Software-Entwicklung II
SS 2004

Prof. Dr. Uwe Kastens

Ziele der Vorlesung

Ziele der Vorlesung Software-Entwicklung II

Die Studierenden sollen

- **graphische Bedienungsoberflächen** mit objektorientierten Techniken entwickeln können,
- die Grundlagen **paralleler Prozesse und deren Synchronisation** verstehen und parallele Prozesse in Java programmieren lernen.
- ihre Kenntnisse in der objektorientierten Programmierung in Java festigen und verbreitern.

Voraussetzungen aus Software-Entwicklung I:

Die Studierenden sollen

- die **Programmentwicklung in Java von Grund auf erlernen**.
- lernen, Sprachkonstrukte sinnvoll und mit **Verständnis** anzuwenden.
- grundlegende **Konzepte der objektorientierten Programmierung** verstehen und anzuwenden lernen. Objektorientierte Methoden haben zentrale Bedeutung im **Software-Entwurf** und in der **Software-Entwicklung**.
- lernen, **Software aus objektorientierten Bibliotheken wiederzuverwenden**.
- **eigene praktische Erfahrungen** in der Entwicklung von **Java**-Programmen erwerben. Darauf bauen größere praktische Entwicklungen in Java oder anderen Programmiersprachen während des Studiums und danach auf.

Inhalt

Nr. d. Vorl.	*Inhalt*	*Abschnitte in „Java lernen, 2. Auflage“*
1	1. Einführung, GUI, AWT	10.1
2	2. Zeichenflächen	10.2
3	3. Komponenten erzeugen und platzieren	10.3
4	4. Hierarchisch strukturierte Fensterinhalte	
	5. Ereignisse an graphischen Benutzungsoberflächen	10.3, 11.1
5	Eingabeverarbeitung	11.1
6	6. Beispiel: Ampelsimulation	10.5
7	7. Entwurf von Ereignisfolgen	11.4
8	8. Java-Programme in Applets umsetzen	12.1, 12.2
9	9. Parallele Prozesse, Grundbegriffe, Threads	13.1, 13.2
10	10. Unabhängige parallele Prozesse,	13.1, 13.2
11	11. Monitore, Synchronisation gegenseitiger Ausschluss	13.3
12	12. Bedingungssynchronisation im Monitor	
13	13. Verklemmungen, Beispiel: Dining Philosophers	
14	14. Zusammenfassung	

Literaturhinweise

Elektronisches Skript zur Vorlesung:

- **U. Kastens: Vorlesung SWE II, 2004, http://ag-kastens.upb.de/lehre/material/sweii**
- **U. Kastens: Vorlesung SWE, 1998/99 (aktualisiert), http://..../swei**

fast ein Textbuch zur Vorlesung, mit dem Vorlesungsmaterial und JDK auf CD:

- **J. M. Bishop: Java lernen, Addison-Wesley, 2. Auflage, 2001**
- **J. M. Bishop: Java Gently - Programming Principles Explained, Addison-Wesley, 1997 3rd Edition (Java 2)**

zu allgemeinen Grundlagen der Programmiersprachen in der Vorlesung:

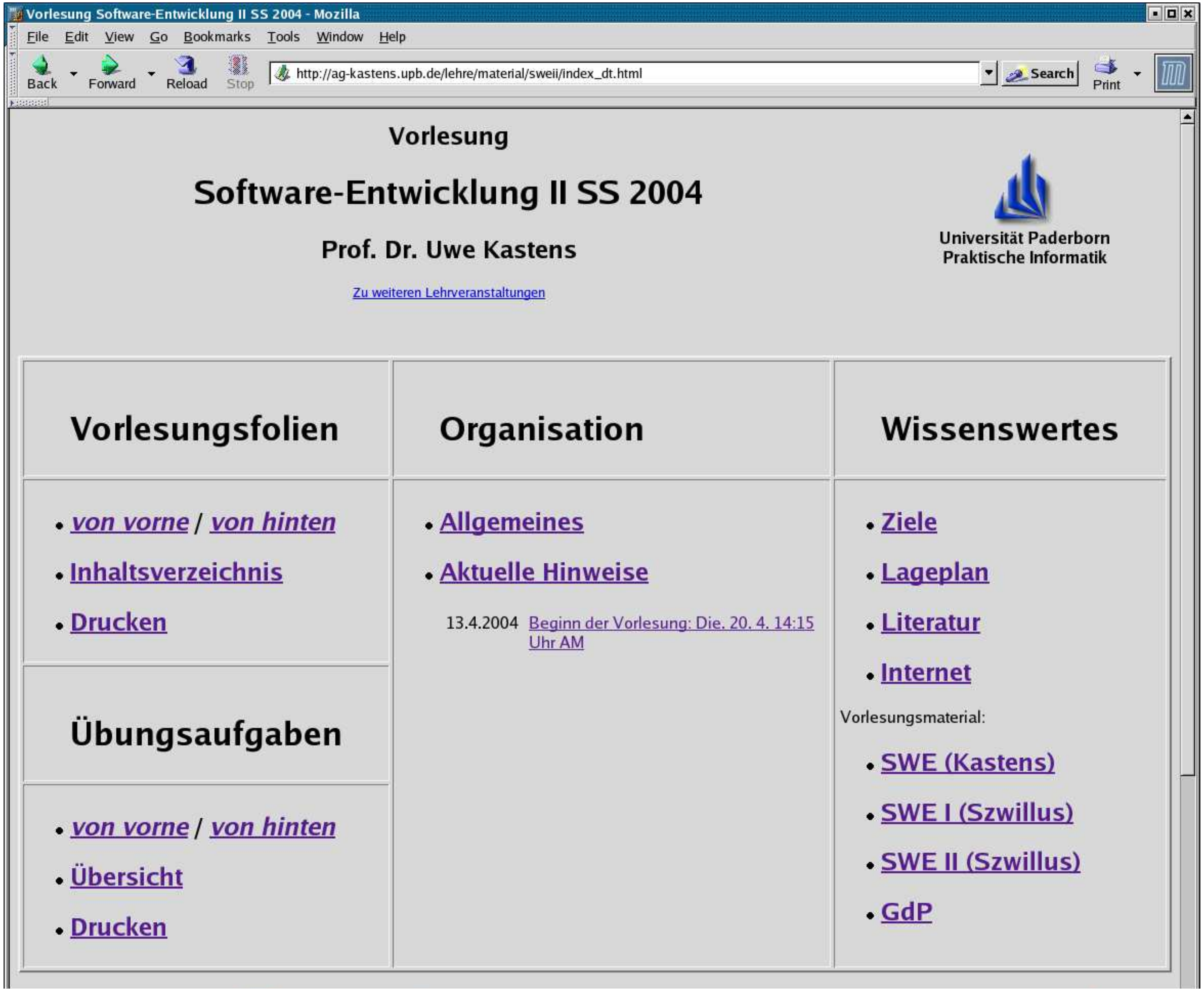
- **U. Kastens: Vorlesung Grundlagen der Programmiersprachen, Skript, 2003 http://www.upb.de/cs/ag-kastens/gdp**
- **D. A. Watt: Programmiersprachen - Konzepte und Paradigmen, Hanser, 1996**

eine Einführung in Java von den Autoren der Sprache:

