Prof. Martin Hofmann, PhD
Dr. Steffen Jost

Ludwig-Maximilians-Universität München
Institut für Informatik
27. November 2017

Lösungsvorschlag zur 6. Übung zur Vorlesung
Einführung in die Programmierung

A6-1 ***Hello Array*** Betrachten wir noch einmal die Deklaration der `main`-Methode:

```
public static void main(String[] args) { ... }
```

Die `main`-Methode bekommt also ein Array über `String` übergeben! Dieses Array enthält alle Kommandozeilenargumente, welche beim Aufruf des Programms übergeben wurden:

```
> java Main a bbb   cc
```

Der Parameter `args` enthält dann ein Array mit den drei Strings `"a"`, `"bbb"` und `"cc"`. Alle Leerzeichen werden dabei verworfen.

Schreiben Sie ein simples Programm, welches alle nummerierten Kommandozeilenargumente in umgekehrter Reihenfolge ausgibt, also zum Beispiel für obigen Aufruf erhalten wir:

```
2: "cc"
1: "bbb"
0: "a"
```

LÖSUNGSVORSCHLAG:

```
public static void main(String[] args){
   for (int i = args.length - 1; i >=0; i--) {
       System.out.println(i + ": \"" + args[i] + "\"");
   }
}
```

A6-2 ***Umgang mit Arrays*** Wir üben die Verwendung von Arrays. Implementieren Sie folgende Methoden:

a) `public static String arrayToStr(String[] input, String glue)`
Die Einträge aus `input` werden in gegebener Reihenfolge aneinander gehängt. Dazwischen wird jeweils als Trennzeichen `glue` eingefügt. *Beipiele:*
`arrayToStr(new String[]{"a", "bc", "def"}, ", " )` liefert `"a, bc, def"`
`arrayToStr(new String[]{"40", "2"}, "+" )` liefert `"40+2"`

b) `public static String[] arraySpiegel(String[] input)`
Ein Array `input` wird gespiegelt, d.h. das erste Element wird mit dem letzten, das zweite mit dem vorletzten, usw. vertauscht.

c) `public static void inPlaceMirror(String[] input)`
Berechnet kein Ergebnis, sondern als *Seiteneffekt* werden die Elemente in `input` so verändert, dass das Array am Ende gespiegelt wurde.

Verwenden Sie dazu keine Hilfsarrays oder dergleichen, sondern ausschließlich das übergebene Array `input`! Eventuell benötigen Sie aber lokale Hilfsvariablen vom Typ `int` oder `String`. Man bezeichnet einen solchen Vorgang auch als "in place": das Ergebnis wird im Speicherplatz der Eingabe konstruiert; unabhängig von der Eingabe wird nur eine kleine Zahl von zusätzlichen Speicherstellen benötigt.

Hinweis: Wie in A3-2 „Zustandshölle" sehen wir wieder, dass Methoden Ihre Parameter in Java ändern können! Dies kann manchmal praktisch sein, aber im Allgemeinen ist davon abzuraten, da es dadurch schwieriger wird, den Code zu verstehen. Methoden sollten im Allgemeinen nur Instanzvariablen des eigenen Objekts `this` abändern; statische Methoden sollten ihre Argumente unverändert lassen und stattdessen ein Ergebnis zurückgeben.

Zur Vereinfachung dürfen Sie annehmen, dass weder die Parameter noch die Array-Einträge den Wert `null` haben. Die beiliegende Dateivorlage `ArraySpiegeln.java` enthält bereits eine `main`-Methode, mit der Sie Ihren Code so testen können:

```
> java ArraySpiegeln 1 2 3 4 5 6 7 8 9
Die Parameter lauten: 1, 2, 3, 4, 5, 6, 7, 8, 9
... und gespiegelt:   9, 8, 7, 6, 5, 4, 3, 2, 1
... und ursprünglich: 1, 2, 3, 4, 5, 6, 7, 8, 9
... wieder gespiegelt:9, 8, 7, 6, 5, 4, 3, 2, 1
```

