Aufgabenblatt 5

Kompetenzstufe 1 & Kompetenzstufe 2

Allgemeine Informationen zum Aufgabenblatt:

- Die Abgabe erfolgt in TUWEL. Bitte laden Sie Ihr IntelliJ-Projekt bis spätestens **Mittwoch, 13.12.2023 23:55 Uhr** in TUWEL hoch.
- Zusätzlich müssen Sie in TUWEL ankreuzen, welche Aufgaben Sie gelöst haben.
- Ihre Programme müssen kompilierbar und korrekt ausführbar sein.
- Ändern Sie bitte **nicht** die **Dateinamen** und die **vorhandene Ordnerstruktur**.
- 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`, `Integer` und `CodeDraw` 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:

- Ein- und zweidimensionale Arrays
- Rekursion
- Grafische Ausgabe
- Zweidimensionale Arrays und Bilder

Aufgabe 1 (1 Punkt)

Implementieren Sie folgende Aufgabenstellung:

a) Implementieren Sie eine Methode `shiftLines`:

```
void shiftLines(int[][] workArray)
```

Diese Methode baut ein ganzzahliges, zweidimensionales Array `workArray` so um, dass die Zeile, die am kürzesten ist, in die erste Zeile verschoben wird. Die erste Zeile wird dann an jene Stelle verschoben, wo zuvor die kürzeste Zeile gewesen ist. Sollte es eine kürzeste Zeile mehrfach geben, dann wird nur jene Zeile verschoben, die den kleinsten Zeilenindex aufweist. Dafür wird das Array `workArray` umgebaut und kein neues Array erstellt. Sie dürfen aber innerhalb der Methode eine temporäre Array-Variable anlegen. Sind alle Zeilen des Arrays gleich lange, dann wird keine Verschiebung und Veränderung des Arrays durchgeführt.

Vorbedingungen: `workArray != null`, `workArray.length > 0` und für alle gültigen `i` gilt `workArray[i] != null` und `workArray[i].length > 0`.

Beispiele:

Aufruf	Ergebnis

`shiftLines(`
`new int[][]{`
`{1,3,2},`
`{6,2,5},`
`{0,7,9}})`
	`1 3 2`
	`6 2 5`
	`0 7 9`

`shiftLines(`
`new int[][]{`
`{1,5,6,7},`
`{1,9,6},`
`{4,3},`
`{6,3,0,6,9},`
`{6,4,3}})`
	`4 3`
	`1 9 6`
	`1 5 6 7`
	`6 3 0 6 9`
	`6 4 3`

`shiftLines(`
`new int[][]{`
`{3,2,4,1},`
`{7,3,6},`
`{4},`
`{5,6,2,4},`
`{9,1},`
`{3}})`
	`4`
	`7 3 6`
	`3 2 4 1`
	`5 6 2 4`
	`9 1`
	`3`

b) Implementieren Sie eine Methode `reformatArray`:

```
void reformatArray(int[][] inputArray)
```

Diese Methode zählt in jeder Zeile von `inputArray` die Anzahl der Nullen und ändert die Zeile so um, dass die Anzahl der Nullen an erster Stelle in der Arrayzeile steht. Hinter dem ersten Arrayeintrag folgen so viele Einser wie im ursprünglichen Array in jeder Zeile gestanden sind. Von den Einsen wird nicht die Summe in die Arrayzeile geschrieben, sondern jeder Einser nimmt für sich einen Platz im Array ein (siehe Beispiele).

Vorbedingungen: `inputArray != null`, `inputArray.length > 0`, dann gilt für alle gültigen `i`, dass `inputArray[i].length > 0` ist. Alle Zahlen in `inputArray` sind Einsen und Nullen.

