Kapitel: 4

Border-(pl.) sind eine nette Art Ein Fenster übersichtlicher zu gestalten. Sie können auf alle Swing Elemente angewendet werden, die eine Set Border Methode haben.
Eine große Hilfe bei der einfachen Erzeugung von Border-(pl.) ist das package BorderFactory, dass der Herr Kächele schon einmal vorgestellt hatte und das auch Gideon und ich schon häufig verwendet haben.
Das zu importierende Package heißt: javax.swing.BorderFactory; .
BorderFactory bietet eine Reihe vorgefertigter Border, die sehr einfach zu implementieren sind.
Ein Beispiel für eine Implementation von Borderfactory:

```
pn_Anmelden.setBorder(BorderFactory.createLineBorder(Color.lightGray));
```

ein Panel bekommt eine LineBorder (einfache Linien-Umrahmung mit der Farbe lightGray (helles Grau).

Ein anderes Beispiel wäre:

```
lst_StatusAuswahl.setBorder(BorderFactory.createBevelBorder(1));
```

mit BevelBorder kann man ein Swing Objekt entweder "eindrücken" (1) oder hervorheben(0) und somit einen netten 3D Effekt erzielen.

Wenn man jetzt also ein TextArea zum Beispiel mit einem Rahmen versehen will, wählt man zunächst das TextArea, nimmt die Methode setBorder, wählt die Klasse BorderFactory und setzt einen Punkt. Jetzt kann man sich eine von vielen Borders aussuchen. Für nähere Informationen kann man die JavaDocs unter dem Stichwort BorderFactory abrufen.
Wichtig! Es ist nicht nötig eine Instanz der Klasse BorderFactory zu bilden!

Viel Spaß mit diesen kleinen Spielereien

Ein Beispiel findet ihr im Anhang (Vorschau_1.bmp) => Ein Ausschnitt aus Gideons und meinem aktuellen Projekt (exklusiv ^^).

(P.S.: Sehr geehrter Herr Ziemke, wenn sich der Text nicht ordentlich anzeigen lässt, sagen sie einfach bescheid. Dann schreib ich demnächst in html)

Kapitel: 5

Nachdem das letzte Kapitel ein paar Spielereien erläutert hat wollen wir jetzt auf ein paar nützliche Funktionen von Swing eingehen (Spielerei ist trotzdem wieder dabei ;))

Die erste, die ich an dieser Stelle erklären möchte ist die Liste auch JList genannt.

dazu muss zunächst ein neues Objekt der Klasse JList nach dem altbekannten Verfahren erzeugt werden:
private JList lst_1 = new JList(Infos);
Infos ist in diesem Falle ein String Array, das zuvor schon definiert werden musste!
private String[] Infos = {"Hallo", "Was?", "jaja"}
das String Array darf natürlich einen beliebigen Inhalt mit beliebig vielen Strings enthalten.

diese zwei Zeilen fügen wir zu unseren Attributen hinzu.

In unserem Konstruktor können wir jetzt wieder die Größe festhalten und dann die Liste dann zu cp adden.
lst_1.setBounds(x, y, width, height); Hier müssen werte eingesetzt werden.
cp.add(lst_1);

