Ludwig-Maximilians-Universität München
Institut für Informatik
Prof. Dr. R. Hennicker, P. Wendler

WS 2017/18
Übungsblatt 11

Übungen zu Einführung in die Informatik: Programmierung und Software-Entwicklung: Lösungsvorschlag

Aufgabe 11-1 Rekursion und Terminierung *Präsenz*

Gegeben ist folgende rekursive Methode zur Berechnung einer Funktion:

```
1 public static int f(int x) {
2   if (x == 0) {
3     return 0;
4   } else if (x > 0) {
5     return f(x - 2);
6   } else {
7     return f(x + 2);
8   }
9 }
```

a) Welcher Wert wird von `f` für die folgenden Funktionsaufrufe berechnet?
`f(2), f(8), f(9), f(-1), f(-8), f(-9)`

Lösung:

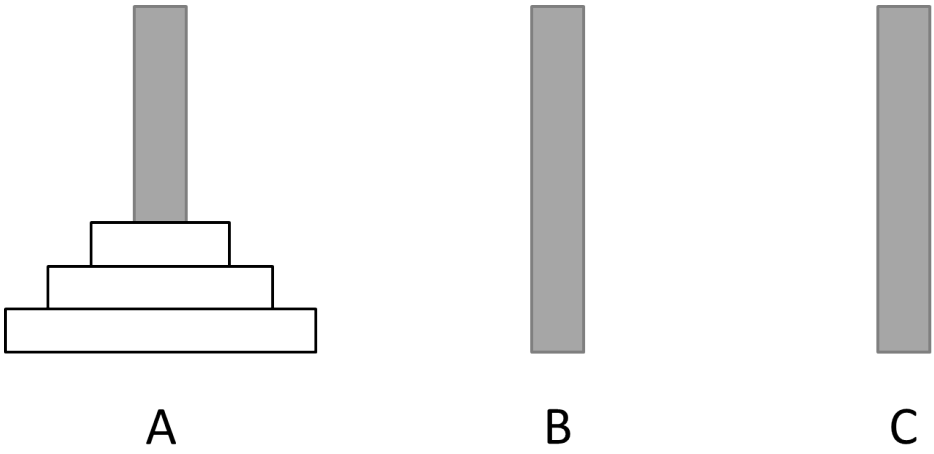
- `f(2) = f(0) = 0` (ein rekursiver Aufruf)
- `f(8) = f(6) = f(4) = f(2) = f(0) = 0` (vier rekursive Aufrufe)
- `f(9) = f(7) = f(5) = f(3) = f(1) = f(-1) = f(1) = f(-1) = ...` usw.
  terminiert nicht, unendlich viele rekursive Aufrufe, Funktionswert undefiniert.
- `f(-1) = f(1) = f(-1) = ...` usw.
  terminiert nicht, unendlich viele rekursive Aufrufe, Funktionswert undefiniert.
- `f(-8) = f(-6) = f(-4) = f(-2) = f(0) = 0` (vier rekursive Aufrufe)
- `f(-9) = f(-7) = f(-5) = f(-3) = f(-1) = f(1) = f(-1) = ...` usw.
  terminiert nicht, unendlich viele rekursive Aufrufe, Funktionswert undefiniert.

b) Charakterisieren Sie die Menge aller Integerzahlen `x`, für die der Funktionsaufruf `f(x)` terminiert.

Lösung:
`f(x)` terminiert genau dann, wenn `x` gerade ist.

Aufgabe 11-2 Türme von Hanoi *Präsenz*

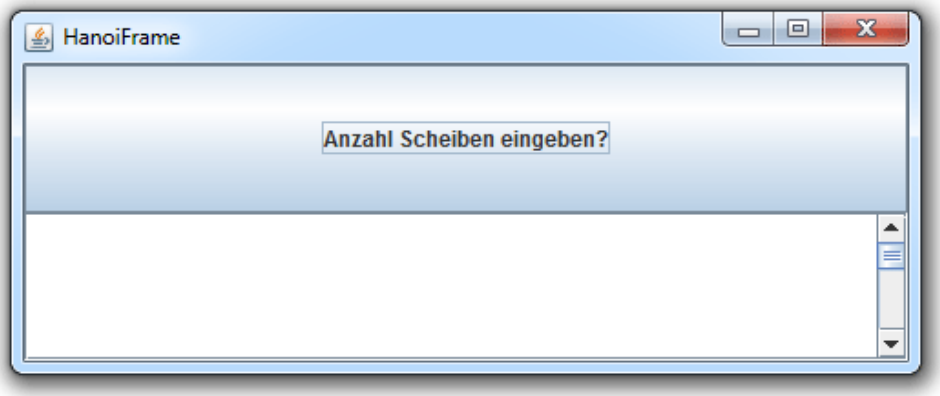
Bei dem Spiel "Türme von Hanoi" sind drei Säulen A, B und C gegeben. Auf Säule A sind `n` Scheiben unterschiedlicher Größe so aufgesteckt, dass jeweils eine kleinere Scheibe auf einer größeren zu liegen kommt.



