Dr. Annette Bieniusa
Mathias Weber, M. Sc.
Peter Zeller, M. Sc.

TU Kaiserslautern
Fachbereich Informatik
AG Softwaretechnik

Lösungshinweise/-vorschläge zum Übungsblatt 4: Software-Entwicklung 1 (WS 2017/18)

Die Hinweise und Vorschläge in diesem Dokument sollen der Lösungsfindung dienen und erheben demnach weder Anspruch auf Vollständigkeit noch Korrektheit. Sollten Sie Fehler finden, würden wir uns freuen, wenn Sie uns diese mitteilen. (Kontaktdaten finden Sie auf unserer Webpräsenz.)

Aufgabe 1 Arrays und Prozeduren (11 Punkte)

Laden Sie sich für diese Aufgabe die Vorlage `ArraysAndProcedures.java` herunter. Diese enthält bereits Vorlagen für die zu implementierenden Prozeduren.

Zum Testen können Sie die `main`-Prozedur anpassen und mit der Bibliotheksfunktion `Arrays.toString(ar)` die Ergebnisse ausgeben. Verwenden Sie für diese Aufgabe sonst keine weiteren Bibliotheksfunktionen.

a) Schreiben Sie eine Prozedur `replaceAll`, welche zwei **`int`** Werte `x` und `y` sowie ein Array von **`int`**-Werten nimmt. Die Prozedur soll dann alle Vorkommen von `x` im gegebenen Array durch `y` ersetzen.

Dies ist eine typischer Fall, bei dem eine Prozedur nur für ihre Seiteneffekte aufgerufen wird: Das übergebene Array wird verändert und es wird kein Wert zurückgegeben. Die Prozedur `replaceAll` ist auch typisches Beispiel für eine Prozedur, bei der eine Operation auf alle Elemente eines Arrays angewendet wird. Solche Prozeduren können immer nach dem gleichen Schema programmiert werden, nur der Schleifenrumpf ist jeweils anders.

```
public static void replaceAll(int x, int y, int[] ar) {
    for (int i=0; i<ar.length; i++) {
        if (ar[i] == x) {
            ar[i] = y;
        }
    }
}
```

b) Schreiben Sie zwei weitere Prozeduren wie `replaceAll`. Eine Variante `replaceFirst`, die nur das erste Vorkommen im Array ersetzt und eine andere `replaceLast`, die nur das letzte Vorkommen im Array ersetzt.

Die Lösungen hier zeigt das Verwenden von **`break`** um die Schleife frühzeitig abzubrechen und das Iterieren durch ein Array in umgekehrter Richtung.

```
public static void replaceFirst(int x, int y, int[] ar) {
    for (int i=0; i<ar.length; i++) {
        if (ar[i] == x) {
            ar[i] = y;
            break;
        }
    }
}

public static void replaceLast(int x, int y, int[] ar) {
    for (int i=ar.length-1; i>=0; i--) {
```

```
            if (ar[i] == x) {
                ar[i] = y;
                break;
            }
        }
    }
```

c) Schreiben Sie eine Prozedur `substAll`, welche wie `replaceAll` funktioniert, jedoch das Eingabe-Array nicht verändert, sondern ein neues Array mit dem Ergebnis zurückliefert.

Wichtig ist hier den Unterschied zu `replaceAll` zu verstehen. Beim Lösen einer Aufgabe ist es wichtig, genau darauf zu achten, welche Variante verlangt ist. Beim Schreiben einer eigenen Prozedur muss man sich überlegen, welche Variante für den konkreten Fall besser geeignet ist. Eine Variante ohne Seiteneffekte wie `substAll` ist oft leichter zu verstehen, ist aber manchmal weniger effizient.

```
public static int[] substAll(int x, int y, int[] ar) {
    int[] res = new int[ar.length];
    for (int i=0; i<ar.length; i++) {
        if (ar[i] == x) {
            res[i] = y;
        } else {
            res[i] = ar[i];
        }
    }
    return res;
}
```

d) Schreiben Sie eine Prozedur `onlyEven`, welche ein Array von **`int`**-Werten nimmt und ein neues Array zurückgibt, in dem sich nur die geraden Zahlen aus dem ursprünglichen Array befinden.

