IX: Grafische Benutzerschnittstellen

- Eine grafische Benutzerschnittstelle (GUI) gestattet es, Eingaben durch Schaltknöpfe, Menüs, Schiebeschalter etc. mausgesteuert zu tätigen und Ausgaben in Fenstern zu präsentieren.
- Java stellt eine große Bibliothek zur Gestaltung grafischer Benutzeroberflächen (GUIs) zur Verfügung. Name der Bibliothek: Swing.
- Swing und alle anderen solchen Bibliotheken bauen sehr stark auf Vererbung auf.

Anwendungsbeispiel: Bankkonten

Wir möchten ein klickbares Fenster der folgenden Art.

Frames

Die Klasse `JFrame` in `javax.swing` stellt ein “nacktes” Fenster zur Verfügung:

In `BankkontoGUI.java`:

```
import javax.swing.JFrame;
public class BankkontoGUI extends JFrame {
    public BankkontoGUI(String name) {
    this.setTitle(name);
    this.setSize(400, 200);
    }
}
```

Frames

In `Main.java`:

```
public class Main {
    public static void main(String[] args) {
        BankkontoGUI gui = new BankkontoGUI("Bankkonto");
        gui.setVisible(true);
    }
}
```

Textfelder

Das Bankkonto-GUI soll zwei Textfelder enthalten: In `BankkontoGUI`

```
...
    private JTextField kontostandFeld;
    private JTextField betragFeld;
    public BankkontoGUI() {
      ...
        kontostandFeld = new JTextField(20);
        kontostandFeld.setEditable(false);
        betragFeld = new JTextField(20);
        betragFeld.setEditable(true);
      ...
```

Textfelder

Jetzt gibt es die Textfelder, aber sie sind nicht sichtbar!

Jeder JFrame hat einen “Inhalt”, die *content pane*, in der man alle möglichen Komponenten unterbringen kann:

Man kann Komponenten entweder direkt in die *content pane* setzen mit der Methode `add`, z.B.

```
Container fensterInhalt = this.getContentPane();
fensterInhalt.add(kontostandFeld,"South");
```

Es gibt die Positionen `South, North, East, West, Center`.

Man spricht von einem Border Layout.

Gitter Layout

Will man die Komponenten nach Art einer Tabelle in Zeilen und Spalten einsetzen, so verwendet man ein Gitterlayout.

```
contentPane.setLayout(new GridLayout(3,1));
```

Hier 3 Zeilen und 1 Spalte. Die Komponenten werden jetzt mit `add` zeilenweise eingehängt.

Tafeln

Mehrere Komponenten kann man in eine Tafel (JPanel) gruppieren.

Man hängt Komponenten in eine Tafel mit `add` ein.

Die Tafel selbst bildet wiederum eine Komponente und wird mit `add` in die *content pane* oder in eine andere Tafel eingehängt.

Tafeln haben per Default ein Flusslayout:

Flusslayout

Beim Flusslayout werden die Komponenten der Reihe nach eingehängt und in dieser Ordnung arrangiert.

Man kann das Flusslayout auch bei der *content pane* verwenden:

```
contentPane.setLayout(new FlowLayout());
```

Oder aber ein Gitterlayout in eine Tafel:

```
myPanel.setLayout(new GridLayout(10,30));
```

Oder auch das Border Layout:

```
myPanel.setLayout(new BorderLayout());
```

Etiketten

Außerdem fehlen unseren Textfeldern noch die Beschriftungen.

Diese werden als Objekte der Klasse `JLabel` (Etiketten) behandelt.

Wir werden also ein JLabel des Namens `Kontostand:` erzeugen

und es zusammen mit dem JTextField `kontostandFeld` in ein JPanel packen, welches wir in die erste Zeile der *content pane* setzen.

```
JPanel kontostandTafel = new JPanel();
JLabel kontostandEtikett = new JLabel("Kontostand:")
kontostandTafel.add(kontostandLabel);
kontostandTafel.add(kontostandFeld);
fensterInhalt.add(kontostandTafel);
```

Etiketten

Das Gleiche machen wir mit dem Betragsfeld:

```
JPanel betragTafel = new JPanel();
JLabel betragEtikett = new JLabel("Betrag: ");
betragTafel.add(betragLabel);
betragTafel.add(betragFeld);
fensterInhalt.add(betragTafel);
```

da `kontostandTafel` schon da ist, kommt `betragTafel` in die zweite Zeile.

Eigentlich käme es in die zweite Spalte der ersten Zeile, aber da wir nur eine Spalte haben, wird gleich die zweite Zeile angefangen.