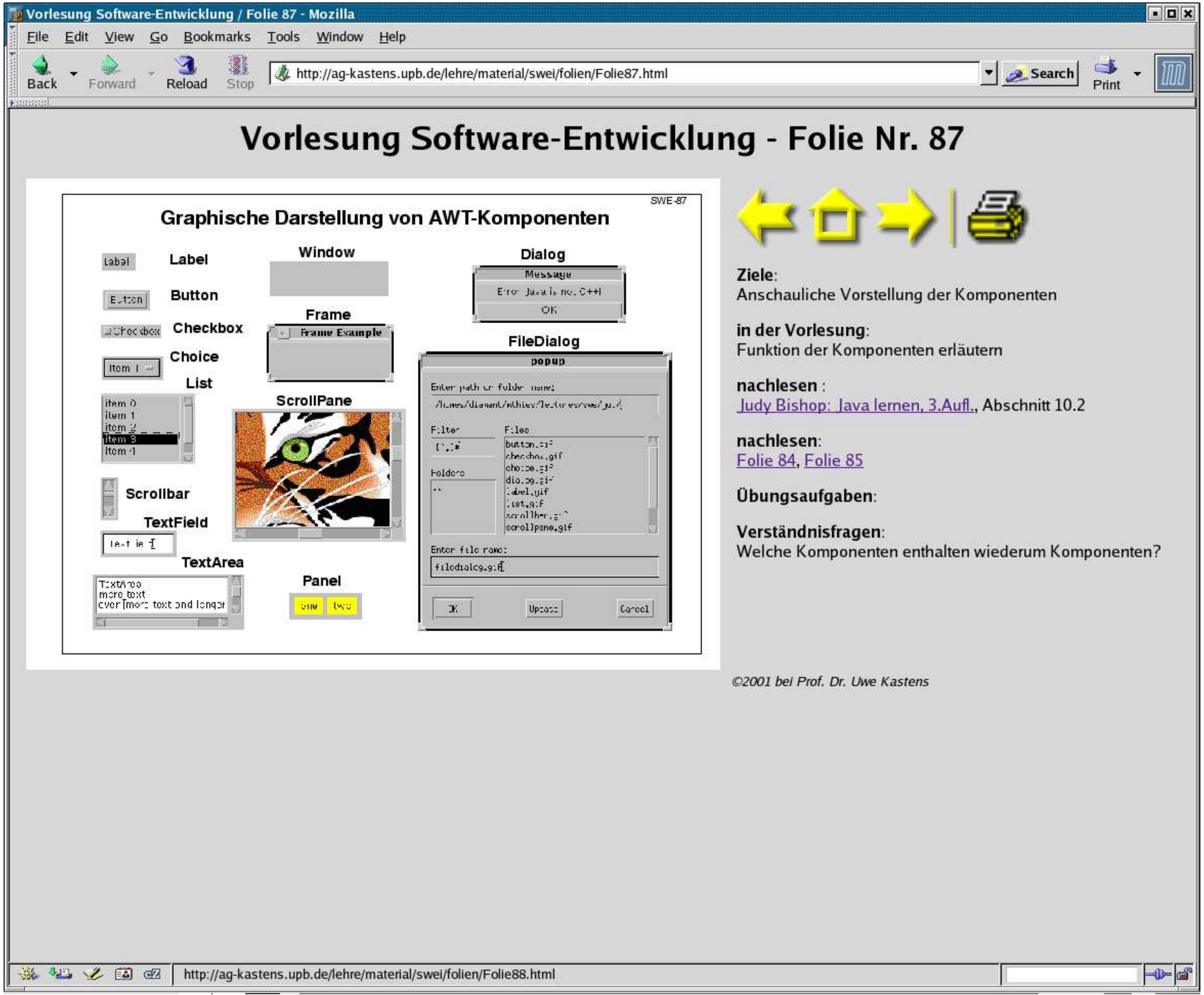
- **Arnold, Ken / Gosling, James: The Java programming language, Addison-Wesley, 1996.**
- **Arnold, Ken / Gosling, James: Die Programmiersprache Java TM, 2. Aufl. Addison-Wesley, 1996**

Hinweise auf weiteres Material im SWE-Skript im WWW

Elektronisches Skript: HomePage



Elektronisches Skript: Folien im Skript



Elektronisches Skript: Organisation der Vorlesung

Vorlesung Software-Entwicklung II - Organisation - Mozilla

File Edit View Go Bookmarks Tools Window Help

Organisation

Sprechstunde Prof. Dr. Uwe Kastens:

- Mo 11.00 - 12:00 F2.308
- Do 16.00 - 17.00 F2.308

Termine

Vorlesung:

- Di 14:15 - 15:45 AM
- Mi 14:15 - 15:45 AM

Beginn: Dienstag, 20.4.2004

Zentralübung:

- Mi 13:15 - 14:00 AM

Beginn: Mittwoch, 5.5.2004

Übungsbetreuer:

Tomasz Mistrzyk

Übungen:

- Gruppe 1 Mo 11 - 13 E4.101 N.N.
- Gruppe 2 Mo 14 - 16 E4.101 N.N.
- Gruppe 3 Mo 11 - 13 E3.143 N.N.
- Gruppe 4 Mo 14 - 16 E3.143 N.N.
- Gruppe 5 Mo 16 - 18 E3.143 N.N.
- Gruppe 6 Mo 11 - 13 D3.344 N.N.
- Gruppe 7 Mo 14 - 16 D3.344 N.N.

