

Einführung in die GUI-Programmierung mit Java

Softwarepraktikum „Datenstrukturen“

Sommersemester 2004

Graphik-Anwendungen in Java

- `java.awt` (Abstract Window Toolkit)
 - Seit JDK 1.0
 - Enthält grundlegende Standard-GUI-Komponenten
 - Look & Feel entspricht der lokalen Plattform
 - I. d. R. nur relative Platzierung der Komponenten zueinander
- `javax.swing`
 - Seit JDK 1.1, basiert auf AWT
 - Umfangreichere Gestaltungsmöglichkeiten als AWT
 - Look & Feel ist auf allen Plattformen gleich
 - Bibliothek wird fortlaufend ausgebaut
- Im folgenden betrachtet: AWT

Hierarchischer Aufbau von Fenstern

- Hauptfenster (*Container*) enthält untergeordnete Fenster (ebenfalls *Container*, 1:n-Beziehung)
- Beziehung wird zur Laufzeit dynamisch aufgebaut („füge Komponente x zu Komponente y hinzu“)

- Klassenhierarchie (Auswahl):

java.lang.Object

↳ java.awt.Component -- abstrakte Klasse

↳ java.awt.Container -- abstrakte Klasse

↳ java.awt.Panel

↳ java.awt.Window -- abstrakte Klasse

↳ java.awt.Frame

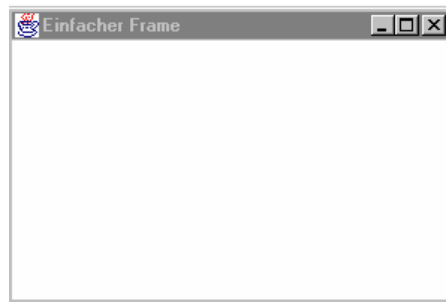
Graphikklassen

- Component
 - Objekt mit graphischer Darstellung und Interaktionsmöglichkeit
 - Bsp.: Buttons, Checkboxes, Scrollbalken
- Container
 - Komponente, die andere Komponenten enthalten kann
 - Verwaltung einer Liste der enthaltenen Komponenten (Reihenfolge „von vorne nach hinten“)
- Window
 - Container ohne Ränder und Menübalken
 - Beim Anlegen muss ein Frame, Dialog oder ein anderes Fenster als „Owner“ angegeben werden
- Panel
 - Einfachste Container-Klasse
 - stellt Raum bereit, in den andere Komponenten eingefügt werden können
- Frame
 - Top-Level-Fenster mit Titelzeile und Rand

Beispiel: Einfaches Fenster (Frame)

```
import java.awt.*;  
public class MyFrameEinfach {  
    public static void main (String [] args) {  
        Frame f = new Frame („Einfacher Frame“);  
        f.setSize(300,200);    -- initiale Größe  
        f.setVisible(true);    -- auf „sichtbar“ setzen  
    }  
}
```

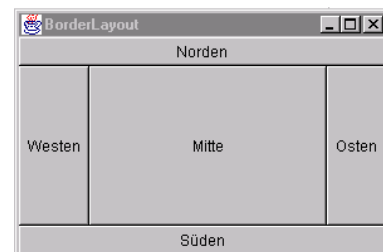
Resultat:



