Ludwig-Maximilians-Universität München
Institut für Informatik
Prof. Dr. R. Hennicker, Dr. A. Klarl

WS 2016/17
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 }
```

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

Lösung:

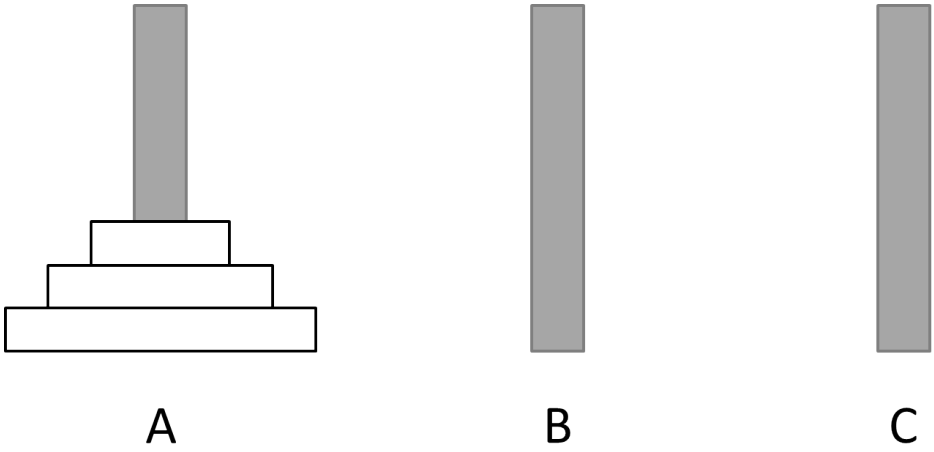
- `f(2) = f(0) = 0` (ein rekursiver Aufruf)
- `f(8) = f(6) = f(4) = f(2) = f(0) = 0` (vier rekursive Aufrufe)
- `f(9) = f(7) = f(5) = f(3) = f(1) = f(-1) = f(1) = f(-1) = ...` usw.
  terminiert nicht, unendlich viele rekursive Aufrufe, Funktionswert undefiniert.
- `f(-1) = f(1) = f(-1) = ...` usw.
 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

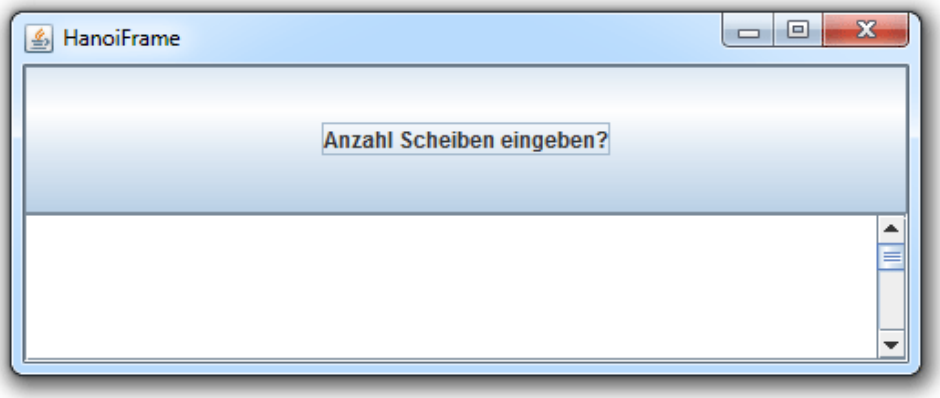
Bei dem Spiel "Türme von Hanoi" sind drei Säulen A, B und C gegeben. Auf Säule A sind `n` Scheiben unterschiedlicher Größe so aufgesteckt, dass jeweils eine kleinere Scheibe auf einer größeren zu liegen kommt.



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`.

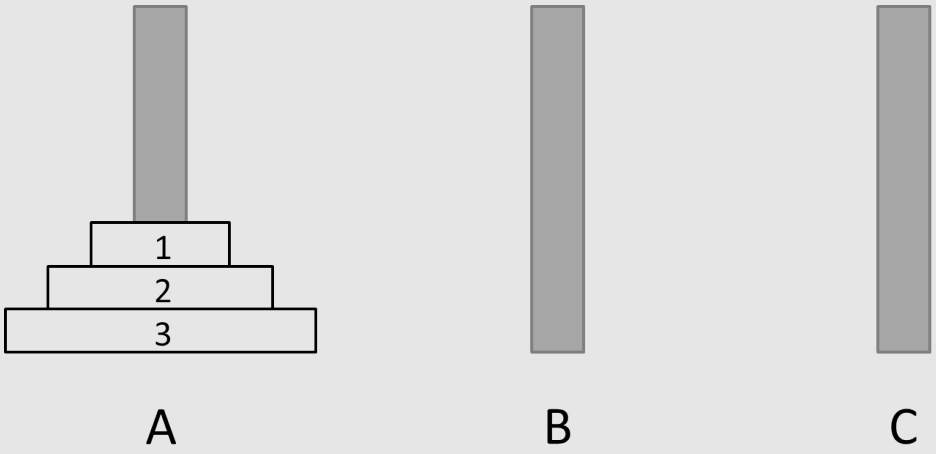
Auf der Webseite finden Sie die Implementierung der Klassen `HanoiFrame` und `HanoiFrameMain`.

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 $\texttt{A} \xrightarrow{1} \texttt{C}$

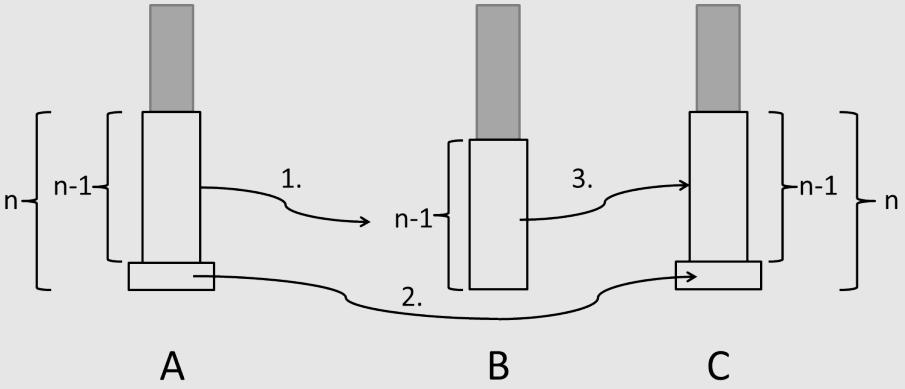
Für vier Scheiben würde die Lösung wie folgt aussehen, falls eine Lösung für drei Scheiben bekannt ist:

$\texttt{A} \to \texttt{B}$		$\texttt{B} \to \texttt{C}$