Knöpfe

Die Klasse `JButton` stellt Knöpfe bereit: wir deklarieren sie als Instanzvariablen:

```
private JButton einzahlKnopf;
private JButton abhebeKnopf;
```

und besetzen sie in `BankkontoGUI()`:

```
einzahlKnopf = new JButton("einzahlen");
abhebeKnopf = new JButton("abheben");
```

Knöpfe

Schließlich stecken wir sie wiederum in eine Tafel und letzteres in die dritte Zeile der *content pane*.

```
JPanel knopfTafel = new JPanel();
knopfTafel.add(einzahlen);
knopfTafel.add(abheben);
fensterInhalt.add(buttonTafel);
```

Gesamtprogramm bis jetzt

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

public class BankkontoGUI extends JFrame {

    private JTextField kontostandFeld;
    private JTextField betragFeld;
    private JButton einzahlKnopf;
    private JButton abhebeKnopf;

    public BankkontoGUI(String name) {
        Container fensterInhalt = this.getContentPane();
        this.setSize(400,200);
        this.setTitle(name);

        kontostandFeld = new JTextField(20);
        kontostandFeld.setEditable(false);
```

Gesamtprogramm bis jetzt

```
JLabel kontostandEtikett = new JLabel("Kontostand:");
JPanel kontostandTafel = new JPanel();
kontostandTafel.add(kontostandEtikett);
kontostandTafel.add(kontostandFeld);
betragFeld = new JTextField(20);
betragFeld.setEditable(true);
JLabel betragEtikett = new JLabel("Betrag:");
JPanel betragTafel = new JPanel();
betragTafel.add(betragEtikett);
betragTafel.add(betragFeld);

einzahlKnopf = new JButton("einzahlen");
abhebeKnopf = new JButton("abheben");

JPanel knopfTafel = new JPanel();
knopfTafel.add(einzahlKnopf);
knopfTafel.add(abhebeKnopf);
```

Gesamtprogramm bis jetzt

```
        fensterInhalt.setLayout(new GridLayout(3,1));
        fensterInhalt.add(kontostandTafel);

        fensterInhalt.add(betragTafel);
        fensterInhalt.add(knopfTafel);

        setVisible(true);
    }
}
```

Aktionen und Ereignisse

Nun müssen wir unser GUI mit Leben füllen.

Den Inhalt der Textfelder kann man mit `getText` und `setText` auslesen und verändern.

Wie reagieren wir auf das Drücken der Knöpfe?

Früher…

…gab es zu jedem Knopf eine Methode (oder Funktion) mit Boole'schem Rückgabewert.

War gerade ein Knopf gedrückt, so war der Rückgabewert `true`; ansonsten `false`.

Im Hauptprogramm musste man dan ständig diese Methode aufrufen um ja keinen Knopfdruck zu verpassen.

Diese Methode, bezeichnet als *polling*, gilt inzwischen als überholt. Stattdessen benutzt man ereignisgesteuerte Eingabenbehandlung (*event-driven*):

Aktionen und Ereignisse

Mit der Methode `addActionListener` kann man an einen Knopf einen *action listener* (Zuhörer) anheften.

Das ist ein Objekt, welches das Interface `ActionListener` implementiert.

Dieses Interface enthält nur eine einzige Methode:

```
public void actionPerformed(ActionEvent e)
```

Der *action listener* muss also so eine Methode implementieren.

Ist der *action listener* an einen Knopf geheftet, so wird diese Methode immer dann aufgerufen, wenn der Knopf gedrückt wird.

Und zwar mit einem Parameter der Klasse `ActionEvent`, aus dem man im Rumpf von `actionPerformed` z.B. den Knopf, der gedrückt wurde, ablesen kann (die “Quelle des Ereignisses”).

Beispiel

```
public class Zuhoerer implements ActionListener {
    public void actionPerformed(ActionEvent ereignis) {
        System.out.println("Es wurde " + ereignis.getSource(
                                + " gedrueckt.");
    }
}
```

und in `BankkontoGUI()`:

```
        ActionListener zuhoerer = new Zuhoerer();
        einzahlKnopf.addActionListener(zuhoerer);
        abhebeKnopf.addActionListener(zuhoerer);
```

Ausgabe

```
Es wurde javax.swing.JButton[...,text=einzahlen,...] gedruec
Es wurde javax.swing.JButton[...,text=abheben,...] gedrueckt
```

Sinvollere Listener

Für sinnvollere Aktionen brauchen wir

- Eine Instanzvariable `konto` der Klasse `Bankkonto`
- Innerhalb von `actionPerformed` Zugriff auf `konto`, sowie auf die Textfelder und Knöpfe.

Übergabe der GUI-Komponenten per Konstruktor

Wir können alle diese Komponenten auch als Instanzvariablen des Zuhoerers führen und über den Konstruktor übergeben.

Z.B.: in `Zuhoerer.java`

