AWT

AWT

Einführung
Components and Containers
Layoutmanagement
Graphikausgabe und AWT-Thread
Ereignisse
MVC Architektur
Komponenten
Zusammenfassung

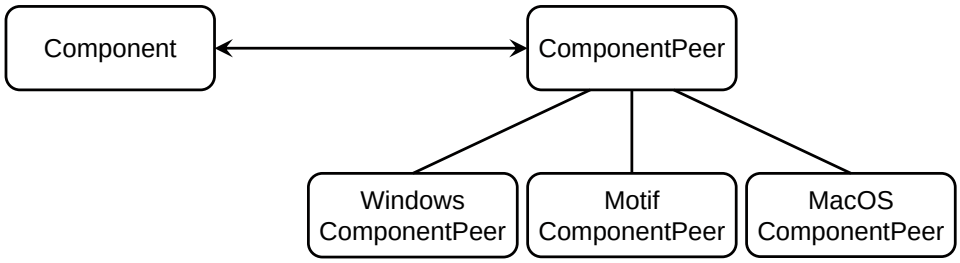
AWT - Abstract Window Toolkit

- Einfache Library für graphische Benutzeroberflächen
- Paket `java.awt` und Unterpakete
- Abstrahiert GUI-Komponenten von konkreter Plattform
- Plattformspezifische Implementierung in "Peer"-Klasse
- Kleinster gemeinsamer Nenner aller Plattformen

Components und Peers

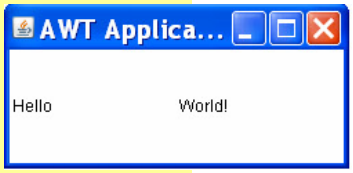
- *Component* stellt Benutzerkomponente dar, z.B.: Window, Button, TextField, Canvas, …
- In AWT alle Komponenten „native“ dargestellt,
  d.h. werden auf Komponenten des Betriebssystems abgebildet

  = „Heavyweight Components“
- *ComponentPeer* kapselt plattformspezifischen Aufruf (z.B. Windows API)



Aufbau von Applikationen

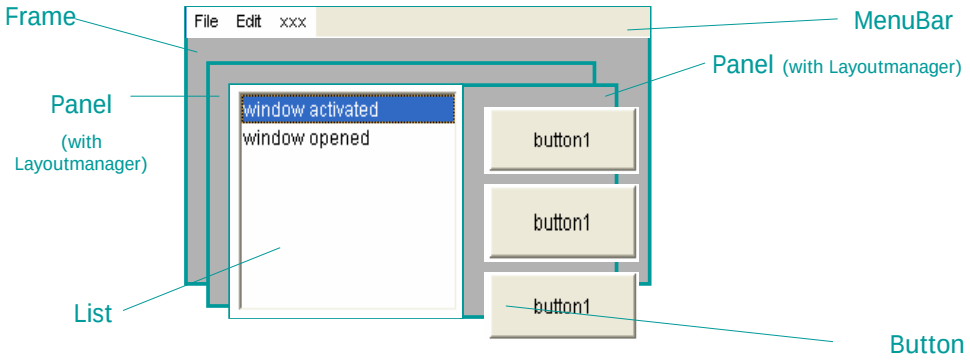
```
import java.awt.Frame;
import java.awt.GridLayout;
import java.awt.Label;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class AWT11_Frame {
    public static void main(String[] args) {
        SimpleFrameApp app = new SimpleFrameApp();
    }
}

class SimpleFrameApp {
    Frame frame;
    public SimpleFrameApp() {
        frame = new Frame("AWT Application");
        frame.addWindowListener(new WindowAdapter() {
           public void windowClosing(WindowEvent e) {
              frame.setVisible(false);
              System.exit(0);
           }
        });
        frame.setLayout(new GridLayout(1, 2));
        frame.add(new Label("Hello"));
        frame.add(new Label("World!"));
        frame.setSize(250, 120);
        frame.setLocation(100, 200);
        frame.setVisible(true);
    }
}
```



AWT

Einführung
Components and Containers
Layoutmanagement
Graphikausgabe und AWT-Thread
Ereignisse
MVC Architektur
Komponenten
Zusammenfassung

Hierarchischer Aufbau von AWT-GUIs



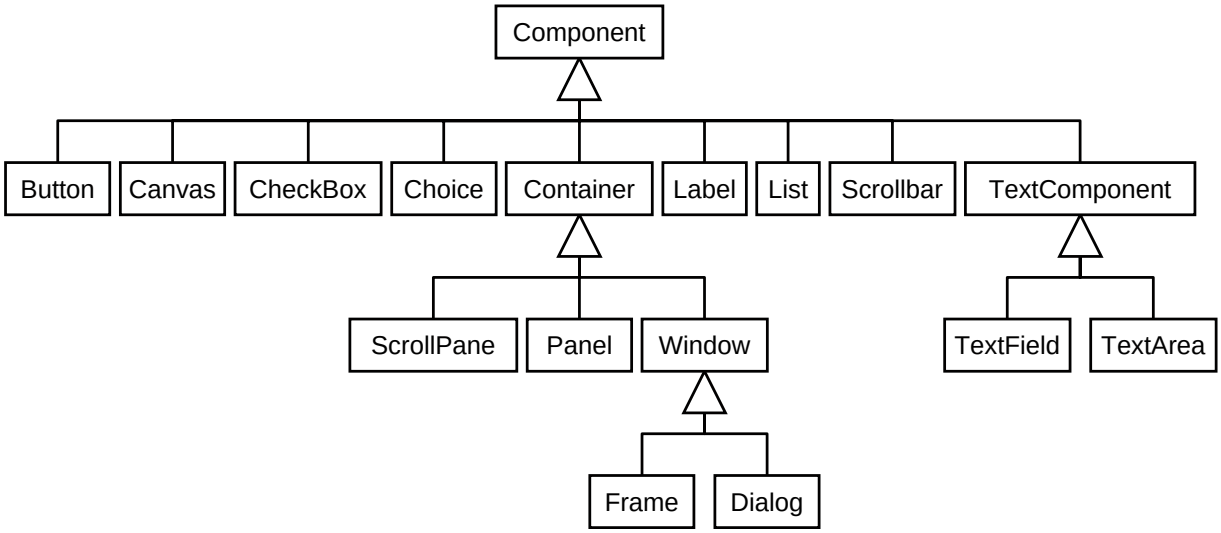
hierarchische Anordnung

- es gibt *Components* und *Containers*
- Components liegen in Container
- Container sind Components

Container hat *Layoutmanager* – bestimmt Anordnung (x,y,w,h) der Components

Components und Containers reagieren auf Benutzeraktionen mit Ereignissen

Komponenten von AWT



Die Basisklasse `java.awt.Component`

- abstrakte Oberklasse aller Benutzerschnittstellenkomponenten
- hat Position (`location`) und Größe (`size, width, height`)
- hat eine grafische Repräsentation (siehe später)
- Reagiert auf Benutzeraktionen mit Ereignissen (siehe später)

Wichtige Methoden:

`setBounds(int x, y, w, h)`	Position und Größe der Komponente setzen
`Rectangle getBounds()`	Position und Größe abfragen
`int getX(), int getY()`	X-, Y-Position abfragen
`int getWidth(), int getHeight()`	Breite, Höhe abfragen
`setLocation(int x, int y)`	Position setzen
`Point getLocation()`	Position abfragen
`setSize(int w, int h)`	Größe setzen
`Dimension getSize()`	Größe abfragen
`setForeground(Color fg)`	Vordergrundfarbe (für Text, Linien, etc.) setzen
`setBackground(Color bg)`	Hintergrundfarbe setzen
`setFont(Font f)`	Schriftart, die für Textausgaben verwendet werden soll, setzen
`setEnabled(boolean enabled)`	bestimmen, ob auf Benutzeraktionen reagieren oder nicht
`setVisible(boolean)`	Setzen, ob Sichtbar oder verborgen
`Container getParent()`	Abfragen des Containers, in dem sie sich Komponente befindet
…	…

Oberklasse `java.awt.Container`

- Container für die hierarchische Anordnung von Components
- Containers sind selbst Components (*Composite Pattern*)
- Containers verwenden Layoutmanager für Anordnung der Components

Wichtige Methoden:

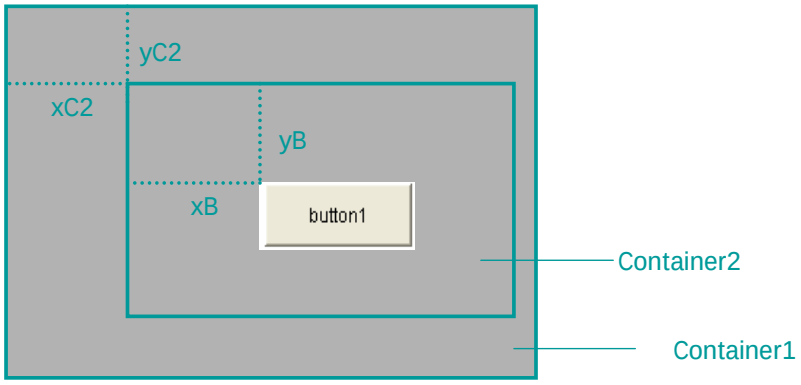
`add(Component c)`	Component hinzufügen
`remove(Component c)`	Component entfernen
`removeAll()`	Alle Components entfernen
`int getComponentCount()`	Anzahl der Components
`setLayout(LayoutManager l)`	LayoutManager setzten (ev: null -> kein LM)
`validate()`	Der Layoutmanager soll alle Components neu anordnen
`validateTree()`	wie oben, mit rekursivem Abstieg

Koordinatensystem bei `Container`

Jeder Container definiert eigenes Koordinatensystem

- mit (0, 0)-Position links oben

D.h. Positionierung von Components immer relativ zum Container



Wichtige Subklassen von Container

Frame

- Hauptfenster eigenständiger Anwendungen
- Rahmen und Titelleiste

Dialog

- Fenster, das zu einem übergeordneten Frame gehört
- Parameter in Konstruktor bestimmt Modalität

Panel

- Container für andere Komponenten
- verschachtelbar zur Kombination von Layout-Managern

ScrollPane

- von Panel abgeleitet
- mit Bildlauf
- kann eine einzige Komponente enthalten

AWT

Einführung

Components and Containers

Layoutmanagement

Graphikausgabe und AWT-Thread

Ereignisse

MVC Architektur

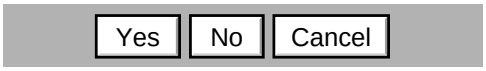
Komponenten

Zusammenfassung

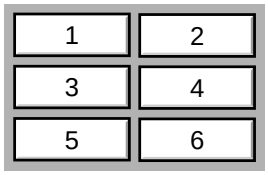
Layoutmanager

- Layoutmanager bei Container definiert Anordnung seiner Komponenten
- logisch (!) beschrieben und nicht in absoluten Koordinaten
- Unterschiedliche Typen von Layoutmanager für unterschiedliche Strategien der Anordnung
- Layoutmanager durch Klassen realisiert

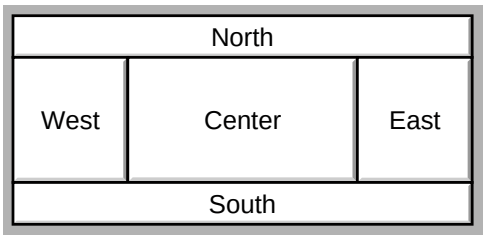
FlowLayout



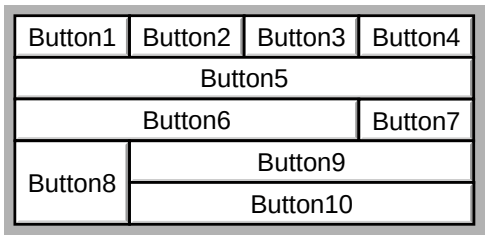
GridLayout



BorderLayout



GridBagLayout



Grundprinzip

Instanz eines Layoutmanagers erzeugen und beim Container setzen

Größe der Components bestimmen: 3 Größeneigenschaften

- `preferredSize`: bevorzugte Größe
- `minimumSize`: minimale Größe
- `maximumSize`: maximale Größe

Achtung: werden von den unterschiedlichen Layoutmanagern unterschiedlich verwendet!

