

## Inhaltsverzeichnis

1. GUI/Desktop Programmierung.....	2
1.1. Ziele.....	2
1.2. Aufgaben: Swing/GUI-Grundlagen.....	2
1.2.1. Aufgabe: SwingHallo.....	3
1.2.2. Aufgabe: SwingLucky7.....	3
1.2.3. Aufgabe: SwingOnMouseOver.....	4
1.2.4. Hinweise: Ereignis: MouseEntered.....	4
1.2.5. Aufgabe: SwingPasswort.....	5
1.2.6. Hinweis: Ereignis JTextField: KeyTyped().....	5
1.2.7. Hinweis: Timer.....	5
1.2.8. Aufgabe: SwingBildlauf.....	6
1.2.9. Hinweis: initTime() Timer erzeugen.....	7
1.2.10. Hinweis: Ereignis JSlider: stateChanged ().....	7
1.3. Aufgaben: Swing/GUI-Funktionen.....	8
1.3.1. Aufgabe: SwingMyAlgo.....	8
1.3.2. Hinweis: jar-files erzeugen und aus anderem Projekt darauf zugreifen.....	8
1.3.3. Aufgabe: SwingMorse (HU).....	9
1.3.4. Hinweis: Einen Ton erzeugen.....	9
1.3.5. Aufgabe: SwingCaesar.....	11
1.3.6. Aufgabe: SwingLotto.....	11
1.3.7. Hinweis: Label undurchsichtig/nichttransparent ( setOpaque(true)).....	12
1.3.8. Aufgabe: SwingSliderSort.....	12
1.3.9. Aufgabe: SwingZahlensysteme (HU).....	13
1.3.10. Aufgabe: SwingBMI.....	13
1.3.11. Aufgabe: SwingTanken (HU).....	14
1.3.12. Aufgabe: SwingAuth (HU).....	14
1.3.13. Aufgabe: SwingGeburtstag (HU).....	15
1.4. Aufgaben: Swing/GUI: OOP.....	15
1.4.1. Aufgabe: SwingAutorennen.....	15
1.4.2. Aufgabe: SwingBlackjack.....	15
1.4.3. Aufgabe: SwingPingPong.....	16
1.4.4. Aufgabe: SwingBildlaufSpiel.....	16
1.4.5. Hinweise: Grafik: / paint().....	16
1.4.6. Aufgabe: SwingDatumZeit.....	17
1.4.7. Hinweis: Datum, Zeit.....	17
1.4.8. +Hinweis: AnalogUhr.....	18
1.4.9. Aufgabe: SwingFAQ.....	18
1.4.10. Aufgabe: SwingSHS.....	18
1.5. Aufgabe: SwingEditor.....	18
1.5.1. Hinweise: FileDialog.....	19
1.5.2. Hinweise: MP3.....	20
1.5.3. Hinweis: Directory.....	23
1.5.4. Hinweis: SwingEditor-FAQ.....	23
1.5.5. Hinweis: MVC.....	28
1.6. Aufgabe: SwingPizza.....	31
1.6.1. Die Datei pizza.ini und die Klasse Pizza.java.....	32
1.6.2. UI-Elemente initialisieren.....	32
1.6.3. Auf UI-events registrieren.....	33
1.6.4. Button: Beleg erstellen.....	33
1.6.5. Hinweis: PDF erstellen.....	33
1.6.6. Hinweis: EXCEL.....	34
1.6.7. Hinweis: XML:.....	35

<a href="#">1.7. Aufgabe: SwingAmpel (HU).....</a>	<a href="#">35</a>
<a href="#">1.7.1. Hinweis: Threads.....</a>	<a href="#">36</a>
<a href="#">1.7.2. Hinweis: Ereignisprozeduren direkt aufrufen.....</a>	<a href="#">38</a>
<a href="#">1.8. Aufgabe: SwingGameOfLife.....</a>	<a href="#">39</a>
<a href="#">1.8.1. See: <a href="https://bitstorm.org/gameoflife/">https://bitstorm.org/gameoflife/</a>.....</a>	<a href="#">39</a>
<a href="#">1.8.2. UI.....</a>	<a href="#">39</a>
<a href="#">1.8.3. The Rules.....</a>	<a href="#">39</a>
<a href="#">1.8.4. Hinweis: mehrere Button mit gleicher Ereignisprozedur.....</a>	<a href="#">40</a>
<a href="#">1.9. Aufgabe: SwingHangman.....</a>	<a href="#">40</a>
<a href="#">1.9.1. Hinweis; Klasse Hangman.....</a>	<a href="#">41</a>
<a href="#">1.10. Zusammenfassung: Ein Beispiel.....</a>	<a href="#">42</a>

## 1. GUI/Desktop Programmierung

---

### 1.1. Ziele

---

☒ Desktop Programme erstellen können

☒ Steuerelemente/Dialogelemente, Menüs erstellen können

Quelle: Guido Krüger: Handbuch der Java Programmierung, [www.javabuch.de](http://www.javabuch.de)

Empfehlenswert, insbesondere zum Nachschlagen: <http://www.java2s.com/>

Einfach toll: <http://java.sun.com/docs/books/tutorial/ui/features/components.html>

### 1.2. Aufgaben: Swing/GUI-Grundlagen

---

In diesem Kapitel werden einfache GUI-Programme erstellt.

Verschiedene Dialogelemente und das Reagieren/Programmieren auf verschiedene Ereignisse steht im Mittelpunkt.

### 1.2.1. Aufgabe: SwingHallo

**Hinweise:**

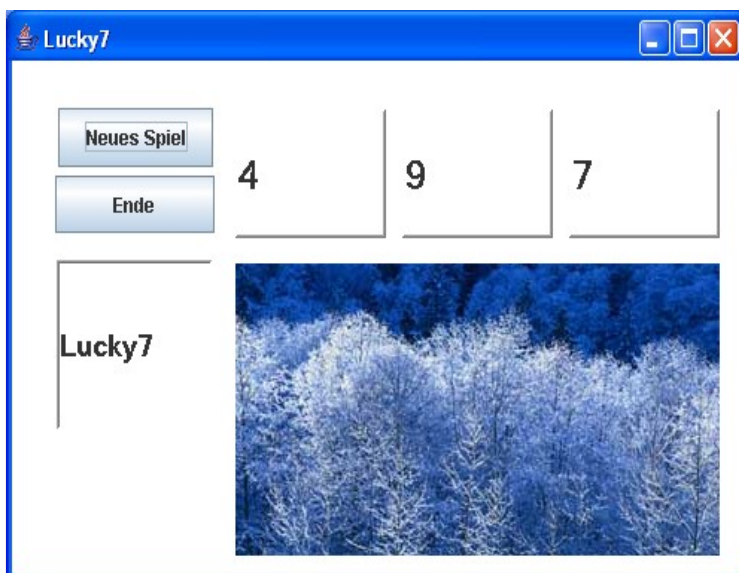
Arbeiten mit eclipse und windowBuilder (Google)  
Visual Class zum Projekt hinzufügen

Dialogelemente: JButton, JLabel

Ereignisse: actionPerformed()

```
jButtonHallo.setEnabled(false)
```

### 1.2.2. Aufgabe: SwingLucky7



```
//Zufallszahlen bestimmen
java.util.Random zufall= new java.util.Random();

int z1,z2,z3;
z1= zufall.nextInt(10);

jLabel1.setText(String.valueOf(z1));

//Image Eigenschaft setzen
bei aktiviertem jLabelBild
Eigenschaft: icon

//Images zur Laufzeit des Programmes laden:
//Ereignis: windowOpened
javax.swing.ImageIcon icon = new javax.swing.ImageIcon("Winter.jpg");
jLabelBild = new JLabel(icon);
...
jLabelBild.setVisible(true);
```

### 1.2.3. Aufgabe: SwingOnMouseOver

Der Button soll, bevor er vom Benutzer aktiviert werden kann, sich an eine zufällig gewählte Stelle bewegen.



### 1.2.4. Hinweise: Ereignis: MouseEntered

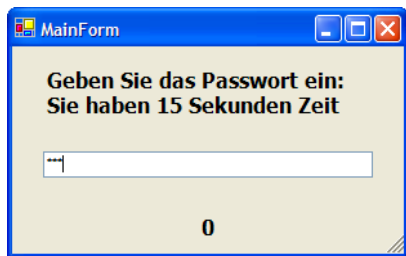
```
jButtonClick.getWidth()
jButtonClick.getHeight()

jContentPane.getWidth()

jButtonClick.setLocation(x,y)
    x ... X-Achse
    y ... Y-Achse
```

Ursprung (0,0) ist links oben

### 1.2.5. Aufgabe: SwingPasswort



Hinweise:

### 1.2.6. Hinweis: Ereignis JTextField: KeyTyped()

```
....  
public void keyTyped(java.awt.event.KeyEvent e) {  
    ....  
    char[] pass;  
    if (e.getKeyChar() == java.awt.event.KeyEvent.VK_ENTER){  
        pass= jPasswordField.getPassword();  
  
        strPass= String.valueOf(pass);  
  
        if (strPass.equals("claudia")){  
  
        ....  
    }
```

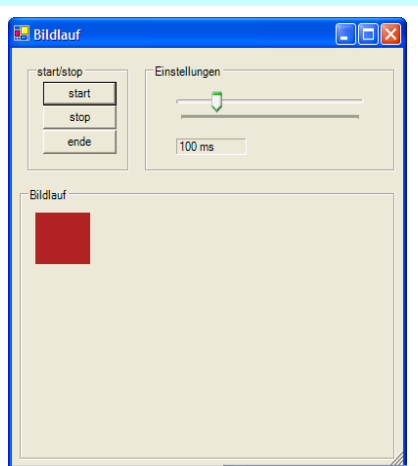
### 1.2.7. Hinweis: Timer

```
public void startTimer()  
{  
    int delay = 1000;           // Jede Sekunde ein Tick  
    javax.swing.Timer t;       // TIMER definieren  
  
    // Befehle bei einem Tick definieren  
    t= new javax.swing.Timer (delay,      new java.awt.event.ActionListener())  
    {  
        public void actionPerformed(java.awt.event.ActionEvent e)  
        {  
            // wert aus dem Label lesen  
            int value= Integer.parseInt(jLabelTime.getText());  
  
            // wert erniedrigen  
        }  
    }  
}
```

```
        value--;  
        // wert wieder ins label schreiben  
        jLabelTime.setText(String.valueOf(value));  
  
        // wenn 0 erreicht -> Meldung an User und Ende  
        if (value == 0)  
        {  
            javax.swing.JOptionPane.showMessageDialog(null,  
                "Ende",  
                "Ende",  
                javax.swing.JOptionPane.INFORMATION_MESSAGE);  
  
            System.exit(0);  
        } //if  
    } // actionPerformed  
});  
  
    // Timer nun starten  
    t.start();  
  
} //startTimer()
```

### 1.2.8. Aufgabe: SwingBildlauf

Ein Bild soll sich innerhalb eines Containers bewegen. Ein Slider dient zum Einstellen der Geschwindigkeit. Die Geschwindigkeit wird durch einen Timer realisiert.



