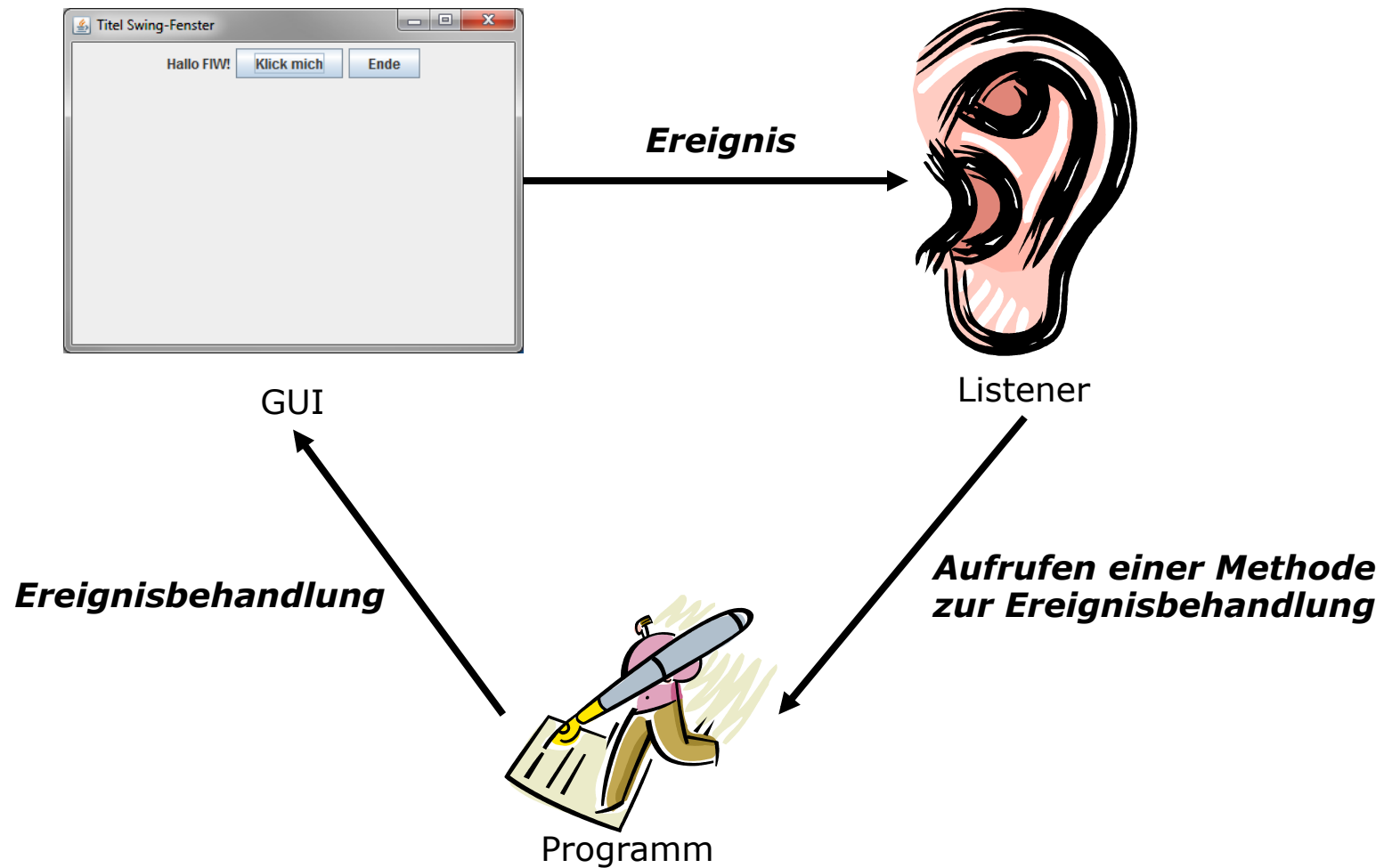


SWING

Ereignisbehandlung

Macht der Nutzer etwas?



Ereignisbehandlung

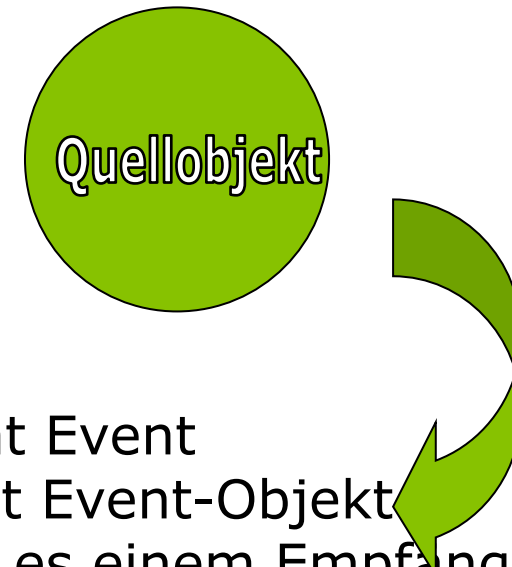
hält Listener bereit



Dispatcher:

