

Swing Workshop

Dieser Workshop enthält grundsätzliche Informationen über Swing und Java und regt mit einigen Beispielen und Übungen zum weiteren Erkunden dieses Themengebietes an.

Kapitel 1: Das erste Gui

Vereinfacht ausgedrückt ist Swing eine große Ansammlung von Klassen, die Sun für Java entwickelt hat, mit denen man mit wenig Arbeit bereits interaktive grafische Benutzeroberflächen erschaffen kann. Alles was man braucht um so eine Benutzeroberfläche zu erstellen ist die Schnittstellenbeschreibung (JavaDocs) der Swing-Bibliothek.

Damit wir die Swing Klassen nutzen können wie **JLabel**, **JButton** etc., müssen wir zunächst diese Packages **importieren**. Das geht wie folgt:

Am Anfang unserer Datei listen wir zeilenweise auf, was wir alles für diese Klasse brauchen.

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

Das Wort **import** versteht sich von selbst.

In unserem Beispiel importieren wir **java.awt.***, da Swing darauf aufbaut und wir somit auch Klassen aus diesem Package brauchen. Der Import von **java.awt.event.*** ist notwendig, damit unsere Oberfläche interaktiv wird und wir auf z.B. Button-Klicks reagieren können. Schließlich importieren wir dann noch **javax.swing***. Der ***** bewirkt, dass alle direkten Unterklassen des angegebenen Packages importiert werden. Wir könnten auch jede Klasse die wir brauchen einzeln importieren. Doch sparen wir uns das an dieser Stelle. Der Debugger läuft idR jedoch schneller, wenn nur das importiert wird, was man wirklich braucht. Wie wir es von Java auch schon vom Ende einer Befehlszeile in einer Methode kennen, muss auch die Importzeile mit einem **;** enden.

Nach diesen Zeilen folgt nun der Beginn unserer Klasse. Diese nennen wir **ErstesFenster**. Was die meisten bei Windows als ein Fenster bezeichnen heißt bei Java **JFrame**. Da wir ein **JFrame** erzeugen wollen, ist unsere Klasse eine Spezialisierung von **JFrame**. Somit müssen wir uns um das Darstellen des Guis die Verwaltung der einzelnen Komponenten etc. gar keine Gedanken mehr machen.

Und so sieht es aus, wenn man eine Klasse, die eine Spezialisierung von **JFrame** ist definieren will:

```
public class ErstesFenster extends JFrame {
    //...
}
```

- | | |
|------------------------|---|
| Public: | Mindestens eine Klasse in einer Datei muss public sein. |
| class: | Deutet an, dass hier eine neue Klasse definiert wird. |
| ErstesFenster: | Der Name, den ich für die Klasse gewählt habe. |
| extends JFrame: | Die Klasse ist also eine Spezialisierung von JFrame . |
| {//...}: | In diese Klammern kommen später alle Methoden und Attribute rein. |

Dass wir nicht genau wissen wie ein **JFrame** funktioniert, ist an dieser Stelle nicht schlimm. Die Schnittstellenbeschreibung reicht uns, um mit **JFrame** arbeiten zu können.

Als nächstes wollen wir ein paar Komponenten dem **JFrame** hinzufügen. Ein **JLabel** ist ein einfaches Feld, das einzeiligen Text darstellen kann. Bei den Attributen fügen wir also ein **JLabel** hinzu:

```
private JLabel lb_InfoText;
```

Attribute sind in der Regel private. Bei diesem **JLabel** gibt es keinen Grund etwas anderes zu wählen. Der Attribut-Typ kommt stets vor dem Namen des Attributes (**lb_InfoText**). Das **lb_** (für Label) steigert bei einer Klasse mit vielen Komponenten die Übersicht.

