4k5a6r7t8e9n10!"`
  `insertIndex("")` liefert `""`

- Implementieren Sie eine rekursive Methode `mixStrings`:

  ```
  String mixStrings(String s1, String s2)
  ```

  Diese Methode nimmt zwei Strings entgegen und kombiniert beide Strings zeichenweise. Auf das erste Zeichen aus `s1` folgt das erste Zeichen aus `s2`, danach das nächste Zeichen aus `s1` und dann wiederum das nächste Zeichen aus `s2`, u.s.w. bis zuletzt an das letzte Zeichen von `s1` ein weiteres Zeichen aus `s2` angehängt wird. Die Länge des Ergebnisstrings ist immer 2 mal die Länge von `s1`. Ist `s2` kürzer als `s1`, werden die Zeichen von `s2` zyklisch wiederholt genutzt. *Vorbedingungen:* `s1 != null` und `s2 != null` .

  Beispiele:
  `mixStrings("GROSS","klein")` liefert `"GkRlOeSiSn"`
  `mixStrings("ABC","klein")` liefert `"AkBlCe"`
  `mixStrings("GROESSER","klein")` liefert `"GkRlOeEiSnSkElRe"`

- Implementieren Sie eine rekursive Methode `shiftMinCharLeft`:

  ```
  String shiftMinCharLeft(String s)
  ```

  Diese Methode nimmt einen String entgegen und verschiebt das Zeichen mit dem kleinsten Wert (ASCII-Wert) an die erste Stelle. Alle Zeichen vom Index 0 bis zu der ursprünglichen Position des Minimums werden dabei um eine Position in Richtung größerem Index verschoben. Das Ergebnis wird als neuer String zurückgeliefert. *Vorbedingung:* `s != null`. Sie können davon ausgehen, dass das Zeichen mit dem kleinsten Wert nur einmal vorkommt.

  Beispiele:
  `shiftMinCharLeft("xdbycfjadfmk")` liefert `"axdbycfjdfmk"`
  `shiftMinCharLeft("bcdefghijklmnoa")` liefert `"abcdefghijklmno"`
  `shiftMinCharLeft("")` liefert `""`

Aufgabe 3

Implementieren Sie folgende Aufgabenstellung:

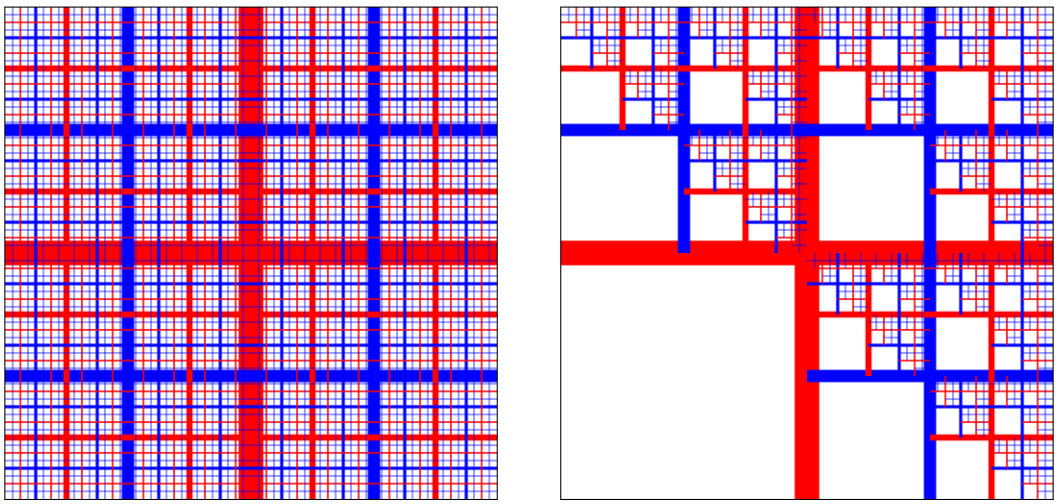
- Implementieren Sie die rekursive Methode `drawCrossPattern`:

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

- Diese Methode zeichnet Kreuze bestehend aus horizontalen und vertikalen Rechtecken. Der Methode werden die Koordinaten `x` und `y` der Rechteckmittelpunkte übergeben. 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 nimmt als vierten Parameter einen boolean-Wert `c` entgegen, der zur Farbsteuerung verwendet wird. Ist der Parameter `c == true`, dann wird das Rechteck (Kreuz) *rot* gezeichnet, bei `false` *blau*. Bei jeder Rekursionsstufe ändert sich die Farbe.