Hinweise:

// Globale Objektreferenzen u. Variablen:

int delayTimer;

javax.swing.Timer theTimer;

**Ereignisse:**

Bei open Window wird  
die Methode `initTimer()` aufgerufen. (s.u. Hinweis: `initTime()` Timer erzeugen)

Bei Button start wird  
Wert vom Slider gelesen. (`int wert= jSlider.getValue()`)  
das Timerdelay gesetzt  
der Timer gestartet. (`theTimer.start()`)

Bei Button stop wird  
der Timer gestoppt

Bei Slider state changed wird (s. u. Hinweis: Ereignis `JSlider: stateChanged()`)  
Wert vom Slider gelesen  
Wert in den Label geschrieben  
Timerdelay gesetzt

### 1.2.9. Hinweis: `initTime()` Timer erzeugen

Diese Funktion wird bei Ereignis `windowOpened()` aufgerufen, um ein globales Timerobjekt namens `theTimer` zu erzeugen.

Achtung: der Timer selbst wird nicht gestartet, sondern es wird nur das Timerobjekt erzeugt.

```
public void initTimer(){
    theTimer= new javax.swing.Timer(
        timerDelay,
        new java.awt.event.ActionListener(){
            public void actionPerformed(java.awt.event.ActionEvent e){
                java.util.Random zufall;
                int x,y;

                // neues x, y bestimmen
                zufall= new java.util.Random();
                x= zufall.nextInt(jPanelBildContainer.getWidth());
                y= zufall.nextInt(jPanelBildContainer.getHeight());

                ...

                // setzen auf neue Position
                jPanelBild.setLocation(x, y);
            }
        });
}
```

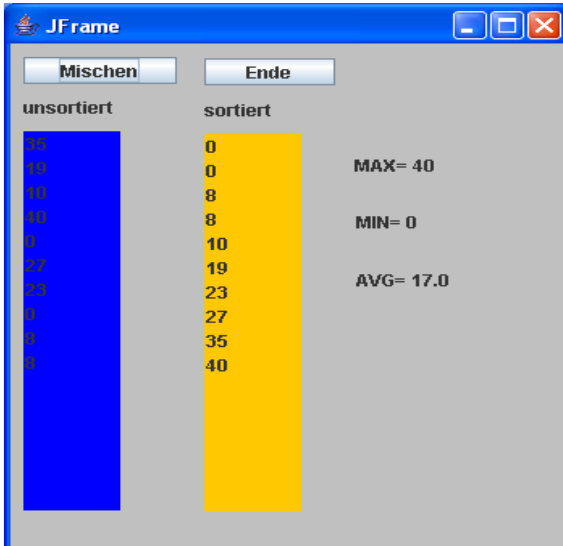
### 1.2.10. Hinweis: Ereignis JSlider: stateChanged ()

```
...  
timerDelay= jSliderGeschwindigkeit.getValue();  
  
// Das Timer-Delay verändern  
theTimer.setDelay(timerDelay);  
  
if (jLabelGeschwindigkeit!= null)  
    jLabelGeschwindigkeit.setText(String.valueOf(timerDelay)+" ms");  
...
```

## 1.3. Aufgaben: Swing/GUI-Funktionen

In diesem Kapitel werden GUI-Programme erstellt, die Funktionen einer anderen Klasse aufrufen. Ein erster Schritt um die Trennung von Oberfläche und den eigentlichen Algorithmen zu erreichen.

### 1.3.1. Aufgabe: SwingMyAlgo



#### Hinweise:

2 Klassen:

FKTMyAlgo.jar (Projekt: FKTMyAlgo)

SwingMyAlgo.java (Projekt: SwingMyAlgo)

### 1.3.2. Hinweis: jar-files erzeugen und aus anderem Projekt darauf zugreifen



### 1. Projekt FKMyAlgo: erzeuge FKMyAlgo.jar aus FKMyAlgo.java

---

Vorbedingung:

packagename: zb.: org.ht

Projekt FKMyAlgo markieren, re. Maustaste, export, jar file,  
(li/re oben) NUR src/org.ht/FKMyAlgo.java wählen  
finish

### 2. FKMyAlgo.jar in anderem Projekt (zB.SwingEditor) verfügbar machen

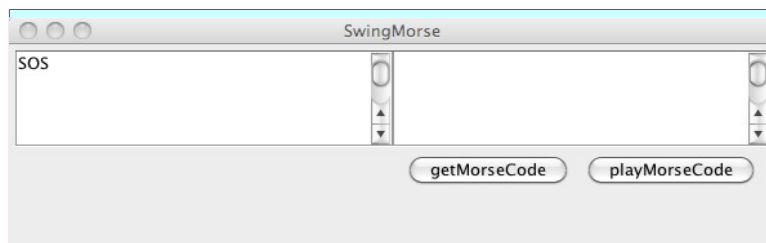
---

Projekt: SwingMyAlgo

Projekteigenschaften: BUILD-Path, Add Jars,  
Projekt: FKMyAlgo, File FKMyAlgo.jar

fertig!

#### 1.3.3. Aufgabe: SwingMorse (HU)



##### Hinweise:

verwendet Funktionen aus FKMyAlgo

```
public static String getMorse(String s){...}
```

```
public static void playMorse(String s) throws LineUnavailableException{
```

```
    for(int i=0; i < s.length();i++)
```

```
        if (s.charAt(i)=='-')
```

```
            // ton(A) Dauer(1sec) lautstaerke          harmonisch  
            generateTone(440, 800, 100, true);
```

```
        else if (s.charAt(i)=='.')
```

```
            // ton(A) Dauer(1sec) lautstaerke          harmonisch  
            generateTone(440, 400, 100, true);
```

```
        else
```

```
            generateTone(440, 400, 0, true);
```

```
    }
```

### 1.3.4. Hinweis: Einen Ton erzeugen

```
//http://www.schulphysik.de/java/physlet/applets/sinus1.html

/** Generates a tone.
 * @param hz Base frequency (neglecting harmonic) of the tone in cycles per second
 * @param msec The number of milliseconds to play the tone.
 * @param volume Volume, form 0 (mute) to 100 (max).
 * @param addHarmonic Whether to add an harmonic, one octave up. */

public static void generateTone(int hz, int msec, int volume,
                                boolean addHarmonic)
    throws javax.sound.sampled.LineUnavailableException {

    float frequency = 44100;
    byte[] buf;

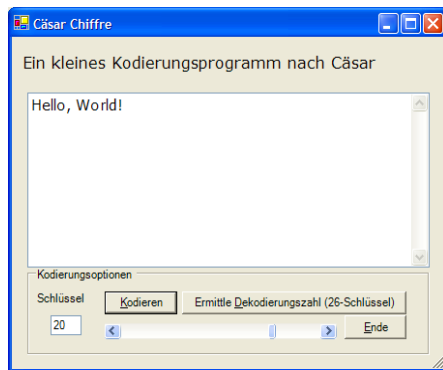
    javax.sound.sampled.AudioFormat af;
    if (addHarmonic) {
        buf = new byte[2];
        af = new javax.sound.sampled.AudioFormat(frequency, 8, 2, true, false);
    } else {
        buf = new byte[1];
        af = new javax.sound.sampled.AudioFormat(frequency, 8, 1, true, false);
    }

    javax.sound.sampled.SourceDataLine sdl;
    sdl = javax.sound.sampled.AudioSystem.getSourceDataLine(af);
    //sdl = AudioSystem.getSourceDataLine(af);
    sdl.open(af);
    sdl.start();
    for(int i=0; i<msec*frequency/1000; i++){
        double angle = i/(frequency/hz)*2.0*Math.PI;
        buf[0]=(byte)(Math.sin(angle)*volume);

        if(addHarmonic) {
            double angle2 = (i)/(frequency/hz)*2.0*Math.PI;
            buf[1]=(byte)(Math.sin(2*angle2)*volume*0.6);
            sdl.write(buf, 0, 2);
        } else {
            sdl.write(buf, 0, 1);
        }
    }
    sdl.drain();
    sdl.stop();
    sdl.close();
}

/* public static void main(String[] args) throws LineUnavailableException {
    //          ton(A)  Dauer(1sec)  lautstaerke          harmonisch
    FKtMyAlgo.generateTone(440, 800, 100, true);
    FKtMyAlgo.generateTone(440, 400, 100, true);
    FKtMyAlgo.generateTone(440, 400, 100, true);
}
*/
```

### 1.3.5. Aufgabe: SwingCaesar



Einen Text nach der Caesar-Methode verschlüsseln.