Components anfügen

LayoutManager bestimmt automatisch Position und tatsächliche Größe

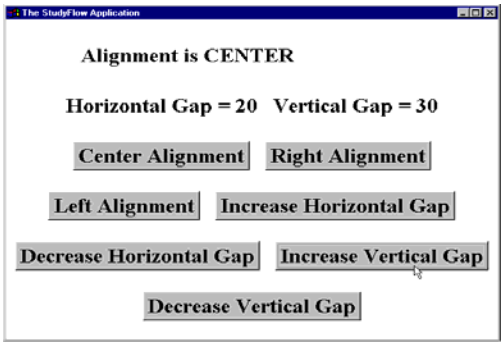
```
aContainer.setLayout(new XYZLayout(…));
...
component1.setPreferredSize(new Dimension(100, 200));
component1.setPreferredSize(new Dimension(200, 100));
otherContainer.setPreferredSize(new Dimension(100, 100));

aContainer.add(component1);
aContainer.add(component2);
aContainer.add(otherContainer);
// Layoutmanager bestimmt Positionen der Components im Container
```


Layoutmanager-Klassen

- **`FlowLayout`**
  Ordnet die Komponenten wie Wörter auf einer Seite an (zentriert, rechts- oder linksbündig)
- **`BorderLayout`**
  Unterstützt genau 5 Komponenten: eine an jeder Seite und eine im Zentrum
- **`GridLayout`**
  Teilt den Container in Zeilen und Spalten. Alle Zellen sind gleich groß.
- **`GridBagLayout`**
  Sehr flexibel und kompliziert. Für jede Komponente können Einschränkungen definiert werden.
- **`CardLayout`**
  Zeigt jeweils nur eine Komponente, die den Container ausfüllt. Die anderen liegen quasi unsichtbar dahinter und man blättert bei Bedarf weiter (z.B. Tabs)

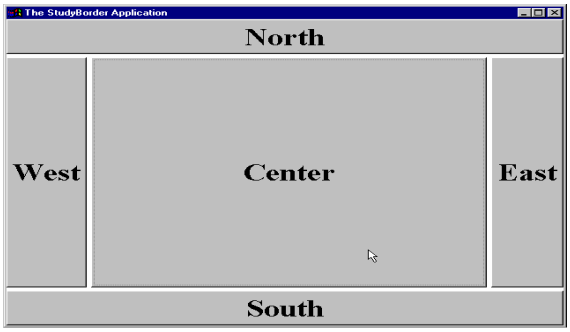
FlowLayout

```
f = new FlowLayout(FlowLayout.CENTER, 20, 20) // Zentriert; horizontaler
                                              // und vertikaler Abstand 20
container.setLayout(f);
container.add(component1);
container.add(component2);
container.add(containerX);
```



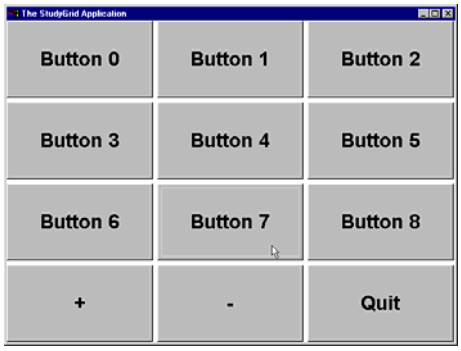
BorderLayout

```
container.setLayout(new BorderLayout());
container.add(component1, BorderLayout.NORTH);
container.add(component2, BorderLayout.CENTER);
container.add(containerX, BorderLayout.WEST);
...
```



GridLayout

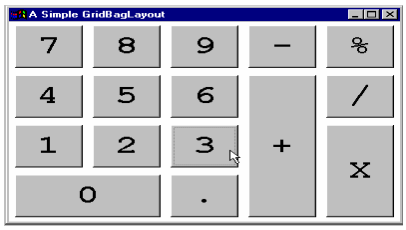
```
g = new GridLayout(rows, cols);    // Zeilen und Spalten
container.setLayout(g);
container.add(component1);         // Auffüllen von links oben zeilenweise
                                   // nach rechts unten
container.add(component2);
container.add(containerX);
```



Weitere

GridBagLayout:

- verwendet Grid
- Components können mehrere Zellen überspannen
- Sehr komplex, wenig verwendet



CardLayout

- mehrere Cards, wobei immer eines sichtbar

Weitere Layoutmanager im Packet Swing definiert (siehe Swing-Teil)

Schreiben von eigenen Layoutmanager-Klassen möglich

null-Layout

- Beim Arbeiten ohne LayoutManager muss man selbst für die Anordnung der Komponenten sorgen.
- Bei Größenänderungen erfolgt keine automatische Anpassung.

```
container.setLayout(null); // explizit nötig, um default-layout zu umgehen
component1.setBounds(10,10,100,100); // x, y, Breite, Höhe
component2.setBounds(10,150,100,100); // x, y, Breite, Höhe
container.add(component1);
container.add(component2);
```


Neu anordnen

Bei Änderungen von Components werden diese nicht automatisch angeordnet

Neu anordnen mit folgenden Methoden:

`invalidate()` – vermerkt, dass eine Component und alle ihre Vorgänger neu angeordnet werden müssen

`validate()` – wird bei Container aufgerufen, um die Komponenten neu anzuordnen; nur aktiv wenn *invalid* vermerkt ist

`validateTree()` – wie `validate()` aber hierarchisch in die Tiefe

```
aContainer.setLayout(new XYZLayout(…));
...
component1.setPreferredSize(new Dimension(100, 200);
...
aContainer.add(component1);
...
component1.setPreferredSize(new Dimension(200, 300));
component1.invalidate();
aContainer.validate();
// Components im Container werden neu angeordnet
```

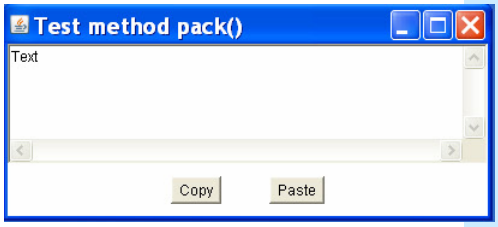
Methode `pack()` bei `Window`s

Methode `pack()` von `Window` für Bestimmung einer optimalen Fenstergröße

Größe des Windows und aller seiner Subcomponents so, dass alle optimal Platz finden (Verwendung von `preferredSize`)

Vorgehen:

- Hierarchie der Components aufbauen
- Layoutmanager setzen
- `pack` aufrufen
- Window sichtbar machen




```
class PackTestFrame extends Frame {

  TextArea textArea;
  Panel panel1, panel2;
  Button b1, b2;

  public PackTestFrame() {
    ...
    panel1 = new Panel();
    panel1.setLayout(new BorderLayout());
    textArea = new TextArea("Text");
    textArea.setPreferredSize(new Dimension(400, 100));
    panel1.add(textArea, BorderLayout.CENTER);
    panel2 = new Panel();
    panel2.setLayout(new FlowLayout());
    b1 = new Button("Copy");
    b2 = new Button("Paste");
    panel2.add(b1);
    panel2.add(b2);
    panel1.add(panel2, BorderLayout.SOUTH);
    add(panel1);
    pack();
    setVisible(true);
  }
}
```


AWT

Einführung

Components and Containers

Layoutmanagement

Graphikausgabe und AWT-Thread

Ereignisse

MVC Architektur

Komponenten

Zusammenfassung

Grundlagen der Graphikausgabe

Components zeichnen sich mit Methode

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

wobei der `Graphics`-Parameter die Zeichenfläche für die Component darstellt

Anwenderprogramm darf `paint` nicht aufrufen,
sondern Aufruf der `paint`-Methoden erfolgt im *AWT-Loop*

Anwenderprogramm kann durch Aufruf von

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

das Neuzeichnen der Komponente (eines Bereichs der Komponente) „fordern“

AWT-Loop

Anwenderprogramm und AWT-Loop laufen in unterschiedlichen Threads

Anwenderprogramm

- führt Änderungen in den Daten durch, die Änderungen in der GUI bewirken sollen
- ruft `repaint` der betroffenen Component auf

AWT-Loop

- durch `repaint` weiss AWT, dass Component neu gezeichnet werden muss
- AWT-Loop arbeitet die Zeichenanforderungen ab
- bewirkt Neuzeichnen der Components

AWT-Loop

Neuzeichnen:

- AWT-Loop ruft Methode `update` bei der Component auf
- `update` ruft `paint` auf