- Der Aufruf von `drawCrossPattern(0, 0, 512, true)` erzeugt durch Selbstaufrufe der Methode `drawCrossPattern` 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.

(!) Durch Verwendung von `StdDraw.setXscale(...)` und `StdDraw.setYscale(...)` können Sie den Ursprung des StdDraw-Fensters verschieben.



(a) (b)

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

(!) Setzen Sie die Fenstergröße auf 512×512 Pixel, um bei einer Auflösungsgrenze von `l<16` Pixel das Muster in Abbildung 1a zu erhalten. Für eine schnellere Anzeige der Grafik verwenden Sie *DoubleBuffering*[1].

Zusatzfrage(n):

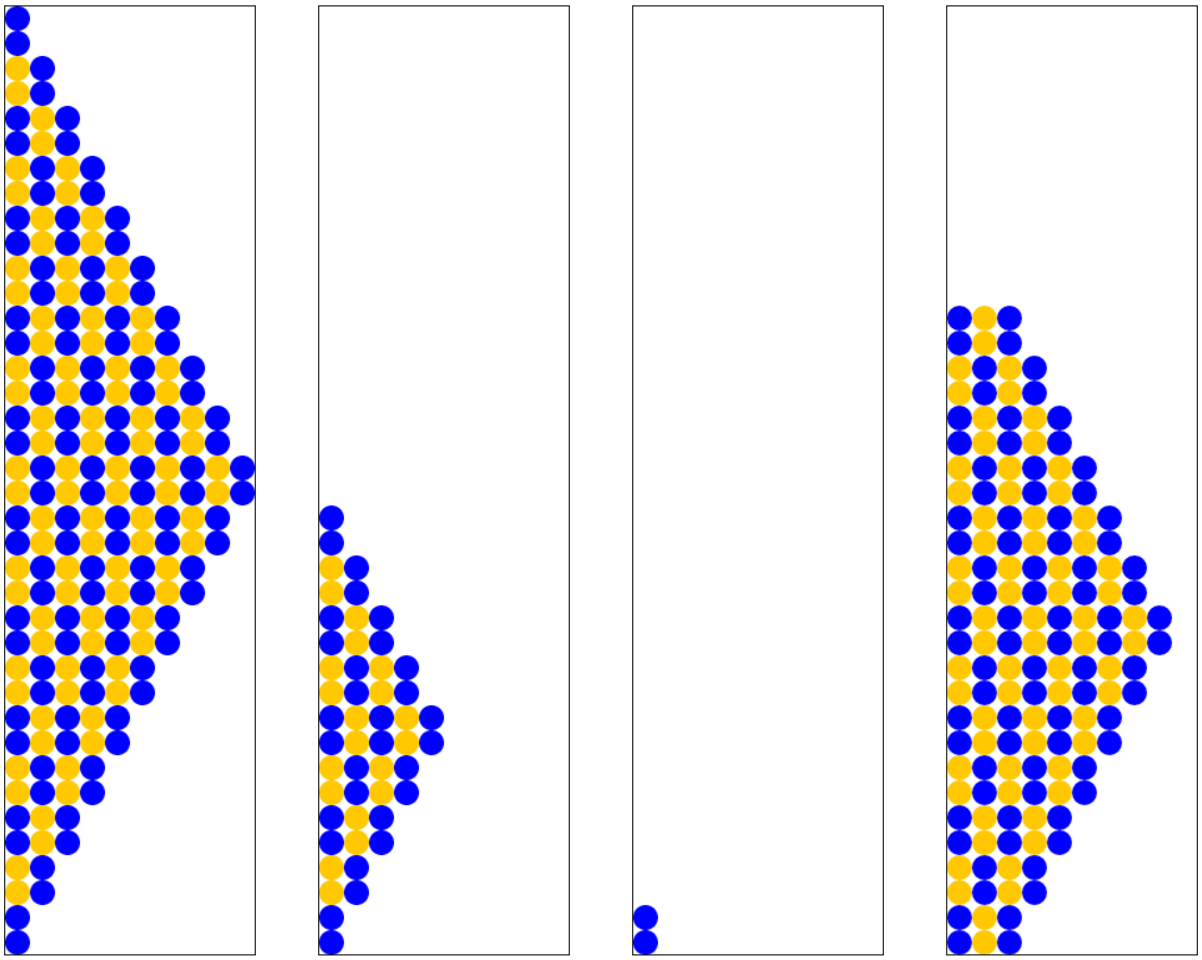
1. Warum benötigen Sie bei einer Rekursion eine Abbruchbedingung?
2. Gibt es eine Limitierung für die Rekursionstiefe?
3. Wie oft wird die Methode `drawCrossPattern` aufgerufen, wenn als Abbruchbedingung die Auflösungsgrenze von `l<16` gewählt wird?
4. Wie viele Kreuze werden auf der letzten Rekursionsstufe (die kleinsten Kreuze) gezeichnet?
5. Wie müssen Sie Ihr Programm 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`