- verwaltet Event-Queue
- stellt Nachricht zu durch Aufruf der Listener.Methode

löst Event aus

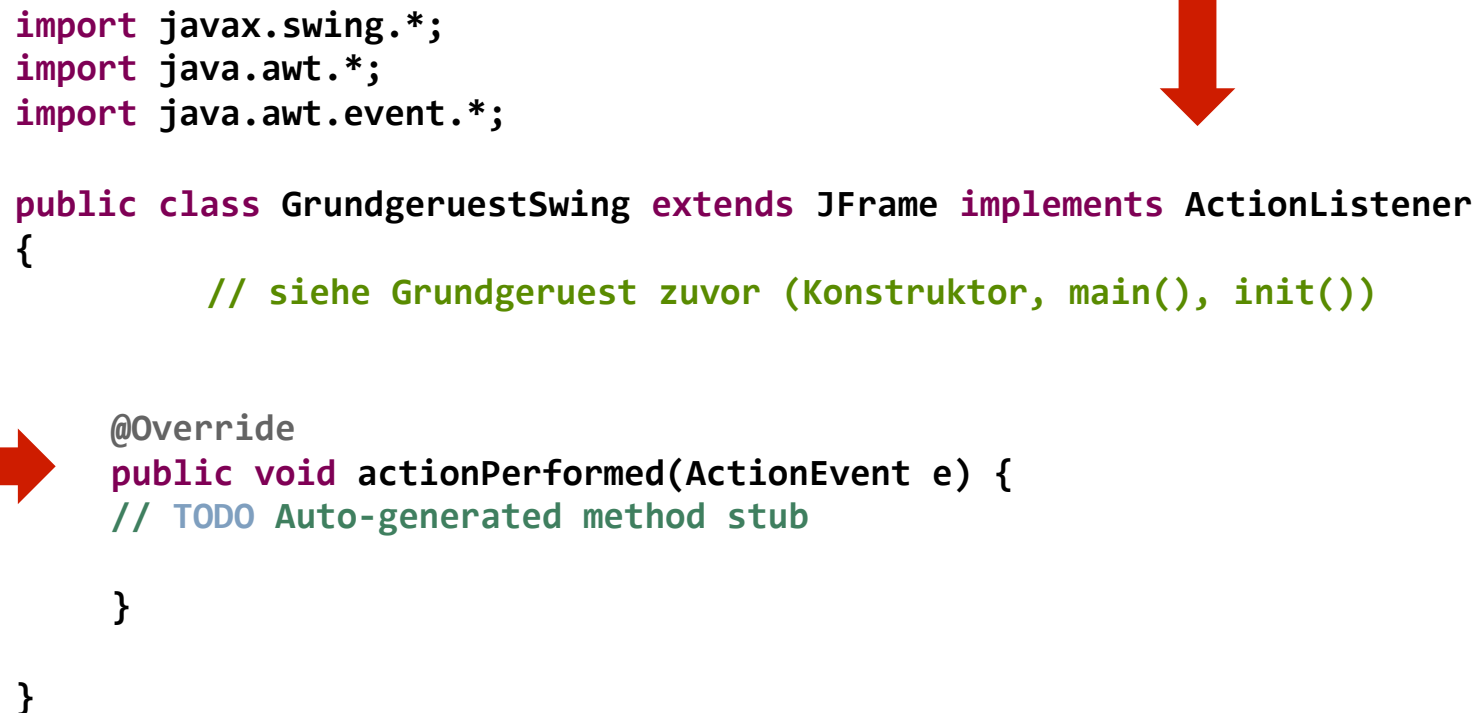


System:

- erkennt Event
- erzeugt Event-Objekt
- ordnet es einem Empfänger zu
- übergibt Event dem Dispatcher

Listener sind Interfaces

Beispiel ActionListener



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

public class GrundgeruestSwing extends JFrame implements ActionListener
{
    // siehe Grundgeruest zuvor (Konstruktor, main(), init())

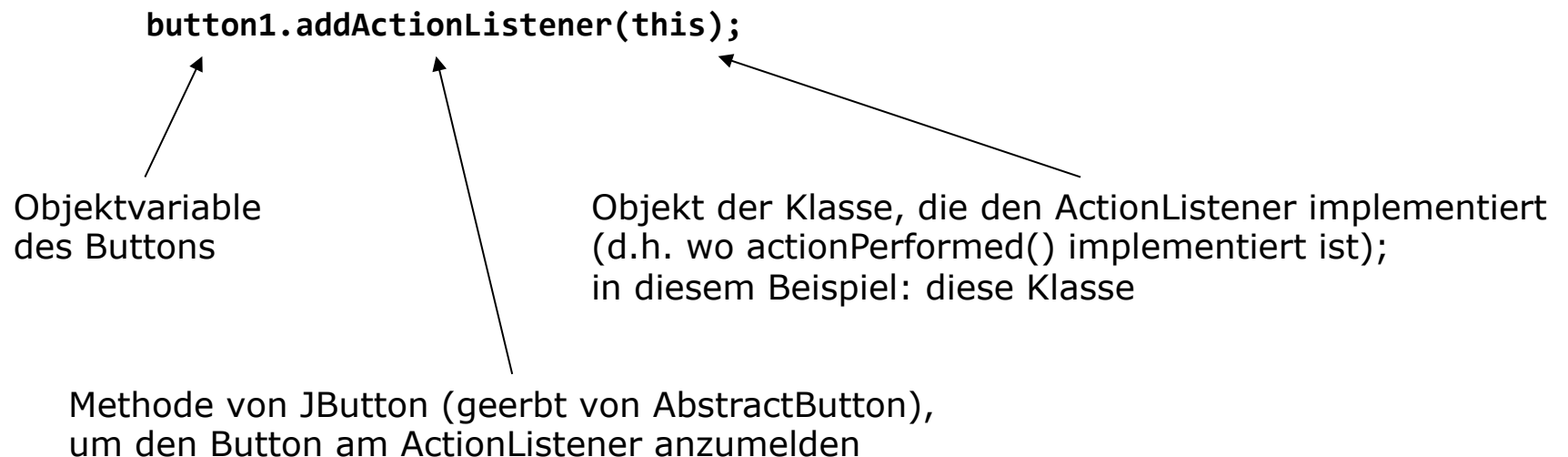
    @Override
    public void actionPerformed(ActionEvent e) {
        // TODO Auto-generated method stub
    }
}
```

- Eclipse:
 - schreiben Sie „implements ActionListener “ → import java.awt.event.*; ->
 - nutzen Sie „Add unimplemented methods “

Listener sind Interfaces

Beispiel ActionListener

- `actionPerformed(ActionEvent e)` ist die Methode, die aufgerufen wird, wenn eine Aktion (ein Ereignis) ausgelöst wird (z.B. Klicken eines Buttons)
- in diese Methode können wir nun schreiben, was passieren soll, wenn das Ereignis ausgelöst wird -> Ereignisbehandlung (event handler)
- vorher muss aber jeder Button an den ActionListener angemeldet werden:



Listener sind Interfaces

Beispiel ActionListener