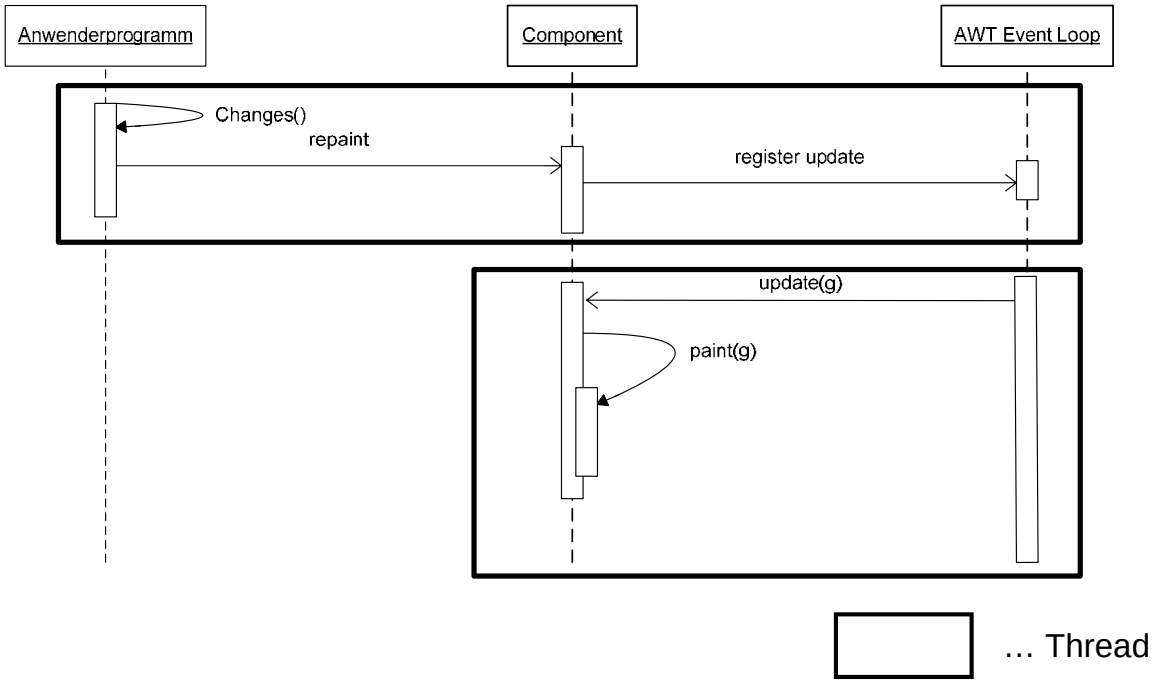
```
public void update(Graphics g) {
  paint(g);
}
```

- `paint` führt Zeichnen der Component durch

```
public void paint(Graphics g) {
  ...
}
```

- `update` wird verwendet, um in Painting grundsätzlich einzugreifen (z.B.: DoubleBuffering)
- `paint` wird überschrieben, um bei Components spezielle Darstellungen zu realisieren

AWT-Loop



Beachte: `update` und `paint` laufen im AWT-Thread
➔ lang laufende `paint`-Methoden blockieren AWT-Thread

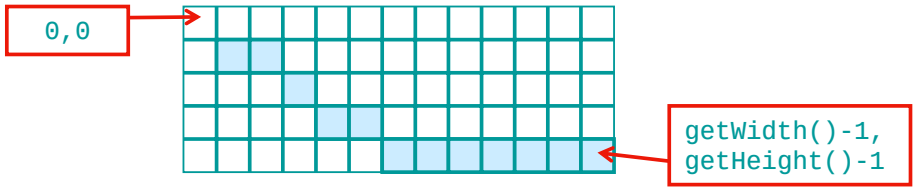
Graphikkontext `Graphics`

In Methode `paint` wird als Parameter ein `Graphics`-Objekt übergeben

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

Stellt virtuelle Zeichenfläche für die Component dar (genannt *Graphikkontext*)

- hat Größe der Component mit 0/0-Koordinate im linken, oberen Eck der Component



- ermöglicht Zeichnen von Linien, Figuren, Text und Bildern, …
- hat eine Reihe von Zuständen, die die Grafikausgabe beeinflussen
  - `Color`: gegenwärtige Zeichenfarbe
  - `Font`: gegenwärtige Schriftart für Text
  - `Clip`: Zeichenoperationen wirken sich nur innerhalb des Clip aus)
  - `XORMode`: logische Pixeloperation (XOR oder Paint)

Zeichenoperationen der Klasse Graphics

```
clearRect(int x,int y,int w, int h)
copyArea(x,y, w,h, dx, dy)
draw3DRect(x,y,w,h, boolean raised)
drawArc(x,y,w,h, int startAngle, int arcAngle)
drawLine(x1,y1,x2,y2)
drawOval(x,y,w,h)
drawPolygon(x[], y[], int nPoints)
drawPolyLine(x[], y[], int nPoints)
drawRect(x,y,w,h)
drawRoundRect(x,y,w,h, int arcWidth, int arcHeight)
drawString(String str, x,y)
fill3DRect(x,y,w,h, boolean raised)
fillArc(...)
fillOval(...)
fillPolygon(...)
fillRoundRect(...)
drawImage(Image img, x,y, ImageObserver obs)

....
```

Weitere Methoden der Klasse Graphics

translate(int x, int y)	// Verschieben des Koordinatenursprungs
setPaintMode()	// normalen Zeichenmodus setzen (default)
setXORMode(Color c1)	// XOR-Modus momentane Farbe xor c1
setFont(Font f)	// Schriftart setzen
setColor(Color c)	// Zeichenfarbe setzten
setClip(x,y,w,h)	// setzt neues Clip (Pixel nur innerhalb betroffen)
FontMetrics getFontMetrics()	// holt Metrik-informationen zur Font
	// zB. wieviele Pixel breit/hoch ist "1. Object"
Font getFont()	// aktuelle Schriftart
Color getColor()	
Rectangle getClipBounds()	// Rechteckigen Clip-Bereich holen; Auch wenn
....	eine Linie über diesen hinausgeht, wird nur
	innerhalb gezeichnet.

Grafikausgabe

Vorgehen:

- Klasse von `Canvas` ableiten
- `paint`-Methode überschreiben
- Canvas-Objekt in Frame einfügen

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

public class PaintTestCanvas extends Canvas {
    public void paint(Graphics g) {
        super.paint(g); // Löschen
        for (int x = 0; x <= getWidth()-1; x++) {
            g.setColor(new Color(0, (int)(255.0/getWidth()*x),
                                    (int)(255.0/getWidth()*x)));
            g.drawLine(x, 0, getWidth()-x, getHeight()-1
        }
    }

  public static void main(String[] args) {
    Frame frame = new Frame("Paint test");
    frame.add(new PaintTestCanvas());
    frame.setSize(new Dimension(200, 200));
    frame.setVisible(true);
  }
}
```

Font und Fontmetrics

Die Klasse Font

- Repräsentiert eine Schriftart:
- Konstruktion mit:
- `Font f = new Font("Monospaced", Font.BOLD, 14);`
- 3 Standard-Namen: "MonoSpaced", "Serif", "SansSerif"

Einige Methoden:

```
Font (String name, int style, int size)
int getSize()              // Schriftgröße in Pixel
boolean isBold()           // fett
boolean isItalic()         // kursiv
static final int PLAIN     // Konstanten für Schriftstil
static final int BOLD
static final int ITALIC
... uvm.
```


Font und Fontmetrics

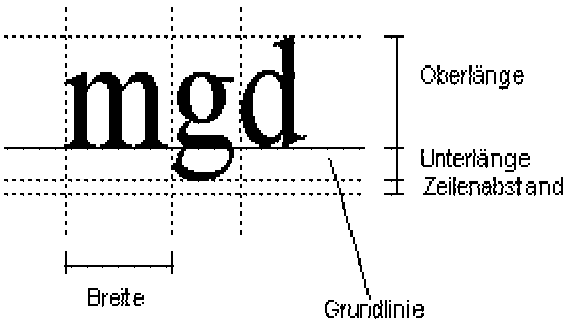
Die Klasse Fontmetrics

- Stellt typographische Informationen über einzelne Zeichen und Strings in konkreten Schriftarten zur Verfügung.
- Erreicht man am besten über Grafikkontext:

  `FontMetrics fm = g.getFontMetrics();`
- oder `fm = g.getFontMetrics(aFont);`

nützliche Methoden:

```
int charWidth(char ch) // Breite ch
int stringWidth(String s) // Breite s
int getAscent()  // Oberlänge
int getDescent() // Unterlänge
int getHeight()  // Höhe
int getLeading() // Zeilenabstand
```



Images

Klasse `Image` für Bitmaps

Images können

- von Datei geladen werden
- von Netzwerk geladen werden
- aus Bytes-Strom erzeugt werden
- …

`drawImage` für Ausgabe

```
public void paint(Graphics g) {
  Image img =
    getToolkit().getImage("duke.gif");
  g.drawImage(img, 100, 100, this);
}
```

Hilfsklasse `Toolkit`

- Zugriff auf Bildschirmeigenschaften und Hilfsmethoden
- Z.B.: Erzeugen von Images
  - Image getImage(String fname)
- Zugriff auf Toolkit mit `getToolkit()` von `Component`

```
public class Toolkit {
  public abstract Dimension getScreenSize()
  public abstract int getScreenResolution()
  public abstract Image getImage(String filename)
  public abstract Image getImage(URL url)
  public abstract void beep()
  public abstract Clipboard getSystemClipboard()
  public Clipboard getSystemSelection()
  …
}
```


Animation

Für bewegte Bilder braucht man Thread

```
class MovingObject implements Runnable {
  ...
  public void run() {
    while (true) {
      try {
        Thread.sleep(delayTime);
      } catch (InterruptedException e) {
      }
      move();
      repaint();
    }
  }
}
```

Verzögerung

Bewegung
Neu zeichnen

Verwendung von repaint mit Bereichsangabe

Optimierte Ausgabe, weil nicht alles gezeichnet werden muss

```
...
move();
repaint(minX, minY, width, height);
...
```

Bereich der neu zu zeichnen ist

Beispiel: MovingImage

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

class MovingImageCanvas extends Canvas implements Runnable {

  private Point point = new Point(0, 50);
  private int speed;
  private Image img;

  public MovingImageCanvas(int speed, String imgName) {
    this.speed = speed;
    img = getToolkit().getImage(imgName);
  }

  private void move() {
    point.x = point.x + speed;
  }

  public void paint(Graphics g) {
    g.drawImage(img, point.x % this.getWidth(), point.y, this);
  }

  public void run() {
    while (true) {
      try {
        Thread.sleep(10);
      } catch (InterruptedException e) {}
      move();
      repaint(point.x % this.getWidth()-speed, point.y,
              img.getWidth(this), img.getHeight(this));
    }
  }
}
```

Double-Buffering

Durch Löschen des Bildschirms und Neuzeichnen kommt es zu flackern

Abhilfe durch Double-Buffering

- zuerst Zeichnen in ein Image
- dann Zeichnen des neuen Image auf den Bildschirm

Realisiert in `update(Graphics g)`

```
public void update(Graphics g) {
  if (dbImage == null) {
    dbImage = createImage(getWidth(), getHeight());
    dbGraphics = dbImage.getGraphics();
  }
  dbGraphics.setColor(getBackground());
  dbGraphics.fillRect(0, 0, getWidth(), getHeight());
  dbGraphics.setColor(getForeground());
  paint(dbGraphics);
  g.drawImage(dbImage,0,0,this);
}
```

Image erzeugen
Graphics-Objekt für Image erzeugen

Hintergrund im Image füllen

Zeichnen in Image
Image auf Bildschirm zeichnen

AWT

Einführung

Components and Containers

Layoutmanagement

Graphikausgabe und AWT-Thread

Ereignisse

MVC Architektur

Komponenten

Zusammenfassung

Ereignisse

Komponenten lösen Ereignisse aus

- z.B. Klick auf Button, Auswahl eines Listenelements, Eingabe von Text

Ereignisbehandler reagieren auf Ereignis

- Implementation der entsprechenden Schnittstelle
- Registration bei Komponente

Ereignisobjekt liefert Zusatzinformationen

- z.B. Quelle, Modifikationstasten, …

```
public class Button extends Component {
  void addActionListener(ActionListener l);
  ...
}
```

