Komponentenbaum erzeugen

```
      // Text der Warning im Array message zusammengefasst auf LabelContainer:
Container warningText = new LabelContainer (message);

                              // Überschrift als Panel mit einem zentrierten Label:
Panel header = new Panel();
header.setBackground(Color.yellow);
header.setLayout(new FlowLayout(FlowLayout.CENTER));
header.add (new Label ("W a r n i n g"));

                                    // Knöpfe im Panel mit GridLayout:
Panel twoButtons = new Panel ();
twoButtons.setBackground(Color.lightGray);
twoButtons.setLayout (new GridLayout(1, 2));
twoButtons.add (new Button ("Wait"));
twoButtons.add (new Button ("Reboot"));

                                    // im Frame mit BorderLayout zusammenfassen:
Frame rootFrame = new Frame ("Try Containers");
rootFrame.setBackground(Color.cyan);
rootFrame.setLayout(new BorderLayout());
rootFrame.add(header, BorderLayout.NORTH);
rootFrame.add(warningText, BorderLayout.CENTER);
rootFrame.add(twoButtons, BorderLayout.SOUTH);
```

Vorlesung Software-Entwicklung II SS 2004 / Folie 95a

Ziele:

Programmierung hierarchischer Fensterstrukturen

in der Vorlesung:

Erläuterung zusammen mit Folie 94, Folie 95

nachlesen:

Judy Bishop: Java lernen, 2.Aufl., Abschnitt 10.4

Übungsaufgaben:

Verständnisfragen:

- Bei welchen Anweisungen bewirkt ein Vertauschen eine Veränderung der Fensterdarstellung.

5. Ereignisse an graphischen Benutzungsoberflächen

Interaktion zwischen Bediener und Programmausführung über **Ereignisse (events)**:

- **Bedien-Operationen lösen Ereignisse aus**, z. B. Knopf drücken, Alternative auswählen, Mauszeiger auf eine graphisches Element schieben.
- **Programmausführung reagiert auf solche Ereignisse** durch Aufruf bestimmter Methoden

Aufgaben:

- **Folgen von Ereignissen** und Reaktionen darauf **planen und entwerfen**
  Modellierung z. B. mit endlichen Automaten oder StateCharts
- **Reaktionen auf Ereignisse** systematisch implementieren
  AWT: Listener-Konzept; Entwurfsmuster „Observer“

Vorlesung Software-Entwicklung II SS 2004 / Folie 96

Ziele:

Einstieg in die Programmierung mit Ereignissen

in der Vorlesung:

Erläuterung der Begriffe und Aufgaben

nachlesen:

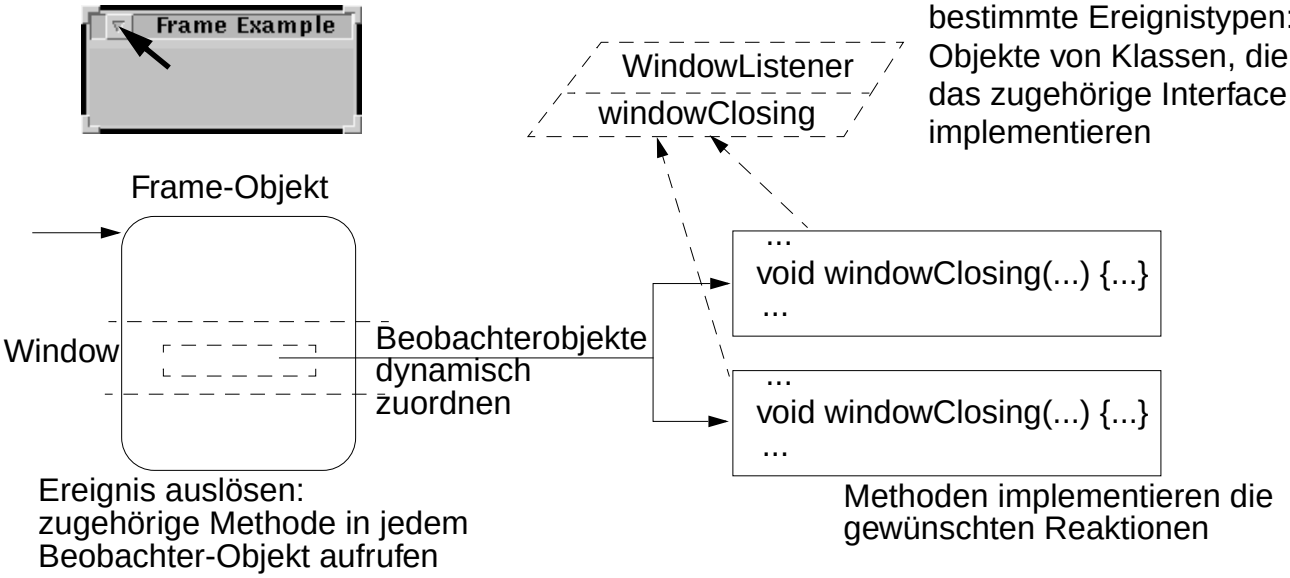
Judy Bishop: Java lernen, 2.Aufl., Abschnitt 11.1

Übungsaufgaben:

Verständnisfragen:

„Observer“-Prinzip in der Ereignisbehandlung

An AWT-Komponenten werden Ereignisse ausgelöst, z. B. ein WindowEvent an einem Frame-Objekt:



Beobachter (LIstener) für bestimmte Ereignistypen: Objekte von Klassen, die das zugehörige Interface implementieren

Entwurfsmuster „Observer“: Unabhängigkeit zwischen den Beobachtern und dem Gegenstand wegen Interface und dynamischem Zufügen von Beobachtern.

Vorlesung Software-Entwicklung II SS 2004 / Folie 97

Ziele:

Entwurfsmuster Observer zur Ereignisbehandlung kennenlernen

in der Vorlesung:

Erläuterung von Gegenstand und Beobachter auf der Ebene

- der Klassen
- der Objekte
- der Reaktion auf Ereignisse.
- Synonyme: Observer (Entwurfsmuster), Listener (AWT-Bibliothek), Beobachter (Deutsch)

nachlesen:

Judy Bishop: Java lernen, 2.Aufl., Abschnitt 11.2

Übungsaufgaben:

Verständnisfragen:

- Begründen Sie weshalb größere Unabhängigkeit durch das Entwurfsmuster erzielt wird.

