Aufgabenblatt 5

Kompetenzstufe 1 & Kompetenzstufe 2

Allgemeine Informationen zum Aufgabenblatt:

- Die Abgabe erfolgt in TUWEL. Bitte laden Sie Ihr IntelliJ-Projekt bis spätestens **Mittwoch, 11.12.2024 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:

- 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. Sind alle Zeilen des Arrays gleich lang, dann werden alle Zeilen um einen Index nach oben (kleineren Index) verschoben und die erste Zeile ganz hinten eingefügt.

Dafür wird das Array `workArray` umgebaut und kein neues Array erstellt. Sie dürfen aber innerhalb der Methode eine temporäre Array-Variable anlegen.

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[][]{`
`{1,5,6,7},`
`{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`

`shiftLines(new int[][]{`
`{3,4,1},`
`{6,2,5},`
`{9,7,8}})`
	`6 2 5`
	`9 7 8`
	`3 4 1`

Aufgabe 2 (1 Punkt)

Implementieren Sie folgende Aufgabenstellung:

- Implementieren Sie eine Methode `reformatArray`:

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

  Diese Methode ordnet in jeder Zeile von `workArray` die geraden und ungeraden Werte. Das heißt, dass in der umgebauten neuen Zeile zuerst alle geraden Werte, gefolgt von allen ungeraden Werten eingetragen werden. Die Werte müssen innerhalb der geraden und ungeraden Werte nicht sortiert werden, das heißt die Reihenfolge der Werte bleibt erhalten. Zusätzlich wird die Zeile um eins länger. An der letzten Position wird die Summe aller Werte in einer Zeile eingefügt. Hinweis: Sie dürfen innerhalb der Methode für die neuen Zeilen ein Hilfsarray verwenden.

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

  Beispiele:

Aufruf	Ergebnis

`reformatArray(new int[][]{`
`{3, 4, 6, 9},`
`{1, 0, 2},`
`{3},`
`{2},`
`{4, 6, 8, 10},`
`{1, 3, 5, 7, 9},`
`{2, 7, 9, 2, 2}})`
	`4 6 3 9 22`
	`0 2 1 3`
	`3 3`
	`2 2`
	`4 6 8 10 28`
	`1 3 5 7 9 25`
	`2 2 2 7 9 22`

`reformatArray(new int[][]{`
`{0},`
`{1},`
`{2},`
`{3, 4},`
`{5, 7, 6},`
`{6, 8, 9}})`
	`0 0`
	`1 1`
	`2 2`
	`4 3 7`
	`6 5 7 18`
	`6 8 9 23`

`reformatArray(new int[][]{`
`{0, 1, 2, 3, 4, 5, 6, 7, 8, 9},`
`{2, 2, 2, 2, 2, 4, 4, 4, 4, 4}})`
	`0 2 4 6 8 1 3 5 7 9 45`
	`2 2 2 2 2 4 4 4 4 4 30`

Aufgabe 3 (2 Punkte)

Erweitern Sie die Aufgabe um folgende Funktionalität:
Bei dieser Aufgabe wird zuerst in einer Methode berechnet, wie viele Tiere in einem Gehege stehen. Dies wird zufällig mit einer gewissen Wahrscheinlichkeit passieren. Anschließend wird in einer zweiten Methode berechnet, wie dicht die Tiere zueinander stehen (8er-Nachbarschaft).

- Implementieren Sie eine Methode `genAnimalCompound`:

  ```
  boolean[][] genAnimalCompound(int compoundSize, float probability)
  ```

  Diese Methode hat den Parameter `compoundSize`, der angibt wie groß die Fläche des Tiergeheges ist. In diesem Fall wird dieses quadratisch angenommen. Der zweite Parameter `probability` gibt an, mit welcher Wahrscheinlichkeit auf ein Feld im Gehege ein Tier steht. Die Wahrscheinlichkeit wird für jedes der `compoundSize`×`compoundSize` Felder berechnet und in ein Array vom Typ `boolean` eingetragen. Es wird `true` eingetragen, wenn auf einem Feld ein Tier steht, ansonsten `false`. Am Ende wird dieses Array zurückgegeben. Sie können das Feld mit Hilfe der Hilfsmethode `printCompound` ausgeben. Die Sterne `'*'` stehen jeweils für ein Tier.

  Vorbedingungen: `compoundSize > 0` und `probability` ist im Intervall [0.0, 1.0[.

