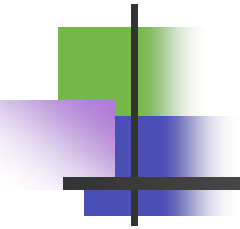


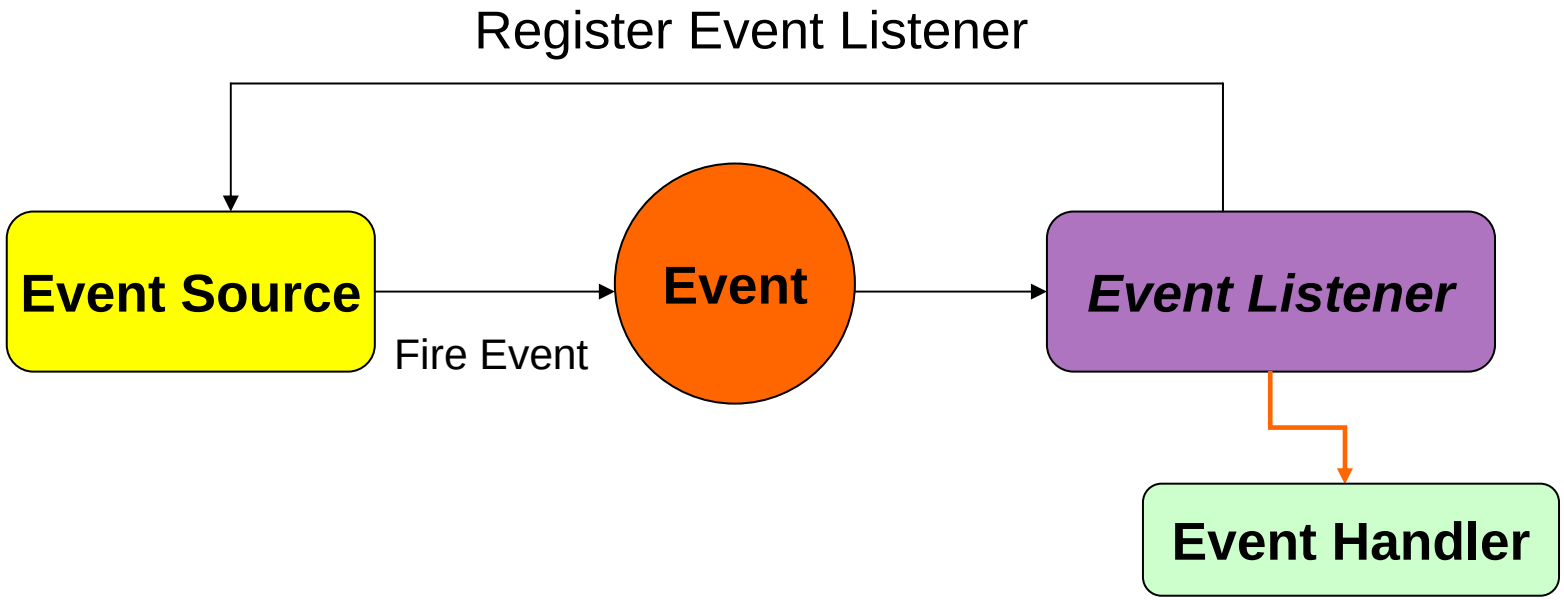
Event-Handling in Java

java.awt.event

Events

- Die Interaktion mit einem GUI geschieht über Events.
- Ein Event ist eine Aktion, welche durch einen Benutzer initialisiert wurde, sei es durch Drücken eines Buttons, Verschieben der Maus, Drücken einer Taste ...
- Damit Java auf die Aktion eines Benutzers reagiert, muss zuerst eine Event-Listener registriert und ein zugehöriger Event-Handler implementiert werden.

