Informatik IIb

Objektorientierte Programmierung mit Java

Oliver Jack

Fachhochschule Jena
Fachbereich Elektrotechnik und Informationstechnik

Wintersemester 2010/11

Vorlesung 6. Grafische Ausgabe und Benutzeroberflächen

Lernziele dieser Vorlesung

Abstract Windowing Toolkit (AWT)

Elementare Grafikroutinen

Benutzeroberflächen (Swing)

- Einfache Anwendungen
- Layout
- Eigenschaften von Swing

Zusammenfassung

Lernziele

- Kenntnis von Grafik-Methoden
- Kenntnis von Benutzeroberflächen-Methoden
- Kenntnis von Listenern und Steuerung über Benutzeroberflächen

Eigenschaften von AWT

- Die Fähigkeiten des AWT lassen sich grob in vier Gruppen unterteilen:
  - Grafische Primitivoperationen zum Zeichnen von Linien oder Füllen von Flächen und zur Ausgabe von Text
  - Methoden zur Steuerung des Programmablaufs auf der Basis von Nachrichten für Tastatur-, Maus- und Fensterereignisse
  - Dialogelemente zur Kommunikation mit dem Anwender und Funktionen zum portablen Design von Dialogboxen
  - Fortgeschrittenere Grafikfunktionen zur Darstellung und Manipulation von Bitmaps und zur Ausgabe von Sound
- Da die grafischen Fähigkeiten Bestandteil der Sprache bzw. ihrer Klassenbibliothek sind, können sie als portabel angesehen werden. Unabhängig von der Zielplattform wird ein GUI-basiertes Programm auf allen verwendeten Systemen gleich oder zumindest ähnlich laufen.

Anlegen eines Fensters

- Alle Klassen von AWT sichtbar machen: import java.awt.*
- Zur Ausgabe von grafischen Elementen benötigt die Anwendung ein Fenster, auf das die Ausgabeoperationen angewendet werden können.
- Während bei der Programmierung eines Applets ein Standardfenster automatisch zur Verfügung gestellt wird, muss eine Applikation ihre Fenster selbst erzeugen.
- Da die Kommunikation mit einem Fenster über eine Reihe von Callback-Methoden abgewickelt wird, wird eine Fensterklasse in der Regel nicht einfach instantiiert.
- Statt dessen ist es meist erforderlich, eine eigene Klasse aus einer der vorhandenen abzuleiten und die benötigten Interfaces zu implementieren.

Anlegen eines Fensters (Forts.)

- Zum Ableiten einer eigenen Fensterklasse wird in der Regel entweder die Klasse Frame oder die Klasse Dialog verwendet, die beide aus Window abgeleitet sind.
- Die wichtigste Klasse zur Ausgabe von Grafiken in Java-Applikationen ist Frame.
- Um ein einfaches Fenster zu erzeugen und auf dem Bildschirm anzuzeigen, muss ein neues Element der Klasse Frame erzeugt werden.
- Das Fenster wird auf die gewünschte Größe gebracht und durch Aufruf der Methode setVisible sichtbar gemacht.

Beispiel: einfaches Fenster

```
import java.awt.*;

class FensterEinfach
{
  public static void main(String[] args)
  {
    Frame wnd = new Frame("Einfaches Fenster");

    wnd.setSize(400,300);
    wnd.setVisible(true);
  }
}
```

Schließen eines Fensters

- Ein einfaches Hauptfenster besitzt keinerlei Funktionalität, um vom Anwender auf geordnete Weise geschlossen werden zu können.
- Alle entsprechenden Dialogelemente im Systemmenü sind ohne Funktion.
- Abbruch durch Drücken von [STRG]+[C].
- Es ist geeigneter Mechanismus zum Schließen des Hauptfensters erforderlich

Schließen eines Fensters (Forts.)

- Soll der Anwender ein Hauptfenster schließen können, muss ein WindowListener registriert werden.
- Dabei handelt es sich um ein Interface, dessen Methode windowClosing aufgerufen wird, wenn der Anwender über das System-Menü oder den Schließen-Button das Fenster schließen will.
- Das Programm wird in diesem Fall setVisible(false) aufrufen, um das Fenster zu schließen.
- Gegebenfalls wird das Programm ein System.exit anhängen, um zusätzlich das Programm zu beenden.

WindowClosingAdapter

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

public class WindowClosingAdapter
extends WindowAdapter
{
  private boolean exitSystem;

/**
* Erzeugt einen WindowClosingAdapter zum Schliessen
* des Fensters. Ist exitSystem true, wird das komplette
* Programm beendet.
*/
  public WindowClosingAdapter(boolean exitSystem)
  {
    this.exitSystem = exitSystem;
  }
```

WindowClosingAdapter (Forts.)