LÖSUNGSVORSCHLAG:

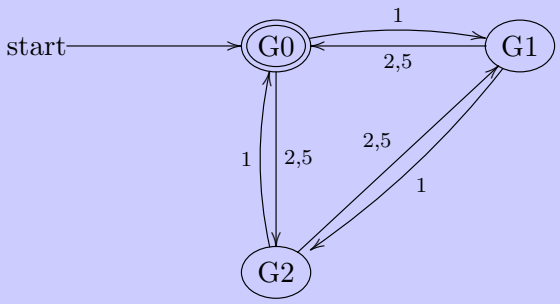
```
public class ArraySpiegeln {

  public static String arrayToStr(String[] input, String glue) {
    String result = "";
    for (int i=0; i<input.length-1; i++)
      result += input[i] + glue;
    if (input.length != 0)
      result += input[input.length-1];
    return result;
  }

  public static String[] arraySpiegel(String[] input) {
    String[] gespiegelt = new String[input.length];
    /* input ins Hilfsarray gespiegelt spiegeln ... */
    for (int i=0; i<input.length; i++)
      gespiegelt[i] = input[input.length-1-i];
    return gespiegelt;
  }

  public static void inPlaceMirror(String[] input) {
    for (int i=input.length/2-1; i>=0; i--) {
      String temp = input[i];
      input[i] = input[input.length-1-i];
      input[input.length-1-i] = temp;
    }
```

```
  }


/*
HINWEISE zu arraySpiegel und inPlaceMirror:
    + Bei ungerade Laenge muss das mittlere
      Element nicht mit sich selbst vertauscht werden.
    + input.length kann man sich vorher auch in einer lokalen Variablen merken,
      aber dies bringt keine nennenswerte Laufzeitverbesserung.
    + Die Schleifen koennen auch in entgegengesetzter Richtung durchlaufen werden.
*/

  public static void main(String[] arg) {
    System.out.println("Die Parameter lauten: " + arrayToStr(arg , ", "));
    String[] arg2 = arraySpiegel(arg);
    System.out.println("... und gespiegelt:   " + arrayToStr(arg2, ", "));
    System.out.println("... und ursprünglich: " + arrayToStr(arg , ", "));
    inPlaceMirror(arg);
    System.out.println("... wieder gespiegelt:" + arrayToStr(arg , ", "));
  }
}
```

A6-3 ***Kaugummi-Automat*** Stellen Sie sich einen Automaten vor, der Münzen zu 1, 2, und 5 Cent akzeptiert und dafür Kaugummis im Wert von 3 Cent pro Stück herausgibt. Diesen Automaten modellieren wir als deterministischen endlichen Automaten. Er soll Folgen von Ziffern aus $\{1, 2, 5\}$ annehmen und akzeptieren, wenn die Summe durch 3 teilbar ist.

a) Zeichnen Sie den Automaten zuerst als Graphen: Zustände als Knoten und Zustandsübergange als gerichtete, beschriftete Kanten. Markieren Sie Start- und Endzustand.

LÖSUNGSVORSCHLAG:



b) Geben Sie die Zustandsübergangsfunktion δ in tabellarischer Form an:

	1	2	5

LÖSUNGSVORSCHLAG:

δ	1	2	5
G0	G1	G2	G2
G1	G2	G0	G0
G2	G0	G1	G1

c) In der Vorlesung wurde ein Automat als Quintupel $(\Sigma, Q, \delta, q_0, F)$ definiert. Geben Sie die Werte von Σ, Q, δ, q_0 und F für Ihren Automaten an.

LÖSUNGSVORSCHLAG:

$\Sigma = \{1, 2, 5\}$
$Q = \{\mathrm{G0}, \mathrm{G1}, \mathrm{G2}\}$
$\delta = (\mathrm{G}q, \sigma) \mapsto \mathrm{G}\big((q + \sigma) \bmod 3\big)$ oder siehe Tabelle der vorangegangenen Teilaufgabe.
$q_0 = \mathrm{G0}$
$F = \{\mathrm{G0}\}$

d) Implementieren Sie den Automaten in Java!