```
public void actionPerformed(ActionEvent event) {
    Object quelle = event.getSource();

    if(quelle == klickMichbutton) {
        // change color
        farbIndex++;

        if(farbIndex == farben.length) {
            farbIndex = 0;
        }


        hauptPanel.setBackground(farben[farbIndex]);
        //label.setText(farben[farbIndex].toString());
    }
    else if(quelle == endeButton) {
        // exit program
        System.exit(0);
    }
}
```

- siehe **Ereignisbehandlung.java** in Moodle

ActionListener

- der ActionListener hat nur eine Methode: `actionPerformed(ActionEvent e)`
- tritt ein `ActionEvent` auf, so wird diese Methode automatisch aufgerufen
- `ActionEvent` ist eine Klasse aus dem Paket `java.awt.event` und erbt von `java.awt.AWTEvent`
- das `ActionEvent` selbst wird als Parameter dem Methodenaufruf übergeben
- `getSource()` ist eine Methode von `ActionEvent` (geerbt von `java.util.EventObject`) und liefert das Objekt (Typ `Object`) zurück, das das `ActionEvent` ausgelöst hat
- durch das Implementieren von `actionPerformed()` wird das durch den ActionListener ausgelöste Ereignis (`ActionEvent`) behandelt
→ dazu ist es aber notwendig, dass die Objekte, die durch den ActionListener abgehört werden sollen, beim ActionListener angemeldet werden (`addActionListener()`)
- brauchen wir auch bei Menüs, Textfeldern, Dateiauswahlfenstern, ...

Innere Klasse als Listener-Klasse



```
public class ChangeColorInnerClass extends JFrame
{
    Container contentPane;
    JButton button;


    ChangeColorInnerClass()
    {
        contentPane = this.getContentPane();
        button = new JButton("Change Background Color");
        contentPane.add(button, BorderLayout.NORTH);

        // am ActionListener anmelden
        button.addActionListener(new ButtonListener());

        this.setTitle("Change Background Color");
        this.setSize(400, 200);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setVisible(true);
    }

    // weiter auf nächster Folie mit innerer Klasse
}
```


Innere Klasse als Listener-Klasse




```
// innere Klasse
class ButtonListener implements ActionListener
{
    @Override
    public void actionPerformed(ActionEvent e) {
        float r = (float) Math.random();
        float g = (float) Math.random();
        float b = (float) Math.random();
        contentPane.setBackground(new Color(r,g,b));
    }
}

public static void main(String[] args) {
    new ChangeColorInnerClass();
}
}
```

innere Klasse ButtonListener in ChangeColorInnerClass

Anonyme Klasse als Listener-Klasse



```
public class ChangeColorAnonymClass extends JFrame{
    // Objektvariablen wie zuvor bei innerer Klasse
    ChangeColorAnonymClass()
    {
        // contentPane und button wie zuvor bei innerer Klasse

        // anonyme Listener-Klasse
        ActionListener al = new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent e) {
                float r = (float) Math.random();
                float g = (float) Math.random();
                float b = (float) Math.random();
                contentPane.setBackground(new Color(r,g,b));
            }

        }; // Ende anonyme Klasse

        button.addActionListener(al);
    }

    // Rest wie zuvor bei innerer Klasse
}
```

Anonyme Klasse als Listener-Klasse

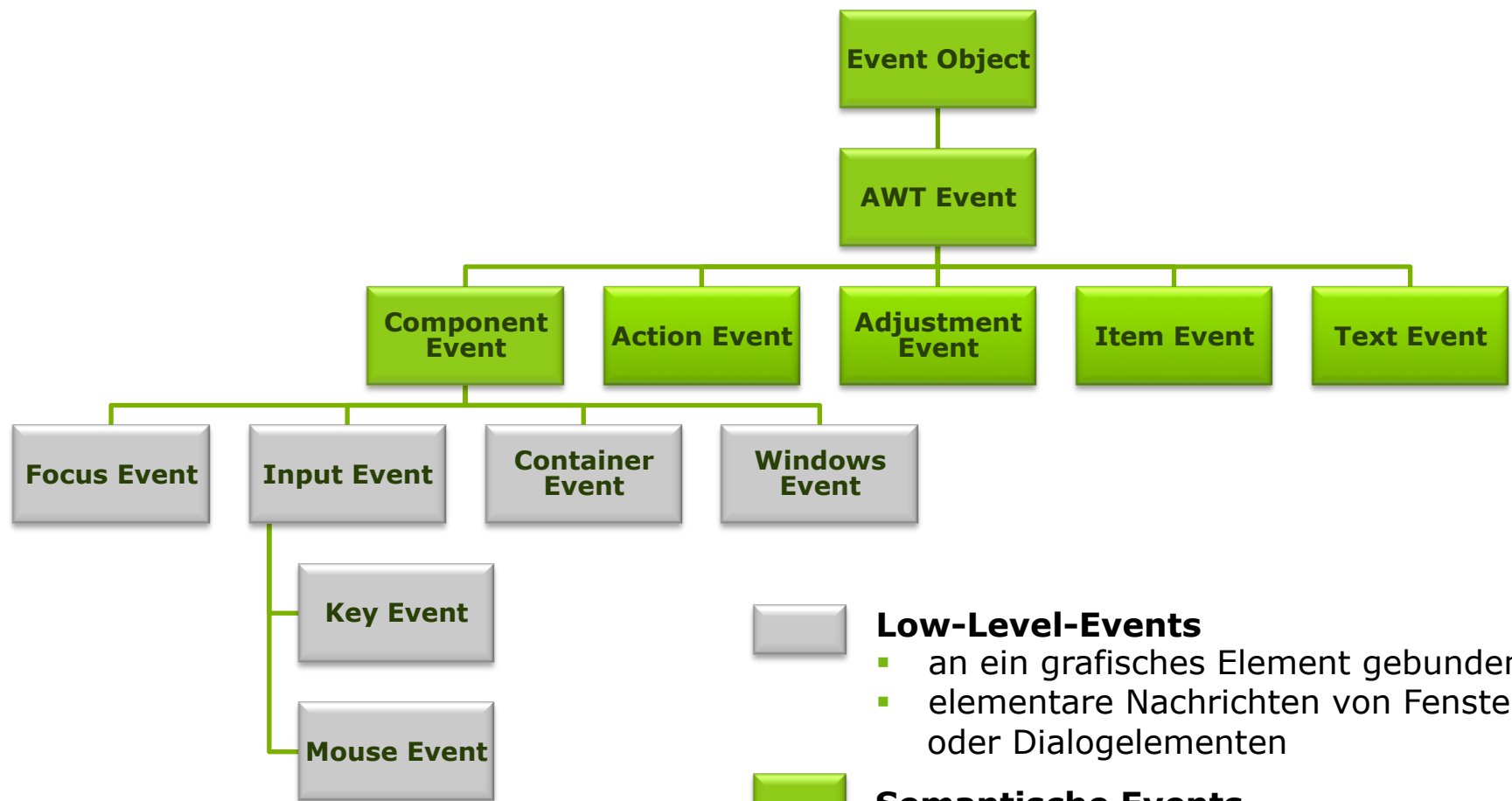
2. Variante



