… und mein Button JButton

javax.swing

- Klassenbibliothek für Oberflächen
- weitestgehend selbst in Java programmiert
- sehr flexibel
- ziemlich langsam
- Layoutmanager und Eventhandling werden von awt übernommen
- Ist ansatzweise nach dem modell-view-controller pattern implementiert, wobei view und controller meist zusammengefasst sind.

Model - View - Controller

Model

Implementiert das Datenmodell

Zustandsinformationen Datenwerte

Applikationscode

View

Darstellung der im Modell verwalteten Daten/Zustände

Darstellung durch geeignete paint Methode

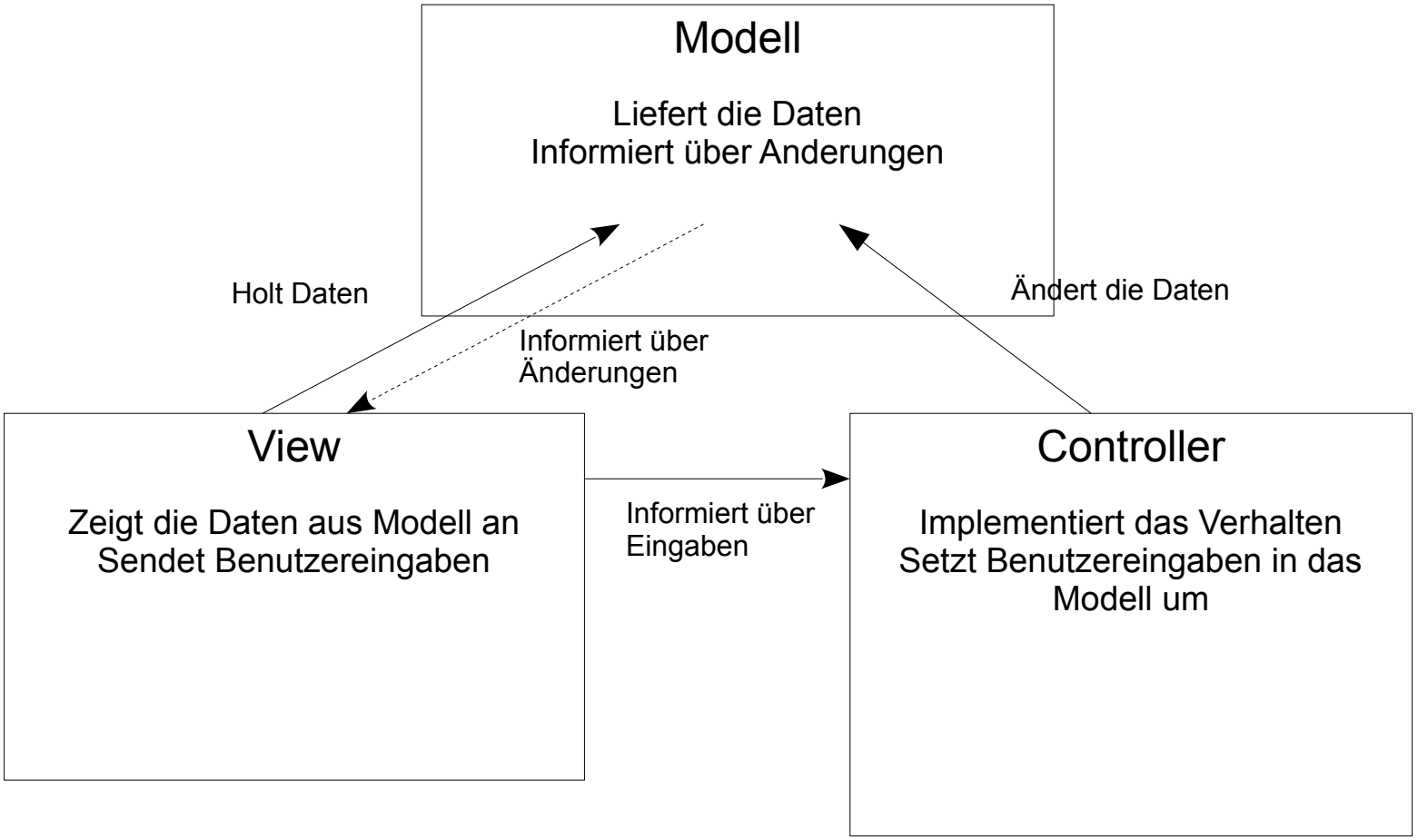
Darstellung durch Benutzung geeigneter GUI Components

