Aufgabenblatt 5

Kompetenzstufe 1 & Kompetenzstufe 2

Allgemeine Informationen zum Aufgabenblatt:

- Die Abgabe erfolgt in TUWEL. Bitte laden Sie Ihr IntelliJ-Projekt bis spätestens **Freitag, 03.01.2020 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.
- Erlaubt sind die Klassen `String`, `Math`, `Integer` und `StdDraw` oder Klassen, die in den Hinweisen zu den einzelnen Aufgaben aufscheinen.

In diesem Aufgabenblatt werden folgende Themen behandelt:

- Zweidimensionale Arrays
- Rekursion mit grafischer Ausgabe
- Rekursion mit zweidimensionalen Arrays

Aufgabe 1

Implementieren Sie folgende Aufgabenstellung:

- Implementieren Sie eine Methode `genFilledArray`:

  ```
  int[][] genFilledArray(int n)
  ```

  Die Methode erzeugt ein zweidimensionales Array der Größe n×n und befüllt dieses mit Zahlen, wie in den nachfolgenden Beispielen gezeigt. Es wird links oben mit 1 begonnen und in jeder weiteren Diagonalen bis zur Hauptdiagonalen die Zahl um 1 erhöht. Nach der Hauptdiagonalen nehmen die Zahlen wieder ab, sodass es rechts unten wieder mit der 1 endet.

  Vorbedingung: `n > 1`.

  Beispiele:

  `genFilledArray(2)` erzeugt →

  ```
  1 2
  2 1
  ```

  `genFilledArray(4)` erzeugt →

  ```
  1 2 3 4
  2 3 4 3
  3 4 3 2
  4 3 2 1
  ```

  `genFilledArray(7)` erzeugt →

  ```
  1 2 3 4 5 6 7
  2 3 4 5 6 7 6
  3 4 5 6 7 6 5
  4 5 6 7 6 5 4
  5 6 7 6 5 4 3
  6 7 6 5 4 3 2
  7 6 5 4 3 2 1
  ```

- Implementieren Sie eine Methode `extendArray`:

  `int[][] extendArray(int[][] inputArray)`

  Diese Methode erstellt ein ganzzahliges zweidimensionales Array, bei dem jede Zeile die gleiche Länge aufweist. Die Länge der Zeilen wird durch die längste Zeile von `inputArray` bestimmt, da das Array `inputArray` unterschiedliche Zeilenlängen aufweisen kann. Der Inhalt jeder Zeile von `inputArray` wird dabei links mit Nullen aufgefüllt, sodass alle Zeilen des neuen Arrays gleich viele Einträge haben.

  Vorbedingungen: `inputArray != null` und `inputArray.length > 0`, dann gilt auch für alle gültigen `i`, dass `inputArray[i].length > 0`.

  Beispiele:

  `extendArray(new int[][]{{1, 1, 0}, {1}, {1, 0}, {1, 0, 0, 1}})` erzeugt →

```
0 1 1 0
0 0 0 1
0 0 1 0
1 0 0 1
```

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

erzeugt →

```
0 0 0 1 1 1 1 1
0 0 0 0 0 0 1 1
0 0 0 0 1 0 0 0
0 0 0 0 0 1 0 1
0 0 0 0 0 0 0 0
1 0 1 1 0 0 1 1
```

```
extendArray(new int[][]
{{1, 3, 2},
{5, 1},
{6, 8, 5, 10},
{9, 4, 1, 9, 2},
{3},
{0, 11, 7, 5, 3, 2, 5}})
```

erzeugt →

```
0 0 0 0 1 3 2
0 0 0 0 0 5 1
0 0 0 6 8 5 10
0 0 9 4 1 9 2
0 0 0 0 0 0 3
0 11 7 5 3 2 5
```

- Implementieren Sie eine Methode `reformatArray`:

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

  Diese Methode interpretiert jede Zeile von `inputArray` als Binärzahl und erstellt ein neues eindimensionales Array, das jede dieser Binärzahlen als Dezimalzahl beinhalten soll. Das Array mit den Dezimalzahlen wird anschließend zurückgegeben. Jede Zeile von `inputArray` kann von hinten nach vorne gelesen werden und dabei kann die Wertigkeit jeder Stelle ermittelt werden. Das letzte Element in jeder Zeile von `inputArray` hat die Wertigkeit 2^0, das vorletzte Element jeder Zeile die Wertigkeit 2^1, usw. Nach diesem Schema wird jede Zeile in eine Dezimalzahl umgewandelt. Die so entstandenen Dezimalzahlen werden im neuen Array der Reihe nach, beginnend beim Index 0, abgelegt.