Programmiertechnik für `Listener`

Im **`java.awt`** Package gibt es zu jedem Ereignistyp **`XXXEvent`** ein Interface **`XXXListener`**:

```
public interface WindowListener extends EventListener
{ void windowActivated (WindowEvent); void WindowClosed (WindowEvent);
  void windowClosing (WindowEvent);...void windowOpened (WindowEvent);
}
```

Eine abstrakte Klasse **`XXXAdapter`** mit leeren Methodenimplementierungen:

```
public abstract class WindowAdapter implements WindowListener
{  public void windowActivated (WindowEvent) { } ...
   public void windowOpened (WindowEvent) { }
}
```

Anwendungen, die nicht auf alle Sorten von Methodenaufrufen des Interface reagieren, deklarieren eine Unterklasse und **überschreiben die benötigten Methoden des Adapters**, meist als innere Klasse, um den Zustand eines Objektes zu verändern:

```
class WClose extends WindowAdapter
{  public void windowClosing (WindowEvent e)
   { System.exit (0); }
}
```

Zufügen eines Listener-Objektes zu einer AWT-Komponente:

```
f.addWindowListener (new WClose ());
```

Vorlesung Software-Entwicklung II SS 2004 / Folie 98

Ziele:

Programmiertechnik zum Konzept der Listener

in der Vorlesung:

- Rolle von Interface und Adapter-Klasse
- Zuordnung des Listener-Objektes

erläutern

nachlesen:

Judy Bishop: Java lernen, 2.Aufl., Abschnitt 11.2

Übungsaufgaben:

Überschreiben Sie weitere Methoden des Interface mit einfachen Reaktionen und erproben Sie sie.

Verständnisfragen:

Innere Klassen

Innere Klassen können z. B. als Hilfsklassen zur Implementierung der umgebenden Klasse verwendet werden:

```
class List { ... static class Node { ...} ...}
```

Die **`List`**-Objekte und **`Node`**-Objekte sind dann **unabhängig voneinander**.

Es wird nur die Gültigkeit des Namens **`Node`** auf die Klasse **`List`** eingeschränkt.

In **inneren Klassen, die nicht `static`** sind, können Methoden der inneren Klasse auf Objektvariable der äusseren Klasse zugreifen. Ein Objekt der inneren Klasse ist dann immer in ein Objekt der äusseren Klasse eingebettet; z. B. die inneren **`Listener`** Klassen, oder auch:

```
interface Einnehmer { void bezahle (int n); }

class Kasse                    // Jedes Kassierer-Objekt eines Kassen-Objekts
{  private int geldSack = 0;                  // zahlt in denselben Geldsack

   class Kassierer implements Einnehmer
   {  public void bezahle (int n)
      {  geldSack += n; }
   }

   Einnehmer neuerEinnehmer ()
   {  return new Kassierer (); }
}
```

Vorlesung Software-Entwicklung II SS 2004 / Folie 98a

Ziele:

Zwei Arten innerer Klassen kennenlernen

in der Vorlesung:

- Die Klasse Node wird als Beispiel für eine Hilfsklasse gezeigt.
- Für das Beispiel "Kasse und Kassierer" wird an einem Programmstück gezeigt, wie ein Kassenobjekt Umgebung für Kassierer-Objekte ist.

nachlesen:

Judy Bishop: Java lernen, 2.Aufl., Abschnitt 8.2

Übungsaufgaben:

Schreiben Sie ein Programmstück, das 2 Kassen-Objekte erzeugt und zu jedem mehrerer Kassierer-Objekte. Geben Sie Aufrufe von "bezahle" für die Kassierer an. Zeigen Sie an Objektdiagrammen die Verbindung der Objekte und die Wirkung der Aufrufe.

Verständnisfragen:

Anonyme Klasse

Meist wird zu der Klasse, mit der Implementierung der Reaktion auf einen Ereignistyp **nur ein einziges Objekt** benötigt:

```
class WClose extends WindowAdapter
{  public void windowClosing (WindowEvent e)
   { System.exit (0); }
}
```

Zufügen eines **`Listener`**-Objektes zu einer AWT-Komponente:

```
f.addWindowListener (new WClose ());
```

Das läßt sich kompakter formulieren mit einer **anonymen Klasse**:
Die Klassendeklaration wird mit der **`new`**-Operation (für das eine Objekt) kombiniert:

```
f.addWindowListener
   (  new WindowAdapter ()
         {  public void windowClosing (WindowEvent e)
            { System.exit (0); }
         }
   );
```

In der **`new`**-Operation wird der **Name der Oberklasse** der deklarierten anonymen Klasse (hier: **`WindowAdapter`**) **oder Der Name des Interface**, das sie implementiert, angegeben!

Vorlesung Software-Entwicklung II SS 2004 / Folie 99

Ziele:

Java-Konstrukt "anonyme Klasse" verstehen

in der Vorlesung:

Erläuterung am Beispiel des WindowListeners

nachlesen:

Judy Bishop: Java lernen, 2.Aufl., Abschnitt 8.2, 11.2

Übungsaufgaben:

Variieren Sie das Beispiel, so dass

- mehrere WindowListener- Objekte ein Fenster-Objekt beobachten und dass
- ein WindowListener-Objekt mehrere Fenster-Objekte beobachtet

Verständnisfragen:

Reaktionen auf Buttons

AWT-Komponenten **`Button, List, MenuItem, TextField`** lösen **`ActionEvents`** aus. Sie werden von **`ActionListener`**-Objekten beobachtet, mit einer einzigen Methode:

```
public void actionPerformed (ActionEvent e) {...}
```

Beispiel der Virus-Warnung (Abweichung vom Stil im Buch Java Gently!):

```
import java.awt.*; import java.awt.event.*;
class ButtonTest extends Frame
{  private Button waitButton, rebootButton; int state = 0;
   ButtonTest (String title)
   {  ...
      waitButton.addActionListener     // Listener für den waitButton
         (new ActionListener ()      // anonyme Klasse direkt vom Interface
            {  public void actionPerformed(ActionEvent e)
               { state = 1; setBackground(Color.red); } });
      rebootButton.addActionListener // Listener für den rebootButton
         (new ActionListener ()
            {  public void actionPerformed(ActionEvent e)
               { state = 2; setVisible(false); System.exit (0);} });
} }
```

Die Aufrufe von **`setBackground`** und **`setVisible`** beziehen sich auf das umgebende **`ButtonTest`**-Objekt - nicht auf das unmittelbar umgebende **`ActionListener`**-Objekt.

