Aufgabenblatt 4

Kompetenzstufe 1 & Kompetenzstufe 2

Allgemeine Informationen zum Aufgabenblatt:

- Die Abgabe erfolgt in TUWEL. Bitte laden Sie Ihr IntelliJ-Projekt bis spätestens **Mittwoch, 06.12.2023 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**.
- 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
- Rekursion mit eindimensionalen Arrays

Aufgabe 1 (1 Punkt)

Die folgenden Fragen beantworten Sie bitte in dem dafür vorgesehenen Bereich (ganz unten) im Code von *Aufgabe1.java*. Die Zusatzfragen können dann anschließend daran beantwortet werden. 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 auch die Funktionalität von `printArray` (Ausgabe des gesamten Arrays in einer Zeile auf der Konsole) korrekt implementiert ist.

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

c) Der Aufruf der Methode `printContentFilteredArray(filledArray)` in Zeile 47 soll alle durch 6 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 999 eingefügt. 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:

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

a) Erstellen Sie ein eindimensionales ganzzahliges Array der Länge 15 und initialisieren Sie es mit Kubikzahlen beginnend bei $2^3 = 8$ ($3^3 = 27$, $4^3 = 64$, ...). Setzen Sie alle Elemente des Arrays auf 0, die durch 9 teilbar sind. Geben Sie anschließend alle Elemente des Arrays nebeneinander getrennt durch Leerzeichen auf der Konsole aus.
Erwartete Ausgabe:
`8 0 64 125 0 343 512 0 1000 1331 0 2197 2744 0 4096`

b) Erstellen Sie ein eindimensionales `char`-Array und initialisieren Sie es mit den Werten `{'a', '8', 'U', '3', ':', '9', 'd', 'F', '-', '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 `'Z'` vor jedem Vorkommen einer Ziffer im Array. Geben Sie anschließend alle Elemente des neuen Arrays nebeneinander getrennt durch Leerzeichen auf der Konsole aus.
Erwartete Ausgabe:
`a Z 8 U Z 3 : Z 9 d F - T`

c) Erstellen Sie ein eindimensionales ganzzahliges Array der Länge 20 und initialisieren Sie es mittels Schleife mit den Werten von 1 bis 20. Geben Sie anschließend jedes zweite Elemente des Arrays in umgekehrter Reihenfolge getrennt durch Doppelpunkte aus. Geben Sie das Array einmal mit Hilfe einer while-Schleife und einmal mit Hilfe einer for-Schleife auf der Konsole aus. Zusätzlich soll zur Unterscheidung der Ausgabe der String `while-Schleife:` bzw. `for-Schleife:` bei der jeweiligen Schleife ausgegeben werden.
Erwartete Ausgabe:
`for-Schleife: 20:18:16:14:12:10:8:6:4:2`
`while-Schleife: 20:18:16:14:12:10:8:6:4:2`

Aufgabe 3 (1 Punkt)

Implementieren Sie folgende Aufgabenstellung:

(!) Für die folgenden Methoden dürfen keine Klassenvariablen oder zusätzliche eigene Hilfsmethoden verwendet werden. Die vorgegebenen Methodenköpfe dürfen nicht erweitert oder geändert werden.

a) Implementieren Sie eine Methode `genRandomArray`:

```
int[] genRandomArray(int length, int maxNumber)
```

Diese Methode generiert ein eindimensionales ganzzahliges Array der Länge `length`, befüllt dieses mit Werten bestehend aus Zufallszahlen und aus diesen Zufallszahlen berechneten Zahlen. Zum Beispiel wird an der Indexposition 0 eine Zufallszahl aus dem Intervall von `[0, maxNumber[` generiert und eingetragen. An der Indexposition 1 wird die zuvor generierte Zufallszahl plus eins eingetragen und an der Indexposition 2 die zuvor generierte Zufallszahl plus zwei. Die nächste neue Zufallszahl wird für Indexposition 3 generiert. Indexposition 4 und 5 bekommen wieder die um eins bzw. zwei größere Zahl als die generierte Zufallszahl an Indexposition 3. In dieser Weise wird das gesamte Array befüllt. Anschließend wird das generierte Array zurückgegeben.
Vorbedingungen: `length` > 0, `length` $\%\ 3 == 0$ und `maxNumber` > 0.