Die Aufgabe des Spiels besteht darin, die `n` Scheiben von A nach C zu transportieren, wobei B als Zwischenlager benutzt werden darf. Spielregel ist, dass nur einzelne Scheiben bewegt werden dürfen und dass nach jedem Einzelschritt niemals eine größere Scheibe auf einer kleineren zu liegen kommt.

In dieser Aufgabe sollen Sie ein Programm mit einer grafischen Benutzeroberfläche implementieren, welches für eine beliebige Zahl `n` eine Lösung des Problems für `n` Scheiben ausgibt. Die Anzahl der Scheiben `n` soll vom Benutzer frei wählbar sein und wie gewohnt durch einen modalen Dialog zu Beginn der Anwendung abgefragt werden.

a) Die grafische Benutzeroberfläche soll wie folgt aussehen:



Es soll einen Button mit der oben angegebenen Aufschrift geben. Darunter soll der Ausgabebereich für die Protokollierung der Scheibenbewegungen platziert werden. Da die Protokollierung aller Schritte relativ lang sein kann, können Sie mit Hilfe der Klassen `ScrollPane` und `ScrollPaneConstants` der Swing-Bibliothek dem Ausgabebereich auf folgende Weise eine vertikale Scrollbar hinzufügen:

```
1    JScrollPane scrollPane = new JScrollPane(ausgabeBereich);
2    scrollPane.setHorizontalScrollBarPolicy(
3        ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
```

Schreiben Sie eine Klasse `HanoiFrame`, die die Hauptklasse dieser grafischen Benutzeroberfläche sein soll und das Fenster erzeugt. Um Ihr Programm ausführen zu können, schreiben Sie eine weitere Klasse `HanoiFrameMain`, die Sie wie gewohnt im gleichen Ordner wie Ihre Klasse `HanoiFrame` abspeichern.

Lösung:
Bei der Implementierung der grafischen Benutzeroberfläche geht man wie folgt vor:

Deklarieren Sie eine Klasse `HanoiFrame`, die die Hauptklasse Ihrer grafischen Benutzeroberfläche wird und deshalb von der Klasse `JFrame` erbt. Ihre Klasse `HanoiFrame` soll zwei Attribute haben:

- ein Attribut vom Klassentyp `JButton` für den Button,
- ein Attribut vom Klassentyp `JTextArea`, das als Ausgabebereich für die spätere Rückmeldung über das Ergebnis dient.

Schreiben Sie einen Konstruktor, der den `HanoiFrame` mit einem entsprechenden Titel und Größe (wir wählen hier 500x200 Pixel) initialisiert. In dem Konstruktor sollen weiterhin alle Attribute korrekt initialisiert werden. Das Layout des `ContentPane` des `HanoiFrame`s wird auf ein `GridLayout` mit zwei Zeilen und einer Spalte initialisiert und der Button sowie der Ausgabebereich darauf platziert. Fügen Sie abschließend noch ein, dass das Programm ordnungsgemäß beendet wird, falls der `HanoiFrame` geschlossen wird. Benutzen Sie dazu die Methode `setDefaultCloseOperation`.