Es kann mehrere Views geben

Controller

Zuständig für das Eventhandling

Applikationscode


Modell
Liefert die Daten
Informiert über Anderungen
Holt Daten
Informiert über
Änderungen
Ändert die Daten
View
Zeigt die Daten aus Modell an
Sendet Benutzereingaben
Informiert über
Eingaben
Controller
Implementiert das Verhalten
Setzt Benutzereingaben in das
Modell um

Swing Components (Stand java 6)

Container	*Layoutmanager*	*JComponents*	*Menuelemente*
JApplet	BoxLayout	JButton	JMenu
JDialog	(OverlayLayout)	JCheckBox	JMenuBar
JFrame	JButtonGroup	JComboBox	JMenuItem
(JLayeredPane)	(SpringLayout)	JColorChooser	JPopupMenu
JPanel	(JViewPortLayout)	JEditorPane	JToolBar
JOptionPane		JFileCooser	
(JWindow)		JLabel	
Box		JList	
JDesktopPane		JPasswordField	
JInternalFrame		JProgressBar	
JScrollPane		JRadioButton	
JSplitPane		JScrollBar	
JTabbedPane		JTextArea	
(JRootPane)		JTextField	
(JViewPort)		JTextPane	
		JTable	
		JTree	
		JFormattedTextField	
		JSpinner	

Components

- JButton, JTextField, JTextArea, JLabel, JCheckBox, JScrollBar entsprechen in ihrer wesentlichen Funktionalität den Komponenten von awt.
- Scrollbars werden an JList und JTextArea nicht automatisch ergänzt, ein JScrollPane ist erforderlich
- JLabel und JButton können Text und Icons enthalten, die Anordnung/Ausrichtung ist sehr variabel

Button/Label

```
JLabel  l=new JLabel("JAVA:",
                     new  ImageIcon("java.png"),
                     SwingConstants.LEFT);
l.setHorizontalTextPosition(SwingConstants.RIGHT );
add(l);
JButton b=new JButton("Suse",
                      new ImageIcon("gnome-suse.png"));
b.setHorizontalTextPosition(SwingConstants.RIGHT );
add(b) ;
```

JComboBox,

ersetzt Choice von awt

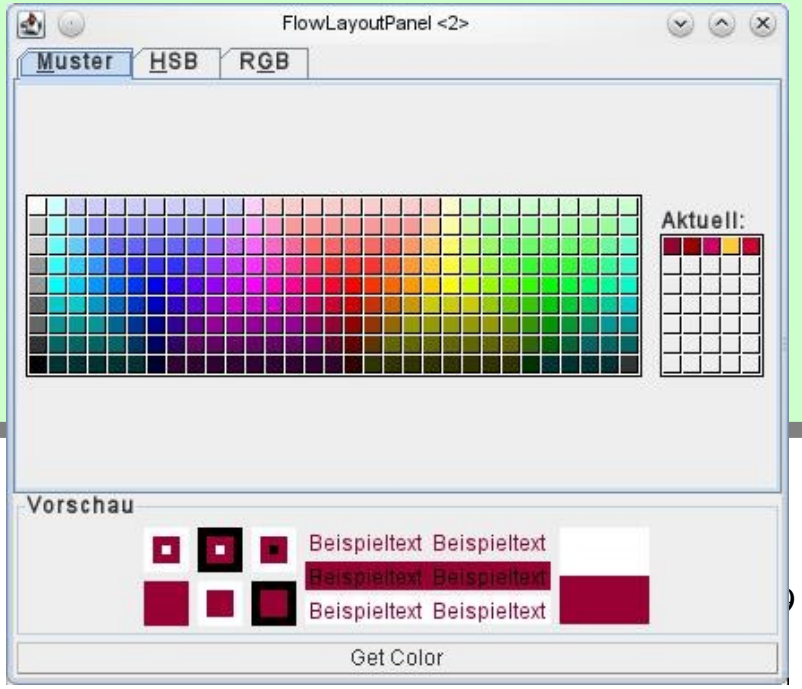
```
JComboBox b;

public JCombo()
{
 Vector<String> v=new Vector<String>();
 v.add("Klein Bubi");
 v.add("Klein Erna");
 v.add("Frau Pumeier");
 b=new JComboBox(v);
 add(b) ;
 b.addItemListener(new ItemListener(){
       public void itemStateChanged(ItemEvent e)
       {
         System.out.println((String)(b.getSelectedItem()));
       }});
}
```

JColorChooser

```
JColorChooser b;

public JColor()
{
 setLayout(new BorderLayout());
 b=new JColorChooser();
 add(b,BorderLayout.CENTER) ;
 Button Bget=new Button("Get Color");
 add(Bget,BorderLayout.SOUTH);
 Bget.addActionListener(new ActionListener()
 {
  public void actionPerformed(ActionEvent e)
  {
    System.out.println
        ("Color: "+b.getColor());
  }
 });
}
```