Vorlesung Software-Entwicklung II SS 2004 / Folie 100

Ziele:

Button-Reaktionen im Zusammenhang des Beispiels

in der Vorlesung:

- Erläuterung der Beobachter-Klassen und -Objekte im Zusammenhang.
- Hinweis auf anonyme Klasse zu Interface;
- Einbettung der Klassen und ihrer Objekte;
- Abweichung vom Stil im Buch Java Gently, Java Lernen.

nachlesen:

Judy Bishop: Java lernen, 2.Aufl., Abschnitt 11.2

Eingabe von Texten

Komponente `TextField`: einzeiliger, editierbarer Text

Ereignisse: **`ActionEvent`** (wie bei **`Button`**) ausgelöst bei der Eingabe von **`return`**

einige Methoden (aus der Oberklasse TextComponent):

`String getText ()`	Textinhalt liefern
`void setText (String v)`	Textinhalt setzen
`void setEditable (boolean e)`	Edierbarkeit festlegen
`boolean isEditable ()`	Edierbarkeit
`void selectText (int start, int end)`	Zeichenbereich anzeigen

Typischer **`ActionListener:`**

```
addActionListener
   (new ActionListener ()
      {  public void actionPerformed (ActionEvent e)
         { str = ((TextField)e.getSource()).getText();
         } });
```

Eingabe von Zahlen: Text in eine Zahl konvertieren, Ausnahme abfangen:

```
try { age = Integer.parseInt (str); }
catch (NumberFormatException e) { ... }
```

Vorlesung Software-Entwicklung II SS 2004 / Folie 101

Ziele:

TextField als Eingabeelement

in der Vorlesung:

Erläuterungen der Ereignisbehandlung und der Methoden

nachlesen:

Judy Bishop: Java lernen, 2.Aufl., Abschnitt 11.1

Übungsaufgaben:

Stellen Sie die Eingabe der Programme aus dem Anfang der Vorlesung auf die sinnvolle Benutzung von AWT-Komponenten um.

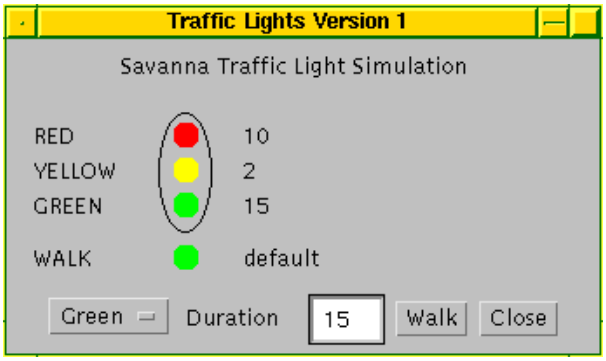
Verständnisfragen:

6. Beispiel: Ampel-Simulation

Aufgabe: Graphische Benutzungsoberfläche für eine Ampel-Simulation entwerfen

Eigenschaften:

- Ampel visualisieren
  mit Knopf und Licht für Fußgänger
  (später auch animieren)
- Phasenlängen der Lichter
  einzeln einstellbar
- Einstellungen werden angezeigt



Entwicklungsschritte:

- Komponenten strukturieren
- zeichnen der Ampel (**`paint`** auf **`Canvas`**)
- Komponenten generieren und anordnen
- Ereignisbehandlung entwerfen und implementieren

Vorlesung Software-Entwicklung II SS 2004 / Folie 102

Ziele:

Aufgabenstellung für ein Anwendungsbeispiel

in der Vorlesung:

Erläuterungen dazu.
Beispiel ausführen.

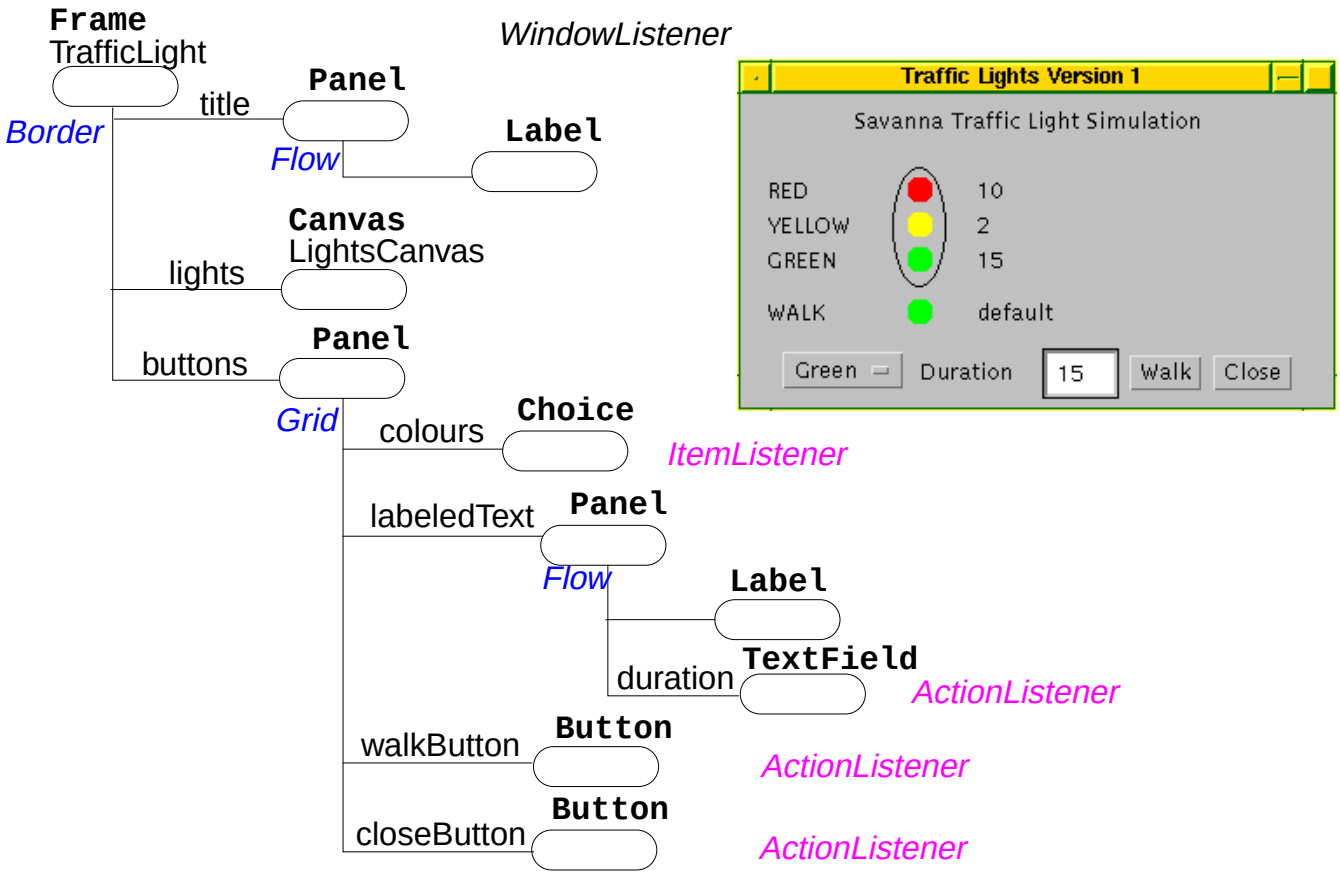
nachlesen:

Judy Bishop: Java lernen, 2.Aufl., Abschnitt 10.3 (Example 10.5), 11.4 (Example 11.2)

Übungsaufgaben:

Verständnisfragen:

Objektbaum zur Ampel-Simulation



Vorlesung Software-Entwicklung II SS 2004 / Folie 103

Ziele:

Struktur der Oberfläche entwerfen

in der Vorlesung:

- Elemente der Oberfläche erläutern
- Strukturentscheidungen begründen

nachlesen:

Judy Bishop: Java lernen, 2.Aufl., Abschnitt 11.4 (Example 11.2)

Übungsaufgaben:

Verständnisfragen:

Warum wird das Panel `labeledText` eingeführt?

Programm zur Ampel-Simulation

Im Konstruktor der zentralen Klasse wird der **Objektbaum** hergestellt:

```
class TrafficLight extends Frame
{                                  // The Traffic Light program by J M Bishop Oct 1997
                                   // revised by U Kastens Jan 1999

                                   // Objektvariable, auf die Listener zugreifen:
   private String [] message =     // Phasendauer für jede Lampe:
      {"default","default","default","default"};
   private int light = 0;          // Die ausgewählte Lampe
   private LightsCanvas lights;    // Enthält die gezeichnete Ampel

   public TrafficLight ()          // Konstruktor der zentralen Klasse
   {                               // Aufbau des Objektbaumes:
      setLayout (new BorderLayout ());   // Layout des Wurzelobjektes

      Panel title = new Panel ();         // Zentrierter Titel:
      title.setLayout (new FlowLayout (FlowLayout.CENTER));
      title.add (new Label("Savanna Traffic Light Simulation"));
      add (title, BorderLayout.NORTH);

                                  // Die Ampel wird in einer getrennt definierten Klasse gezeichnet:
      lights = new LightsCanvas (message);
      add (lights, BorderLayout.CENTER);
```

Vorlesung Software-Entwicklung II SS 2004 / Folie 104

Ziele:

Objektbaum konstruieren

in der Vorlesung:

- Objektvariable des Gegenstandsobjektes werden durch Objektmethoden der Listener verändert.
- Für gezeichnete Teile ist eine Unterklasse von Canvas nötig.
- Hinweis auf Unterschiede zum Programm im Buch Java Gently

nachlesen:

Judy Bishop: Java lernen, 2.Aufl., Abschnitt 11.4 (Example 11.2)

Übungsaufgaben:

Verständnisfragen:

Auswahl-Komponente

Auswahl-Komponenten (**Choice**) lösen **ItemEvent**s aus, wenn ein Element ausgewählt wird. Mit der Methode **itemStateChanged** kann ein **ItemListener** darauf reagieren:

```
Choice colours = new Choice (); // Choice of a light

colours.addItem ("Red"); colours.addItem ("Yellow");
colours.addItem ("Green"); colours.addItem ("Walk");

colours.addItemListener
   (new ItemListener ()
      {  public void itemStateChanged (ItemEvent e)
         {  String s = (String) e.getItem ();

            if (s.equals ("Red"))    {light = 0;} else
            if (s.equals ("Yellow")) {light = 1;} else
            if (s.equals ("Green"))  {light = 2;} else
            if (s.equals ("Walk"))   {light = 3;}
         }
      }
   );
```

Über den **Event**-Parameter kann man die Zeichenreihe des gewählten Elementes zugreifen.

Vorlesung Software-Entwicklung II SS 2004 / Folie 105

Ziele:

Anwendung der Choice-Komponente lernen

in der Vorlesung:

- Auf das Namensschema für Events hinweisen
- Informationsfluss über den Event-Parameter
- Zugriff auf die Objektvariable `light` erläutern
- Auf Abweichungen vom Buch Java Gently hinweisen

nachlesen:

Judy Bishop: Java lernen, 2.Aufl., Abschnitt 11.2 (Table 11.1)

Übungsaufgaben:

Verständnisfragen:

Zeichnen Sie Gegenstands- und Beobachterobjekt und -klassen und erläutern Sie daran die Zuweisung an die Objektvariable `light`.

Eingabe der Phasenlänge

Eingabe mit einem **`TextField`**. Reaktion auf ein **`ActionEvent`**:

```
Panel labeledText = new Panel ();//fasst TextField und Beschriftung zusammen

labeledText.setLayout (new FlowLayout (FlowLayout.LEFT));
labeledText.add(new Label("Duration"));

                                  // Eingabeelement für die Phasendauer einer Lampe:
TextField duration = new TextField("", 3);
duration.setEditable (true);

duration.addActionListener
   (new ActionListener ()
      {  public void actionPerformed(ActionEvent e)
         {                                  // Zugriff auf den eingegebenen Text:
         // message[light] = ((TextField)e.getSource).getText();
                                                        // oder einfacher:
            message[light] = e.getActionCommand ();
            lights.repaint();  // Die Zeichenmethode des gezeichneten Objektes
                               // wird erneut ausgeführt,
                               // damit der geänderte Text sichtbar wird.
      }  });

labeledText.add (duration);
```


Ziele:

Textein- und -ausgabe im Zusammenhang

in der Vorlesung:

- Informationsfluss vom Eingabeelement über den Event-Parameter zum Canvas-Objekt zeigen
- Auf Abweichungen vom Buch Java Lernen (Java Gently) hinweisen

nachlesen:

Judy Bishop: Java lernen, 2.Aufl., Abschnitt 11.4 (Example 11.2)

Übungsaufgaben:

Verständnisfragen:

Button-Zeile

Einfügen der **Button-Zeile** in den Objektbaum:

```
   Button walkButton = new Button ("Walk"); // noch keine Reaktion zugeordnet

   Button closeButton = new Button ("Close");
   closeButton.addActionListener
      (new ActionListener ()                          // Programm beenden:
         {  public void actionPerformed(ActionEvent e)
            { setVisible (false); System.exit (0); }
         }
      );
                                                      // Zusammensetzen der Button-Zeile:
   Panel buttons = new Panel();
   buttons.setLayout (new FlowLayout (FlowLayout.CENTER));
   buttons.add (colours);
   buttons.add (labeledText);
   buttons.add (walkButton);
   buttons.add (closeButton);

   add (buttons, BorderLayout.SOUTH);
}// TrafficLight Konstruktor

}// TrafficLight Klasse
```


Ziele:

Die Zentrale Klasse vervollständigen

in der Vorlesung:

Erläuterungen dazu

nachlesen:

Judy Bishop: Java lernen, 2.Aufl., Abschnitt 11.4 (Example 11.2)

Übungsaufgaben:

Verständnisfragen:

Ampel zeichnen und beschriften

Eine Unterklasse der AWT-Klasse **`Canvas`** stellt eine **Zeichenfläche** bereit.
Die Methode **`paint`** wird zum Zeichnen und Beschriften überschrieben:

```
class LightsCanvas extends Canvas
{  private String [] msg;

   LightsCanvas (String [] m)              // Die Array-Elemente enthalten die
   { msg = m; }                          // Phasendauern der Lampen. Sie können
                                         // durch Eingaben verändert werden.
   public void paint (Graphics g)
   {  g.drawOval (87, 10, 30, 68);         // Ampel zeichnen und beschriften
      g.setColor (Color.red);    g.fillOval (95, 15, 15, 15);
      g.setColor (Color.yellow); g.fillOval (95, 35, 15, 15);
      g.setColor (Color.green);  g.fillOval (95, 55, 15, 15);
      g.fillOval (95, 85, 15, 15);            // walk Lampe ist auch grün

      g.setColor(Color.black);
      g.drawString ("RED", 15 ,28); g.drawString ("YELLOW", 15, 48);
      g.drawString ("GREEN", 15, 68); g.drawString ("WALK", 15, 98);
                                    // eingegebenen Phasendauern der Lampen:
      g.drawString (msg[0], 135 ,28); g.drawString (msg[1], 135, 48);
      g.drawString (msg[2], 135, 68); g.drawString (msg[3], 135, 98);
} }
```


Ziele:

Auf einer Canvas-Komponente zeichnen

in der Vorlesung:

Erläuterungen zum Zeichnen und zur Änderung der Beschriftung

nachlesen:

Judy Bishop: Java lernen, 2.Aufl., Abschnitt 11.4 (Example 11.2)

Übungsaufgaben:

Implementieren Sie das Programm und experimentieren Sie damit.

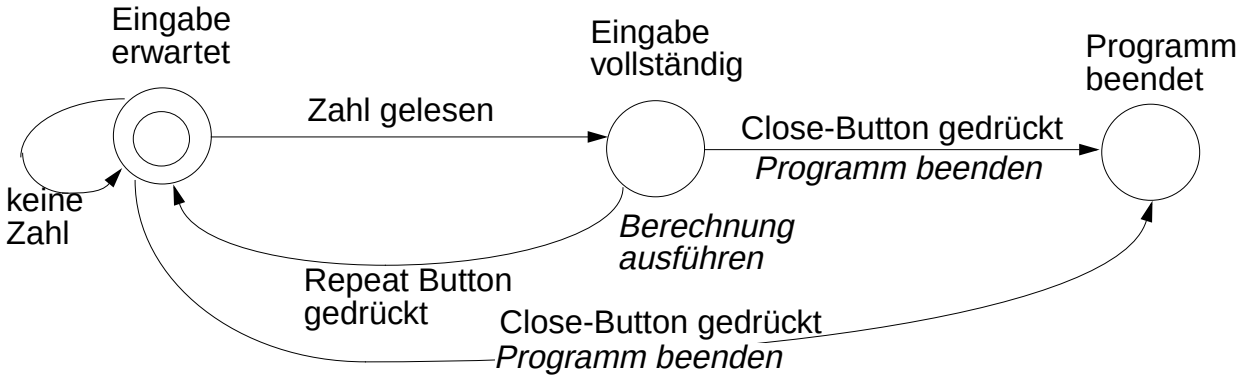
Verständnisfragen:

7. Entwurf von Ereignisfolgen

Die zulässigen **Folgen von Bedienereignissen und Reaktionen** darauf müssen für komplexere Benutzungsoberflächen geplant und entworfen werden.

Modellierung durch **endliche Automaten** (auch durch StateCharts)

- **Zustände** unterscheiden Bediensituationen (z. B. „Eingabe erwartet“, „Eingabe vollständig“)
- **Übergänge** werden durch Ereignisse ausgelöst.
- **Aktionen** können mit Übergängen verknüpft werden; Reaktion auf ein Ereignis z. B. bei Eingabe einer Phasenlänge Ampel neu zeichnen, und
- **Aktionen** können mit dem Erreichen eines Zustandes verknüpft werden, z. B. wenn die Eingabe vollständig ist, Berechnung beginnen.