```
/**
* Erzeugt einen WindowClosingAdapter zum Schliessen
* des Fensters. Das Programm wird nicht beendet.
*/
   public WindowClosingAdapter()
   {
      this(false);
   }

   public void windowClosing(WindowEvent event)
   {
      event.getWindow().setVisible(false);
      event.getWindow().dispose();
      if (exitSystem) {
        System.exit(0);
      }
   }
}
```

WindowClosingAdapter (Forts.)

- Um den gewünschten Effekt zu erzielen, muss der Listener beim zu schließenden Fenster durch Aufruf von addWindowListener registriert werden.
- Dadurch wird beim Anklicken des Schließen-Buttons (bzw. beim Aufrufen des entsprechenden Systemmenüeintrags) die Methode windowClosing aufgerufen und das Fenster geschlossen.
- Falls true an den Konstruktor übergeben wurde, beendet der Listener das gesamte Programm.

Beispiel: Fenster schließen

```
import java.awt.*;

class SchliessenTest
{
  public static void main(String[] args)
  {
    Frame wnd = new Frame("Fenster schliessen");
    wnd.addWindowListener(new
                          WindowClosingAdapter(true));
    wnd.setSize(400,300);
    wnd.setVisible(true);
  }
}
```

Die Methode paint

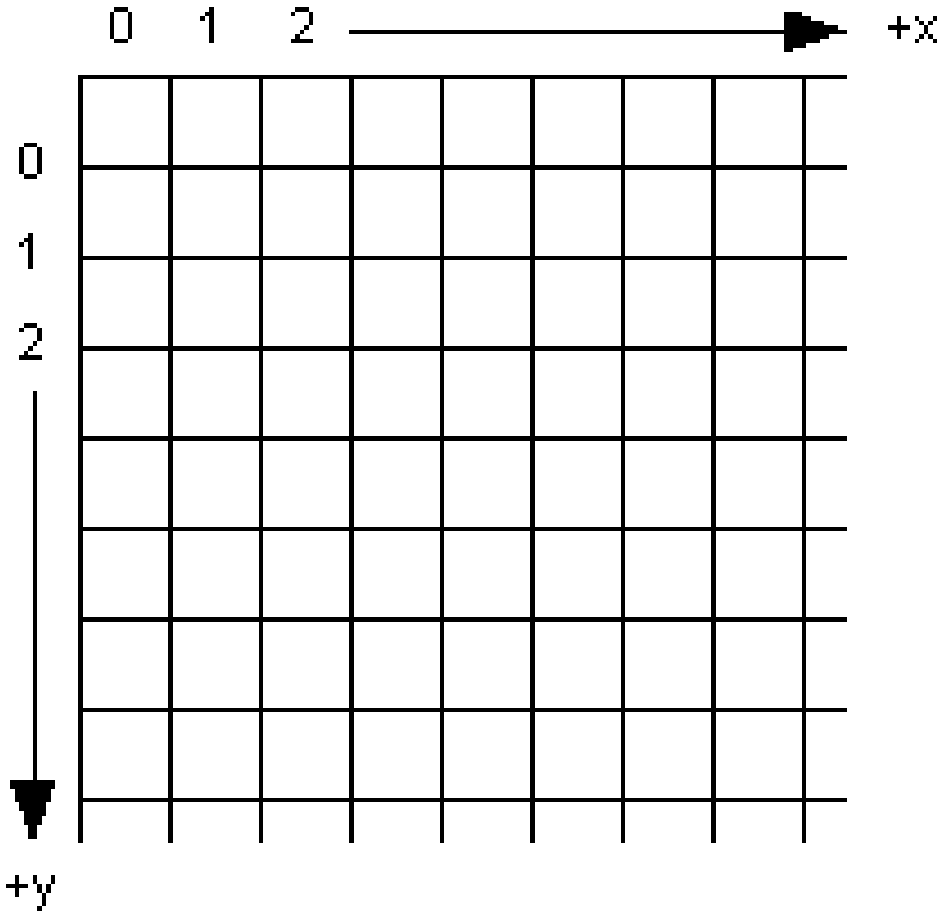
- Die Ausgabe in ein Fenster erfolgt durch Überlagern der Methode paint, die immer dann aufgerufen wird, wenn das Fenster ganz oder teilweise neu gezeichnet werden muss.
- Dies ist dann der Fall, wenn das Fenster zum ersten Mal angezeigt wird oder durch Benutzeraktionen ein Teil des Fensters sichtbar wird, der bisher verdeckt war.
- Die Methode paint bekommt beim Aufruf eine Instanz der Klasse Graphics übergeben:

```
public void paint(Graphics g)
```

- Graphics ist Javas Implementierung eines Device-Kontexts (auch Grafikkontext genannt) und stellt somit die Abstraktion eines universellen Ausgabegeräts für Grafik und Schrift dar.

Das grafische Koordinatensystem

- Die Ausgabe von Grafik basiert auf einem zweidimensionalen Koordinatensystem, dessen Ursprungspunkt (0,0) in der linken oberen Ecke liegt.



Rahmenprogramm

- Die Klasse Graphics stellt neben vielen anderen Funktionen auch eine Sammlung von linienbasierten Zeichenoperationen zur Verfügung.
- Diese sind zur Darstellung von einfachen Linien, Rechtecken oder Polygonen sowie von Kreisen, Ellipsen und Kreisabschnitten geeignet.
- Für die Beispiele wird nur die paint-Methode angegeben, die in ein Rahmenprogramm eingebunden wird.

Beispiel Rahmenprogramm

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

public class GrafikBeispiel
extends Frame
{
  public static void main(String[] args)
  {
    GrafikBeispiel wnd = new GrafikBeispiel();
  }
```

Beispiel Rahmenprogramm (Forts.)

```
  public GrafikBeispiel()
  {
    super("GrafikBeispiel");
    addWindowListener(new WindowClosingAdapter(true));
    setBackground(Color.lightGray);
    setSize(300,200);
    setVisible(true);
  }

  public void paint(Graphics g)
  {
    //wird in den folgenden Beispielen ueberlagert
  }
}
```

Linie

```
public void drawLine(int x1, int y1, int x2, int y2)
```

- Zieht eine Linie von der Position (x1,y1) zur Position (x2,y2).
- Beide Punkte dürfen an beliebiger Stelle im Fenster liegen, das Einhalten einer bestimmten Reihenfolge ist nicht erforderlich.