Wenn wir jetzt unser Programm aufrufen stellen wir fest:
Die Wörter stehen mitten im Raum. Außerdem fällt auf, dass man mehrer Objekte markieren kann. Das mag in bestimmten Situationen richtig sein. Wir gehen jetzt aber davon aus, dass nur ein Objekt ausgewählt werden sollte.
in unserem Konstruktor ergänzen wir die Zeile:
lst_1.setSelectionMode(0);
eine Alternative wäre:
lst_1.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
Außerdem wollen wir das ganze Farblich etwas abgrenzen:
lst_1.setBackground(Color.BLUE);
Aus Kapitel vier arbeiten wir mit BorderFactory. Such dir einfach einen Border aus.
In Meinem Programm:
lst_1.setBorder(BorderFactory.createLoweredBevelBorder());
Wir stellen fest, dass das Blau ziemlich unansehnlich aussieht.
Lieber wäre uns wenn die Liste nur einen Hauch von Blau als Hintergrund hätte.
Dafür definieren wir einfach unsere eigene Farbe.
bei unseren Attributen fügen wir folgende Zeile ein:
Ein Objekt der Klasse Color mit dem Namen cl_1 und den RGB Werten von 0-255 50, 50, 255 den Alpha-Wert lassen wir weg, da sich sonst Grafisch unschöne Dinge beim selektieren ergeben.
die Zeile sieht dann so aus:
private Color cl_1 = new Color(200, 200, 255);
In unserer setBackground Methode ändern wir Color.BLUE in cl_1.
Jetzt ist die Liste recht angenehm.
Doch wie man sieht, ist das Markierte nur noch sehr schwer zu erkennen.
Wir werden also unsere Selektion auch noch anpassen:
Zunächst erzeugen wir ein neues Color Objekt:
private Color cl_2 = new Color(255, 20, 20, 30); <-- der Alpha Wert am Ende lässt dieses Rot blasser erscheinen.

Aus den JavaDocs:
An alpha value of 1.0 or 255 means that the color is completely opaque and an alpha value of 0 or 0.0 means that the color is completely transparent.

jetzt nehmen wir die Methode setSelectionBackground um unsere SelektionsFarbe zu bestimmen.
lst_1.setSelectionBackground(cl_2);

Als letztes legen wir noch fest welche Zeile am Anfang selektiert sein sollte.
lst_1.setSelectedIndex(1); legt fest, dass das ZWEITE Objekt als erstes Markiert sein soll (0 ist das erste).

Sind die Spielereien geklärt können wir jetzt anfangen zu sehen, was man mit einer List alles machen kann.
Füge zunächst drei Button und ein TextArea ein. Zusätzlich brauchen wir noch ein TextField.
Alternativ kannst du ein PopUp verwenden, sofern du damit vertraut bist (siehe meine Lösung zum Kapitel 3).

Fangen wir mit dem ersten Button an. Dieser soll den Markierten Text aus der Liste auslesen und in das TextArea eintragen. Zusätzlich soll die Position angegeben werden.
Das ganze geht über die Methoden getSelectedValue (gibt einen String (den Inhalt) aus) und getSelectedIndex (gibt einen int (Reihe-1 !) aus der mit String.valueOf als String verfügbar ist).

BSP.:
```
ta_Ausgabe.append(lst_1.getSelectedValue()+" in Reihe:
"+String.valueOf(lst_1.getSelectedIndex()+1));
```

Der Button 2 sollte alles auslesen können.
Dafür benutzen wir die Methode getMaxSelectionIndex (gibt den Index der letzten Reihe an).
Tipp: Mit einer While Schleife lässt sich das ganze relativ schnell lösen.
Da man aber nur den Selektierten Inhalt auslesen kann tricksen wir an dieser Stelle ein wenig in dem wir alle Objekte einmal selektieren lassen.

BSP.:

```
int i1 = 0;
   lst_1.setSelectedIndex(i1);
   while (i1 < lst_1.getMaxSelectionIndex()+1)
   {
    lst_1.setSelectedIndex(i1);
    ta_Ausgabe.append("*"+lst_1.getSelectedValue()+"* in Reihe:
"+String.valueOf(lst_1.getSelectedIndex()+1)+"\n");
    i1++;
    lst_1.setSelectedIndex(i1);

   }
```

Dies ist nicht die beste Lösung, aber ich habe keine andere Methode gefunden.
Wichtig an dieser Stelle: Nicht auf den String Array zugreifen! Er könnte veraltet sein und nicht mehr die Aktuellsten Informationen enthalten. Denn ein Array hat immer eine feste Anzahl an Strings und kann nicht erweitert werden!