Unser Gui soll außerdem ein **JTextArea** bekommen. Ein **JTextArea** ist ideal zum Darstellen von viel Text in mehreren Zeilen, wenn die Formatierung des Textes einheitlich ist.

```
private JTextArea ta_TextAusgabe;  
private JScrollPane sp_ta_TextAusgabe;
```

Es mag zunächst mal verwundern, dass von **einem JTextArea** die Rede ist jedoch **zwei** Zeilen zum Code hinzukommen. Einmal - wie zu erwarten - ein **JTextArea** und ein mal ein **JScrollPane**. Das **JScrollPane** ist nicht zwingend notwendig um ein funktionierendes **JTextarea** zu erzeugen. Allerdings kann man mit dem **JScrollPane** mit wenig Aufwand Scrollbalken hinzufügen und so das **JTextArea** wesentlich benutzerfreundlicher gestalten.

Zum Schluss brauchen wir noch einen **JButton**, damit wir auch etwas machen können in unserem Gui.

```
private JButton bt_Aktion;
```

Unsere Datei sollte demnach wie folgt aussehen:

```
import java.awt.*;  
import java.awt.event.*;  
import javax.swing.*;  
  
public class ErstesFenster extends JFrame {  
    private JLabel lb_InfoText;  
    private JTextArea ta_TextAusgabe;  
    private JScrollPane sp_ta_TextAusgabe;  
    private JButton bt_Start;  
}
```

Als nächstes werden wir uns ein wenig mit dem Konstruktor unserer Klasse befassen.

```
public ErstesFenster () {}
```

Wie bereits bekannt sein sollte muss der Konstruktor in Java exakt so betitelt sein wie die Klasse. Hier besonders auf Groß- und Kleinschreibung achten! Wie bereits vorher gesagt, wollen wir uns gar nicht um die ganze Darstellung der Komponenten kümmern. Also rufen

wir den Konstruktor von **JFrame** auf damit bereits alles nötige erzeugt wird, was zum anzeigen der Komponenten notwendig ist.

```
public ErstesFenster () {
    super("Erstes Programm");
}
```

Die Methode **super** kann nur ganz am Anfang des Konstruktors aufgerufen werden. Sie ruft automatisch den Konstruktor der Klasse auf dessen Spezialisierung **ErstesFenster** ist (also **JFrame**).

Der Konstruktor von **JFrame** verlangt einen String. Dieser String wird nachher in der Fensterleiste dargestellt. Wir übergeben hier einfach den String **Erstes Programm**.

Als nächstes müssen wir dafür sorgen, dass unser Programm beim Klicken auf das "X" (bei Windows) im Fenster auch beendet wird.

Dafür dient folgender Code:

```
public ErstesFenster () {
    super("Erstes Programm");
    setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
}
```

Die Methode **setDefaultCloseOperation** haben wir von **JFrame** geerbt. Wir können sie also einfach mitbenutzen. **WindowConstants** ist Teil von **Swing**. **EXIT_ON_CLOSE** ist eine Konstante (gekennzeichnet durch den vollständig in Großbuchstaben gehaltenen Namen (Es ist dem Compiler bei Konstanten theoretisch egal ist, ob diese aus Großbuchstaben bestehen oder nicht. Aber bei Java ist dies nunmal der durchgehende "Stil")) davon.

Der nächste Schritt ist es dem Fenster eine Größe zu geben und auf dem Monitor zu platzieren. Wir könnten dies in Abhängigkeit der Monitorgröße tun. Uns soll es aber genügen einfach eine feste Größe vorzugeben.

```
public ErstesFenster () {
    super("Erstes Programm");
    setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    setSize(300, 300);
    setLocation(100, 50);
}
```

Die Größe wird mit Hilfe der Methode **setSize** festgelegt (Breite, Höhe). Mit der Methode **setLocation**(Abstand zum linken Bildschirmrand, Abstand zum oberen Bildschirmrand).

```
Container cp = getContentPane();
cp.setLayout(null);
```

ContentPane ist vereinfacht gesagt unsere Pinnwand. Wir stecken all unsere Komponenten fest. Die Komponente die wir als letztes hinzufügen verdeckt möglicherweise dann Komponenten die zu einem früheren Zeitpunkt hinzugefügt wurden. Die Methode **getContentPane()** haben wir ebenfalls von **JFrame** geerbt.