Beispiel Line

```
public void paint(Graphics g)
 {
   int i;
   int x = 80;

   for (i=0; i<60; ++i) {
     g.drawLine(x,40,x,100);
     x += 1+3*Math.random();
   }
 }
```

Linie (Forts.)

Rechteck

```
public void drawRect(int x, int y, int width,
                     int height)
```

- Zeichnet ein Rechteck der Breite width und der Höhe height, dessen linke obere Ecke an der Position (x,y) liegt.
- Eine Variante von drawRect ist die Methode drawRoundRect.
- arcWidth und arcHeight bestimmen den horizontalen und vertikalen Radius des Ellipsenabschnitts, der zur Darstellung der runden „Ecke“ verwendet wird.

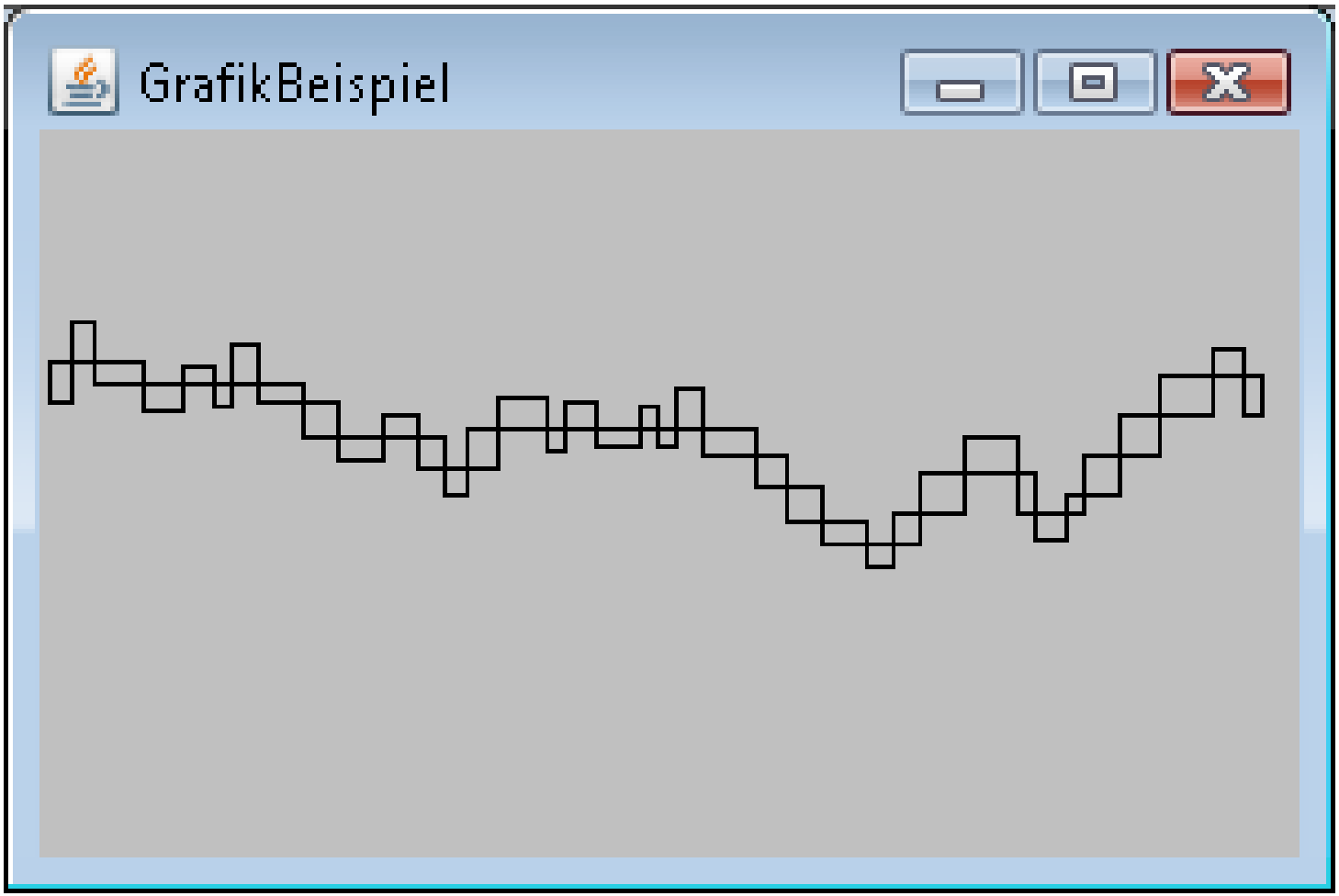
```
public void drawRoundRect(
    int x, int y,
    int width, int height,
    int arcWidth, int arcHeight
)
```

Beispiel Rechteck

```
public void paint(Graphics g)
{
  int x = 10, y = 80;
  int sizex, sizey = 0;

  while (x < 280 && y < 180) {
    sizex = 4 + (int) (Math.random() * 9);
    if (Math.random() > 0.5) {
      y += sizey;
      sizey = 4 + (int) (Math.random() * 6);
    } else {
      sizey = 4 + (int) (Math.random() * 6);
      y -= sizey;
    }
    g.drawRect(x,y,sizex,sizey);
    x += sizex;
  }
}
```

Beispiel Rechteck (Forts.)