#### Hinweise: Kodieren

```
String s= JTextPane.getText();
int key= jSlider.getValue();

s= FKTMMyAlgo.caesar(s, key);

JTextPane.setText(s);
```

#### Hinweise zu Strings und char-Arrays:

```
...
String str;
....
char[] b= str.toCharArray();
int len= str.length;

char ch= str.charAt(0);

if (str.charAt(i) >= 'A' && str.charAt(i) <= 'Z'){ // Großbuchstabe?
    ...
}

b[i] += key; // oder b[i]= b[i] + key;

String str2= String.valueOf(b);
```

### 1.3.6. Aufgabe: SwingLotto

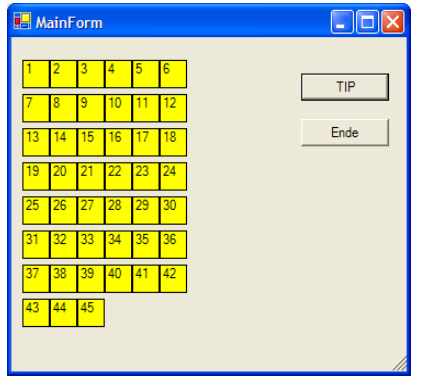
todo:

lotto: FKTMysAlgo.getZiehung()

lotto: FKTMysAlgo.getTipp()

lotto: FKTMysAlgo.getGewinn()

Labels soll per Programm, also dynamisch erzeugt werden.



**Hinweise:**

```
private JLabel[] jLabelZahlen;
```

Ereignis: windowOpened

Labels erzeugen und Eigenschaften setzen

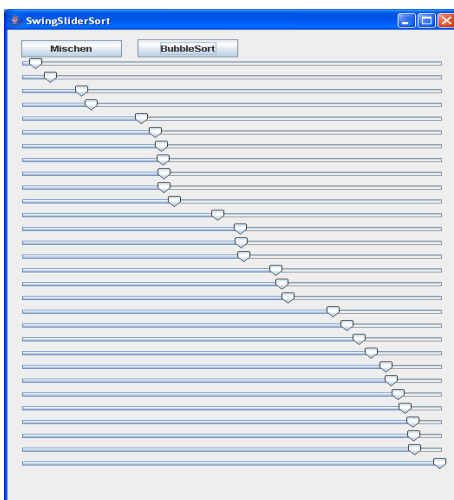
### 1.3.7. Hinweis: Label undurchsichtig/nichttransparent ( setOpaque(true))

// Label undurchsichtig schalten,d.h. Nichttransparent

// dann kann die Hintergrundfarbe gesetzt und angezeigt werden `jLabelZahlen[i].setOpaque(true);`

### 1.3.8. Aufgabe: SwingSliderSort

Jslider zur Laufzeit erzeugen und mittels Button sortieren bzw. mischen.



**Hinweise:**

Arrays, JSlider, Algorithmus, Sortieren, Pseudocode, PAP , (ProgrammAblaufPlan)

PAP: sortieren

fertig=false

while false==fertig

    fertig=true

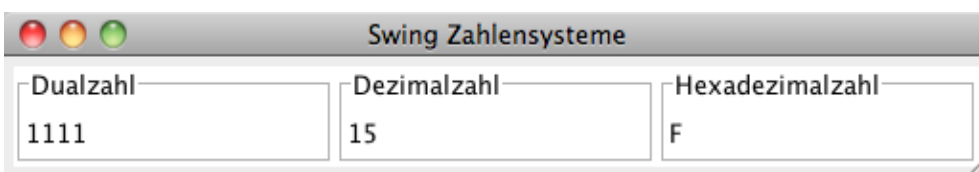
    Für alle Arrayelemente

        wenn aktuelles Element > als nächstes Element

            tausche

    fertig=false

### 1.3.9. Aufgabe: SwingZahlensysteme (HU)

**Aufgabe:**

Wenn man in eines der 3 Textfelder Ziffern eingibt, werden in den anderen Feldern die Inhalte automatisch angepasst. Im obigen Beispiel wurden im Textfeld Dualzahl vier 1 eingegeben.

Um wiederverwendbare Programmteile zu haben, wollen wir in der Klasse FKTMylgo folgende Funktionen schreiben:

☒ **TODO:** Programmiere in FKTMylgo

```
public static int dual2int (String ziffern)
public static int dezimal2int(String ziffern)
public static int hex2int(String ziffern)
```

```
public static String int2dual(int zahl)
public static String int2hex(int zahl)
public static String int2dezimal(int zahl)
```

☒ Für jedes Textfeld erstellen Sie die Ereignismethode

```
textFieldDual.addKeyListener(new KeyAdapter() {
    @Override
    public void keyReleased(KeyEvent arg0) {
        // nur 0 und 1 zulassen
        ...
    }
})
```

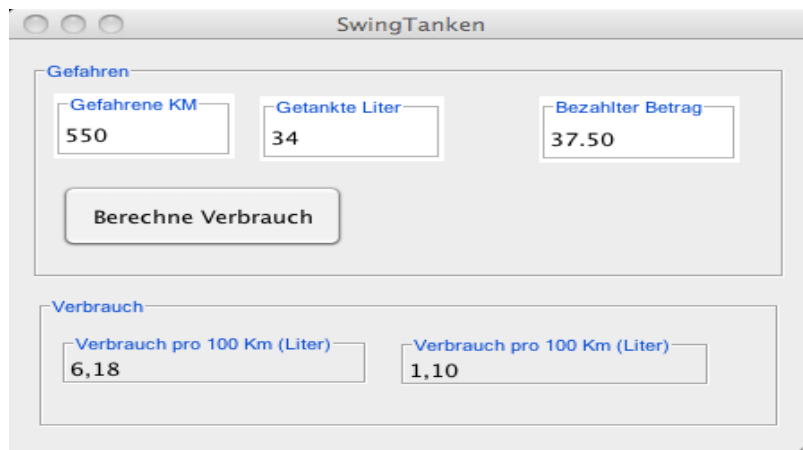
☒ Als Vorlage verwende die Datei:

/JAVA/04-java-gui1/02-ueben/SwingZahlensystemeWB\_UE.zip

### 1.3.10. Aufgabe: SwingBMI

Todo

### 1.3.11. Aufgabe: SwingTanken (HU)



### 1.3.12. Aufgabe: SwingAuth (HU)

Schreiben Sie für folg. Aufgabenstellung eine Swing-Anwendung zur Demonstration/Übung der  
Hier nun die Aufgabenstellung

gegeben sind:

```
int[] zahlen={12,11,10,9,8,7,6,5,4,3,2,1};
```

```
String[] fragen={  
    "Zahl4 * Zahl2 + Zahl1 = ?",  
    "Zahl3 * Zahl2 - Zahl8 = ?",  
    "Zahl7 * Zahl2 + Zahl9 = ?",  
    "Zahl1 * Zahl4 + Zahl3 = ?",  
    "Zahl3 * Zahl2 + Zahl6 = ?",  
    "Zahl7 * Zahl5 - Zahl8 = ?",  
    "Zahl4 * Zahl10 - Zahl12 = ?",  
    "Zahl3 * Zahl11 - Zahl3 = ?",  
    "Zahl8 * Zahl7 + Zahl6 = ?",  
    "Zahl5 * Zahl2 - Zahl9 = ?",  
    "Zahl2 * Zahl3 - Zahl1 = ?",  
    "Zahl7 * Zahl8 + Zahl2 = ?"  
};
```

Erstellen Sie die Funktion (in FKTMMyAlgo)

```
public static int getAntwort(int[] zahlen, String frage);  
    @param zahlen Array mit Zahlenwerten (s.o.),  
    @param frage eine einzelne Frage (Aufbau einer Frage s.o.)  
    @return die berechnete Zahl
```

Erstellen Sie ein Testprogramm SwingAuth, das folg. Elemente/Funktionalität hat.

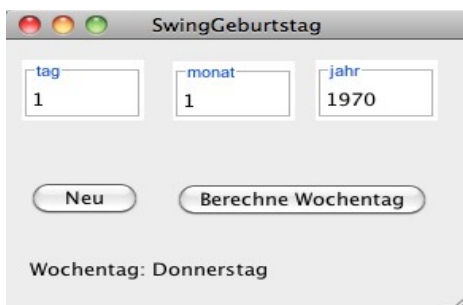
Eingaben:

- ☒ Der Benutzer soll die Werte für das Array zahlen eingeben/ändern können.
- ☒ Der Benutzer soll eine Frage aus einer Combo-Box auswählen können.
- ☒ Der Benutzer soll in ein Textfeld die seiner Meinung nach richtige Antwort eingeben können
- ☒ Der Benutzer soll durch einen Button-Click den Computer die richtige Antwort ermitteln lassen und
- ☒ der Computer soll angeben wieviele Fragen der Benutzer richtig und wieviele Fragen der Benutzer falsch angegeben hat.

Verwenden Sie die obige Funktion: getAntwort().

### 1.3.13. Aufgabe: SwingGeburtstag (HU)

---



Verwenden Sie die Funktionen aus FKTMylgo:

```
public static String berechneWochentag(int tag, int monat, int jahr){....}
```

## 1.4. Aufgaben: Swing/GUI: OOP

---

In diesem Kapitel werden GUI-Programme erstellt, die mit Objekten, Streams, etc. kombiniert werden.

