

Praktische Informatik 1

Abstract Window Toolkit

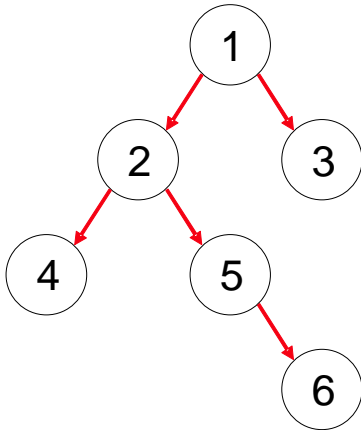
Thomas Röfer

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

Rückblick

„Generisches Programmieren“

Motivation



Generische Klassen/Interfaces

```

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 Typen

```

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

Typebounds

```

class Node<T extends Comparable<T>> {
    :
}
  
```

Polymorphe Methoden

```

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

Wildcard-Typen

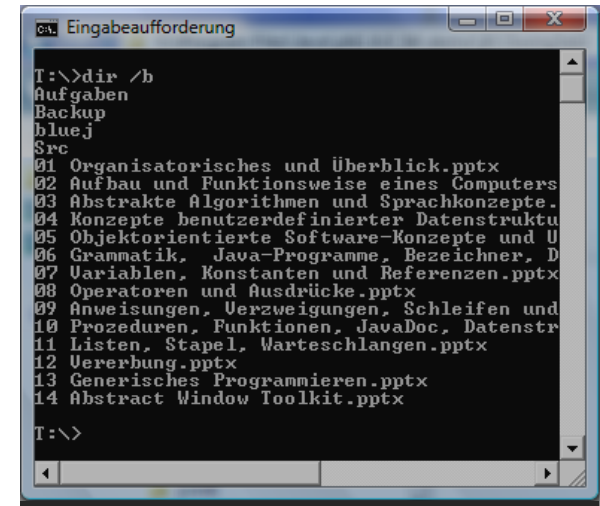
	Typ	Lesen	Schreib.	Kompatible Typargumente
Invarianz	C<T>	ja	ja	T
Bivarianz	C<?>	nein	nein	Alle
Covarianz	C<? extends B>	ja	nein	B und abgeleitete Typen
Contravarianz	C<? super B>	nein	ja	B und Basistypen

```

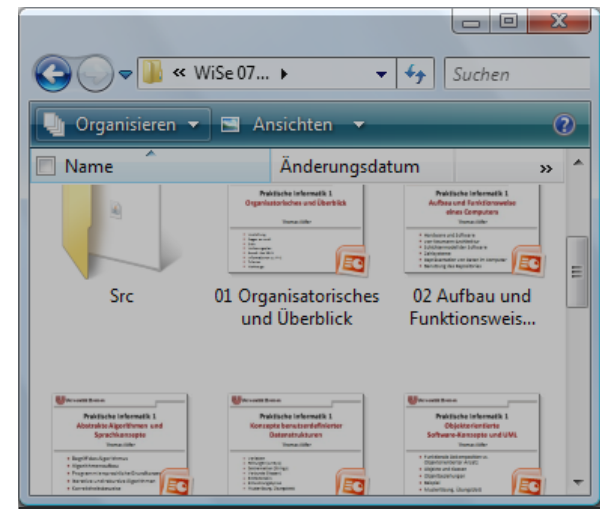
static <A, B> A foldl(FoldIFn<A, B> f, A a, B[] b) {
    for (B b0 : b) {
        a = f.fn(a, b0);
    }
    return a;
}
  
```

Desktop-Metapher

- 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 der Bearbeitung
- *modal* vs. *nicht-modal*