Polygon

```
public void drawPolygon(int[] arx, int[] ary, int cnt)
```

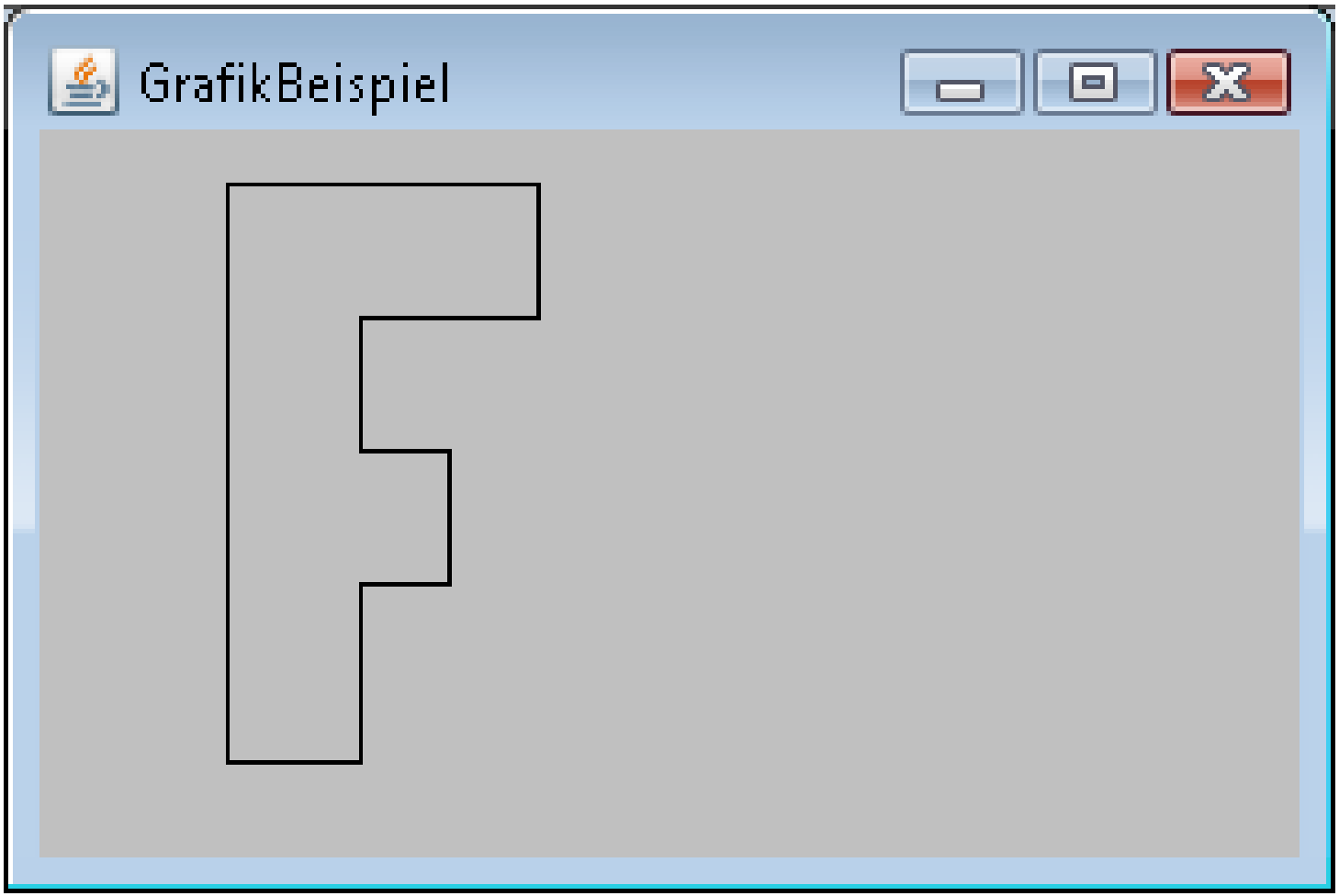
- Mit Hilfe der Methode drawPolygon ist es möglich, Linienzüge zu zeichnen, bei denen das Ende eines Elements mit dem Anfang des jeweils nächsten verbunden ist.
- Drei Parameter. Der erste ist ein Array mit einer Liste der x-Koordinaten und der zweite ein Array mit einer Liste der y-Koordinaten.
- Beide Arrays müssen so synchronisiert sein, dass ein Paar von Werten an derselben Indexposition immer auch ein Koordinatenpaar ergibt.
- Die Anzahl der gültigen Koordinatenpaare wird durch den dritten Parameter festgelegt.