  Beispiele:

`compoundSize = 8`		
1) `probability = 0.2`	2) `probability = 0.5`	3) `probability = 0.8`

`0 * * 0 0 0 0 0`
`0 0 0 0 0 0 0 0`
`0 0 0 0 0 0 0 0`
`0 * 0 0 0 0 0 0`
`0 * 0 0 0 * 0 0`
`0 0 * 0 * 0 0 0`
`0 0 0 0 0 0 * *`
`0 0 0 0 0 0 * 0`
	`0 * * 0 * 0 * 0`
	`* * 0 0 * * 0 0`
	`* * 0 0 0 * * *`
	`0 * * * * 0 0 0`
	`0 * * 0 * * 0 *`
	`0 * * 0 * * 0 0`
	`* * * 0 * * * *`
	`* 0 0 0 * 0 * *`
		`* * * * * 0 * 0`
		`* * * * * * * 0`
		`* * 0 * 0 * * *`
		`0 * * * * * 0 *`
		`0 * * * * * * *`
		`* * * * * * * 0`
		`* * * * * * * *`
		`* * * * * * * *`

- Implementieren Sie eine Methode `calcAnimalDensity`:

```
int[][] calcAnimalDensity(boolean[][] animalCompound)
```

Die Methode verwendet das zweidimensionale `boolean`-Array `animalCompound` und berechnet, wie viele Tiere (*density*) sich in der direkten Nachbarschaft jedes Feldes befinden. Es wird ein zweidimensionales `int`-Array mit derselben Größe wie `animalCompound` erzeugt, wo für jede Position von `animalCompound` die Anzahl der Tiere ermittelt wird, die direkt an das betrachtete Feld angrenzen. Das betrachtete Feld selbst wird ebenfalls mitberechnet. Anbei sind Beispiele, die 3×3 Tiergehege betrachten und die berechneten *density*-Werte:

`animalCompound`	Ergebnis density	`animalCompound`	Ergebnis density

`0 0 0`
`* 0 0`
`0 0 0`
	`1 1 0`
	`1 1 0`
	`1 1 0`
		`0 0 0`
		`* 0 0`
		`0 * 0`
			`1 1 0`
			`2 2 1`
			`2 2 1`

`0 * 0`
`0 * 0`
`0 0 0`
	`2 2 2`
	`2 2 2`
	`1 1 1`
		`0 * 0`
		`* * 0`
		`0 0 *`
			`3 3 2`
			`3 4 3`
			`2 3 2`

Für den Rand müssen Sie die Werte ebenfalls berechnen, aber hier nehmen Sie an, dass es einen zusätzlichen unsichtbaren Rand außerhalb des Tiergeheges gibt, der mit Nullen gefüllt ist.

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

Beispiele der *density* zu den Tiergehegen von oben:

1) Ergebnis density	2) Ergebnis density	3) Ergebnis density

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

Aufgabe 4 (2 Punkte)

Implementieren Sie folgende Aufgabenstellung:

- Bei dieser Aufgabe geht es um die Implementierung eines kleinen Zeichenprogramms *Mini-Paint*. In *Mini-Paint* können mit Mausklicks Linien gezeichnet werden, wodurch auch geschlossene Flächen entstehen können. Diese geschlossenen Flächen sollen mit verschiedenen Farben gefüllt werden können. Ein Grundgerüst, das den Zeichenbereich mit der Farbpalette zeichnet, ist in `main` vorgegeben (Abbildung 1a). Erweitern Sie `main` so, dass Sie ein