```
button.addActionListener(new ActionListener() {  
  
    @Override  
    public void actionPerformed(ActionEvent e) {  
        float r = (float) Math.random();  
        float g = (float) Math.random();  
        float b = (float) Math.random();  
        contentPane.setBackground(new Color(r,g,b));  
    }  
  
} // Ende anonyme Klasse  
); // Ende addActionListener
```

(LOW-LEVEL-)EVENTS

Ereignistypen



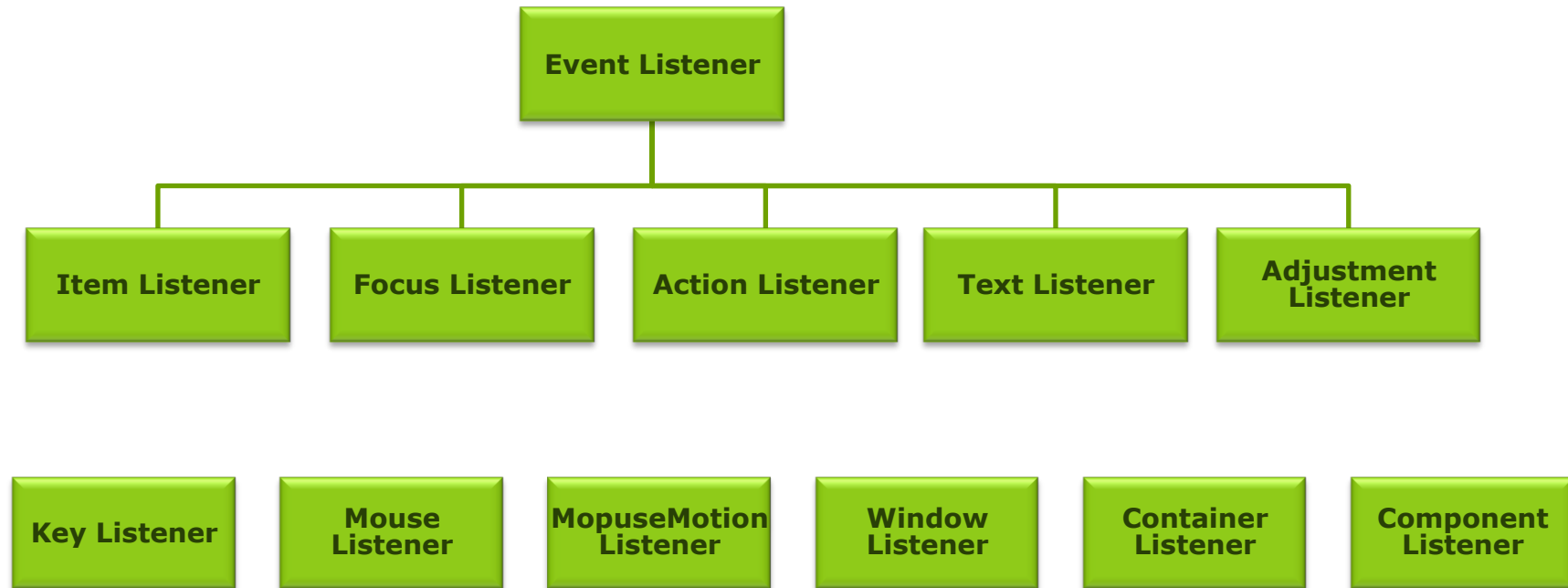
Low-Level-Events

- an ein grafisches Element gebunden
- elementare Nachrichten von Fenstern oder Dialogelementen

Semantische Events

- nicht an ein grafisches Element gebunden
- Ausführen von Kommandos oder Änderung von Zuständen

Ereignisempfänger



auch alles Subklassen von EventListener (und noch viel, viel mehr)

- jede der Methoden eines Ereignis-Listener erhält immer genau ein Objekt vom zugehörigen Ereignistypen als Argument; keine Rückgabe (void)

WEITERE INTERESSANTE LISTENER

Ereignisquellen

- Ereignisquellen sind Fenster, Dialogelemente, Buttons, Textfelder usw.
- jede Ereignisquelle muss sich **registrieren** (Registrierungsmethode add...),
z.B. addMouseListener(Klasse_die_MouseListener_implementiert Objekt)

