Grundlagen der Informatik für Ingenieure I

10. Eventhandling in Java

10.1 Eventhandling

◆ *Eventhandling* in Java ist Teil des *AWT-package*.

◆ Es dient der Kommunikation eines Programmsystems mit der "Benutzeroberfläche", dem *User-Interface (UI; GUI)*.

◆ Es handelt sich hierbei z. B. um

- *Mouse*-Bewegungen oder *Mouse*-Clicks
- Keyboard-Betätigungen
- Interaktionen mit der Graphischen Benutzeroberfläche.

◆ Wir müssen in unserem Programm eine Konstruktion vorsehen, die aufpasst, ob ein solches Ereignis (Event) eintritt.

10.2 Listener

- Das *AWT-Package* stellt uns, geordnet nach *Eventklassen*, ***"Listener"*** zur Verfügung.

- ***Listener*** werden mit Hilfe von ***Interfaces*** implementiert, die durch ihre Definition das *Handling* mit *Events* vorgeben.

- Beispiele für *Listener* sind:

```
MouseListener
ActionListener
KeyListener
```

- Damit das System weiß, daß wir einen solchen Dienst in Anspruch nehmen wollen, müssen wir ***Listener*** bei den jeweiligen AWT-Komponenten (Button, Checkbox, ...) anmelden (registrieren) .

10.2 Listener

- Für jeden ***Listener*** gibt es eine ***add-method()*** mit der man ihn registrieren lassen kann, z. B.:

```
addMouseListener(...)
addActionListener(...)
addKeyListener(...)
```

- Zu jedem ***Listener*** existiert eine zugehörige ***Event class*****,**
  - z. B. zum *MouseListener* die Klasse *MouseEvent* oder zum *ActionListener* die Klasse *ActionEvent.*
  - In diesen Klassen sind Methoden definiert, die man bei der Behandlung der zugehörigen *Events* benötigt (z.B: *getX(), getY()*)
  - Tritt eine *Event* auf, wird ein Objekt der zugehörigen *Event class* erzeugt und der entsprechenden *Listener*-Methode übergeben.

10.3 Adapter

- Bei der Verwendung eines ***Interfaces*** müssen alle Methoden implementiert werden, auch dann, wenn man nur einen Teil der Funktionalität ausnutzen will. Aus diesem Grund gibt es für viele *Listener* ***Adapter.***

- ***Adapter*** sind (abstrakte) Klassen! Sie implementieren vollständig - im Sinne von Methodendefinitionen, u. U. auch mit leerem Rumpf - die entsprechenden *Interfaces* z. B.

```
public abstract class MouseAdapter.....
       implements MouseListener {...}
```

- Wenden wir ***Adapter*** an, so brauchen wir nur die Methoden zu implementieren *(overriding)*, die wir zur Lösung unseres Problems benötigen.

10.4 Beispiel: Buttons

- Beispiel für die Anwendung eines Listeners: **ButtonActionsTest**

```
// button actions

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

public class ButtonActionsTest extends java.applet.Applet {

    Button redButton, blueButton, greenButton;

    class RedButtonListener implements ActionListener {
        public void actionPerformed( ActionEvent e ) {
            setBackground( Color.red );
            repaint();
        }
    }

    class BlueButtonListener implements ActionListener {
        public void actionPerformed( ActionEvent e ) {
            setBackground( Color.blue );
            repaint();
        }
    }
```

```
    class GreenButtonListener implements ActionListener {
        public void actionPerformed( ActionEvent e ) {
        setBackground( Color.green );
        repaint();
        }
    }

    public void init() {
        setFont( new Font( "Helvetiva", Font.BOLD, 72 ) );
        setBackground( Color.white );
        setLayout( new FlowLayout( FlowLayout.CENTER, 10, 10 ) );

        redButton = new Button( "Red" );
        RedButtonListener redListener = new RedButtonListener();
        redButton.addActionListener( redListener );
        add( redButton );

        blueButton = new Button( "Blue" );
        BlueButtonListener blueListener = new BlueButtonListener();
        blueButton.addActionListener( blueListener);
        add( blueButton );

        greenButton = new Button( "Green" );
        GreenButtonListener greenListener = new GreenButtonListener();
        greenButton.addActionListener( greenListener);
        add( greenButton );
    }
}
```