### 1.4.1. Aufgabe: SwingAutorennen

---

Siehe: java-oop1.odt

### 1.4.2. Aufgabe: SwingBlackjack

---

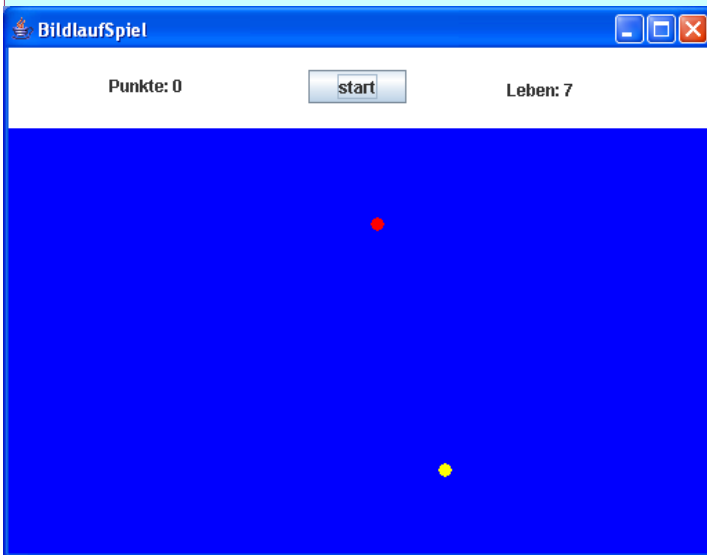
Siehe: java-oop1.odt

### 1.4.3. Aufgabe: SwingPingPong

Siehe: java-grafik.odt

### 1.4.4. Aufgabe: SwingBildlaufSpiel

Zwei Bälle bewegen sich innerhalb eines Containers. Wenn ein Ball den Container verlässt, kostet dies einen Lebenspunkt. Wenn der Benutzer einen Ball anklickt, wird dieser innerhalb des Containers neu positioniert (Zufallszahlen). Auf diese Weise kann der Spieler verhindern, dass der Ball den Container verlässt.



### 1.4.5. Hinweise: Grafik: / paint()

```
Graphics g = jPanel.getGraphics();

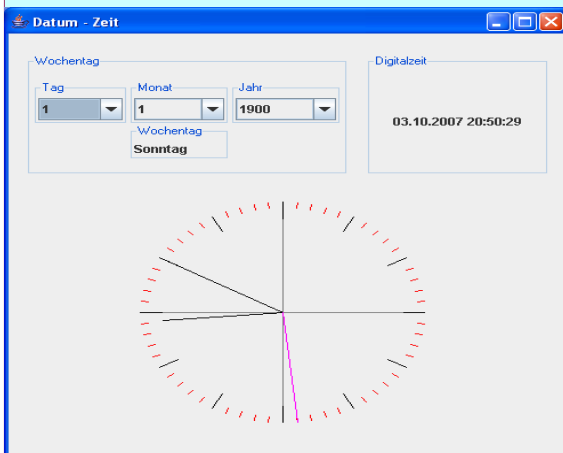
g.setColor(Color.yellow);
g.fillOval(beginBall1X, beginBall1Y, 10, 10);

g.setColor(Color.red);
g.fillOval(beginBall2X, beginBall2Y, 10, 10);
```



### 1.4.6. Aufgabe: SwingDatumZeit

Verschiedene Datumsfunktionen und Graphikfunktionen auf einen Blick. Wochentagsberechnung, Aktuelle Uhrzeit.



Siehe auch das nächste Kapitel: Java-Grafik

### 1.4.7. Hinweis: Datum, Zeit

```
java.text.SimpleDateFormat df= new java.text.SimpleDateFormat("dd.MM.yyyy HH:mm:ss");  
jLabelDigitalZeit.setText(df.format(new java.util.Date()));
```

**Hinweis:** Formel zur Berechnung des Wochentages

$$w=[a+\text{int}((a/4) - \text{int}(a/100) + \text{int}(a/400) + d) \bmod 7;$$

a jahreszahl des vorjahres

d anzahl der tage im gefragten jahr (inkl. gewuenschter tag)

w= 0 so

w= 1 mo

w= 2 di

w= 3 mi

w= 4 do

w= 5 fr

w= 6 sa

Beispiel:

Auf welchen Wochentag fiel der 1.03.1984

a=1983

d= 31+29+1=61 (1984 ist ein Schaltjahr)

**Hinweise zum Schaltjahr:**

Jahr ist ein Schaltjahr, wenn

1. (Jahr durch 4 teilbar UND

2. Jahr nicht durch 100 teilbar)

3. ODER aber Jahr durch 400 teilbar

**Beispiel:**

1900 kein Schaltjahr aber 2000 ein Schaltjahr

#### 1.4.8. +Hinweis: AnalogUhr

```
/* Mittelpunkt (h,k):
   x = h + r*cos(t)
   y = k + r*sin(t)
   */

// Stundenzeiger
grf.setColor (Color.black);
grf.drawLine(M.x, M.y,
             M.x + (int)((width/2 * 0.85)*Math.cos(Math.toRadians(30*hrs - 90 + min/2))),
             M.y + (int)((height/2 * 0.85)*Math.sin(Math.toRadians(30*hrs - 90 + min/2)))
            );

// Minutenzeiger
grf.setColor (Color.black);
grf.drawLine(M.x, M.y,
             M.x + (int)((width/2)*Math.cos(Math.toRadians(6*min - 90))),
             M.y + (int)((height/2)*Math.sin(Math.toRadians(6*min - 90)))
            );
```

#### 1.4.9. Aufgabe: SwingFAQ

todo

#### 1.4.10. Aufgabe: SwingSHS

todo

### 1.5. Aufgabe: SwingEditor

```
Menü:
Datei
    öffnen
    öffnen mit spamliste
    (m)öffnen bmi
    (m)öffnen f_noten
    speichern
    speichern unter
    exit

+MP3
    öffnen mp3
    play
    play again
    play directory
```

pause  
stop

#### Programme

start vlc  
start SwingLucky7  
start SwingVisualSort  
start SwingCaesar

#### (m)Filter

buffer filtern mit spamdatei  
buffer filtern mit spamliste

#### (m)Statistik

gesamtstatistik

#### Verschlüsseln

caesar  
caesar knacken  
+skytale  
+skytale knacken  
Substitution  
XORSubstitution  
AdditiveSubstitution  
StreamCipher

#### Komprimieren

RunLength  
Huffman

#### FAQ

faq datei öffnen  
faqs bearbeiten  
faq save as html  
faq save as csv  
faq save as sql

#### +Moodle-Test/GIFT

#### +TAF

öffnen  
übersetzen  
save as html  
save as XML  
+start Webservice

### 1.5.1. Hinweise: FileDialog

```
FileDialog fileDialog= new FileDialog(new Frame(), "Datei öffnen", FileDialog.LOAD);
```

```
fileDialog.setDirectory("e:\\temp");

fileDialog.setVisible(true);

fileDir= fileDialog.getDirectory();
fileName= fileDialog.getFile();

BufferedReader f= new BufferedReader(new FileReader(fileDir+fileName));
```

### 1.5.2. Hinweise: MP3

Siehe: Java-Programm SwingEditor  
<http://javazoom.net>

Bsp: MP3 abspielen

```
/*
 * Version1: MP3 abspielen
 */
    try{
        // InputStream aus MP3
        FileInputStream in = new FileInputStream(fileDir+fileName);

        // Player-Instanz
        Player p = new Player(in);

        // Abspielen
        p.play();

        p.close();
        in.close();
    }
    catch (JavaLayerException jle) {
        System.err.println ("Error: " + jle);
    }
    catch (FileNotFoundException fnf) {
        System.err.println ("Error: " + fnf);
    }
    catch (IOException e) {
        e.printStackTrace();
    }
}
```

Problem:

Für die Zeit des Abspielens kann niemand mit dem Editor arbeiten  
Das ist so nicht zu gebrauchen

Lösung:

parallele Prozesse (Threads)

```
/*
 * Version2:
 * einen eigenen Thread(=Prozess) f. das Abspielen starten
```

```
*/

    java.lang Runnable mp3Player= new Runnable() {
        public void run() {
            try{
                //InputStream aus MP3
                FileInputStream in = new FileInputStream(fileName);

                // Player-Instanz
                Player p = new Player(in);

                // Abspielen
                p.play();

            }
            catch (JavaLayerException jle) {
                System.err.println ("Error: " + jle); }
            catch (FileNotFoundException fnf) {
                System.err.println ("Error: " + fnf); }

        } //end run()
    };

...
    Thread mp3PlayerThread= new Thread(mp3Player);
...
                                mp3PlayerThread.start();

...

Eine andere Version ist die Erstellung einer eigenen Klasse:
// -----
// Version 3: Eigene Thread Klasse
// -----
ThreadMP3 p= new ThreadMP3(_fileName);
p.start();
```

☒ Die Klasse ThreadMP3

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

//JLayer vom Javazoom
/* *
 * Lade
 * http://www.javazoom.net/javayer
 *
 * Kopiere: jl1.0.jar nach C:\Programme\Java\jdk1.5.0\jre\lib\ext
 * oder:
 * Projekt->Eigenschaften-> Java Build Path-> Libraries / add Jar File jl1.0.jar
 *
 */

import javazoom.jl.decoder.JavaLayerException;
import javazoom.jl.player.Player;
```

```
public class ThreadMP3 extends Thread {  
    // 1. PRIVATE Daten (member)  
    private String[] files;  
    private Player p = null;  
    private FileInputStream in;  
  
    // 2. PUBLIC Funktionen/Methoden  
    // Konstruktoren  
    public ThreadMP3(String[] pfiles) {  
        this.files = pfiles;  
    }  
  
    public ThreadMP3(String file) {  
        files = new String[1];  
        files[0] = file;  
    }  
  
    public void ende() {  
        // schliesse input streams und player  
        if (p != null)  
            p.close();  
        try {  
            in.close();  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
  
        // unterbrich laufenden Thread  
        this.interrupt();  
    }  
  
    public void run() {  
        for (int i = 0;  
            i < files.length && this.isInterrupted() == false;  
            i++) {  
            try {  
                // Inputstream aus MP3  
                in = new FileInputStream(files[i]);  
  
                // Player-Instanz  
                p = new Player(in);  
  
                // Abspielen  
                p.play();  
  
                p.close();  
                in.close();  
            } catch (JavaLayerException jle) {  
                System.err.println("Error: " + jle);  
                if (p != null) p.close();  
            }  
        }  
    }  
}
```

