Aufgabenblatt 4

Kompetenzstufe 1 & Kompetenzstufe 2

Allgemeine Informationen zum Aufgabenblatt:

- Die Abgabe erfolgt in TUWEL. Bitte laden Sie Ihr IntelliJ-Projekt bis spätestens **Mittwoch, 27.11.2024 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**.
- 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 `Integer`, es sei denn in den Hinweisen zu den einzelnen Aufgaben ist etwas anderes angegeben.
- 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:

- Code-Verstehen mit Arrays
- Verwendung von ein- und zweidimensionalen Arrays

Aufgabe 1 (1 Punkt)

Beantworten Sie die folgenden Fragen bitte in dem dafür vorgesehenen Bereich (ganz unten) im Code von *Aufgabe1.java*. Sie können die Antworten zu den Zusatzfragen direkt darunter angeben. Änderungen im Code sind nicht durchzuführen, außer für die erste Frage, wo es darum geht, die Exception zu vermeiden:

a) Warum kommt es in der Methode `printArray` (Zeile 6) zu einem Fehler (Exception)? Korrigieren Sie den Fehler, sodass die Methode keine Exception wirft und die Ausgabe des Arrays durchführt. In welcher Reihenfolge die Elemente ausgegeben werden, ist an dieser Stelle nicht relevant und muss daher auch nicht geändert werden.

b) Wieso hat die Methode `fillArray` keinen Rückgabewert, obwohl ein Array befüllt werden soll?

c) Der Aufruf `printContentFilteredArray(filledArray)` in Zeile 47 soll im Array `filledArray` alle durch 9 teilbaren Zahlen auf -1 setzen und das Array ausgeben. Warum aber ergibt der Aufruf `printArray(filledArray)` in Zeile 48 dann ebenfalls dieses gefilterte Array, obwohl innerhalb der Methode `printContentFilteredArray` anscheinend auf einer Kopie gearbeitet wurde?

d) In Zeile 50 wird in `filledArray` an der Stelle 0 der Wert 123 gespeichert. Danach wird in Zeile 53 die Methode `fillArrayWithNewContent` aufgerufen, welche ein neues Array mit neuem Inhalt erzeugen soll. Wie in Zeile 39 gezeigt, befindet sich ein neuer Arrayinhalt in `workArray`, aber wieso ergibt der Aufruf in Zeile 54 wiederum den alten Arrayinhalt?

Zusatzfrage(n): Gehen Sie hier von eindimensionalen Arrays aus!

1. Welchen Datentyp muss der Indexausdruck haben, mit dem die Position in einem Array bestimmt wird?
2. Wie kann die Länge eines Arrays verändert werden?
3. Wie gehen Sie vor, wenn Sie ein `int`-Array kopieren müssen?
4. Ist es sinnvoll, zwei Arrays mit `"=="` zu vergleichen? Was passiert im Detail bei einem Vergleich mit `"=="`?

Aufgabe 2 (1 Punkt)

Implementieren Sie folgende Aufgabenstellung:

(!) Implementieren Sie die folgenden Aufgabenstellungen direkt in `main`. Sie können sich aber für die Ausgabe der Arrays Hilfsmethoden schreiben.

a) Erstellen Sie ein eindimensionales `short`-Array und initialisieren Sie es mit den Werten `{3, 6, 24, 31, 35, 44, 67, 73, 89, 92}`. Geben Sie anschließend jeden Arrayeintrag getrennt durch Semikolon auf der Konsole aus. Zusätzlich geben Sie vorne und hinten noch ein Pipe-Zeichen (Oder-Zeichen) aus.
Erwartete Ausgabe:
`|3;6;24;31;35;44;67;73;89;92|`

b) Erstellen Sie ein eindimensionales `char`-Array und initialisieren Sie es mit den Werten `{'a', '8', '8', 'G', '2', ':', ':', ':', 'F', '7', 'b', 'b', '-', 'T'}`. Nach der Erstellung und Initialisierung des Arrays soll ein neues Array erstellt werden, das die Inhalte des zuvor erstellten Arrays enthält und zusätzlich noch das Zeichen `'+'` zwischen zwei benachbarten gleichen Zeichen im Array. Die Implementierung darf dabei nicht auf den konkreten Werten des ursprünglichen Arrays basieren, sondern muss auch für unterschiedliche Arrays (unterschiedlicher Inhalt, unterschiedliche Länge) funktionieren. Geben Sie anschließend alle Elemente des neuen Arrays nebeneinander getrennt durch Leerzeichen auf der Konsole aus.
Erwartete Ausgabe für das gegebene Array:
`a 8 + 8 G 2 : + : + : F 7 b + b - T`