Beispiel Polygon

```
public void paint(Graphics g)
{
  int[] arx = {50,50,120,120,80,80,100,100,80,80};
  int[] ary = {170,40,40,70,70,100,100,130,130,170};

  g.drawPolygon(arx,ary,arx.length);
}
```

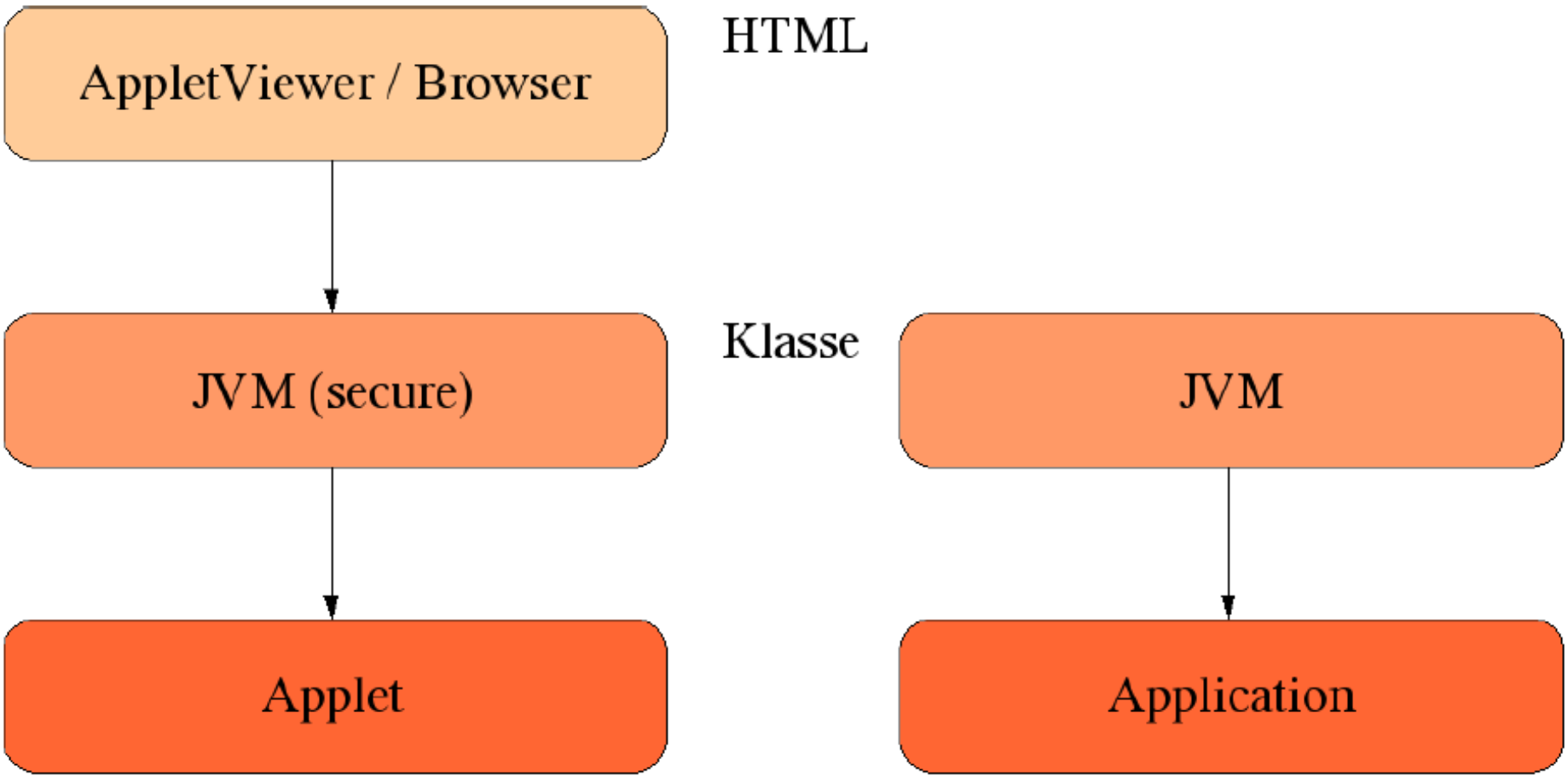
Beispiel Polygon (Forts.)



Übersicht

- Application vs. Applet
- LayoutManager : FlowLayout, GridLayout, BorderLayout
- Basiskomponenten: Panels, Buttons, TextElemente, Listen, Menüs,
- Ereignissteuerung : ActionListener, WindowListener
- individuelles Zeichnen

Applet vs. Application



Ein einfaches Applet

```
import javax.swing.*;
import java.awt.*;
public class HelloWorldApplet extends JApplet {
  public void init() {
    getContentPane().add(new JLabel("Hello␣World"));
  }
}
```

```
<html>
  <head>
    <title>Hello World</title>
  </head>
  <body>
    <applet code="HelloWorldApplet" width="200" height="100">
    </applet>
  </body>
</html>
```

Eine einfache Applikation

```
import javax.swing.*;
import java.awt.*;
public class HelloWorldApplication extends JFrame {
  public HelloWorldApplication() {
    getContentPane().add(new JLabel("Hello␣World"))
  }
  public static void main(String [] args) {
    HelloWorldApplication helloworld =
      new HelloWorldApplication();
    HelloWorld.pack();
    HelloWorld.setVisible(true);
  }
}
```

Eine einfache Applikation, zweite Version

```
import javax.swing.*;
import java.awt.*;
public class HelloWorldApplication {
  private JFrame jframe = new JFrame();
  public HelloWorldApplication() {
    jframe.getContentPane().add(new JLabel("Hello World"));
  }
  public void run() {
    jframe.pack();
    jframe.setVisible(true);
  }
  public static void main(String [] args) {
    HelloWorldApplication helloworld =
      new HelloWorldApplication();
    helloworld.run();
  }
}
```

Schließen der Applikation

```
public class HelloWorldApplication {
  private JFrame jframe = new JFrame();
  public HelloWorldApplication() {
    jframe.getContentPane().add(
      new JLabel("Hello␣World"));
    this.setDefaultCloseOperation(
      JFrame.EXIT_ON_CLOSE);
  }
  ...
}
```

