Aufgabenblatt 4

Kompetenzstufe 1 & Kompetenzstufe 2

Allgemeine Informationen zum Aufgabenblatt:

- Die Abgabe erfolgt in TUWEL. Bitte laden Sie Ihr IntelliJ-Projekt bis spätestens **Freitag, 06.12.2019 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:

- Code-Verstehen mit Arrays
- Verwendung von eindimensionalen Arrays
- Massendatenverarbeitung und eindimensionale Arrays
- Rekursion mit eindimensionalen Arrays und Strings

Aufgabe 1

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:

1. Warum kommt es in der Methode `printArray` (Zeile 35) zu einem Fehler (Exception)? Korrigieren Sie die 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.

2. Wieso hat die Methode `genArray` keinen Rückgabewert, obwohl ein Array befüllt werden soll?

3. Der Aufruf der Methode `printFilteredArrayContent(filledArray)` in Zeile 47 soll alle durch 3 teilbaren Zahlen auf 0 setzen und das Array ausgeben. Warum aber ergibt der Aufruf `printArray(filledArray)` in Zeile 48 dann ebenfalls dieses gefilterte Array, obwohl innerhalb der Methode `printFilteredArrayContent` auf einer Kopie gearbeitet wurde?

4. In Zeile 50 wird in `filledArray` an der Stelle 0 der Wert 1000 eingefügt. Danach wird in Zeile 53 die Methode `genNewArrayContent` aufgerufen, welche ein neues Array mit neuem Inhalt erzeugen soll. Wie in Zeile 32 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. Müssen Sie ein Array initialisieren?

3. Wie kann die Länge eines Arrays verändert werden?

4. Wie gehen Sie vor, wenn Sie ein int-Array kopieren müssen?

5. Beginnt die Indexzählung eines Arrays immer bei 0?

6. Ist es sinnvoll, zwei Arrays mit `"=="` zu vergleichen? Was passiert im Detail, bei einem Vergleich mit `"=="`?

Aufgabe 2

Implementieren Sie folgende Aufgabenstellung:

- In diesem Beispiel sollen Sie aus einer csv-Datei Wetterdaten einlesen, auswerten und visualisieren. Dazu ist die Datei `weather_data.csv` in Ihrem Projekt bereitgestellt. Sie benötigen für die Aufgabe keinen Scanner und können die Datei mit der Klasse *In*[1] einlesen und verarbeiten. Diese Klasse ist in der Verwendung ähnlich zum Scanner und hat den Vorteil, dass Sie sich um kein Exception-Handling kümmern müssen.

- In dieser csv-Datei befinden sich verschiedenste Wetterdaten der Stadt Wien seit Jänner 1955. Jede Zeile entspricht dem Zeitintervall von einem Monat. Wir wollen aus den Daten die sogenannten Frosttage, Eistage, Sommertage und Hitzetage der letzten 40 Jahre auslesen und visuell darstellen. Die vier Kategorien sind wie folgt definiert:

  Frosttag: $t_{min} < 0.0°\text{C}$ (Anzahl dieser Tage pro Monat in der Spalte `NUM_FROST`).

  Eistag: $t_{max} < 0.0°\text{C}$ (Anzahl dieser Tage pro Monat in der Spalte `NUM_ICE`).

  Sommertag: $t_{max} \geq 25.0°\text{C}$ (Anzahl dieser Tage pro Monat in der Spalte `NUM_SUMMER`).

  Hitzetag: $t_{max} \geq 30.0°\text{C}$ (Anzahl dieser Tage pro Monat in der Spalte `NUM_HEAT`).

  Beispiele:
  - Im Jänner 2005 gab es 14 Frosttage.
  - Im Juli 2012 gab es 11 Hitzetage.

