Kapitel 2: Einfache Verarbeitung

Jetzt gibt es sicher den ein oder anderen, der es etwas stupide findet auf einen Button zu drücken um „Der Button funktioniert anscheinend.“ angezeigt zu bekommen. Deshalb wollen wir jetzt noch ein paar neue Komponenten kennenlernen, um das GUI interessanter zu gestalten.

Der Plan sieht aus wie folgt:

Wir verwenden unsere alte Datei ErstesFenster. Wir bauen ein Popup Fenster und ein TextFeld ein. Außerdem implementieren wir einen weiteren Button.

Beginnen wir also mit der Implementierung:

das Textfeld platziere ich unter das TextArea. D. h., dass der Button erst einmal ein wenig nach unten wandern muss. ich erhöhe also den zweiten Wert bei der setBounds Methode des Buttons in unserem alten Projekt um 20. Jetzt habe ich genug Platz um ein TextFeld zwischen TextArea und Button zu platzieren.

Als erstes tragen wir das TextFeld bei den Attributen ein. Dafür erstelle ich eine neue Leerzeile unter den Attributen und trage folgendes ein:

```
    private JLabel lb_InfoText;
    private JTextArea ta_TextAusgabe;
    private JScrollPane sp_ta_TextAusgabe;
    private JButton bt_Start;
    private JTextField tf_Eingabe;   //Diese Zeile kommt neu hinzu
```

Jetzt muss ich Position und Größe bestimmen und es dann zu cp hinzufügen (im Konstruktor natürlich).

```
    tf_Eingabe = new JTextField(„Bitte gib ‚Hallo’ ein“);
    tf_Eingabe.setBounds(50, 210, 200, 18);
    cp.add(tf_Eingabe);
```

Jetzt fügen wir noch einen Button hinzu. Da wir das schonmal ausführlich gemacht haben, sei an dieser Stelle nur gesagt, dass in meinem Code der Button **bt_Fortfahren** heißt.

Hier noch meine Bounds:

bt_Start: (20, 240, 120, 25)
bt_Fortfahren: (160, 240, 120, 25)

Was soll jetzt passieren? Nun, der Benutzer soll in das TextField **Hallo** eingeben. Drückt er anschließend auf bt_Start, so soll geprüft werden, ob **Hallo** und kein anderes Wort eingegeben wurde. Ist dies der Fall, so öffnet sich ein Popup und fragt, ob man den zweiten Button aktivieren möchte. Man kann darauf mit **ja, nein** und **abbrechen** antworten.

Zunächst einmal ist es wichtig, dass man bt_Fortfahren nicht am Anfang drücken kann. Deshalb fügen wir im Konstruktor ein:

```
    bt_Fortfahren.setEnabled(false);
```

sinnvoll ist es auch, wenn man das TextArea uneditierbar macht, da es ja lediglich Informationen ausgeben soll (auch im Konstruktor):

```
    ta_Ausgabe.setEditable(false);
```

Jetzt entfernen wir **ta_TextAusgabe.setText(„Der Button funktioniert anscheinend.“);** da wir es nicht mehr brauchen. Die Überprüfung der Eingabe schreiben wir auch nicht im Konstruktor auf, da dieser sonst zu unübersichtlich wird. Anstattdessen lagern wir das ganze in eine Methode aus. Wir rufen im Konstruktor also nur noch die Methode auf:

```
ActionListener meinNeuerActionListener = new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        pruefeEingabe();
    }
};
```

Die Methode habe ich hier **pruefeEingabe()** genannt. Diese müssen wir jetzt noch in der Klasse definieren. Dafür schreiben wir unter den Konstruktor folgendes:

```
private void pruefeEingabe() {
    //... hier folgt dann die Überprüfung
}
```

private:	Keiner außerhalb diese Klasse muss auf diese Überprüfen-Methode zugreifen.
void:	Unsere Methode gibt nichts an die Methode **actionPerformed** (aus der heraus sie aufgerufen wurde) zurück, deshalb: **void**.
pruefeEingabe:	Name der Methode
():	Leere Klammern, da unserer Methode keine Werte aus der Methode **actionPerformed** übergeben („mitgegeben“) werden.
{//...}:	Wie beim Konstruktor wird der Methoden-Rumpf in geschweifte Klammern eingeschlossen.

Jetzt die Überprüfung.