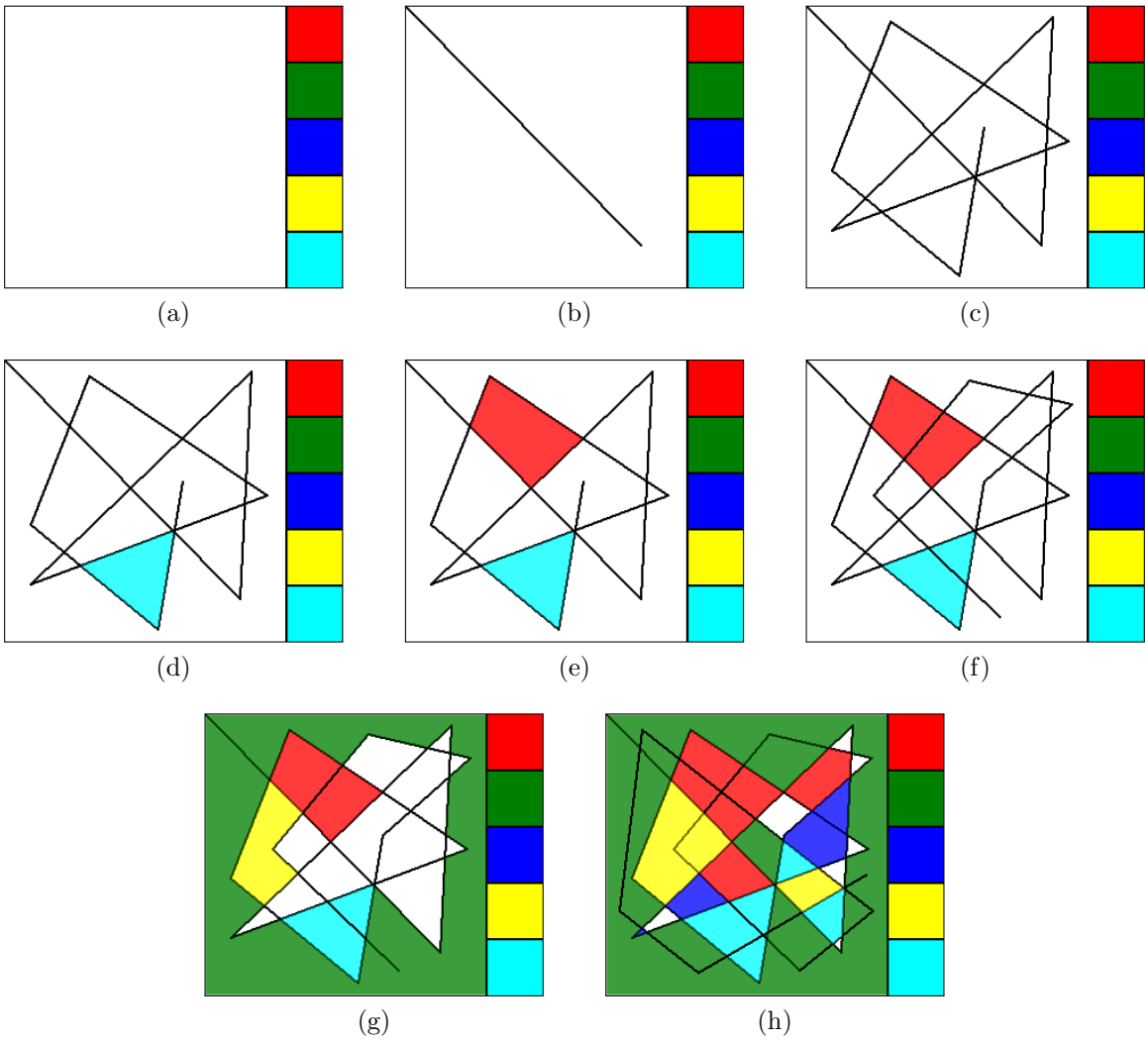

Abbildung 1: Mini-Paint Oberfläche in verschiedenen Zuständen.

funktionierendes Zeichenprogramm erhalten, das folgende Anforderungen erfüllt:

  - Der weiße Zeichenbereich hat eine Größe von 250×250 Pixel.
  - Die Farbpalette am rechten Rand hat eine Größe von 50×250 Pixel, wobei jedes der Farbvierecke eine Größe von 50×50 Pixel hat.
  - Wird im Zeichenbereich mit der Maus geklickt, dann werden die Informationen über diesen Mausklick mit `nextMouseClickEvent()` ausgelesen und in einem `MouseClickEvent`

gespeichert. Mit den Methoden `getY()` und `getX()` von `MousClickEvent` werden die Koordinaten des Mausklicks ausgelesen.

Es gibt zwei int-Arrays `yClick` und `xClick`, die die Koordinaten von zwei Mausklicks speichern. Zu Beginn sind alle Einträge 0 und dadurch wird nach dem ersten Klick eine Linie vom Ursprung (0,0) (linke obere Ecke) bis zum Klickpunkt gezeichnet (Abbildung 1b). Um nach einem Mausklick eine Linie zu zeichnen, wird die Methode `paintLine(...)` aufgerufen. Danach kann erneut im Zeichenbereich geklickt werden, um eine weitere Linie zu zeichnen, die dann vom Ende der vorherigen Linie bis zum aktuellen Klickpunkt verläuft (Abbildung 1c). Bitte beachten Sie hier, dass in den Arrays (`yClick` und `xClick`) an Indexposition 0 die Koordinaten des vorangegangenen Mausklicks stehen und an Indexposition 1 die Koordinaten des aktuellen Mausklicks. Das Linienzeichnen kann beliebig oft wiederholt werden.