  Wir wollen diese 4 Kategorien von Jänner 1979 (Datensatz ist in Zeile 290 zu finden) bis Dezember 2018 (Datensatz ist in Zeile 769 zu finden) betrachten. Für das Auslesen, Verarbeiten und Darstellen sind die folgenden drei Methoden zu implementieren:

- Implementieren Sie eine Methode `readFileData`:

  ```
  String[] readFileData(String fileName, int lineStart, int lineEnd)
  ```

  Der Parameter `fileName` gibt die einzulesende Datei an. Mit den Parametern `lineStart` und `lineEnd` wird der Bereich angegeben, der eingelesen werden soll. Dazu wird ein String-Array angelegt, in das jede eingelesene Zeile als Arrayeintrag abgelegt wird. Das Array wird am Ende der Methode zurückgegeben.

- Implementieren Sie eine Methode `extractData`:

  ```
  void extractData(String[] dataArray, int[] resultArray,
                   int numColumn, int entriesPerYear)
  ```

  Das String-Array `dataArray` beinhaltet alle eingelesenen Daten. In das ganzzahlige Array `resultArray` werden die extrahierten Daten geschrieben. Mit dem Parameter `numColumn` können Sie angeben, von welcher Spalte Sie die Daten extrahieren möchten. Der Parameter `entriesPerYear` gibt an, in welcher Auflösung die Daten in `dataArray` gespeichert sind. In unserem Fall wird dieser Parameter 12 sein, da es immer 12 Einträge pro Jahr gibt. Diese

[1] `https://introcs.cs.princeton.edu/java/stdlib/javadoc/In.html`

Methode wird jeweils einmal für Frost-, Eis-, Sommer- und Hitzetage aufgerufen. Da Sie die Anzahl dieser Tage per Jahr darstellen sollen, müssen Sie die Einträge für die Anzahl solcher Tage für ein Jahr summieren, bevor Sie diese in das `resultArray` schreiben. Das Array `resultArray` hat die Länge 40, da wir die vergangenen 40 Jahre darstellen wollen. Der Parameter `numColumn` bestimmt, welche besonderen Tage gezählt werden. Abhängig von diesem Parameter `numColumn` wird nun die Anzahl aller z.B. Frosttage eines Jahres im Array `resultArray` gespeichert. Beim Beispiel Frosttage bedeutet dies nun konkret: die Anzahl aller Frosttage im Jahr 1979 wird in `resultArray[0]` gespeichert, die Anzahl aller Frosttage im Jahr 2018 in `resultArray[39]`.

(!) Für diese Methode können die Methoden `split` der Klasse `String` und `parseInt` der Klasse `Integer` hilfreich sein.

- Implementieren Sie eine Methode `drawChart`:

```
void drawChart(int[] frostDays, int[] iceDays,
      int[] summerDays, int[] heatDays)
```

In den vier ganzzahligen Arrays `frostDays` (Frosttage), `iceDays` (Eistage), `summerDays` (Sommertage) und `heatDays` (Hitzetage) sind alle Daten für die Darstellung in Abbildung 1 enthalten. Auf der x-Achse finden Sie die Jahre (19)79 bis (20)18. Auf der y-Achse werden für jedes Jahr die Anzahl der Frost-, Eis-, Sommer-, und Hitzetage dargestellt. Dazu wird das Diagramm in einen oberen und einen unteren Teil aufgeteilt. Oberhalb der grünen Mittellinie beschreibt die Höhe eines nicht gefüllten roten Balkens die Anzahl der Sommertage des entsprechenden Jahres und die Höhe eines gefüllten roten Balkens die Anzahl der Hitzetage. Unterhalb der grünen Mittellinie repräsentieren die nicht gefüllten blauen Balken die Frosttage und die gefüllten blauen Balken innerhalb die Anzahl der Eistage des entsprechenden Jahres. Zusätzlich wird die y-Achse links und rechts mit 25, 50, 75 und 100 beschriftet um die Anzahl der Tage besser abschätzen zu können.