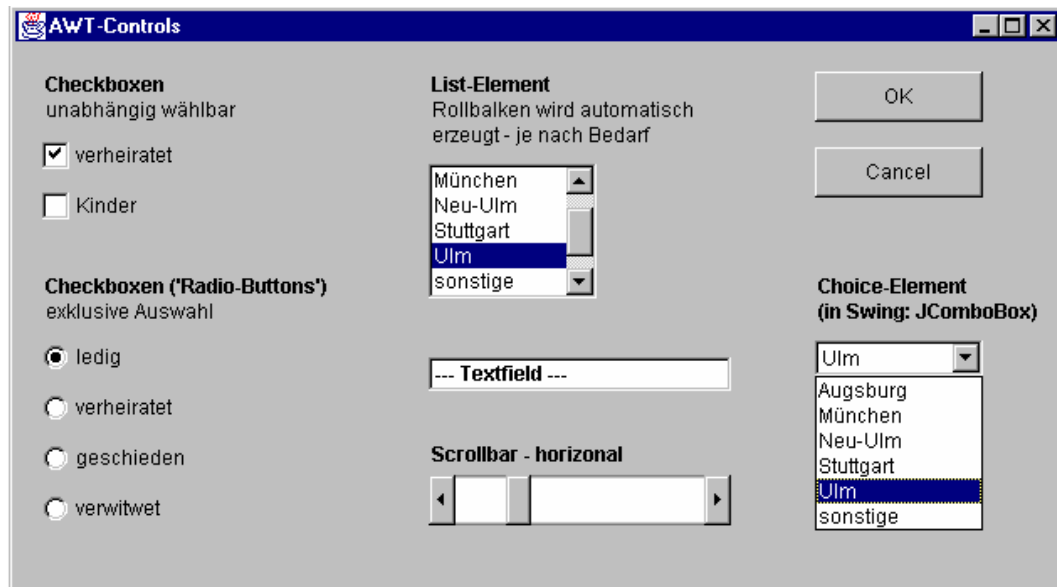
Beispiel: Fenster mit Button (eigene Klasse)

```
import java.awt.*;  
public class MyFrameWithButton extends Frame {  
    MyFrameWithButton (String Title) {  
        super(Title);  
        add(„North“, new Button(„Schalter 1“));  
        setSize(300,100);  
        setVisible(true);  
    }  
    public static void main (String [] args) {  
        new MyFrameWithButton(„Fenster mit Button“);  
    }  
}
```

Resultat:

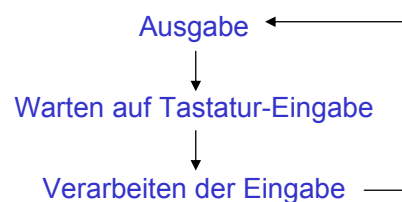


AWT-Graphik-Elemente



Ereignisorientierte Programmierung (1)

- „Text-orientierte“ Programme:
 - Steuerung des Programms durch die main()-Funktion
 - main() = Einsprungspunkt beim Starten des Programms
 - Typischer Zyklus bei interaktiven Programmen (bislang):