```
Eingabeaufforderung
T:\>dir /b
Aufgaben
Backup
bluej
Src
01 Organisatorisches und Überblick.pptx
02 Aufbau und Funktionsweise eines Computers
03 Abstrakte Algorithmen und Sprachkonzepte.
04 Konzepte benutzerdefinierter Datenstruktu
05 Objektorientierte Software-Konzepte und U
06 Grammatik, Java-Programme, Bezeichner, D
07 Variablen, Konstanten und Referenzen.pptx
08 Operatoren und Ausdrücke.pptx
09 Anweisungen, Verzweigungen, Schleifen und
10 Prozeduren, Funktionen, JavaDoc, Datenstr
11 Listen, Stapel, Warteschlangen.pptx
12 Vererbung.pptx
13 Generisches Programmieren.pptx
14 Abstract Window Toolkit.pptx
T:\>
```

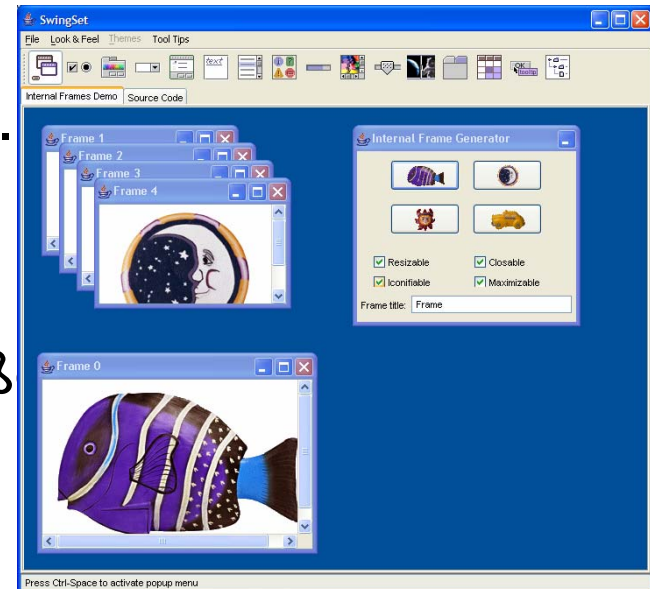


Desktop-Metapher – Beispiele

- 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

- 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!



Applets – Methoden

- Initialisierung
 - *void init()*
- Benutzer betritt Seite mit Applet (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)*

Applets – Beispiel

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

public class Boss extends Applet {
    boolean hurt = false;
    int i = 0;

    public void init() {
        boss = getImage(getDocumentBase(), "boss.gif");
        pain = getImage(getDocumentBase(), "boss_in_pain.gif");
        enableEvents(MouseEvent.MOUSE_CLICKED);
    }

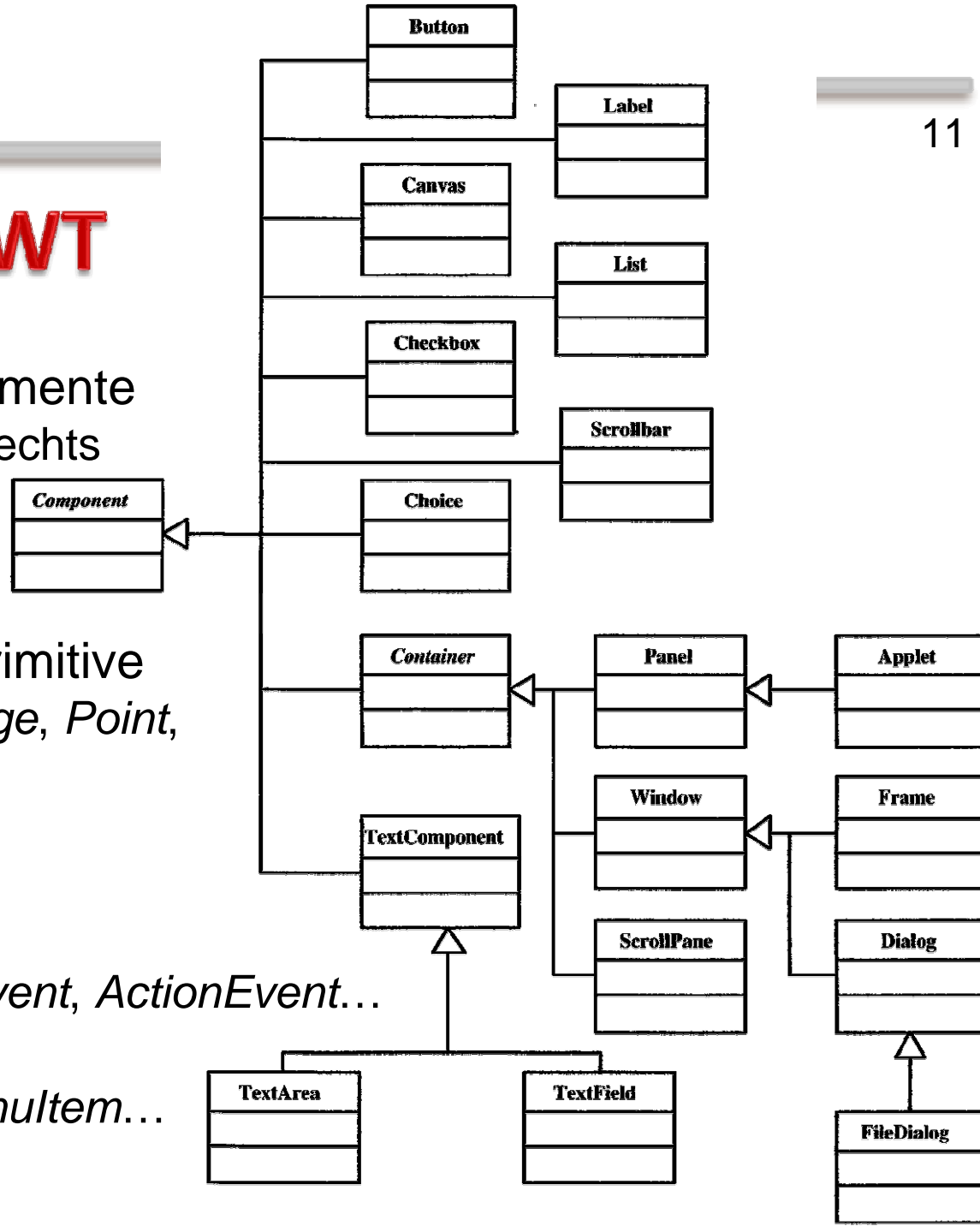
    public void start() {
        repaint();
    }
}
```