*Falls im TextField **Hallo** steht, Öffne Popup und verarbeite die Auswahl.*
Sonst gib eine Fehlermeldung im TextArea aus.

In Java:

```
private void pruefeEingabe() {
    if(tf_Eingabe.getText().equals("Hallo")) {
        int iAuswahl = JOptionPane.showConfirmDialog(this, "Möchten sie den "
                                +"zweiten Button aktivieren?", "Frage!",
                                JOptionPane.YES_NO_CANCEL_OPTION);
        if(iAuswahl == 0) {
            bt_Start.setEnabled(false);
            tf_Eingabe.setEnabled(false);
            ta_TextAusgabe.append("Sie können jetzt auf den zweiten Button"
                                    +"klicken\n");
            bt_Fortfahren.setEnabled(true);
        } else if(iAuswahl == 1) {
            tf_Eingabe.setText("Bitte gib 'Hallo' ein");
        }
    } else {
        ta_TextAusgabe.append("\nDie Eingabe stimmt nicht! Bitte 'Hallo' eingeben");
    }
}
```

Das sieht jetzt erst mal mehr als kompliziert aus. Aber das liegt vorallem an den vielen langen Strings die den Code etwas unübersichtlich machen. Gehen wir das ganze mal der Reihe nach durch:
tf_Eingabe ist unser TextField. **getText()** ist eine Methode vom TextField, die den aktuellen Inhalt des TextField in Form eines String zurückgibt. Von diesem String rufen wir jetzt die equals- Methode auf(eine Methode der String-Klasse). Der Methode übergeben wir das Wort "Hallo". Unser String aus dem TextField (ein String-Objekt) vergleicht also seinen Wert mit dem übergebenen "Hallo" mithilfe seiner Methode equals. Stimmen beide Werte überein, gibt die Methode **true** zurück. Sonst **false**.
tf_Eingabe.getText().equals("Hallo") gibt also **true** oder **false** aus.
Damit in unserer If-Abfrage die Befehle in der geschweiften Klammer ausgeführt werden, muss in der runden Klammer ein **true** "stehen", der Wert im TextField also "Hallo" sein.
Gehen wir jetzt zunächst davon aus, dass **NICHT** "Hallo" im TextFeld steht und somit auch false bei der Überprüfung rauskommt. Jetzt werden die Befehle in der geschweiften Klammer **NICHT** ausgeführt. Anstattdessen werden jetzt die Befehle des **else**-Statement ausgeführt, da im **else**-Statement immer steht, was getan werden soll, falls die Aussage der If-Abfrage falsch bzw. unwahr ist.

```
ta_TextAusgabe.append("\nDie Eingabe stimmt nicht! Bitte 'Hallo' eingeben");
```

Append ist eine Methode, die uns unser TextArea zur Verfügung stellt. Mit ihr können wir Text an das Ende des bereits im TextArea stehenden Textes anfügen. "\n" ist ein Zeilenumbruch um das Ganze übersichtlicher zu gestalten. Es wird also, sollte nicht "Hallo" im TextFeld stehen, "Die Eingabe stimmt nicht! Bitte 'Hallo' eingeben" im TextArea erscheinen.

```
private void pruefeEingabe() {
    if(tf_Eingabe.getText().equals("Hallo")) {

    } else {
        ta_TextAusgabe.append("\nDie Eingabe stimmt nicht! Bitte 'Hallo' eingeben");
    }
}
```

Bis jetzt ist das ganze noch übersichtlich. Jetzt müssen wir die Lücke unter dem if füllen. Den Teil, der beschreibt, was passiert, wenn **Hallo** im TextFeld steht. Gehen wir auch hier Schritt für Schritt durch.

```
int iAuswahl = JOptionPane.showConfirmDialog(this, "Möchten sie den "
                    +"zweiten Button aktivieren?", "Frage!",
                    JOptionPane.YES_NO_CANCEL_OPTION);
```