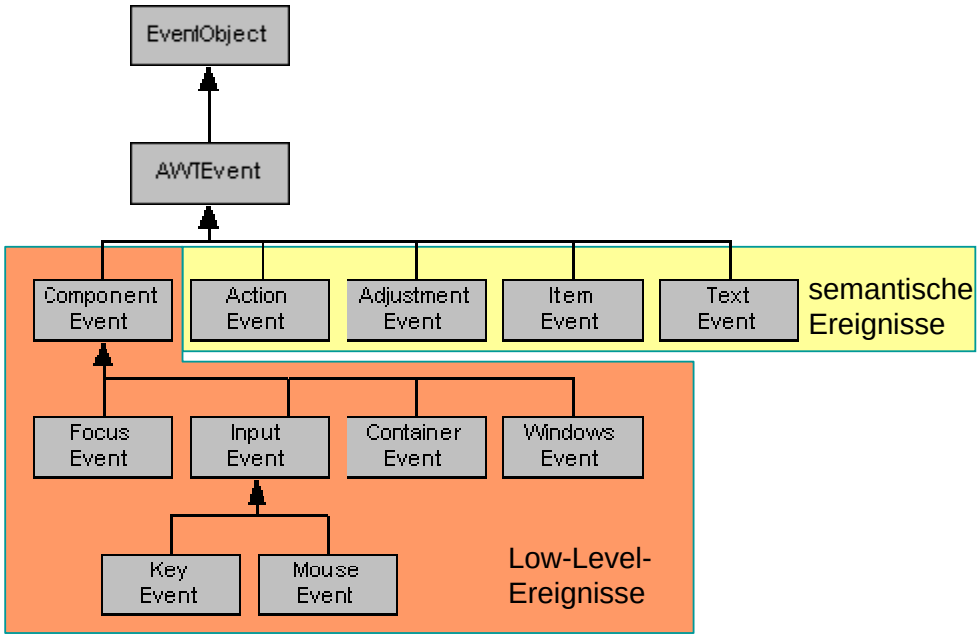
```
public interface ActionListener extends EventListener {
  void actionPerformed(ActionEvent e);
}
```

```
public class ActionEvent extends AWTEvent {
  String getActionCommand();
  int    getModifiers();
  String paramString();
}
```

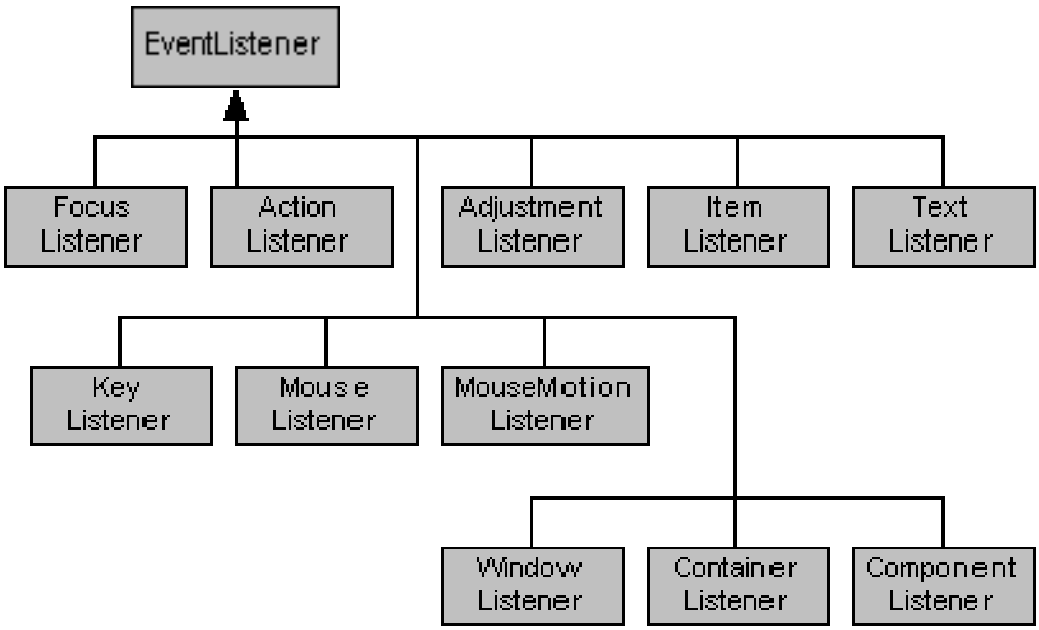
Hierarchie der Ereignisklassen

Low-Level-Ereignisse: elementaren Nachrichten

semantische Ereignisse: höherwertige Ereignisse



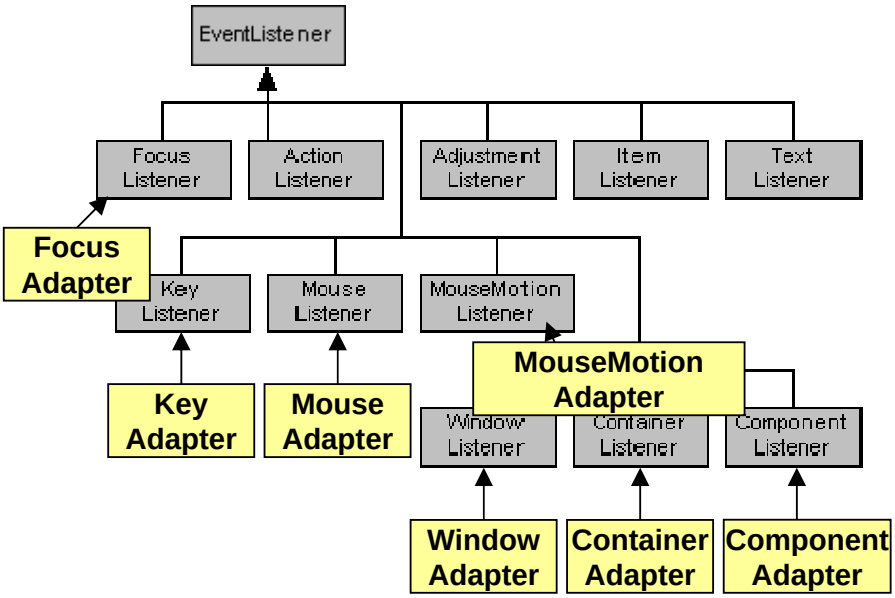
Hierarchie der EventListener-Interfaces



Adapterklassen

Implementieren Interface mit leeren Methoden

Bequeme Art, nur bestimmte Ereignisse zu behandeln



Beispiel: ActionEvents bei Button

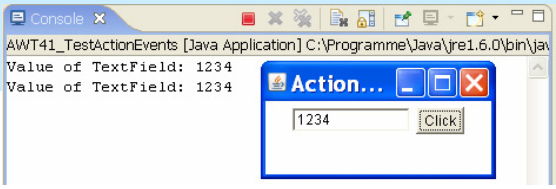
Components `Button` und `TextField`

Signalisieren `ActionEvent` bei

- Click (`Button`)
- Enter (`TextField`)

Anhängen von `ActionListeners` mit Ausgabe des Inhalts des `TextFields`

```
clickButton = new Button("Click");
textField = new TextField();

clickButton.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    System.out.println("Value of TextField: " + textField.getText());
  }
});

textField.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    System.out.println("Value of TextField: " + textField.getText());
  }
});
```



Maus und Tastatur

Ereignisse von Maus und Tastatur

- z.B. Reaktion auf Mausbewegung oder Tastatureingabe
- Tastenereignisse empfängt nur Komponente mit Fokus

```
public interface MouseListener {
  void mouseClicked(MouseEvent e);
  void mouseEntered(MouseEvent e);
  void mouseExited(MouseEvent e);
  void mousePressed(MouseEvent e);
  void mouseReleased(MouseEvent e);
}
```

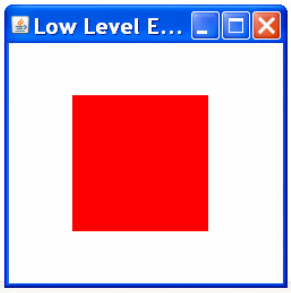
```
public interface KeyListener {
  void keyPressed(KeyEvent e);
  void keyReleased(KeyEvent e);
  void keyTyped(KeyEvent e);
}
```

```
public interface MouseMotionListener {
  void mouseDragged(MouseEvent e);
  void mouseMoved(MouseEvent e);
}
```


Beispiel: Low-Level-Events

Ausgabe der Low-Level-Events bei Panel

```
Mouse dragged at (49/65)
Mouse dragged at (48/65)
Mouse dragged at (48/66)
Mouse dragged at (47/66)
Mouse dragged at (46/66)
Mouse released at (46/66)
Mouse moved at (47/66)
Mouse moved at (50/66)
Mouse pressed at (50/66)
Mouse released at (50/66)
Mouse clicked at (50/66)
Key pressed: V - modifier: 0
Key released: V - modifier: 0
Mouse moved at (50/65)
Mouse moved at (51/64)
```




```
panel = new Panel();

panel.addMouseListener(new MouseListener() {
  public void mouseClicked(MouseEvent e) {
    System.out.println("Mouse clicked at ("+e.getX
  }
  public void mouseEntered(MouseEvent e) {
    System.out.println("Mouse entered at ("+e.getX()+"/"+e.getY()+")");
  }
  public void mousePressed(MouseEvent e) {
    System.out.println("Mouse pressed at ("+e.getX()+"/"+e.getY()+")");
  }
  ...
});

panel.addMouseMotionListener(new MouseMotionListener() {
  public void mouseDragged(MouseEvent e) {
    System.out.println("Mouse dragged at ("+e.getX()+"/"+e.getY()+")");
  }
  public void mouseMoved(MouseEvent e) {
    System.out.println("Mouse moved at ("+e.getX()+"/"+e.getY()+")");
  }
});

panel.addKeyListener(new KeyAdapter() {
  public void keyPressed(KeyEvent e) {
    System.out.println("Key pressed: "+(char)e.getKeyCode()+" - modifier:"+e.getModifiers());
  }
  ...
});
```

Focus-Ereignisse

Eigenschaft	Klasse, Interface oder Methode
Ereignisklasse	FocusEvent
Listener-Interface	FocusListener
Registrierungsmethode	addFocusListener
Mögliche Ereignisquellen	Component

Ereignismethode	Bedeutung
focusGained	Eine Komponente erhält den Focus.
focusLost	Eine Komponente verliert den Focus.

