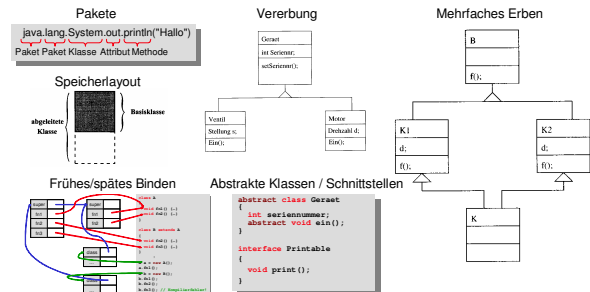


## Abstract Window Toolkit

Thomas Röfer

Desktop-Metapher  
AWT und Swing  
Applets und Anwendungen  
Rahmen und Menüs  
Komponenten und Container  
Panels und Layouts  
Grafik und Schriften  
Ereignisbehandlung

## Rückblick „Vererbung“

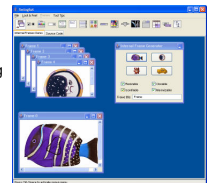


## Desktop-Metapher

- ▶ **Ansatz**
  - ▶ Graphische Oberflächen bedienen sich Metaphern, d.h. sie bilden reale Objekte und Abläufe nach, die dem Benutzer den Umgang mit dem Computer erleichtern sollen, da er diese Objekte aus der realen Welt bereits kennt
  - ▶ Der Benutzer bestimmt, was zu tun ist, d.h. es gibt keine globale feste Reihenfolge
  - ▶ modal vs. nicht-modal
- ▶ **Beispiele**
  - ▶ Desktop (Schreibtischoberfläche)
    - ▶ Man kann Dinge ablegen, hin- und herschieben und in den Papierkorb werfen
    - ▶ Teilweise schräg: Disketten in Papierkorb werfen, um sie entnehmen zu können
  - ▶ Fenster
    - ▶ Zeigen einen Ausschnitt der Welt. Die Sicht kann durch Rollbalken geändert werden
  - ▶ Buttons (IBM-Deutsch: Schaltflächen)
  - ▶ Listen, Dialoge, Karteikarten, Assistenten ...

## AWT und Swing

- ▶ **Abstract Window Toolkit**
  - ▶ Seit Java 1.0
  - ▶ Nutzt die Elemente des unterliegenden Fenstersystems
  - ▶ Ist relativ schnell
  - ▶ Der Leistungsumfang ist eingeschränkt
  - ▶ Findet sich unter `java.awt.*`
- ▶ **Swing**
  - ▶ Seit Java 1.1
  - ▶ „Leichtgewichtige“ Komponenten (d.h. nur wenig Code des unterliegenden Fenstersystems)
  - ▶ Ist langsamer als AWT
  - ▶ Der Leistungsumfang ist deutlich größer
  - ▶ Unterstützt unterschiedliche „Look & Feels“
  - ▶ Findet sich unter `javax.swing.*`



## Applets und Anwendungen

- ▶ **Applets**
  - ▶ Können nur innerhalb eines Webbrowsers oder im AppletViewer ausgeführt werden
  - ▶ Sind immer grafisch
  - ▶ Zeichenfläche ist statisch
  - ▶ Starten und Beenden automatisch durch Browser
- ▶ **Anwendungen**
  - ▶ Können sowohl textuell als auch grafisch sein
  - ▶ Grafische Anwendungen generell aufwändiger als Applets
  - ▶ Müssen eine Funktion `main()` haben
  - ▶ `main()` erzeugt typischerweise nur ein Objekt des Hauptfensters und kann daher in BlueJ entfallen



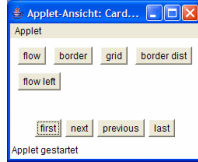
## Applets

- ▶ **Allgemein**
  - ▶ Ein Applet ist eine Klasse, die von `java.applet.Applet` erbt
  - ▶ Funktionalität wird implementiert, indem existierende Methoden überschrieben oder Schnittstellen implementiert werden
  - ▶ Der Programmfluss wird von außen bestimmt (Ereignisbasierte Programmierung)
    - ▶ *Don't call us, we'll call you!*
- ▶ **Methoden**
  - ▶ Initialisierung
    - ▶ `void init()`
  - ▶ Benutzer betritt mit Applet Seite (wieder)
    - ▶ `void start()`
  - ▶ Benutzer verlässt Seite mit Applet (wieder)
    - ▶ `void stop()`
  - ▶ Applet wird nicht mehr gebraucht
    - ▶ `void destroy()`
  - ▶ Alle Methoden von `java.awt.Panel`, z.B.
    - ▶ `void paint(Graphics g)`