Beispiel:
`int[] array1 = genRandomArray(21, 50);` führt zu
`[39, 40, 41, 6, 7, 8, 45, 46, 47, 7, 8, 9, 2, 3, 4, 46, 47, 48, 2, 3, 4]`

b) Implementieren Sie eine Methode `replaceValues`:

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

Diese Methode berechnet den Mittelwert aller Arrayeinträge als ganzzahligen (Nachkommaanteil abschneiden) Wert. Danach werden alle Werte innerhalb des Arrays vom Index 0 beginnend bis zum Index `idx` des Arrays, die größer gleich als der Mittelwert sind, auf den Maximalwert des Arrays gesetzt. Für alle Werte, die unterhalb des Mittelwertes liegen, wird der Arrayeintrag mit dem Minimalwert des Arrays ersetzt.
Vorbedingungen: `workArray != null`, `workArray.length > 0`, `idx` $\geq$ `0` und `idx` $<$ `workArray.length`.

Beispiele:
`int[] array2 = new int[]{12, 3, 15, 18, 22, 9, 5, 8, 16, 21};`
`replaceValues(array2, 4)` führt zu
`[22, 3, 22, 22, 22, 9, 5, 8, 16, 21]`
`int[] array3 = new int[]{12, 3, 15, 18, 22, 9, 5, 8, 16, 21};`
`replaceValues(array3, 9)` führt zu
`[22, 3, 22, 22, 22, 3, 3, 3, 22, 22]`
`int[] array4 = new int[]{21, 14, 12, 17};`
`replaceValues(array4, 2)` führt zu `[21, 12, 12, 17]`
`int[] array5 = new int[]{3, 4, 6, 7};`
`replaceValues(array5, 2)` führt zu `[3, 3, 7, 7]`

Aufgabe 4 (1 Punkt)

Implementieren Sie folgende Aufgabenstellung:

(!) Folgende Methoden dürfen keine Klassenvariablen oder zusätzliche eigene Hilfsmethoden verwenden. Die vorgegebenen Methodenköpfe dürfen nicht erweitert oder geändert werden. Für die Implementierung der Aufgabenstellung dürfen keine Schleifen verwendet werden.

a) Implementieren Sie eine **rekursive** Methode `compareTemperatures`:

```
char[] compareTemperatures(int[] lastYear, int[] currentYear, int index)
```

Diese Methode soll zwei Arrays mit Temperaturwerten vollständig von vorne bis hinten vergleichen. Es wird dazu ein neues Array gleicher Länge erstellt, das je nach Ergebnis des Vergleichs mit `char`-Zeichen beschrieben wird. Ist der Wert an einer Stelle des Arrays `currentYear` größer als der Wert an derselben Stelle im Array `lastYear`, dann wird im neuen Array an dieser Stelle das Zeichen `'>'` eingetragen. Ist aber der Wert in `currentyear` kleiner als der Wert in `lastYear`, dann wird im neuen Array das Zeichen `'<'` eingetragen. Bei gleichen Werten wird im neuen Array das Zeichen `'='` eingefügt. Der Parameter `index` wird für die Position im Array während der Rekursion verwendet und ist beim ersten Aufruf der Methode immer 0, da immer alle Monate eines Jahres verglichen werden.
Vorbedingungen: `lastYear.length == currentYear.length`, `lastYear != null`, `currentYear != null`, `index` ≥ 0 und `index` $<$ `lastYear.length`.

Beispiel:
```
int[] year22 = {-3, 5, 6, 11, 10, 20, 26, 29, 23, 14, 11, -5};
int[] year23 = {-3, 3, 7, 11, 10, 19, 27, 33, 21, 16, 16, -6};
compareTemperatures(year22, year23, 0)
```
führt zu `[=, <, >, =, =, <, >, >, <, >, >, <]`

b) Implementieren Sie eine **rekursive** Methode `shiftLowestHighestValue`:

```
void shiftLowestHighestValue(int[] workArray, int index)
```

Diese Methode soll den größten Wert innerhalb des Array beginnend vom Index `index` bis zum Ende des Arrays ermitteln und diesen an die letzte Stelle im Array schreiben. Gleichzeitig soll der kleinste Wert ermittelt und an die erste Stelle des Arrays geschrieben werden. Immer wenn ein größerer Wert als der letzte Arrayeintrag gefunden wird, dann tauscht dieser den Platz mit dem letzten Arrayeintrag. Wird ein kleinerer Wert als der erste Arrayeintrag gefunden, dann tauscht dieser den Platz mit dem ersten Arrayeintrag. Die ursprüngliche Reihenfolge der Arrayeinträge darf sich verändern. Hinweis: In Abhängigkeit ob der Vergleich vor oder nach dem Rekursionsschritt stattfindet und welchen der beiden Vergleiche (größten/kleinsten Wert) Sie zuerst durchführen, kann das Endergebnis des Arrays unterschiedlich aussehen.
Vorbedingungen: `workArray != null`, `workArray.length > 1`, `index` ≥ 0 und `index` $<$ `workArray.length`.