Verwenden Sie dazu die beiliegende Dateivorlage `Kaugummi.java` und folgen Sie dem Implementierungsbeispiel aus der Vorlesung.

LÖSUNGSVORSCHLAG:

```
// Automaton, takes coins of 1, 2, or 5 cents and accepts if sum = 0 mod 3

class Mod3Automaton {

    private int state;           // 0, 1 or 2

    public Mod3Automaton()      { this.state = 0; }

    public void read (int coin) { this.state = (this.state + coin) % 3; }

    public boolean accepting()  { return this.state == 0; }
```

```
}

public class Mod3AutomatonTest {

    public static void test(String input) {

        Mod3Automaton automaton = new Mod3Automaton();
        boolean error = false;
        for (int i=0; i<input.length(); i++) {
            char c = input.charAt(i);
            if (c == '1') automaton.read(1);
            else if (c == '2') automaton.read(2);
            else if (c == '5') automaton.read(5);
            else {
                System.out.println("Illegal character: " + c);
                error = true;
            }
        }
        if (error) System.out.println("Faulty input.");
        else if (automaton.accepting())
             System.out.println("Digit sum is divisible by 3.");
        else System.out.println("Digit sum is not divisible by 3.");
    }

    public static void main(String[] args) {
        for (String s : args) {
            System.out.println("Running automaton on input " + s);
            test(s);
        }
    }
}
```

e) Die Vorlage enthält bereits eine `main`-Methode, welche für jedes Kommandozeilenargument Ihren Automaten mit diesem String als Eingabe startet.

Testen Sie Ihre Implementierung wie folgt:

```
java Mod3AutomatonTest 1 12 245 5chr0tt 125 1251 525252521 525252522
```

LÖSUNGSVORSCHLAG:

```
> java Mod3AutomatonTest 1 12 245 5chr0tt 125 1251 525252521 525252522
Running automaton on input 1
```

```
Digit sum is not divisible by 3.
Running automaton on input 12
Digit sum is divisible by 3.
Running automaton on input 245
Illegal character: 4
Faulty input.
Running automaton on input 5chr0tt
Illegal character: c
Illegal character: h
Illegal character: r
Illegal character: 0
Illegal character: t
Illegal character: t
Faulty input.
Running automaton on input 125
Digit sum is not divisible by 3.
Running automaton on input 1251
Digit sum is divisible by 3.
Running automaton on input 525252521
Digit sum is not divisible by 3.
Running automaton on input 525252522
Digit sum is divisible by 3.
```

H6-1 ***Mit Schleifen malen II*** (5 Punkte; MalSchleife.java)
Ähnlich zur herzhaften Aufgabe H5-2 möchten wir erneut eine gesamte Fläche färben und in einem GraphicsWindow darstellen.

Als Eingabe ist dieses Mal ein Array von gefärbten Punkten vorgegeben. Schreiben Sie ein Programm, welches ein GraphicsWindow öffnet, welches gerade groß genug ist, um alle gegebenen Punkte anzeigen zu können. Ihr Programm soll weiterhin jeden Punkt im Fenster mit der gleichen Farbe einfärben, wie der nächstgelegene Punkt im vorgegebenen Array!
Hinweise:

- Aus der Schule sollten Sie wissen, wie man den euklidischen Abstand zweier Punkte in der zweidimensionalen Ebene mit dem Satz des Pythagoras bestimmt!
- Dem Übungsblatt sollten 4 Dateien beliegen: `MalSchleife.java`, `ColoredPoint.java`, `DemoInput.java` und `GraphicsWindow.java`, welche kompilierbar und lauffähig sind.