```
private Bankkonto konto;

public Zuhoerer(Bankkonto konto, JButton abhebeKnopf, ...) {
   this.konto = konto;
   ...
}
```

Vorteil: Anschaulich und im Einklang mit bisherigem Stoff

Nachteil: Sehr umständlich, wenn viele Komponenten übergeben werden müssen. Schon in diesem Beispiel sind es ja fünf! (Konto, zwei Textfelder, zwei Knöpfe)

Innere Klassen

Alternativ hat man die Möglichkeit, die Definition der Klasse `Zuhoerer` innerhalb von `BankkontoGUI {...}` vorzunehmen.

Hierbei verwendet man ein neues Java Sprachkonstrukt: Innere Klassen.

Eine innere Klasse hat auch Zugriff auf die privaten Instanzvariablen ihrer umgebenden Klasse und verhält sich ansonsten wie eine normale Klasse.

Im Computer realisiert werden innere Klassen wie Möglichkeit 1, also explizite Übergabe der Instanzvariablen bei Konstruktion.

Vorteil: Wird oft benutzt, steht so im Buch.

Nachteil: Theorie etwas unklar (Sichtbarkeitsbereiche, etc.), Aufblähung der Sprache.

Beispiel

```
public class BankkontoGUI extends JFrame {
    private Bankkonto konto;
    ...
    public BankkontoGUI(Bankkonto konto) {
        this.konto = konto;
        ...
        ActionListener zuhoerer = new Zuhoerer();
        einzahlKnopf.addActionListener(zuhoerer);
        abhebeKnopf.addActionListener(zuhoerer);
        kontostandFeld.setText("" + konto.getKontostand());
        ...
    }
    class Zuhoerer implements ActionListener {
        public void actionPerformed(ActionEvent ereignis) {
     /*Fortsetzung folgt*/
```

Beispiel

```
            Object gedrueckt = ereignis.getSource();
            double betrag = Double.parseDouble(betragFeld.getText());
            if (gedrueckt == einzahlen)
                konto.einzahlen(betrag);
            else if (gedrueckt == abheben)
                konto.abheben(betrag);
            else
                ;
        }
} /* Ende der Klasse BankkontoGUI */
```

Achtung: == ist hier die physikalische Gleichheit von Objekt(referenz)en.

Beobachter

Wir haben das Problem, die Kontostandanzeige mit dem tatsächlichen Kontostand konsistent zu halten.

Jede Änderung des Kontostandes (auch durch andere GUIs, das Hauptprogramm, o.ä.) sollen sofort wiedergegeben werden.

Die Lösung besteht im Entwurfsmuster *Beobachter* (*Observer*).

Jedes Bankkonto verwaltet eine Liste von Beobachtern, die sich bei ihm registriert haben. Ändert sich der Kontostand, so ruft es bei jedem registrierten Beobachter eine bestimmte Methode auf.

Konkretes Beispiel

Wir verwenden die Schnittstelle

```
public interface BankkontoBeobachter {
    public void anzeigen(Bankkonto b);
}
```

Die Klasse Bankkonto muss nun so erweitert werden, dass eine Liste von BankkontoBeobachtern verwaltet wird und die Methode anzeigen entsprechend aufgerufen wird:

Konkretes Beispiel

```
public class Bankkonto {
    private double kontostand;
    private ArrayList<BankkontoBeobachter> beobachter;
    public void benachrichtigen() {
        for (BankkontoBeobachter beob : beobachter)
            beob.anzeigen(this);
    }
    public void einzahlen(double betrag) {
        kontostand = kontostand + betrag;
        benachrichtigen();
    }
    public void abheben(double betrag) {...}
    public void anheften(BankkontoBeobachter beob) {
        beobachter.add(beob);
    }
}
```

Konkretes Beispiel

Die Klasse `BankkontoGUI` implementiert nun `BankkontoBeobachter`.

```
public class BankkontoGUI extends JFrame
                    implements BankkontoBeobachter {
    ...

    public void anzeigen(Bankkonto k) {
        kontostandFeld.setText("" +
                        konto.getKontostand());
    }
```

Observer und Observable

Die Beobachterverwaltung ist in Java auch vorgefertigt in Form einer Klasse `Observable` von der beobachtbare Klassen, z.B. `Bankkonto` dann erben koennen ...

und einer Schnittstelle `Observer`, die die Beobachter, z.B. `BankkontoGUI` implementieren müssen. Diese Schnittstelle enthält eine Methode

```
void update(Observable o,
            Object arg)
```

Die Klasse `Observable` stellt Methoden `addObserver` (entspricht unserem `anheften`) und `notifyObserver` (entspricht unserem `benachrichtigen`) bereit.

Zusammenfassung GUI

- Die Bibliothek Swing stellt GUI Komponenten bereit.
- Mit Ereignissen (Event) und Zuhörern (Listener) wird auf Benutzereingaben reagiert. Letztendlich wird bei Benutzereingaben eine vom Programmierer für diesen Zweck bereitgestellte Methode aufgerufen.
- Innere Klassen erlauben die komfortable Implementierung der erforderlichen Zuhörer-Klassen.
- Die GUI kann durch Verwendung des Beobachter-Musters mit den dargestellten Daten konsistent gehalten werden. Vgl. MVC Muster.