Ludwig-Maximilians-Universität München
Institut für Informatik
Prof. Dr. Peer Kröger
Janina Bleicher, Florian Richter

München, 12.12.2016

Einführung in die Programmierung

WS 2016/17

Übungsblatt 8: Arrays, Strings

Besprechung: 19.12./21.12./23.12.2016

Ende der Abgabefrist: Freitag, 16.12.2016 14:00 Uhr.

Geben Sie Ihre Lösung in **Zweierteams** ab.
Der Javacode muss kompilieren. Löschen Sie jegliche packages aus Ihrer Abgabe heraus, sofern nicht anders verlangt. Außerdem soll jeglicher Code mit JavaDoc-Kommentaren dokumentiert sein.

Aufgabe 8-1 *Arrays* 1+1+1+1+1 Punkte

Diese Aufgabe beschäftigt sich mit Arrays und einigen ihrer Grundoperationen. Daher dürfen Sie nur Basisoperationen verwenden, d.h. Sie müssen auch auf die Objekt-Methoden von Arrays (z.B. Arrays.sort()) verzichten. `length` ist allerdings ein Array-Attribut und darf verwendet werden.

(a) Implementieren Sie eine Methode `int arrayGet(int[] array, int i)`, die für ein int-Array den Wert an der i-ten Position im Array zurück gibt. Achten Sie darauf, dass auch Positionen sinnvoll behandelt werden, die nicht im Array vorkommen.

(b) Implementieren Sie eine Methode `int sum(int[] array)`, die alle Einträge des Arrays aufaddiert und diese Summe zurückgibt.

(c) Implementieren Sie eine Methode `int mean(int[] array)`, die den Mittelwert aller Einträge eines Arrays zurückgibt.

(d) Implementieren Sie eine Methode `void square(int[] array)`, die jeden Eintrag eines Arrays quadriert.

(e) Implementieren Sie eine Methode `int max(int[] array)`, die den größten Eintrag eines Arrays zurückgibt.

Aufgabe 8-2 *Arrays 2* 1+2+1+2+1 Punkte

Hier gibt es noch mehr Aufgaben zu Arrays. Auch hier dürfen Sie nur Basisoperationen verwenden, d.h. Sie müssen auch auf die Objekt-Methoden von Arrays verzichten.

(a) Implementieren Sie eine Methode `void swap(int[] array, int i, int j)`. Diese soll die Einträge i und j des Arrays array miteinander vertauschen.

(b) Implementieren Sie eine Methode `void sort(int[] array)`, die die Einträge eines Arrays der Größe nach aufsteigend sortiert. Legen Sie dazu ein neues leeres Array der gleichen Größe an. Durchlaufen Sie das Array von links nach rechts und tauschen Sie das Element an der i-ten Position mit dem i-t ~~Größten~~ kleinsten Element.

(c) Implementieren Sie eine Methode `int median(int[] array)`, die den Median eines Arrays bestimmt. Bei Arrays mit gerader Anzahl an Elementen entspricht hier der Median dem größeren der beiden Kandidaten. Beachten Sie, dass der Median nicht das Gleiche ist wie der Mittelwert.

(d) Implementieren Sie eine Methode ~~`void`~~ `int[] resize(int[] array, int length)`. Diese Methode soll die Länge eines Arrays wie folgt ändern:

- ist die neue Länge kürzer als die Bisherige, wird das Array an den letzten Positionen beschnitten.
- ist die neue Länge länger als die Bisherige, werden die zusätzlichen Positionen am Ende des Arrays mit dem Wert 0 angefügt.

Korrektur: `resize` muss natürlich den Rückgabewert `int[]` haben. Überlegen Sie für sich selbst, warum!

(e) Implementieren Sie eine Methode `String show(int[] array)`, die die Einträge des Array elementweise ausgibt.
Für `int[] bsp = {6,4,9};` liefert `show(bsp)` den String ~~`"[6 4 9 ]"`~~ `"[ 6 4 9 ]"` .

Aufgabe 8-3 *Caesar-Chiffre* **3 Punkte**

Zum Verschlüsseln von Daten gibt es viele Methoden. Eine sehr alte Variante ist die Rotation des Alphabets um eine bestimmte Anzahl von Stellen. Die bekannteste davon ist vermutlich die Caesar-Chiffre, bei der eine Rotation von 13 Positionen ausgeführt wird. Dies ermöglicht bei 26 Buchstaben, die gleiche Methode zum Ver- und Entschlüsseln zu verwenden. Hier werden wir beliebige Rotationen berücksichtigen. Im Folgenden sehen Sie eine Rotation um 5 Stellen.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e

Implementieren Sie ein Programm, das das Argument `args` des Benutzers verwendet und alle 26 möglichen Rotationen ausführt und ausgibt. Wenn der Nutzer noch eine Zahl vorgibt, wird nur diese Rotation angezeigt. Das Grundgerüst können Sie herunterladen. Sie benötigen für diese Aufgabe sehr wahrscheinlich die `String`-Methoden `charAt()` und `length()`. Es genügt, wenn Ihr Tool nur mit Kleinbuchstaben arbeiten kann. Belassen Sie alle anderen Zeichen bei ihrer Klardarstellung. Entziffern Sie mit Ihrem Tool folgende Textzeile:
`robjvsmrox qveomugexcmr. ne rkcd nso kepqklo qovyocd.`