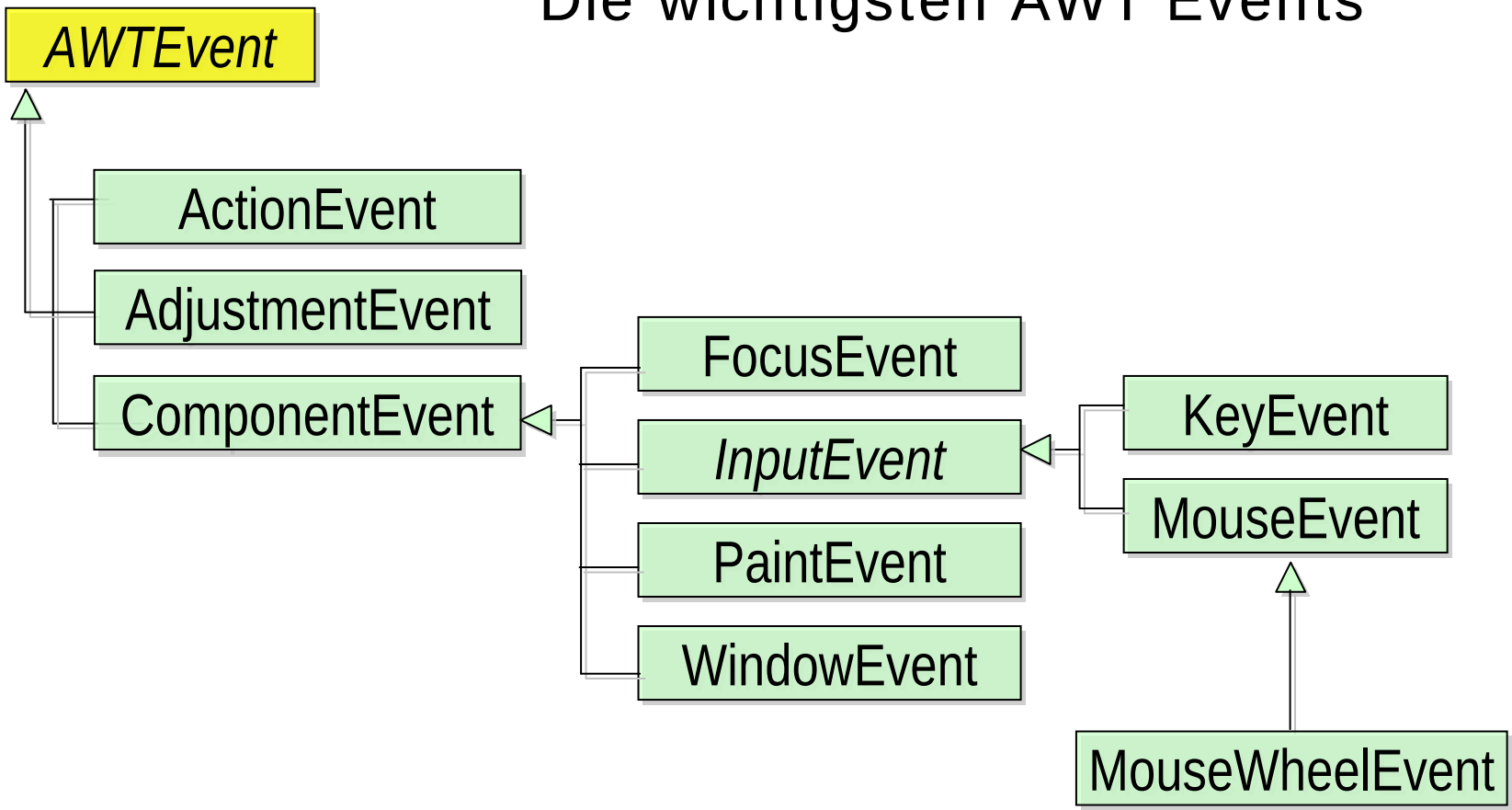
Event Handling in Java



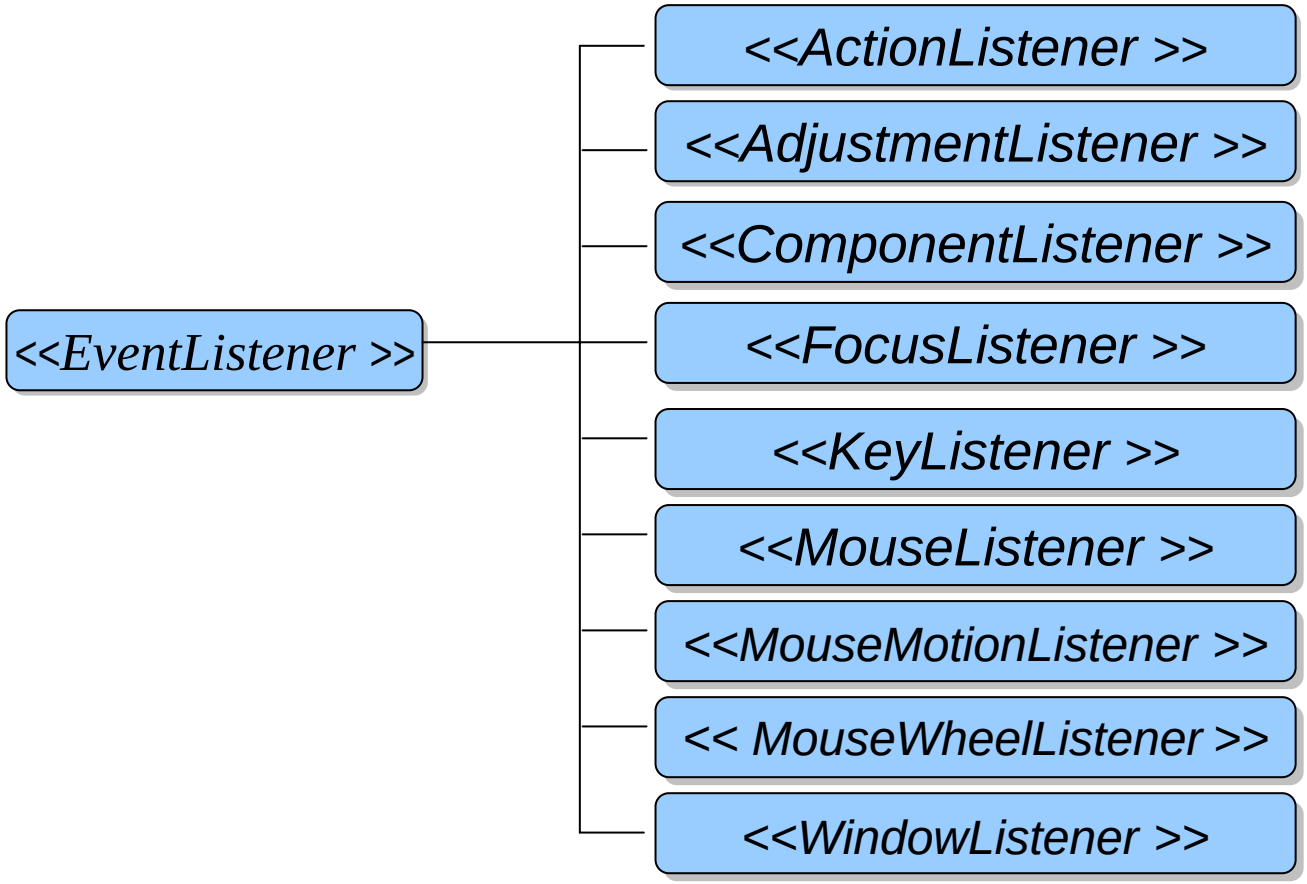
Für jedes (beobachtete) Event wird ein Event Listener erzeugt, welcher die entsprechende(n) Methode(n) (*Event Handler, Callback-Funktion*) implementiert.