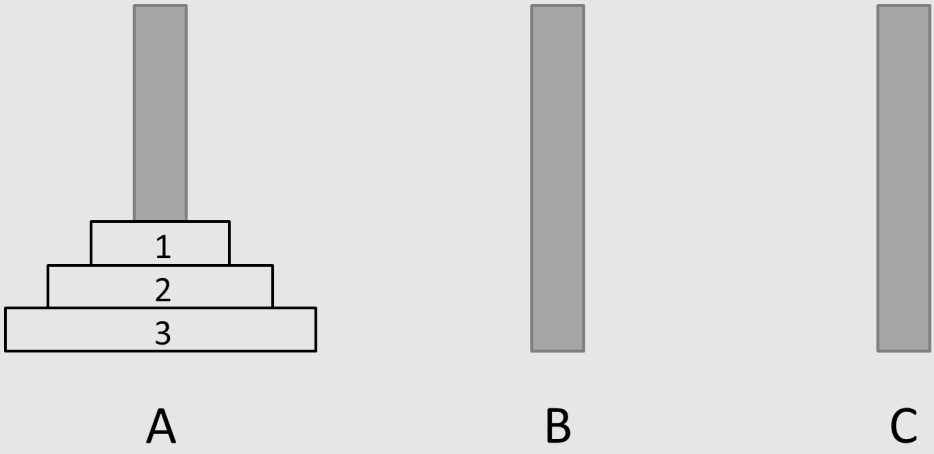
Weiter unten finden Sie eine Implementierung der Klassen `HanoiFrame` und `HanoiFrameMain` für die gesamte Aufgabe.

b) Erweitern Sie Ihre Klasse `HanoiFrame` um eine Ereignisbehandlung für den Button. Wird dieser Button gedrückt, soll der Benutzer zunächst mit Hilfe der Klasse `JOptionPane` nach der Anzahl `n` der zu bewegenden Scheiben gefragt werden. Anschließend sollen die korrekten Scheibenbewegungen zur Lösung des Problems der Türme von Hanoi protokolliert werden. Verwenden Sie dazu eine rekursive Methode mit dem Kopf `public void hanoi(int scheiben, char quelle, char mitte, char ziel)`, die beim Aufruf von `hanoi(n, 'A', 'B', 'C')` die einzelnen Scheibenbewegungen der Reihe nach durch Ausgaben der Form "Scheibe von x nach y" mit $x, y \in \{A, B, C\}$ im Ausgabebereich protokolliert.

Das folgende Protokoll zeigt eine Ausgabe des gewünschten Programms für drei Scheiben:

```
Scheibe von A nach C
Scheibe von A nach B
Scheibe von C nach B
Scheibe von A nach C
Scheibe von B nach A
Scheibe von B nach C
Scheibe von A nach C
```

Lösung: Algorithmus für das Problem der Türme von Hanoi:
Für drei Scheiben würde die Lösung wie folgt aussehen:



`Scheibe von A nach C` oder $\texttt{A} \xrightarrow{1} \texttt{C}$
`Scheibe von A nach B` oder $\texttt{A} \xrightarrow{2} \texttt{B}$
`Scheibe von C nach B` oder $\texttt{C} \xrightarrow{1} \texttt{B}$

`Scheibe von A nach C` oder $\texttt{A} \xrightarrow{3} \texttt{C}$

`Scheibe von B nach A` oder $\texttt{B} \xrightarrow{1} \texttt{A}$
`Scheibe von B nach C` oder $\texttt{B} \xrightarrow{2} \texttt{C}$
`Scheibe von A nach C` oder $\texttt{A} \xrightarrow{1} \texttt{C}$

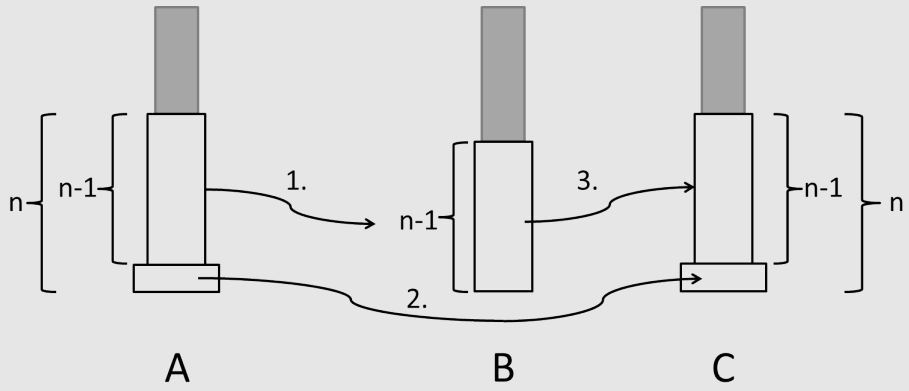
Für vier Scheiben würde die Lösung wie folgt aussehen, falls eine Lösung für drei Scheiben bekannt ist:

A → B		B → C
A → C		B → A
B → C		C → A
A → B	A → C	B → C
C → A		A → B
C → B		A → C
A → B		B → C

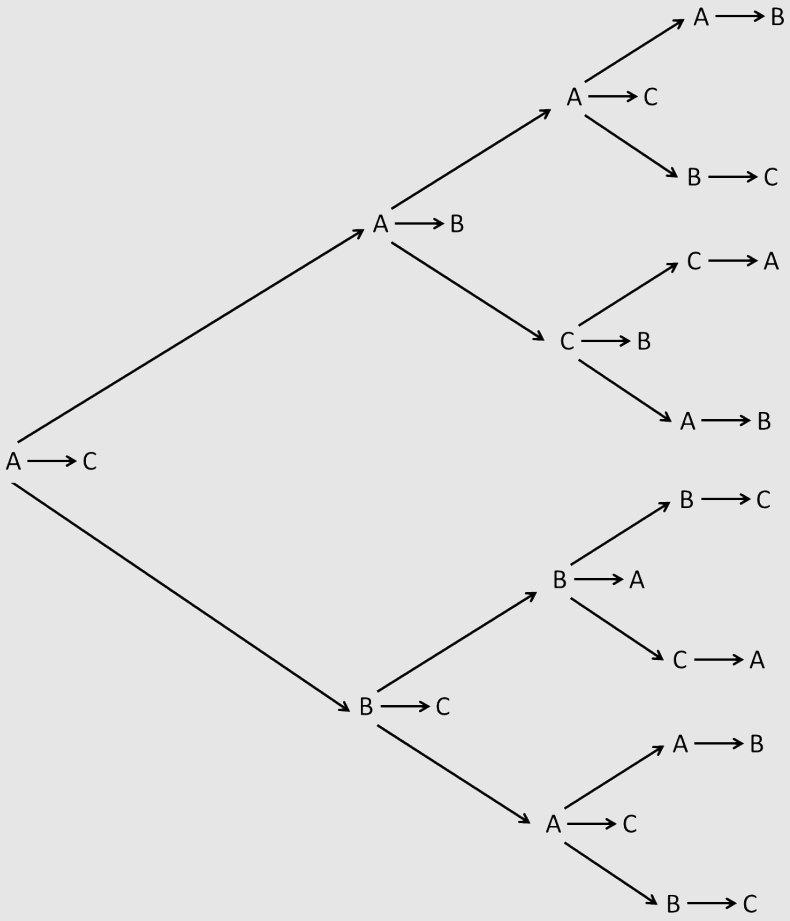
Daraus folgt die allgemeine Lösung des Problems für `n` Scheiben, falls eine Lösung für `n-1` Scheiben bekannt ist:

Versetze `n-1` Scheiben von A nach B
Versetze die `n`-te Scheibe von A nach C
Versetze `n-1` Scheiben von B nach C

oder grafisch:



Der Algorithmus lässt sich auch als baumrekursive Struktur darstellen, wobei der Wurzelknoten links ist (so lässt sich die Abfolge der Schritte von oben nach unten lesen wie in der Ausgabe):



Implementierung des baumrekursiven Algorithmus mithilfe der Methode `hanoi`:

```
1 public void hanoi(int scheiben, char quelle, char ablage, char ziel) {
2     if (scheiben <= 0) {
3         return;
4     } else {
5         hanoi(scheiben - 1, quelle, ziel, ablage);
6         String schritt = "Scheibe von " + quelle + " nach " + ziel + "\n";
7         this.ausgabeBereich.append(schritt);
8         hanoi(scheiben - 1, ablage, quelle, ziel);
9     }
```

```
}
```