Aufgabe 8-4 *Bildbearbeitung* **2+2+2+2+0+4 Punkte**

In dieser Aufgabe sollen Sie ein Programm zur Bildbearbeitung vervollständigen. Dieses soll ein paar grundlegende Bildmanipulationen beherrschen, sodass mittels Konsole eine Bilddatei eingelesen, manipuliert und anschließend die veränderte Version wieder abgespeichert wird. Ein Grundgerüst wird Ihnen bereits zur Verfügung gestellt. Für die volle Funktionalität müssen Sie lediglich die mit TODO gekennzeichneten Methodenrümpfe ergänzen.

(a) Ein Bild ist im Wesentlichen ein zweidimensionales Array, also ein Array aus Arrays. Jedes Matrixelement heißt Pixel und wird durch eine ganze Zahl repräsentiert. Der hier verwendete Datentyp dafür ist `int`, der 4 Byte enthält. Um einen Pixel als Farbpunkt zu identifizieren, brauchen wir Informationen über seinen Rot-, Grün- und Blauwert. Zusätzlich gibt es eine Transparenz α. Diese werden derart als Pixelwert gespeichert, dass jeweils 8 Bits für α und die drei Farbwerte in einem Wert codiert sind:

In dem gegebenen Grundgerüst finden Sie bereits die Methode `setColors(...)`. Vervollständigen Sie gemäß der Signatur die Methode `getColors(...)`.

(b) Vervollständigen Sie die Methode `mirror()`, die das Bild an der vertikalen Mittelachse spiegeln soll.

(c) Vervollständigen Sie die Methode `rotate(n)`, die das Bild n-mal um 90 Grad drehen soll. Es obliegt Ihnen, die Drehrichtung zu wählen. Spiegeln Sie aber dabei nicht das Bild! Beachten Sie, dass sich die Dimension des Bilds damit ändern kann.

(d) Vervollständigen Sie die Methode `invert()`, die jeden Farbwert invertiert. Die Sättigung sollen Sie unverändert lassen. Da eine Farbe einen 8-bit großen Wert zur Darstellung nutzt, bewegen diese sich in einem Interval von 0 bis 255 einschließlich. Der inverse Farbwert zu x ist damit $255 - x$. Denken Sie insbesondere daran, dass sie die Farbwerte getrennt behandeln müssen und anschließend wieder zusammenfügen und im Bildarray speichern.

(e) Die Methode `meanFilter(filter, factor)` nutzt eine sogenannte Faltungsmatrix, um Pixel abhängig von ihrer Nachbarschaft zu verändern. In Ihrem Grundgerüst gibt es bereits einige vordefinierte Filter. Die Anwendung dergleichen müssen Sie aber noch implementieren. Vervollständigen Sie die Methode, sodass zu jedem Pixel seine 3×3-Nachbarschaft mit dem durch `filter` gegebenen Filter gewichtet aufsummiert und anschließend durch den `factor` skaliert wird. Auch hier müssen die 3 Farbwerte gesondert betrachtet und anschließend zusammengefügt werden.
Beispiel: Angenommen, Sie nutzen einen Gauß-Filter mit `filter` $= [1, 2, 1, 2, 4, 2, 1, 2, 1]$ und einem `factor` $= 1/16$. Stellen Sie sich die Faltungsmatrix als 3×3-Matrix vor:

$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{16}$
$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{8}$
$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{16}$

Für einen Pixel sei sein Rotwert an einer Stelle gegeben. Die für diesen Pixel relevante Nachbarschaft ist ebenfalls 3×3 Pixels groß:

32	24	24
32	16	24
32	16	8

`gauss-filter` $\Longrightarrow$

32	24	24
32	$\frac{22}{16}$	24
32	16	8

Implementieren Sie NUR 3×3 Filter. Bei den Randpunkten (die ja keine vollständige Nachbarschaft haben) nehmen Sie an, dass Bildpunkte jenseits des Rands den Wert 0 haben. Sie können dazu die (noch zu implementierende) Zugriffsmethode `getPixel(int x, int y)` verwenden.

(f) Zuletzt vervollständigen Sie die Methode `medianFilter()`, die ähnlich wie zuvor die Nachbarschaft betrachtet. Allerdings soll hier keine Summe gebildet werden, sondern der Median aus dem Pixel selbst sowie seinen 4 direkten Nachbarn. Verfahren Sie mit Randpunkten wie zuvor. Sie können Ergebnisse aus der vorherigen Übung verwenden, oder zum Sortieren die Methode `Arrays.sort(array)` verwenden. Der Median liegt dann in der Mitte des Arrays.

Das fertige Programm sollte in der Lage sein, das Testbild `testbild.png` ein wenig zu verbessern, indem etwas Rauschen entfernt und das Bild wieder in seine native Position gebracht wird, zum Beispiel mittels

```
java Bildbearbeitung 'testbild.png' -i -m -rot180 -median
```