Applets – Beispiel

```
public void paint(Graphics g) {  
    g.drawImage(hurt ? pained : boss, 0, 0, this);  
    hurt = !hurt;  
}  
  
public void processMouseEvent(MouseEvent m) {  
    if(m.getID() == MouseEvent.MOUSE_CLICKED) {  
        if(i % 10 == 0) {  
            play(getDocumentBase(), "pain.au");  
        } else if(i % 5 == 0) {  
            play(getDocumentBase(), "ouch1.au");  
        } else if(i % 3 == 0) {  
            play(getDocumentBase(), "ouch2.au");  
        } else {  
            play(getDocumentBase(), "ouch3.au");  
        }  
        i++;  
        repaint();  
    }  
}
```

Struktur des AWT

- Fenster und Dialogelemente
 - *CheckBoxGroup*, s. rechts
- Layouts
 - *BorderLayout*, *CardLayout*...
- Zeichnen und Grafikprimitive
 - *Color*, *Graphics*, *Image*, *Point*, *Polygon*, *Rectangle*
- Schriften
 - *Font*, *FontMetrics*
- Ereignisse
 - *WindowEvent*, *KeyEvent*, *ActionEvent*...
- Menüs
 - *MenuBar*, *Menu*, *MenuItem*...
- Toolkit



Rezept für eine AWT-Anwendung

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

Erzeugen eines Rahmens

- 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 BoulderDashGUI
    extends Frame {
    BoulderDashGUI() {
        super("Boulder Dash");
        setSize(480, 480);
        setVisible(true);
    }
}
```

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 e)*
- Vordefinierte *Events* und *Listener*
 - *ActionEvent/Listener* *AdjustmentEvent/Listener*
 - *ComponentEvent/Listener* *ContainerEvent/Listener*
 - *FocusEvent/Listener* *ItemEvent/Listener*
 - *KeyEvent/Listener* *MouseEvent/Listener*
 - *TextEvent/Listener* *WindowEvent/Listener*

Ereignisbehandlung – Beispiel

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

```
class BoulderDashGUI extends Frame  
    implements WindowListener {  
    BoulderDashGUI() {  
        super("Boulder Dash");  
        addWindowListener(this);  
        setSize(480, 480);  
        setVisible(true);  
    }
```

```
    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) {}  
}
```