Beispiele:

Aufruf	Ergebnis

`reformatArray(new int[][]{`
`{1, 0, 1, 1},`
`{0, 1, 1},`
`{0, 1, 0, 1, 1},`
`{0, 0, 0, 1, 0},`
`{1, 0},`
`{1, 1, 1, 1, 1}})`
	`1 1 1 1`
	`1 1 1`
	`2 1 1 1`
	`4 1`
	`1 1`
	`0 1 1 1 1 1`

`reformatArray(new int[][]{`
`{1, 0, 1, 1, 0, 0, 0, 0},`
`{0, 1, 1, 1, 1, 1, 0, 0},`
`{0, 0, 0, 0, 0, 0, 1, 1},`
`{1, 0, 0, 0, 0, 0, 0, 0},`
`{0, 0, 0, 0, 1, 1, 0, 1},`
`{0, 0, 0, 0, 0, 0, 0, 0},`
`{0, 0, 0, 0, 0, 0, 0, 1}})`
	`5 1 1 1`
	`3 1 1 1 1 1`
	`6 1 1`
	`7 1`
	`5 1 1 1`
	`8`
	`7 1`

`reformatArray(new int[][]{`
`{0},`
`{1},`
`{0, 0},`
`{0, 1},`
`{1, 0},`
`{1, 1}})`
	`1`
	`0 1`
	`2`
	`1 1`
	`1 1`
	`0 1 1`

Aufgabe 2 (1 Punkt)

Implementieren Sie folgende Aufgabenstellung:

- Implementieren Sie eine Methode `makeOver`:

  ```
  char[][] makeOver(char[][] inputArray)
  ```

  Diese Methode liefert ein neues Array zurück, bei dem das ursprüngliche Array `inputArray` durch das Zeichen `*` umrandet wird. Das neue Array hat somit 2 Zeilen mehr als `inputArray`, und seine erste Zeile (Index 0) hat 2 Einträge mehr als die erste Zeile von `inputArray` und besteht aus lauter `*`. Ebenso verhält es sich mit der letzten Zeile. Die übrigen Zeilen werden aus `inputArray` kopiert, wobei an der ersten und letzten Stelle ein `*` eingefügt wird.

  Vorbedingung(en): `inputArray != null`, `inputArray.length > 0`, `inputArray[i] != null` und `inputArray[i].length > 0` für alle gültigen `i`.

  Beispiele:

Aufruf	Ergebnis

`makeOver(new char[][]{`
`{'H', 'i', '!'},`
`{'E', 'P'},`
`{'1'}`
`});`
	`*****`
	`*Hi!*`
	`*EP*`
	`*1*`
	`***`

`makeOver(new char[][]{`
`{'O'}`
`});`
	`***`
	`*O*`
	`***`

`makeOver(new char[][]{`
`{'H'},`
`{'H', 'H', 'H'},`
`{'H'}`
`});`
	`***`
	`*H*`
	`*HHH*`
	`*H*`
	`***`

- Implementieren Sie eine Methode `change`:

  ```
  void change(int[][] workArray)
  ```

  Diese Methode ändert die Zeilen des angegebenen Arrays `workArray`, deren erstes Element `x` eine negative Zahl ist, indem so viele Elemente vom Anfang der Zeile entfernt werden, wie der Absolutbetrag von `x` ausmacht (falls weniger Elemente vorhanden sind, wird die Zeile einfach geleert). Zeilen, die kein Element enthalten oder mit einem nicht negativen Wert beginnen, bleiben unverändert.

  Vorbedingung(en): `workArray != null` und es gilt `workArray[i] != null` für alle gültigen Indizes `i`.

  Beispiele:

Aufruf	Ergebnis

`change(`
`new int[][]{`
`{1, 4, -2, 0},`
`{2},`
`{-3, -1},`
`{-2, 6, -5, 8, 3}`
`});`
	`1 4 -2 0`
	`2`
	
	`-5 8 3`

`change(`
`new int[][]{`
`{0, 2},`
`{},`
`{-1, 5}`
`});`
	`0 2`
	
	`5`

`change(`
`new int[][]{`
`{-2, -2, 0},`
`{-1},`
`{2, -1, -3},`
`{-3, 4, -2, 6, 8, -5}`
`});`
	`0`
	
	`2 -1 -3`
	`6 8 -5`

Aufgabe 3 (2 Punkte)