  Vorbedingungen: `inputArray != null`, `inputArray.length > 0`, dann gilt für alle gültigen `i`, dass `inputArray[i].length > 0` $\wedge$ `inputArray[i].length < 32` ist. Alle Zahlen in `inputArray` sind Nullen oder Einsen.

  Beispiele:

  ```
  reformatArray(new int[][]
  {{1,0,1},
  {0,1,1}})
  ```
  erzeugt $\rightarrow$

  Zeile 1: `{1,0,1}` $\rightarrow 2^2 * 1 + 2^1 * 0 + 2^0 * 1 = 5$
  Zeile 2: `{0,1,1}` $\rightarrow 2^2 * 0 + 2^1 * 1 + 2^0 * 1 = 3$

  ```
  5 3
  ```

  ```
  reformatArray(new int[][]
  {{0,1,1,0},
  {0,0,0,1},
  {0,0,1,0},
  {1,0,0,1}})
  ```
  erzeugt $\rightarrow$

  ```
  6 1 2 9
  ```

  ```
  reformatArray(new int[]
  {{0,0,0,1,1,1,1,1},
  {0,0,0,0,0,0,1,1},
  {0,0,0,0,1,0,0,0},
  {0,0,0,0,0,1,0,1},
  {0,0,0,0,0,0,0,0},
  {1,0,1,1,0,0,1,1}})
  ```
  erzeugt $\rightarrow$

  ```
  31 3 8 5 0 179
  ```

Aufgabe 2

Implementieren Sie folgende Aufgabenstellung:
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), sodass am Ende eine gültige Sudoku-Lösung vorhanden ist. 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 und eine Methode, die das Einlesen von Spielfeldern aus Dateien ermöglicht. Nachfolgend werden alle von Ihnen zu implementierenden Methoden genauer beschrieben:

- Implementieren Sie eine Methode `readArrayFromFile`:

  ```
  int[][] readArrayFromFile(String filename)
  ```

  Diese Methode wird benötigt, um ein vorgegebenes Sudoku-Spielfeld von einer csv-Datei einzulesen. Sie benötigen für die Aufgabe keinen Scanner und können die Datei mit der Klasse *In*[2] einlesen und verarbeiten. Der Parameter `fileName` beschreibt die gewünschte Datei, die eingelesen werden soll. In Ihrem Projekt befinden sich 8 Sudoku-Spielfelder mit den Namen `sudoku0.csv ... sudoku7.csv`. In jeder Datei finden Sie 9×9 Einträge, die mit `";"` abgetrennt sind. Lesen Sie alle Werte in ein zweidimensionales ganzzahliges Array der Größe 9×9 ein. Dieses Array wird anschließend zurückgegeben.

  Vorbedingung: Die angegebene Datei ist vorhanden und die Datei hat 9×9 Werte abgespeichert.

  Beispiel:

  Der Aufruf `readArrayFromFile("sudoku2.csv")` liefert das in Abbildung 1 gezeigte Array zurück.

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

Abbildung 1: Beispiel für ein ungelöstes Sudoku-Spielfeld.

[1] `https://de.wikipedia.org/wiki/Sudoku`
[2] `https://introcs.cs.princeton.edu/java/stdlib/javadoc/In.html`

- 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 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:

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

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

- 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:

  Der Aufruf `isNumUsedInRow(array, 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(array, 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`, `num` ist ein Wert 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:

  Der Aufruf `isNumUsedInCol(array, 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(array, 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`, `num` ist ein Wert 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`.