Vorlesung Software-Entwicklung II SS 2004 / Folie 109

Ziele:

Notwendigkeit zur Modellierung erkennen

in der Vorlesung:

Erläuterungen zu endlichen Automaten an dem Beispiel

nachlesen:

Judy Bishop: Java lernen, 2.Aufl., Abschnitt 11.4

nachlesen:

Vorlesung Modellierung

Übungsaufgaben:

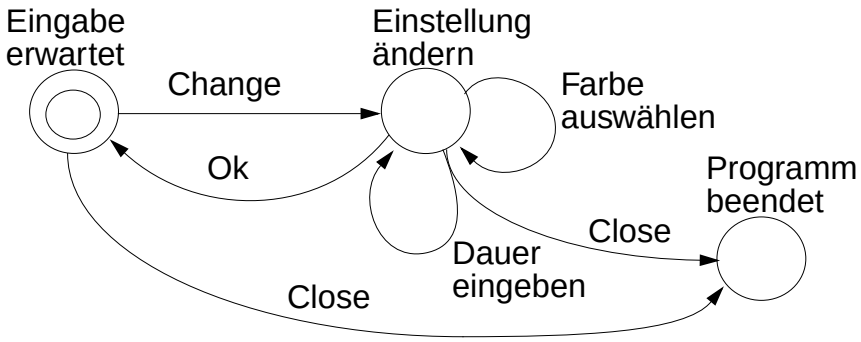
Verständnisfragen:

- Geben Sie zu dem Beispiel zulässige und unzulässige Ereignisfolgen an.

Unzulässige Übergänge

In manchen Zuständen sind **einige Ereignisse nicht als Übergang definiert**.
Sie sind in dem Zustand **unzulässig**, z. B. „Farbe auswählen“ im Zustand „Eingabe erwartet“.

Beispiel: Ampel-Simulation erweitert um zwei Buttons **Change** und **Ok**:



Robuste Programme dürfen auch an unzulässigen Ereignisfolgen nicht scheitern.
Verschiedene Möglichkeiten für **nicht definierte Übergänge**:

- Sie bleiben **ohne Wirkung**
- Sie bleiben ohne Wirkung und es wird eine **Erklärung** gegeben (Warnungsfenster).
- **Komponenten** werden so **deaktiviert**, dass unzulässige Ereignisse nicht ausgelöst werden können.

Vorlesung Software-Entwicklung II SS 2004 / Folie 110

Ziele:

Sinnvolle Ereignisfolgen planen

in der Vorlesung:

Erläuterung am Beispiel

nachlesen:

Judy Bishop: Java lernen, 2.Aufl., Abschnitt 11.4

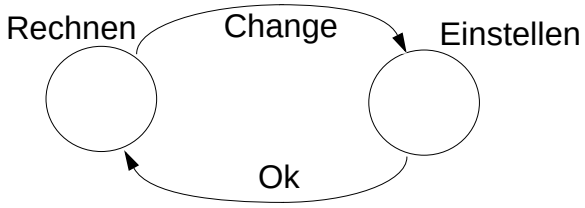
Übungsaufgaben:

Verständnisfragen:

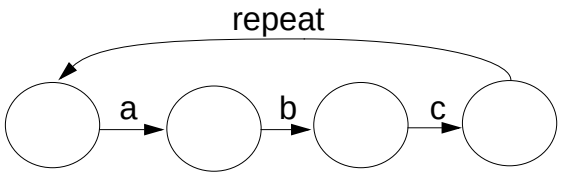
- Geben Sie die unzulässigen Paare (Zustand, Ereignis) an.
- In welchen Zuständen oder bei welchen Übergängen muss man welche Komponenten deaktivieren?

Muster für Ereignisfolgen

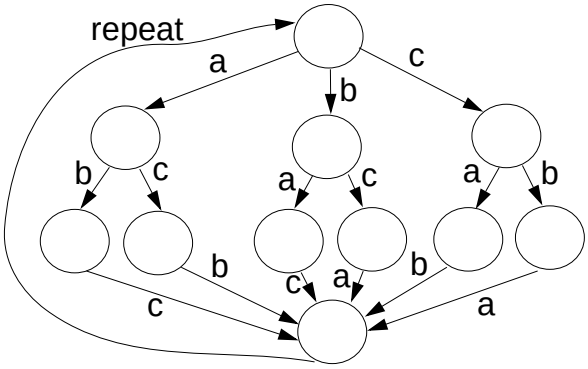
- Verschiedene **Bedienungsarten** (mode): Vorsicht: Nicht unnötig viele Zustände entwerfen. „Don‘t mode me in!“



- Festgelegte **sequentielle Reihenfolge**: Vorsicht: Nicht unnötig streng vorschreiben. Bediener nicht gängeln.



- **Beliebige Reihenfolge** von Ereignissen: Modellierung mit endlichem Automaten ist umständlich (Kombinationen der Ereignisse); einfacher mit StateCharts.



- **Voreinstellungen** (defaults) können Zustände sparen und Reihenfolgen flexibler machen. Vorsicht: Nur sinnvolle Voreinstellungen.

Vorlesung Software-Entwicklung II SS 2004 / Folie 111

Ziele:

Einige Regeln zum Entwurf von Ereignisfolgen

in der Vorlesung:

Erläuterungen und Beispiele dazu

nachlesen:

Vorlesungen: "Modellierung / (Endliche Automaten)", "Modellierung von Benutzungsschnittstellen" und "Implementation von Benutzungsschnittstellen"

Übungsaufgaben:

Verständnisfragen:

Implementierung des Ereignis-Automaten

Zustände ganzzahlig **codieren**; **Objektvariable** speichert den **augenblicklichen Zustand**:

```
private int currentState = initialState;
private final int initialState = 0, settingState = 1, ...;
```

Einfache **Aktionen der Übergänge** bleiben in den Reaktionsmethoden der **`Listener`**; Methodenaufruf für den Übergang in einen neuen Zustand zufügen:

```
changeButton.addActionListener
   (new ActionListener ()
      {  public void actionPerformed(ActionEvent e)
         { toState (settingState); }});
```

Aktionen der Zustände in **Übergangsmethode** plazieren, z. B. Komponenten (de)aktivieren:

```
private void toState (int nextState)
{  currentState = nextState;
   switch (nextState)
   {  case initialState:
         lights.repaint();
         okButton.setEnabled(false); changeButton.setEnabled (true);
         colours.setEnabled (false); duration.setEnabled (false);
         break;
      case settingState: ...
         break;
      default: ...
} }
```

Vorlesung Software-Entwicklung II SS 2004 / Folie 112

Ziele:

Systematische Implementierungstechnik

in der Vorlesung:

- Am Beispiel der Ampel-Simulation erläutern.
- Plazierung der Aktionen begründen.
- `switch`-Anweisung erklären
- Aktivierung und Deaktivierung von Komponenten zeigen (Folie 112a).

nachlesen:

Judy Bishop: Java lernen, 2.Aufl., Abschnitt 11.4

Übungsaufgaben:

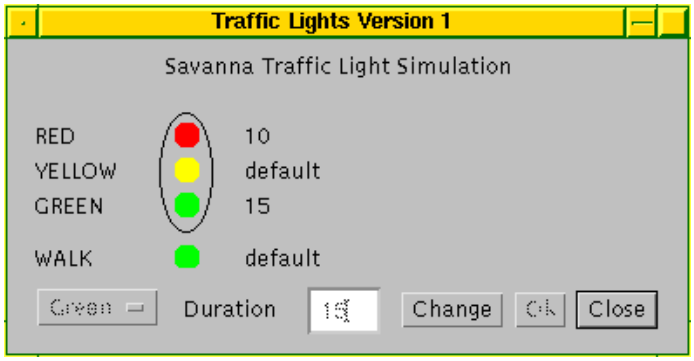
Implementieren und erproben Sie das Programm.