Key-Ereignisse

Eigenschaft	Klasse, Interface oder Methode
Ereignisklasse	KeyEvent
Listener-Interface	KeyListener
Registrierungsmethode	addKeyListener
Mögliche Ereignisquellen	Component

Ereignismethode	Bedeutung
keyPressed	Eine Taste wurde gedrückt.
keyReleased	Eine Taste wurde losgelassen.
keyTyped	Eine Taste wurde gedrückt und wieder losgelassen.

Component-Ereignisse

Eigenschaft	Klasse, Interface oder Methode
Ereignisklasse	ComponentEvent
Listener-Interface	ComponentListener
Registrierungsmethode	addComponentListener
Mögliche Ereignisquellen	Component

Ereignismethode	Bedeutung
componentHidden	Eine Komponente wurde unsichtbar.
componentMoved	Eine Komponente wurde verschoben.
componentResized	Die Größe einer Komponente hat sich geändert.
componentShown	Eine Komponente wurde sichtbar.

Container-Ereignisse

Eigenschaft	Klasse, Interface oder Methode
Ereignisklasse	ContainerEvent
Listener-Interface	ContainerListener
Registrierungsmethode	addContainerListener
Mögliche Ereignisquellen	Container

Ereignismethode	Bedeutung
componentAdded	Eine Komponente wurde hinzugefügt.
componentRemoved	Eine Komponente wurde entfernt.

Window-Ereignisse

Eigenschaft	Klasse, Interface oder Methode
Ereignisklasse	WindowEvent
Listener-Interface	WindowListener
Registrierungsmethode	addWindowListener
Mögliche Ereignisquellen	Dialog, Frame, Window

Ereignismethode	Bedeutung
windowActivated	Das Fenster wurde aktiviert.
windowClosed	Das Fenster wurde geschlossen.
windowClosing	Das Fenster wird geschlossen.
windowDeactivated	Das Fenster wurde deaktiviert.
windowDeiconified	Das Fenster wurde wiederhergestellt.
windowIconified	Das Fenster wurde auf Symbolgröße verkleinert.
windowOpened	Das Fenster wurde geöffnet.

Action-Ereignisse

Eigenschaft	Klasse, Interface oder Methode
Ereignisklasse	ActionEvent
Listener-Interface	ActionListener
Registrierungsmethode	addActionListener
Mögliche Ereignisquellen	Button, List, MenuItem, TextField

Ereignismethode	Bedeutung
actionPerformed	Eine Aktion wurde ausgelöst.

Adjustment-Ereignisse

Eigenschaft	Klasse, Interface oder Methode
Ereignisklasse	AdjustmentEvent
Listener-Interface	AdjustmentListener
Registrierungsmethode	addAdjustmentListener
Mögliche Ereignisquellen	Scrollbar

Ereignismethode	Bedeutung
adjustmentValueChanged	Der Wert wurde verändert.

Item-Ereignisse

Eigenschaft	Klasse, Interface oder Methode
Ereignisklasse	ItemEvent
Listener-Interface	ItemListener
Registrierungsmethode	addItemListener
Mögliche Ereignisquellen	Checkbox, Choice, List, CheckboxMenuItem

Ereignismethode	Bedeutung
itemStateChanged	Der Zustand hat sich verändert.

Text-Ereignisse

Eigenschaft	Klasse, Interface oder Methode
Ereignisklasse	TextEvent
Listener-Interface	TextListener
Registrierungsmethode	addTextListener
Mögliche Ereignisquellen	TextField, TextArea

Ereignismethode	Bedeutung
textValueChanged	Der Text wurde verändert.

AWT

Einführung
Components and Containers
Layoutmanagement
Graphikausgabe und AWT-Thread
Ereignisse
MVC Architektur
Komponenten
Zusammenfassung

Model – View – Controller (MVC)

Architekturmuster für die Gestaltung von interaktiven Oberflächen

3 Komponenten

- *Model*: Datenmodell
- *View*: Ansicht
- *Controller*: Interaktion

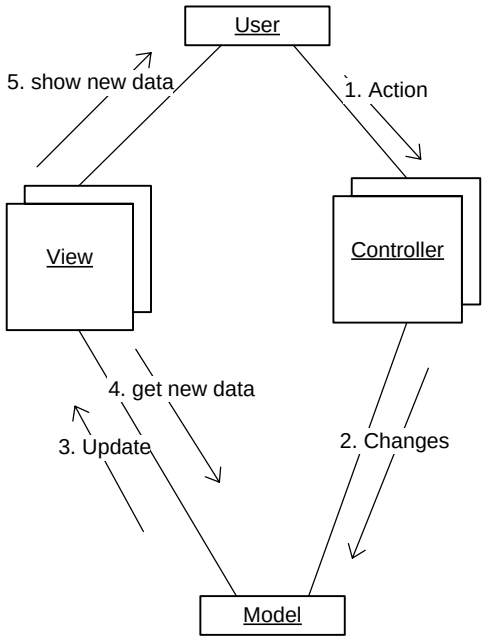
Bei Java AWT und Swing sind View und Controller integriert

Motivation:

- Datenmodell unabhängig von Views und Controller
- Mehrere Views und Controller für ein Datenmodell

MVC Ablauf

- Aktion durch Benutzer
- Controller reagiert und Daten werden geändert
- Model verständigt alle Views, dass sich Daten geändert haben
- View holen sich die neuen Daten vom Model
- Views stellen die neuen Daten dar



Ereigniskonzept in Java

Kommunikation mit anderen Beans

- Zustandsänderung wird allen Interessierten mitgeteilt
- Lose gekoppelte Kommunikation

Quelle

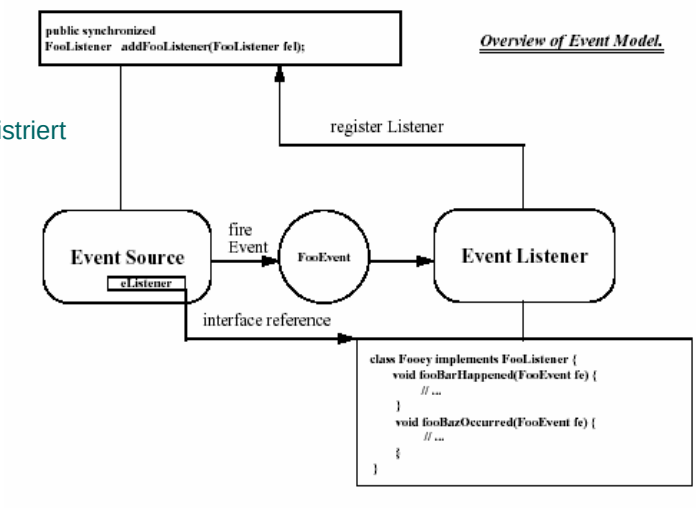
- Auslöser eines Ereignisses
- Benachrichtigt Interessierte

Interessierter (Listener)

- Reagiert auf Ereignis
- Wird bei Quelle registriert und deregistriert

Ereignisobjekt (EventObject)

- Informationen über Ereignis



Design Pattern für Ereignisse

Listener

- Interface welche Ereignismethoden implementiert
- *muss* `java.util.EventListener` erweitern

```
public interface TimerListener extends java.util.EventListener {
  public void timerAction(TimerEvent e);
}
```

Quelle

- implementiert Methoden für das Anfügen und Entfernen von Listener

```
public void addListenerType(ListenerType listener)
public void removeListenerType(ListenerType listener);
```

```
public class TimerBean {
  public void addTimerListener(TimerListener l) { … }
  public void removeTimerListener(TimerListener l) { … }

  private void fireTimerEvent(TimerEvent e) { … }
  …
}
```

Design Pattern für Ereignisse (Fortsetzung)

Ereignisobjekt

- erweitert `java.util.EventObject`
- kommuniziert Informationen über Ereignis

```
public class TimerEvent extends java.util.EventObject {
  private long time;

  public TimerEvent(Object source) {
    super(source);
    time = System.currentTimeMillis();
  }

  public long getTime() {
    return time;
  }
}
```


MVC in Java

Java MVC arbeitet stark mit Ereignissen

Controller:

- sendet AWT Events bei Benutzeraktionen
- Eventhandler implementieren Datenänderungen

Model:

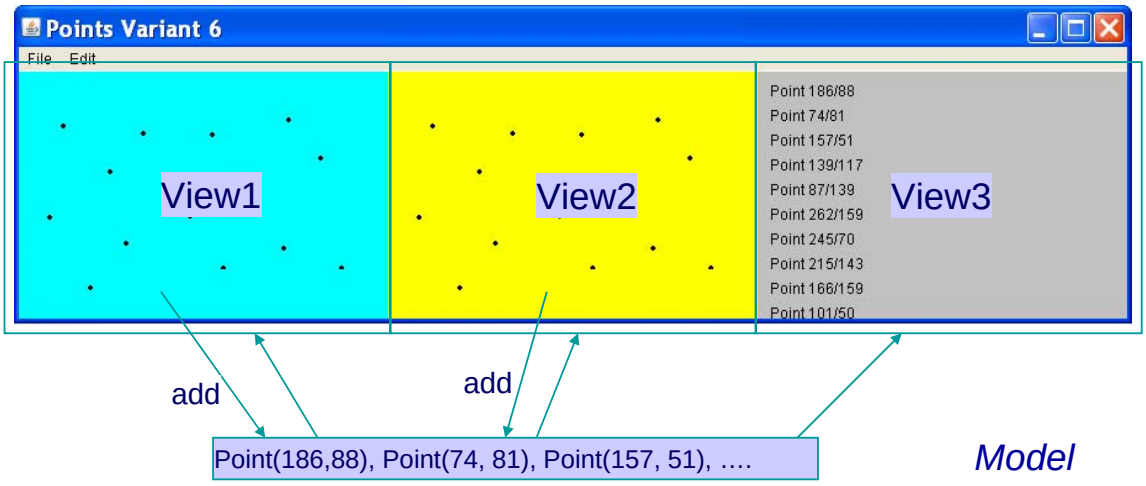
- meldet Datenänderungen mit Ereignissen
- Model stellt Quelle der Ereignisse dar (hat Methoden zum An-/Abmelden der Listener)
- meist eigenes Listener-Inteface und EventObject

Views:

- Implementieren die Listener-Interfaces
- melden sich beim Model als Listener an
- reagieren auf die gemeldeten Datenänderungen mit neuer Darstellung

Beispiel Points: Architektur

Mehrere Views und Editors für ein Modell



Beispiel Points: PointListener und PointEvent

`PointListener`: Listener-Interface für Änderungen bei Points

```
public interface PointsListener {

    public void pointChanged(PointEvent event);

}
```

`PointEvent`: Event-Object mit Informationen über Ereignis

```
import java.awt.Point;
import java.util.EventObject;

public class PointEvent extends EventObject {
  Point point; String eventName;

  public PointEvent(Object source, String eventName, Point point) {
    super(source); this.point = point; this.eventName = event;
  }
  public Point getPoint() { return point; }

  public String getEventName() { return eventName; }
}
```