Beispiel:

Der Aufruf `isValidSudokuSolution(array)` liefert für das 9×9 Array in Abbildung 2d `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. Der Aufruf der Methode `isValidSudokuSolution` mit einem dieser drei Spielfelder muss `false` zurückliefern.

						5	0	2
						3	8	0
						0	0	4

(a)

4	0	0	7	0	2	3	8	0

(b)

							0	
							2	
							5	
							0	
							8	
							0	
							9	
							4	
							6	

(c)

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

(d)

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

Aufgabe 3

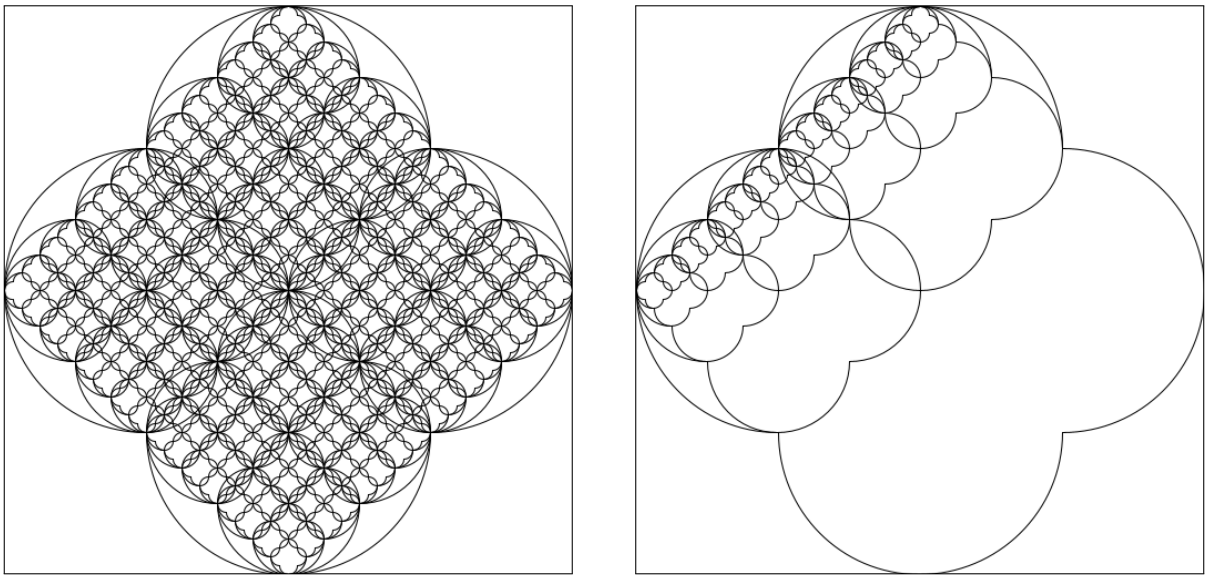
Implementieren Sie folgende Aufgabenstellung:

(!) 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.

- Implementieren Sie die rekursive Methode `drawArcPattern`:

```
void drawArcPattern(int x, int y, int radius)
```

Diese Methode zeichnet immer kleiner werdende Blumen bestehend aus Kreisbögen. Die Koordinaten `x` und `y` beschreiben dabei den Mittelpunkt einer Blume. Jede Blume besteht aus vier Kreisbögen, wobei jeder Kreisbogen einen Halbkreis darstellt und in eine der vier Richtungen (oben, unten, links und rechts) zeigt. Ausgehend von `x` und `y` sind die vier Mittelpunkte der Kreisbögen um den Wert `radius` in die vier genannten Richtungen verschoben. Die Kreisbögen haben den Radius `radius` und einen Winkel von 180°. Der Aufruf von `drawArcPattern(0, 0, 128)` erzeugt durch Selbstaufrufe der Methode `drawArcPattern` ein Blumenmuster, wie in Abbildung 3a dargestellt. Bei jedem rekursiven Aufruf wird der Mittelpunkt der nächsten Blume um `radius` in die vier Richtungen verschoben. Der Radius `radius` der Kreisbögen halbiert sich bei jedem Rekursionsschritt. Bei einer Auflösung von `radius < 8` Pixel soll das Zeichnen beendet werden.