Die Prozedur `onlyEven` ist ein typisches Beispiel für eine Prozedur, die alle Elemente aus einem Array auswählt, die ein bestimmtes Kriterium erfüllen. Oft wird diese Art der Operation auch `filter` genannt. In der folgenden Lösung ist das Kriterium `isEven` in eine extra Prozedur ausgelagert um die Lesbarkeit zu verbessern. Für ein anderes Auswahlkriterium kann hier einfach eine andere Prozedur eingesetzt werden.

Die Prozedur `onlyEven` zeigt auch das Problem, dass sich Arrays nach dem Erstellen nicht in der Länge verändern lassen. Wir berechnen deshalb zuerst die benötigte Länge und erstellen dann das neue Array für die Rückgabe.

```
public static boolean isEven(int x) {
    return x % 2 == 0;
}

public static int[] onlyEven(int[] ar) {
    // Anzahl der Geraden Zahlen berechnen:
    int evenCount = 0;
    for (int i=0; i<ar.length; i++) {
        if (isEven(ar[i])) {
            evenCount = evenCount + 1;
        }
    }
    // Array der richtigen Größe anlegen:
    int[] res = new int[evenCount];
    int insertPosition = 0;
    for (int i=0; i<ar.length; i++) {
        if (isEven(ar[i])) {
            res[insertPosition] = ar[i];
            insertPosition++;
        }
    }
    return res;
```

```
}
```

e) Schreiben Sie eine Prozedur `allHaveZero`, welche ein Array von **`int`**-Arrays nimmt und prüft, ob alle Arrays die Zahl 0 enthalten.

Diese Aufgabe ist ein Beispiel für das Verwenden von mehrdimensionalen Arrays. Die Lösung zeigt auch zwei Muster mit denen man Fragen wie die folgenden beantworten kann: "existiert ein Element mit einer bestimmten Eigenschaft im Array?" und "gilt eine bestimmte Eigenschaft für alle Elemente im Array?". Für die Existenz-Frage geben wir **`true`** zurück sobald wir ein entsprechendes Element gefunden haben. Wenn wir nach Durchlaufen des Arrays kein passendes Element gefunden haben geben wir **`false`** zurück. Für die "Für alle"-Frage geben wir **`false`** zurück, sobald ein Element die Eigenschaft nicht erfüllt. Wenn wir am Ende kein solches Element gefunden haben, geben wir **`true`** zurück.

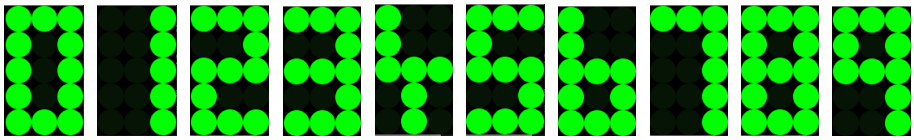
```
public static boolean containsZero(int[] ar) {
    for (int j=0; j<ar.length; j++) {
        if (ar[j] == 0) {
            return true;
        }
    }
    return false;
}

public static boolean allHaveZero(int[][] arrays) {
    for (int i=0; i<arrays.length; i++) {
        if (!containsZero(arrays[i])) {
            return false;
        }
    }
    return true;
}
```

Aufgabe 2 2D-Arrays (11 Punkte)

In dieser Aufgabe soll ein Programm entwickelt werden, welches Zahlen mit bis zu drei Stellen in ein geeignetes Format für eine LED-Matrix-Anzeige umwandelt.

Bei einer LED-Matrix-Anzeige werden Ziffern durch eine Anzahl an Leuchtdioden angezeigt. Diese sind in einem Rechteck angeordnet. Durch das aktivieren und deaktivieren bestimmter LEDs kann das angezeigte Muster verändert werden. Wir gehen in dieser Aufgabe von einer Matrix von drei LEDs in der Breite und fünf LEDs in der Höhe für jede Ziffer aus.

Die Ziffern sollen auf der Anzeige wie folgt angezeigt werden:



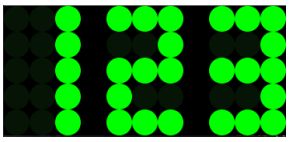
Die Matrixrepräsentation einer Zahl besteht aus einer Matrix mit 5 Zeilen und jeweils 9 Spalten aus 0 und 1. Ist eine Stelle gleich 1, so leuchtet die LED an dieser Stelle, ist die Stelle gleich 0 so ist die LED aus. Zur Anzeige stehen drei Ziffern zur Verfügung, sodass Zahlen zwischen 0 und 999 angezeigt werden können. Bei einstelligen und zweistelligen Zahlen sollen die ersten Ziffern dunkel bleiben.