Sie haben ein Programm gegeben, das ein gegebenes 9×9 Sudoku[1]-Spielfeld lösen soll. Die bereits fertig implementierte Methode `solveSudoku` bearbeitet ein ganzzahliges zweidimensionales Array der Größe 9×9 und vervollständigt leere Felder (mit 0 gekennzeichnet, siehe Abbildung 1a), sodass am Ende eine gültige Sudoku-Lösung (siehe Abbildung 1b) vorhanden ist. Es sind 8 Spielfelder vorhanden und über die Variable `fileName` können sie die 8 Sudoku-Spielfelder mit den Namen `sudoku0.csv ... sudoku7.csv` ansprechen und mit der Methode `readArrayFromFile` einlesen. Dazu müssen Sie nur den Inhalt der Variable `fileName` entsprechend verändern. Für die korrekte Funktionsweise von `solveSudoku` fehlen noch die Implementierungen von verschiedenen Hilfsmethoden. Zusätzlich müssen Sie noch eine Testmethode erstellen, die überprüft, ob es sich bei einem komplett gefüllten 9×9 Sudoku-Spielfeld um eine gültige Lösung handelt.

0	0	0	1	0	0	0	0	9
0	4	0	0	6	9	0	2	8
0	0	3	2	0	0	0	5	0
0	3	8	0	0	0	5	0	2
4	0	0	7	0	2	3	8	0
2	7	0	5	0	0	0	0	4
0	6	5	4	2	0	0	9	7
8	1	7	6	0	0	2	4	5
0	2	0	8	0	0	1	6	3

(a)

6	8	2	1	5	7	4	3	9
5	4	1	3	6	9	7	2	8
7	9	3	2	8	4	6	5	1
1	3	8	9	4	6	5	7	2
4	5	9	7	1	2	3	8	6
2	7	6	5	3	8	9	1	4
3	6	5	4	2	1	8	9	7
8	1	7	6	9	3	2	4	5
9	2	4	8	7	5	1	6	3

(b)

Abbildung 1: a) Beispiel für ein ungelöstes Sudoku-Spielfeld und b) Beispiel für eine gültige Sudoku-Lösung.

Implementieren Sie folgende Aufgabenstellung:

- Implementieren Sie eine Methode `isNumUsedInBox`:

  ```
  boolean isNumUsedInBox(int[][] array, int num, int row, int col)
  ```

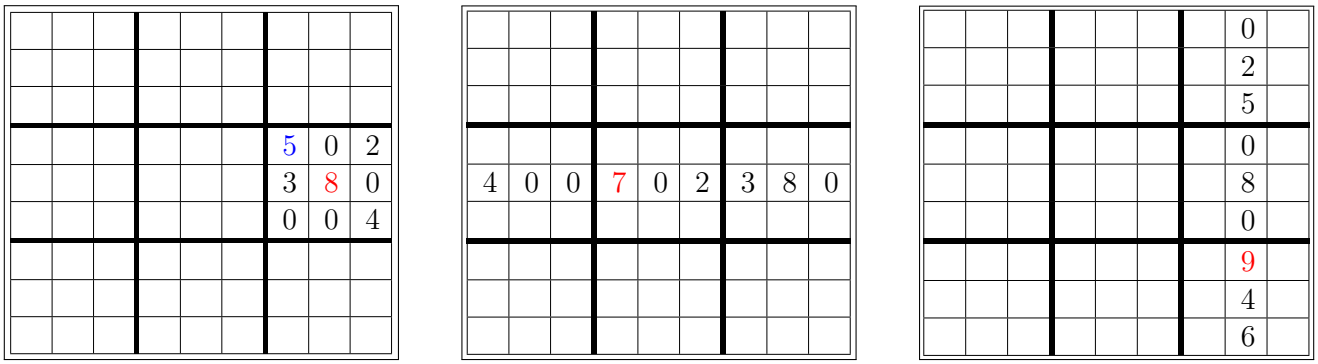
  Diese Methode überprüft, ob ein Wert `num` bereits innerhalb eines 3×3 Feldes vorkommt. Das Sudoku-Spielfeld besteht ja aus 9 3×3 großen Subfeldern, bei denen laut Sudoku-Regeln die Zahlen 1-9 nur einmal vorkommen dürfen. Die Parameter `row` und `col` geben den Index des linken oberen Elements eines 3×3 Feldes an.