Vorlesung Software-Entwicklung II - Organisation - Mozilla

File Edit View Go Bookmarks Tools Window Help

- Gruppe 21 Fr 14 - 16 E3.301 N.N.

Beginn: Montag, 26.4.2004

Sichern Sie sich Ihren Platz bis Dienstag, 25.04.2004: Benutzen Sie Ihren Eintrag aus der Teilnehmer-Datenbank von SWE I (WS 2003/04) (siehe unten). Dort können Sie den Wunschtermin angeben, zu dem Sie eine Übungsgruppe besuchen möchten. Wenn in der Gruppe noch Plätze frei sind, wird Ihr Wunsch erfüllt.

Teilnehmer-Datenbank:

- In Ihrem Teilnehmerkonto können Sie jederzeit die aktuell über Sie gespeicherten Informationen abrufen.

  Hier geht es zu Ihrem Teilnehmerkonto: Mein Konto NEU

  Sie können Ihre persönlichen Informationen, sowie die Zugehörigkeit zu einer Übungsgruppe und Ihre Punkte aus den Hausaufgaben einsehen. Später finden Sie hier die Klausuranmeldung und erfahren Ihr Klausurergebnis.

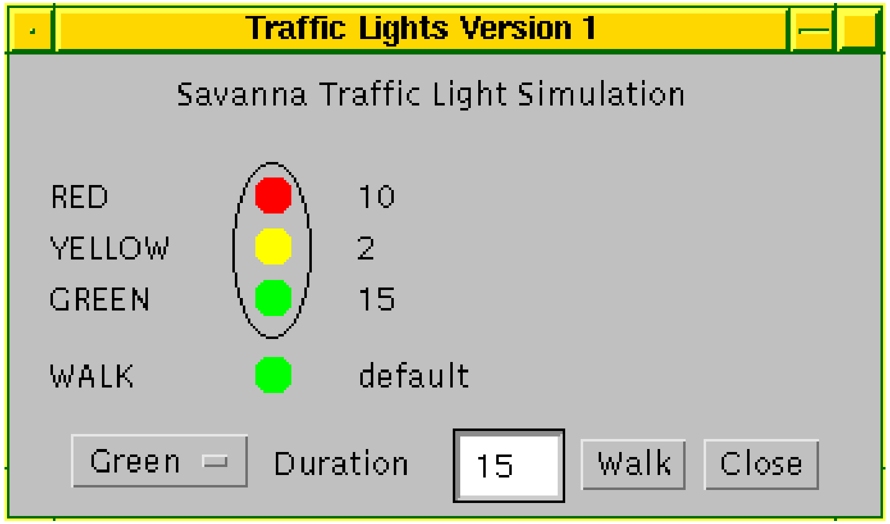
Übungsaufgaben:

- Die Übungsblätter werden einmal pro Woche (meist freitags) im WWW veröffentlicht und können dann bis zur darauffolgenden Woche bearbeitet werden. Abgabetermin: siehe Übungsblatt (meist freitags).
- Insgesamt 6 der Übungsaufgaben werden als **Korrekturaufgabe** gekennzeichnet. Abgaben dieser Aufgaben werden auf einer Punkteskala von 0-2 bewertet (nicht bearbeitet, sinnvoll bearbeitet, gut bearbeitet). Insgesamt können durch die bewerteten Aufgaben also 12 Punkte erreicht werden.
- Die Übungspunkte lassen sich wie folgt verwenden, um die Noten von **bestandenen** Klausuren zu verbessern: Wer mindestens 6 Punkte erreicht, verbessert sein Klausurergebnis um einen Zwischenwert, z. B. von 2.3 auf 2.0.
- **Abgabe der Korrekturaufgaben**
  - Geben Sie **in Gruppen von bis zu drei Teilnehmern** ab.
  - Bitte notieren Sie im Kopf Ihrer Abgabe die Nummer des Übungsblattes und die Namen und Matrikelnummern aller Beteiligten. Die Nummer der Übungsgruppe muss deutlich auf die Abgabe geschrieben werden, damit wir die korrigierten Zettel in die richtige Gruppe mitnehmen können (Beispiel).
  - Geben Sie Ihre Lösungen im entsprechend gekennzeichneten orangenen Kasten auf dem D3-Flur ab.
  - Sie erhalten die korrigierten Lösungen in den Übungen zurück. Nicht abgeholte Lösungen finden Sie auf dem orangenen Schrank im D3-Flur.
  - Bei abgeschriebenen Lösungen werden wir sowohl Abschrift als auch das Original mit 0 Punkten bewerten. Bei wiederholtem Abschreiben werden Sie vom Bonusnotenverfahren ausgeschlossen.