- Soll keine weitere Linie gezeichnet, sondern eine Fläche mit einer bestimmten Farbe gefüllt werden, dann muss auf eine der 5 Farbflächen geklickt werden. Dann wird keine Linie gezeichnet, sondern die Zeichenfarbe auf die entsprechende Farbe gesetzt und das Zeichenprogramm in den Zustand des *Flächenfüllens* versetzt. Sie müssen aufgrund der Klickkoordinaten unterscheiden, welche Farbe ausgewählt werden soll. Wenn nach Auswahl der Farbe auf den Zeichenbereich geklickt wird, dann wird nicht eine Linie gezeichnet, sondern vom entsprechenden Klickpunkt aus eine zusammenhängende Fläche mit der ausgewählten Farbe gefüllt (Abbildung 1d). Dazu wird die Methode `floodFill(...)` aufgerufen. Nach diesem Klick zum Füllen einer Fläche ist das Zeichenprogramm wieder im Zustand des *Linienzeichnens*. Dann können wieder im Zeichenbereich Linien gezeichnet werden und es wird vom Endpunkt der letzten Linie weiter gezeichnet. (Abbildung 1f). Soll erneut eine Fläche gefüllt werden, muss wieder zuerst eine Farbe ausgewählt werden. (Abbildung 1e).
- Eine bereits eingefärbte Fläche kann nicht neu gefüllt werden.

(!) Es dürfen in `main` zusätzliche Variablen (auch außerhalb der `while`-Schleife) angelegt werden.

- Gegeben ist eine Methode `paintLine`:

```
void paintLine(int[][] picArray, int[] yClick, int[] xClick)
```

Diese bereits vorgegebene Methode nimmt ein zweidimensionales ganzzahliges Array `picArray` entgegen, das das aktuelle Bild mit allen gezeichneten Linien und gefüllten Farbflächen enthält. Angelehnt an das CodeDraw-Fenster befindet sich der Ursprung (0,0) von `picArray` ebenfalls links oben. Die Parameter `yClick` und `xClick` beinhalten die Koordinaten von zwei Mausklicks, die für das Zeichnen einer Linie benötigt werden. Es wird eine Linie von (`yClick[0]`,`xClick[0]`) bis (`yClick[1]`,`xClick[1]`) gezeichnet und alle Punkte (Pixel) zwischen diesen beiden Punkten schwarz gezeichnet und an der entsprechenden Stelle des `picArray` mit dem Wert 1 eingetragen. Es können auch über gefüllte Farbflächen Linien gezeichnet werden.

- Implementieren Sie eine rekursive Methode `floodFill`:

  ```
  void floodFill(int[][] picArray, int sy, int sx)
  ```

  Diese Methode bekommt ein zweidimensionales ganzzahliges Array `picArray` übergeben, das das aktuelle Bild mit allen gezeichneten Linien und gefüllten Farbflächen enthält. Die Parameter `sy` und `sx` entsprechen beim ersten Aufruf den Koordinaten des Mausklicks. Ausgehend von den Koordinaten des Mausklicks wird in alle 4 Richtungen (oben, unten, links, rechts) rekursiv weiter gesucht, ob noch ein weißer Wert vorhanden ist. Sollte das der Fall sein, dann wird dieses Pixel mit der gesetzten Farbe eingefärbt (CodeDraw-Fenster) und die entsprechende Koordinate im `picArray` als *bemalt* (Wert wird auf 1 gesetzt) gekennzeichnet. Danach werden wieder die Nachbarn von diesem Pixel betrachtet. So arbeitet sich der Algorithmus in alle 4 Richtungen weiter, bis dieser auf Grenzen trifft. Diese Grenzen können gezeichnete Linien bzw. die Grenzen des Zeichenfensters sein (Abbildung 1g oder 1h).

  Hinweis: Die Stackgröße ist für dieses Beispiel erhöht und voreingestellt. Wenn diese Einstellung nicht vorhanden ist, oder verändert werden möchte, dann können Sie *Edit Configurations* von Aufgabe 4 rechts oben in IntelliJ öffnen. Dort müssen Sie bei *VM options* folgenden Parameter hinzufügen: `-Xss16m`. Mit dieser Einstellung wird die Stackgröße auf 16 MB erhöht und ist für diese Aufgabe genug.