Auszug aus dem java.awt.event Package

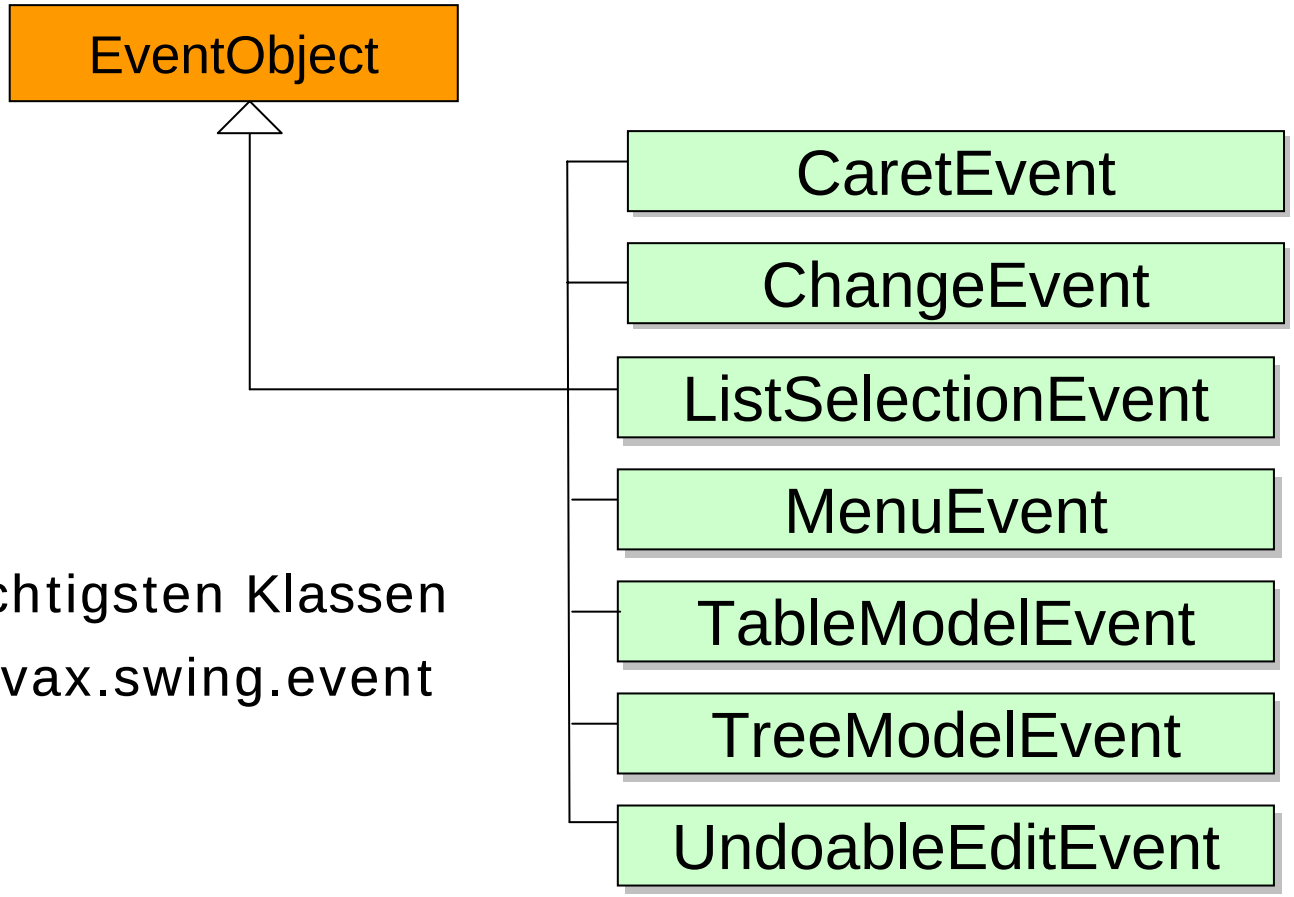
Die wichtigsten AWT Events



Die entsprechenden EventListener

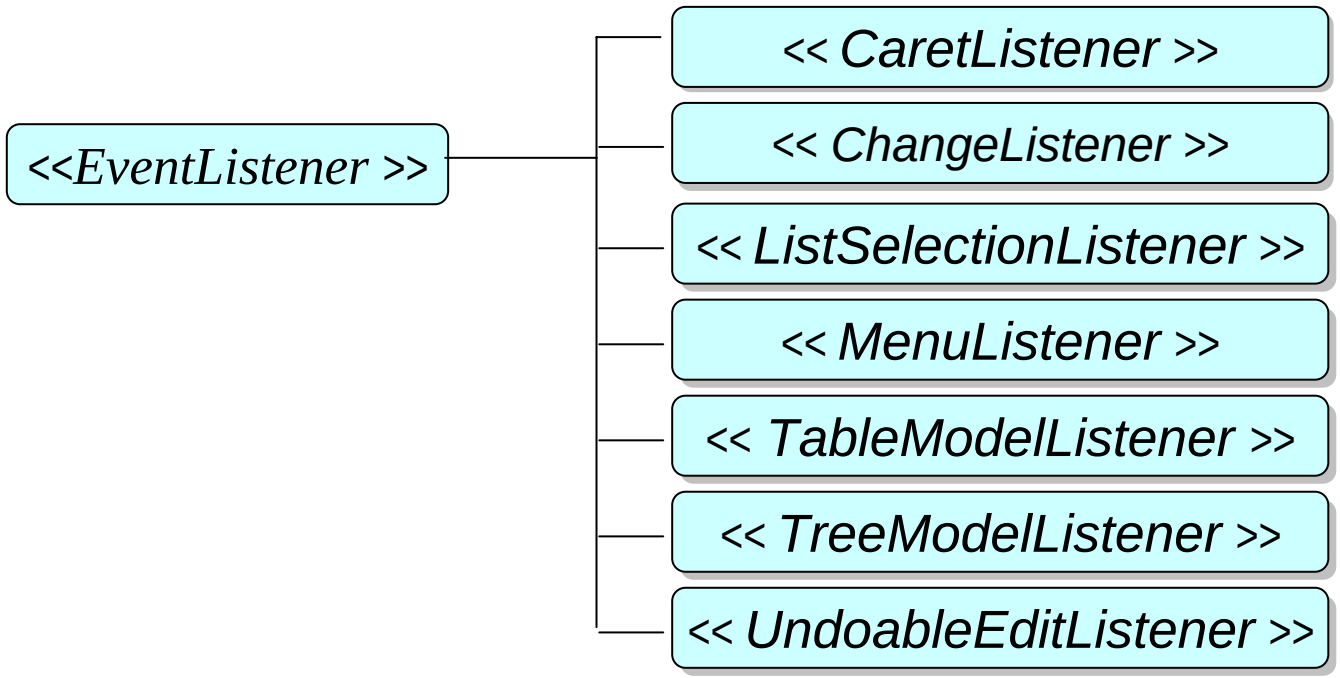


Auszug aus dem Swing Event Package



Die wichtigsten Klassen aus javax.swing.event

Die entsprechenden EventListener



Einige Komponenten und ihre Events

	Event Listener							
Component	**action**	**caret**	**change**	**document**	**item**	**list selection**	**window**	**other**
abstract button	x		x		x			
combo box	x				x			
frame							x	
menu								*MenuListener*
menu item	x							*MenuKeyListener*
tabbed pane			x					
table						x		*TableModelListener*
text area		x		x				
text field	x	x		x				
tree								*TreeModelListener*

Beispiele von EventListener Interfaces

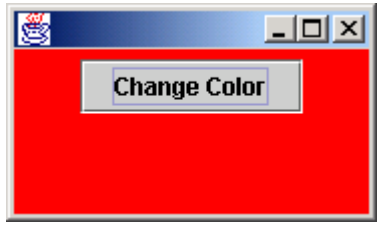
```
public interface ActionListener extends EventListener
{
   public void actionPerformed(ActionEvent e);
}

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

public interface MouseListener extends EventListener
{
   public void mouseClicked(MouseEvent e);
  ...
}
```

Events auf einem Button

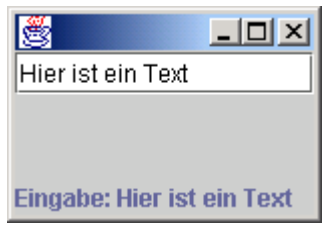
```
import java.awt.event.*; import java.awt.*; import javax.swing.*;
public class ButtonEvent extends JFrame {
 JPanel panel;
 ButtonEvent() {
   panel = new JPanel(); add(panel);
   JButton b = new JButton("Change Color");
   b.addActionListener(new bAction());
   panel.add(b);
 }
 class bAction implements ActionListener {
   public void actionPerformed(ActionEvent event) {
     if (panel.getBackground() == Color.red)
