

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

PI-1: Abstract Window Toolkit 2

Rückblick „Generisches Programmieren“

Motivation

```

class Node<T>
{
    private T info;
    private Node<T> left, right;

    Node(T i) {info = i;}
    Node(T i, Node<T> l, Node<T> r)
    {info = i; left = l; right = r;}
    T getInfo() {return info;}
    Node<T> getLeft() {return left;}
    Node<T> getRight() {return right;}
    void setInfo(T i) {info = i;}
}
    
```

Generische Klassen/Interfaces

Generische Typen

```

Node<String> n = new Node<String>("foo");
String a = n.getInfo();
    
```

Typebounds

```

class Node<T extends Comparable<T>>
{
    // ...
}
    
```

Polymorphe Methoden

```

interface FoldIf<A, B> { A fn(A a, B b); }

<A, B> A fold(FoldIf<A, B> f, A a, B[] b)
{
    for(B b0 : b)
        a = f.fn(a, b0);
    return a;
}
    
```

Wildcard-Typen

	Typ	Lesen	Schreib.	Kompatible Typargumente
Invarianz	C<T>	ja	ja	T
Bivarianz	C<?>	nein	nein	Alle
Covarianz	C<? extends B>	ja	nein	B und abgeleitete Typen
Contravarianz	C<? super B>	nein	ja	B und Basistypen

PI-1: Abstract Window Toolkit 2

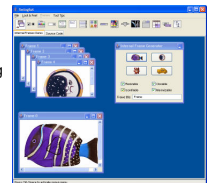
Desktop-Metapher

- 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 ...

PI-1: Abstract Window Toolkit 3

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.*



PI-1: Abstract Window Toolkit 4

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



PI-1: Abstract Window Toolkit 5

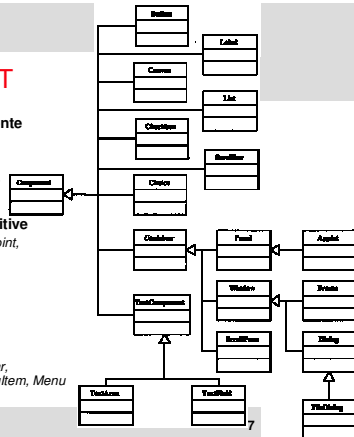
Applets

- 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)`

PI-1: Abstract Window Toolkit 6

Struktur des AWT

- Fenster und Dialogelemente**
 - CheckboxGroup, s. rechts
- Layouts**
 - BorderLayout, CardLayout, FlowLayout, GridLayout, GridBagLayout ...
- Zeichnen und Grafikprimitive**
 - Color, Graphics, Image, Point, Polygon, Rectangle
- Schriften**
 - Font, FontMetrics
- Ereignisse**
 - Event
- Menüs**
 - MenuComponent, MenuBar, MenuItem, CheckboxMenuItem, Menu
- Toolkit**



Rezept für eine AWT-Anwendung

- Benutzungsschnittstelle**
 - Erzeugen eines Rahmens (Ableitung von *class Frame*)
 - Initialisierung der Schriften, Farben, Layouts und anderer sog. *Resourcen*
 - Erzeugung der Menüs und des Menübalkens
 - Erzeugung der Steuerelemente, Dialoge, Fenster usw.
- Implementierung**
 - Ereignisbehandler (event handlers) hinzufügen
 - Funktionalität hinzufügen
 - Fehlerbehandlung hinzufügen

Erzeugen eines Rahmens

- Allgemein**
 - Jede grafische Anwendung benötigt ein Hauptfenster (einen Rahmen)
 - Üblicherweise wird dazu eine Klasse von *Frame* abgeleitet und ein einzelnes Objekt davon konstruiert (z.B. in *main()*)
- Einschränkungen**
 - Eine so erzeugte Anwendung reagiert noch auf keine Ereignisse, z.B. nicht auf das Drücken der *Schließen*-Schaltfläche
 - Man kann es nur beenden, indem man es „abschießt“ (BlueJ: Virtuelle Maschine zurücksetzen)

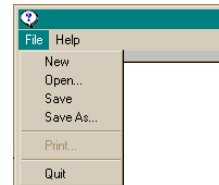
```
import java.awt.*;

class AWTApp extends Frame
{
    AWTApp ()
    {
        super("My App");
        setSize(300, 300);
        setVisible(true);
    }
}
```