Das Ganze fängt an mit der Definition einer lokalen Variablen. Eine Lokale Variable existiert nur, innerhalb der geschweiften Klammern, die sie umgrenzen. So kann der Konstruktor nicht auf iAuswahl zugreifen, da iAuswahl nur im "If-Teil" der Methode pruefeEingabe existiert. Der Typ int sollte bekannt sein (ganze Zahlen). Der Name verrät schon, was für einen Wert die Variable erhalten soll: Die Auswahl, die im PopUp-Fenster getroffen werden soll.
Genau diese Information erhalten wir auf der rechten Seite des =. Am meisten wundert einen wahrscheinlich die Tatsache, dass wir auf die Methode **showConfirmDialog** von **JOptionPane** zugreifen, obwohl in der gesamten Klasse noch nie die Rede von einem Attribut **JOptionPane** war. Hierzu muss man die JavaDocs zur Hilfe nehmen. Dort steht

nämlich über die Methode **showConfirmDialog**, dass diese **static** ist. **static** bedeutet nichts anderes, als dass diese Methode jederzeit ohne den Konstruktor von **JOptionPane** aufzurufen zur Verfügung steht. Wir müssen also nicht mit dem **new** Befehl, den wir auch schon kennengelernt haben arbeiten, sondern greifen sofort auf die Methode **showConfirmDialog** zu. Diese Methode will von uns folgende Informationen:

parentComponent:	Der Frame, aus dem das PopUp-Fenster aufgerufen wurde. (dient u.a. zur Positionierung des PopUps. **this** ist ein Aufruf von unserem Fenster auf sich selbst. Es übergibt also sich selbst als Aufrufenden Frame. Man kann auch **null** übergeben.
message	Hier wird das übergeben, was nachher als Information im PopUp stehen soll. Man kann hier einen String, aber auch komplexe Objekte wie eine Tabelle bspws. übergeben.
title	Dieser String steht nachher in der Titelleiste des PopUp.
optionType	Dieser int bestimmt, welche Button-Konfiguration angezeigt werden soll. Da man sich solche Zahlen schlecht merken kann, stellt Sun diese als Konstanten der Klasse JOptionPane (man achte auf die Großbuchstaben) zur Verfügung. Wir wählen YES_NO_CANCEL_OPTION um drei Buttons mit "Ja", "Nein" und "Abbrechen" zu erhalten.

Diese Informationen übergeben wir :

- this
- "Möchten sie den zweiten Button aktivieren?"
- "Frage!"
- JOptionPane.YES_NO_CANCEL_OPTION

Die Methode wartet so lange, bis der Benutzer eine Eingabe gemacht hat. Dann gibt es einen Integer zurück:
Bei "Ja": 0
Bei "Nein": 1
Bei "Abbrechen": 3
Dieser Wert wird dann, da die Methode hinter dem = steht automatisch in unsere int-Variable eingetragen. Jetzt muss dieser Wert noch ausgewertet werden:

```
if(iAuswahl == 0) {
        bt_Start.setEnabled(false);
         tf_Eingabe.setEnabled(false);
         ta_TextAusgabe.append("Sie können jetzt auf den zweiten Button"
                              +"klicken\n");
         bt_Fortfahren.setEnabled(true);

} else if(iAuswahl == 1) {
         tf_Eingabe.setText("Bitte gib 'Hallo' ein");
}
```

Ist die Antwort "Ja" so wird alles gemacht, was zwischen dem ersten Klammerpaar steht. Ist die Antwort nicht "Ja" sondern "Nein" so wird alles gemacht, was zwischen dem zweiten Klammerpaar steht. Ist die Antwort "Abbrechen" so geschieht nichts. Das **else if** bewirkt, dass diese Abfrage nur gemacht wird, falls die erste If-Abfrage ein unwahres Ergebnis hatte. So

spart man sich unnötige Abfragen. Wenn iAuswahl 0 ist, kann die Variable nicht auch gleichzeitig 1 sein. Das Ganze muss jetzt in die Lücke nach

```
    if(tf_Eingabe.getText().equals("Hallo")) {
```

Jetzt brauchen wir noch eine Aktion für unseren zweiten Button. Wir können hier z.B. wieder einen Text im TextArea anzeigen lassen. Das ist zwar nichts neues, aber dafür sollte man das jetzt schon eigenständig hinbekommen. Man kann sich ja an Kapitel 1 notfalls orientieren.

Was haben wir jetzt erreicht:

Wir können jetzt JTextAreas, JButtons, JOptionPanes, JTextFields, Labels und JFramess erstellen. Außerdem können wir Eingaben per IF-abfrage auswerten und das Programm entsprechend reagieren lassen. Natürlich ist noch wesentlich mehr mit den Komponenten möglich und es gibt auch noch wesentlich mehr Komponenten. Aber für Kapitel 2 soll dies erstmal genügen.