Aufgabe ist es, ein Programm `ToMatrix` zu schreiben, welches Zahlen von der Standardeingabe ließt bis diese leer ist und für jede dieser Zahlen die Matrixrepräsentation auf die Standardausgabe ausgibt. Die Matrix der nächsten Zahl folgt direkt darunter (ohne Trennung).

Beispiel:

Die Matrixrepräsentation der Zahl `123` ist auf der linken Seite dargestellt. Diese wird wie rechts zu sehen auf der LED-Matrix angezeigt:

```
001111111
001001001
001111111
001100001
001111111
```



Hinweise:

- Zum Aufteilen der eingelesenen Zahl in Ziffern sind die Modulo Operation (`%`) und die Integer-Division (`/`) hilfreich.
- Überlegen Sie sich eine geeignete Einteilung in Prozeduren. Die Aufgabe, eine Ziffer an eine bestimmte Stelle in der Matrix zu schreiben kommt mehrfach mit verschiedenen Stellen vor.
- Zum Einlesen der Zahlen von der Standardeingabe können Sie die Bibliothek `StdIn` verwenden.
- Sie können das Programm `ShowMatrix` von der Vorlesungsseite verwenden, um die Anzeige zu kontrollieren. Dazu leiten Sie die Ausgabe Ihrer Implementierung von `ToMatrix` an das Programm `ShowMatrix` weiter durch `java ToMatrix | java ShowMatrix`.

```
public class ToMatrix {
    static int[][][] digits = new int[][][]{
            {
                    {1, 1, 1},
                    {1, 0, 1},
                    {1, 0, 1},
                    {1, 0, 1},
                    {1, 1, 1}
            },
            {
                    {0, 0, 1},
                    {0, 0, 1},
                    {0, 0, 1},
                    {0, 0, 1},
                    {0, 0, 1}
            },
            {
                    {1, 1, 1},
                    {0, 0, 1},
                    {1, 1, 1},
                    {1, 0, 0},
                    {1, 1, 1}
            },
            {
                    {1, 1, 1},
                    {0, 0, 1},
                    {1, 1, 1},
                    {0, 0, 1},
                    {1, 1, 1}
            },
            {
                    {1, 0, 0},
                    {1, 0, 0},
                    {1, 1, 1},
                    {0, 1, 0},
                    {0, 1, 0}
            },
            {
                    {1, 1, 1},
                    {1, 0, 0},
                    {1, 1, 1},
```

```
                {0, 0, 1},
                {1, 1, 1}
        },
        {
                {1, 0, 0},
                {1, 0, 0},
                {1, 1, 1},
                {1, 0, 1},
                {1, 1, 1}
        },
        {
                {1, 1, 1},
                {0, 0, 1},
                {0, 0, 1},
                {0, 0, 1},
                {0, 0, 1}
        },
        {
                {1, 1, 1},
                {1, 0, 1},
                {1, 1, 1},
                {1, 0, 1},
                {1, 1, 1}
        },
        {
                {1, 1, 1},
                {1, 0, 1},
                {1, 1, 1},
                {0, 0, 1},
                {0, 0, 1}
        }
};

static void put(int[][] character, int[][] display, int offsetx, int offsety) {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 5; j++) {
            display[i + offsetx][j + offsety] = character[j][i];
        }
    }
}

public static void main(String[] args) {
    // Display mit drei Spalten
    int spalten = 3;

    while (!StdIn.isEmpty()) {
        int[][] display = new int[spalten * 3][5];

        int input = StdIn.readInt();
        if (input < 0) {
            System.err.println("Negative Zahlen nicht unterstuetzt!");
            System.exit(1);
        }
        if (input > 999) {
            System.err.println("Input zu gross!");
            System.exit(1);
        }
        for (int position = spalten - 1; position >= 0; position--) {
            int digit = input % 10;
            put(digits[digit], display, position * 3, 0);
            input = input / 10;
            // wenn keine Digits mehr folgen, brich ab
            if (input == 0) {
                break;
            }
```

```
            }

            // Ausgabe
            for (int j = 0; j < 5; j++) {
                for (int i = 0; i < spalten * 3; i++) {
                    StdOut.print(display[i][j]);
                }
                StdOut.println();
            }
        }
    }
}
```

Aufgabe 3 StdIn und StdOut (4 Punkte)

Schreiben Sie eine Prozedur **static** `String[] readStrings()` in einer Datei `ReadStrings.java`, welche mit `StdIn.readString()` `String`-Werte von der Standardeingabe liest, bis die Standardeingabe leer ist und die Werte in einem Array speichert.