```
class Klasse_die_MouseListener_implementiert implements MouseListener{  
    public void mousePressed(Mouse event){ tue etwas }  
    ...  
}
```

- Adapterklassen:
 - existiert für jeden Low-Level-Ereignisempfänger (FocusAdapter, KeyAdapter, MouseAdapter, MouseMotionAdapter, ComponentAdapter, ContainerAdapter)
 - abstrakte Klassen, die das jeweilige Listener-Interface implementieren
 - wenn man nicht alle Methoden aus dem Interface braucht, dann leitet man aus Adapter ab und überlagert die benötigten Methoden

Wir betrachten im Folgenden...

- WindowListener
- ComponentListener
- MouseListener
- MouseMotionListener
- FocusListener
- KeyListener

WindowListener: Window-Events

java.awt.event.WindowEvent

- wird immer dann ausgelöst, wenn sich der Status des Fensters geändert hat (erstellt, geschl., aktiviert, deaktiviert, minimiert, maximiert, wiederhergestellt)
- WindowListener (Interface) wird implementiert
 - `public void addWindowListener(WindowListener l)`
- Events sind vom Typ WindowEvent
- Methoden von WindowListener:
 - `windowActivated(WindowEvent e)` : Fenster wurde aktiviert
 - `windowClosed(WindowEvent e)` : Fenster wurde geschlossen
 - `windowClosing(WindowEvent e)` : Fenster soll geschlossen werden
 - `windowDeactivated(WindowEvent e)` : Fenster wurde deaktiviert
 - `windowDeiconified(WindowEvent e)` : Fenster wurde wiederhergestellt
 - `windowIconified(WindowEvent e)` : Fenster wurde minimiert
 - `windowOpened(WindowEvent e)` : Fenster wurde geöffnet
- <https://docs.oracle.com/javase/8/docs/api/java/awt/event/WindowListener.html>

Beispiel Fenster schließen

```
public class WindowClosingListener extends JFrame implements WindowListener
{
    WindowClosingListener() {
        super("Window closing");
        this.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        this.addWindowListener(this);
        this.setSize(200,100);
        this.setVisible(true);
    }

    // implemented empty - not used
    @Override public void windowOpened(WindowEvent e) {}
    @Override public void windowClosed(WindowEvent e) {}
    @Override public void windowIconified(WindowEvent e) {}
    @Override public void windowDeiconified(WindowEvent e) {}
    @Override public void windowActivated(WindowEvent e) {}
    @Override public void windowDeactivated(WindowEvent e) {}

    // next slide
}
```

beides!

Beispiel Fenster schließen

```
@Override public void windowClosing(WindowEvent e) {  
    final int answer = JOptionPane.showConfirmDialog(this,  
                                                    "Programm wirklich beenden?");  
  
    if(answer==JOptionPane.YES_OPTION)  
    {  
        this.setVisible(false);  
        this.dispose();  
        System.exit(0);  
    }  
}  
  
public static void main(String[] args) {  
    new WindowClosingListener();  
}  
  
}
```

- windowClosing definiert neues Verhalten beim Schließen
- wichtig: Standardverhalten beim Schließen ausstellen (DO_NOTHING_ON_CLOSE)

COMPONENT-EVENTS

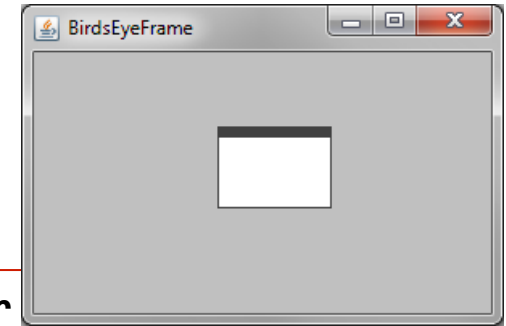
Component-Events

java.awt.event.ComponentEvent

- wird immer dann ausgelöst, wenn eine Komponente verschoben oder ihre Größe geändert wurde oder sich deren Anzeigezustand geändert hat
- da aus Component alles abgeleitet wird (siehe Klassenhierarchie AWT), haben diese Events für alle Klassen Bedeutung
- es wird das Interface ComponentListener implementiert
 - `(public void addComponentListener(ComponentListener l));`
- Events sind vom Typ ComponentEvent
- Methoden von ComponentListener:
 - `componentHidden(ComponentEvent e)`: Komponente wurde unsichtbar
 - `componentShown(ComponentEvent e)`: Komponente wurde sichtbar
 - `componentMoved(ComponentEvent e)`: Komponente wurde verschoben
 - `componentResized(ComponentEvent e)`: Größe der Komponente wurde geändert
- <http://docs.oracle.com/javase/7/docs/api/java/awt/event/ComponentListener.html>

Beispiel Component-Events

TestComponentEvents.java (Moodle)



```
class ComponentRepaintAdapter extends ComponentAdapter
{
    public void componentMoved(ComponentEvent event) {
        event.getComponent().repaint();
    }

    public void componentResized(ComponentEvent event) {
        event.getComponent().repaint();
    }
}
```

Adapter-Klasse
verwendet