Für die gezeigte Grafik wurden folgende Zeichenparameter verwendet:

  - Ein Tag entspricht 2 Pixel auf der y-Achse
  - Abstand des ersten und letzten Balkens zum Rand entspricht 20 Pixel
  - Abstand zwischen den äußeren Balken entspricht 10 Pixel
  - Die äußere Balkenbreite entspricht 20 Pixel
  - Die innere Balkenbreite entspricht 10 Pixel
  - Die Liniendicke der Nulllinie entspricht 0.0055
  - Die Liniendicke der roten und blauen Balken entspricht 0.005
  - Die Schriftformatierung für die Anzahl der Tage (y-Achse) und den entsprechenden Jahren (x-Achse) entspricht `Font("Times", Font.PLAIN, 10)`
  - Der Text für die Beschriftung rechts und links wurde um jeweils 8 Pixel eingerückt.

(!) Wir haben bei dieser Aufgabe auf Vorbedingungen verzichtet. Achten Sie darauf, dass die Methoden von Ihrer Implementierung in der main-Methode mit sinnvollen Werten (z.B. keine null-Referenzen, Dateiname ist korrekt und Datei existiert, Arrays haben korrekte Längen etc.) aufgerufen werden.

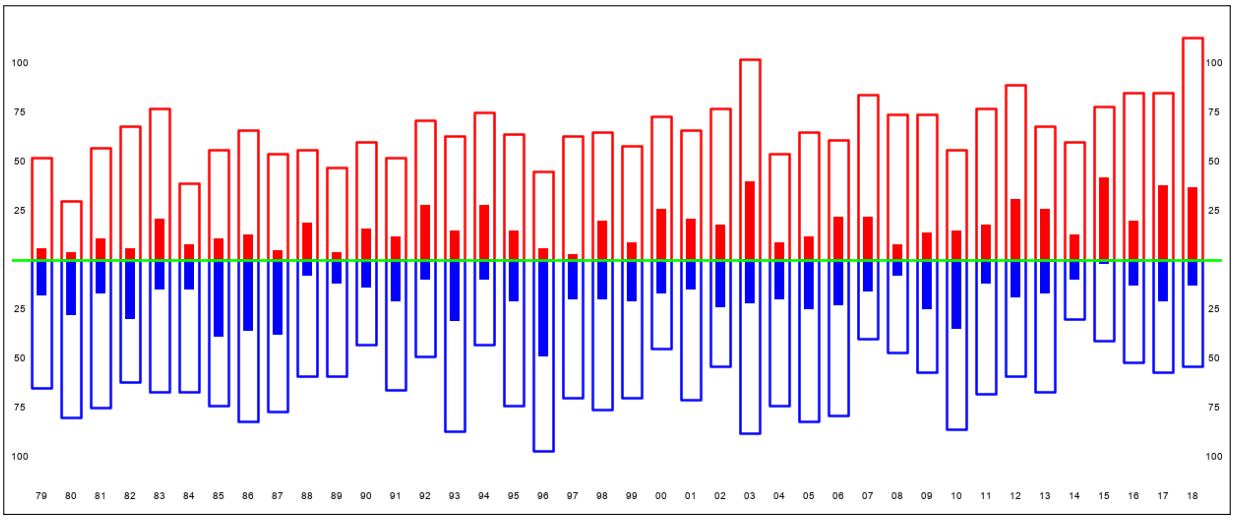

Abbildung 1: Darstellung der Frost-, Eis-, Sommer-, und Hitzetage der letzten 40 Jahre.

Aufgabe 3

Implementieren Sie folgende Aufgabenstellung:

(!) Gilt für alle zu implementierenden Methoden: Sie dürfen keine globalen Variablen oder zusätzliche eigene Hilfsmethoden verwenden. Die vorgegebenen Methodenköpfe dürfen nicht erweitert oder geändert werden. Für die Implementierung der Aufgabenstellung dürfen keine Schleifen verwendet werden.

