universität ulm

Einführung in die Informatik

Weihnachtstutorium • WiSe 2021/22

22. Dezember 2021

Florian Sihler (florian.sihler@uni-ulm.de)

Einführende Worte:

Dies sind ein paar Aufgaben von derem Durchlesen ich euch abzuhalten versuche indem sie alle auf einer einzelnen Seite stehen. Die Bearbeitung wird ein wenig wie folgt ablaufen — hoffentlich — ich rede, ihr bleibt topfit und motiviert und zwischendurch werden wir auf ein paar der Aufgaben zurückkommen. So a „have fun“ for starters!

1 Leonardo-Zahlen

Mittel, 7 Minuten

Analysieren Sie den Algorithmus hinsichtlich seiner totalen Korrektheit im Bezug auf die Berechnung der Leonardo–Zahlen. Weisen Sie dafür alle notwendigen Eigenschaften nach oder liefern Sie geeignete Gegenbeispiele.

Die Leonardo-Zahlen sind definiert als (für $n \geq 0$):

$$L(0) = 1,\ L(1) = 1 \qquad L(n) = L(n-1) + L(n-2) + 1$$

```
public static long L(int n) {
    if(n == 0 || n == 1) return 1;
    int l1 = 1, l2 = 1;
    for(int i = 0; i < n - 1; i++) {
        int tmp = l1;
        l1 = l2;
        l2 = tmp + l2 + 1;
    }
    return l2;
}
```

2 Algorithmusanalyse

Schwer, 3 + 4 + 10 Minuten

Lesen Sie sich den Anforderungstext durch und bearbeiten Sie dann die drei folgenden Teilaufgaben.

Guten Tag ich bin Besitzer eines Plätzchenladens. Dafür verwende ich verschiedene Mehle (Weizen oder Dinkel), verwende Zucker oder Süßungsmittel, sowie stets eine weitere Komponente (Kokos oder Zimt).

Da ich alle Plätzchen gleich gern habe, werde ich sie alle immer zum selben Preis verkaufen. Geld an sich ist auch gar nicht das Problem, leider können mir meine Zulieferer immer nur eine begrenzte Menge (in Gramm) an den jeweiligen Stoffen zur Verfügung stellen - die kann ich ihnen aber geben. Die habe ich. Weiter habe ich pro Jahr immer eine handvoll an Rezepten, die immer eine bestimmte Anzahl eines Mehls, eines Süßungsstoffes und einer weiteren Komponente pro Plätzchen benötigen.

Auf der Basis hätte ich jetzt gerne eine Möglichkeit herauszufinden, wie ich insgesamt immer maximal viele Plätzchen aus den gegebenen Ressourcen und Rezepten backen kann.

a) Führen Sie eine Problemspezifikation durch

b) Führen Sie eine Problemabstraktion durch

c) Entwickeln Sie einen Pseudocode-Algorithmus, welcher das Problem löst

3 Typidentifikation

Leicht, $1 + 2 + 2 + 1$ Minuten

Bestimmen Sie für jeden der folgenden Java-Ausdrücke den Typ *jedes Teilausdrucks*, sowie das Gesamtergebnis. Sie dürfen Rundungsprobleme der Sprache ignorieren:

a) `14 - 3D * 2 / 12`

b) `(char) ((byte) 'a' + 3) + " Sonne"`

c) `14 - 2 + (3 > 9 || true ^ false ? "Hallo" : "Welt")`

d) `7 / 2 - 3 * 2.5 + 2f`

4 Programmausgabe

Leicht, 2 + 3 + 2 + 1 Minuten

Betrachten Sie das folgende Java-Programm und geben Sie für die kommentierten Zeilen jeweils an, welche Ausgabe diese erzeugen (tragen Sie diese in die entsprechende Box ☐ ein). Beschreiben Sie weiter kurz, warum die jeweilige Ausgabe zustande kommt, indem Sie die Werte der beteiligten Variablen erklären.

```
public class Example {
    public static int main;
    public static String a = "Hallo";

    public static void main(String[] args) {
        int a = 7;
        int b = 15;
        System.out.println(a + "; " + main); // i)
        {
            float main = 3 * b++;
            Example.a += b;
            System.out.println(b + "; " + main); // ii)
        }
        System.out.println(a + "; " + b + "; " + main); // iii)
        System.out.println(Example.a + "; " + Example.main); // iv)
    }
}
```

i)