c) Erstellen Sie ein eindimensionales ganzzahliges Array der Länge 20 und initialisieren Sie es mittels Schleife mit den Werten von 19 bis 0. Geben Sie anschließend alle Elemente des Arrays in umgekehrter Reihenfolge, getrennt durch Rufzeichen, aus. Geben Sie das Array einmal mit Hilfe einer for-Schleife und einmal mit Hilfe einer while-Schleife auf der Konsole aus. Zusätzlich soll zur Unterscheidung der Ausgabe der String `"for-Schleife:"` bzw. `"while-Schleife:"` bei der jeweiligen Schleife ausgegeben werden.
Erwartete Ausgabe:
`for-Schleife: 0!1!2!3!4!5!6!7!8!9!10!11!12!13!14!15!16!17!18!19`
`while-Schleife: 0!1!2!3!4!5!6!7!8!9!10!11!12!13!14!15!16!17!18!19`

Aufgabe 3 (1 Punkt)

Implementieren Sie folgende Aufgabenstellung:

a) Implementieren Sie eine Methode `replaceValues`:

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

Diese Methode ermittelt den kleinsten und größten Wert innerhalb des Arrays `workArray`. Danach wird mit Hilfe der absoluten Differenz berechnet, ob ein Wert innerhalb des Arrays näher am kleinsten (Minimum), oder näher am größten (Maximum) ermittelten Wert liegt. Liegt der Wert eines Arrayelements näher am kleinsten Wert, dann wird an diese Stelle des Arrays der Minimumwert gespeichert. Liegt der Wert eines Arrayelements jedoch näher beim größten Wert, dann wird das Arrayelement an dieser Stelle durch das Maximum ersetzt. Ist die Differenz gleich, dann wird der Minimumwert an dieser Stelle im Array gespeichert.
Vorbedingungen: `workArray != null` und `workArray.length > 0`.

Beispiele:
`int[] array1 = new int[]{12, 3, 15, 18, 22, 9, 5, 8, 16, 21};`
`replaceValues(array1)` führt zu
`[3, 3, 22, 22, 22, 3, 3, 3, 22, 22]`
`int[] array2 = new int[]{12, 10, 27, 18, 22, 60, 35, 36, 16, 21};`
`replaceValues(array2)` führt zu
`[10, 10, 10, 10, 10, 60, 10, 60, 10, 10]`
`int[] array3 = new int[]{4, 7, 7, 4};`
`replaceValues(array3)` führt zu `[4, 7, 7, 4]`
`int[] array4 = new int[]{-5, -1, 0, 1, 5};`
`replaceValues(array4)` führt zu `[-5, -5, -5, 5, 5]`
`int[] array5 = new int[]{1, 2, 3};`
`replaceValues(array5)` führt zu `[1, 1, 3]`
`int[] array6 = new int[]{7};`
`replaceValues(array6)` führt zu `[7]`

Aufgabe 4 (1 Punkt)

Implementieren Sie folgende Aufgabenstellung:

- Implementieren Sie eine Methode `generateFilled2DArray`:

  ```
  int[][] generateFilled2DArray(int n)
  ```

  Die Methode erzeugt ein zweidimensionales Array der Größe `n`×`n` und befüllt dieses mit Zahlen wie in den nachfolgenden Beispielen gezeigt. Ausgehend von der Hauptdiagonalen, die mit 1 befüllt ist, wird nach unten und rechts gehend bis zum Rand des Arrays jeder weitere Arrayeintrag um 1 erhöht eingetragen. Zusätzlich wird nach der Befüllung um das mittlere Element herum eine -1 in das Array eingefügt (erst ab `n >= 3` möglich).

  Vorbedingung: `n >= 1` und `n % 2 == 1`.

  Beispiele:

  `generateFilled2DArray(1)` erzeugt →

  ```
  1
  ```

  `generateFilled2DArray(3)` erzeugt →

  ```
  -1 -1 -1
  -1 1 -1
  -1 -1 -1
  ```

  `generateFilled2DArray(5)` erzeugt →

  ```
  1 2 3 4 5
  2 -1 -1 -1 4
  3 -1 1 -1 3
  4 -1 -1 -1 2
  5 4 3 2 1
  ```

  `generateFilled2DArray(7)` erzeugt →

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

Aufgabe 5 (1 Punkt)

Implementieren Sie folgende Aufgabenstellung:

- Implementieren Sie eine Methode `generateExtendedArray`:

  ```
  int[][] generateExtendedArray(int[] inputArray)
  ```