Hinweis: Arrays können in Java nach ihrer Erstellung nicht vergrößert oder verkleinert werden. Legen Sie deshalb zu Beginn der Prozedur ein Array der Größe 5 an und erstellen Sie ein neues Array der doppelten Größe, sobald das aktuelle Array zu klein wird. Kopieren Sie dann am Ende der Prozedur die Werte in ein Array der passenden Größe.

```
    static String[] readStrings() {
        String[] result = new String[5];
        int n = 0;
        while (!StdIn.isEmpty()) {
            if (n >= result.length) {
                result = resized(result, result.length*2);
            }
            result[n] = StdIn.readString();
            n++;
        }
        if (n < result.length) {
            result = resized(result, n);
        }
        return result;
    }

    /** erstellt eine Kopie von einem Array mit anderer Groesse */
    static String[] resized(String[] values, int newSize) {
        String[] res = new String[newSize];
        copyValues(values, res);
        return res;
    }

    private static void copyValues(String[] source, String[] destination) {
        for (int i = 0; i < source.length && i < destination.length; i++) {
            destination[i] = source[i];
        }
    }
```

Aufgabe 4 Formatierte Ausgabe (3 Punkte)

Schreiben Sie ein Programm `SqrtTable`, welches **double**-Werte von der Standardeingabe liest und nach jeder gelesenen Zahl die Zahl selbst und die Wurzel der Zahl, jeweils durch Komma getrennt in einer Zeile aus-

gibt. Die Zahlen sollen dabei mit zwei Nachkommastellen ausgegeben werden. Verwenden Sie hierzu die Prozeduren `StdIn.isEmpty`, `StdIn.readDouble`, `StdOut.printf`, und `Math.sqrt`.

```
public class SqrtTable {
    public static void main(String[] args) {
        while (!StdIn.isEmpty()) {
            double d = StdIn.readDouble();
            StdOut.printf("%.2f,%.2f%n", d, Math.sqrt(d));
        }
    }
}
```

Aufgabe 5 StdDraw (11 Punkte)

a) Ändern Sie das Beispiel-Programm so ab, dass ein Balken-Diagramm gezeichnet wird, das etwa wie das oben gezeigte aussieht.

Die Lösung zu Teilaufgabe a) müssen Sie nicht abgeben. Geben Sie am Ende nur eine Datei mit dem kompletten Programm ab, wie es in Aufgabe d) gefordert ist.

Siehe Lösung von b)

b) Schreiben Sie eine Prozedur **void** `drawBar(`**double** `x, `**double** `width, String text, `**double** `height)`, welche einen einzelnen Balken des Balken-Diagramms zeichnet. Dabei ist der Parameter `text` der Text, der unter dem Balken angezeigt wird, `x` ist die x-Koordinate für die Mitte des Balkens und `width` und `height` geben die Breite und Höhe des Balken an. Schreiben Sie Ihr Programm so um, dass diese Prozedur zum Zeichnen des Diagramms verwendet wird.

```
public class BalkenDiagramm1 {

    private static void drawBar(double x, double width, String text, double
    height) {
        StdDraw.filledRectangle(x, 0.1 + height/2, width/2, height/2);
        StdDraw.text(x, 0.05, text);
    }

    public static void main(String[] args) {
        double width = 0.1;
        drawBar(0.2, width, "A", 0.5);
        drawBar(0.4, width, "B", 0.8);
        drawBar(0.6, width, "C", 0.3);
        drawBar(0.8, width, "D", 0.4);
        StdDraw.line(0, 0.1, 1, 0.1);
    }
}
```

c) Schreiben Sie eine Prozedur `void drawBars(String[] labels, double[] heights, double width, double gap)`, welche ein Array von String-Werten und ein Array von `double`-Werten nimmt, die jeweils die-Beschriftung und Höhe eines Balken angeben. Außerdem nimmt die Prozedur die Breite der Balken `width` und den Abstand zwischen den einzelnen Balken `gap`. Sie können annehmen, dass es für jede Höhe ein Label gibt.