(a) (b)

Abbildung 3: a) Rekursives Blumenmuster bestehend aus Kreisbögen. b) Abgeänderte Variante des Blumenmusters.

(!) Setzen Sie die Fenstergröße auf 512×512 Pixel, um bei einer Auflösungsgrenze von `radius < 8` Pixel das Muster in Abbildung 3a zu erhalten. Setzen Sie für diese Aufgabe

den Koordinatenursprung des StdDraw-Fensters in die Mitte des Fensters. Für eine schnellere Anzeige der Grafik verwenden Sie *DoubleBuffering*[3].

Zusatzfrage(n):

1. Wie oft wird die Methode `drawArcPattern` aufgerufen, wenn als Abbruchbedingung die Auflösungsgrenze von `radius < 8` gewählt wird?

2. Wie viele Kreisbögen werden auf der letzten Rekursionsstufe (die kleinsten Kreisbögen) gezeichnet, wenn als Abbruchbedingung die Auflösungsgrenze von `radius < 8` gewählt wird?

3. Wie müssen Sie Ihr Programm abändern, um das Muster in Abbildung 3b zu erzeugen?

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

Aufgabe 4

Erweitern Sie die Aufgabe um folgende Funktionalität:

- Implementieren Sie eine rekursive Methode `waterFlow`:

```
void waterFlow(int[][] map, int row, int col, int prevValue)
```

Diese Methode simuliert einen Wasserfluss. Dazu wird ein Punkt als Quelle definiert, von dem aus das Wasser auf gleicher Höhe oder abwärts fließt. In einer Landkarte `map` (zweidimensionales Array) sind Zahlenwerte eingetragen, die fiktiven Höhen entsprechen (siehe Tabelle 1).

9	5	2	9	6	11	7	8	9
9	6	3	4	6	11	1	1	7
6	9	8	5	10	11	1	1	6
9	7	9	7	9	3	2	6	5
9	12	8	8	20	8	6	7	8
9	12	8	5	7	9	5	7	8
6	4	8	4	9	10	5	4	3
5	3	3	4	11	10	8	9	9
2	2	6	6	9	10	10	10	9

Tabelle 1: Landkarte mit fiktiven Höhenwerten. In der Mitte ist der höchste Punkt und entspricht der Wasserquelle.

In der Mitte der Landkarte befindet sich immer der höchste Wert, der der Quelle entspricht und als Startpunkt für die Simulation des Wasserflusses verwendet wird. Das Wasser kann nur in die vier Himmelsrichtungen (Norden, Osten, Süden, Westen) fließen und nur dann, wenn der Zahlenwert kleiner oder gleich ist.

Für den Aufruf `waterFlow(map, map.length / 2, map.length / 2, Integer.MAX_VALUE);` in `main` wird die Landkarte `map` in Tabelle 2 generiert.

9	5	-1	9	6	11	7	8	9
9	6	-1	-1	6	11	-1	-1	7
6	9	8	-1	10	11	-1	-1	6
9	7	9	-1	-1	-1	-1	6	5
9	12	-1	-1	-1	-1	-1	7	8
9	12	-1	-1	-1	9	-1	7	8
6	-1	-1	-1	9	10	-1	-1	-1
5	-1	-1	-1	11	10	8	9	9
-1	-1	6	6	9	10	10	10	9

Tabelle 2: Landkarte mit dem Ergebnis der Wassersimulation. Wasserflächen sind mit -1 gekennzeichnet.