Eine Implementierung der Klassen `HanoiFrame` und `HanoiFrameMain` finden Sie auch im ZIP-Achiv.

```
import java.awt.Container;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.ScrollPaneConstants;

/**
 * Diese Klasse repraesentiert das Hauptfenster der Tuerme von
 * Hanoi-Praesenzaufgabe.
 *
 * @author Annabelle Klarl
 *
 */
public class HanoiFrame extends JFrame implements ActionListener {
    private JButton scheibenbewegungenButton;

    private JTextArea ausgabeBereich;

    /**
     * In diesem Programmstueck wird das Fenster erzeugt
     */
    public HanoiFrame() {
        /* In der Kopfleiste des Fenster steht "HanoiFrame" */
        this.setTitle("HanoiFrame");

        /* Das Fenster ist 500 Pixel breit und 200 Pixel hoch. */
        this.setSize(500, 200);

        /* Hier werden alle Buttons erzeugt. */
        this.scheibenbewegungenButton = new JButton(
                "Anzahl Scheiben eingeben?");

        /* Hier wird der Ausgabe-Bereich erzeugt. */
        this.ausgabeBereich = new JTextArea(60, 100);
        JScrollPane scrollPane = new JScrollPane(this.ausgabeBereich);
        scrollPane.setHorizontalScrollBarPolicy(
            ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);

        /*
         * Der ContentPane ist der Ausschnitt des Fensters, auf dem Widgets d.h.
         * Interaktionselemente (wie eine TextArea oder ein Button) platziert
         * werden koennen.
         */
        Container contentPane = this.getContentPane();
        contentPane.setLayout(new GridLayout(2, 1));
        /* Hier wird der Button platziert. */
        contentPane.add(this.scheibenbewegungenButton);
        /* Hier wird der Ausgabebereich platziert. */
        contentPane.add(scrollPane);

        /*
         * Hier wird der Frame als Listener fuer Knopfdruck-Ereignisse bei
         * jedem der Buttons registriert.
         */
```

```
61          this.scheibenbewegungenButton.addActionListener(this);
62
63          /*
64           * Wird das Fenster geschlossen (d.h. auf X gedrueckt), wird mit Hilfe
65           * dieser Programmzeile auch unser Programm ordnungsgemaess beendet.
66           */
67          this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
68      }
69
70      @Override
71      public void actionPerformed(ActionEvent e) {
72          // je nach Button entsprechende Methode ausfuehren
73          Object source = e.getSource();
74          if (source == this.scheibenbewegungenButton) {
75              this.protokolliereScheibenbewegungen();
76          }
77      }
78
79      /**
80       * implementiert die Funktionalitaet des
81       * "Protokoll der Scheibenbewegungen"-Buttons
82       */
83      private void protokolliereScheibenbewegungen() {
84          String einlesenAnzahlScheiben = JOptionPane
85                  .showInputDialog("Wieviele Scheiben: ");
86          int anzahlScheiben = Integer.parseInt(einlesenAnzahlScheiben);
87
88          this.ausgabeBereich.setText("");
89          this.hanoi(anzahlScheiben, 'A', 'B', 'C');
90      }
91
92      /**
93       * protokolliert rekursiv die Scheibenbewegungen
94       */
95      public void hanoi(int scheiben, char quelle, char ablage, char ziel) {
96          if (scheiben <= 0) {
97              return;
98          } else {
99              hanoi(scheiben - 1, quelle, ziel, ablage);
100             String schritt = "Scheibe von " + quelle + " nach " + ziel + "\n";
101             this.ausgabeBereich.append(schritt);
102             hanoi(scheiben - 1, ablage, quelle, ziel);
103         }
104     }
105 }
```

```
1  /**
2   * Diese Klasse startet den HanoiFrame
3   *
4   * @author Annabelle Klarl
5   */
6  public class HanoiFrameMain {
7
8      /**
9       * Dieses Programmstueck startet das Programm.
10      */
11     public static void main(String[] args) {
12         HanoiFrame hanoiFrame = new HanoiFrame();
13         hanoiFrame.setVisible(true);
14     }
15
16 }
```

Aufgabe 11-3 **Türme von Hanoi** *Hausaufgabe*

In dieser Aufgabe soll die Präsenzaufgabe 11-2 soll erweitert werden, dass auch die Nummer der aktuellen Scheibenbewegung mit ausgegeben wird. Modifizieren Sie das in Aufgabe 11-2 erstellte Programm so, dass die einzelnen Scheibenbewegungen durch Ausgaben der Form "i-ter Schritt: Scheibe von x nach y" durchnummeriert werden. Benennen Sie die abzugebenden Klassen mit `HanoiFrameH` und `HanoiFrameHMain`.

Das folgende Protokoll zeigt eine Ausgabe des gewünschten Programms für 3 Scheiben:

```
1-ter Schritt: Scheibe von A nach C
2-ter Schritt: Scheibe von A nach B
3-ter Schritt: Scheibe von C nach B
4-ter Schritt: Scheibe von A nach C
5-ter Schritt: Scheibe von B nach A
6-ter Schritt: Scheibe von B nach C
7-ter Schritt: Scheibe von A nach C
```

Hinweis: Verwenden Sie ein Klassenattribut `iterSchritt` vom Typ `int`. Das Attribut ist entsprechend der Auswertungsreihenfolge der rekursiven Aufrufe an geeigneter Stelle in der Methode `hanoi` zu inkrementieren.