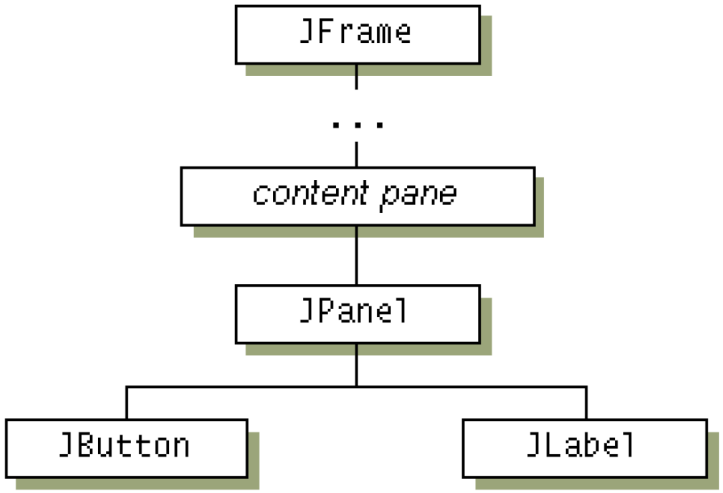
Look and feel

Look and feel der jeweiligen Plattform

```
public static void main (String args[])
  throws Exception {
  UIManager.setLookAndFeel(
    UIManager.getSystemLookAndFeelClassName());
  new MySwingFrame().setVisible(true);
}
```

Eine Hilfe zum Verstehen des JFrame-Aufbaus

- Aufbau eines JFrame



- Der Aufbau kann angezeigt werden: auf den Frame klicken und Strg+Shift+F1 drücken → erscheint auf STDOUT

```
<terminated> MyBorderLayout [Java Application] C:\Program Files\Java\jre1.6.0_03\bin\javaw.exe (10.11.2008 18:49:11)
MyBorderLayout[frame0,0,0,189x96,layout=java.awt.BorderLayout,title=,resizable,normal,defaultCloseOperation=EXIT_ON_CLOSE,rootPane=javax.swing.
 javax.swing.JRootPane[,4,23,181x69,layout=javax.swing.JRootPane$RootLayout,alignmentX=0.0,alignmentY=0.0,border=,flags=16777673,maximumSize=,n
  javax.swing.JPanel[null.glassPane,0,0,181x69,hidden,layout=java.awt.FlowLayout,alignmentX=0.0,alignmentY=0.0,border=,flags=16777217,maximumSi
  javax.swing.JLayeredPane[null.layeredPane,0,0,181x69,alignmentX=0.0,alignmentY=0.0,border=,flags=0,maximumSize=,minimumSize=,preferredSize=,c
   javax.swing.JPanel[null.contentPane,0,0,181x69,layout=java.awt.BorderLayout,alignmentX=0.0,alignmentY=0.0,border=,flags=9,maximumSize=,minin
    javax.swing.JButton[,0,0,181x23,alignmentX=0.0,alignmentY=0.5,border=javax.swing.plaf.BorderUIResource$CompoundBorderUIResource@a39137,flag
    javax.swing.JButton[,0,46,181x23,alignmentX=0.0,alignmentY=0.5,border=javax.swing.plaf.BorderUIResource$CompoundBorderUIResource@92e78c,fla
    javax.swing.JButton[,126,23,55x23,alignmentX=0.0,alignmentY=0.5,border=javax.swing.plaf.BorderUIResource$CompoundBorderUIResource@9fbe93,fl
    javax.swing.JButton[,0,23,59x23,alignmentX=0.0,alignmentY=0.5,border=javax.swing.plaf.BorderUIResource$CompoundBorderUIResource@198dfaf,fla
    javax.swing.JButton[,59,23,67x23,alignmentX=0.0,alignmentY=0.5,border=javax.swing.plaf.BorderUIResource$CompoundBorderUIResource@1858610,fl
```

Ein Button

```
import javax.swing.*;
import java.awt.*;
public class MyButton extends JFrame {
   JButton button;
   public MyButton() {
      Container cp = getContentPane();
      button = new JButton("Button␣1");
      cp.add(button);
   }
   public static void main (String [] args) {
      MyButton1 mybutton = new MyButton();
      mybutton.pack();
      mybutton.setVisible(true);
   }
}
```

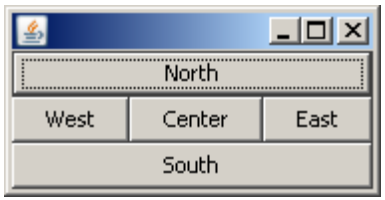
Eine einfache Ereignissteuerung

```
public class MyButton extends JFrame {
  JButton button;
  public MyButton() {
    Container cp = getContentPane();
    button = new JButton("Button 1");
    button.addActionListener(new ActionLister(){
      public void actionPerformed(ActionEvent e) {
        System.out.println("Der Button wurde gedrueckt!");
      }
    });
    cp.add(button);
  }
  public static void main(String [] args) {
    MyButton mybutton = new MyButton();
    mybutton.pack();
    mybutton.setVisible(true);
  }
}
```

Übersicht

- JApplet, JFrame, JWindow und JDialog
  - produzieren mittels getContentPane() einen Container
  - einem Container kann mittels setLayout(LayoutManager) ein anderes Layout verpasst werden
- JPanel und andere direkte Unterklassen von JComponent können direkt mittels setLayout() modifiziert werden.
- typische LayoutManager : BorderLayout, FlowLayout, GridLayout, GridBagLayout, BoxLayout