JEditorPane

```
JEditorPane p=new JEditorPane(new URL(args[0]));
JScrollPane x=new JScrollPane(p);
p.setEditable(false);
p.addHyperlinkListener(new Hyperactive());
```

```
class Hyperactive implements HyperlinkListener
 {
   public void hyperlinkUpdate(HyperlinkEvent e)
   {
      if (e.getEventType() == HyperlinkEvent.EventType.ACTIVATED)
      {
        JEditorPane pane = (JEditorPane) e.getSource();
        if (e instanceof HTMLFrameHyperlinkEvent)
        {
          HTMLFrameHyperlinkEvent  evt =
                                          (HTMLFrameHyperlinkEvent)e;
          HTMLDocument doc = (HTMLDocument)pane.getDocument();
          doc.processHTMLFrameHyperlinkEvent(evt);
        } else
        {
           try {pane.setPage(e.getURL()); }
           catch (Throwable t) {t.printStackTrace(); }
        }
      }
   }
 }
```

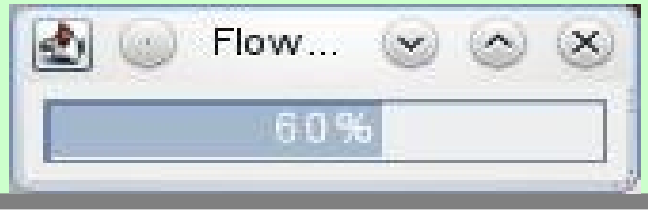
JProgressBar

```
public class Progress extends JPanel implements Runnable
{
  int count=0;
  JProgressBar b;
  Thread t=new Thread(this);
  public Progress()
  {
    b=new JProgressBar(JProgressBar.HORIZONTAL,0,100);
    b.setStringPainted(true);
    add(b) ;
  }

  public static void main(String args[])throws Exception
  {
    Progress p=new Progress();
    . . .
    new BufferedReader(new
            InputStreamReader(System.in)).readLine();
    p.t.start();
  }
}
```

Start des Threads

Eine Eingabe, um zu warten




```
public void run()
{
  while (true)
  {
    try
    {
     Thread.sleep(count*20);
    }catch(Exception e){}
    count+=10;
    b.setValue(count);
    b.setString(""+count+"%");
    System.out.println(count);
    if (count==100) return;
  }
}
```

JList

- JList gehört zu den JComponents, die sich eines expliziten Datenmodells bedienen.
- Es gibt eine Reihe Standardimplementationen, die es erlauben, eine Listbox aus einem Objektarray oder einem Vector Objekt zu erzeugen.
- Liegen die Daten jedoch in einem proprietären Format vor, vielleicht in Form einer sortierten doppelt verketteten Ringliste vom Beck, so kann man ein eigenes Datenmodell dafür implementieren.

Das Datenmodell

```
class myListModel extends AbstractListModel
{
    // Liefert die Anzahl der darzustellenden Datensaetze
   @Override
   public int getSize()
   {
      myData S;
      int i;
      for(i=0, S=(myData)L.getFirst();
          S!=null;
          S=(myData)L.getNext(),i++);
      return i;
   }
    // Liefert das Datenelement mit Index index
   @Override
   public Object getElementAt(int index)
   {
      myData S;
      return (myData)L.getIndexed(index);
   }
}
```

Dieses Datenmodell verwendet eine selbst programmierte verkettete Liste. Die Klasse **`MyListModel`** ist als Memberclass konzipiert und übernimmt die Liste L aus der umgebenden Klasse.

Die Daten in der Liste

```
class myData extends Object implements sortableObject
{
   public myData(String Data)
   { this.Data=Data; }

   public String toString()
   { return Data; }

   public int compareTo(sortableObject o)
   { return Data.compareTo(((myData)o).getData()); }

   public String getData(){return Data;}

   private String Data;
}
```