1. Einführung in graphische Benutzungsoberflächen

Graphische Benutzungsoberflächen (graphical user interfaces, **GUI**) dienen zur

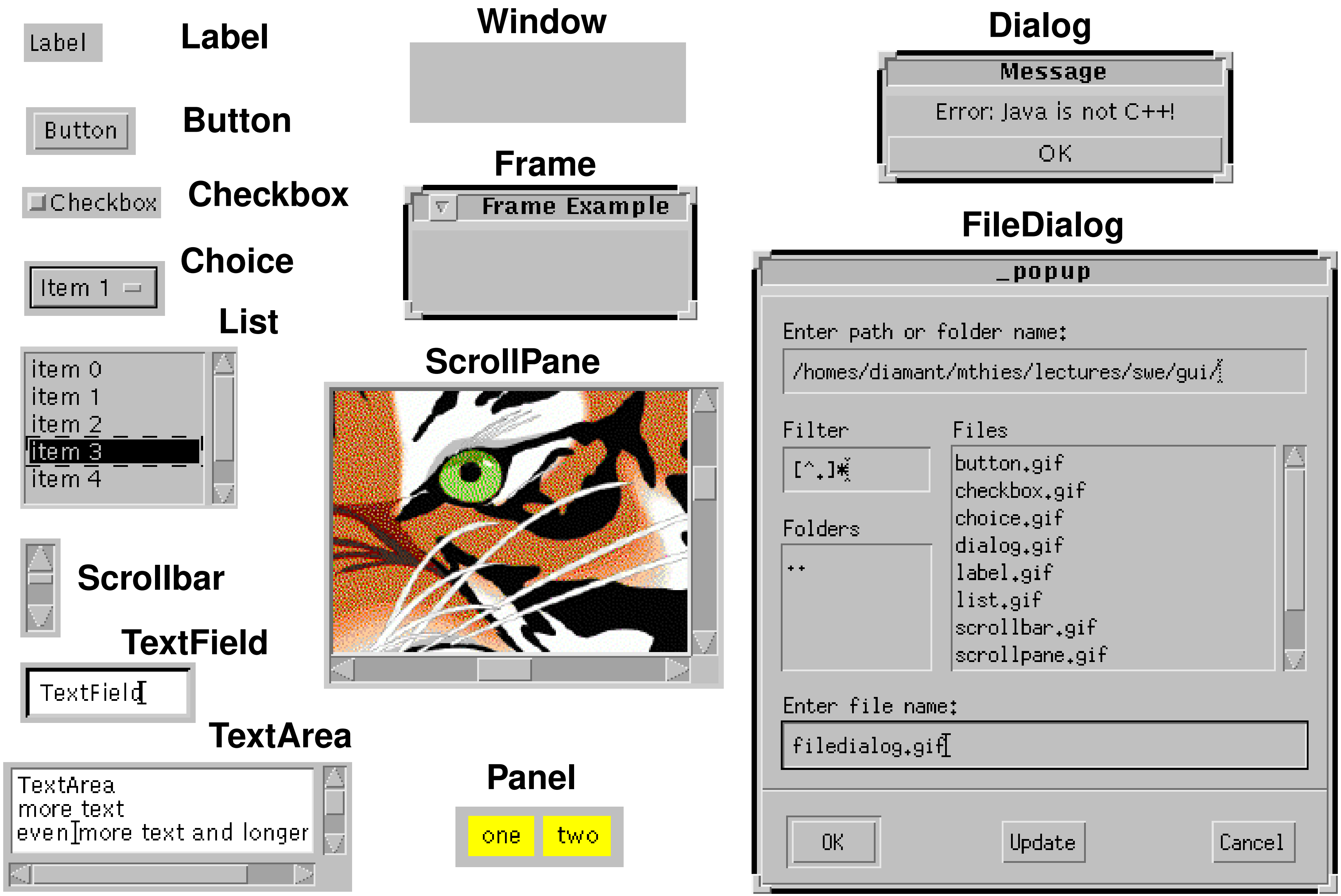
- interaktiven Bedienung von Programmen,
- Ein- und Ausgabe mit graphischen Techniken und visuellen Komponenten



Javas Standardbibliothek `jawa.awt`
(abstract windowing toolkit) enthält wiederverwendbare Klassen zur Implementierung und Benutzung der wichtigsten GUI-Komponenten:

- Graphik
- GUI-Komponenten (siehe SWE-84)
- Plazierung, Layoutmanager
- Ereignisbehandlung (**`java.awt.event`**)
- Bildmanipulation

Graphische Darstellung von AWT-Komponenten



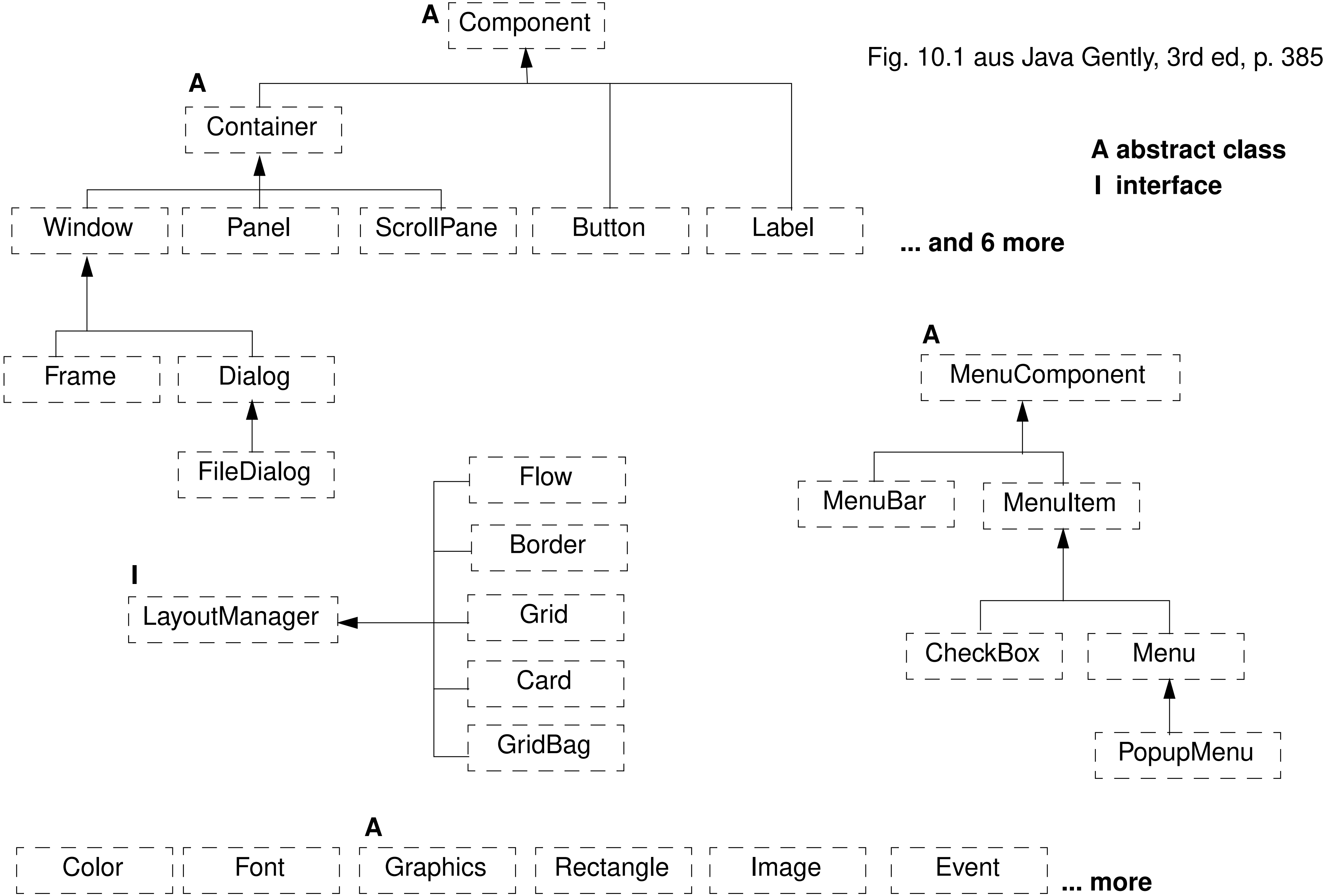
Klassenhierarchie für Komponenten von Benutzungsoberflächen