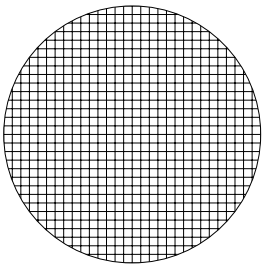
ii)

iii)

iv)

5 Kachelland

Mittel, 5 + 4 + 3 Minuten



Sie befinden sich in Kachelland, welches auf einem schachbrettartigen Feld geplant wurde. Für diese Aufgabe betrachten wir die Fortbewegungskosten, die ein Charakter auf den jeweiligen Feldern hat, wobei wir zwei Formen des Transports unterscheiden: ohne Hilfsmittel und im Boot. Auf diese Weise definieren wir für jede Kachel des zweidimensionalen Landes wie teuer es ist sie ohne Hilfsmittel, oder mit einem Boot, zu überqueren und geben eine Zeit in Minuten an. Dafür verwenden wir ein Array mit zwei Elementen: `cell = {⟨foot⟩, ⟨boat⟩}`. Eine negative Zahl kennzeichnet, dass es unmöglich ist, das Feld mit dem Transportmittel zu überqueren. Konstruieren Sie die folgenden drei Methoden in Java. Achten Sie auch auf die Rückseite, des Blattes:

1. `static int[][][] newLand(int width, int height)`
   Erschaffen Sie ein neues rechteckiges Kachelland mit den gegebenen Dimensionen, in dem kein Feld passierbar ist. Sie dürfen annehmen, dass weder `width` noch `height` negativ sind.

2. `static boolean set(int[][][] board, int x, int y, int foot, int boat)`
   Setzt die Fortbewegungskosten des Feldes (x, y) auf `{⟨foot⟩, ⟨boat⟩}`. Prüfen Sie ob die Koordinaten auch wirklich auf dem Brett existieren und geben Sie über den `boolean` zurück, ob die Kosten tatsächlich gesetzt werden konnten.

3. `static int costs(int[][][] board, int x, int y, boolean byBoat)`
   Geben Sie die Kosten des jeweiligen Feldes zurück. Dabei entscheide `byBoat`, ob die Kosten ohne Hilfsmittel (`byBoat=false`) oder in einem Boot (`byBoat=true`) von Interesse sind. Ist das Feld nicht auf dem Brett, so ist es per Konvention „nicht passierbar“.

```
public class TileCountry {
    static int[][][] newLand(int width, int height) {




    }
// } → Nächste Seite
```

```
// Fortsetzung
// public class TileCountry {
    static boolean set(int[][][] board, int x, int y, int foot, int boat) {









    }

    static int costs(int[][][] board, int x, int y, boolean byBoat) {









    }
}
```

6 Schleifenspiel

Mittel, 4 + 4 Minuten

Im Folgenden finden Sie zwei Code-Ausschnitte. Schreiben Sie diese so um, dass sie nur noch Schleifen der angegebenen Art verwenden, ohne dass sich das Verhalten des Codes verändert. Dabei repräsentiert `k` eine natürliche Zahl (exklusive 0), welche Sie nicht kennen.

1. Verwenden Sie eine `do-while` Schleife:

```
int[] x = new int[]{17, 22, 13, 0};
for(int i = 0; i < x.length - k; i += 1) {
    System.out.println(x[i]);
}
```

2. Verwenden Sie eine `for`-Schleife:

```
int i = 1;
do {
    i *= k;
    k -= i;
} while(k >= 5);
System.out.println(i);
```

7 Methodenverstehung

Mittel, 2 + 2 + 3 + 3 Minuten

Betrachten Sie das folgende Java-Programm und geben Sie für die kommentierten Zeilen jeweils die Werte der Variablen `a`, `b` und `c` an (tragen Sie diese in die entsprechende Box ☐ ein). Erklären Sie die Ausgaben dabei durch eine kurze Erläuterung der beteiligten Java-Mechanismen. Relevant sind: (i) Überladungen, (ii) Seiteneffekte, (iii) Überschattungen und (iv) Standardwerte. Sie dürfen in ihrer Skizzierung die römischen Zahlen verwenden, um auf die Mechanismen hinzuweisen.

```
H → 72
a → 97
l → 108
o → 111
```

```java
public class MethodExample {
   public static char a;
   public static int b = 5;
   public static int c = 3;

   static int x(int a) {
      a = a * a;
      return b + 2;
   }

   static int x(char[] b) {
      a += b[1];
      return b[0];
   }

   static String x() {
      String b = "Hallo";
      return b;
   }

   public static void main(String[] args) {
      // 1)
      String c = x();
      // 2)
      int temp = x(c.toCharArray());
      // 3)
      int b = x(temp);
      // 4)
   }
}
```