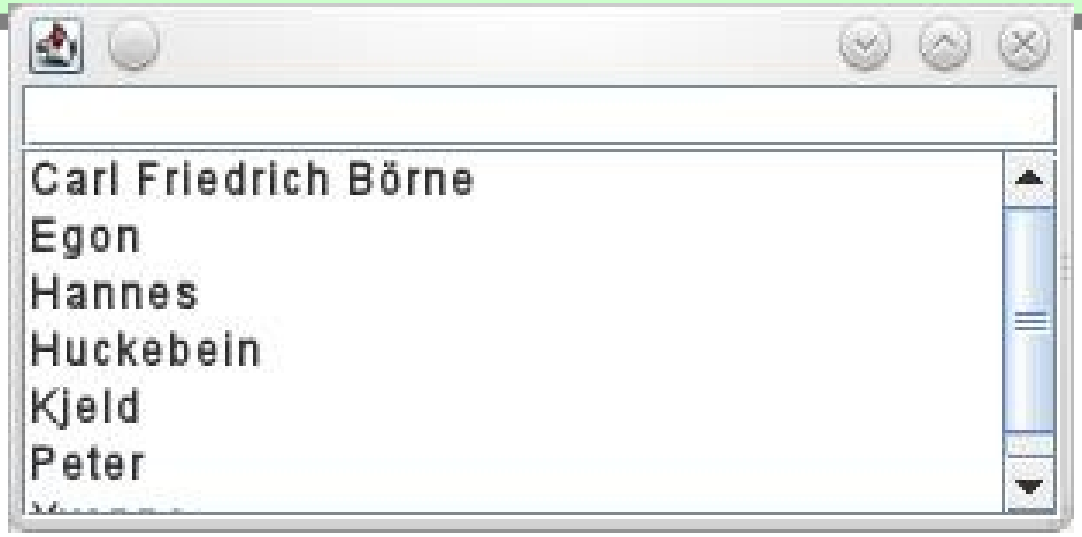
Die Daten, die von der Liste verwaltet werden, sind vom Typ sortableObject, einem selbst erdachten Interface mit der Methode compareTo. Diese Daten kann die Klasse beck.becksList.becksList.class dann sortiert verwalten.

Constructor der Anwendug

```
public ListTest(String args[])
{
  setLayout(new BorderLayout());
  int i;
  L=new becksList();
  for (i=0; i<args.length;i++)
  {
    L.insertSort(new myData(args[i]));
  }
  JL=new JList(new myListModel());
  add(new JScrollPane(JL),BorderLayout.CENTER);
  add(T,BorderLayout.NORTH);
  T.addActionListener(new ActionListener()
  {
    public void actionPerformed(ActionEvent e)
    {
       L.insertSort(new myData(T.getText()));
       JL.setModel(new myListModel());
    }
  });
}
```

Die main-Funktion

```
public static void main(String args[])
{
  ListTest T=new ListTest(args);
  JFrame F=new JFrame();
  F.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  F.getContentPane().add(T);
  F.setVisible(true);
  F.pack();
}
```



JTable

- Darstellung von Daten in tabellarischer Form
- Verwendet ebenfalls ein Modell
  - `GetRowCount` liefert die Anzahl der Zeilen der Tabelle.
  - `GetColumnCount` liefert die Anzahl der Spalten der Tabelle.
  - `getValueAt` liefert den Wert, der in Zeile und Spalten ausgegeben werden soll.
  - `GetColumnName` liefert den Namen der Spalte in der Überschrift
  - `GetColumnClass` liefert die Klasse von der die darzustellenden Daten sind.
- JTable hat nicht automatisch ScrollBars, sie müssen mittel eines `JscrollPane` Containers ergänzt werden.

JTable

```
class myAbstractTableModel extends AbstractTableModel
{
        String[] columnNames={" Name ",  . . .  ," Phone ",""};
        int nRow;
        Object[][] data;
        Address A;

        myTableModel()
        { . . .}

        public int getRowCount(){return data.length;}

        public int getColumnCount(){return data[0].length;}

        public Object getValueAt(int row, int col)
                    {return data[row][col];}

        public String getColumnName(int ColIdx)
                    {return columnNames[ColIdx];}

        public Class getColumnClass(int ColIdx)
                    {return data[0][ColIdx].getClass();}
};
```

Constructor füllt die Adressen in das Objektarray

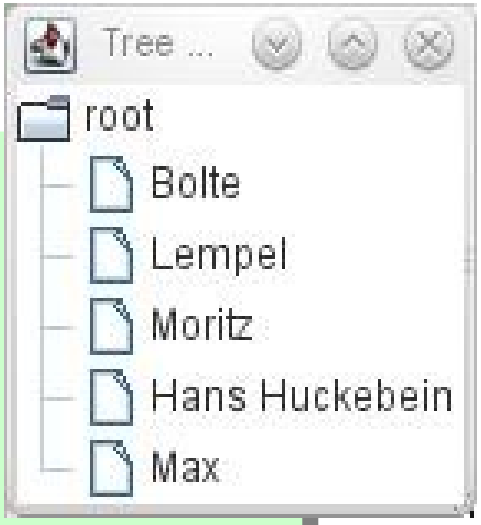
Erzeugen der JTable

```
myTableModel model=new myTableModel();
Tbl =new JTable(model);
S=new JScrollPane(Tbl);
add(S);
```

JTree

- Dient der Darstellung baumstrukturierter Daten.
- Als Modell können Defaultmodelle von Collections verwendet werden.
- Richtig interessant werden die dynamisch aufgebaut werden.
- Als Beispiel wird eine JtreeA-wendung zur baulstrukturierten Ansicht von XML-Daten auzugsweise betrachtet.
- Zum Lesen der XML-Daten wird ein org.xml.sax.ContentHandler verwendet, er befindet sich in xerxes.jar (kann kostenlos per Download bezogen werden).