```
class BirdsEyeFrame extends Frame {
    public BirdsEyeFrame()
    {
        super("BirdsEyeFrame");
        addWindowListener(new WindowClosingAdapter(true));
        addComponentListener(new ComponentRepaintAdapter());
        setBackground(Color.lightGray);
    }
}
```



MOUSE-EVENTS

Mouse-Events

java.awt.event.MouseEvent

- wird immer dann ausgelöst, wenn der Anwender innerhalb der Client-Area des Fensters eine Maustaste drückt oder loslässt
- sowohl linke als auch rechte Maustaste plus STRG, ALT, UMSCHALT
- es wird das Interface `MouseListener` implementiert
 - `public void addMouseListener(MouseListener l)`
- Events sind vom Typ `MouseEvent`
- Methoden von `MouseListener`:
 - `mousePressed(MouseEvent e)`: eine Maustaste wurde gedrückt
 - `mouseReleased(MouseEvent e)`: eine Maustaste wurde losgelassen
 - `mouseClicked(MouseEvent e)`: gedrückt und wieder losgelassen
 - `mouseEntered(MouseEvent e)`: Mauszeiger wurde in die Client-Area bewegt
 - `mouseExited(MouseEvent e)`: Mauszeiger wurde aus der Client-Area bewegt
- <http://docs.oracle.com/javase/8/docs/api/java/awt/event/MouseListener.html>

Die Klasse MouseEvent (extends InputEvent)

java.awt.event.MouseEvent

- bietet Methoden, um weitere Informationen über das ausgelöste Ereignis zu erhalten:
 - `public int getX()` - x-Koordinate der Maus (relativ zur Komponente*)
 - `public int getY()` - y-Koordinate der Maus (relativ zur Komponente*)
 - `public Point getPoint()` - x- und y-Koordinate als Point-Objekt*
 - `public int getButton()` - welche Maus-Taste für das Ereignis
 - `public int getClickCount()` - Anzahl der Klicks für das Ereignis
 - `public int getXOnScreen()` - x-Koordinate der Maus (relativ zum Monitor)
 - `public int getYOnScreen()` - y-Koordinate der Maus (relativ zum Monitor)
 - `public boolean isPopupTrigger()` - Popup-Menü gedrückt? (-> später)

* d.h. wenn Sie links oben in Ihrem JFrame klicken (in der ContentPane), ist es nicht der Punkt (0,0), sondern (`getInsets().left`, `getInsets().top`) → falls an JFrame angemeldet

von InputEvent geerbte Methoden

java.awt.event.InputEvent


- InputEvent ist Oberklasse von MouseEvent und KeyEvent
- bietet Methoden, um Informationen über den Zustand der Umschalttasten ALT, STRG, UMSCHALT und META zu erhalten:
 - `public boolean isShiftDown()` - UMSCHALT gedrückt?
 - `public boolean isControlDown()` - STRG gedrückt?
 - `public boolean isMetaDown()` - META gedrückt (Windows/Apple)?
oder rechte Maustaste
 - `public boolean isAltDown()` - ALT gedrückt?
 - `public boolean isAltGraphDown()` - ALT-GR gedrückt?
- Zeitpunkt des MouseEvents:
 - `public long getWhen()` - Zeitpunkt in Millisekunde (ab 1.1.1970)
- Erkennen von Doppelklicks:
 - Differenz der Zeit zweier Ereignisse (`getWhen()`)
und Abstand der Koordinatenpaare (`getPoint()`, `getX()`, `getY()`) vergleichen

Beispiel Mouse-Events (MouseListener)

```
public class TestMouseEvents extends JFrame implements MouseListener
{
    int cnt = 0;

    public TestMouseEvents()
    {
        super("Mausklicks");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.addMouseListener(this);
        this.setSize(300,200);
        this.setVisible(true);
    }

    public static void main(String[] args)
    {
        new TestMouseEvents();
    }
}
```



Beispiel Mouse-Events (MouseListener)

```
@Override
public void mouseClicked(MouseEvent e) {
    System.out.println("Mouse clicked");
}

@Override
public void mousePressed(MouseEvent event) {
    int x = event.getX();
    int y = event.getY();
    int nrClicks = event.getClickCount();
    boolean rightClick = event.isMetaDown();
    System.out.println("Mouse pressed");
    System.out.println("--> x : " + x + " y : " + y);
    System.out.println("--> Anzahl clicks " + nrClicks);
    if(rightClick) System.out.println("--> rechte Taste ");
    else System.out.println("--> linke Taste");
}

@Override
public void mouseReleased(MouseEvent e) {
    System.out.println("Mouse released");
}
```

Beispiel Mouse-Events (MouseListener)

```
@Override
public void mouseEntered(MouseEvent e) {
    System.out.println("ins Fenster bewegt");
}

@Override
public void mouseExited(MouseEvent e) {
    System.out.println("aus dem Fenster bewegt");
}
}
```


MOUSEMOTIONLISTENER

„MouseEvent-Events“

java.awt.event.MouseEvent

- wird immer dann ausgelöst, wenn die Maus bewegt wird (mit Taste gedrückt oder ohne)
- es wird das Interface `MouseListener` implementiert
 - `public void addMouseListener(MouseListener l)`
- Events sind vom Typ `MouseEvent`
- Methoden von `MouseListener`:
 - `mouseMoved(MouseEvent e)` : Maus wurde bewegt, keine Taste gedrückt
 - `mouseDragged(MouseEvent e)` : Maus wurde bewegt bei gedrückter Taste
- <http://docs.oracle.com/javase/8/docs/api/java/awt/event/MouseListener.html>

Beispiel Mouse-Events (MouseMotionListener)



```
public class TestMouseMotionEvents extends JFrame implements
MouseMotionListener
{
    public TestMouseMotionEvents()
    {
        super("Mausklicks");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.addMouseMotionListener(this);
        this.setSize(300,200);
        this.setVisible(true);
    }

    public static void main(String[] args)
    {
        new TestMouseMotionEvents();
    }
}
```

Beispiel Mouse-Events (MouseMotionListener)

```
@Override
public void mouseDragged(MouseEvent e) {
    System.out.println("Mouse dragged");
    System.out.println("--> (" + e.getX() + ", "+e.getY()+")");
}