Beispiel Points: PointModel

Datenobjekt PointModel:

```
public class PointModel {
  private List<Point> points = new ArrayList<Point>();
  private List<PointsListener> listeners = new ArrayList<PointsListener>();
```

Methoden für Datenänderungen

```
  public void clear() {
    points.clear();
    firePointEvent("removed all", null);
  }
  public void add(Point p) {
    points.add(p);
    firePointEvent("added“, p);
  }
  ...
```

Datenzugriff

```
  public Point[] getPoints() {
    return points.toArray(new Point[0]);
  }
  ...
```


Beispiel Points: PointModel (2)

An-/Abmelden der Listener und Feuern der Ereignisse

```
  public void addPointsListener(PointsListener l) {
    listeners.add(l);
  }

  public void removePointsListener(PointsListener l) {
    listeners.remove(l);
  }

  private void firePointEvent(Point p) {
    PointEvent evt = new PointEvent(this, p);
    for (PointsListener pointL: listeners) {
      prointL.pointChanged(evt);
    }
  }
}
```

Beispiel Points: PointCanvas

PointCanvas implementiert `PointsListener` und reagiert auf Mouse-Clicks

```
public class PointCanvas extends Canvas implements PointsListener {
  PointModel pointsModel;

  public PointCanvas(PointModel points) {
    this.pointsModel = points;
    pointsModel.addPointsListener(this);
    this.addMouseListener(new MouseClickHandler());
  }
  public void paint(Graphics g) {
    super.paint(g);
    for (Point p: pointsModel.getPoints()) {
      g.fillOval(p.x-2, p.y-2, 4, 4);
    }
  }
  public void pointChanged(PointEvent event) {
     repaint();
  }

  class MouseClickHandler extends MouseAdapter {
    public void mouseClicked(MouseEvent e) {
      pointsModel.add(new Point(e.getX(), e.getY()));
    }
  }
}
```

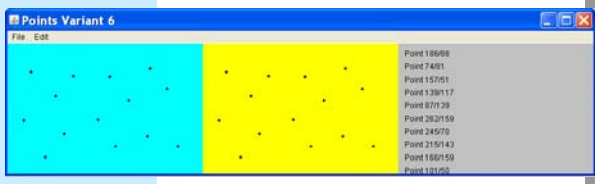

Beispiel Points: PointTextPanel

PointTextPanel gibt Points textuell aus; implementiert `PointsListener`

```
public class PointTextPanel extends Panel implements PointsListener {

  PointModel pointsModel;

  public PointTextPanel(PointModel points) {
    this.pointsModel = points;
    pointsModel.addPointsListener(this);
  }

  public void paint(Graphics g) {
    super.paint(g);
    int line = 20;
    for (Point p: pointsModel.getPoints()) {
      g.drawString("Point "+p.x+"/"+p.y, 10, line);
      line = line + 20;
    }
  }

  public void pointChanged(PointEvent event) {
     repaint();
  }
}
```

Beispiel Points: Frame mit 3 Views

```
public class PointFrame extends Frame {
  private PointModel pointModel;
  private PointCanvas panel1, panel2;
  private PointTextPanel textPanel;

  public PointFrame(PointModel model) {
    ...
    this.pointModel = model;
    MenuItem clear = new MenuItem("Clear");
    editMenu.add(clear);
    clear.addActionListener((new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        pointModel.clear();
      }
    }));
    ...
    canvas1 = new PointCanvas(this.pointModel);
    canvas1.setBackground(Color.CYAN);
    add(canvas1);

    canvas2 = new PointCanvas(this.pointModel);
    canvas2.setBackground(Color.YELLOW);
    add(canvas2);

    textPanel = new PointTextPanel(this.pointModel);
    textPanel.setBackground(Color.LIGHT_GRAY);
    add(textPanel);
}
```



Model

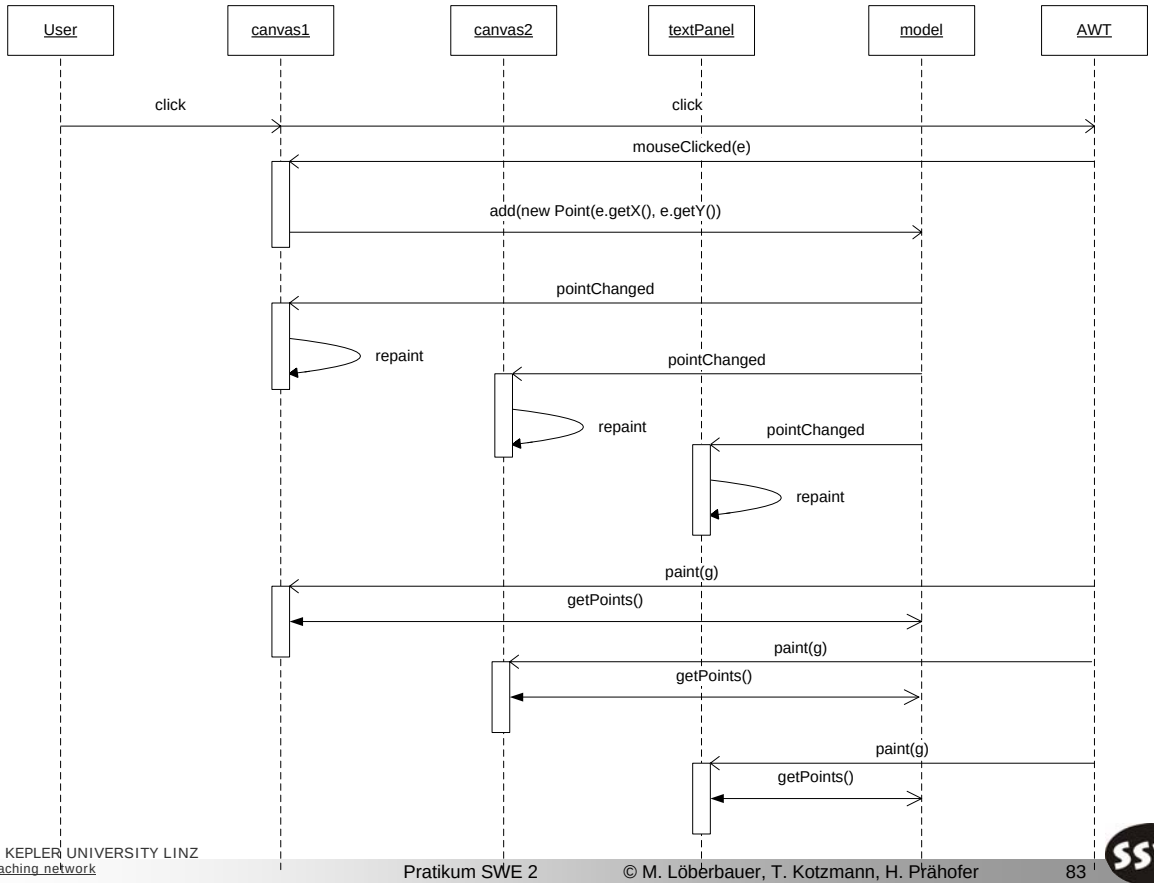
MenuItem "Clear"

PointCanvas canvas1

PointCanvas canvas2

PointTextPanel

Beispiel Points: Ablauf



AWT

Einführung

Components and Containers

Layoutmanagement

Graphikausgabe und AWT-Thread

Ereignisse

MVC Architektur

Komponenten

Zusammenfassung

Label

- Stellt einzeiligen Text dar
- Passiv, d.h. feuert keine Ereignisse

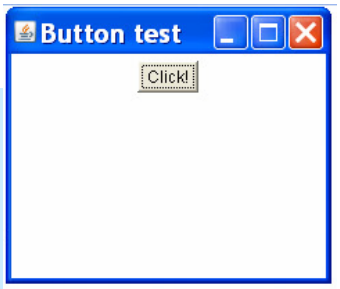
```
public class Label extends Component
  public Label(String text)
  public Label(String text, int alignment)
  public void setText(String text)
  public String getText()
  public void setAlignment(int alignment)
  public int getAlignment()
```

```
public static void main(String[] args) {

  Frame frame = new Frame("Button test");
  frame.setLayout(new FlowLayout());
  ...

  Label name = new Label("Name");
  Label descr = new Label("Description");

  frame.add(name);
  frame.add(descr);
  frame.setSize(200, 200);
  frame.setVisible(true);
}
```

Button

- feuert ActionEvent
- hat Label

```
public class Button extends Component
  public Button(String label)
  public String getLabel()
  public void setLabel(String label)
  public void addActionListener(ActionListener l)
  public void removeActionListener(ActionListener l)
}
```



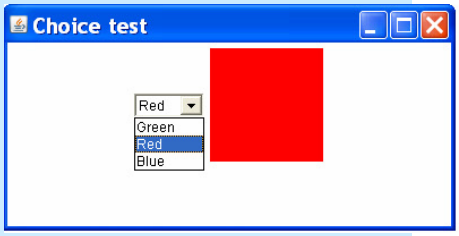

```
public static void main(String[] args) {
  Frame frame = new Frame("Button test");

  Button click = new Button("Click!");
  click.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      System.out.println("Click has been pressed!");
    }
  });

  frame.add(click);
  ...
}
```


Choice

- Selektion aus Liste von Strings
- feuert ItemEvents

```
public class Choice extends Component
  public Choice()
  public void addItem(String item)
  public void insert(String item, int index)
  public String getItem(int index)
  public int getItemCount()
  public int getSelectedIndex()
  public String getSelectedItem()
  public void addItemListener(ItemListener l)
  public void removeItemListener(ItemListener l)
```

```
Choice colorChooser = new Choice();
colorChooser.add("Green");
colorChooser.add("Red");
colorChooser.add("Blue");

colorChooser.addItemListener(new ItemListener() {
  public void itemStateChanged(ItemEvent e) {
    if (e.getItem().equals("Green")) {
      panel.setBackground(Color.GREEN);
    } else if (e.getItem().equals("Red")) {
      panel.setBackground(Color.RED);
    } else if (e.getItem().equals("Blue")) {
      panel.setBackground(Color.BLUE);
    }
    panel.repaint();
  }
});
frame.add(colorChooser);
…
```



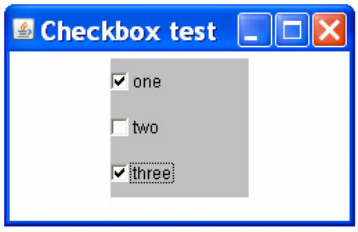
Checkbox

- Kann selektiert werden
- Hat Label und Zustand
- Feuert ItemEvents

```
public class Checkbox
  public Checkbox()
  public Checkbox(String label)
  public Checkbox(String label, boolean state)
  public Checkbox(String label, boolean state,  CheckboxGroup g)
  public String getLabel()
  public void setLabel(String label)
  public boolean getState()
  public void setState(boolean state)
  public void addItemListener(ItemListener l)
  public void removeItemListener(ItemListener l)
```

```
Checkbox cb1 = new Checkbox("one", true);
Checkbox cb2 = new Checkbox("two");
Checkbox cb3 = new Checkbox("three");
cb1.addItemListener(new ItemListener() {
  public void itemStateChanged(ItemEvent e) {
    if (cb1.getState()) {
      System.out.println("One selected");
    } else {
      System.out.println("One deselected");
    }
  }
});
cb2.addItemListener(new ItemListener() {
  public void itemStateChanged(ItemEvent e) {
    if (cb2.getState()) {
      System.out.println("Two selected");
    } else {
      System.out.println("Two deselected");
  ...
```

```
One deselected
One selected
Two selected
Three selected
Two deselected
Three deselected
Three selected
```



