Hochschule Harz	FB Automatisierung und Informatik
3. Labor: Interface	Grafische Nutzerschnittstellen Thema: Vererbung, Interface und Java „Beans“

Versuchsziele

- Verwendung von abgeleiteten und zusammengesetzten GUI-Elementen
- Benutzung von Vererbung in GUI-Programmen
- Verwendung von Schnittstellen um Funktionen zu definieren

Aufgabenstellung:

Für einen Großhändler soll ein vereinfachter Rechnungsdialog entwickelt werden. Programmieren Sie ein Dialogfenster, das die beiden Klassenbäume „Fahrzeug“ und „Fahrrad“ zusammen in einer Angebotsliste (links) anzeigt. Rechts werden die gekauften Artikel in einer ListView dargestellt.

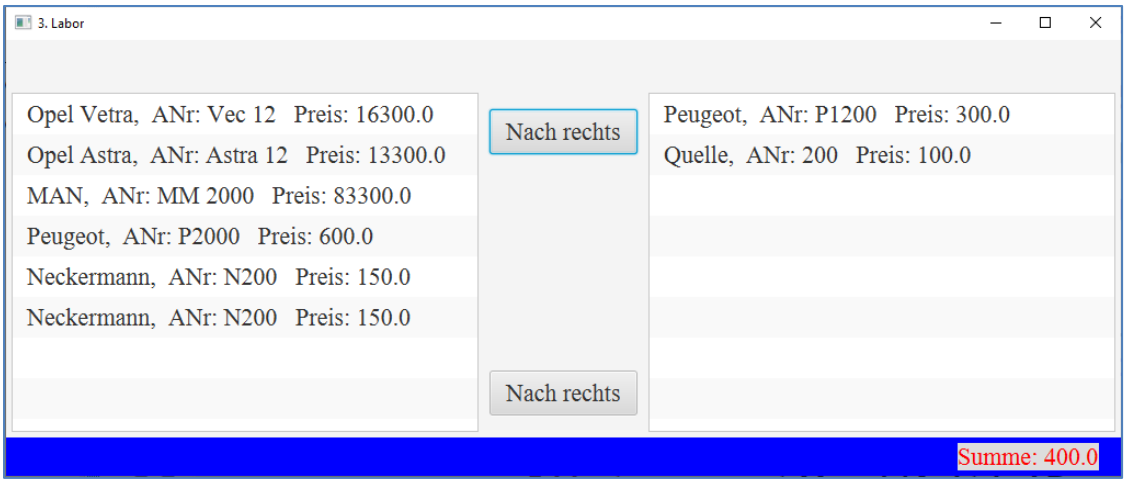

Abbildung der Musterlösung

Das Panel der Klasse „myArtikelPanel“ besteht aus den **beiden** ListView, den beiden Schalter und das JLabel

Erläuterung:

Der erste Schalter fügt die Artikel in das „Bean-Objekt“ ein. Das Bean-Objekt soll hier ein Pane resp. ein GridPane sein, welches mehrere GUI-Elemente beinhaltet. Diese werden in einer separaten Klasse zusammengestellt. Die Artikel werden mit Hilfe zweier interner Listen und zweier ListView verwaltet. Die beiden Schalter „Nach Rechts“ und „Nach Links“ stellen den Kauf bzw. die Rückgabe eines Artikels dar. Das Label zeigt die aktuelle Summe aller bestellten Artikel an.

Wichtig:
Um ein einheitliches Aussehen innerhalb verschiedener Programme einer Firma zu haben, werden die GUI-Elemente durch eine Klasse „**MyArtikelPanel**“ erzeugt. Das bedeutet, dass nur ein Panel in das Hauptframe eingefügt wird. Dies ist das Panel aus der Klasse MyArtikelPanel. Der Rest wird natürlich auch in dieser Klasse verwaltet. Dies entspricht „weitgehend“ dem Verfahren bei Benutzung eines Java Beans.

Wie stellt man sich nun dieses MyArtikelPanel vor?

In normalen Programmen werden alle grafischen Elemente „einzeln“ in ein Frame eingefügt. Danach muss man für alle diese Elemente die „Business-Logik“ einbauen. Für komplexe GUI-Strukturen ist es sinnvoller, dass man fertige Bausteine entwickelt.

Das objekt-orientierte Konzept wird nun auf die Entwicklung der grafischen Programmierung übertragen (**Beans**):

- Man entwickelt einen „Karton“, in dem alle grafischen Elemente „**intern**“ zusammengefasst werden.
- Die Elemente werden auf einer Pinnwand, ein Pane, eingetragen.
- Die Elemente sind von außen **nicht** sichtbar (für den Programmierer!).
- Man muss geeignete Schnittstellen, Methoden, in der Klasse hinzufügen, damit die Anwender auf die internen Elemente zugreifen können.
- Der Vorteil dieser Programmierung liegt in der Wiederverwendbarkeit. Bei normaler Technik müsste man die „Business-Logik“ mehrfach einbauen. So muss man nur die Beans-Komponente in ein Frame einfügen.