$\texttt{A} \to \texttt{C}$		$\texttt{B} \to \texttt{A}$
$\texttt{B} \to \texttt{C}$		$\texttt{C} \to \texttt{A}$
$\texttt{A} \to \texttt{B}$	$\texttt{A} \to \texttt{C}$	$\texttt{B} \to \texttt{C}$
$\texttt{C} \to \texttt{A}$		$\texttt{A} \to \texttt{B}$
$\texttt{C} \to \texttt{B}$		$\texttt{A} \to \texttt{C}$
$\texttt{A} \to \texttt{B}$		$\texttt{B} \to \texttt{C}$

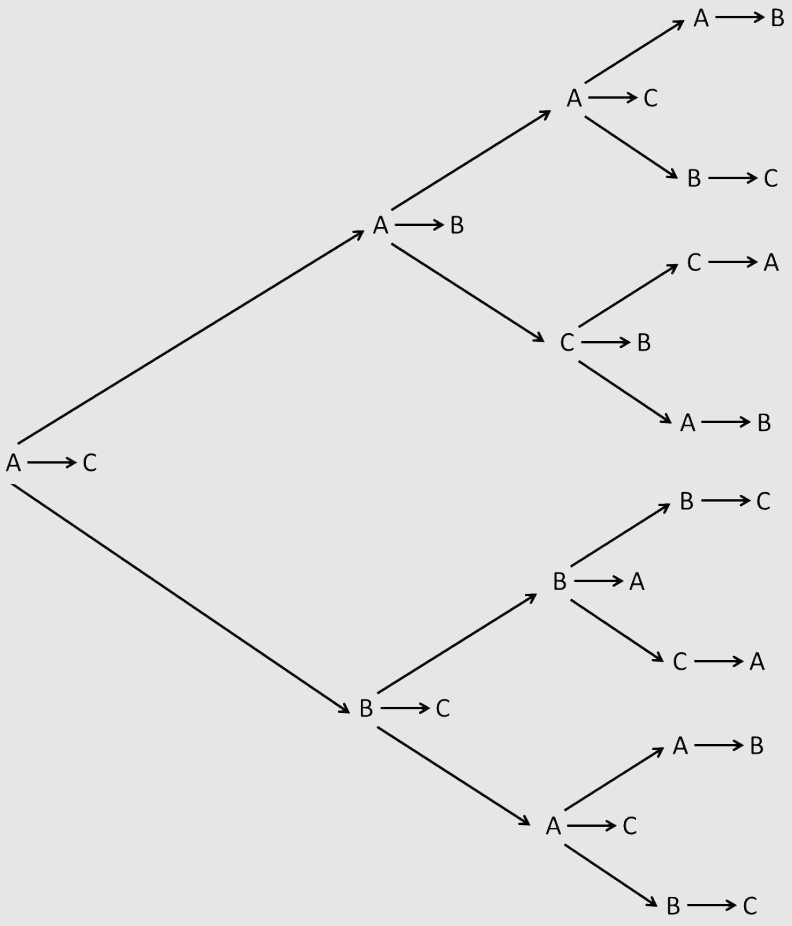
Daraus folgt die allgemeine Lösung des Problems für `n` Scheiben, falls eine Lösung für `n-1` Scheiben bekannt ist:

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    }
5    else {
6      hanoi(scheiben - 1, quelle, ziel, ablage);
7      String schritt = "Scheibe von " + quelle + " nach " + ziel + "\n";
8      this.ausgabeBereich.append(schritt);
9      hanoi(scheiben - 1, ablage, quelle, ziel);
10   }
11 }
```

Auf der Webseite finden Sie die Implementierung der Klassen `HanoiFrame` und `HanoiFrameMain`.

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.

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)`

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

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.

*Besprechung der Präsenzaufgaben in den Übungen vom 11.01.2017 bis zum 18.01.2017. Abgabe der Hausaufgaben bis Mittwoch, 25.01.2017, 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.*