Packages

Man kann mehrere Klassen in eine *package* zusammenfassen.

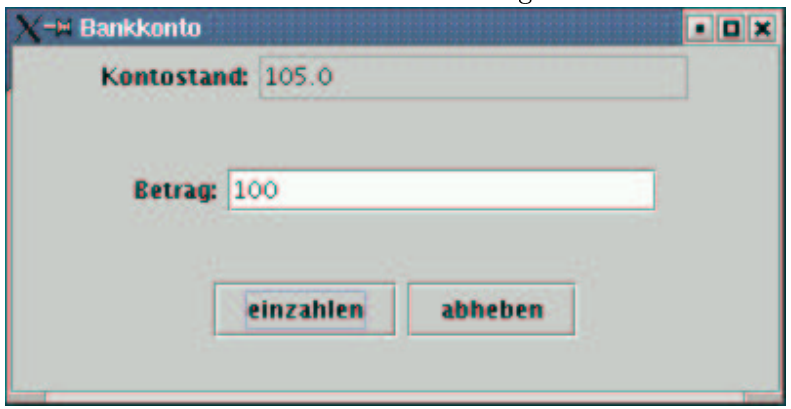
- Diese müssen dann in einem Unterverzeichnis liegen, dessen Name der Packagename ist.
- Jede Datei der Package muss mit `package <name>;` beginnen, wobei `<name>` der Packagename ist.
- Auf Klassen der Package greift man durch Vorschalten von `<name>.` zu.
- Alternativ kann man `import <name>.*;` verwenden.
- Ist eine Klasse oder Methode oder Instanzvariable weder `public` noch `private`, so ist sie nur innerhalb ihrer Package sichtbar.
- Beispiel: Alle Bankkonten der letzten Vorlesung kommen in eine Package `bankkonto`.

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 java.awt.*
import java.awt.event.*
import javax.swing.*;
public class BankkontoGUI extends JFrame {
    public BankkontoGUI() {
    this.setTitle("Bankkonto");
    this.setSize(400, 200);
    }
}
```

Frames

In `Main.java`:

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


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 eine "Inhaltsscheibe", 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 contentPane = this.getContentPane();
contentPane.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.

Paneele

Mehrere Komponenten kann man in ein Paneel (JPanel) gruppieren.

Man hängt Komponenten in ein Paneel mit `add` ein.

Das Paneel selbst bildet wiederum eine Komponente und wird mit `add` in die content Pane oder in ein anderes Paneel eingehängt.

Paneel 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 einem Paneel:

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

Oder auch das Border Layout:

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


Labels

Außerdem fehlen unseren Textfeldern noch die Beschriftungen.

Diese werden als Objekte der Klasse `JLabel` 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 kontostandPanel = new JPanel();
JLabel kontostandLabel = new JLabel("Kontostand: ");
kontostandPanel.add(kontostandLabel);
kontostandPanel.add(kontostandFeld);
contentPane.add(kontostandPanel);
```


Labels

Das gleiche machen wir mit dem Betragsfeld:

```
JPanel betragPanel = new JPanel();
JLabel betragLabel = new JLabel("Betrag: ");
betragPanel.add(betragLabel);
betragPanel.add(betragFeld);
contentPane.add(betragPanel);
```

da `kontostandPanel` schon da ist, kommt `betragPanel` 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 einzahlen;
private JButton abheben;
```

und besetzen sie in `BankkontoGUI()`:

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


Knöpfe

Schließlich stecken wir sie wiederum in ein Paneel und letzteres in die dritte Zeile der content pane.

```
JPanel buttonPanel = new JPanel();
buttonPanel.add(einzahlen);
buttonPanel.add(abheben);
contentPane.add(buttonPanel);
```


Gesamtprogramm bis jetzt

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

public class BankkontoGUI extends JFrame {
    private JTextField kontostandFeld;
    private JTextField betragFeld;
    private JButton einzahlen;
    private JButton abheben;

    public BankkontoGUI(Bankkonto konto) {
        this.setSize(400,200);
        this.setTitle("Bankkonto");
        Container contentPane = this.getContentPane();
        contentPane.setLayout(new GridLayout(3,1));
```


Gesamtprogramm bis jetzt

```
kontostandFeld = new JTextField(20);
kontostandFeld.setEditable(false);
JLabel kontostandLabel = new JLabel("Kontostand:");
JPanel kontostandPanel = new JPanel();
kontostandPanel.add(kontostandLabel);
kontostandPanel.add(kontostandFeld);
contentPane.add(kontostandPanel);

betragFeld = new JTextField(20);
betragFeld.setEditable(true);
JLabel betragLabel = new JLabel("Betrag:");
JPanel betragPanel = new JPanel();
betragPanel.add(betragLabel);
betragPanel.add(betragFeld);
contentPane.add(betragPanel);
```

Gesamtprogramm bis jetzt

```
        einzahlen = new JButton("einzahlen");
        abheben = new JButton("abheben");
        JPanel buttonPanel = new JPanel();
        buttonPanel.add(einzahlen);
        buttonPanel.add(abheben);
        contentPane.add(buttonPanel);
    }
}
```


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* (Aktionslauscher) 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 MyListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Es wurde " + e.getSource()
                           + " gedrueckt.");
    }
}
```

und in `BankkontoGUI()`:

```
MyListener listener = new MyListener();
einzahlen.addActionListener(listener);
abheben.addActionListener(listener);
```

Ausgabe

```
Es wurde javax.swing.JButton[...,text=einzahlen,...] gedrueckt.
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 listeners führen und über den Konstruktor übergeben.