```
                in.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        } catch (FileNotFoundException fnf) {
            System.err.println("Error: " + fnf);
        } catch (IOException e) {
            System.err.println("Error: " + e);
        }
    }
}

} //for
} //run
} //end class
```

### 1.5.3. Hinweis: Directory

```
FileDialog fd;
File dir;
String[] files;

fd= new FileDialog(new Frame(),"Verzeichnis wählen",FileDialog.LOAD);
//fd.setDirectory("e:\\mp3");
fd.setVisible(true);

// wurde Abbrechen gedrückt?
if (fd.getFile()== null)
    return;

dir= new File (fd.getDirectory());
// !!!!!!!!
if (dir.isDirectory()) {
    files = dir.list();

    for (int i=0; i<files.length; ++i) {
        files[i] = dir + "\\ " + files[i];
        System.out.println(dir + " --- "+files[i]);
    }

    ...
}
```

### 1.5.4. Hinweis: SwingEditor-FAQ

```
Menüeintrag
FAQ
    faq datei öffnen
    faqs bearbeiten
    faq save as html
    (m)faq save as csv
    faq save as sql
```

☒ Trennung von Daten und der Anzeige der Daten.

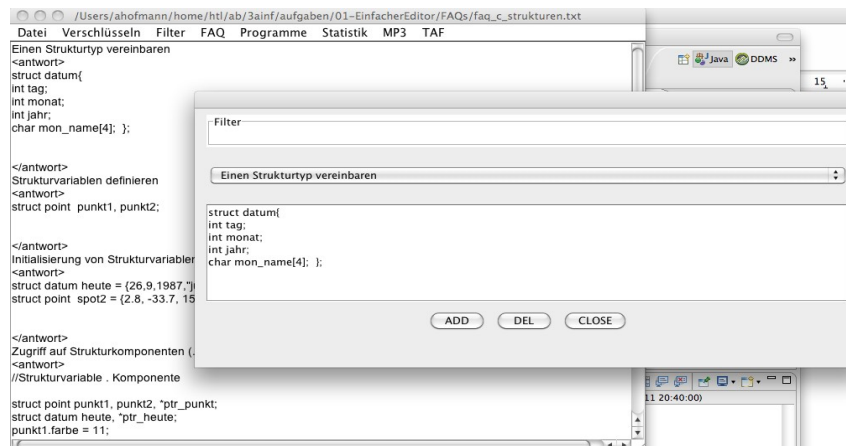
Wir wollen ein 2. Fenster (**SwingFAQ**) entwickeln, das vom SwingEditor aus gestartet wird (faqs bearbeiten).

Dieses Fenster hat eine Combobox und eine Textarea, um sog. FAQ-Daten zu verarbeiten.

Die FAQ-**Daten** werden in einer eigenen Klasse **ModelFAQ** gespeichert.

Wir haben also eine **Trennung des Datenbestandes und der Anzeige**.

Dieses Konzept ist sehr beliebt und ist unter dem Namen **Model-View-Controller** Konzept bekannt.



Bevor wir auf das MVC-Konzept eingehen, wollen wir den Aufbau des Datenbestandes kennen lernen.

Zu verarbeiten ist die Datei faq.txt

☑ Aufbau der Datei: faq.txt

```
frage1
<antwort>
....
....
</antwort>
frage2
<antwort>
....
....
</antwort>
....
```

☑ Aufgabe: **ModelFAQ**

Erstellen Sie die Klasse **ModelFAQ**, die folgenden Aufbau hat und zur Speicherung der Daten dient. Die Daten selbst werden über den Konstruktor als String übergeben. Der Swingeditor hat diese Daten zuvor aus einer Datei (faq.txt) (Menu: FAQ öffnen) in seine EditorPane gelesen.

```
import java.util.Vector;

public class ModelFAQ {
    // PRIVATE Daten
```



```
private Vector<String> fragen;  
private Vector<String> antworten;  
  
// Konstruktor  
public ModelFAQ(){  
}  
  
// befüllt die Vektoren fragen und antworten  
// nach folg. Regel:  
// zerlege den Eingabestring s, der  
// inhaltlich dem Aufbau der Datei  
// faq.txt (s.o.) entspricht.  
// die i-te Frage im Vector fragen soll  
// der i-ten Antwort im Vector antworten entsprechen  
public void setFAQ (String s){  
}  
  
public Vector<String> getFragen(){  
}  
  
public Vector<String> getAntworten(){  
}  
  
// löscht den i-ten Eintrag  
public void del (int i){  
}  
  
// fügt einen neuen Eintrag ein  
public void add (String f, String a){  
}  
  
// liefert einen String, der inhaltlich  
// dem Aufbau der Datei faq.txt (s.o.) entspricht.  
public String toFAQString(){  
}  
  
// liefert einen String, der in eine  
// html datei gespeichert werden kann.  
// Aufbau: siehe faq.htm  
public String toHTMLString(){  
}  
}
```

☒ Aufgabe: **SwingFAQ**

Erstellen Sie die Klasse SwingFAQ:

```
public class SwingFAQ extends JFrame {  
  
    private static final long serialVersionUID = 1L;  
    private JPanel jContentPane = null;  
    private JTextField jTextFieldFilter = null;  
    private JComboBox jComboBoxFrage = null;
```

```
private JPanel jPanel = null;
private JButton jButtonAdd = null;
private JButton jButtonDel = null;
private JButton jButtonClose = null;
private JTextArea jTextAreaAntwort = null;
private JScrollPane jScrollPane = null;

????????
// Ich selbst (d.h die View)
private SwingFAQ thisClass=null;

// Das Model (d.h. die Datenquelle)
private ModelFAQ modelFAQ=null; // // @jve:decl-index=0:

// und jetzt noch der Verweis zum HauptFenster des SwingEditor
private SwingEditor swingEditor=null;

/*
 * setModel: merke die Datenquelle,
 * sodass die SwingFAQ die Daten anzeigen kann
 */
public void setModel(ModelFAQ m){
    modelFAQ= m;
}

/*
 * Konstruktor:
 */
public SwingFAQ() {
    super();
    initialize();

    // Damit die actionPerformed() Methoden
    // auf die View zugreifen können
    thisClass= this;
}

...
```

Wie man sieht, wird in SwingFAQ ein Verweis zu ModelFAQ gespeichert.

Wenn zB. Der DEL-Button gedrückt wird:

```
private JButton getJButtonDel() {
    if (jButtonDel == null) {
```

```
jButtonDel = new JButton();
jButtonDel.setText("DEL");
jButtonDel.addActionListener(
    new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent e) {
            int i;

            i= jComboBoxFrage.getSelectedIndex();

            modelFAQ.del(i);
        }
    });
return jButtonDel;
}
```

Der ADD Button

```
...
String frage,antwort;

//Benutzereingabe
frage= JOptionPane.showInputDialog("Frage");
antwort= JOptionPane.showInputDialog("Antwort");

//im model Eintragen
modelFAQ.add(frage, antwort);
...
```

☒ Aufgabe: SwingEditor: Menu: **faq save as html**

Die Fragen und Antworten sollen als html-Datei gespeichert werden.

Hier ein kl. Beispiel:

faq.html

```
<html>
<head>
<title>FAQ</title>
</head>

<body bgcolor="yellow">

<a name="toc">
<h1>FAQ by Max Mustermann</h1>
<ul>
<li><a href="#1">frage1</a></li>
<li><a href="#2">frage2</a></li>
<li><a href="#3">frage3</a></li>
...
</ul>
```

```
<hr>
<a name= "1">
<h2>Frage1</h2>
<pre>
Antwort
...
</pre>
<p><a href="#toc">Seitenanfang</a></p>

<hr>
<a name= "2">
<h2>Frage2</h2>
<pre>
Antwort
...
</pre>
<p><a href="#toc">Seitenanfang</a></p>

...

</body>
</html>
```

SwingEditor: Aufgabe: SwingEditor: Menu: faq save as html  
bzw.  
ModelFAQ.toHTMLString()

```
...
    PrintWriter out= new PrintWriter(new File(dir+fn));

    // Daten holen
    ModelFAQ modelFAQ= new ModelFAQ();
    modelFAQ.setFAQ(editorPane.getText());

    String s;
    s= modelFAQ.toHTMLString();

    // Daten ausgeben
    out.print(s);

    out.close();

...
```

Jetzt fehlt nur noch die Verbindung der beiden Views mit dem Model.

### 1.5.5. Hinweis: MVC

<http://www.javaworld.com/javaworld/jw-10-1996/jw-10-howto.html?page=3>

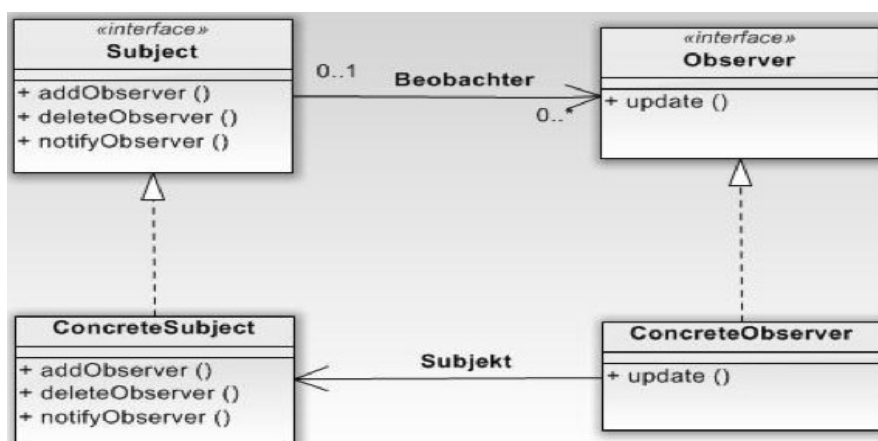
Wir wollen nun das MVC-Konzept kennen lernen und anwenden.

Zum aktuellen Stand:

☒ 2 Views  
class SwingEditor  
class SwingFAQ

☒ Model:  
class ModelFAQ

Hier das Observer-Design-Pattern



Wir passen unsere Klassen an:

Das Model:

```
public class ModelFAQ extends Observable{

    // 1. Schritt:
    // Bei Datenänderungen werden die Views(=Observers) benachrichtigt
    ...
    public void add (String f, String a){
        fragen.addElement(f);
        antworten.addElement(a);

        // MVC
        this.setChanged();
        this.notifyObservers(this);
    }

    // 2. Schritt:
    // Zugriffsmethoden auf die veränderten Daten bereitstellen
    ...
}
```

```
    public Vector<String> getAntworten(){
        return antworten;
    }
}
```

Die Views:

```
public class SwingEditor extends JFrame implements Observer{
...