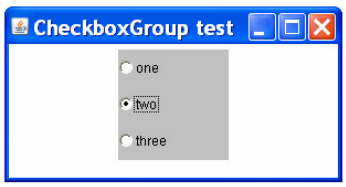
CheckboxGroup

- Checkboxes können gruppiert werden
- Nur eine in der Gruppe selektiert

```
public class CheckboxGroup
  public CheckboxGroup()
  public Checkbox getSelectedCheckbox()
  public void setSelectedCheckbox(Checkbox box)
```

```
CheckboxGroup group = new CheckboxGroup();
Checkbox cb1 = new Checkbox("one", true, group);
Checkbox cb2 = new Checkbox("two", false, group);
Checkbox cb3 = new Checkbox("three", false, group);
cb1.addItemListener(new ItemListener() {
  public void itemStateChanged(ItemEvent e) {
    if (cb1.getState()) {
      System.out.println("One selected");
    } else {
      System.out.println("One deselected");
    }
  }
});
cb2.addItemListener(new ItemListener() {
  public void itemStateChanged(ItemEvent e) {
    if (cb2.getState()) {
      System.out.println("Two selected");
    } else {
      System.out.println("Two deselected");
  ...
```

```
Two selected
Three selected
Two selected
One selected
Two selected
Three selected
Two selected
```



TextField

- für Eingabe eines einzeiligen Textes
- hat Curser
- hat Selektion
- kann EchoChar haben (z.B. für Passworteingabe)
- feuert ActionEvent bei Enter
- feuert TextEvents bei Änderungen

```
public class TextField extends TextComponent
  public TextField()
  public TextField(int columns)
  public TextField(String text)
  public TextField(String text, int columns)

  public void setText(String t)
  public String getText()

  public int getCaretPosition()
  public void setCaretPosition(int position)

  public void setEchoChar(char c)

  public String getSelectedText()
  public int getSelectionStart()
  public void setSelectionStart(int selectionStart)
  public int getSelectionEnd()
  public void setSelectionEnd(int selectionEnd)

  public void addActionListener(ActionListener l)
  public void removeActionListener(ActionListener l)

  public void addTextListener(TextListener l)
  public void removeTextListener(TextListener l)
```


Beispiele TextField (1)

```
tf1 = new TextField("Hello");
tf2 = new TextField();
tf3 = new TextField();
tf4 = new TextField(30);

panel.add(new Label("Normales TextField: "));
panel.add(tf1);
tf1.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    System.out.println("TextField 1 : " +
       tf1.getText() + " eingegeben");
  }
});

panel.add(new Label("TextField mit EchoC:"));
tf2.setEchoChar('*');
tf2.addTextListener(new TextListener() {
  public void textValueChanged(TextEvent e) {
    System.out.println("TextField 2 : " + tf2.getText());
  }
});
panel.add(tf2);

panel.add(new Label("Einfügen am Anfang: "));
tf3.addTextListener(new TextListener() {
  public void textValueChanged(TextEvent e) {
    tf3.setCaretPosition(0);
  }
});
panel.add(tf3);
```

Ausgabe bei Enter

EchoChar '*'

Ausgabe jeder Änderung

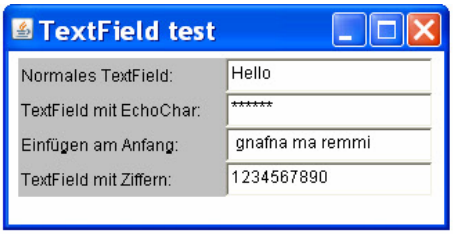
Setzen des Caret auf Position 0

Beispiele TextField (2)

```
panel.add(new Label("TextField mit Ziffern: "));
panel.add(tf4);
tf4.addTextListener(new TextListener() {
  public void textValueChanged(TextEvent e) {
    StringBuffer b = new StringBuffer(tf4.getText());
    int pos = tf4.getCaretPosition() - 1;
    if (pos >= 0 && pos < b.length()) {
      char inserted = b.charAt(pos);
      if (!Character.isDigit(inserted)) {
        b.deleteCharAt(pos);
        tf4.setText(b.toString());
        tf4.setCaretPosition(pos);
      }
    }
  }
});
...
```

Löschen aller Nicht-Ziffern

```
TextField 2 : g
TextField 2 : ge
TextField 2 : geh
TextField 2 : gehe
TextField 2 : gehei
TextField 2 : geheim
```



TextArea

- für mehrzeiliges Editieren von Text
- erlaubt Einfügen, Anfügen, Ersetzen von Text
- hat Curser
- hat Selektion
- feuert ActionEvent bei Enter
- feuert TextEvents bei Änderungen

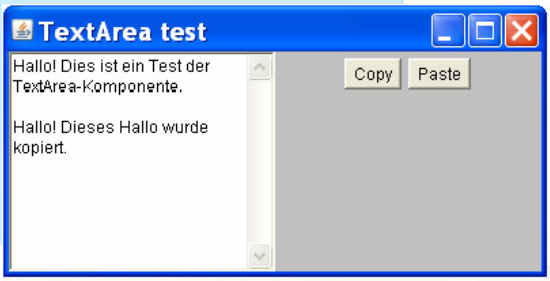
```
public TextArea extends TextComponent
  public TextArea()
  public TextArea(int rows, int columns)
  public TextArea(String text)
  public TextArea(String text, int rows, int columns)
  public TextArea(String text, int rows, int columns,
                  int scrollbars)

  public void insert(String str, int pos)
  public void append(String str)
  public void replaceRange(String t, int start, int end)
  public String getText()
  public void setText(String text)

  public int getCaretPosition()
  public void setCaretPosition(int position)

  public String getSelectedText()
  public int getSelectionStart()
  public void setSelectionStart(int selectionStart)
  public int getSelectionEnd()
  public void setSelectionEnd(int selectionEnd)

  public void addTextListener(TextListener l)
  public void removeTextListener(TextListener l)
```

Beispiel TextArea

```
static TextArea textArea;
static Button copy, paste;
static String copiedText = "";

public static void main(String[] args) {
  final Frame frame = new Frame(„TextArea test");

  textArea = new TextArea("Hallo", 20, 10, TextArea.SCROLLBARS_VERTICAL_ONLY);
  frame.add(textArea);

  copy = new Button("Copy");
  copy.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      copiedText = textArea.getSelectedText();
    }
  });

  paste = new Button("Paste");
  paste.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      textArea.replaceRange(copiedText,
                            textArea.getSelectionStart(),
                            textArea.getSelectionEnd());
    }
  });
  Panel panel = new Panel();
  panel.add(copy);
  panel.add(paste);
  ...
}
```

Zwischenpeichern der aktuellen Selektion

Einfügen des kopierten Textes bei aktueller Selektion



List

- Liste von String-Items
- Anfügen, Löschen, Ersetzen von String-Items
- Selektion von String-Items
- feuert ItemEvents bei Änderung der Selektion
- feuert ActionEvent bei Enter oder Doppelklick

```
public List extends Component implements ItemSelectable
  public List()
  public List(int size)
  public List(int size, boolean multiselect)

  public void add(String item)
  public void add(String item, int index)
  public void delItem(int index)
  public void remove(int index)
  public void replaceItem(String newValue, int index)

  public int[] getSelectedIndexes()
  public String[] getSelectedItems()
  public void select(int index)
  public void deselect(int index)

  public void addItemListener(ItemListener l)
  public void removeItemListener(ItemListener l)
  public void addActionListener(ActionListener l)
  public void removeActionListener(ActionListener l)
```

Beispiel List (1)

```
static TextField itemField;
static List itemList;
static Button insert, remove;
static Choice mode;

public static void main(String[] args) {
  ...
  itemList = new List();
  itemList.addItemListener(new ItemListener() {
    public void itemStateChanged(ItemEvent e) {
      System.out.print("Selektion: ");
        for (String item: itemList.getSelectedItems())
          System.out.print(item + " ");

      System.out.println();
    }
  });
  itemList.addActionListener(new ActionListener() {
     public void actionPerformed(ActionEvent e) {
       System.out.println("Ausgewählt: " + itemList.getSelectedItem());
     }
   });
   remove = new Button("Remove");
   remove.addActionListener(new ActionListener() {
     public void actionPerformed(ActionEvent e) {
       String[] selectedItems = itemList.getSelectedItems();
       for (String item: selectedItems)
         itemList.remove(item);
     }
  });
```

Änderungen der Selektion ausgeben

Ausgewähltes Item (Doppelklick) ausgeben

Selektierte Items löschen

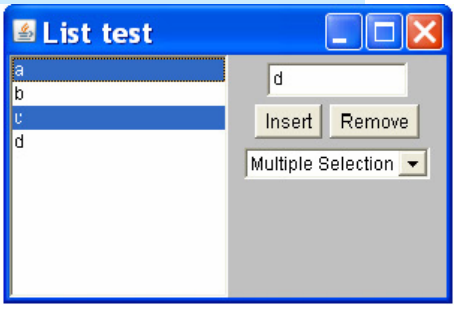
Beispiel List (2)

```
  itemField = new TextField(10);
  insert = new Button("Insert");
  insert.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      String item = itemField.getText();
      if (! item.equals(""))
         itemList.add(item);
    }
  });

  mode = new Choice();
  mode.add("Single Selection");
  mode.add("Multiple Selection");
  mode.addItemListener(new ItemListener() {
    public void itemStateChanged(ItemEvent e) {
      if (e.getItem().equals("Single Selection")) {
        itemList.setMultipleMode(false);
      } else if (e.getItem().equals("Multiple Selection")) {
        itemList.setMultipleMode(true);
      }
    }
  });
  ...
}
```

String aus TextField lesen und anfügen

Selektionsmodus ändern

```
Selektion: a c
Selektion: a c d
Selektion: a c
Selektion: c
Selektion:
Ausgewählt: c
Selektion: c
Selektion: a c
```



Menus

MenuBar

- für Menuleiste
- direkt bei Frame angefügt

```
public class MenuBar extends MenuComponent
          implements MenuContainer
  public MenuBar()
  public void add(Menu m)
  public void remove(MenuComponent m)
  public void remove(int index)
```