Verständnisfragen:

- Begründen Sie, wann Aktionen den Übergängen bzw. den Zuständen zugeordnet werden.

Ampel-Simulation mit kontrollierten Zustandsübergängen

Zwei Knöpfe wurden zugefügt:
Der **Change-Button** aktiviert die Eingabe, der **Ok-Button** schliesst sie ab.



Die Komponenten zur Farbauswahl, Texteingabe und der Ok-Knopf sind im gezeigten Zustand deaktiviert.

Vorlesung Software-Entwicklung II SS 2004 / Folie 112a

Ziele:

Illustration von Folie 112

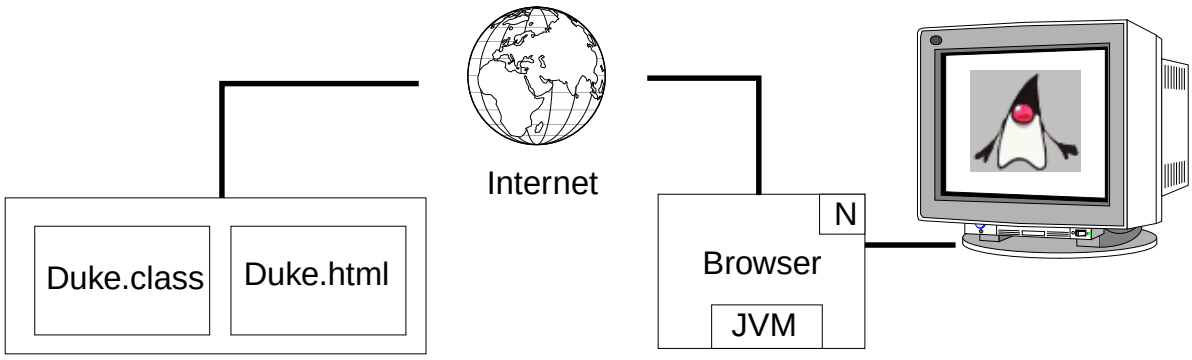
in der Vorlesung:

Erläuterungen dazu

8. Applets

Applet (small application):

- kleines Anwendungsprogramm in Java für eine spezielle Aufgabe,
- an eine WWW-Seite (world wide web) gekoppelt;
- das Applet wird mit der WWW-Seite über das Internet übertragen;
- der Internet Browser (Werkzeug zum Navigieren und Anzeigen von Informationen) enthält eine Java Virtual Machine (JVM); sie führt eintreffende Applets aus.



Programm (Java-Applet) wird übertragen, läuft beim Empfänger, bewirkt dort Effekte.

Applets benutzen Bibliotheken des Empfängers.

Vorlesung Software-Entwicklung II SS 2004 / Folie 124

Ziele:

Prinzip der Applets im Internet

in der Vorlesung:

- Begriffe erläutern
- Applet vorführen Catch the Duke
- Applet vorführen Traffic Lights

nachlesen:

Judy Bishop: Java lernen, 2.Aufl., Abschnitt 12.1

Übungsaufgaben:

Verständnisfragen:

- Welche Vorteile hat es, ein Programm beim Empfänger statt beim Sender auszuführen?
- Welche Nachteile?

Programmierung von Applets

Applets werden wie Programme mit AWT-Benutzungsoberfläche geschrieben, aber:

- Die Hauptklasse ist Unterklasse von **`Applet`** statt von **`Frame`**.
- Es gibt die Methode **`main`** nicht.
- **`exit`** darf nicht aufgerufen werden.
- Die Methode **`public void init ()`** tritt an die Stelle des Konstruktors.
- Ein- und Ausgabe mit AWT-Komponenten statt mit Dateien.

Programmschema: Fenster mit Zeichenfläche (SWE-88):

```
class WarningApplet extends Applet
{ ...
   public void paint (Graphics g)
   { ... auf g schreiben und zeichnen ...}
}
```

Programmschema: Bedienung mit AWT-Komponenten (SWE-103):

```
class TrafficLight extends Applet
{  public void init ()
   { ... Objektbaum mit AWT-Komponenten aufbauen ... }
   ...
}
```


Vorlesung Software-Entwicklung II SS 2004 / Folie 125

Ziele:

Programmstruktur von Applets kennenlernen

in der Vorlesung:

Begründung der Strukturen

nachlesen:

Judy Bishop: Java lernen, 2.Aufl., Abschnitt 12.1

Übungsaufgaben:

Verständnisfragen:

Warum ist Datei-E/A für Applets nicht sinnvoll?

Applet-Methoden

Die Klasse **`Applet`** liegt in der Hierarchie der AWT-Komponenten:

```
Component
   Container
      Panel
         Applet
```

`Applet` definiert Methoden ohne Effekt zum Überschreiben in Unterklassen:

	wird aufgerufen, wenn das Applet ...
`void init ()`	geladen wird
`void start ()`	seine Ausführung (wieder-)beginnen soll
`void stop ()`	seine Ausführung unterbrechen soll
`void destroy ()`	das Applet beendet wird

Methoden zum Aufruf in Unterklassen von **`Applet`**, z. B.

`void showStatus (String)`	Text in der Statuszeile des Browsers anzeigen
`String getParameter (String)`	Wert aus HTML-Seite lesen

Weitere Methoden aus Oberklassen, z. B

`void paint (Graphics g)`	auf **`g`** schreiben und zeichnen (aus **`Container`**)

Vorlesung Software-Entwicklung II SS 2004 / Folie 126

Ziele:

Wichtige Methoden für die Applet-Programmierung kennenlernen

in der Vorlesung:

- Applet in die Hierarchie einordnen
- Methoden erläutern

nachlesen:

Judy Bishop: Java lernen, 2.Aufl., Abschnitt 12.1

Übungsaufgaben:

Verständnisfragen:

In welcher Verwandtschaftsbeziehung steht Applet zu Window und zu Frame?