Damit wir unsere Objekte jetzt erstmal einfach platzieren können und uns nicht mit **LayoutManager** rumschlagen müssen löschen wir den **StandardLayoutManager** in dem wir die Methode **setLayout(null)** vom **ContentPane** aufrufen und **null** (also nichts) übergeben.

Jetzt müssen wir nur noch unsere Komponenten platzieren.

```
lb_InfoText = new JLabel("Drücke auf den Button");
```

ODER

```
lb_InfoText = new JLabel();  
lb_InfoText.setText("Drücke auf den Button");
```

Mit der ersten Zeile oben haben wir gleich zwei Dinge auf einmal getan. Zunächst haben wir ein neues **JLabel** erzeugt. Dann haben wir gleichzeitig auch noch dem **JLabel** den Text **Drücke auf den Button** verpasst. Man kann das Ganze auch in zwei Schritte aufteilen, wenn man das übersichtlicher findet. Siehe dazu den zweiten Vorschlag.

```
lb_InfoText.setBounds(50, 10, 200, 16);
```

Die Methode **setBounds** platziert unser **JLabel** auf dem **ContentPane**. (x-Position der LO Ecke, y-Position der LO Ecke, Breite, Höhe) Die Position ist immer in Relation zum **JFrame** und nicht in Relation zum Bildschirm!

```
cp.add(lb_InfoText);
```

Nachdem unser **JLabel** nun alle notwendigen Informationen hat, können wir es über die **add(...)** Methode des **ContentPane** "anpinnen".

Kommen wir nun zu unserem **TextArea**. Die sichtbare Größe legen wir über das **JScrollPane** fest.

```
sp_ta_TextAusgabe = new JScrollPane();  
ta_TextAusgabe = new JTextArea();  
sp_ta_TextAusgabe.setBounds(50, 50, 200, 150);  
ta_TextAusgabe.setText("Dieser Text ist beim Start des Programmes bereits zu sehen");  
sp_ta_TextAusgabe.setViewportViewView(ta_TextAusgabe);  
cp.add(sp_ta_TextAusgabe);
```

Die **setBounds(...)** und **setText(...)** Methode kennen wir bereits von **JLabel**. Hier erkennt man schon, dass diese Methode möglicherweise von einer allgemeineren Klasse vererbt wurde. Über die Methode **setViewportView(ta_TextAusgabe)** sagen wir dem **JScrollPane** was denn mit Scrollbars umgeben werden soll. Anschließend fügen wir das **JScrollPane** noch dem **ContentPane** zu (unserer Pinnwand).

INFO

Und tatsächlich ist die Methode **setBounds()** beispielsweise ursprünglich von der Klasse **Component**. **JScrollPane** ist nämlich eine Spezialisierung der Klasse **JComponent**, die eine Spezialisierung der Klasse **Container**, die eine Spezialisierung der Klasse **Component** ist.

Zum Schluss noch der **Button**:

```
bt_Start = new JButton();  
bt_Start.setBounds(104, 240, 97, 25);  
bt_Start.setText("Klick Mich");  
cp.add(bt_Start);
```

Soweit sollte alles klar sein.

Jetzt muss dem Button noch gesagt werden, was er zu tun hat, wenn man auf ihn klickt.

```
    ActionListener meinNeuerActionListener = new ActionListener() {  
        public void actionPerformed(ActionEvent evt) {  
            ta_TextAusgabe.setText("Der Button funktioniert anscheinend.");  
        }  
    };  
    bt_Start.addActionListener(meinNeuerActionListener);
```

Wir fügen dem Button einen Listener hinzu, der überwacht, ob der Button gedrückt wird. Die Art des Listeners ist ein **ActionListener**. Immerhin soll der Button gedrückt werden und die Maus sich nicht nur über ihn bewegen. Anschließend überschreiben wir in {} die Methode **actionPerformed** von unserem neu erzeugten **ActionListener**. Diese wird jedesmal aufgerufen, wenn der **ActionListener** registriert, dass der Button gedrückt wurde. Dass wir hierbei ein **ActionEvent** übernehmen interessiert uns gar nicht. Das hängt einzig und allein damit zusammen, dass wir die Methode überschreiben wollen. Weiter brauchen wir das **ActionEvent** aber nicht. Anstattdessen ändern wir einfach den Text des TextAreas. Wir könnten hier z.B. auch eine Methode unserer Klasse ErstesFenster aufrufen.

```
    setResizable(false);  
    setVisible(true);
```

Diese von **JFrame** stammenden Methoden sollten leicht verständlich sein.