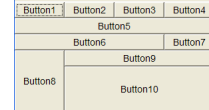
## Panels und Layouts

- FlowLayout**
  - Standard-LayoutManager für Panels
  - Kann beliebig viele Komponenten enthalten, die an den Fenstergrenzen umgebrochen werden
  - Unterschiedliche Ausrichtungen
    - FlowLayout.LEFT, FlowLayout.CENTER, FlowLayout.RIGHT
- GridLayout**
  - Ordnet Komponenten in einem Gitter an
  - Alle Komponenten haben die gleiche Größe



## Panels und Layouts

- GridBagLayout**
  - Kompliziertester, aber auch mächtigster LayoutManager
  - Erlaubt die Zuordnung sog. Einschränkungen (*Constraints*) zu jeder Komponente
- GridBagConstraints**
  - Legt die Einschränkungen für Komponenten in GridBagLayout fest, z.B.
    - Relative Breite und Höhe
      - gridwidth, gridweight
      - Anzahl Zellen, GridBagConstraints.RELATIVE, GridBagConstraints.REMAINDER
    - Gewichtungen
      - weightx, weighty



## Grafik

- Allgemein**
  - In alle Komponenten kann gezeichnet werden, in dem `paint(Graphics)` überschreibt
  - Aber es kann zu Konflikten mit den Zeichenroutinen der Komponenten kommen
- Allgemeine Zeichenfläche**
  - class `Canvas`
- Zeichenschnittstelle**
  - interface `Graphics`
  - Wird an `paint` übergeben
  - Arbeitet zustandsbasiert
  - Speichert den aktuellen Zustand des Zeichenwerkzeugs (Farbe, Schriftart usw.)

```
import java.awt.*;

class FrameBeispiel extends Frame
{
    FrameBeispiel()
    {
        setSize(320, 280);
        setVisible(true);
    }

    public void paint(Graphics g)
    {
        g.setColor(Color.black);
        g.drawString("Vereinzelungseinheit", 150, 50);
        g.fillRect(90, 5, 5, 240);
        g.fillRect(135, 5, 5, 200);
        g.fillRect(90, 245, 145, 5);
        g.fillRect(135, 205, 100, 5);

        g.setColor(Color.red);
        g.fillOval(95, 80, 40, 40);
        g.setColor(Color.yellow);
        g.fillOval(95, 120, 40, 40);
    }
}
```



## Schriften

- Begriffe**
  - Grundlinie, Oberlänge, Unterlänge, Höhe
  - Proportionschrift, Schrittweite
  - Unterschneidung, Lingaturen
  - Serifen
- Auswahl einer Schrift (Font)**
  - Name, z.B. „SansSerif“
  - Stil, (ver-odert) z.B. Font.BOLD | Font.ITALIC
  - Größe in Punkten (1pt = 1/72 Zoll), z.B.
    - g.setFont(new Font("SansSerif", Font.PLAIN, 16));
- Informationen über eine Schrift (FontMetrics)**
  - getAscent(), getDescent(), getHeight()
  - stringWidth(String)



## Ereignisbehandlung (event-handling)

- Konzept**
  - Programmsteuerung erfolgt durch Ereignisse
  - Man kann Listener an Komponenten hängen, die über Ereignisse (...Event) informiert werden
  - Die Mitteilung erfolgt über den Aufruf von Objekt-Methoden
  - Welche Ereignisse mitgehört werden können, ist über Interfaces (...Listener) vordefiniert
  - Die Ereignisbehandler heißen oft `process...Event(...Event)`
- Vordefinierte Ereignisse und Listener**
  - ActionEvent/Listener
  - ComponentEvent/Listener
  - ContainerEvent/Listener
  - FocusEvent/Listener
  - KeyEvent/Listener
  - MouseEvent/Listener
  - TextEvent/Listener
  - AdjustmentEvent/Listener
  - ItemEvent/Listener
  - MouseEvent/Listener
  - WindowEvent/Listener

## WindowEvent / WindowListener

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

class AWTApp extends Frame
implements WindowListener,
        ActionListener
{
    AWTApp()
    {
        super("My App");
        addWindowListener(this);
        Button b = new Button("Close");
        add(b);
        b.addActionListener(this);
        setSize(300, 300);
        setVisible(true);
    }

    public void actionPerformed(ActionEvent e)
    {
        if(e.getActionCommand().equals("Close"))
            windowClosing(null);
    }

    public void windowClosing(WindowEvent e)
    {
        dispose();
        System.exit(0);
    }

    public void windowOpened(WindowEvent e)
    {
    }

    public void windowClosed(WindowEvent e)
    {
    }

    public void windowIconified(WindowEvent e)
    {
    }

    public void windowDeiconified(WindowEvent e)
    {
    }

    public void windowActivated(WindowEvent e)
    {
    }

    public void windowDeactivated(WindowEvent e)
    {
    }
}
```