  Verändern Sie zur Lösung der Aufgabe ausschließlich die Datei `MalSchleife.java` und geben Sie nur diese ab!
- Die Eingabe ist ein Array von Objekten der Klasse `ColoredPoint`, deren Programmcode Sie inzwischen lesen können sollten (abgesehen von Zeilen 32–26). Beachten Sie jedoch, dass die Klasse GraphicsWindow keine Methode zum Zeichnen von `ColoredPoint`

anbietet; verwenden Sie stattdessen die Methoden `GraphicsWindow.drawPoint` und `ColoredPoint.getPoint` oder `ColoredPoint.getX`/`ColoredPoint.getY`.

- Wer in der Schule nicht aufgepasst hat kann den Abstand zweier Punkte mit Koordinaten (x_1, y_1) und (x_2, y_2) durch die Formel $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ bestimmen.

LÖSUNGSVORSCHLAG:

```
public static GraphicsWindow zeichenSchleife(ColoredPoint[] punkte){
  // Zuerst Größe des Fensters bestimmen mit einer Schleife:
  int xmax = 50; // Minimale x-Größe
  int ymax = 50; // Minimale y-Größe
  for (ColoredPoint cp : punkte ) {
    xmax = Math.max(xmax,cp.getX());
    ymax = Math.max(ymax,cp.getY());
  }
  GraphicsWindow graph = new GraphicsWindow(xmax,ymax);
  graph.setText("Wir malen in einer Schleife...");
  // Alle Punkte einfärben, Reihenfolge der beiden for-Schleifen egal
  for (int y=0; y<ymax; y++) {
    for (int x=0; x<xmax; x++) {
      // Nächstgelegenden Punkt merken
      ColoredPoint closest = punkte[0];
      // Entfernung zum nächstegelegenen Punkt merken,
      // besser long statt int verwenden, wegen Quadrierung
      int distance = xmax * xmax + ymax * ymax + 1; //Größer als Größtmöglich
      for (ColoredPoint cp : punkte) {
        int xdiff = x - cp.getX();
        int ydiff = y - cp.getY();
        int dist_cp = xdiff * xdiff  +  ydiff * ydiff;
        // Ziehen der Wurzel zum Vergleich der beiden Längen ist überflüssig,
        // da die Wurzelfunktion ja streng monton ist!
        if (dist_cp < distance) {
          closest = cp;
          distance =dist_cp;
        }
      }
      graph.setColor(closest.getColor());
      graph.drawPoint(new Point(x,y));
      //graph.sleep(1); // Wenn man dabei zuschauen mag.
    }
  }
  return graph;
}
```

H6-2 ***Cha-Cha-Cha*** (6 Punkte; Abgabe: `H6-2.txt` oder `H6-2.pdf`)
EiP-Student Tomat Au möchte den Tanz „Cha-Cha-Cha“ erlernen, um seine Tanz-begeisterte Kommilitonin Tonia Tanzer zu beeindrucken. Leider sind alle Tanzkurse, welche Tomat frei verfügbar online findet nicht in der unmissverständlich klaren Sprache der Informatik verfasst.

Helfen Sie Tomat, indem Sie einen deterministischen endlichen Automaten (DEA) angeben, welcher Tomat erklärt, welche Figuren des vereinfachten Cha-Cha-Cha Tanzes in welcher Position jeweils möglich sind und zu welcher Position die Figur jeweils führt.

Für diese Aufgaben betrachten wir nur die folgenden vier Figuren mit den angegebenen Einschränkungen:

d	Damensolo	h	Hockey Stick (oder auch Linksdrehung)
f	Fächer	w	Wischer (oder auch Hand to Hand)

Die Figur „Wischer“ darf in der Grundhaltung getanzt werden, danach befindet man sich jedoch nicht mehr in der Grundhaltung. In dieser neuen Haltung darf „Wischer“ auch erneut getanzt werden, in allen übrigen Haltungen ist die Figur „Wischer“ jedoch nicht erlaubt.