BorderLayout

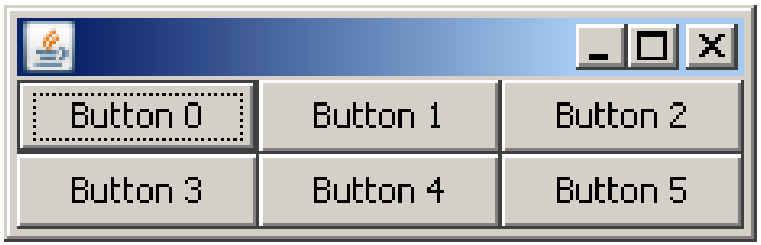



```
public class MyBorderLayout extends JFrame {
  public MyBorderLayout() {
    Container cp = getContentPane();
    cp.setLayout(new BorderLayout());
    cp.add(BorderLayout.NORTH, new JButton("North"));
    cp.add(BorderLayout.SOUTH, new JButton("South"));
    cp.add(BorderLayout.EAST, new JButton("East"));
    cp.add(BorderLayout.WEST, new JButton("West"));
    cp.add(BorderLayout.CENTER, new JButton("Center"));
  }
  public static void main(String[] args) {
    MyBorderLayout test = new MyBorderLayout();
    test.pack(); test.setVisible(true);
  }
}
```

FlowLayout


```
public class MyFlowLayout extends JFrame {
  public MyFlowLayout() {
    Container cp = getContentPane();
    cp.setLayout(new FlowLayout());
    for (int i = 0; i < 5; i++) {
      cp.add(new JButton("Button " + i));
    }
  }
  public static void main(String[] args) {
    MyFlowLayout test = new MyFlowLayout();
    test.pack(); test.setVisible(true);
  }
}
```

GridLayout




```
public class MyGridLayout extends JFrame {
  public MyGridLayout() {
    Container cp = getContentPane();
    cp.setLayout(new GridLayout(2, 3));
    for (int i = 0; i < 6; i++) {
      cp.add(new JButton("Button " + i));
    }
  }
  public static void main(String[] args) {
    MyGridLayout test = new MyGridLayout();
    test.pack(); test.setVisible(true);
  }
}
```

BoxLayout


```
public class MyBoxLayout extends JFrame {
  public MyBoxLayout() {
    Container cp = getContentPane();
    cp.setLayout(new BoxLayout(cp, BoxLayout.Y_AXIS));
    for (int i = 0; i < 6; i++) {
      cp.add(new JButton("Button " + i));
    }
  }
  public static void main(String[] args) {
    MyBoxLayout test = new MyBoxLayout();
    test.pack(); test.setVisible(true);
  }
}
```

JPanel

- Behälter für andere JComponenten (JPanel, JButtons, JTextField)
- ein JPanel zeichnet nur seinen Hintergrund (default)
- benutzt einen LayoutManager zum JComponenten im JPanel anzuordnen (voreingestellt: FlowLayout())
- häufigste Anwendung : Positionieren von JCompents in JFrame, JApplet

JPanel (Forts.)


```
JPanel first = new JPanel();
JPanel second = new JPanel();
public MyPanel() {
  Container cp = getContentPane();
  cp.setLayout(new FlowLayout());
  first.setLayout(new BorderLayout());
  first.add(BorderLayout.NORTH,new JButton("North"));
  ...
  first.add(BorderLayout.CENTER,new JButton("Center"));
  cp.add(first);
  cp.add(new JLabel("ein einfaches Label"));
  second.setLayout(new GridLayout(2,3));
  for (int i = 0; i < 6; ++i){
    second.add(new JButton("Label "+i));
  }
  cp.add(second);
...
```

Ereignissteuerung

- eine Komponente kann ein Ereignis (Event) auslösen
- jedes Event wird durch eine Klasse repräsentiert (z.B. ActionListener
- eine ausgelöstes Event kann durch ein oder mehrere „Listener“ verarbeitet werden
- strikte Trennung von Quelle des Events und Ziel der Verarbeitung ist möglich (und häufig sinnvoll)
- ähnliche Methoden zum Hinzufügen und Entfernen von Listenern: addXXXListener(XXXListener) und removeXXXListener()

Listener als anonyme interne Klasse

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

public class MyButton extends JFrame {
  JButton button;

  public MyButton() {
    Container cp = getContentPane();
    button = new JButton("Button 1");
    button.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        System.out.println("Der Button wurde gedrueckt!");
      }
    });
    cp.add(button);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
```

Listener als externe Klasse

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

class MyActionListener implements ActionListener {
  public void actionPerformed(ActionEvent e) {
    System.out.println("Der Button wurde gedrueckt!");
  }
}
```

Listener als externe Klasse (Forts.)

```
public class MyButton extends JFrame {
  JButton button;

  public MyButton() {
    Container cp = getContentPane();
    button = new JButton("Button 1");
    button.addActionListener(new MyActionListener());
    cp.add(button);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }

  public static void main(String[] args) {
    MyButton mybutton = new MyButton();
    mybutton.pack();
    mybutton.setVisible(true);
  }
}
```

