TU Ilmenau, Fakultät IA
FG Telematik/Rechnernetze
Prof. Dr.-Ing. G. Schäfer
https://www.tu-ilmenau.de/telematik/pua

TECHNISCHE UNIVERSITÄT
ILMENAU

Programmierung und Algorithmen – WS 26/27

Übungsblatt 3

Die Lösungen der Aufgaben sind bis zum 08.11.26, 23:59 Uhr abzugeben.

Die Besprechung der Aufgaben erfolgt in KW 46.

Aufgabe 1 (Übergabe von Parametern in Java) **2 + 2 + 0 Punkte**

Gegeben sei das unten stehende Java-Programm.

(a) Wie lautet die Ausgabe des Programms wenn es mit dem Kommandozeilenparameter „`Hello?`“ aufgerufen wird?

(b) Wie müssen Sie den Methodenaufruf in Zeile 7 ändern, so dass der zweite Teil (d.h. nach dem Komma) der Ausgabe aus Zeile 7 stets der erste Kommandozeilenparameter ist? Beachten Sie, dass die Methode `fun` nach wie vor aufgerufen werden soll (Zeile also nicht einfach löschen)! Änderungen in anderen Zeilen sind nicht erlaubt.
Wichtig: Nach Ihrer Modifikation soll der Inhalt von `args` noch immer die Funktion `fun` erreichen.

(c) **Zusatzaufgabe:** Bei zweidimensionalen Arrays (z.B. `int[][] matrix`) müssen weitere Vorkehrungen getroffen werden. Welche und warum?

```
1  public class Parameter {
2    public static void main(String[] args) {
3      if(args.length == 0) {
4        return;
5      }
6      int a = 42;
7      fun(a, args); // <-- Zeile 7
8      System.out.println(a + ", " + args[0]);
9      String[] array = new String[] { "Swapped!" };
10      swap(array, args);
11      System.out.println(args[0]);
12   }
13
14   public static void fun(int a, String[] array) {
15     a = array.length;
16     array[0] = "Hello!";
17   }
18
19   public static void swap(String[] a, String[] b) {
20     String[] tmp = a;
21     a = b;
22     b = tmp;
23   }
24 }
```

Hinweis: Sie können sich zunächst z.B. hier[1] in die Thematik einlesen (ohne den Teil zu „RMI“).

Bitte wenden!

[1] http://www.javadude.com/articles/passbyvalue.htm

Aufgabe 2 (Mehrdimensionale Mustersuche) **6 + 1 + 1 Punkte**

In dem auf unserer Webseite oder in der WebIDE zu Verfügung gestellten Java-Programm[2] sollen Muster in einem zweidimensionalen Feld gefunden werden. Wesentliche Teile sind jedoch noch nicht implementiert.

(a) Ergänzen Sie die Funktion `find1(char[][] a, char[][] m)`, welche im zweidimensionalen Feld `a` nach dem ersten Vorkommen des in `m` übergebenen Musters sucht. Falls das Muster vorkommt, existieren Koordinaten (x_a, y_a) in `a`, so dass für alle Positionen (x_m, y_m) in `m` die Aussage `a[`x_m`+`x_a`][`y_m`+`y_a`] == m[`x_m`][`y_m`]` zutrifft. Wurde ein Vorkommen gefunden, geben Sie dessen Koordinaten auf der Kommandozeile aus. Ist das Muster nicht in `a` enthalten, geben Sie $(-1, -1)$ aus.

(b) Erweitern Sie die Methode `find2(char[][] a, char[][] m)`, so dass sie prinzipiell wie `find1` arbeitet, jedoch an Positionen des Musters welche mit ‚`*`' besetzt sind beliebige Inhalte zulässt.

(c) Welche Ausgabe liefert Ihr Programm?

Hinweis: Beachten Sie, dass zweidimensionale Felder in Java eindimensionale Felder sind, die in jeder Position eine Instanz eines Feld-Typs enthalten. Insbesondere können die ‚enthaltenen' Felder verschiedene Längen besitzen. Plastisch ausgedrückt *muss das zweidimensionale Feld somit nicht rechteckig sein.*

Aufgabe 3 (Perfekte Zahlen in Java) **4 Punkte**

Eine Zahl $n \in \mathbb{N}$ heißt perfekt oder vollkommen, wenn die Summe ihrer Teiler gleich n ist. Dabei werden nur die Teiler echt kleiner als n betrachtet. Beispielsweise ist `6 = 1 + 2 + 3` eine perfekte Zahl. Schreiben Sie ein Java-Programm, das alle perfekten Zahlen bis 5000 bestimmt. Geben Sie diese auch in Ihrer Lösung an!

Aufgabe 4 (Perfekte Zahlen applikativ) **5 Punkte**

Formulieren Sie einen *applikativen Algorithmus* `perfect(n)` welcher bestimmt, ob die Eingabe `n` $\in \mathbb{N}$ eine perfekte Zahl ist. Halten Sie sich an die Definition eines applikativen Algorithmus aus der Vorlesung!

Hinweise:

- Sie werden eine Hilfsfunktion benötigen.
- Java-Methoden sind keine applikativen Algorithmen. Halten Sie sich an die Definition eines applikativen Algorithmus aus der Vorlesung und beachten Sie die dortige Syntax! Wenn Sie hier Java-Code abgeben wird Ihre Lösung nicht gewertet.

[2] https://www.tu-ilmenau.de/fileadmin/Bereiche/IA/telematik/Lehre/Programmierung_und_Algorithmen/WS_2026_files/MultiDimSearch.java

Aufgabe 5 (Fibonacci-Zahlen effizient) **4 Punkte**

In der Vorlesung wurde Ihnen ein applikativer Algorithmus zur Berechnung der Fibonacci-Zahlen vorgestellt (Kapitel 4, Folie 16). Leider ist dieser Algorithmus nicht besonders effizient, da viele Zwischenergebnisse mehrfach berechnet werden. Beispielsweise werden beim Aufruf von `fib(5)` die beiden rekursiven Aufrufe `fib(3)` und `fib(4)` getätigt. In beiden dieser Fälle wird unabhängig voneinander (unter anderem) `fib(2)` als Zwischenergebnis berechnet.

Folgende Java-Methode umgeht dieses Problem, indem die komplette Fibonacci-Folge, beginnend bei f_0, bis zur gewünschten Fibonacci-Zahl in einer Schleife berechnet wird:

```
1  public static int fib(int n) {
2      // die Variablen e und f enthalten immer zwei aufeinanderfolgende
3      // Fibonacci-Zahlen, am Anfang die ersten beiden
4      int e = 1;
5      int f = 1;
6      for(int i = 0; i < n; ++i) {
7          int tmp = f;
8          f = e + f; // neues f ergibt sich aus der nächsten Fibonacci-Zahl
9          e = tmp; // neues e ergibt sich aus dem alten f
10      }
11      return e; // wenn i = n gilt (siehe for-Schleife), ist e das Ergebnis
12 }
```

Formulieren Sie nun einen *applikativen* Algorithmus, der dieselbe Idee wie die Java-Methode verfolgt. Denken Sie daran, dass Sie Schleifen in einem applikativen Algorithmus nicht direkt verwenden können.

Hinweis: Definieren Sie eine Hilfsfunktion `h(n, i, e, f)`.

Hinweis: Java-Methoden sind keine applikativen Algorithmen. Halten Sie sich an die Definition eines applikativen Algorithmus aus der Vorlesung und beachten Sie die dortige Syntax! Wenn Sie hier Java-Code abgeben wird Ihre Lösung nicht gewertet.