Hier ist der fertige Konstruktor:

```
public ErstesFenster () {  
    super("Erstes Programm");  
    setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);  
    setSize(300, 300);  
    setLocation(100, 50);  
    Container cp = getContentPane();  
    cp.setLayout(null);  
  
    lb_InfoText = new JLabel("Drücke auf den Button");  
    lb_InfoText.setBounds(50, 10, 200, 16);  
    cp.add(lb_InfoText);  
  
    sp_ta_TextAusgabe = new JScrollPane();  
    ta_TextAusgabe = new JTextArea();  
    sp_ta_TextAusgabe. setBounds(50, 50, 200, 150);  
    ta_TextAusgabe.setText("Dieser Text ist beim Start des Programmes bereits zu  
                            sehen");  
    sp_ta_TextAusgabe.setViewportView(ta_TextAusgabe);  
    cp.add(sp_ta_TextAusgabe);  
  
    bt_Start = new JButton();  
    bt_Start.setBounds(104, 240, 97, 25);  
    bt_Start.setText("Klick Mich");  
    cp.add(bt_Start);  
    ActionListener meinNeuerActionListener = new ActionListener() {
```

```

        public void actionPerformed(ActionEvent evt) {
            ta_TextAusgabe.setText("Der Button funktioniert anscheinend.");
        }
    };
    bt_Start.addActionListener(meinNeuerActionListener);

    setResizable(false);
    setVisible(true);
}

```

Noch ein Schritt fehlt zum ersten fertigen Swing Gui. Wir brauchen wie für jedes Java-Programm eine **main**-Methode.

```

public static void main(String[] args) {
    new ErstesFenster();
}

```

Dieser main Methode wird eine String-array variable mit dem Namen args übergeben. Hier wird noch ein String-Array übergeben. Da wir aber nicht erwarten, dass der Benutzer beim Starten unserer Applikation irgendwelche Argumente übergibt können wir das String-Array getrost ignorieren.

Das Entscheidende ist der **Konstruktor-Aufruf** in der Main Methode.

Wenn wir alles zusammenflicken sieht unsere Datei so aus:

```

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


public class ErstesFenster extends JFrame {
    private JLabel lb_InfoText;
    private JTextArea ta_TextAusgabe;
    private JScrollPane sp_ta_TextAusgabe;
    private JButton bt_Start;

    public ErstesFenster () {
        super("Erstes Programm");
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        setSize(300, 300);
        setLocation(100, 50);
        Container cp = getContentPane();
        cp.setLayout(null);

        lb_InfoText = new JLabel("Drücke auf den Button");
        lb_InfoText.setBounds(50, 10, 200, 16);
        cp.add(lb_InfoText);

        sp_ta_TextAusgabe = new JScrollPane();
        ta_TextAusgabe = new JTextArea();
        sp_ta_TextAusgabe.setBounds(50, 50, 200, 150);
    }
}

```



```
ta_TextAusgabe.setText("Dieser Text ist beim Start des Programmes bereits zu  
sehen");  
sp_ta_TextAusgabe.setViewportView(ta_TextAusgabe);  
cp.add(sp_ta_TextAusgabe);  
  
bt_Start = new JButton();  
bt_Start.setBounds(104, 240, 97, 25);  
bt_Start.setText("Klick Mich");  
cp.add(bt_Start);  
ActionListener meinNeuerActionListener = new ActionListener() {  
    public void actionPerformed(ActionEvent evt) {  
        ta_TextAusgabe.setText("Der Button funktioniert anscheinend.");  
    }  
};  
bt_Start.addActionListener(meinNeuerActionListener);  
  
setResizable(false);  
setVisible(true);  
}  
  
public static void main(String[] args) {  
    new ErstesFenster();  
}  
}
```