Teil der Standardbibliothek **`java.awt`** (abstract windowing toolkit)

Klasse in der Hierarchie	Kurzbeschreibung
Component (abstract)	darstellbare Komponenten von Benutzungsoberflächen
Container (abstract)	Behälter für Komponenten
Panel	konkrete Klasse zu Container, Behälter für Komponenten
ScrollPane	Sicht auf große Komponente, 2 Rollbalken
Window	Fenster (ohne Rand, Titel, usw.); Wurzel der Objektbäume
Dialog	Fenster; fordert interaktive Bearbeitung
FileDialog	Fenster zur interaktiven Dateiauswahl
Frame	Fenster mit Rand, Titel, Menüleiste; veränderbare Größe
Button	Schaltknopf
Canvas	frei verwendbare rechteckige Zeichenfläche
CheckBox	An/Aus-Schalter
Choice	Auswahl aus einer Liste von Texten (ausklappbar)
List	Auswahl aus einer Liste von Texten (mit Rollbalken)
Scrollbar	Rollbalken zur Einstellung eines ganzzahligen Wertes
Label	Textzeile
TextComponent	edierbarer Text
TextField	einzelne Textzeile
TextArea	mehrzeiliger Text mit Rollbalken

Klassenhierarchie der AWT-Bibliothek

Fig. 10.1 aus Java Gently, 3rd ed, p. 385



Wiederverwendung von Klassen aus Bibliotheken

Die Klasse **`java.awt.Frame`** implementiert **gerahmte Fenster** in graphischen Benutzungsoberflächen (GUI). Sie ist eine Blattklasse in der Hierarchie der GUI-Komponenten:

`java.lang.Object`	
`java.awt.Component`	GUI-Komponenten
`java.awt.Container`	solche, die wieder Komponenten enthalten können
`java.awt.Window`	Fenster ohne Rahmen
`java.awt.Frame`	Fenster mit Rahmen

Methoden zum Zeichnen, Platzieren, Bedien-Ereignisse Behandeln, etc. sind auf den jeweils passenden Hierarchieebenen implementiert.
In der **`abstract class Component`** ist die Methode

```
public void paint (Graphics g)
```

definiert, aber nicht ausgefüllt. Mit ihr wird auf der Fläche des Fensters gezeichnet.

Benutzer definieren Unterklassen von `Frame`. Sie erben die Funktionalität der Oberklassen.
Die Methode **`paint`** **wird überschrieben** mit einer Methode, die das Gewünschte zeichnet:

```
public class Rings extends Frame
{  public Rings () { setTitle ("Olympic Rings"); }
   public void paint (Graphics g) { // draw olympic rings ... }
   public static void main (String [] args)
   {  Frame f = new Rings(); ... }
}
```

Einige Eigenschaften auf den Ebenen der `Frame`-Hierarchie

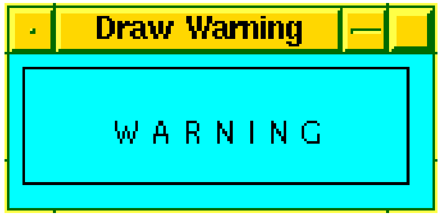
Klasse	Datenelemente	Ereignisse	Methoden
Component	Location, Size, Bounds, Visible	Key, Mouse, Focus, Component	paint
Container	Layout	Container	add, getComponents, paint
Window	Locale	Window	show, pack, toBack, toFront
Frame	Title, MenuBar, Resizable, IconImage		

Namenskonventionen:

zum Datenelement XXX
gibt es die Methoden
getXXX und ggf. setXXX

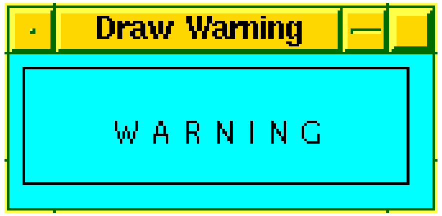
Ereignisse sind Objekte
der Klassen YYYEvent

2. Zeichenflächen benutzen, Programmschema



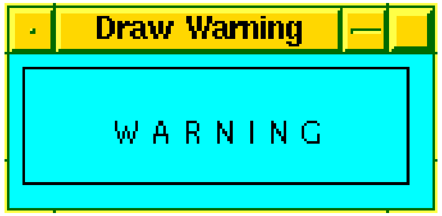

```
import java.awt.*; import java.awt.event.*;
          // Hauptklasse als Unterklasse von Frame:
public class GraphicWarning extends Frame
{  GraphicWarning (String title)                          // Konstruktor
   {  super (title);                          // Aufruf des Konstruktors von Frame









   }

   public void paint (Graphics g)     // überschreibt paint in einer Oberklasse
   {

   }

   public static void main (String[] args)
   {  Frame f = new GraphicWarning ("Draw Warning");  // Objekt erzeugen
   }
}
```

Programmschema: Eigenschaften und Ereignisbehandlung



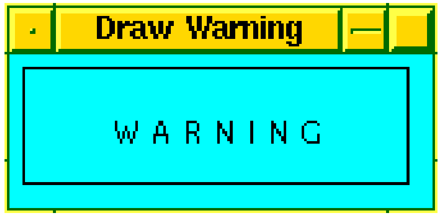

```
import java.awt.*; import java.awt.event.*;
         // Hauptklasse als Unterklasse von Frame:
public class GraphicWarning extends Frame
{  GraphicWarning (String title)                           // Konstruktor
   {  super (title);                  // Aufruf des Konstruktors von Frame
      setBackground (Color.cyan);
      setSize (35*letter,6*line);                          // Größe festlegen
      addWindowListener             // Ereignisbehandlung: Beim Drücken des
         (new WindowAdapter ()        // Schließknopfes Programm beenden.
            { public void windowClosing(WindowEvent e)
               { System.exit(0); } });
      setVisible (true);               // führt zu erstem Aufruf von paint
   }
   static private final int line = 15, letter = 5;    // zur Positionierung
   public void paint (Graphics g)    // überschreibt paint in einer Oberklasse
   {
   }

   public static void main (String[] args)
   {  Frame f = new GraphicWarning ("Draw Warning");   // Objekt erzeugen
   }
}
```