- Implementieren Sie eine rekursive Methode `getMaxPairSum`:

  ```
  int getMaxPairSum(int[] workArray, int start, int end)
  ```

  Diese Methode soll die größte Summe zweier benachbarter Arrayeinträge innerhalb des Arrays `workArray` im Indexintervall von `[start, end]` suchen und retournieren.
  Vorbedingungen: `workArray != null`, `start` $<$ `end` und `start` und `end` sind gültige Indizes von `workArray`.

  Beispiele:
  `int[] array = {3, 1, 7, 9, 4, 10, 5, 3};`
  `getMaxPairSum(array, 0, 7))` liefert `16`
  `getMaxPairSum(array, 4, 7))` liefert `15`
  `getMaxPairSum(array, 1, 3))` liefert `16`
  `getMaxPairSum(array, 0, 2))` liefert `8`
  `getMaxPairSum(array, 0, 1))` liefert `4`

- Implementieren Sie eine rekursive Methode `countValuesWithSmallerNeighbors`:

  ```
  int countValuesWithSmallerNeighbors(int[] workArray, int index)
  ```

  Diese Methode zählt beginnend beim Index `index` bis zum Index `workArray.length-2` des Arrays `workArray` die Anzahl solcher Arrayeinträge, die einen Vorgänger und Nachfolger im Array besitzen, die beide kleiner sind als der Eintrag selbst.
  Vorbedingungen: `workArray != null`, `workArray.length > 2` und `index` > 0.

  Beispiele:
  `int[] array = {3, 9, 7, 19, 8, 10, 6, 3, 11, 5};`
  `countValuesWithSmallerNeighbors(array, 1))` liefert `4`
  `countValuesWithSmallerNeighbors(array, 4))` liefert `2`
  `countValuesWithSmallerNeighbors(array, 6))` liefert `1`
  `countValuesWithSmallerNeighbors(array, 8))` liefert `1`

- Implementieren Sie eine rekursive Methode `getArrayWithNegativeValues`:

  ```
  int[] getArrayWithNegativeValues(int[] workArray, int index)
  ```

  Diese Methode erstellt eine Kopie von `workArray`, filtert alle positiven Werte im Indexintervall von `[index, workArray.length[` heraus und ersetzt diese durch 0. Für das Erzeugen der Kopie dürfen Sie die Methode `clone` verwenden. Die Werte werden nur in der Kopie verändert, sodass das Array `workArray` unverändert bleibt. Die gefilterte Kopie wird am Ende zurückgegeben.
  Vorbedingungen: `workArray != null`, `workArray.length > 0` und `index` ist ein gültiger Index von `workArray`.

  Beispiele:
  `int[] array = {5, -3, 7, -15, 20, -5, 0, 14, 3, -2, -8};`
  `getArrayWithNegativeValues(array, 0)` → `[0, -3, 0, -15, 0, -5, 0, 0, 0, -2, -8]`
  `getArrayWithNegativeValues(array, 10)` → `[5, -3, 7, -15, 20, -5, 0, 14, 3, -2, -8]`
  `getArrayWithNegativeValues(array, 8)` → `[5, -3, 7, -15, 20, -5, 0, 14, 0, -2, -8]`
  `getArrayWithNegativeValues(array, 4)` → `[5, -3, 7, -15, 0, -5, 0, 0, 0, -2, -8]`

- Implementieren Sie eine rekursive Methode `containsValue`:

  ```
  boolean containsValue(int[] workArray, int value)
  ```

  Diese Methode prüft, ob eines der Elemente innerhalb des Arrays `workArray` dem Wert von `value` entspricht. Die Methode soll in ihrem Methodenrumpf zwei rekursive Aufrufe enthalten, wobei ein rekursiver Aufruf die erste Hälfte des Arrays durchsucht und der zweite Aufruf die zweite Hälfte. Sie können die Methode `Arrays.copyOfRange(...)` für die Implementierung der Methode benutzen.
  Vorbedingungen: `workArray != null` und `workArray.length > 0`.