Lösung:

1. Deklaration eines Attributs `iterSchritt`:

```
1 public class HanoiFrameH extends JFrame implements ActionListener {
2   private int iterSchritt = 0;
3   //... Implementierung der Klasse HanoiFrameH
4 }
```

2. Erweiterung der Methode `hanoi`:

```
1 private void hanoi(int scheiben, char quelle, char ablage, char ziel) {
2   if (scheiben > 0) {
3     hanoi(scheiben - 1, quelle, ziel, ablage);
4     iterSchritt = iterSchritt + 1;
5     String schritt = iterSchritt
6         + "-ter Schritt: Scheibe von " + quelle
7         + " nach " + ziel + "\n";
8     this.ausgabeBereich.append(schritt);
9     hanoi(scheiben - 1, ablage, quelle, ziel);
10  }
11 }
```

Eine Implementierung der Klassen `HanoiFrameH` und `HanoiFrameHMain` finden Sie auch im ZIP-Archiv.

```
1  import java.awt.Container;
2  import java.awt.GridLayout;
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5
6  import javax.swing.JButton;
7  import javax.swing.JFrame;
8  import javax.swing.JOptionPane;
9  import javax.swing.JScrollPane;
10 import javax.swing.JTextArea;
11 import javax.swing.ScrollPaneConstants;
12
13 /**
14  * Diese Klasse repraesentiert das Hauptfenster der Tuerme von
```

```
15  * Hanoi-Hausaufgabe.
16  *
17  * @author Annabelle
18  *
19  */
20 public class HanoiFrameH extends JFrame implements ActionListener {
21     private JButton scheibenbewegungenButton;
22
23     private JTextArea ausgabeBereich;
24
25     private int iterSchritt = 0;
26
27     /**
28      * In diesem Programmstueck wird das Fenster erzeugt
29      */
30     public HanoiFrameH() {
31         /* In der Kopfleiste des Fenster steht "HanoiFrame" */
32         this.setTitle("HanoiFrame");
33
34         /* Das Fenster ist 500 Pixel breit und 200 Pixel hoch. */
35         this.setSize(500, 200);
36
37         /* Hier werden alle Buttons erzeugt. */
38         this.scheibenbewegungenButton = new JButton(
39                 "Anzahl Scheiben eingeben?");
40
41         /* Hier wird der Ausgabe-Bereich erzeugt. */
42         this.ausgabeBereich = new JTextArea(60, 100);
43         JScrollPane scrollPane = new JScrollPane(this.ausgabeBereich);
44         scrollPane.setHorizontalScrollBarPolicy(
45             ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
46
47         /*
48          * Der ContentPane ist der Ausschnitt des Fensters, auf dem Widgets d.h.
49          * Interaktionselemente (wie eine TextArea oder ein Button) platziert
50          * werden koennen.
51          */
52         Container contentPane = this.getContentPane();
53         contentPane.setLayout(new GridLayout(2, 1));
54         /* Hier wird der Button platziert. */
55         contentPane.add(this.scheibenbewegungenButton);
56         /* Hier wird der Ausgabebereich platziert. */
57         contentPane.add(scrollPane);
58
59         /*
60          * Hier wird der Frame als Listener fuer Knopfdruck-Ereignisse bei
61          * jedem der Buttons registriert.
62          */
63         this.scheibenbewegungenButton.addActionListener(this);
64
65         /*
66          * Wird das Fenster geschlossen (d.h. auf X gedrueckt), wird mit Hilfe
67          * dieser Programmzeile auch unser Programm ordnungsgemaess beendet.
68          */
69         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
70     }
71
72     @Override
73     public void actionPerformed(ActionEvent e) {
74         // je nach Button entsprechende Methode ausfuehren
75         Object source = e.getSource();
76         if (source == this.scheibenbewegungenButton) {
77             this.protokolliereScheibenbewegungen();
78         }
79     }
```

```
80
81      /**
82       * implementiert die Funktionalitaet des
83       * "Protokoll der Scheibenbewegungen"-Buttons
84       */
85      private void protokolliereScheibenbewegungen() {
86          String einlesenAnzahlScheiben = JOptionPane
87                  .showInputDialog("Wieviele Scheiben: ");
88          int anzahlScheiben = Integer.parseInt(einlesenAnzahlScheiben);
89
90          iterSchritt = 0;
91          this.ausgabeBereich.setText("");
92          this.hanoi(anzahlScheiben, 'A', 'B', 'C');
93      }
94
95      /**
96       * protokolliert rekursiv die Scheibenbewegungen
97       */
98      private void hanoi(int scheiben, char quelle, char ablage,
99              char ziel) {
100         if (scheiben <= 0) {
101             return;
102         } else {
103             this.hanoi(scheiben - 1, quelle, ziel, ablage);
104             iterSchritt = iterSchritt + 1;
105             String schritt = iterSchritt
106                     + "-ter Schritt: Scheibe von " + quelle
107                     + " nach " + ziel + "\n";
108             this.ausgabeBereich.append(schritt);
109             this.hanoi(scheiben - 1, ablage, quelle, ziel);
110         }
111     }
112 }
```

```
1  /**
2   * Diese Klasse startet den HanoiFrame
3   *
4   * @author Annabelle Klarl
5   */
6  public class HanoiFrameHMain {
7
8      /**
9       * Dieses Programmstueck startet das Programm.
10      */
11     public static void main(String[] args) {
12         HanoiFrameH hanoiFrame = new HanoiFrameH();
13         hanoiFrame.setVisible(true);
14     }
15
16 }
```