       panel.setBackground(Color.blue);
     else
       panel.setBackground(Color.red);
   }
 }
  // main ...
}
```



Events in einem TextField

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

public class TextEvent1 extends JFrame
        implements ActionListener
{
  JLabel output;
  TextEvent1() {
     JTextField text = new JTextField( 20 );
     output = new JLabel();
     text.addActionListener( this );
     add(text, BorderLayout.NORTH);
     add(output, BorderLayout.SOUTH);
  }
  public void actionPerformed(ActionEvent e) {
     output.setText("Eingabe: " +
           e.getActionCommand() );
  }
     ...
}
```



Key Events

```
import javax.swing.*; import java.awt.*;
import java.awt.event.*;
public class TextEvent2 extends JFrame
                        implements KeyListener
{
  JTextField text; JLabel output;
  TextEvent2(){
    text = new JTextField( );
    output = new JLabel();
    text.addKeyListener( this );
    add(text, BorderLayout.NORTH);
    add(output, BorderLayout.SOUTH);
  }
  public void keyPressed( KeyEvent e ){
    output.setText("Eingabe: " + text.getText());
  }
  public void keyTyped( KeyEvent e ){ ... }
  public void keyReleased( KeyEvent e ){ ... }
  ...
}
```

Mouse Events

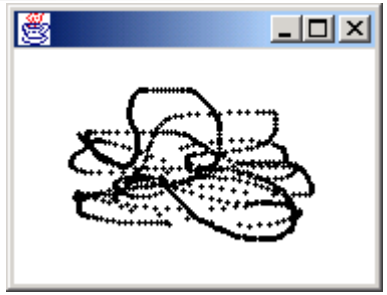
Es gibt drei Sorten von MouseListeners:

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

MouseMotionListener
public void mouseDragged( MouseEvent e );
public void mouseMoved( MouseEvent e );

MouseWheelListener
public void mouseWheelMoved( MouseWheelEvent e );
```