ListView:

- Eine ListView zeigt in einer Liste die vorhandenen Einträge an.
- Im Konstruktor können die Einträge mittels einer ObservableList, vergleichbar einer ArrayList, übergeben werden.
- Methoden einer ListView:
  - getSelectionModel().setSelectionMode(SelectionMode.SINGLE)
  - getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE)
  - setItems(*ObservableList*);
  - setPlaceholder(new Label("Bitte etwas auswählen"));
  - getSelectedIndex liefert den aktuellen Index (0..n-1) ODER -1
    - -1 zeugt an, dass kein Element in der ListView markiert ist.
  - ObservableList selListe = listview1.getSelectionModel().getSelectedItems();
  - Listview1.getSelectionModel().clearSelection();
  -
  - ContextMenu:
    - MenuItem mnItems2Right = new MenuItem("Nach Rechts");
    - mnItems2Right.setOnAction(e->bn2Right_Click());
    - contextmenu.getItems().add(mnItems2Right);
    - listview1.setOnContextMenuRequested(new EventHandler<ContextMenuEvent>() {
    - @Override
    - public void handle(ContextMenuEvent event) {
    - contextmenu.show(listviewArtikel, event.getScreenX(), event.getScreenY());
    - }
    - });
- Methoden einer ObservableList:
  - add(Object) Hinzufügen eines Objektes
  - addAll(Object1, Object2) Hinzufügen eines Objektes
  - addAll(ObservableList)
  - removeAll(ObservableList)

Einstiegsskript Interface:

- Ein Interface definiert abstrakte Methoden, die andere Klassen implementieren müssen.
- Es verhält sich ähnlich einer abstrakte Klasse. Nur kann man beliebig viele Interfaces implementieren und man darf keine Attribute definieren.
- Beispiel:

Interface-Definition:

```
interface IDrucken {
   public void print();              // „abstrakte“ Methode
}
```

Klasse hat ein Interface:

```
class Student extends  Person  implements IDrucken {
   public void print() {
      syso("Name: "+this.name);
   }
}

class JumboJet extends  Flugzeug  implements IDrucken {
   public void print() {
      syso("Flugzeugnr: "+this.flugzeugnummer);
   }
}
```

// Problem ist nun, dass es keine Oberklasse für die Speicherung der beiden Instanzen geben kann.
// Abhilfe: interface

Speicherung:

```
void test1() {
      Student std = new Student("Meier");
      Flugzeug fl = new Flugzeug("A380");
      // nun die Speicherung
      IDrucken[] feld = new IDrucken[2];
      feld[0] = std;
      feld[1] = fl;
         // nun kann man mit einer For-each-Schleife die Objekte ausgeben!!!
         // Dass der Datentyp fehlt, wurde erst nach den Drucken erkannt.  ;-)
      for(??????  drk  :  feld) {
         syso(drk.toString());
      }
}
```

Beispielcode:

```
class My ArtikelPanel {
   private JPanel myPanel = new JPanel();
   ListView ListViewLinks = new ListView();
   ListView ListViewRechts = new ListView();
   JLabel _SummenLabel1 = new JLabel("Summe");
   MyLabel _SummenLabel2 = new MyLabel(0.0);  // Neuer Konstruktor mit double-Wert

   public MyArtikelPanel() {
      setGUI();
   } //MyArtikelPanel()

  private void setGUI() {
    myPanel.setLayout(  new GridBagLayout()  );
       …
  }  // setGUI

}  // class myArtikelPanel
```

Alle Event-Methoden müssen in der Klasse definiert und abgefangen werden. Folgende öffentliche Methoden hat die Klasse **myArtikelPanel**:

- public ??? getPanel() {
- public void addArtikel(???);
- public void BnNachRechts_click();
- public void calcSumme();

Diese Klasse simuliert ein Java-Bean, mit dem man Funktionalität durch neue GUI-Elemente erhalten kann.

Grober Aufbau der Klassen

Hauptklasse

- public class Aufgabe3 extends JFrame

Neue Label-Klasse

- class MyLabel extends Label

Der **neue** Konstruktor erlaubt hier einen double-Wert.
Der Default-Konstruktor und der String-Konstruktor dürfen nicht benutzt werden.

“Java Bean Klasse“

- class MyArtikelPanel

Diese Klasse speichert alle GUI-Elemente-. Sie nimmt **automatisch** Berechnungen vor.

Interface

- interface IArtikel

Artikel-Klassen

- class KFZ
- class PKW extends KFZ
- class LKW extends KFZ

- class Fahrrad
- class Rennrad extends Fahrrad
- class Klapprad extends Fahrrad
- class Tourenrad extends Fahrrad

Versuchsdurchführung:

Beim ersten Durchlauf wird es Fehlermeldungen geben. Danach erscheint nur ein JFrame mit einem Schalter.

1) Einfügen des GUI-Containers:

- Fügen Sie eine Instanz der Klasse „myArtikelPanel“ in das Hauptpanel. Dazu dürfen Sie aber nicht direkt auf das interne Panel zugreifen (das Panel ist private deklariert und soll es bleiben)

2) Einfügen der Schnittstelle in die Klassen:

- Fügen Sie die Schnittstelle und deren Methoden in die jeweiligen Klassenfamilien

3) Freigeben der Aktion-Methode des ersten Schalters:

- Bei korrekter Programmierung sollte die Aktion-Methode des ersten Schalters eingefügt werden können. Diese ist am Anfang auskommentiert

4) Ableiten von Label:

- Erzeugen Sie eine neue Klasse, abgeleitet von Label.
- Die Default- und der String- Konstruktor dürfen nicht verwendet werden.
- Entwickeln Sie einen neuen Konstruktor, der als Parameter einen Text und einen **Double-Wert** erhält.
- Fügen Sie die Methode „setPreis“ hinzu.

6) Einfügen der Actionmethoden:

- Programmieren Sie die Methoden der beiden Schalter. Das Berechnen der Summe und setzen des Summenlabels soll in einer eigenen Methode geschehen.