    /**
     * MVC
     * ObserverPattern
     * Diese VIEW (class SwingEditor) implementiert das
     * OBSERVER-Interface
     *
     * Wenn sich im Model (class ModelFAQ) (=OBSERVABEL) Daten ändern,
     * so wird
     * bei allen Observern die Methode update() wird aufgerufen.
     */

    @Override
    public void update(Observable o, Object arg) {
        editorPane.setText(
            ((ModelFAQ)o).toFAQString() );
    }
}
```

```
public class SwingFAQ extends JFrame implements Observer{
...

    /**
     * MVC
     * ObserverPattern
     * Diese VIEW (class SwingFAQ) implementiert das OBSERVER-Interface
     *
     * Wenn sich im Model (class ModelFAQ) (=OBSERVABEL) Daten ändern,
     * so wird
     * bei allen Observern die Methode update() wird aufgerufen.
     */

    @Override
    public void update(Observable arg0, Object arg1) {
        // VOM MODEL die Daten holen und anzeigen

        //zuerst löschen
        jComboBoxFrage.setEnabled(false);
        jComboBoxFrage.removeAllItems();

        // jetzt alles auffüllen
        for (int i= 0; i < ((ModelFAQ) arg0).getFragen().size(); i++){
```

```
        jComboBoxFrage.addItem(  
            ((ModelFAQ) arg0).getFragen().elementAt(i));  
    }  
    jComboBoxFrage.setSelectedIndex(0);  
    jComboBoxFrage.setEnabled(true);  
  
    jTextAreaAntwort.setText(  
        ((ModelFAQ) arg0).getAntworten().elementAt(0));  
  
    jTextFieldFilter.setText("");  
}  
}
```

## 1.6. Aufgabe: SwingPizza

Verschiedene Dialogelemente, Erstellen von PDF/RTF-Dokumenten, Exceldateien, ...

### Hinweise:

#### ButtonGroup:

☒ Die Button markieren-> re.Maus->set Buttongroup-> new...

**Panel mit Überschrift:**

Eigenschaft:Border-&gt; Titled

**Benennungen:**

rdButtonSize1

rdButtonSize2

rdButtonSize3

chkBoxAuflage1

chkBoxAuflage2

chkBoxAuflage3

**1.6.1. Die Datei pizza.ini und die Klasse Pizza.java**

Erstellen Sie die Datei pizza.ini mit folg. Inhalt:

```
3
BIG:10:selected
Medium:6:not selected
Small:4:not selected
3
Oliven:1:selected
Zwiebel:2:not selected
Salami:1:selected
5
Pommes:3:not selected
Cola:2:selected
Salat:3:selected
Eis:1:not selected
Ketchup:1:not selected
```

Auf der Grundlage dieser Datei wollen wir die Klasse Pizza erstellen, die zur Speicherung der Daten dient. Wir verwenden dazu sog. HashMaps

```
public class Pizza {
    private HashMap<String, String> selectedSize= new HashMap<String,
String>();
    private HashMap<String, Integer> preisSize= new HashMap<String,
Integer>();

    private HashMap<String, String> selectedAuflagen= new
HashMap<String, String>();
    private HashMap<String, Integer> preisAuflagen= new HashMap<String,
Integer>();

    private HashMap<String, String> selectedZutaten= new
HashMap<String, String>();
    private HashMap<String, Integer> preisZutaten= new HashMap<String,
Intrege>();

...
    public Pizza(String filename){...}
}
```



Im Konstruktor sollen die Daten von der Datei ins Ram gelesen werden.

### 1.6.2. UI-Elemente initialisieren

Die Objekterzeugung soll beim WindowOpened-Ereignis erfolgen.

```
Pizza pizza;
...
pizza= new Pizza("pizza.ini");

// ui-Element: size
String size1, size2, size3;
size1= pizza.getTextSize1();
size2= pizza.getTextSize2();
size3= pizza.getTextSize3();

rdButtonSize1.setText(size1);
rdButtonSize2.setText(size2);
rdButtonSize3.setText(size3);

if (pizza.getSelectedSize(size1))
    rdButtonSize1.setSelected(true);
....

txtPreis= String.valueOf(pizza.getPreis());
```

### 1.6.3. Auf UI-events registrieren

Wenn nun die UI-Element vom User aktiviert wird (zB: chkBoxAuflage1), so muss diese Änderung auch den Pizza-Objekt mitgeteilt werden, sodass auch anschliessend der richtige Preis berechnet werden kann.

### 1.6.4. Button: Beleg erstellen

Hier werden alle Information zur aktuellen Bestellung gesammelt und in der Textarea txtBeleg angezeigt.

### 1.6.5. Hinweis: PDF erstellen

```
* Schritt 1:
* lies:
* http://www-128.ibm.com/developerworks/opensource/library/os-javapdf/
*
* Schritt2:
```

- \* install:
- \* <http://itextdocs.lowagie.com/tutorial/objects/index.php>
- \* Project->Properties->Java Build path->Libraries
- \* ->add jar-file->itext-2.0.6.jar
- \* oder
- \* itext-2.0.6.jar kopieren nach C:\Programme\Java\jdk1.5.0\jre\lib\ext
- \* sofern dies ihr verwendete JRE ist.

PDF erstellen:

siehe: <http://itextdocs.lowagie.com/tutorial/objects/index.php>

- \* Step 1: Create an instance of `com.lowagie.text.Document`:

```
Document document = new Document();
```

- \* Step 2:

Create a `Writer` (for instance `com.lowagie.text.pdf.PdfWriter`) that listens to this document and writes the document to the `OutputStream` of your choice:

```
PdfWriter.getInstance(document,  
    new FileOutputStream("HelloWorld.pdf"));
```

- \* Step 3:

Open the document:

```
document.open();
```

- \* Step 4: Add content to the document:

```
document.add(new Paragraph("Hello World"));
```

- \* Step 5: Closes the document:

```
document.close();
```

siehe auch:

- \* [http://javamagazin.de/itr/online\\_artikel/psecom,id,441,nodeid,11.html](http://javamagazin.de/itr/online_artikel/psecom,id,441,nodeid,11.html)
- \* <http://schmidt.devlib.org/java/libraries-pdf.html>

### 1.6.6. Hinweis: EXCEL

- \* zeigt die Verwendung von excel und word dateien
- \* <http://poi.apache.org/hssf/quick-guide.html>
- \*

```
import org.apache.poi.hssf.usermodel.*;  
public static void main(String[] args)  
    throws IOException  
{
```

```
HSSFWorkbook wb = new HSSFWorkbook();
HSSFSheet sheet = wb.createSheet("new sheet");
HSSFRow row = sheet.createRow((short) 2);
createCell(wb, row, (short) 0, HSSFCellStyle.ALIGN_CENTER);
createCell(wb, row, (short) 1, HSSFCellStyle.ALIGN_CENTER_SELECTION);
createCell(wb, row, (short) 2, HSSFCellStyle.ALIGN_FILL);
createCell(wb, row, (short) 3, HSSFCellStyle.ALIGN_GENERAL);
createCell(wb, row, (short) 4, HSSFCellStyle.ALIGN_JUSTIFY);
createCell(wb, row, (short) 5, HSSFCellStyle.ALIGN_LEFT);
createCell(wb, row, (short) 6, HSSFCellStyle.ALIGN_RIGHT);

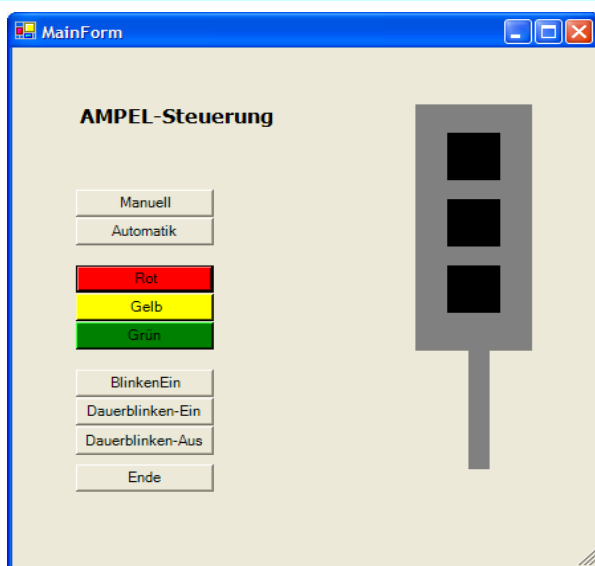
// Write the output to a file
FileOutputStream fileOut = new FileOutputStream("workbook.xls");
wb.write(fileOut);
fileOut.close();