Verwenden von Mouse Events



```
import javax.swing.*; import java.awt.*;
import java.awt.event.*;
public class MEvent extends JFrame {
 private int xVal = 0, yVal = 0;
 MEvent(){
  addMouseMotionListener( new MouseMotionAdapter()
  {
    public void mouseDragged( MouseEvent e )
    {
      xVal = e.getX(); yVal = e.getY();
      repaint();
    }
  });
 }
 public void paint( Graphics g ) {
      g.fillOval( xVal, yVal, 3, 3 ); }

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

Menu-Events

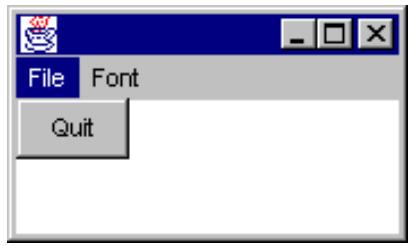
```
  // create menubar
bar = new JMenuBar();

// create the file menu
fileMenu = new JMenu( "File" );
quit = new JMenuItem( "Quit" );
fileMenu.add( quit );
quit.addActionListener(new ActionListener()
{
   public void actionPerformed( ActionEvent e )
   {  System.exit( 0 ); }
} );

// create font menu
fontMenu = new JMenu( "Font" );

// add menu to menu bar
bar.add( fileMenu );
bar.add( fontMenu );

// set the menubar for the frame
setJMenuBar( bar );
```



EventListener mit Parameter

```
Color _color;
JRadioButtonMenuItem red, blue;

JMenu colorMenu = new JMenu("Color");
red = new JRadioButtonMenuItem ("Red");
blue = new JRadioButtonMenuItem ("Blue");
colorMenu.add(red); colorMenu.add(blue);

red.addItemListener(new ClAction(Color.red));
blue.addItemListener(new ClAction(Color.blue));
                    ……

class ClAction implements ItemListener
{
  Color color;
  ClAction( Color color )
    { this.color = color; }

  public void itemStateChanged( ItemEvent e )
    { _color = color; }
}
```

Window Listener/ Window Adapter

Der WindowListener unterstützt sieben verschiedene Ereignisse:

```
public void windowActivated( WindowEvent e );
public void windowDeactivated( WindowEvent e );
public void windowOpened( WindowEvent e );
public void windowClosing( WindowEvent e );
public void windowClosed( WindowEvent e );
public void windowIconified( WindowEvent e );
public void windowDeiconified( WindowEvent e );
```

Darum gibt es zur Vereinfachung den entsprechenden WindowAdapter.

Window Listener/ Window Adapter

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

public class MyFrame extends JFrame
{
  public MyFrame()
  {
    JLabel label = new JLabel("Close Window to Quit");
    add( label );
    setFont( new Font("Helvetica", Font.BOLD, 16 ));
    addWindowListener( new WindowAdapter()
    {
      public void windowClosing( WindowEvent e )
          { System.exit(0); }
    } );
  }

  static public void main( String args[] )
  {
    MyFrame mf = new MyFrame();
    mf.setSize( 200, 80 ); mf.setVisible( true );
  }
}
```

AbstractAction

```
cutAction = new CutAction(textArea);

EditMenu() {  // create edit menu
  super("Edit");
  add(cutAction);
  add(copyAction);
}

MyToolBar() {  // create toolbar
  add(cutAction);
  add(copyAction);
}

class CutAction extends AbstractAction {
  CutAction(JTextPane text) {
    super("Cut", new ImageIcon("Cut.gif"));
  }

  public void actionPerformed(ActionEvent e) {
      text.cut();
  }
}
```

Anonyme Klasse, innere Klasse, Interface...

Den EventListener implementieren kann

- die (Nutzer-) Klasse selbst
  - welche das Interface implementiert
  - oder von Adapter ableitet
- eine separate Klasse
  - welche das Interface implementiert
  - oder von Adapter ableitet
- eine innere Klasse
  - welche das Interface implementiert
  - oder von Adapter ableitet
  - eventuell anonym

Vorteile / Nachteile ?