Aufgabe 4

Implementieren Sie folgende Aufgabenstellung:

- Für die Ausgaben in Abbildung 2 müssen Sie Rekursion anwenden und dürfen keinerlei Schleifen verwenden. Dazu stehen Ihnen nur zwei Methoden zur Verfügung, deren Methodenköpfe nachfolgend beschrieben werden und nicht verändert werden dürfen.



(a) (b) (c) (d)

Abbildung 2: Rekursiver Aufbau einer Figur mit Hilfe von zweireihigen Kreiszeilen.

- Implementieren Sie eine Methode `printCirclesInLine`:

```
void printCirclesInLine(int val, int x, int y)
```

Diese Methode nimmt einen ganzzahligen Parameter `val` entgegen, der angibt, wie viele Kreisepaare (noch) in einer Zeile zu zeichnen sind. Jede Zeile ist zweireihig und besteht aus zwei Kreisen übereinander (Abbildung 2). In jeder zweireihigen Zeile werden die Kreise abwechselnd *blau* und *orange* gezeichnet. Die beiden Parameter `x` und `y` geben den Mittelpunkt des aktuell zu zeichnenden Kreises der unteren Kreisreihe an. Die zweite Kreisreihe darüber wird in Relation dazu gezeichnet.

- Implementieren Sie eine Methode `printShape`:

  ```
  void printShape(int val, int max, int y)
  ```

  Diese Methode nimmt einen ganzzahligen Parameter `val` entgegen, der angibt, wie viele Kreise in der aktuellen Zeile der Figur zu zeichnen sind. Der zweite ganzzahlige Parameter `max` gibt die Maximalanzahl der Kreise in der mittleren Zeile der Figur an. Der dritte Parameter `y` gibt an, bei welcher y-Koordinate aktuell gezeichnet werden soll. Bei Erstellung der Figur ist darauf zu achten, dass jede Zeile (von unten nach oben) abwechselnd *blau* und *orange* beginnt.

- Jetzt müssen Sie diese beiden Methoden so implementieren, dass darin keine Schleifen vorkommen und eine korrekte Ausgabe von Kreisen erzeugt wird. Der Radius der Kreise wird als Konstante `RADIUS` mit 10 Pixel deklariert. Für die einzelnen Ausgaben in Abbildung 2 werden folgende Aufrufe benutzt:

  (a) `printShape(1, 10, RADIUS);`

  (b) `printShape(1, 5, RADIUS);`

  (c) `printShape(1, 1, RADIUS);`

  (d) `printShape(3, 9, RADIUS);`