Die Prozedur soll das Diagramm mit den gegeben Balken und Beschriftungen zeichnen, so wie es im obigen Bild zu sehen ist.

Verwenden Sie zur Implementierung die Prozedur `drawBar`.

```
static void drawBars(String[] labels, double[] heights, double width, double
gap) {
    // aktuelle Position auf der x-Achse
    double x = gap + width / 2;
    for (int i = 0; i < heights.length; i++) {
        drawBar(x, width, labels[i], heights[i]);
        x += width + gap;
    }
    // Trennlinie zwischen Text und Balken
    StdDraw.line(0, 0.1, 1, 0.1);
}
```

d) Schreiben Sie eine `main`-Prozedur für Ihr Programm.

Das Programm soll 2 Programm-Parameter nehmen:

1. Die Breite der Balken
2. Der Abstand zwischen den Balken

Die Werte und Beschriftungen für das Diagramm sollen von der Standardeingabe gelesen werden (bis die Eingabe leer ist). Dabei soll immer zuerst ein `String` gelesen werden, der die Beschriftung angibt und dann ein dazugehöriger `double`-Wert.

Das gezeichnete Diagramm soll entsprechend den gegebenen Parametern die Prozedur `drawBars` verwenden, um ein Balkendiagramm zu zeichnen.

```
java BarChart 0.1 0.03 < wahlen.txt
```

```
public static void main(String[] args) {
    double width = Double.parseDouble(args[0]);
    double gap = Double.parseDouble(args[1]);

    // Wiederverwenden von readStrings zum Einlesen der Labels und Werte
    String[] input = ReadStrings.readStrings();

    // Aufteilen der Eingabe in Labels und Werte:
    String[] labels = new String[input.length / 2];
    double[] values = new double[input.length / 2];
    for (int i=0; i<labels.length; i++) {
        labels[i] = input[i*2];
        values[i] = Double.parseDouble(input[i*2+1]);
    }

    drawBars(labels, values, width, gap);
}
```

e) (Zusatzaufgabe) Verbessern Sie Ihr Programm so, dass die Breite der Balken und der Abstand zwischen den Balken automatisch sinnvoll berechnet werden, wenn sie nicht als Programm-Parameter angegeben werden.

f) (Zusatzaufgabe) Normalisieren Sie die Eingabewerte so, dass der größte Balken eine Höhe von 80% des Bildschirms hat. Beim Normalisieren sollen alle Werte mit dem gleichen Faktor multipliziert werden.

```
    static double max(double[] values) {
        double result = values[0];
        for (int i = 1; i < values.length; i++) {
            result = Math.max(result, values[i]);
        }
        return result;
    }

    static void normalize(double[] values, double max) {
        double maxInArray = max(values);
        double scale = max / maxInArray;
        for (int i = 0; i < values.length; i++) {
            values[i] = values[i] * scale;
        }
    }

    public static void main(String[] args) {
        // Wiederverwenden von readStrings zum Einlesen der Labels und Werte
        String[] input = ReadStrings.readStrings();

        // Aufteilen der Eingabe in Labels und Werte:
        String[] labels = new String[input.length / 2];
        double[] values = new double[input.length / 2];
        for (int i=0; i<labels.length; i++) {
            labels[i] = input[i*2];
            values[i] = Double.parseDouble(input[i*2+1]);
        }

        if (labels.length == 0) {
            System.out.println("Keine Daten angegeben");
            return;
        }

        // Zusatzaufgabe: Breite und Abstand automatisch anpassen, wenn nicht
    spezifiziert:
        double width;
        double gap;
        if (args.length == 2) {
            width = Double.parseDouble(args[0]);
            gap = Double.parseDouble(args[1]);
        } else {
            // Abstand zwischen Balken macht 20% der Breite aus
            double relativeGapSize = 0.2;
            // Es soll gelten:
            // 1) n*width + (n+1)*gap == 1.0
            // 2) gap == relativeGapSize * (gap + width)
            // Daraus ergibt sich folgende Berechnung:
            double widthPlusGap = 1.0 / (labels.length + relativeGapSize);
            width = (1 - relativeGapSize) * widthPlusGap;
            gap = relativeGapSize * widthPlusGap;
        }

        // Zusatzaufgabe: Höhen normalisieren:
        normalize(values, 0.8);

        drawBars(labels, values, width, gap);
    }
```