Programmschema: `paint`-Methode ausfüllen




```
import java.awt.*; import java.awt.event.*;
          // Hauptklasse als Unterklasse von Frame:
public class GraphicWarning extends Frame
{  GraphicWarning (String title)                          // Konstruktor
   {  super (title);                       // Aufruf des Konstruktors von Frame
      setBackground (Color.cyan);
      setSize (35*letter,6*line);                          // Größe festlegen
      addWindowListener                // Ereignisbehandlung: Beim Drücken des
         (new WindowAdapter ()            // Schließknopfes Programm beenden.
            { public void windowClosing(WindowEvent e)
               { System.exit(0); } });
      setVisible (true);                       // führt zu erstem Aufruf von paint
   }
   static private final int line = 15, letter = 5;    // zur Positionierung
   public void paint (Graphics g)     // überschreibt paint in einer Oberklasse
   {  g.drawRect (2*letter, 2*line, 30*letter, 3*line);// auf der Fläche g
      g.drawString ("W A R N I N G", 9*letter, 4*line);  // zeichnen und
   }                                                          // schreiben

   public static void main (String[] args)
   {  Frame f = new GraphicWarning ("Draw Warning");   // Objekt erzeugen
   }
}
```

Ablauf des Zeichen-Programms



1. **`main`** aufrufen:
    1.1. **`GraphicWarning`**-Objekt erzeugen, Konstruktor aufrufen:
        1.1.1 **`Frame`**-Konstruktor aufrufen
        1.1.2 Eigenschaften setzen, z. B. Farben,
        1.1.3 ggf. weitere Initialisierungen des Fensters
        1.1.4 Größe festlegen, **`setSize (...)`**,
        1.1.5 Reaktion auf Drücken des Schließknopfes vorbereiten **`addWindowListener`**,
        1.1.6 Fenster sichtbar machen, **`setVisible(true)`**
            parallele Ausführung von (2) initiieren
    1.2 Objekt an **`f`** zuweisen
    1.3 ggf. weitere Anweisungen zur Programmausführung

in Methoden der Oberklassen:

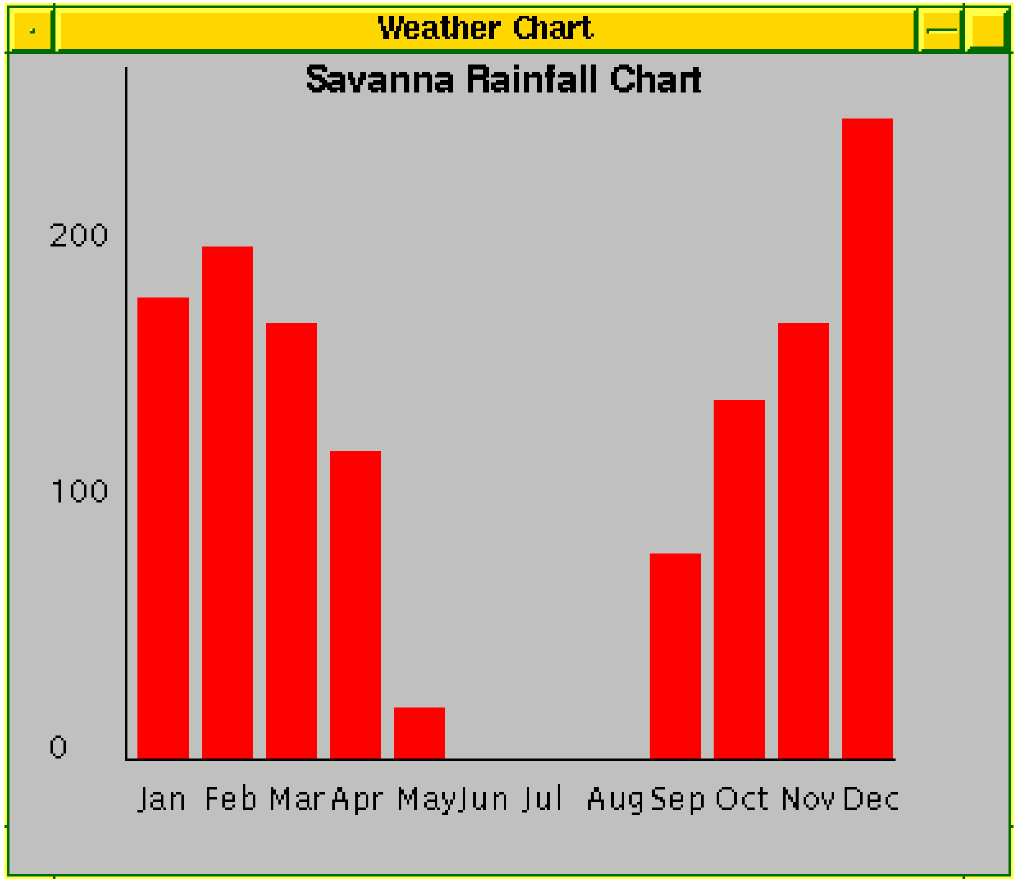
2. **`Graphics`** Objekt erzeugen
    2.1 damit erstmals **`paint`** aufrufen (immer wieder, wenn nötig):
    weiter in **`paint`** von **`GraphicWarning`**
        2.1.1 auf der Zeichenfläche des Parameters **`g`** schreiben und zeichnen

3. Schließknopf Drücken

in Methoden der Oberklassen:

    3.1 **`windowClosing`** des **`WindowListener`** aufrufen
        3.1.1 Programm beenden, **`exit(0)`**