Z.B.: in `MyListener.java`

```
private Bankkonto konto;

MyListener(Bankkonto konto) {
   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)

Die GUI selbst als Listener

Wie man weiss, kann jede Klasse jedes Interface implementieren, solange nur die verlangten Methoden präsent sind.

Man kann also die Methode `actionPerformed` in die GUI selbst hineinnehmen, im Beispiel also in `BankkontoGUI` und deklarieren

```
implements ActionListener
```

man hat dann automatisch Zugriff auf alle Instanzvariablen.

Die Installation des Listeners erfolgt dann mit `addActionListener(this)`.

Vorteil: Einfacher

Nachteil: Etwas gewöhnungsbedürftig, umständlicher bei Mausereignissen (nicht in dieser VL).

Innere Klassen

Schließlich hat man die Möglichkeit, die Definition der Klasse `MyListener` 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

Wir verwenden Möglichkeit 2 und schreiben daher

```
public class BankkontoGUI extends JFrame
                  implements ActionListener{
    private Bankkonto konto;
    ...
    public BankkontoGUI(Bankkonto konto) {
        this.konto = konto;
        ...
        einzahlen.addActionListener(this);
        abheben.addActionListener(this);
        kontostandFeld.setText("" + konto.getKontostand());
        ...
    }
    public void actionPerformed(ActionEvent event) {
     /*Fortsetzung folgt*/
```

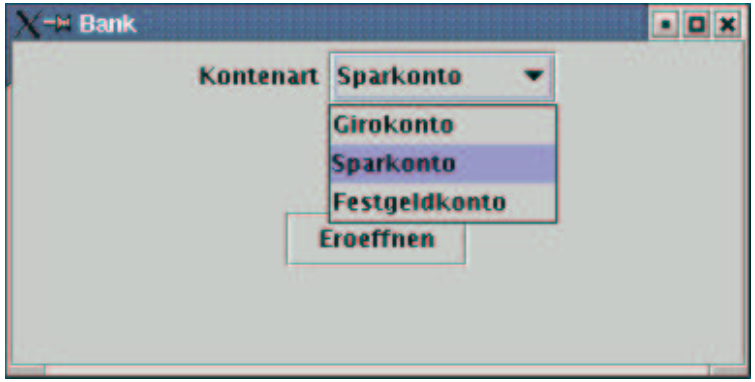

Beispiel

```
        Object source = event.getSource();
        double betrag = Double.parseDouble(betragFeld.getText());
        if (source == einzahlen)
            konto.einzahlen(betrag);
        else if (source == abheben)
            konto.abheben(betrag);
        else
            ;
        kontostandFeld.setText("" + konto.getKontostand());
    }
```

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

Die Bank

Jetzt möchten wir interaktiv Konten erzeugen:



Combobox

Das Auswahlmenü hinter “Kontenart” ist ein Combobox (`JComboBox`).

Mit `new JComboBox()` wird sie erzeugt.

Mit `addItem` lassen sich Auswahlobjekte einfügen,

durch Aufruf von `toString` werden die Auswahlobjekte dann angezeigt.

Also z.B.: `myCombo.addItem("Festgeldkonto");`

Mit `getSelectedItem()` kriegt man das gewählte Objekt zurück.

Beispiel

```
public class BankGUI extends JFrame
            implements ActionListener {
    private Bankkonto[] konten;
    private int anzKonten;
    private JComboBox chooserCombo;

    public BankGUI() {
        konten = new Bankkonto[1000];
        anzKonten = 0;

        JPanel chooserPanel = new JPanel();
        JLabel chooserLabel = new JLabel("Kontenart");
        chooserCombo = new JComboBox();
        chooserCombo.addItem("Girokonto");
        chooserCombo.addItem("Sparkonto");
        chooserCombo.addItem("Festgeldkonto");
        chooserPanel.add(chooserLabel);
```


Beispiel

```
        chooserPanel.add(chooserCombo);

        JPanel openPanel = new JPanel();
        JButton openButton = new JButton("Eroeffnen");
        openPanel.add(openButton);
        openButton.addActionListener(this);

        contentPane.add(chooserPanel);
        contentPane.add(openPanel);

    }
    public void actionPerformed(ActionEvent event) {
        String selection = (String)chooserCombo.getSelectedItem();
        Bankkonto konto;

        if (selection.equals("Girokonto"))
            konto = new Girokonto(0.25);
```


Beispiel

```
        else if (selection.equals("Sparkonto"))
            konto = new Sparkonto(1.25);
        else if (selection.equals("Festgeldkonto"))
            konto = new Festgeldkonto(10, 200., 3.25);
        else
            konto = null;
        konten[anzKonten] = konto;
        anzKonten++;
        BankkontoGUI gui = new BankkontoGUI(konto);
        gui.show();
    }
}
```

Weitere Möglichkeiten

- Mit `setEnabled(false)` wird ein Knopf oder eine Combooption abgeschaltet. Sie erscheint dann grau und kann nicht gedrückt / gewählt werden (Sie ist "ausgegreyt").
- Eine Checkbox `JCheckBox` ist ein spezieller Knopf. Wurde er gedrückt, so wird er mit einem Häkchen ausgefüllt. Beim nächsten Drücken wird das Häkchen wieder entfernt. Mit `isSelected()` stellt man fest, ob eine Checkbox aktiv ist.
- Radiobuttons `JRadioButton` sind spezielle Checkboxen, die in eine Gruppe `ButtonGroup` zusammengefasst werden. Es kann dann stets nur einer der Radiobuttons aktiviert sein.

  Mit `add` fügt man die Radiobuttons in die Gruppe ein.

Übungen

- Man erweitere das GUI um Überweisungen.
- Man suche in der Swing Doku den `JSlider` auf (Schiebeschalter) und implementiere damit ein Feld, dessen Farbe sich durch kontinuierliche Wahl der Rot, Grün, Blau-Werte (mittels dreier JSlider) verändern lässt.