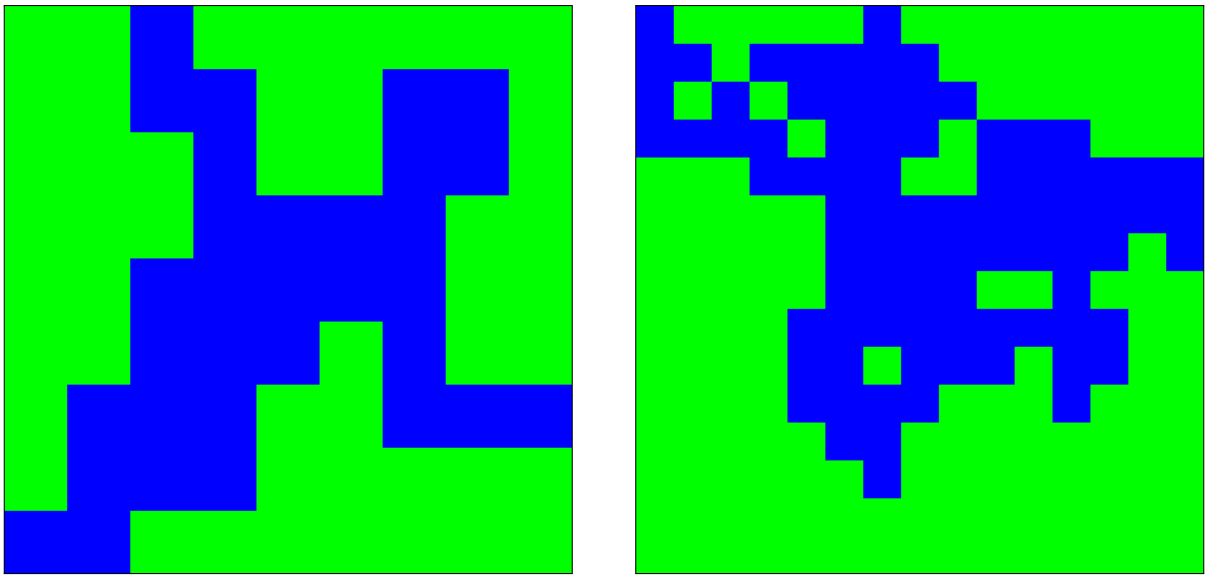
Hier sind alle gefundenen Wege, wohin das Wasser von der Quelle aus fließen kann, mit -1 gekennzeichnet. Die Suche beginnt nach dem Aufruf in `main` in der Mitte der Karte. Nun wird mit weiteren rekursiven Aufrufen der Methode `waterFlow` in alle vier Himmelsrichtungen nach einem möglichen Pfad für das Wasser weiter gesucht. Bevor ein Aufruf in eine der vier Himmelsrichtungen stattfinden kann, muss für jede Richtung überprüft werden, ob mit diesem Schritt bereits der Rand der Karte erreicht wird. Nur wenn der Rand nicht erreicht wird, dann darf in diese Richtung weiter gesucht werden. In der nächsten Rekursionsstufe wird dann überprüft, ob der aktuelle Höhenwert kleiner oder gleich ist als der von der vorherigen Rekursionsstufe. Dazu gibt es den Parameter `prevValue`, mit dem der Höhenwert der vorherigen Rekursionsstufe in die aktuelle Rekursionsstufe mitgegeben wird.

(!) Sie dürfen für die Methode `waterFlow` 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.

- Implementieren Sie eine Methode `drawMap`:

```
void drawMap(int[][] map)
```

Diese Methode zeichnet die Karte `map` (symmetrisches zweidimensionales Array) in ein 450×450 Pixel großes `StdDraw`-Fenster und visualisiert den Wasserfluss. In Abbildung 4a wird das Ergebnis für die Werte in Tabelle 2 gezeigt. Für alle Werte -1 in der Landkarte `map` wird ein blaues Quadrat gezeichnet und für die restlichen Zahlenwerte ein grünes Quadrat. Die Methode soll mit verschieden großen zweidimensionalen Karten umgehen können und die Größe der Quadrate entsprechend anpassen, sodass das gesamte `StdDraw`-Fenster ausgenutzt wird. In Abbildung 4b wird das Ergebnis einer Wasserfluss-Simulation gezeigt, für



(a) (b)

Abbildung 4: Landkarte der Größe a) 9×9 und b) 15×15 mit den eingezeichneten Landregionen (grün) und dem Wasserfluss (blau).

die eine zufällig generierte Landkarte verwendet wurde. Für die Generierung von zufälligen Landkarten können Sie die vorhandene Methode `genMap(...)` verwenden.