Der Dritte Button ist schon etwas schwieriger. Ich konnte keine Methode finden, eine JList zu erweitern, was bei Dropdown Listen ohne weiteres möglich ist. Deshalb können wir nur den Inhalt ersetzten! Bzw. den Inhalt erst auslesen und dann etwas anfügen und dann ersetzten.
Hier kann man selbst etwas herum experimentieren. Ich füge im Anhang ein Beispiel ein wie es möglich ist. Aber angesichts der nicht aufzufindenden Methoden ist der Dritte Button wohl nicht "sauber" zu programmieren. (Sollte ich mich irren BITTE MELDEN).

Für das DropDown Menu ist es etwas anders.

dort kann die Methode addItem*(String)* verwendet werden. Somit verkürzt sich in dieser Hinsicht der Aufwand.
Außerdem kommen dort die Methoden getItemAt*(int)* und getItemCount() zum Einsatz.
Das neu schreiben mit einem Dropdown Menu sollte kein Problem sein.
Für alle Fälle im Anhang ein Beispiel.

Es gilt ansonsten das selbe Rot gedruckte wie bei Kapitel 1-3

Kapitel 6:

Java Docs:
When a Java Virtual Machine starts up, there is usually a single [...] thread (which typically calls the method named `main` of some designated class).

Ich versuche Jetzt mal das Thema Threads anzuschneiden.
Da ich selbst noch nicht sooo viel über Threads weiß wird es denke ich für alle verständlich und interessant um selbst Nachforschungen anzustellen.

Die Programme die wir bis jetzt geschrieben haben bestanden immer aus einem Thread. Daher hat sich auch im Gui nichts mehr getan, wenn wir eine Endlos-While Schleife konstruiert hatten, weil nur eine Sache zu einem bestimmten Zeitpunkt gemacht werden kann. Die Implementation von Threads erlaubt es uns, mehrere Dinge gleichzeitig zu tun.
Ich werde versuchen es zu zeigen indem ich ein Projekt erkläre in dem ich eine Uhr implementiert habe, die Pro Sekunde einmal auf den neuesten Stand gebracht wird, wobei ich jederzeit im Programm alles machen kann.

Und los geht's

Als erstes brauchen wir wieder ein geeignetes Gui. Ein Textarea, ein Button und ein Label sollen genügen.
Auf dem Label wollen wir eine Digitaluhr sehen.
Um Uhrzeit und ähnliches zu verwenden müssen zunächst folgende Klassen importiert werden:
import java.util.*;
import java.text.DateFormat;

oder

import java.util.Date;
import java.text.DateFormat;

Als Button Funktion sollte man sich einfache Dinge überlegen. Wenn die Funktion zu aufwendig wird ist der Vorführeffekt hin, da das Gui dann nicht wegen der Uhrzeit, sondern auf Grund des Buttons freezed.

JavaDocs:
The `Runnable` interface should be implemented by any class whose instances are intended to be executed by a thread. The class must define a method of no arguments called `run`.

Da wir eine Instanz von Date aufrufen brauchen wir das auch.
und so wird das ganze IN DEN KONSTRUKTOR implementiert, damit es von anfang an läuft. Man könnte natürlich auch die Uhr per Button starten, aber das sparen wir uns jetzt.

```
new Thread( new Runnable()   <--siehe Zitat
   {
     public void run()   <-- siehe Zitat
     {
       while ( true )   <-- immer Wahr (die Uhr hält nicht an)
       {
         try
         {
           Date date = new Date();     <-- eine Instanz von Date wird erschaffen
           DateFormat format = DateFormat.getTimeInstance();   <---enthält Formatierungen der
Uhrzeiten
           lb_Uhrzeit.setText( format.format(date) );   <-- hier wird die fertig formatierte Uhrzeit in das
Label Uhrzeit geschrieben
           Thread.sleep( 1000 );   <--- jetzt wird 1000ms / 1s gewartet bis es weiter geht
         }
         catch ( InterruptedException e ) { }   <---muss mit implementiert werden, falls etwas den Thread
unterbricht (beenden des Ursprungsthreads)
       }
     }
   }).start();  <---startet den Thread
```

Und schon habt ihr eine funktionierende Uhr

Viel Spaß mit Threads