ulm university universität
uulm

Praktische Zulassungsklausur (A) zur Vorlesung

Einführung in die Informatik II - Vertiefung

im Sommersemester 2019

Dr. J. Kohlmeyer, T. Heß

08.06.2019

Musterlösung

Aufgabe 1 - Allgemeines

3 + 3 + 3 + 3 = 12 Punkte

a) Implementieren Sie eine Methode `public static void firstSteps()`, welche die Zahlen von 1 bis 10 in eine Zeile ausgibt und dann in einer weiteren Zeile deren Quadrate. Formatieren Sie ihre Ausgabe wie die folgende Ausgabe.

Ausgabe:

```
1 2 3 4 5 6 7 8 9 10
1 4 9 16 25 36 49 64 81 100
```

Lösung

```java
public static void firstSteps() {
    for (int i = 1; i <= 10; i++) {
        System.out.print(i + " ");
    }
    System.out.println();

    for (int i = 1; i <= 10; i++) {
        System.out.print(i * i + " ");
    }
    System.out.println();
}

//Alternativ
public static void firstSteps() {
    System.out.println("1 2 3 4 5 6 7 8 9 10");
    System.out.println("1 4 9 16 25 36 49 64 81 100");
}
```

b) Implementieren Sie eine Methode `public static char[] toCharArray(String str)`, welche einen `String` übergeben bekommt und ein Array mit dessen Zeichen zurückgibt.

Lösung

```java
public static char[] toCharArray(String str) {
    return str.toCharArray();
}
```

c) Eine Zahl ist durch 3 teilbar, wenn ihre Quersumme durch 3 teilbar ist. Implementieren Sie eine Methode `public static boolean isDivBy3(int n)`, welche genau dann `true` zurückliefert, wenn `n` durch 3 teilbar ist.

Lösung

```java
public static boolean isDivBy3(int n) {
    return n % 3 == 0;
}
```

d) Implementieren Sie eine Methode `public static String[] getSentences(String str)`, welche, der Reihe nach, jeden in `str` vorkommenden Satz in ein Feld eines zu erstellenden Arrays schreibt und dieses am Ende zurückgibt. Sie können davon ausgehen, dass jeder Satz mit einem Punkt (`.`) oder Ausrufezeichen (`!`) endet. Punkte, Ausrufezeichen und Leerzeichen zwischen den Sätzen können, müssen aber nicht, im resultierenden Array vorkommen.

Beispiel: `getSentences("A. B. Test!")` $\rightarrow$

0	*1*	*2*
"A"	*"B"*	*"Test"*

Lösung

```
1  public static String[] getSentences(String str) {
2      str = str.replaceAll("\\.", "!");
3      return str.split("!");
4  }

5  //Alternativ
6  public static String[] getSentences(String str) {
7      return str.split("[.!]");
8  }
```

Aufgabe 2 - Graphische Oberfläche **12 Punkte**

a) Implementieren Sie eine graphische Oberfläche, welche aussieht, wie die in Abb. 1 abgebildete.
Hinweis: Benutzen Sie für die schwarze Fläche ein `JPanel`.

b) Bei einem Klick auf den Button, soll die schwarze Fläche weiß (sh. Abb. 2) werden.

c) Abhängig von der Farbe der Fläche soll bei einem Klick auf den Button die Farbe der Fläche ändern.

- Fläche ist schwarz $\xrightarrow{*klick*}$ Fläche ist weiß
- Fläche ist weiß $\xrightarrow{*klick*}$ Fläche ist schwarz