Aufgabe 11-4 **Rekursion und Iteration** *Hausaufgabe*

Wir betrachten die Funktion $\log_2$, die für jede Integerzahl $n \geq 1$ den ganzzahligen Logarithmus zur Basis 2 berechnet, d.h. das Ergebnis ist vom Typ `int`. Die Funktion $\log_2$ ist folgendermaßen spezifiziert: Für alle $n \geq 1$ gilt:

$$2^{\log_2(n)} \leq n < 2^{\log_2(n)+1}$$

Beispielsweise ist:

$\log_2(1) = 0$,
$\log_2(2) = \log_2(3) = 1$,
$\log_2(4) = ... = \log_2(7) = 2$,
$\log_2(8) = ... = \log_2(15) = 3$.

Es sollen zwei Methoden, die jeweils die Funktion $\log_2$ implementieren, geschrieben werden:

a) eine nicht-rekursive Methode mit Kopf `public static int logiterativ(int n)`

Lösung:
Induktion:
Falls $n = 1$ gilt $\log_2(1) = 0$ und $\log_2(n) = 1 + \log_2(n/2)$ sonst.

Iteratives Java-Programm:

```
1  private int logiterativ(int n) {
2      if (n <= 0) {
3          return -1;
4      }
5      int akk = 0;
6      while (n > 1) {
7          akk = 1 + akk;
8          n = n / 2;
9      }
10     return akk;
11 }
```

b) eine rekursive Methode mit Kopf `public static double logrek(int n)`

Lösung:

Rekursives Java-Programm:

```
1  private int logrek(int n) {
2      if (n <= 0) {
3          return -1;
4      } else if (n == 1) {
5          return 0;
6      } else {
7          return 1 + logrek(n / 2);
8      }
9  }
```

c) Schreiben Sie wie üblich eine Umgebung mit Benutzerschnittstelle, in der Sie beide Methoden testen können. Verwenden Sie dabei zwei Buttons, einen zum Testen der iterativen Methode und einen zum Testen der rekursiven Methode.

Lösung:

Eine Implementierung der Klassen `LogarithmusFrame` und `LogarithmusFrameMain` finden Sie auch im ZIP-Archiv.

```
1  import java.awt.Container;
2  import java.awt.GridLayout;
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5
6  import javax.swing.JButton;
```

```
7  import javax.swing.JFrame;
8  import javax.swing.JOptionPane;
9  import javax.swing.JTextArea;
10
11 /**
12  * Diese Klasse repraesentiert das Hauptfenster der Logarithmus-Praesenzaufgabe.
13  *
14  * @author Annabelle Klarl
15  *
16  */
17 public class LogarithmusFrame extends JFrame implements ActionListener {
18     private JButton iterativButton;
19     private JButton rekursivButton;
20
21     private JTextArea ausgabeBereich;
22
23     /**
24      * In diesem Programmstueck wird das Fenster erzeugt
25      */
26     public LogarithmusFrame() {
27         /* In der Kopfleiste des Fenster steht "LogarithmusFrame" */
28         this.setTitle("LogarithmusFrame");
29
30         /* Das Fenster ist 300 Pixel breit und 200 Pixel hoch. */
31         this.setSize(300, 200);
32
33         /* Hier werden alle Buttons erzeugt. */
34         this.iterativButton = new JButton("iterative log-Berechnung");
35         this.rekursivButton = new JButton("rekursive log-Berechnung");
36
37         /* Hier wird der Ausgabe-Bereich erzeugt. */
38         this.ausgabeBereich = new JTextArea(60, 100);
39         /*
40          * Der ContentPane ist der Ausschnitt des Fensters, auf dem Widgets d.h.
41          * Interaktionselemente (wie eine TextArea oder ein Button) platziert
42          * werden koennen.
43          */
44         Container contentPane = this.getContentPane();
45         contentPane.setLayout(new GridLayout(3, 1));
46         /* Hier werden die Buttons platziert. */
47         contentPane.add(this.iterativButton);
48         contentPane.add(this.rekursivButton);
49         /* Hier wird der Ausgabebereich platziert. */
50         contentPane.add(ausgabeBereich);
51
52         /*
53          * Hier wird der Frame als Listener fuer Knopfdruck-Ereignisse bei
54          * jedem der Buttons registriert.
55          */
56         this.iterativButton.addActionListener(this);
57         this.rekursivButton.addActionListener(this);
58
59         /*
60          * Wird das Fenster geschlossen (d.h. auf X gedrueckt), wird mit Hilfe
61          * dieser Programmzeile auch unser Programm ordnungsgemaess beendet.
62          */
63         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
64     }
65
66     @Override
67     public void actionPerformed(ActionEvent e) {
68         // je nach Button entsprechende Methode ausfuehren
69         Object source = e.getSource();
70         if (source == this.iterativButton) {
71             this.berechneLogIterativ();
```

```
72          } else if (source == this.rekursivButton) {
73              this.berechneLogRekursiv();
74          }
75      }
76
77      /**
78       * implementiert die Funktionalitaet des Buttons zur iterative Berechnung
79       * des Logarithmus
80       */
81      private void berechneLogIterativ() {
82          String einlesenN = JOptionPane.showInputDialog("Zahl n: ");
83          int n = Integer.parseInt(einlesenN);
84
85          int log = logiterativ(n);
86
87          this.ausgabeBereich.setText("Der Logarithmus von n ist " + log);
88      }
89
90      /**
91       * implementiert die Funktionalitaet des Buttons zur iterative Berechnung
92       * des Logarithmus
93       */
94      private void berechneLogRekursiv() {
95          String einlesenN = JOptionPane.showInputDialog("Zahl n: ");
96          int n = Integer.parseInt(einlesenN);
97
98          int log = logrek(n);
99
100         this.ausgabeBereich.setText("Der Logarithmus von n ist " + log);
101     }
102
103     /**
104      * berechnet den Logarithmus von n auf iterative Weise
105      */
106     private int logiterativ(int n) {
107         if (n <= 0) {
108             return -1;
109         }
110         int akk = 0;
111         while (n > 1) {
112             akk = 1 + akk;
113             n = n / 2;
114         }
115         return akk;
116     }
117
118     /**
119      * berechnet den Logarithmus von n auf rekursive Weise
120      */
121     private int logrek(int n) {
122         if (n <= 0) {
123             return -1;
124         } else if (n == 1) {
125             return 0;
126         } else {
127             return 1 + logrek(n / 2);
128         }
129     }
130 }
```

```
1 /**
2  * Diese Klasse startet den LogarithmusFrame
3  *
4  * @author Annabelle Klarl
5  */
```

```
6  public class LogarithmusFrameMain {
7
8      /**
9       * Dieses Programmstueck startet das Programm.
10      *
11      * @param args
12      */
13     public static void main(String[] args) {
14         LogarithmusFrame logarithmusFrame = new LogarithmusFrame();
15         logarithmusFrame.setVisible(true);
16     }
17
18 }
```

Besprechung der Präsenzaufgaben in den Übungen vom 19.01.2018 und 22.01.2018. Abgabe der Hausaufgaben bis Mittwoch, 31.01.2018, 14:00 Uhr über UniworX (siehe Folien der ersten Zentralübung). Erstellen Sie zu jeder Aufgabe Klassen, die die Namen tragen, die in der Aufgabe gefordert sind. Geben Sie nur die entsprechenden `.java`*-Dateien ab. Wir benötigen* ***nicht*** *Ihre* `.class`*-Dateien.*