@Override
public void mouseMoved(MouseEvent e) {
    System.out.println("Mouse moved");
    System.out.println("--> (" + e.getX() + ", "+e.getY()+")");
}
```

FOCUS-EVENTS

Focus-Events

java.awt.event.FocusEvent

- wird immer dann ausgelöst, wenn eine Eingabe erfolgt, um anzuzeigen, welches Fenster gerade aktiv ist (welches Fenster die Eingabe empfängt)
- es wird das Interface FocusListener implementiert
 - `public void addFocusListener(FocusListener l)`
- Events sind vom Typ FocusEvent
- Methoden von FocusListener:
 - `focusLost(FocusEvent e)` : Komponente hat Fokus abgegeben
 - `focusGained(FocusEvent e)` : Komponente hat Fokus bekommen
- <http://docs.oracle.com/javase/8/docs/api/java/awt/event/FocusListener.html>

KEY-EVENTS

Key-Events

java.awt.event.KeyEvent

- alle Tastatureingaben werden an die fokussierte Komponente gesendet
- es wird das Interface `KeyListener` implementiert
 - `public void addKeyListener(KeyListener l)`
- Events sind vom Typ `KeyEvent`
- Methoden von `KeyListener`:
 - `keyPressed(KeyEvent e)` : beliebige Taste gedrückt
 - `keyReleased(KeyEvent e)` : beliebige Taste losgelassen
 - `keyTyped(KeyEvent e)` : Zeichentaste* gedrückt
- <http://docs.oracle.com/javase/8/docs/api/java/awt/event/KeyListener.html>
- *Zeichentaste: alle gültigen Unicode-Zeichen (auch ESC, LEER und TAB)
- Funktionstasten: F1, F2, ..., POS1, CURSORTASTEN, STRG, ALT, UMSCHALT, ALT-GR

Erkennen, welche Taste gedrückt wurde

java.awt.event.KeyEvent

- Methoden der Klasse KeyEvent (von 12):
 - `public int getKeyCode()` : liefert den Tasten-Code als int*
 - `public char getKeyChar()` : liefert das Tasten-Zeichen als char
- aus InputEvent geerbte Methoden:
 - `(public boolean) isShiftDown(), isControlDown(), isMetaDown(), isAltDown(),...`
- Tasten-Codes (symbolische Konstanten aus KeyEvent):
 - `(static int) VK_0, ..., VK_9, VK_A, ..., VK_Z, VK_ENTER, VK_SPACE, VK_TAB, VK_ESCAPE, VK_F1, ..., VK_F12, VK_HOME, VK_END, VK_INSERT, VK_DELETE, VK_BACK_SPACE, VK_PAGE_UP, VK_PAGE_DOWN, VK_DOWN, VK_UP, VK_RIGHT, VK_LEFT`
- <http://docs.oracle.com/javase/8/docs/api/java/awt/event/KeyEvent.html>

Erkennen, welche Taste gedrückt wurde

Probleme

- Zeichentasten: für `keyTyped` mit `getKeyChar` die Zeichentasten abfragen; Funktionstasten werden nicht berücksichtigt (wird `keyTyped` nicht aufgerufen)
- Achtung!: `keyTyped` mit `getKeyCode` liefert (bei Zeichentasten) immer `KeyEvent.VK_UNDEFINED` zurück
- `keyPressed` wird bei allen Tasten ausgelöst (inklusive Tastatur-Repeats – mehrmaliges Auslösen)
 - `getKeyChar` liefert bei Funktionstasten `KeyEvent.CHAR_UNDEFINED` zurück

		<code>getKeyCode</code>	<code>getKeyChar</code>
<code>keyTyped</code>	Zeichentaste	<code>VK_UNDEFINED</code>	Taste als char
	Funktionstaste	--	--
<code>keyPressed</code>	Zeichentaste	<code>VK_...</code>	Taste als char
	Funktionstaste	<code>VK_...</code>	<code>CHAR_UNDEFINED</code>

Beispiel Key-Events

```
public class TestKeyEvents extends JFrame implements KeyListener{
    JTextField textField = new JTextField(20);
    JLabel label = new JLabel("Enter some text (at least 3 ");
    JButton button = new JButton("Press me");

    public TestKeyEvents()
    {
        super("Tastaturereignisse");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JPanel inputPanel = new JPanel();
        inputPanel.add(label);
        inputPanel.add(textField);
        this.add(inputPanel, BorderLayout.CENTER);
        JPanel buttonPanel = new JPanel();
        buttonPanel.add(button);
        button.setEnabled(false);
        this.add(buttonPanel, BorderLayout.SOUTH);
        textField.addKeyListener(this);
        this.setSize(300,200);
        this.setVisible(true);
    }
}
```



Beispiel Key-Events

```
public static void main(String[] args)
{
    new TestKeyEvents();
}

@Override
public void keyTyped(KeyEvent e) {
    System.out.println("keyTyped --> Taste getKeyCode() : " +
                       e.getKeyCode());
    System.out.println("keyTyped --> Taste getKeyChar() : " +
                       e.getKeyChar());
}

@Override
public void keyPressed(KeyEvent e) {
    System.out.println("keyPressed --> Taste getKeyCode() : " +
                       e.getKeyCode());
    System.out.println("keyPressed --> Taste getKeyChar() : " +
                       e.getKeyChar());
}
```

Beispiel Key-Events

```
@Override
public void keyReleased(KeyEvent e) {
    System.out.println("keyReleased --> Taste getKeyCode() : " +
                        e.getKeyCode());

    switch(e.getKeyCode())
    {
        case KeyEvent.VK_ENTER : System.out.println("Enter"); break;
        case KeyEvent.VK_ESCAPE : System.out.println("ESC"); break;
        case KeyEvent.VK_F1 : System.out.println("f1"); break;
        case KeyEvent.VK_DOWN : System.out.println("down"); break;
    }
    System.out.println("keyReleased --> Taste getKeyChar() : " +
                        e.getKeyChar());