Beispiel: Balkendiagramm zeichnen



Beispiel: Balkendiagramm zeichnen

```
public void paint (Graphics g)
{  int x = 50, y = 300;                    // Schnittpunkt der Achsen
   int width = 20, gap = 5;                // Balken und Zwischenraum

   g.drawLine (x, y, x+12*(width+gap), y);                 // x-Achse
   g.drawLine (x, y, x, 30);                               // y-Achse

   for (int m = 0; m < 12; m++)              // Monate an der x-Achse
      g.drawString(Months[m], m*(width+gap)+gap+x, y+20);

   for (int i = 0; i <y; i+=100)              // Werte an der y-Achse
      g.drawString(String.valueOf(i), 20, y-i);

   g.setFont(new Font("SansSerif", Font.BOLD, 14));    // Überschrift
   g.drawString("Savanna Rainfall Chart", 120, 40);

   g.setColor(Color.red);                                // die Balken
   for (int month = 0; month < 12; month++)
   {  int a = (int) rainTable[month]*10;
      g.fillRect(month*(width+gap)+gap+x, y-a, width, a);
   }
}
static double[] rainTable = new double[12];
static String Months [] =
   {"Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"};
```

3. AWT-Komponenten erzeugen und platzieren

Ein einfaches Beispiel für Text und Schaltknöpfe:

Aufgaben zur Herstellung:

Aussehen:

- Label- und Button-Objekte generieren
- Anordnung der Komponenten festlegen

Ereignisse:

- ein Listener-Objekt mit den Buttons verbinden
- call-back-Methode für Buttons implementieren

Komponenten platzieren

Die Klasse **`Container`** (Oberklasse von **`Frame`**) sorgt für die Platzierung der Komponenten, die ein **`Container`**-Objekt enthält.

Dazu wird für den **`Container`** ein **`LayoutManager`** installiert; z. B. mit folgender Anweisung im Konstruktor der Unterklasse von **`Frame`**:

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

Er bestimmt die Anordnung der Komponenten nach einer speziellen Strategie;
z. B. **`FlowLayout`** ordnet zeilenweise an.

Komponenten werden generiert und mit der Methode **`add`** dem **`Container`** zugefügt, z. B.

```
add (new Label ("W A R N I N G")); ...

Button waitButton = new Button ("Wait"); add (waitButton); ...
```

Die Reihenfolge der **`add`**-Aufrufe ist bei manchen **`LayoutManager`**n relevant.

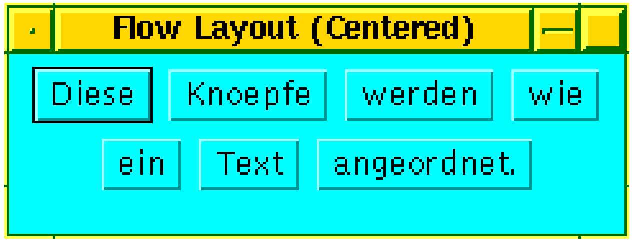
Wird die Gestalt des **`Container`**s verändert, so ordnet der **`LayoutManager`** die Komponenten ggf. neu an.

Programmschema zum Platzieren von Komponenten

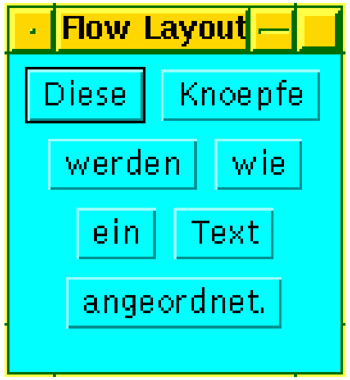
```
import java.awt.*; import java.awt.event.*;

class FlowTest extends Frame                          // Definition der Fenster-Klasse
{  FlowTest ()                                                      // Konstruktor
   {  setBackground(Color.cyan);            // Eigenschaften des Fensters setzen

      setLayout(new FlowLayout(FlowLayout.CENTER));        // LayoutManager

      add(new Button("Diese")); ...                    // Komponenten zufügen

      setTitle("Flow Layout (Centered)"); setSize(250,100);
      setVisible(true);
   }
}

public class LayoutTry
{  public static void main(String[] args)
   { Frame f = new FlowTest();                 // Ein FlowTest-Objekt erzeugen
      WindowListener winListen =                         // Ereignisbehandlung
         new WindowAdapter ()
         {  public void windowClosing(WindowEvent e)
            {  System.exit(0); }
         };
      f.addWindowListener (winListen);
   }
}
```

LayoutManager FlowLayout

```
class FlowTest extends Frame
{  FlowTest ()
   {  setBackground(Color.cyan); setForeground(Color.black);

      setLayout(new FlowLayout(FlowLayout.CENTER));

      add(new Button("Diese")); add(new Button("Knoepfe"));
      add(new Button("werden")); add(new Button("wie"));
      add(new Button("ein")); add(new Button("Text"));
      add(new Button("angeordnet."));

      setTitle("Flow Layout (Centered)"); setSize(250,100);
      setVisible(true);
   }
}
```

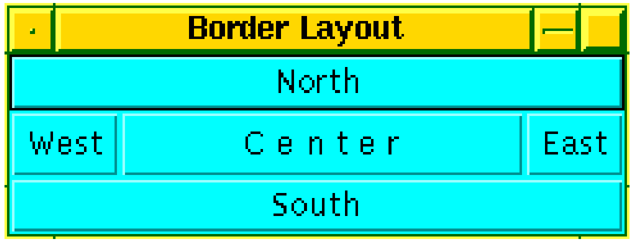


nach
Ändern
der Form:

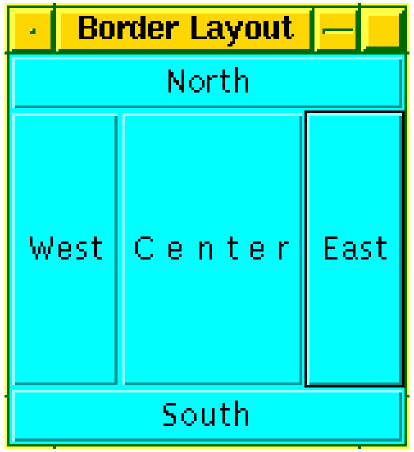


LayoutManager BorderLayout

```
class BorderTest extends Frame
{  BorderTest ()
   {  setBackground(Color.cyan); setForeground(Color.black);

      setLayout(new BorderLayout());

      add(new Button("North"), BorderLayout.NORTH);
      add(new Button("East"), BorderLayout.EAST);
      add(new Button("South"), BorderLayout.SOUTH);
      add(new Button("West"), BorderLayout.WEST);
      add(new Button("C e n t e r"), BorderLayout.CENTER);

      setTitle("Border Layout"); setSize(250,100); setVisible(true);
   }
}
```