  Beispiele:
  `int[] array = {2, 4, 7, 10, -10, 4, 0, 0, 27, 11, 4, 6};`
  `containsValue(array,11)` liefert `true`
  `containsValue(array,2)` liefert `true`
  `containsValue(array,25)` liefert `false`
  `containsValue(array,0)` liefert `true`
  `containsValue(array,14)` liefert `false`
  `containsValue(array,6)` liefert `true`

Zusatzfrage(n):

1. Wie könnte `containsValue(...)` optimiert werden, wenn die Vorbedingung ist, dass `workArray` aufsteigend sortiert ist?

Aufgabe 4

Implementieren Sie folgende Aufgabenstellung:

(!) Gilt für alle zu implementierenden Methoden: Sie dürfen keine globalen Variablen oder zusätzliche eigene Hilfsmethoden verwenden. Die vorgegebenen Methodenköpfe dürfen nicht erweitert oder geändert werden. Für die Implementierung der Aufgabenstellung dürfen keine Schleifen verwendet werden.

- Implementieren Sie eine rekursive Methode `insertCharValue`:

  ```
  String insertCharValue(String text)
  ```

  Diese Methode fügt vor jedes Zeichen des Strings `text` dessen ASCII-Wert als Dezimalzahl ein. Zusätzlich werden Dezimalwert und Zeichen in runden Klammern eingeschlossen. Das Ergebnis wird als neuer String zurückgeliefert.
  Vorbedingungen: `text != null` und `text.length > 0`.

  Beispiele:
  `insertIndex("hallo")` liefert `"(104h)(97a)(108l)(108l)(111o)"`
  `insertIndex("a!")` liefert `"(97a)(33!)"`
  `insertIndex("Ein Test")` liefert `"(69E)(105i)(110n)(32 )(84T)(101e)(115s)(116t)"`

- Implementieren Sie eine rekursive Methode `removeCharsInString`:

  ```
  String removeCharsInString(String text)
  ```

  Diese Methode entfernt das erste Zeichen, sowie jedes weitere Vorkommen dieses Zeichens im String `text` und gibt anschließend diesen neu entstandenen String zurück. Für die Implementierung darf die Methode `replaceAll` **nicht** verwendet werden.
  Vorbedingungen: `text != null` und `text.length > 0`.

  Beispiele:
  `removeCharsInString("testtrompete")` liefert `"esrompee"`
  `removeCharsInString("test")` liefert `"es"`
  `removeCharsInString("t")` liefert `""`
  `removeCharsInString("angabe")` liefert `"ngbe"`

- Implementieren Sie eine rekursive Methode `shiftMaxCharRight`:

  ```
  String shiftMaxCharRight(String text)
  ```

  Diese Methode verschiebt das Zeichen mit dem größten Wert (ASCII-Wert) innerhalb des Strings `text` an die letzte Stelle. Alle Zeichen von der ursprünglichen Position des Maximums bis zum letzten Index werden dabei um eine Position in Richtung kleinerem Index verschoben. Das Ergebnis wird als neuer String zurückgeliefert.
  Vorbedingungen: `text != null` und das Zeichen mit dem größten Wert kommt nur einmal vor.

  Beispiele:
  `shiftMaxCharRight("acmbwxdzfjdk")` liefert `"acmbwxdfjdkz"`
  `shiftMaxCharRight("")` liefert `""`
  `shiftMaxCharRight("za")` liefert `"az"`
  `shiftMaxCharRight("a")` liefert `"a"`
  `shiftMaxCharRight("habcdefg")` liefert `"abcdefgh"`

Zusatzfrage(n):

1. Was bedeutet der Begriff *Endrekursion*?
2. Was ist der Unterschied zwischen *linearer* und *verzweigter* Rekursion?