JTree – JTree aus Collection




```
import javax.swing.*;
import javax.swing.tree.*;
import java.util.*;
public class HashHandleTree
{
 public static void main(String[] args)
 {
  JFrame f = new JFrame("Tree from a Hashtable");
  Hashtable<String,String> h =
                                new Hashtable<String,String>();
  int i=0;
  for(String s:args) h.put(s,""+(i++));
  JTree t = new JTree(h);
  t.putClientProperty("JTree.lineStyle", "Angled");
  t.setRootVisible(true);
  t.setShowsRootHandles(false);
  f.add(t);
  f.pack();
  f.setVisible(true);
 }
}
```

JTree & XML

```
JTree T;
DefaultMutableTreeNode Curr;

xmlsax(String xmlFileName)
{
  Curr=new DefaultMutableTreeNode("XML: "+xmlFileName);
  T=new JTree(Curr);
  . . .
  // preparing XML-Reader

}
```

XML sax Contenthandler (Ausschnitt)

```
class myContentHandler implements org.xml.sax.ContentHandler
{
  public void characters(char[] ch, int start, int length)
  {
   if (ch[start]!='\n'&&length>=1)
   {
    DefaultMutableTreeNode pi=
                          new DefaultMutableTreeNode("Chars: "+new String (ch,start,length));
    Curr.add(pi);
   }
  }
  public void endElement(String namespaceURI, String localName, String qName)
  {
   Curr=(DefaultMutableTreeNode)Curr.getParent();
  }
  public void startElement(String namespaceURI, String localName, String qName, Attributes atts)
  {
   DefaultMutableTreeNode pi=new DefaultMutableTreeNode("Element "+localName);
   Curr.add(pi);
   Curr=pi;
  }
  . . .
}
```

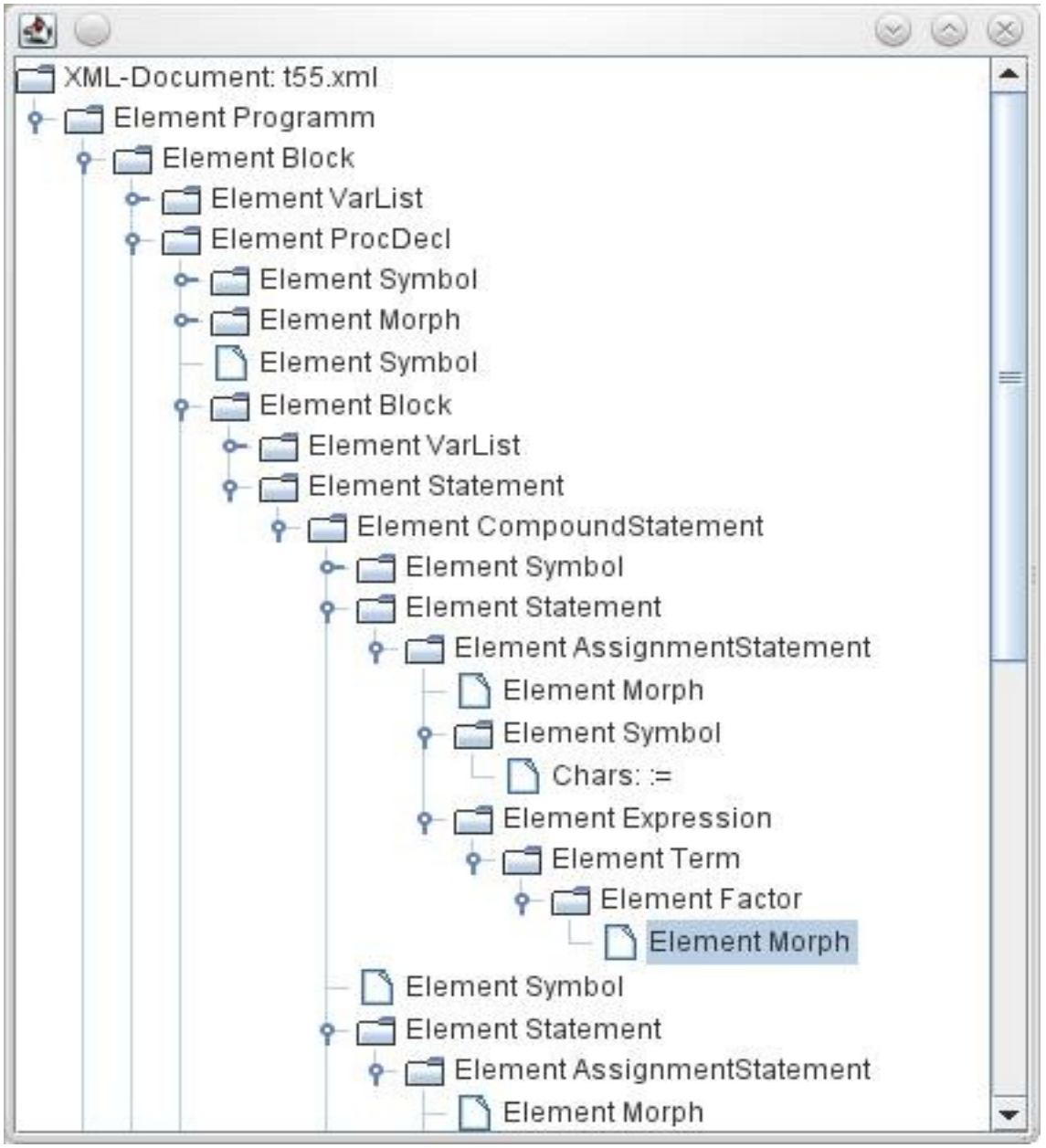
neues Blatt erzeugen
und anhängen an akt. Zweig