Listener als eigene interne Klasse

```
public class MyButton extends JFrame {
  JButton button;

  class MyActionListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      System.out.println("Der Button wurde gedrueckt!");
    }
  }

  public MyButton() {
    Container cp = getContentPane();
    button = new JButton("Button 1");
    button.addActionListener(new MyActionListener());
    cp.add(button);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
```

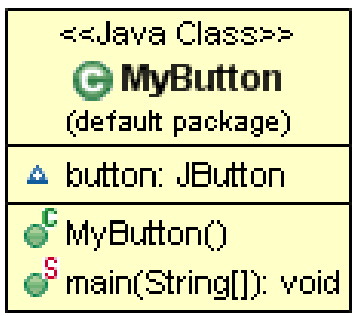
Listener als eigene interne Klasse (Forts.)

```
    public static void main(String[] args) {
        MyButton mybutton = new MyButton();
        mybutton.pack();
        mybutton.setVisible(true);
    }
}
```

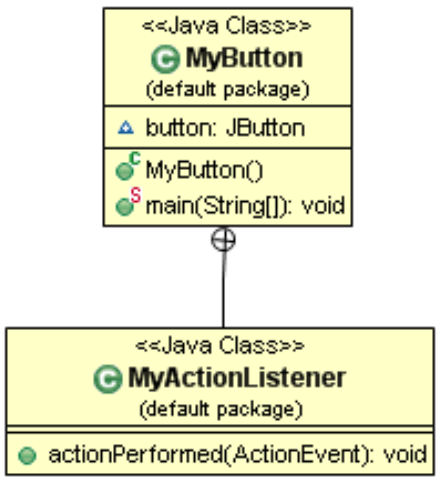
Listener in der Klasse

```
public class MyButton extends JFrame
  implements ActionListener {
  JButton button;
  public MyButton() {
    Container cp = getContentPane();
    button = new JButton("Button 1");
    button.addActionListener(this);
    cp.add(button);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
  public void actionPerformed(ActionEvent e) {
    System.out.println("Der Button wurde gedrueckt!");
  }
  public static void main(String[] args) {
    MyButton mybutton = new MyButton();
    mybutton.pack();
    mybutton.setVisible(true);
  }
}
```

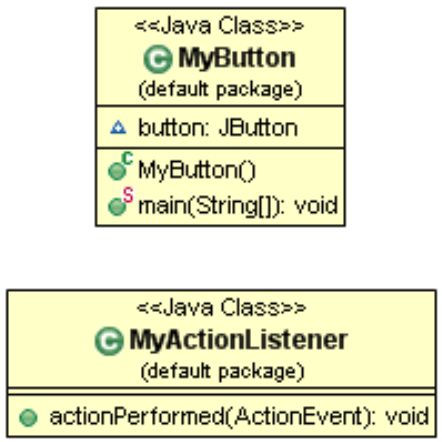
ActionListener in der Übersicht



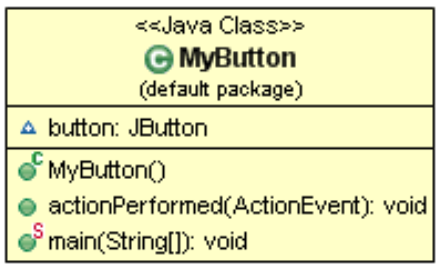
anonym



intern



extern



in der Klasse

Leichtgewichtige Komponenten (Lightweight Components)

- Keine plattformspezifischen Besonderheiten: der Code zur Implementierung der Dialogelemente vereinfacht sich im Vergleich zu AWT deutlich.
- Keine Unterschiede in der Bedienung: Der Anwender findet auf allen Betriebssystemen dasselbe Aussehen und dieselbe Bedienung vor.
- Komplexe Dialogelemente wie Bäume, Tabellen oder Registerkarten können plattformunabhängig realisiert werden.
- Swing stellt eine sehr viel umfassendere und anspruchsvollere Menge an Dialogelementen als das AWT zur Verfügung.

Leichtgewichtige Komponenten (Forts.)

- In Swing wird die paint-Methode eines Component-Objects nicht mehr an die betriebssystemspezifische Klasse weitergeleitet, sondern in den Komponentenklassen überlagert und mit Hilfe grafischer Primitivoperationen selbst implementiert.
- Die im AWT vorhandenen Dialogelemente werden im Gegensatz dazu als schwergewichtige Komponenten bezeichnet (Heavyweight Components).

Model-View-Controller-Prinzip

Drei unterschiedliche Bestandteile eines grafischen Elements werden unterschieden:

- Das Modell enthält die Daten des Dialogelements und speichert seinen Zustand.
- Der View ist für die grafische Darstellung der Komponente verantwortlich.
- Der Controller wirkt als Verbindungsglied zwischen beiden. Er empfängt Tastatur- und Mausereignisse und stößt die erforderlichen Maßnahmen zur Änderung von Model und View an.

Model-Delegate-Prinzip

- Vereinfachte Variante von MVC in Swing
- Funktionalität von View und Controller in einem UI Delegate zusammengefasst
- Komplexität reduziert (oft ist der Controller so einfach strukturiert, dass es sich nicht lohnt, ihn separat zu betrachten)
- In der Praxis mitunter unhandliche Trennung zwischen View und Controller aufgehoben

Zusammenfassung: Benutzeroberflächen

- Elementare Grafik ist im Abstract Windowing Toolkit realisiert.
- Benutzeroberflächen können mit dem Swing-Paket implementiert werden.
- Ereignissteuerung erfolgt mittels Listenern, z. B. ActionListener.