Die Figur „Fächer“ darf nur immer einmal in Folge ausgeführt werden, entweder aus der Grundhaltung heraus, oder wenn vorher „Wischer“ ausgeführt wurde.

Die Figur „Hockey Stick“ darf nur unmittelbar nach der Figur „Fächer“ ausgeführt werden. Danach muss ein „Damensolo“ getanzt werden, welches einem in die gleiche Position bringt, wie nach Ausführung der Figur „Wischer“.

Das „Damensolo“ kann jederzeit getanzt werden und es führt meistens in die Grundhaltung zurück. Es gibt zwei Ausnahmen: Wird es aus der Grundhaltung heraus getanzt, so muss es gleich zwei mal ausgeführt werden. Danach befindet man sich in der gleichen Position wie nach Ausführung der Figur „Wischer“. Die zweite Ausnahme ist die bereits zuvor erwähnte Ausführung nach der Figur „Hockey Stick“.

Eine Abfolge von Figuren ist nur dann akzeptabel, wenn diese in der Grundhaltung begonnen werden kann und nach Ablauf wieder in der Grundhaltung endet.

Beispiele: Die Abfolge der Figuren „wd“, „ddd“, „wwwfd“, „wdfhdd“, „fhdfhdfd“ oder „wwdwwdwwdfdddd“ sind akzeptabel; dagegen sind zum Beispiel die Abfolgen „w“, „h“, „dd“, „wh“, „www“, „wwwfhd“ oder „dhfd“ nicht akzeptabel. Der DEA bekommt als Eingabewort also eine Abfolge von Figuren und akzeptiert genau dann, wenn die Abfolge der Figuren akzeptabel ist.

a) Überlegen Sie sich zuerst, welche Zustände Sie für den Automaten brauchen und fertigen Sie eine Zustandsübergangs-Tabelle an! Benutzen Sie so wenige Zustände wie möglich! (Sie werden 5–7 Zustände benötigen.)

   Die „Grundhaltung“ ist sicherlich ein benötigter Zustand, den wir mit G bezeichnen. Da nach obigen Angaben die Tänzer nach Ausführung von „Wischer“ nicht in der Grundhaltung sind, benötigen wir dafür einen weiteren Zustand. In diesem Zustand kann erneut „Wischer“ ausgeführt werden, welche uns aber wieder zu dem gleichen Zustand zurückführt (denn „Wischer“ darf mehrfach ausgeführt werden); aber mit der Figur „Damensolo“ kommen wir wieder in Zustand G zurück. Nach diesem Muster sind weitere Zustände hinzuzufügen.

   Da ein DEA von jedem Zustand aus für jedes Eingabezeichen einen Übergang haben muss, benötigten wir auch einen Zustand X, welcher einen Fehlerzustand darstellt. Dieser Fehlerzustand kann nie wieder verlassen werden, d.h. jede Eingabe weitere Eingabe

führt von X erneut zu X. Inakzeptable Abfolgen von Figuren können dort enden (z.B. „wh“) müssen es aber nicht (z.B. „w“).

LÖSUNGSVORSCHLAG:

Wir modellieren die Tanzpositionen wie folgt:

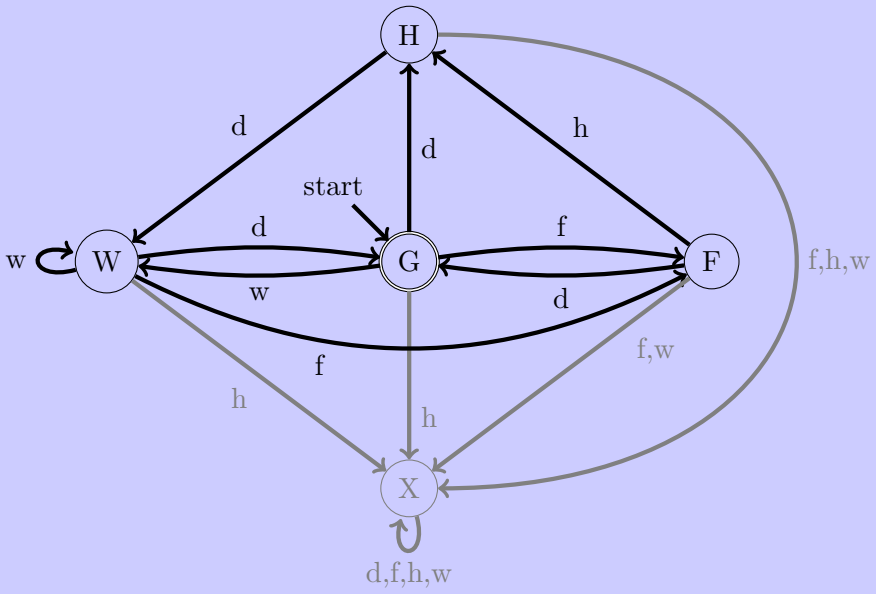
- G bedeutet Grundhaltung, Start- und einziger Endzustand
- W bedeutet nach ein oder mehreren „Wischern“
- F bedeutet nach einem „Fächer“
- H bedeutet nach einem „Hockey Stick“
- X bedeutet Fehlerzustand

Einen zusätzlichen Zustand „nach dem Damensolo aus dem Grundhaltung“ benötigen wir nicht, da dieser identisch zu Zustand H wäre: in beiden Fällen muss ein Damensolo erfolgen und wir sind in Zustand/Tanzposition W!

δ	d	f	h	w
G	H	F	X	W
W	G	F	X	W
F	G	X	H	X
H	W	X	X	X
X	X	X	X	X

b) Zeichnen Sie den DEA möglichst übersichtlich als Graph wie in Folie 5.19 gezeigt: Zustände als Knoten und Zustandsübergange als gerichtete, beschriftete Kanten (Pfeile). Markieren Sie den Startzustand und alle akzeptierenden Endzustände entsprechend!

LÖSUNGSVORSCHLAG:



Wichtig: Für einen DEA gilt, dass von jedem Knoten für jede Eingabe ein Pfeil abgeht!

Den Fehlerzustand X und alle Pfeile dorthin haben wir zur Verbesserung der Übersichtlichkeit grau dargestellt. Dies hat aber keine weitere Bedeutung. In der Literatur werden bei der Darstellung von DEAs solche Fehlerzustände oft gar nicht eingezeichnet, sie sind aber trotzdem vorhanden.

c) Geben Sie den Automaten nun als Quintupel an, wie in der Vorlesung gezeigt,

LÖSUNGSVORSCHLAG:

$$\begin{array}{lcl} \Sigma & = & \{f, h, d, w\} \\ Q & = & \{\text{F,G,H,W,X}\} \\ \delta & = & \text{gemäß Tabelle aus erster Teilaufgabe} \\ q_0 & = & \text{G} \\ F & = & \{\text{G}\} \end{array}$$

LÖSUNGSVORSCHLAG:

Hinweis: Wer in einem 2-seitigen Aufsatz begründet, dass Tänze in Wirklichkeit zwingend nicht-deterministische Abläufe haben, darf in Zukunft auch gerne nicht-deterministische endliche Automaten (NEA) basteln. Es ist aber unklar, ob dies die Aufgabe für die EiP-Teilnehmer schwerer oder leichter machen würde. `;)`

Abgabe: Lösungen zu den Hausaufgaben können bis Sonntag, den 3.12.17, mit UniWorX nur als `.zip` abgegeben werden. Aufgrund des Klausurbonus müssen die Hausaufgaben von Ihnen alleine gelöst werden. Abschreiben bei den Hausaufgaben gilt als Betrug und kann zum Ausschluss von der Klausur zur Vorlesung führen. Bitte beachten Sie auch die Hinweise zum Übungsbetrieb auf der Vorlesungshomepage (`www.tcs.ifi.lmu.de/lehre/ws-2017-18/eip/`).