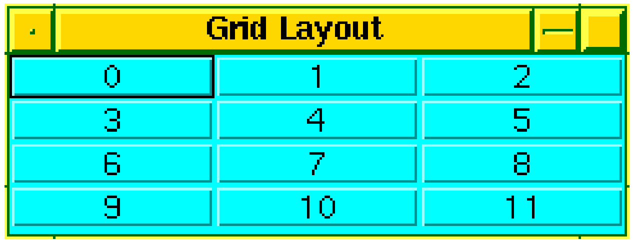


nach
Ändern
der Form:

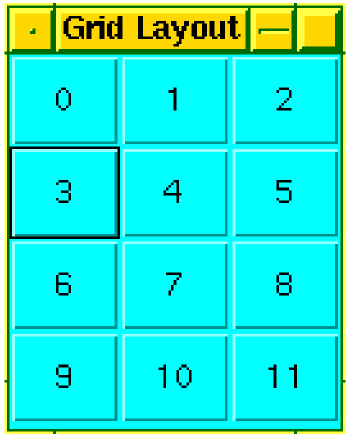


LayoutManager GridLayout

```
class GridTest extends Frame
{  GridTest ()
   {  setBackground(Color.cyan); setForeground(Color.black);

      setLayout(new GridLayout(4, 3));

      for (int i = 0; i < 12; i++)
         add(new Button((new Integer(i)).toString()));

      setTitle("Grid Layout"); setSize(250,100); setVisible(true);
   }
}
```

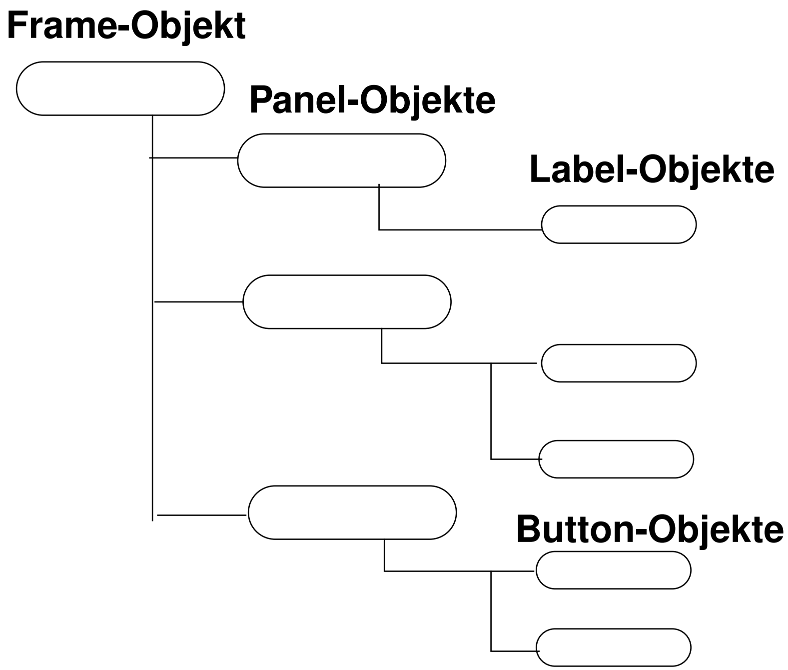


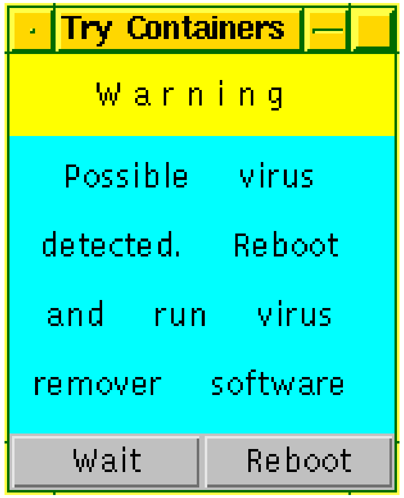
nach
Ändern
der Form:

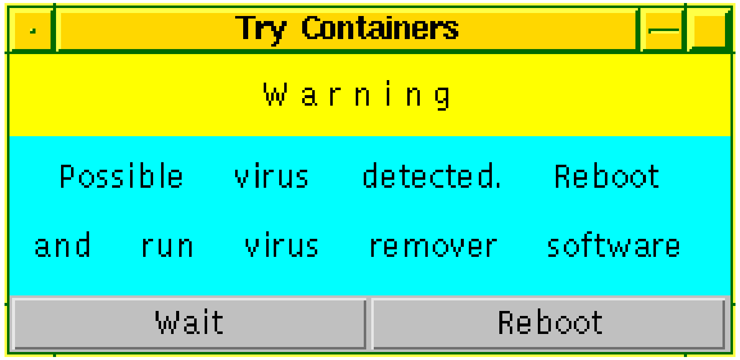


4. Hierarchisch strukturierte Fensterinhalte

- **Zusammengehörige Komponenten** in einem Objekt einer **`Container`**-Unterklasse unterbringen (**`Panel`**, **`Window`**, **`Frame`** oder selbstdefinierte Unterklasse).
- Eigenschaften der Objekte im **`Container`** können dann gemeinsam bestimmt werden, z. B. Farbe, **`LayoutManager`**, usw. zuordnen.
- Mit **`Container`**-Objekten werden beliebig tiefe **Baumstrukturen** von AWT-Komponenten erzeugt. In der visuellen Darstellung sind sie **ineinander geschachtelt**.







Programm zu hierarchisch strukturiertem Fenster

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

class LabelContainer extends Container         // Klasse zur Herstellung von
{  LabelContainer (String[] words)             // Fließtext aus String-Array
   {  setLayout(new FlowLayout(FlowLayout.CENTER));
      for (int i = 0; i < words.length; i++)
         add (new Label (words[i]));
   }
}

class LayoutComp                       // Hauptklasse erzeugt den Komponentenbaum
{  public static void main(String[] args)
   {  String [] message = {"Possible", "virus", "detected.", "Reboot",
                           "and", "run", "virus", "remover", "software"};
      Container warningText = new LabelContainer (message);

      ... Erzeugung des Komponentenbaumes einfügen ...

      WindowListener winListen = new WindowAdapter ()
      { public void windowClosing(WindowEvent e) { System.exit(0); }};
      rootFrame.addWindowListener (winListen);
      rootFrame.setLocation (100, 100); rootFrame.setSize (160, 200);
      rootFrame.setVisible(true);
   }
}
```