  Beispiel (`field` ist ein Array, das ein 9×9 Sudoku-Spielfeld repräsentiert):

  Der Aufruf `isNumUsedInBox(field, 8, 3, 6)` liefert `true` zurück, da dieses Subfeld (siehe Abbildung 2a) bereits den Wert 8 (rot) beinhaltet. Der Wert für `row == 3` und der Wert für `col == 6` gibt das Element in der linken oberen Ecke des Subfeldes an (in diesem Beispiel die blaue 5).

  Für den Aufruf `isNumUsedInBox(field, 6, 3, 6)` liefert die Methode `false` zurück, da der Wert 6 im gezeigten Subfeld noch nicht vorhanden ist.

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



(a) (b) (c)

Abbildung 2: a) Beispiel für ein 3×3 Sudoku-Subfeld, b) Beispiel für eine Zeile innerhalb des Sudoku-Spielfeldes und c) Beispiel für eine Spalte innerhalb des Sudoku-Spielfeldes.

Vorbedingungen: `array != null`, `array.length == 9`, dann gilt auch für alle gültigen `i`, dass `array[i].length == 9` ist. Die Indizes `row` und `col` sind gültige Indizes der oberen linken Ecke eines Sudoku-Subfeldes und `num` ist ein Wert von 1-9.

- Implementieren Sie eine Methode `isNumUsedInRow`:

```
boolean isNumUsedInRow(int[][] array, int num, int row)
```

Diese Methode überprüft, ob ein Wert `num` bereits innerhalb einer Zeile `row` des Arrays `array` vorkommt. Laut Sudoku-Regeln dürfen die Zahlen 1-9 nur einmal in jeder Zeile vorkommen.

Beispiel (`field` ist ein Array, das ein 9×9 Sudoku-Spielfeld repräsentiert):

Der Aufruf `isNumUsedInRow(field, 7, 4)` liefert `true` zurück, da in der Zeile (siehe Abbildung 2b) `row == 4` bereits der Wert 7 (rot) vorhanden ist.

Für den Aufruf `isNumUsedInRow(field, 5, 4)` liefert die Methode `false` zurück, da der Wert 5 in der gezeigten Zeile noch nicht vorhanden ist.

Vorbedingungen: `array != null`, `array.length == 9`, dann gilt auch für alle gültigen `i`, dass `array[i].length == 9` ist. Der Wert `num` ist eine Zahl von 1-9 und `row` ist ein gültiger Index.

- Implementieren Sie eine Methode `isNumUsedInCol`:

```
boolean isNumUsedInCol(int[][] array, int num, int col)
```

Diese Methode überprüft, ob ein Wert `num` bereits innerhalb einer Spalte `col` des Arrays `array` vorkommt. Laut Sudoku-Regeln dürfen die Zahlen 1-9 nur einmal in jeder Spalte vorkommen.

Beispiel (`field` ist ein Array, das ein 9×9 Sudoku-Spielfeld repräsentiert):

Der Aufruf `isNumUsedInCol(field, 9, 7)` liefert `true` zurück, da in der Spalte (siehe Abbildung 2c) `col == 7` bereits der Wert 9 (rot) vorhanden ist.

Für den Aufruf `isNumUsedInCol(field, 1, 7)` liefert die Methode `false` zurück, da der Wert 1 in der gezeigten Spalte noch nicht vorhanden ist.

Vorbedingungen: `array != null`, `array.length == 9`, dann gilt auch für alle gültigen `i`, dass `array[i].length == 9` ist. Der Wert `num` ist eine Zahl von 1-9 und `col` ist ein gültiger Index.

- Implementieren Sie eine Methode `isValidSudokuSolution`:

  ```
  boolean isValidSudokuSolution(int[][] array)
  ```

  Diese Methode überprüft, ob es sich bei einem vollständig gefüllten 9×9 Sudoku-Spielfeld `array` um eine gültige Sudoku-Lösung handelt. Dazu kontrolliert die Methode, ob die Zahlen 1-9 in jeder Zeile, jeder Spalte und jedem 3×3 Subfeld nur einmal vorkommen. Wenn dies der Fall ist, dann wird `true` zurückgegeben, ansonsten `false`.