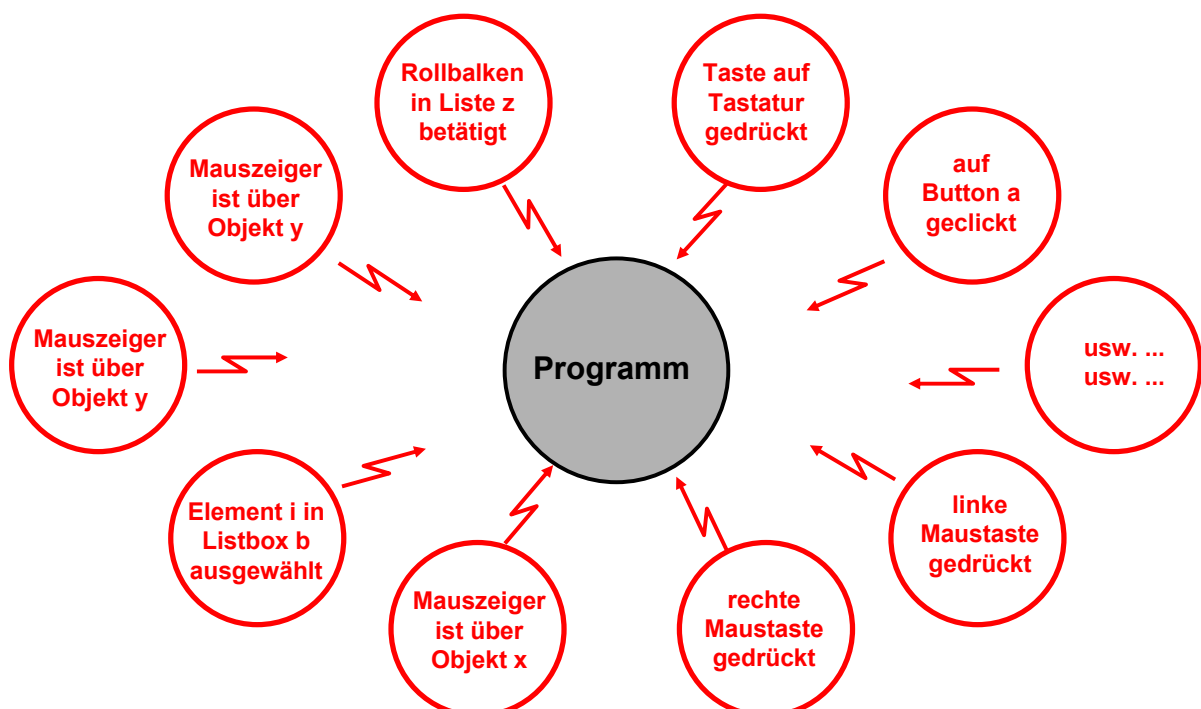


- Warten auf Tastatur-Eingabe: „Singuläres“ Ereignis
- Warten auf Return von einem Methodenaufruf, z. B. readLine()

Ereignisorientierte Programmierung (2)

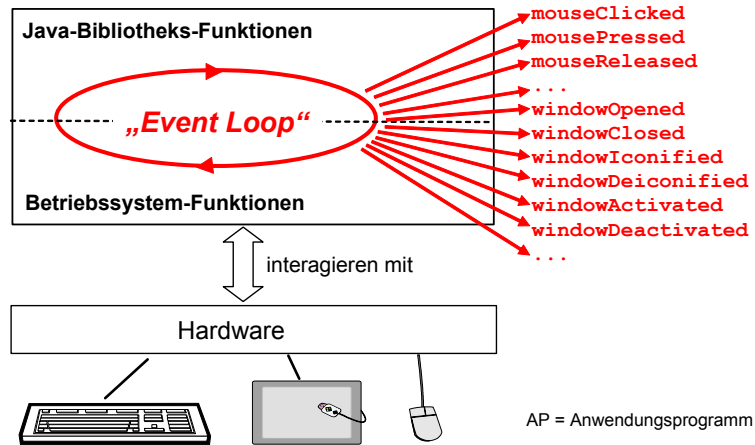
- Hier nun:
 - Graphische Oberfläche mit diversen Steuerelementen
 - Unterschiedliche Ereignisse erfordern verschiedene Methoden zur Behandlung (= Reaktion auf Ereignis)
 - Evtl. viele Ereignisse gleichzeitig (z. B. „Maus befindet sich jetzt über Objekt x“ und „linke Maustaste wird gedrückt“)
- Daher andere „Philosophie“ bei Programmen für graphische Benutzeroberflächen:
 - Die Programme werden von der Umgebung gesteuert!

Ereignisorientierte Programmierung (3)