}
```

### 1.6.7. Hinweis: XML:

\* siehe JDOM

## 1.7. Aufgabe: SwingAmpel (HU)



Es soll eine primitive manuelle Schaltung für eine Straßenbaustellenampel erstellt werden. Ein Mann/Frau soll die Ampel per Hand steuern können.

Die Ampel hat die drei Lampen Rot, Grün und Gelb und zum Schalten für jede Lampe einen Button.

Wenn z. B. die rote Lampe leuchtet, müssen natürlich dann die beiden anderen ausgeschaltet werden. Beim Programmstart sind alle Lampen schwarz.

### Übung 1.0: Die Ampel erstellen

☒ Bauen Sie aus Panels und Buttons die "Ampel" und Steuerung zusammen!

- ☒ Erzeugen Sie durch Klick auf die Buttons (Rot, Gelb, Grün) die Ereignismethoden und implementieren Sie den Code!

### Übung 1.1 – Mehrfaches Blinken: Schleifen (Zählschleife, vor- und nachprüfende Schleife)

#### BlinkenEin

Als Erweiterung soll eine Blinkschaltung für die gelbe Lampe eingebaut werden. Dazu wird ein neuer Button: **jButtonBlinkenEin** (Text: BlinkenEin) eingebaut.

Das Blinken lässt sich mit einer Schleife realisieren, die dafür sorgt, dass die gelbe Lampe im Wechsel 'an' oder 'aus' gezeigt wird (mit einer kleinen Pause dazwischen). Dies soll z.B. 5 mal passieren.

#### Hinweis:

```
final int intAnzahl =5;
.....
for (int i=0; i < intAnzahl; i++){
    ....
    java.lang.Thread.sleep( zeit in Millisekunden)
    ....
}
```

#### 1.7.1. Hinweis: Threads

##### Problem: `java.lang.Thread.sleep(1000);`

Da durch dieses Sleep der aktuelle Thread (hier die GUI Oberfläche) schlafen gelegt wird, können keine Buttons gedrückt werden. Es können auch keine Panels eingefärbt werden.

##### Lösung: Thread

Deswegen muss man einen eigenen Thread erstellen, der parallel zur möglichen Usereingabe, die Panels einfärbt und dann auch noch genügend lang schläft.

- ☒ Zunächst definieren wir eine Variable namens `runBlinkenEin` vom Typ `java.lang Runnable`.

```
java.lang.Runnable runBlinkenEin= new java.lang.Runnable()
{
    public void run()
    {
        for (int i=0; i < intAnzahl; i++)
        {
            jPanelGelb.setBackground(Color.yellow);

            try{
                java.lang.Thread.sleep(1000);

                jPanelGelb.setBackground(Color.black);

                java.lang.Thread.sleep(1000);

            } catch (InterruptedException err){
                err.printStackTrace(System.out);
            }
        }
    }
    }
};
```

☒ Wenn nun der Button **jButtonBlinkenEin** gedrückt wird, dann wird mit

....

```
public void actionPerformed(java.awt.event.ActionEvent e) {
```

```
    java.lang.Thread blinkenEinThread= new java.lang.Thread(runBlinkenEin);  
    blinkenEinThread.start();
```

....

der oben programmierte Thread gestartet.

siehe auch:

[http://www.dpunkt.de/java/Programmieren\\_mit\\_Java/Oberflaechenprogrammierung/44.html](http://www.dpunkt.de/java/Programmieren_mit_Java/Oberflaechenprogrammierung/44.html)

### Übung 1.2 – DauerBlinken: Globale Zustandsvariablen und while

Wenn die Zählschleife fehlerfrei läuft, erstellen Sie bitte eine weitere Funktionalität der Ampel, die die WHILE-Schleife verwendet:

Dazu wollen wir zwei weitere Buttons einsetzen:

☒ **jButtonDauerblinkenEin** (Text: Dauerblinken-Ein)

☒ **jButtonDauerblinkenAus** (Text: Dauerblinken-Aus)

Die Ereignisprozedur von **jButtonDauerblinkenEin** enthält eine Schleife, die keine voreingestellte Wiederholzahl hat. Man verwendet dabei eine sog. WHILE-Schleife, die jedesmal die aktuelle Abbruchbedingung prüft. Dazu führen wir eine Zustandsvariable

```
boolean boolBlinken;
```

ein, die mit einem neuen Aus-Button (**jButtonDauerblinkenAus**) auf den Wert *false* gesetzt wird.

#### Hinweis: Definition einer globalen Variablen

Damit mehrere Ereignisprozeduren eine Information gemeinsam verwenden, kann man u.a. so genannte globale Variablen verwenden. Diese werden am Beginn des Programmes (ausserhalb von Funktionen/Methoden) definiert.

```
// Meine gloablen Variablen  
boolean boolBlinken= false;
```

Die einzelnen Funktionen/Methoden können dann diese Variable auslesen bzw. einen neuen Wert setzen.

Hinweis: Synchronisation

Natürlich darf nicht jeder Thread eine globale Variable beliebig ändern. Dieses Problem der Synchronisation wollen wir aber hier nicht behandeln.

### Übung 1.3 – Notabschaltung: Entscheidungen (if - then - else)

Im letzten Ausbauschnitt soll ein Zähler eingebaut werden, der die Blinkvorgänge mitzählt, um bei bestimmten Situationen automatisch abzuschalten.

**Wenn** der Zähler den Wert 10 erreicht,  
dann eine Warnung ausgegeben

**sonst wenn** der Wert 15 erreicht ist,  
dann eine Notabschaltung ausführen

**ansonsten** Zählerwert anzeigen

### Übung 1.4 – Automatik: Entscheidungen ()

Baustellenampel automatisieren

Eine ordentliche Ampel muss auch nachts und am Wochenende automatisch laufen können. Das ist ziemlich einfach zu programmieren. Die Schaltphasen der drei Lampen brauchen nur nacheinander mit kleinen Pausen dazwischen abzulaufen.

Die Button-Clicks werden nicht mehr manuell bedient, sondern in einer Methode vom Programm selbst wiederholt bis die Automatik abgestellt wird.

Dazu sind folgende Schritte erforderlich:

#### Ausbaustufe 1 : Automatik einbauen

- ☒ Führen Sie zwei neue Buttons ein: **jButtonAutomatik** und **jButtonManuell**
- ☒ Deklarieren Sie eine globale Variable mit **boolean boolAutomatik**;  
mit der sich das Programm merkt, ob die automatische Steuerung gerade ein- oder ausgeschaltet ist.
- ☒ Erzeugen Sie eine Runnable-Variable:

```
java.lang.Runnable runAutomatik= new Runnable()
{
    public void run()
    {

        boolAutomatik=true; // @jve:decl-index=0:
        while (boolAutomatik)
        {
            try
            {
                // rot
                java.lang.Thread.sleep(2000 );

                //gelb
                java.lang.Thread.sleep(2000 );

                //gruen
                java.lang.Thread.sleep(2000 );
            }
            catch (InterruptedException err){
                err.printStackTrace(System.out);
            }
        }
    }
};
```

### 1.7.2. Hinweis: Ereignisprozeduren direkt aufrufen

Man kann das Klick-Ereignis mittels Aufruf einer Funktion nachahmen:

```
jButtonRot.doClick(10);
```

- ☒ Starten Sie das Programm und testen Sie es!

**Ausbaustufe 2 : Umschalten**

Damit die Handsteuerung und die automatische Steuerung nicht gleichzeitig bedient werden können, sollten die Einzelbuttons deaktiviert sein.

```
JbuttonGelb.enabled(false);
```

```
....
```

**Zusammenfassung:**

Erstelle eine Zusammenfassung des in diesem Kapitel gelernten Stoffes in einer Word-Datei

java-ampel.doc (mit kl. Programmbeispielen)

- ☒ Variablen, Konstanten definieren
- ☒ Datentypen verstehen und anwenden
- ☒ Globale Variable, Lokale Variable
- ☒ Verzweigungen
- ☒ Schleifen
- ☒ Threads

## 1.8. Aufgabe: SwingGameOfLife

---

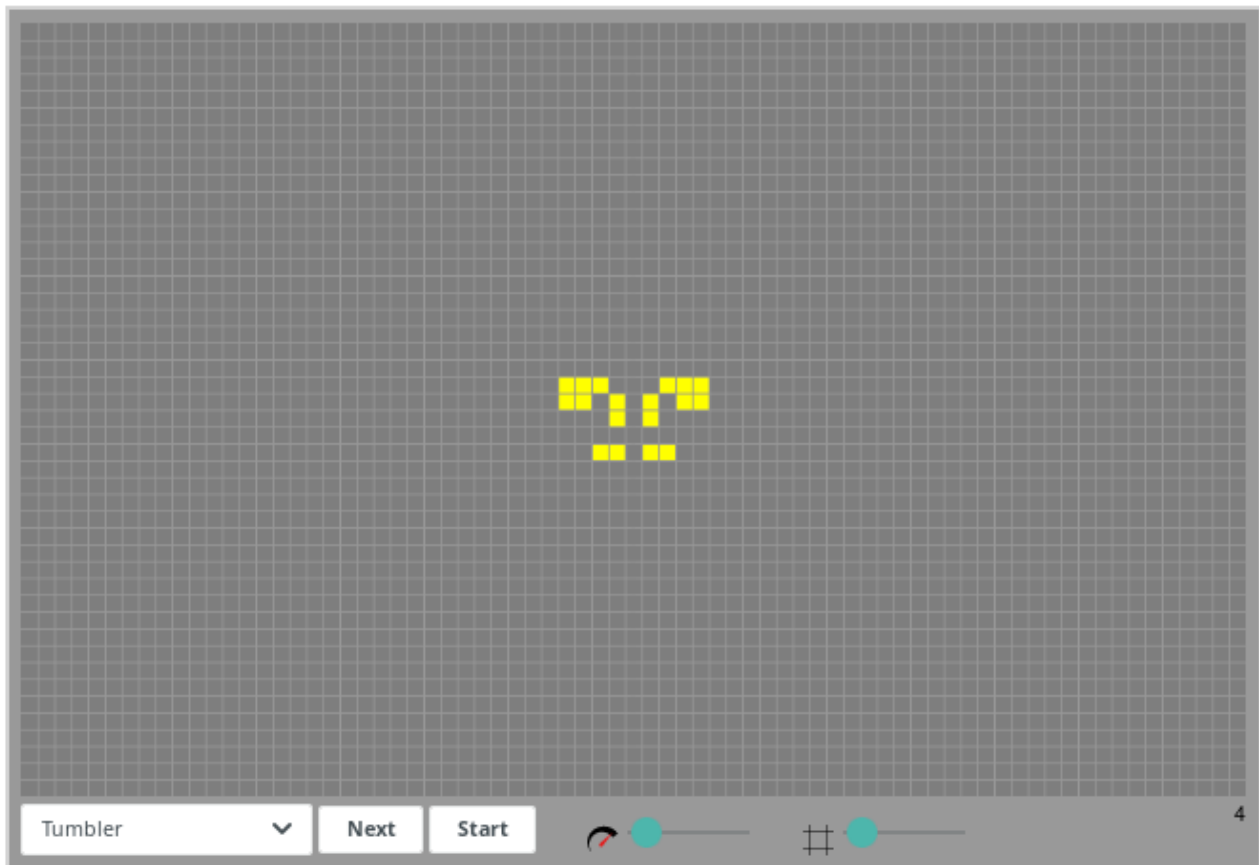
### 1.8.1. See: <https://bitstorm.org/gameoflife/>

---

### 1.8.2. UI

---

## The Simulation



### 1.8.3. The Rules

#### For a space that is 'populated':

- Each cell with one or no neighbors dies, as if by solitude.
- Each cell with four or more neighbors dies, as if by overpopulation.
- Each cell with two or three neighbors survives.

#### For a space that is 'empty' or 'unpopulated'

- Each cell with three neighbors becomes populated.

### 1.8.4. Hinweis: mehrere Button mit gleicher Ereignisprozedur

Hier ein Beispiel:

```
// Tastatur
char ch='A';
for (int y = 0; y < 2; y++) {
    for (int x = 0; x < 13; x++) {
        tastatur[x][y]= new JButton(String.valueOf(ch));
        tastatur[x][y].setBounds(x * 30, y*30, 30, 30);
        tastatur[x][y].addActionListener(new ActionListener() {
```



```
        @Override
        public void actionPerformed(ActionEvent e) {
            JButton btn= (JButton) e.getSource();

            btn.setEnabled(false);
            String sCode= btn.getText();
            char tryChar= sCode.charAt(0);
            System.out.println(tryChar);

        }
    });

    getPanel_Tastatur().add(tastatur[x][y]);

    ch ++;
}
}
}
```

## 1.9. Aufgabe: SwingHangman

### Aufgabe: Hangman

Realisieren Sie das bekannte Wortratespiel auf dem Computer. Zeigen Sie zunächst von einem zu ratenden Wort für die einzelnen Zeichen jeweils nur Striche an.

Anschließend fragen Sie nach Buchstaben. Sollten die Buchstaben im zu ratenden Wort vorkommen, so werden die entsprechenden Positionen aufgedeckt, d.h. an dieser Stelle werden die tatsächlichen Zeichen angezeigt. Durch Eingabe eines Zeichens (z.B. '#' – welches dann in keinem Wort vorkommen darf) kann das Raten abgebrochen werden und die Eingabe des vollständigen Worts ist nun möglich. Stimmt das eingegebene Wort mit dem verdeckten Wort überein, ist das Spiel zu Ende, ansonsten wird wieder in den obigen Buchstabenratemodus zurückgekehrt.

Beispiel: Zu ratendes Wort: "Hangman" (verdeckt)

```
- - - - -      Eingabe (a)
- a - - - a -  Eingabe (g)
- a - g - a -  Eingabe (e)
- a - g - a -  Eingabe (n)
- a n g - a n  Eingabe (#) ~ Ende Buchstabenraten
```

Eingabe: Hangman → richtig geraten / bzw. Zurückkehren zum Buchstabenraten

### Hinweis:

Sie können das Spiel abwechslungsreicher gestalten, wenn Sie sich eine Tabelle von zu ratenden Wörtern im Programm anlegen und aus dieser Tabelle zufällig einen Eintrag auswählen, anstatt diesen vorher einzugeben.

### 1.9.1. Hinweis; Klasse Hangman

---

Allg. Vorgehen:

=====

1. ANALYSE (WAS)
2. ENTWURF (WIE)
  - 2.1. Grobentwurf (Module, Datenstrukturen; Aufteilung auf Personen)
  - 2.2. Feinentwurf (Klassen/Methoden/Funktionen)
3. IMPLEMENTIERUNG
  - 3.1. Grobgerüst ('leere' Funktionen)
  - 3.2. Feinimplementierung ('fertige' Funktionen)
4. TESTS
5. ABNAHME

=====

```
/*
 *
Hangman: Klassen ...
class Hangmann:
    private String word2show; // -E--O
    private String word2guess; // HELLO

public
    void newGame(void);

    String getWord2show();
    String getWord2guess();
    int getBadTries();
    int getMaxTries();

    int tryChar(char ch);
    int tryWord(word);

    int success();
    int gameOver();

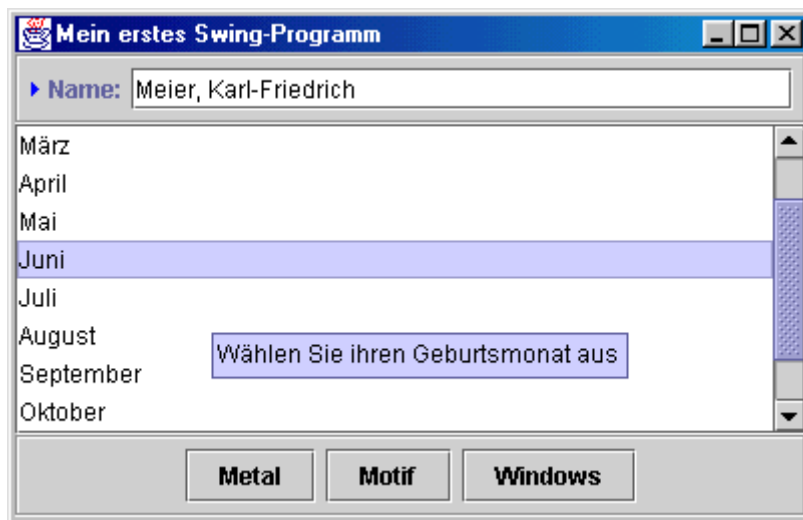
*/
```

TODO

Add: SwingPromille, SwingZinsen, SwingTanken

### 1.10. Zusammenfassung: Ein Beispiel

---



```
/* Gui1.java */

import java.awt.event.*;
import java.awt.*;
import javax.swing.*;

public class Gui1
extends JFrame
implements ActionListener
{
    private static final String[] MONTHS = {
        "Jänner",    "Februar", "März",    "April",
        "Mai",       "Juni",    "Juli",    "August",
        "September", "Oktober", "November", "Dezember"
    };

    public Gui1()
    {
        super("Mein erstes Swing-Programm");

        //Panel zur Namenseingabe hinzufügen
        JPanel namePanel = new JPanel();

        JLabel label = new JLabel(
            "Name:",
            new ImageIcon("triblue.gif"),
            SwingConstants.LEFT
        );

        namePanel.add(label);

        JTextField tf = new JTextField(30);
        tf.setToolTipText("Geben Sie ihren Namen ein");

        namePanel.add(tf);
    }
}
```

```
namePanel.setBorder(BorderFactory.createEtchedBorder());

getContentPane().add(namePanel, BorderLayout.NORTH);

//Monatsliste hinzufügen
JList list = new JList(MONTHS);
list.setToolTipText("Wählen Sie ihren Geburtsmonat aus");
getContentPane().add(new JScrollPane(list), BorderLayout.CENTER);

//Panel mit den Buttons hinzufügen
JPanel buttonPanel = new JPanel();

JButton button1 = new JButton("Metal");
button1.addActionListener(this);
button1.setToolTipText("Metal-Look-and-Feel aktivieren");
buttonPanel.add(button1);

JButton button2 = new JButton("Motif");
button2.addActionListener(this);
button2.setToolTipText("Motif-Look-and-Feel aktivieren");
buttonPanel.add(button2);

JButton button3 = new JButton("Windows");
button3.addActionListener(this);
button3.setToolTipText("Windows-Look-and-Feel aktivieren");
buttonPanel.add(button3);

buttonPanel.setBorder(BorderFactory.createEtchedBorder());
getContentPane().add(buttonPanel, BorderLayout.SOUTH);

//Windows-Listener
addWindowListener(new WindowClosingAdapter(true));
}

public void actionPerformed(ActionEvent event)
{
    String cmd = event.getActionCommand();
    try {
        //PLAF-Klasse auswählen
        String plaf = "unknown";
        if (cmd.equals("Metal")) {
            plaf = "javax.swing.plaf.metal.MetalLookAndFeel";
        } else if (cmd.equals("Motif")) {
            plaf = "com.sun.java.swing.plaf.motif.MotifLookAndFeel";
        } else if (cmd.equals("Windows")) {
            plaf = "com.sun.java.swing.plaf.windows.WindowsLookAndFeel";
        }

        //LAF umschalten
        UIManager.setLookAndFeel(plaf);
        SwingUtilities.updateComponentTreeUI(this);

    } catch (UnsupportedLookAndFeelException e) {
        System.err.println(e.toString());
    }
}
```

```
    } catch (ClassNotFoundException e) {  
        System.err.println(e.toString());  
    } catch (InstantiationException e) {  
        System.err.println(e.toString());  
    } catch (IllegalAccessException e) {  
        System.err.println(e.toString());  
    }  
}  
  
public static void main(String[] args)  
{  
    Listing3501 frame = new Listing3501();  
    frame.setLocation(100, 100);  
    frame.pack();  
    frame.setVisible(true);  
}  
}
```