  Vorbedingungen: `array != null`, `array.length == 9`, dann gilt auch für alle gültigen `i`, dass `array[i].length == 9` ist. Alle Werte in `array` sind Zahlen von 1-9.

  Beispiel (`field` ist das 9×9 Array in Abbildung 1b):

  Der Aufruf `isValidSudokuSolution(field)` liefert `true`, da es sich um eine gültige Sudoku-Lösung handelt. In den Dateien `sudoku5.csv ... sudoku7.csv` finden Sie vollständig ausgefüllte 9×9 Sodoku-Spielfelder, aber mit Fehlern eingebaut. Jeder Aufruf der Methode `isValidSudokuSolution` mit diesen drei Spielfeldern muss `false` zurückliefern.

Aufgabe 4 (2 Punkte)

Bei dieser Aufgabe geht es darum, eine Anwendung aus der Bildverarbeitung zu realisieren. Es sollen zusammenhängende Segmente mit der Farbe `Palette.LIME` gefüllt werden. Dazu wird ein sogenannter *Flood-Fill*-Algorithmus verwendet. Sie haben ein Graustufenbild (siehe Abbildung 3) von einer Weltkarte gegeben. Nach einem Klick im Bild werden Sie zur Eingabe eines Graustufenschwellwertes zwischen 1-50% aufgefordert. Dazu wird die Methode `readIntValue` verwendet. Das Pixel an der Position des Klicks hat einen gewissen Grauwert. Mit dem Schwellwert geben Sie an, um wie viel Prozent der Grauwert abweichen (positive und negative Richtung) darf, um noch demselben Segment zugeordnet zu werden. Danach wird die Methode `floodFill` aufgerufen, die nachfolgend genauer beschrieben wird.
Hinweis: Beachten Sie, dass bei diesem Projekt bei den VM-Optionen die Stackgröße (`-Xss16m`) erhöht wurde.

Abbildung 3: Graustufenbild einer einfachen Weltkarte. Oranger Kreis in Afrika dient zur Beispielsdemonstration

Implementieren Sie folgende Aufgabenstellung:

- Implementieren Sie eine Methode `readIntValue`:

  ```
  int readIntValue(Scanner sc)
  ```

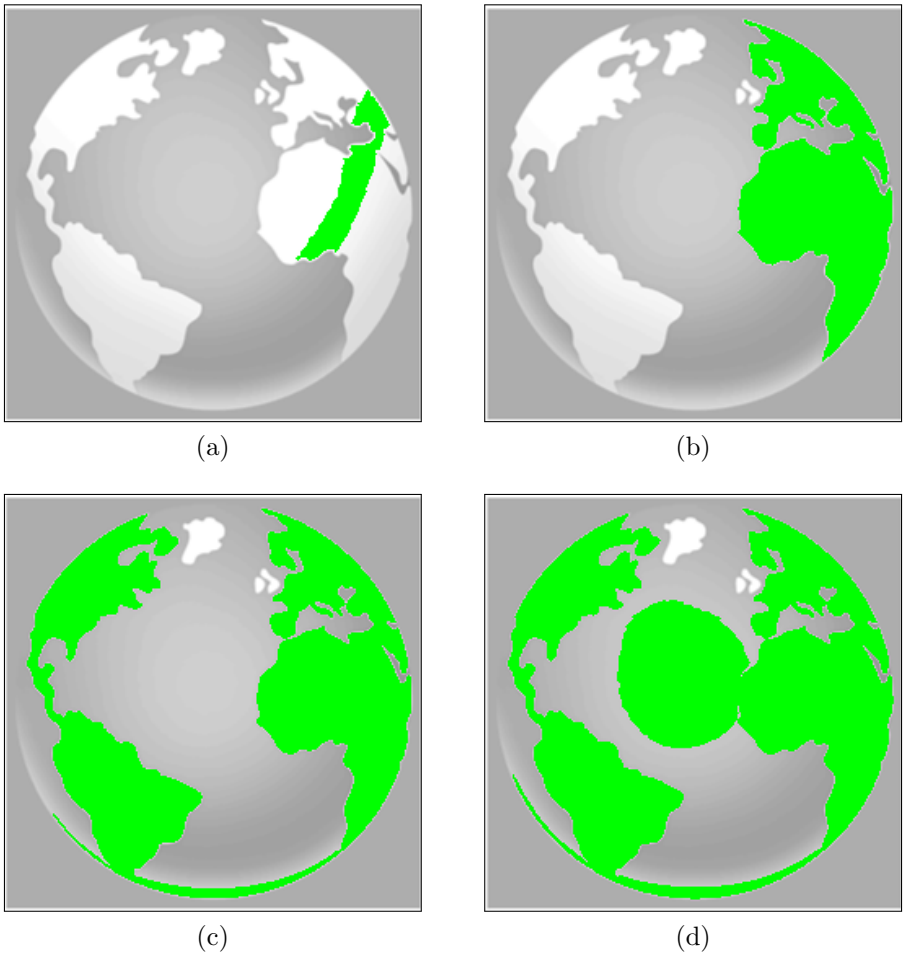
  Diese Methode wird in `main` wiederholt aufgerufen, bis ein korrekter Integer-Wert im Intervall [1, 50] eingegeben wurde. Falsche Eingaben werden ignoriert und der User wird mit einem entsprechenden Kommentar benachrichtigt. Innerhalb der Methode wird mittels `Scanner` von der Konsole solange eingelesen, bis ein Integer-Wert eingegeben wurde. Falsche Eingaben (Datentypen) werden ignoriert und der User wird mit einer Meldung darauf hingewiesen. Der ganzzahlige Wert wird anschließend zurückgegeben.