  Diese Methode erstellt ein ganzzahliges zweidimensionales Array, bei dem die Größe aus dem Array `inputArray` abgeleitet wird. Das `inputArray` hat genau die Länge drei. Der erste Wert `inputArray[0]` gibt die Länge der ersten Zeile an. Der zweite Wert `inputArray[1]` gibt die Länge der letzten Zeile an. Jede Zeile wird um ein Element länger. Das heißt, dass es `inputArray[1]-inputArray[0]+1` Zeilen gibt. Der dritte Wert `inputArray[2]` gibt an, mit welchem Wert das erste Element in der ersten Zeile befüllt wird. Danach werden die einzelnen Elemente im Array von links nach rechts und von oben nach unten verlaufend befüllt, wobei die Zahlen kontinuierlich um 1 erhöht werden (siehe Beispiele). Anschließend wird das neu erstellte Array zurückgegeben.

  Vorbedingungen: `inputArray != null`, `inputArray.length = 3`, `inputArray[0] > 0`, `inputArray[1] > 1` und `inputArray[0] < inputArray[1]`.

  Beispiele:

  `generateExtendedArray(new int[]{1, 2, 10})` erzeugt →

  ```
  10
  11 12
  ```

  `generateExtendedArray(new int[]{3, 6, 8})` erzeugt →

  ```
  8  9  10
  11 12 13 14
  15 16 17 18 19
  20 21 22 23 24 25
  ```

  `generateExtendedArray(new int[]{6, 8, 4})` erzeugt →

  ```
  4  5  6  7  8  9
  10 11 12 13 14 15 16
  17 18 19 20 21 22 23 24
  ```

Aufgabe 6 (2 Punkte)

Implementieren Sie folgende Aufgabenstellung:

- Implementieren Sie eine Methode `generateReformattedArray`:

  ```
  int[][] generateReformattedArray(int[][] inputArray)
  ```

  Diese Methode erstellt ein ganzzahliges zweidimensionales Array und befüllt es mit Daten, welche den Inhalt des Arrays `inputArray` in einer anderen Form darstellen. Im `inputArray` stehen komprimierte Daten, die in der Methode decodiert in einem neuen Array dargestellt werden sollen.

  Jede Zeile von `inputArray` enthält an geraden Indizes eine Anzahl und an ungeraden Indizes einen Wert. Es wird in der neuen Zeile der Wert so oft hintereinander eingefügt, wie durch die Anzahl angegeben wird. Dies wird solange gemacht, bis es kein Wertepaar in einer Zeile mehr gibt. Zum Beispiel wird eine Zeile | 4 | 1 | 2 | 0 | 1 | 1 | im neuen Array dargestellt als | 1 | 1 | 1 | 1 | 0 | 0 | 1 |. In der originale Zeile steht 4 (Anzahl) gefolgt von einer 1 (Wert), was dazu führt, dass in der neuen Arrayzeile viermal eine 1 eingefügt wird. Danach kommt eine 2 gefolgt von einer 0 und daher wird in der neuen Zeile zweimal eine 0 eingefügt. Zuletzt steht eine 1 gefolgt von einer 1, was einmal einer 1 im Zielarray entspricht.

  Am Ende wird das neu erstellte Array zurückgegeben. Sie finden unten noch weitere Beispiele.

  Vorbedingungen: `inputArray != null`, `inputArray.length > 0` und es gilt für alle gültigen `i`, dass `inputArray[i].length > 0` und `inputArray[i].length % 2 == 0` ist. Für alle Indizes innerhalb einer Zeile mit `i % 2 == 0` gilt, `inputArray[i] > 0` und für alle `i` einer Zeile mit `i % 2 == 1` gilt, `inputArray[i] == 1` oder `inputArray == 0`.

  Beispiele:

  ```
  generateReformattedArray(new int[][]{
  {1, 1, 1, 0, 2, 1},
  {1, 0, 2, 1},
  {1, 0, 1, 1, 1, 0, 2, 1},
  {3, 0, 1, 1, 1, 0},
  {1, 1, 1, 0},
  {5, 1}}) erzeugt →


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

`generateReformattedArray(new int[][]{`

```
{1, 1, 1, 0, 2, 1, 4, 0},
{1, 0, 5, 1, 2, 0},
{6, 0, 2, 1},
{1, 1, 7, 0},
{4, 0, 2, 1, 1, 0, 1, 1},
{8, 0},
{7, 0, 1, 1}}) erzeugt →
```

```
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
```

`generateReformattedArray(new int[][]{`

```
{1, 0},
{1, 1},
{2, 0},
{1, 0, 1, 1},
{1, 1, 1, 0},
{2, 1}}) erzeugt →
```

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

`generateReformattedArray(new int[][]{{12, 0}}) erzeugt →`

```
0 0 0 0 0 0 0 0 0 0 0 0
```