Abb. 1

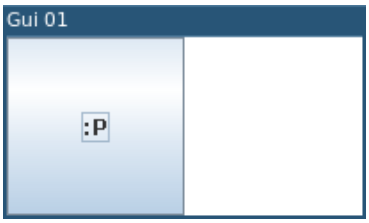

Abb. 2

Lösung auf der nächsten Seite

Lösung

```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class Gui01 {

    public static void main(String[] args) {

        JFrame frame = new JFrame("Gui 01");
        frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);


        frame.setPreferredSize(new Dimension(200, 100));

        JPanel panel = new JPanel();
        panel.setBackground(Color.BLACK);

        JButton button = new JButton(":P");
        button.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent actionEvent) {

                if(panel.getBackground() == Color.BLACK){
                    panel.setBackground(Color.WHITE);
                }else{
                    panel.setBackground(Color.BLACK);
                }
            }
        });

        frame.getContentPane().setLayout(new GridLayout(1, 2));
        frame.getContentPane().add(button);
        frame.getContentPane().add(panel);

        frame.pack();
        frame.setLocationRelativeTo(null); //optional
        frame.setResizable(false); //optional

        frame.setVisible(true);
    }
}
```

Aufgabe 3 - Dateiverarbeitung **12 Punkte**

Eine csv-Datei enthält durch Semikola getrennte Werte, in unserem Fall in jeder Zeile gleich viele. Im Ordner `~/material` finden Sie eine csv-Datei `data01.csv`, kopieren Sie diese in ihr Projekt (auf die gleiche Ebene, wie auch ihr `src`-Ordner).

a) Implementieren Sie die Funktionalität um die Werte aus der csv-Datei einzulesen.

b) Parsen Sie die Werte als `double`. Berechnen Sie die Summe und den Durchschnitt jeder Zeile und geben Sie diesen gemeinsam mit der Zeilennummer in die Konsole aus. Formatieren Sie ihre Ausgabe genau wie im nachfolgenden Ausschnitt.
Ausgabe der ersten 2 Zeilen:

```
0 93.31 5.83
1 75.56 4.72
```

Lösung

```java
//Importe (java.io.IOException, java.nio.file.Files,
//         java.nio.file.Path, java.nio.file.Paths, java.util.List)
public class ExerciseA03 {

    public static void main(String[] args) {

        Path path = Paths.get("data01.csv");
        List<String> raw = null;

        try {
            raw = Files.readAllLines(path);
        } catch (IOException e) {
            e.printStackTrace();
        }

        if (raw == null) {
            return;
        }

        for (int i = 0; i < raw.size(); i++) {
            String[] splitData = raw.get(i).split(";");
            double sum = 0;
            for (String val : splitData) {
                sum += Double.parseDouble(val);
            }

            System.out.printf("%d %.2f %.2f%n", i, sum,
                sum / splitData.length);
        }
    }
}
```

Aufgabe 4 - Listen **12 Punkte**

Im Ordner `~/material` finden Sie eine Datei `materialListe01.zip`, welche eine rudimentäre Implementierung einer einfach-verketteten Liste beinhaltet.

a) Importieren Sie das Material.

b) Die Liste soll mit einer forEach-Schleife durchlaufen werden können, implementieren Sie diese Funktionalität. Implementieren Sie dazu in der Klasse `IntList` das Interface `Iterable<Integer>` und in einer weiteren Klasse `IntListIterator` das Interface `Iterator<Integer>`.

c) Schreiben Sie in einer Klasse `IntListTests` einen JUnit-Test der die implementierte Funktionalität geeignet testet.

`IntList`

Lösung

```
1  import org.junit.Test;
2  import java.util.Iterator;
3  public class IntList implements Iterable<Integer> {
4      IntElement first;
5      public IntList prepend(int val) {
6          //wie gegeben
7      }
8      @Override
9      public Iterator<Integer> iterator() {
10         return new IntListIterator(this, first);
11     }
12     @Test
13     public void testIterator() {
14         IntList list = new IntList();
15         list.prepend(5).prepend(4).prepend(3).prepend(2).prepend(1);
16         IntElement temp = list.first;
17         for (Integer val : list) {
18             assert val == temp.val;
19             temp = temp.next;
20         }
21     }
22 }
```

`IntListIterator`

Lösung

```java
import java.util.Iterator;

public class IntListIterator implements Iterator<Integer> {

    private IntList list;
    private IntElement current;

    public IntListIterator(IntList list, IntElement first) {
        this.list = list;
        this.current = first;
    }

    @Override
    public boolean hasNext() {
        return current != null;
    }

    @Override
    public Integer next() {
        int val = current.val;
        current = current.next;
        return val;
    }
}
```