



Praktische Zulassungsklausur (C) 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 zuerst alle geraden Zahlen zwischen 1 und 10 und dann alle ungeraden Zahlen zwischen 1 und 10 in die gleiche Zeile ausgibt. Im Anschluss soll umgebrochen werden. Formatieren Sie ihre Ausgabe wie die folgt:

**Ausgabe:**

2 4 6 8 10 1 3 5 7 9

**— Lösung —**

```
1 public static void firstSteps() {
2     for (int i = 1; i <= 10; i++) {
3         if (i % 2 == 0) {
4             System.out.print(i + " ");
5         }
6     }
7     for (int i = 1; i <= 10; i++) {
8         if (i % 2 != 0) {
9             System.out.print(i + " ");
10        }
11    }
12    System.out.println();
13 }
```

- b) Implementieren Sie eine Methode `public static String toLowerCase(String str)`, welche einen String übergeben bekommt und alle Großbuchstaben durch Kleinbuchstaben ersetzt.

**Beispiel:** `toLowerCase("Ananas")` → `"ananas"`**— Lösung —**

```
1 public static String toLowerCase(String str) {
2     return str.toLowerCase();
3 }
```

- c) Eine Zahl ist durch 9 teilbar, wenn ihre Quersumme durch 9 teilbar ist. Implementieren Sie eine Methode `public static boolean isDivBy9(int n)`, welche genau dann `true` zurückliefert, wenn `n` durch 9 teilbar ist.

**Beispiele**

- `isDivBy9(9)` → `true`
- `isDivBy9(423)` → `true`
- `isDivBy9(26)` → `false`

**— Lösung —**

```
1 public static boolean isDivBy9(int n) {
2     return n % 9 == 0;
3 }
```

- d) Implementieren Sie eine Methode `public static void printDuplicates(String[] strings)`, welche alle Strings ausgibt, die im Array `strings` mehr als einmal vorkommen. In der Ausgabe soll jeder String maximal einmal vorkommen, die Reihenfolge ist egal.

**Ausgabe** für `printDuplicates(new String[]{"A", "A", "C", "B", "A", "B"})`

A

B

### — Lösung —

```
1 public static void printDuplicates(String[] strings) {
2     ArrayList<String> duplicates = new ArrayList<>();
3     for (int i = 0; i < strings.length; i++) {
4         for (int j = i + 1; j < strings.length; j++) {
5             if (strings[i].equals(strings[j])) {
6                 if (!duplicates.contains(strings[i])) {
7                     duplicates.add(strings[i]);
8                     break;
9                 }
10            }
11        }
12    }
13    for (String duplicate : duplicates) {
14        System.out.println(duplicate);
15    }
16 }
```

## Aufgabe 2 - Graphische Oberfläche

12 Punkte

- a) Implementieren Sie eine graphische Oberfläche, welche aussieht, wie die in Abb. 1 abgebildete.
- b) Implementieren Sie eine Funktionalität welche für jeden Button zählt wie oft dieser angeklickt wurde. Die Summe der beiden Zähler soll im Fenstertitel stehen. In Abb. 2 ist die Gui dargestellt, nachdem Button 1 dreimal und Button 2 fünfmal geklickt wurde.



Abb. 1



Abb. 2

## — Lösung —

```
1 //Importe (javax.swing.*, import java.awt.*,
2 //      java.awt.event.ActionEvent, java.awt.event.ActionListener)

3 public class Gui03 {

4     private static int a;
5     private static int b;

6     public static void main(String[] args) {

7         JFrame frame = new JFrame("Gui 03");
8         frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

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

10        JButton button1 = new JButton("1");
11        button1.setActionCommand("A");
12        JButton button2 = new JButton("2");
13        button2.setActionCommand("A");

14        ActionListener ctrl = new ActionListener() {
15            @Override
16            public void actionPerformed(ActionEvent actionEvent) {

17                if (actionEvent.getActionCommand().equals("A")) {
18                    a++;
19                } else {
20                    b++;
21                }

22                frame.setTitle((a + b) + "");
23            }
24        };

25        button1.addActionListener(ctrl);
26        button2.addActionListener(ctrl);

27        frame.getContentPane().setLayout(new GridLayout(1, 2));
28        frame.getContentPane().add(button1);
29        frame.getContentPane().add(button2);

30        frame.pack();
31        frame.setLocationRelativeTo(null);
32        frame.setResizable(false);

33        frame.setVisible(true);
34    }
35 }
```

## Aufgabe 3 - Listen

**12 Punkte**

In dieser Aufgabe sollen Sie mit der vordefinierten Datenstruktur `LinkedList`<sup>1</sup> aus dem Package `java.util` arbeiten.