Hier geht's wieder zurück

neues Blatt erzeugen

Das neue Blatt wird
neuer aktueller Zweig

teilweise aufgeklappter Baum

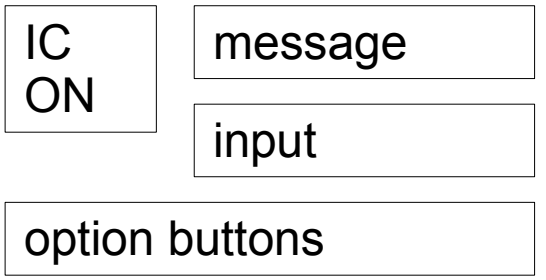


Container

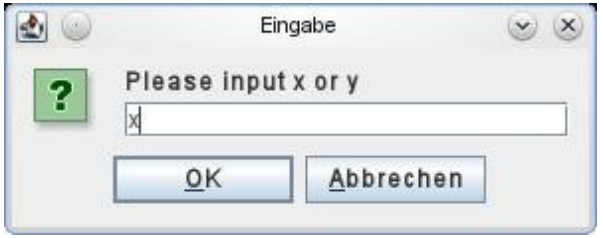
- Container, die weitestgehend denen von awt entsprechen, allerdings über darüber hinaus gehende Funktionalität verfügen.
  - JApplet
  - JFrame
  - JPanel
  - JDialog

JOptionPane

- Erzeugt eine zusammengesetzte Komponente aus bis zu 4 Bestandteilen
- Über spezielle Methoden wird diese Komponente in einen Dialog eingebettet und angezeigt




```
String axis = JOptionPane.showInputDialog("Please input x or y");
```