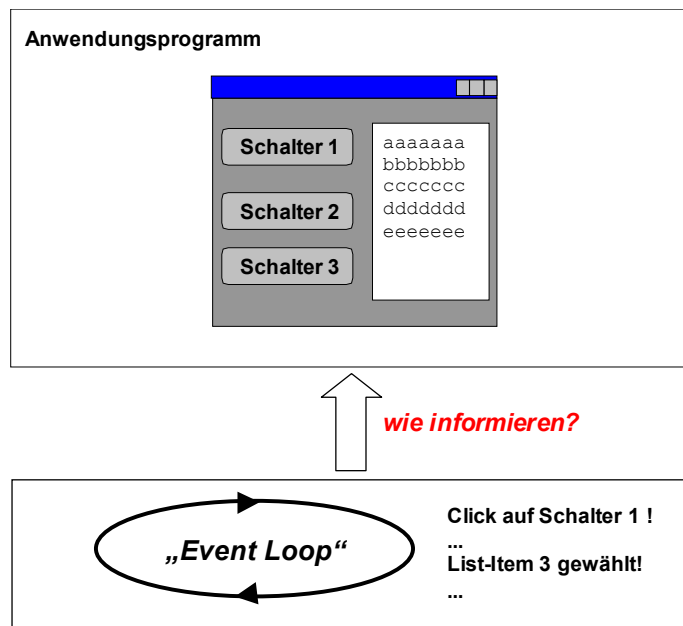
Ereignisorientierte Programmierung (4)

- „Event Loop“



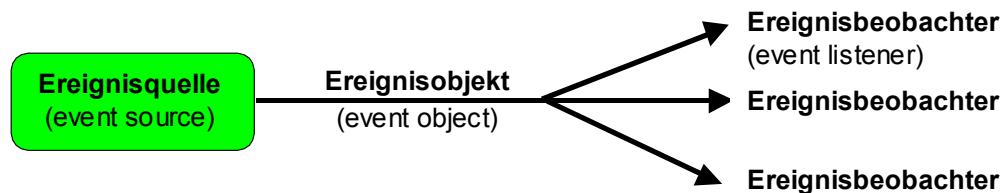
Ereignisorientierte Programmierung (5)

Zu lösendes Problem:
Verbindung von
Programm und
Events



Ereignisorientierte Programmierung (6)

- Java-Ansatz: Ereignisse mit Quellen und Beobachtern



Ereignisorientierte Programmierung (7)

- Typen von Ereignissen in Java
 - Package `java.awt.event` beschreibt und kapselt alle möglichen Ereignisse in entsprechenden Klassen
 - Unterscheidung von Ereignissen:
 - `ComponentEvent`
 - `FocusEvent`
 - `KeyEvent`
 - `MouseEvent`
 - `ContainerEvent`
 - `WindowEvent`
 - `ActionEvent`
 - `AdjustmentEvent`
 - `ItemEvent`
 - `TextEvent`

Ereignisorientierte Programmierung (8)

- Event-Quellen und Beobachter
 - Jedes „interaktive“ Objekt (bzw. Klasse) definiert und implementiert, welche Ereignisse von ihm erzeugt bzw. bei ihm beobachtet werden können
 - Will man GUI-Objekte im Programm verwenden, so kann man
 - sich bei der Komponente für bestimmte Ereignisse als Beobachter (Listener) registrieren lassen
 - nichts tun (Events gehen ins Leere)
 - Registrierung als Beobachter:
Anmeldung beim zu beobachtenden Objekt durch Aufruf von dessen `addxxxListener`-Methode:
 - `guiObject.addMouseListener(beobachterObjekt)`
 - `guiObject.addKeyListener(beobachterObjekt)`
 - `guiObject.addActionListener(beobachterObjekt)`
 - ...

Ereignisorientierte Programmierung (9)

- Beispiel: Button mit Event-Handler

```
import java.awt.*;
public class MyFrameWithBeepButton extends Frame {
    MyFrameWithBeepButton (String Title) {
        super(Title);
        Button beepButton = new Button(„Beep“);
        add(beepButton);
        beepButton.addActionListener(new Beeper());    // Event-Handler
    }
    public static void main (String [] args) {
        new MyFrameWithButton(„Fenster mit Button“);
    }
}
```

Ereignisorientierte Programmierung (10)

- Beispiel: Event-Handler für voriges Beispiel

```
public class Beeper implements ActionListener {  
    public void actionPerformed(ActionEvent event) {  
        Component c = (Component)event.getSource();  
        c.getToolkit().beep();  
    }  
}
```