Beispiele:
`int[] array1 = {32, 46, 22, 38, 41, 24, 12, 28, 33};`
`shiftLowestHighestValue(array1, 0)` führt zu `[12, 41, 22, 38, 33, 24, 28, 32, 46]`
`shiftLowestHighestValue(array1, 4)` führt zu `[12, 46, 22, 38, 33, 24, 28, 32, 41]`
`shiftLowestHighestValue(array1, 5)` führt zu `[12, 46, 22, 38, 41, 24, 28, 32, 33]`
`shiftLowestHighestValue(array1, 7)` führt zu `[28, 46, 22, 38, 41, 24, 12, 32, 33]`
`int[] array2 = {5, 4, 3, 2, 1};`
`shiftLowestHighestValue(array2, 0)` führt zu `[1, 4, 3, 2, 5]`
`int[] array3 = {7, 2, 10, 5, 3};`
`shiftLowestHighestValue(array3, 0)` führt zu `[2, 3, 7, 5, 10]`
`int[] array4 = {2, 1, 3, 9, 6};`
`shiftLowestHighestValue(array4, 0)` führt zu `[1, 2, 3, 6, 9]`

Aufgabe 5 (2 Punkte)

Implementieren Sie folgende Aufgabenstellung:

a) Implementieren Sie eine Methode `generateFilledArray`:

```
int[][] generateFilledArray(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 zeilenweise links oben bei der Stelle (0,0) mit 1 begonnen, bis die Zahl `(n*n+1)/2` eingetragen wird. Die restlichen Stellen des Arrays werden dann mit fortlaufenden Zahlen in absteigender Reihenfolge befüllt, wobei die Zahl `(n*n+1)/2` wiederholt wird, falls n gerade ist.

Vorbedingung: `n > 1`.

Beispiele:

`generateFilledArray(2)` erzeugt →

```
1 2
2 1
```

`generateFilledArray(4)` erzeugt →

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

`generateFilledArray(5)` erzeugt →

```
1  2  3  4  5
6  7  8  9  10
11 12 13 12 11
10 9  8  7  6
5  4  3  2  1
```

`generateFilledArray(7)` erzeugt →

```
1  2  3  4  5  6  7
8  9  10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 24 23 22
21 20 19 18 17 16 15
14 13 12 11 10 9  8
7  6  5  4  3  2  1
```

b) 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 vier und gibt die Dimensionen zweier zweidimensionaler Arrays an. Das erste Array wird angegeben durch die Werte bei Index 0 und 1 von `inputArray` und das zweite Array durch die Werte bei Index 2 und 3. Hier geben die Werte bei Index 0 und 2 die Höhen (Zeilen) und die Werte bei Index 1 und 3 die Breiten (Spalten) der Arrays an. Die beiden Arrays werden aber nicht einzeln erstellt, sondern zu einem gemeinsamen Array zusammengefasst, das möglicherweise unterschiedlich langen Zeilen haben wird. Der Inhalt jeder Zeile dieses neuen Arrays wird beginnend von 1 bis Zeilenlänge durchnummeriert bzw. befüllt (siehe Beispiele). Anschließend wird das neu erstellte Array zurückgegeben.

Vorbedingungen: `inputArray != null`, `inputArray.length = 4` und für alle gültigen `i` gilt `inputArray[i] > 0`.

Beispiele:

`generateExtendedArray(new int[]{4, 3, 2, 6})` erzeugt →

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

`generateExtendedArray(new int[]{2, 4, 1, 5})` erzeugt →

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

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

```
1
1
```

`generateExtendedArray(new int[]{4, 4, 1, 4})` erzeugt →

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

c) Implementieren Sie eine Methode `generateReformattedArray`:

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

Diese Methode erstellt ein ganzzahliges zweidimensionales Array und befüllt es mit Daten aus dem Array `inputArray`, Es werden für das neue Array pro Zeile alle Nullen zum Anfang der Zeile verschoben, gefolgt von allen Einsen in einer Zeile. Das heißt, dass alle Nullen linksbündig stehen, gefolgt von allen Einsen bis zum Ende der Zeile (siehe Beispiele). Am Ende wird das neu erstellte Array zurückgegeben.

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

Beispiele:

`generateReformattedArray(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}})
```

ändert das Array zu →

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

```
generateReformattedArray(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}})
```

ändert das Array zu →

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

```
generateReformattedArray(new int[][]{

{0},
{1},
{0, 0},
{0, 1},
{1, 0},
{1, 1}})
```

ändert das Array zu →

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