    if(textField.getText().length()>=3) button.setEnabled(true);
}
}
```

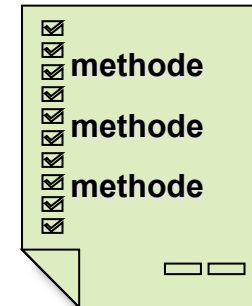
Übersicht & Zusammenfassung



Source-Objekt

löst Ereignis  wird zugestellt dem: **Listener**

Event Type



Source löst aus: Event

Source-Object	Event Type
JButton	ActionEvent
TextField	ActionEvent
JCheckBox	ItemEvent, ActionEvent
JRadioButton	ItemEvent, ActionEvent
JComboBox	ItemEvent, ActionEvent
JList	ListSelectionEvent
JMenuItem	ActionEvent
JScrollBar	AdjustmentEvent

Source löst aus: Event

Source-Object	Event Type
Window	WindowEvent
Container	ContainerEvent
Component	ComponentEvent, FocusEvent, KeyEvent, MouseEvent

Event wird zugestellt an: Listener

Event Type	Listener Interface
ActionEvent	ActionListener
ItemEvent	ItemListener
WindowEvent	WindowListener
ContainerEvent	ContainerListener
ComponentEvent,	ComponentListener
FocusEvent	FocusListener

Event wird zugestellt an: Listener

Event Type	Listener Interface
MouseEvent	MouseListener, MouseMotionListener
AdjustmentEvent	AdjustmentListener
KeyEvent	KeyListener
ListSelectionEvent	ListSelectionListener

Listener - Methoden

Listener Interface	Listener Methode(n)
ActionListener	actionPerformed()
ItemListener	itemStateChanged()
WindowListener	windowActivated windowDeactivated windowClosed windowClosing windowIconified windowDeiconified windowOpened

Listener - Methoden

Listener Interface	Listener Methode(n)
MouseListener	mousePressed(), mouseReleased(), mouseEntered(), mouseExited(), mouseClicked()
MouseMotionListener	mouseDragged(), mouseMoved()

Listener - Methoden

Listener Interface	Listener Methode(n)
KeyListener	keyPressed(), keyReleased(), keyTyped()
FocusListener	focusGained(), focusLost()
AdjustmentListener	adjustmentValueChanged()

Listener - Methoden

Listener Interface	Listener Methode(n)
ContainerListener	componentAdded(), componentRemoved()
ComponentListener	componentHidden componentMoved componentResized componentShow
ListSelectionListener	valueChanged()

Übersicht

<i>Komponente</i>	<i>Erzeugte Events</i>	<i>Listener-Interface</i>	<i>Listener- Methode</i>
JButton	ActionEvent	ActionListener	actionPerformed
JCheckbox	ItemEvent	ItemListener	itemStateChanged
	ActionEvent	ActionListener	actionPerformed
JRadioButton	ItemEvent	ItemListener	itemStateChanged
	ActionEvent	ActionListener	actionPerformed
JTextField	ActionEvent	ActionListener	actionPerformed

Übersicht

<i>Komponente</i>	<i>Erzeugte Events</i>	<i>Listener-Interface</i>	<i>Listener- Methode</i>
JMenuItem	ActionEvent	ActionListener	actionPerformed
JComboBox	ItemEvent	ItemListener	itemStateChanged
	ActionEvent	ActionListener	actionPerformed
JScrollBar	AdjustmentEvent	AdjustmentListener	adjustmentValueChanged

Übersicht

<i>Komponente</i>	<i>Erzeugte Events</i>	<i>Listener-Interface</i>	<i>Listener- Methode</i>
Window	WindowEvent	WindowListener	windowActivated windowDeactivated windowClosed windowClosing windowIconified windowDeiconified windowOpened
Container	ContainerEvent	ContainerListener	componentAdded componentRemoved

Übersicht

<i>Komponente</i>	<i>Erzeugte Events</i>	<i>Listener-Interface</i>	<i>Listener- Methode</i>
Component	ComponentEvent	ComponentListener	componentHidden componentMoved componentResized componentShow
	FocusEvent	FocusListener	focusGained focusLost
	KeyEvent	KeyListener	keyPressed keyReleased keyTyped

Übersicht

<i>Komponente</i>	<i>Erzeugte Events</i>	<i>Listener-Interface</i>	<i>Listener- Methode</i>
Component	MouseEvent	MouseListener	mouseClicked mouseEntered mouseExited mousePressed mouseReleased
	MouseEvent	MouseMotionListener	mouseDragged mouseMoved

Source Code

Bsp: ActionListener implementieren

```
class ActionListener1 implements ActionListener{
    public void actionPerformed((ActionEvent ev) {
        Object source = ev.getSource();

        if( source == butLaf[Metal]){
            updateLAF(laf[Metal]);
        }
        else if(source == butLaf[Motif]){
            updateLAF(laf[Motif]);
        }
        else if(source == butLaf[Win]){
            updateLAF(laf[Win]);
        }
    }
}
// end of inner class ActionListener1
```

Source Code

Nicht vergessen, ActionListener an das Quellobjekt anzubinden!

```
JButton[] butLaf = new JButton[laf.length]; // Array anlegen

butLaf[Metal] = new JButton("Metal");
butLaf[Motif]= new JButton("Motif");
butLaf[Win]= new JButton("Windows");

// Listener an Buttons anbinden
for( int i = 0; i < laf.length; i++){
    butLaf[i].addActionListener( new ActionListener1() );
}
```

„Schema F“

```
class SwingFenster extends JFrame{
    // Instanz- oder Membervariablen
    JButton button;
    public SwingFenster(){ // Default-Konstruktor
        this("Mein erstes Swing-Fenster"); // Aufruf des eigenen Konstruktors mit Parametern
    }

    public SwingFenster( String titel ){ // Konstruktor mit Parameter
        super(titel); // Aufruf des Konstruktors der Oberklasse

        ...
        // Listener anbinden – nicht vergessen!
        button.addActionListener( new ActionListener1() );

    } // end of constructor

    // eigene Methoden

    // innere Klassen
    class ActionListener1 implements ActionListener{
        public void actionPerformed((ActionEvent ev) ){
            ...
        }
    } // end of inner class ActionListener1
} // end of class SwingFenster
```