Applets zu HTML-Seiten

HTML (Hypertext Markup Language):

- Sprache zur Beschreibung formatierter, strukturierter Texte und deren Gestaltung
- Standardsprache zur Beschreibung von Internet-Seiten
- Einfache Sprachstruktur: Marken **
** und Klammern **
** ... **
** mit bestimmter Bedeutung, z. B.

```
Wir unterscheiden
<ul>
   <li> diesen Fall,
   <li> jenen Fall
</ul>
und viele andere Fälle.
```

Wir unterscheiden
- diesen Fall,
- jenen Fall

und viele andere Fälle.

Ein Applet wird auf einer HTML-Seite aufgerufen, z. B.

```
<title>Catch the Duke!</title>
<hr>
<applet code=CatchM.class width=300 height=300>
</applet>
```

Beim Anzeigen der HTML-Seite im Browser oder Appletviewer wird das Applet auf einer Fläche der angegebenen Größe ausgeführt.

Vorlesung Software-Entwicklung II SS 2004 / Folie 127

Ziele:

HTML-Seite für Applets verstehen

in der Vorlesung:

- HTML-Sprache kurz erläutern
- HTML-Text zeigen
- Unterschied zu WYSIWIG-Textsystemen (What-you-see-is-what-you-get)

nachlesen:

Judy Bishop: Java lernen, 2.Aufl., Abschnitt 12.1

Übungsaufgaben:

Verständnisfragen:

Finden Sie mindestens 5 verschieden HTML-Klammern und geben Sie deren Bedeutung an.

Java-Programm in Applet umsetzen

Ein Java-Programm kann man in folgenden Schritten in ein Applet transformieren:

1. Alle Datei-E/A in Benutzung von AWT-Komponenten umsetzen, z. B. **`System.out.println(...)`** in **`g.drawstring(...)`** in der Methode **`paint`**.
2. Programm nicht anhalten, **`exit`**-Aufrufe und **`close`**-Button entfernen.
3. Layoutmanager explizit wählen, Default ist **`FlowLayout`**.
4. **`java.applet.*`** importieren,
   Hauptklasse als Unterklasse von **`Applet`** definieren
5. Konstruktor durch **`init`**-Methode ersetzen; darin wird der Objektbaum der Komponenten aufgebaut.
6. **`main`**-Methode entfernen;
   die Hauptklasse des Java-Programms entfällt einfach, wenn sie nur die **`main`**-Methode enthält und darin nur das **`Frame`**-Objekt erzeugt und plaziert wird (wie im Beispiel **`Traffic`**).
7. HTML-Datei herstellen mit Aufruf der **`.class`**-Datei.
8. Testen des Applets; erst mit dem **`appletviewer`** dann mit dem Browser.

siehe Beispiel Ampel-Simulation SWE-104 bis 108.

Vorlesung Software-Entwicklung II SS 2004 / Folie 128

Ziele:

Einfache Umsetzungsregeln lernen

in der Vorlesung:

Erläuterungen dazu am Beispiel

nachlesen:

Judy Bishop: Java lernen, 2.Aufl., Abschnitt 12.1

Übungsaufgaben:

Wandeln Sie das Java-Programm zur Ampel-Simulation in ein Applet um. Welche der Schritte sind nötig?

Verständnisfragen:

Wie muss die Hauptklasse eines Java-Programms beschaffen sein, damit sie für das Applet einfach entfallen kann?

Parameter zum Start des Applet

Dem Applet können zum Aufruf von der HTML-Seite Daten mitgegeben werden.

Notation: Paare von Zeichenreihen für Parametername und Parameterwert

```
<PARAM NAME="Parametername" VALUE="Parameterwert">
```

eingesetzt im Applet-Aufruf:

```
<APPLET CODE="COffeeShop.class" WIDTH=600 HEIGHT=200>
<PARAM NAME="Columbian" VALUE="12">
<PARAM NAME="Java" VALUE="15">
<PARAM NAME="Kenyan" VALUE="9">
</APPLET>
```

Im Applet auf die Parameterwerte zugreifen:

```
preis = Integer.parseInt (getParameter ("Java"));
```


Vorlesung Software-Entwicklung II SS 2004 / Folie 129

Ziele:
Technik der Applet-Parameter kennenlernen

in der Vorlesung:
Erläuterungen dazu

nachlesen:
Judy Bishop: Java lernen, 2.Aufl., Abschnitt 12.2

Übungsaufgaben:

Verständnisfragen:
Unter welchen Umständen sind solche Parameter sinnvoller als Programmkonstante?

Sicherheit und Applets

Beim Internet-Surfen kann man nicht verhindern, daß fremde Applets auf dem eigenen Rechner ausgeführt werden. Deshalb sind ihre **Rechte i. a. stark eingeschränkt**:

Operation	Java Programm	Applet im applet viewer	lokales Applet im Browser	fremdes Applet im Browser
lokale Datei zugreifen	X	X		
lokale Datei löschen	X			
anderes Programm starten	X	X		
Benutzernamen ermitteln	X	X	X	
zum Sender des Applet verbinden	X	X	X	X
zu anderem Rechner verbinden	X	X	X	
Java Bibliothek laden	X	X	X	
exit aufrufen	X	X		
Pop-up Fenster erzeugen	X	X	X	X

Diese Einstellungen der Rechte können im Browser geändert werden.

Man kann auch **signierten Applets** bestimmter Autoren weitergehende Rechte geben.

Außerdem wird der **Bytecode** jeder Klasse auf Konsistenz **geprüft**, wenn er in die JVM geladen wird.

Vorlesung Software-Entwicklung II SS 2004 / Folie 130

Ziele:

Rechte der Applets im Vergleich

in der Vorlesung:

Erläuterungen und Begründungen dazu

nachlesen:

Judy Bishop: Java lernen, 2.Aufl., Abschnitt 12.2

Übungsaufgaben:

Verständnisfragen:

Warum ist es einem Applet nicht erlaubt, lokale Dateien zu lesen?