Menu

- hat Namen
- enthält SubMenus, MenuItems, Seperators

```
public class Menu extends MenuItem
          implements MenuContainer
  public Menu(String label)
  public void add(MenuItem mi)
  public void add(String label)
  public void remove(int index)
  public void remove(MenuComponent item)
  public void addSeparator()
  public void insertSeparator(int index)
```

MenuItem

- hat Namen
- stellt Menueintrag dar
- feuert ActionEvent bei Selektion
- kann aktiviert oder deaktiviert sein (*enabled*)

```
public class MenuItem extends MenuComponent
  public MenuItem(String label)
  public String getLabel()
  public void setLabel(String label)
  public void setEnabled(boolean b)
  public boolean isEnabled()
  public void addActionListener(ActionListener l)
  public void removeActionListener(ActionListener l)
```

CheckboxMenuItem

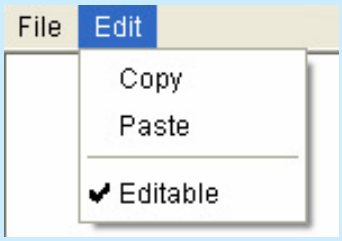
- MenuItem mit Funktion wie Checkbox

```
public class CheckboxMenuItem extends MenuItem
  public boolean getState()
  public void setState(boolean state)
  public void addItemListener(ItemListener l)
  public void removeItemListener(ItemListener l)
```


Beispiel Menus (1)

```
static String copiedText = "";

public static void main(String[] args) {
  final Frame frame = new Frame("Menu test");

  final TextArea textArea = new TextArea("Hallo", 20, 10);
  frame.add(textArea);

  MenuBar menuBar = new MenuBar();
  frame.setMenuBar(menuBar);
  Menu fileMenu = new Menu("File");
  menuBar.add(fileMenu);
  final MenuItem exit = new MenuItem("Exit");
  fileMenu.add(exit);
  Menu editMenu = new Menu("Edit");
  menuBar.add(editMenu);
  final MenuItem copy = new MenuItem("Copy");
  editMenu.add(copy);
  final MenuItem paste = new MenuItem("Paste");
  editMenu.add(paste);
  editMenu.addSeparator();
  final CheckboxMenuItem editable = new CheckboxMenuItem("Editable", true);
  editMenu.add(editable);
```

Aufbau der Menüs



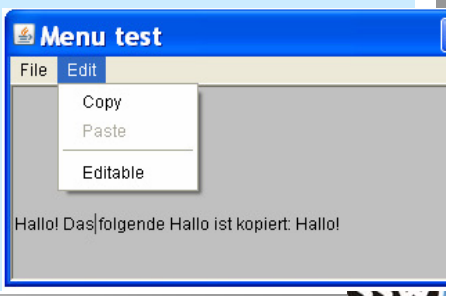
Beispiel Menus (2)

```
copy.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
   copiedText = textArea.getSelectedText();
  }
});
paste.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    if (textArea.isEditable()) {
      textArea.replaceRange(copiedText,
      textArea.getSelectionStart(),
      textArea.getSelectionEnd());
    }
  }
});
editable.addItemListener(new ItemListener() {
  public void itemStateChanged(ItemEvent e) {
    if (editable.getState()) {
      textArea.setEditable(true);
      paste.setEnabled(true);
      textArea.setBackground(Color.WHITE);
    } else {
      textArea.setEditable(false);
      paste.setEnabled(false);
      textArea.setBackground(Color.LIGHT_GRAY);
    }
  }
});
...
```

Kopieren der Selektion bei Copy-Menueintrag

Einfügen der Selektion bei Paste-Menueintrag

Enable/Disable Editieren durch CheckboxMenuItem



Menu Shortcuts

MenuShortcut

- Konzept für Shortcuts
- Definiert key

```
public class MenuShortcut
  public MenuShortcut(int key)
  public MenuShortcut(int key, boolean useShiftModifier)
  public int getKey()
  public boolean usesShiftModifier() {
```

KeyEvent

- Enthält Konstante mit allen Tastencodes
- „virtuelle Keytcodes“

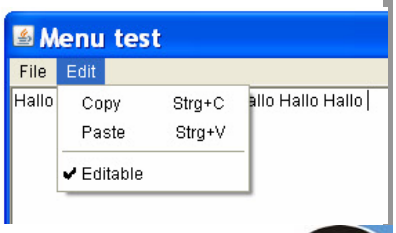
```
public class KeyEvent extends InputEvent
  public static final int VK_ENTER          = '\n';
  public static final int VK_BACK_SPACE     = '\b';
  public static final int VK_A              = 0x41;
  public static final int VK_B              = 0x42;
  …
```

MenuItem

- setShortcut-Methode
- Constructor mit Shortcut

```
public class MenuItem extends MenuComponent
  public MenuItem(String label, MenuShortcut s)
  public void setShortcut(MenuShortcut s)
```

```
MenuShortcut copyShortcut = new MenuShortcut(KeyEvent.VK_C);
copy.setShortcut(copyShortcut);
MenuShortcut pasteShortcut = new MenuShortcut(KeyEvent.VK_V);
paste.setShortcut(pasteShortcut);
```



ScrollPane

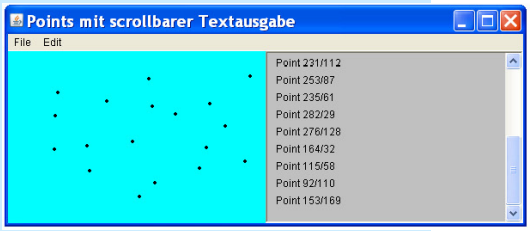
- Srolling für eine Komponente
- add fügt einzige Komponente ein
- preferredSize der Komponente definiert Scrollbereich
- Setzen von
  - scrollPosition
  - ScrollPolicy
- Beeinflussen des Scrollverhaltens mit Adjustable

```
public ScrollPane()
  public static final int SCROLLBARS_AS_NEEDED
  public static final int SCROLLBARS_ALWAYS
  public static final int SCROLLBARS_NEVER

  public ScrollPane(int scrollbarDisplayPolicy)
  public void add(Component c)

  public Adjustable getHAdjustable()
  public Adjustable getVAdjustable()

  public void setScrollPosition(int x, int y)
  public Point getScrollPosition()
```

```
public interface Adjustable
  void setUnitIncrement(int u)
  int getUnitIncrement()
  void setBlockIncrement(int b)
  int getBlockIncrement()
  void setVisibleAmount(int v)
  int getVisibleAmount()
  void setMinimum(int min)
  void setMaximum(int max)
  int getMinimum()
  int getMaximum()
  void addAdjustmentListener(AdjustmentListener l)
  void removeAdjustmentListener(AdjustmentListener l)
  ...
```


Beispiel ScrollPane

```
public class PointScrollPanel extends ScrollPane implements PointsListener {
  PointModel pointsModel;
  PointTextPanel textPanel;

  public PointScrollPanel(PointModel points) {
    this.pointsModel = points;
    pointsModel.addPointsListener(this);
    textPanel = new PointTextPanel();
    this.add(textPanel);
  }

  private class PointTextPanel extends Canvas {
    public void paint(Graphics g) {
      super.paint(g);
      int line = 20;
      for (Point p: pointsModel.getPoints()) {
        g.drawString("Point "+p.x+"/"+p.y, 10, line);
        line = line + 20;
      }
    }
    public Dimension getPreferredSize() {
        return new Dimension(100, (pointsModel.getPointsCount()+1)*20);
    }
  }

  public void pointChanged(PointEvent event) {
    textPanel.repaint();
    textPanel.invalidate();
    validate();
    this.setScrollPosition(0, textPanel.getHeight());
  }
}
```



Innere Klasse für Textausgabe

Anpassung der Höhe aufgrund der Anzahl der Punkte

Bei Änderungen TextPanel neu zeichnen, neu layouten und zum Ende scrollen

Window

- Fenster ohne Titel und Menubar
- Wird normalerweise mit Parent-Frame oder Window erzeugt
- feuert WindowEvents
- Basisklasse von Frame
- Verwendung ähnlich Frame

```
public class Window extends Container
  public Window(Frame owner)
  public Window(Window owner)
  public void setIconImage(Image image)
  public Window getOwner()
  public Window[] getOwnedWindows()
  public void addWindowFocusListener(
         WindowFocusListener l)
  public void removeWindowFocusListener(
         WindowFocusListener l)
```


Dialog

Modale[1] und nicht modale Dialoge

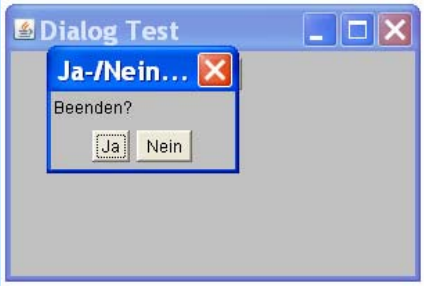
Verwendung wie Window

```
public class Dialog extends Window
  public Dialog(Window owner, String title)
  public Dialog(Frame owner, String title,
                boolean modal)
  ...
  public void setModal(boolean modal)
  public boolean isModal()
  public void toBack()

  public void setResizable(boolean resizable)
  public boolean isResizable()
  ...
```

[1] modal heisst, Dialog blockiert andere Fenster bis geschlossen!

Beispiel Dialog: Ja/Nein-Dialog

```
class YesNoDialog extends Dialog implements ActionListener {
  boolean result;

  public YesNoDialog(Frame owner, String msg) {
    super(owner, "Ja-/Nein-Auswahl", true);
    setLayout(new BorderLayout());
    setResizable(false);
    setLocation(owner.getX() + 30, owner.getY() + 30);

    add(new Label(msg), BorderLayout.CENTER);
    Panel panel = new Panel();
    panel.setLayout(new FlowLayout(FlowLayout.CENTER));
    Button button = new Button("Ja");
    button.addActionListener(this);
    panel.add(button);
    button = new Button("Nein");
    button.addActionListener(this);
    panel.add(button);
    add(panel, BorderLayout.SOUTH);
    pack();
  }

  public void actionPerformed(ActionEvent event) {
    result = event.getActionCommand().equals("Ja");
    setVisible(false);
    dispose();
  }

  public boolean getResult() {
    return result;
  }
}
```




```
YesNoDialog dlg = new YesNoDialog(this, "Beenden?");
dlg.setVisible(true);
if (dlg.getResult()) {
   System.exit(0);
}
```


AWT

- Einführung
- Components and Containers
- Layoutmanagement
- Graphikausgabe und AWT-Thread
- Ereignisse
- MVC Architektur
- Komponenten
- Zusammenfassung

Zusammenfassung

AWT stellt eine einfache Bibliothek für GUI dar

„Heavyweight Components“ auf native API des jeweiligen Betriebssystems abgebildet

Definiert

- Komponenten und Komponentenhierarchie
- Automatisches Layoutmanagement
- AWT-Thread und graphische Ausgabe (Painting)
- Ereignismodell mit Listener und EventObjects

AWT bildet die Basis für Swing