Implementieren Sie die folgenden Aufgaben in einer Klasse `LinkedListUtil`. Beachten Sie den folgenden Code für die Beispiele:

```
1 | LinkedList<Integer> list = new LinkedList<>();
2 | list.add(1);
3 | list.add(2);
4 | list.add(3);
```

a) Implementieren Sie eine Methode

```
public static <T> void print(LinkedList<T> list)
```

welche die in der Liste gespeicherten Werte in eine Zeile ausgibt (in  $\mathcal{O}(n)$ ). Im Anschluss soll die Zeile umgebrochen werden.

**Beispiel:** `print(list);` → 1 2 3

— Lösung —

```
1 | public static <T> void print(LinkedList<T> list) {
2 |     for (T val : list) {
3 |         System.out.print(val + " ");
4 |     }
5 |     System.out.println();
6 | }
```

b) Implementieren Sie eine Methode

```
public static <T> void printReversed(LinkedList<T> list)
```

welche die in der Liste gespeicherten Werte in umgekehrter Reihenfolge in eine Zeile ausgibt (in  $\mathcal{O}(n)$ ). Im Anschluss soll die Zeile umgebrochen werden.

**Beispiel:** `printReversed(list);` → 3 2 1

— Lösung —

```
1 | public static <T> void printReversed(LinkedList<T> list) {
2 |     Iterator<T> iterator = list.descendingIterator();
3 |     while (iterator.hasNext()) {
4 |         System.out.print(iterator.next() + " ");
5 |     }
6 |     System.out.println();
7 | }
```

<sup>1</sup><https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html>

c) Implementieren Sie eine Methode

```
public static <T> void removeDuplicates(LinkedList<T> list)
```

welche alle in der Liste `list` enthaltenen Duplikate entfernt (in  $\mathcal{O}(n^2)$ ).

**Beispiel:**

```
1 list.addFirst(1);
2 list.add(3);
3 list.add(1);
4 print(list); //-> 1 1 2 3 3 1
5 removeDuplicates(list);
6 print(list); //-> 1 2 3
```

---

### Lösung

Lösung (in  $\mathcal{O}(n^2)$ )

```
1 public static <T> void removeDuplicates(LinkedList<T> list) {
2     LinkedList<T> list2 = new LinkedList<>();
3
4     for (T val : list) {
5         if (!list2.contains(val)) {
6             list2.add(val);
7         }
8     }
9
10    list.clear();
11    list.addAll(list2);
12 }
```

## Aufgabe 4 - Bäume

**12 Punkte**

Im Ordner ~/material finden Sie eine Datei materialTree03.zip, welche eine rudimentäre Implementierung eines Binärbaumes beinhaltet.

Beachten Sie den folgenden Code für die Beispiele:

```
1 | IntTree tree = new IntTree();  
2 | tree.add(5).add(3).add(3).add(1).add(7).add(9);  
3 | System.out.println(tree); //-> 1 3 3 5 7 9
```

- a) Importieren Sie das Material.
- b) Implementieren Sie eine Methode `public int maxRec()`, welche rekursiv nach dem größten im Baum gespeicherten Wert sucht. Falls der Baum keine Werte enthält, soll eine `IllegalStateException` geworfen werden.  
**Hinweis:** Sie dürfen eine Hilfsmethode benutzen.  
**Beispiel:** `tree.maxRec()` → 9
- c) Implementieren Sie eine Methode `public int maxIt()`, welche iterativ nach dem größten im Baum gespeicherten Wert sucht. Falls der Baum keine Werte enthält, soll eine `IllegalStateException` geworfen werden.  
**Beispiel:** `tree.maxIt()` → 9
- d) Implementieren Sie eine Methode `public int countOcc(int val)`, welche zählt wie oft der Wert `val` im Baum vorkommt und diese Häufigkeit zurückgibt.  
**Beispiel:** `tree.countOcc(3)` → 2



## — Lösung —

```
1  //IntTree
2  public int maxRec() { //b)
3      if (root == null) {
4          throw new IllegalStateException();
5      }
6      return maxRec(root);
7  }
8
9  private int maxRec(IntNode node) {
10     if (node.right != null) {
11         return maxRec(node.right);
12     }
13     return node.value;
14 }
15
16 public int maxIt() { //c)
17     if (root == null) {
18         throw new IllegalStateException();
19     }
20
21     IntNode temp = root;
22     while (temp.right != null) {
23         temp = temp.right;
24     }
25
26     return temp.value;
27 }
28
29 public int countOcc(int val) { //d)
30     return countOcc(val, root);
31 }
32
33 private int countOcc(int val, IntNode node) {
34     if (node == null) {
35         return 0;
36     }
37
38     int count = 0;
39     if (node.value == val) {
40         count++;
41     }
42
43     return count + countOcc(val, node.left)
44         + countOcc(val, node.right);
45 }
```