Menüs

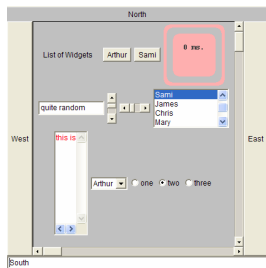
```
// im Konstruktor
MenuBar mbar = new MenuBar();
Menu m = new Menu("File");
m.add(new MenuItem("New"));
m.add(new MenuItem("Open..."));
m.add(new MenuItem("Save"));
m.add(new MenuItem("Save As..."));
m.addSeparator();
m.add(new MenuItem("Print..."));
m.addSeparator();
m.add(new MenuItem("Quit"));
mbar.add(m);

m = new Menu("Help");
m.add(new MenuItem("Help!!!!"));
m.addSeparator();
m.add(new MenuItem("About..."));
mbar.add(m);
setMenuBar(mbar);
```



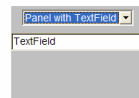
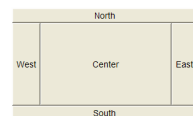
Komponenten und Container

- Allgemein**
 - Component* ist die Basisklasse aller Elemente, mit denen der Benutzer interagieren kann
- Arten**
 - List, Scrollbar, TextArea, TextField, Choice, Button, Label, ...
 - Alle Container
- Methoden, z.B.**
 - setBackground(Color)
 - setFont(Font)
 - setVisible()
 - getLocation(int, int), getSize(int, int)
 - paint(Graphics), update(Graphics), repaint()
- Container**
 - Komponenten, die andere Komponenten enthalten können



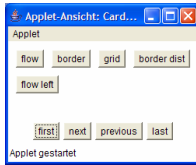
Panels und Layouts

- Panel**
 - Ein rechteckiger Bereich, der andere Komponenten enthalten kann und dem ein *LayoutManager* zugeordnet ist
- LayoutManager**
 - Erlauben die Anordnung von Komponenten nach vorgegebenen Kriterien
 - Notwendig für Plattformunabhängigkeit
- BorderLayout**
 - Standard-LayoutManager für alle Fenster, z.B. Rahmen und Dialoge
 - Speichert Komponenten in fünf Bereichen
 - north, south, east, west und center
 - Der überschüssige Raum wird dem mittleren Bereich zugeordnet
- CardLayout**
 - Speichert mehrere Komponenten, die nacheinander angezeigt werden sollen, d.h. man kann zwischen verschiedenen Komponenten umschalten



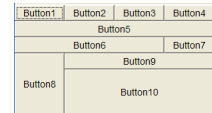
Panels und Layouts

- 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, gridheight
 - 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("Vereinzellungseinheit", 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
 - Proportionalchrift, Schrittweite
 - Unterschneidung, Ligaturen
 - 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
 - AdjustmentEvent/Listener
 - ComponentEvent/Listener
 - ContainerEvent/Listener
 - FocusEvent/Listener
 - ItemEvent/Listener
 - KeyEvent/Listener
 - MouseEvent/Listener
 - TextEvent/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)
    {
    }
}
```

Fachgespräche

- ▶ **Ziel**
 - ▶ Überprüfung der Individualität der Leistung
- ▶ **Ablauf**
 - ▶ Kleine Programmieraufgabe, z.B.
 - ▶ *Schreibe eine iterative Methode, die alle Elemente einer Reihung quadriert*
 - ▶ *Gegeben: class Elem (int value; Elem next;)*
 - ▶ *Schreibe eine rekursive Methode, die alle Elemente einer Liste quadriert*
 - ▶ *Schreibe eine rekursive Methode, die alle Elemente einer Reihung quadriert*
 - ▶ Fragen nach Begriffen und Konzepten
 - ▶ *Keine Java-Syntax*
 - ▶ *Schwerpunkt auf Vorlesungen 2-6*
 - ▶ *Aber auch allgemeine Fragen aus anderen Vorlesungen, z.B. Unterschied zwischen Vorrang und Assoziativität von Operatoren?*

Fachgespräche

- ▶ **Bewertung**
 - ▶ Ausgangspunkt: Vornote aus dem Übungsbetrieb
 - ▶ Aufwertung: Maximal 1/3-Note
 - ▶ Abwertung: beliebig
- ▶ **Wiederholung**
 - ▶ Nach mindestens 4 Wochen
- ▶ **SBLN mitbringen!**
 - ▶ Außer Systems Engineering
 - ▶ *Haben eigene Scheine*
 - ▶ Von PI-1-Homepage zu Ende ausfüllen und ausdrucken
 - ▶ *Name, Matrikel-Nr., Studiengang*