- Implementieren Sie eine Methode `floodFill`:

  ```
  void floodFill(CodeDraw myDrawObj, int[][] imgArr,
        int sx, int sy, int gVal, int gThresh)
  ```

  Diese Methode bekommt ein ganzzahliges Integer-Array `imgArray` übergeben, das die Grauwerte des Bildes gespeichert hat. Die Parameter `sx` und `sy` entsprechen beim ersten Aufruf

den Koordinaten des Mausklicks. Der Parameter `gVal` gibt den Grauwert des geklickten Pixels an und der Parameter `gThresh` gibt den gewählten Graustufenschwellwert in Prozent an. Nun müssen Sie ausgehend von den Koordinaten des Mausklicks in alle 4 Richtungen (oben, unten, links, rechts) rekursiv weiter suchen, ob der Grauwert des entsprechenden Nachbarpixels innerhalb des Graustufenschwellwertes liegt. Sollte das der Fall sein, dann wird dieses Pixel im CodeDraw-Fenster grün (`Palette.LIME`) eingefärbt und im Array `imgArray` mit -1 markiert. Danach werden wieder die Nachbarn von diesem Pixel betrachtet. So arbeitet sich der Algorithmus in alle 4 Richtungen weiter, bis der Graustufenschwellwert erreicht wird. Zum Beispiel wurde für die Generierung von Abbildung 4a bis 4d auf den Kontinent Afrika geklickt (siehe Abbildung 3 oranger Kreis) und ein Schwellwert von 1%, 15%, 17% und 20% eingegeben. Nun füllt der Algorithmus die zusammenhängenden Nachbarpixel, welche sich um maximal den Schwellwert unterscheiden, mit der grünen Farbe auf. Der Anfangswert (Graustufe des Klickpunktes) und der Schwellwert bestimmen den Unterschied, wie weit die Füllung verläuft und welche Bereiche eingefärbt werden. Experimentieren Sie mit verschiedenen Klickpunkten und Schwellwerten. Hinweis: Die Füllung kann je nach Klickpunkt abweichen. Das heißt, dass vor allem bei 1% Graustufenschwellwrt ein anderes Ergebnis bei Ihnen entstehen kann.



(a) (b)

(c) (d)

Abbildung 4: Ergebnisse mit den Schwellwerten a) 1%, b) 15%, c) 17% und d) 20%, wenn auf den orangen Kreis in Abbildung 3 geklickt wurde.