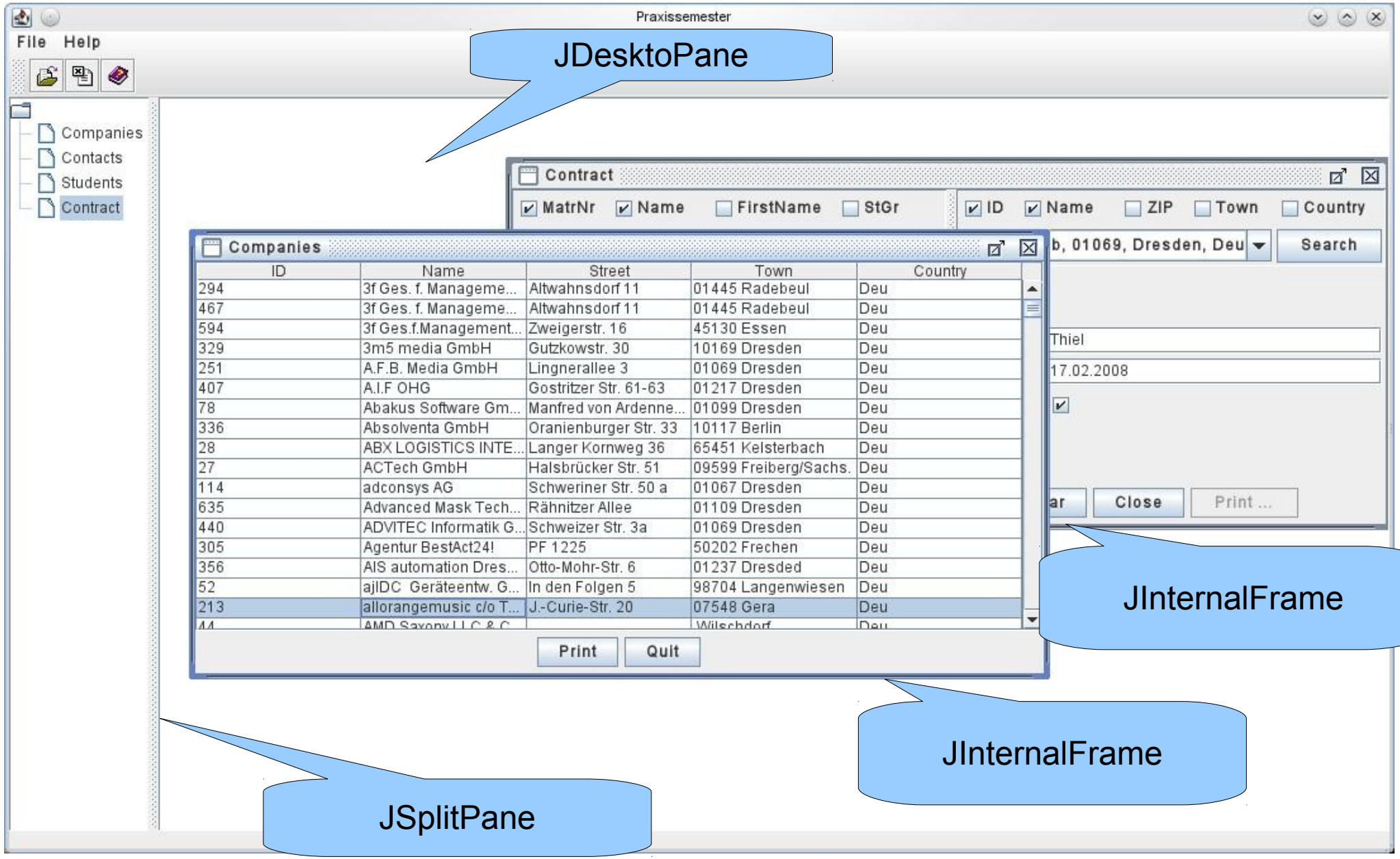


Weitere Beispiele JOptionPane

```
JOptionPane.showMessageDialog(this, "alert", "alert",
JOptionPane.ERROR_MESSAGE);
```


```
JOptionPane.showMessageDialog(this,
"information","information",
JOptionPane.INFORMATION_MESSAGE);
```

JSplitPane / JDesktopPane



JSplitPane

- Teilt einen Container in zwei Bereiche
  - Horizontal
  - Vertikal
- Nimmt zwei Komponenten auf

```
JSplitPane jSplitPane1 = new JSplitPane();

jSplitPane1.add(Desk,   JSplitPane.RIGHT);
jSplitPane1.add(jTree1, JSplitPane.LEFT );
```

```
JSplitPane jSp=new JSplitPane(int newOrientation)

newOrientation –

JSplitPane.HORIZONTAL_SPLIT

JSplitPane.VERTICAL_SPLIT
```

JDesktopPane

Dient dem Aufbau von Multidocumentanwendungen.
Stellt als Komponente eine Art Desktop bereit.
Auf diesem Desktop können InternalFrame-Objekte dargestellt werden. Diese verhalten sich in etwa, wie Frames auf dem Rechnerdesktop, können verschoben, in der Größe verändert oder iconifiziert werden.

```
JDesktopPane Desk=new JDesktopPane();

JInternalFrame F=new JInternalFrame(Title,true,true,true);
```

```
JInternalFrame(String title,
               boolean resizable,
               boolean closable,
               boolean maximizable)
```