1) a: b: c:

2) a: b: c:

3) a: b: c:

4) a: b: c:

8 Reverse

Leicht, 5 Minuten

Schreiben Sie eine Methode `static void reverse(int[] arr)` welche das übergebene Array umdreht, sodass das erste Elemente nun an letzter Stelle steht, das zweite an vorletzter Stelle, ... und das letzte Element nun an erster Stelle steht.

```
static void reverse(int[] arr) {




}
```

9 Kameraszenario

Mittel, 2 + 6 Minuten

Im Folgenden sehen Sie das Grundgerüst einer Kameraklasse. Bearbeiten Sie die folgenden beiden Teilaufgaben, welche aufeinander aufbauen. Sie dürfen keine (Java-)Ausnahmen verwenden.

1. Schreiben Sie einen leeren Konstruktor, welcher die von Java zugewiesenen Standardwerte explizit initialisiert. Dieser Konstruktor soll nicht aus Code abseits der `Camera`-Klasse aufrufbar sein und so verhindern, dass weitere Kamera-Objekte erstellt werden können.
2. Erweitern Sie die Klasse nun in sofern, dass es immer maximal fünf Kamera-Objekte gibt, welche durch `get(int)` durch eine Zahl $x \in \{0, 1, 2, 3, 4\}$ abgefragt werden können. Dabei soll ein Kamera-Objekt erst (mit ihrem vorher erzeugten Konstruktor) erzeugt werden, wenn es durch die Zahl angefragt wird. Weiter sollen Folgeanfragen mit der Zahl dasselbe Objekt zurückliefern. Geben Sie für $x < 0$ und $x \geq 5$ den Wert `null` zurück.

Falls Ihnen der Platz nicht reicht, markieren Sie bitte die Fortsetzung auf dem letzten Blatt.

```
public final class Camera {
   private int x, y, z;
   private boolean recording;
```

```
   public void move(int dx, int dy, int dz) { x += dx; y += dy; z += dz; }
   public void start() { this.recording = true; }
   public void stop() { this.recording = false; }
   public int[] position() { return new int[]{x, y, z}; }
   public boolean running() { return this.recording; }
}
```

10 Fehlersuche

Mittel, 9 Minuten

Im Code auf der nächsten Seite finden sich **10 Fehler**, welche vom Java-Compiler erkannt werden. Identifizieren Sie **acht** dieser Fehler durch

1. die Angabe der Code-Zeilennummer und
2. eine kurze Erklärung, was den Fehler verursacht.

Tragen Sie die Informationen in die Tabelle ein. Wenn Sie mehrere Fehler in einer Code-Zeile finden, tragen sie diese separat ein — die zusätzlichen Zeilen existieren dabei nur, um Ihnen eine Selbstkorrektur zu ermöglichen. Folgefehler zählen dabei nicht als separate Fehler. Wird also beispielsweise ein Bezeichner inkorrekt initialisiert, so zählt nicht jede Verwendung als zusätzlicher Fehler sondern nur die inkorrekte Initialisierung.

#	Zeile	Grund
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		

```
public class iDoGood {
    final void i = 0x7;
    private final String omega = "Sunshine";

    static int iMultiply(final char x, double y) {
        return x * y;
    }

    public IDoGood(String dromeda) {
        this.omega = dromeda;
    }

    static int double(byte i) {
        return 2 * ++i;
    }

    private iDoGood() {
        this(andromeda);
    }

    private boolean iDoGood() {
        if(this.omega.equals("andromeda"));
        int i = 0 // /** // **/ /* ;
        for(;i < 2_00;i++, i += 2)
            System.Err.println(i);
        return false /** // */ ;
    }

    static class iDoBetta {
        global doBetta() {
            iDoGood veryGud = iDoGood();
            System.out.println(veryGud.omega);
        }
    }
}
```

```
// Freie Seite als zusätzlicher Platz
```