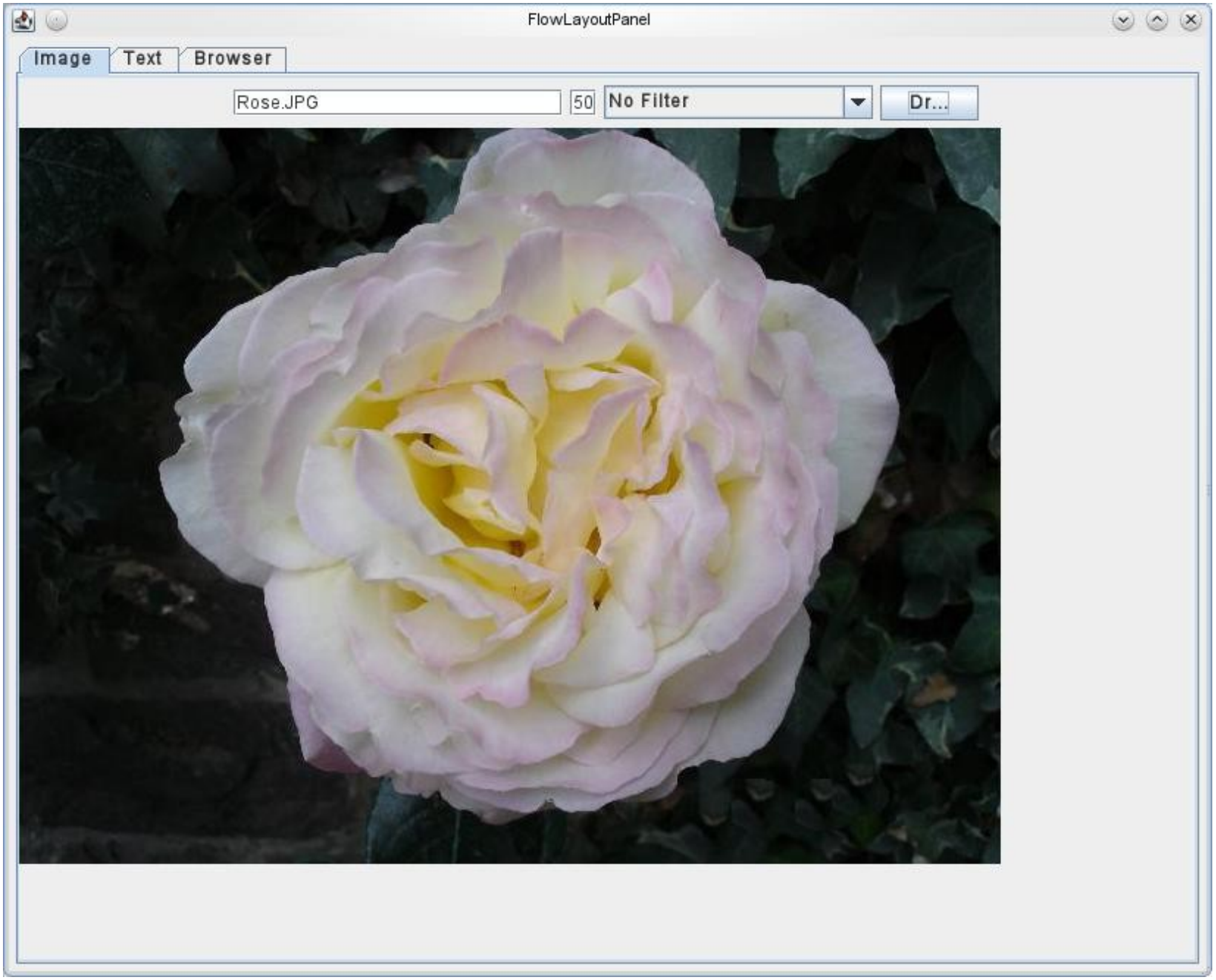
JTabbedPane

- Stellt einen Karteikartenkasten zur Verfügung
- Jede Karte verfügt am oberen Rand über einen Reiter mit einem Label
- Es wird die Karte, deren Reiter per Click aufgerufen wird, angezeigt.
- Jede Karte kann eine Komponente darstellen.
- Im Beispiel sind die Klassen
  - `MyImageLines`
  - `FileViewer`
  - `TestEditorPane`

  selbst erstellte Componenten.
-

JTabbedPane

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class Tabbed extends JPanel
{
  public Tabbed(JFrame f)throws Exception
  {
    JTabbedPane T=new JTabbedPane();
    T.addTab("Image",new myImageLines("Rose.JPG",f));
    T.addTab("Text", new FileViewer("rose.txt"));
    T.addTab("Browser",new TestEditorPane(
    "http://www.welt-der-rosen.de/rosged/ronsard_geb.htm"));
    add(T);
  }
  public static void main(String args[])throws Exception
  {
    JFrame f=new JFrame("FlowLayoutPanel");
    Tabbed p=new Tabbed(f);
    f.add(p,BorderLayout.CENTER);
. . .
  }
}
```


FlowLayoutPanel
Image
Text
Browser
Rose.JPG
50
No Filter
Dr...