10.4 Beispiel: Buttons

◆ Zum Beispiel:

- Es wird das Interface ***ActionListener*** verwendet
- Der ***ActionListener*** enthält nur eine Methode:

  `actionPerformed( ActionEvent e )`

- Zu ***actionPerformed*** gehört die Eventklasse: ***ActionEvent***
- Mit der ***add-method()****:*

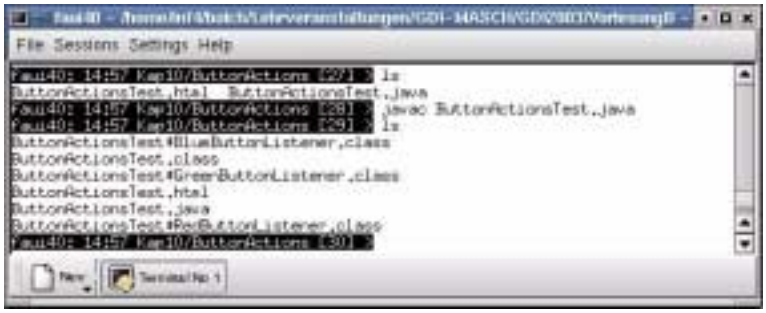
  `addActionListener( ... )`

  wird der *ActionListener* bei dem jeweiligen Button registriert.

- Die *ButtonListeners* wurden **innerhalb** einer anderen Klasse (der Applet-Klasse) definiert. Man nennt solche Klassen ***Inner Classes.***

10.4 Beispiel: Buttons

◆ Compilieren mit ***javac***:



- Es werden für die äußere Klasse *ButtonActionsTest* und für die 3 *Inner Classes* jeweils eine Class-Datei (Byte-Code) erzeugt.

10.4 Beispiel: Buttons

◆ Ergebnis mit Appletviewer:

10.5 Anonymous Inner Classes

- Eine Variante der ***Inner Class*** ist die ***Anonymous Inner Class.***
- Beispiel mit der Definition von *Eventhandler-Classes* als *Anonymous Inner Classes:*
  - Es wird für jeden Button ein *Eventhandler-Object* **`"ButtonListener"`** erzeugt.
  - Die Definitionen erfolgt "innerhalb" einer Klasse als sogenannte *Anonymous Inner Classes*.
  - Der Compiler "generiert" jeweils ein separates *.class-file*.
  - Zur Laufzeit ist dadurch die Zuordnung Event / Eventhandler gegeben.

10.5 Anonymous Inner Classes

- Beispiel: ButtonActionsInnerAnonym

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

public class ButtonActionsInnerAnonym extends java.applet.Applet {

  Button redButton,blueButton,greenButton;

  public void init() {

    setFont( new Font( "Helvetiva", Font.BOLD, 72 ) );
    setBackground( Color.white );
    setLayout( new FlowLayout( FlowLayout.CENTER, 10, 10 ) );

    redButton = new Button( "Red" );
    add( redButton );

    blueButton = new Button( "Blue" );
    add( blueButton );

    greenButton = new Button( "Green" );
    add( greenButton );
```

10.5 Anonymous Inner Classes

◆ Beispiel: ButtonActionsInnerAnonym

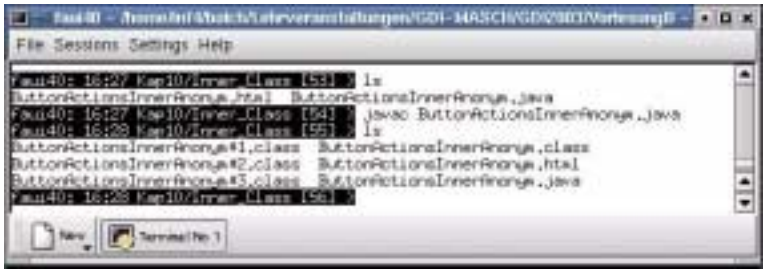
```
redButton.addActionListener( new ActionListener() {
    public void actionPerformed( ActionEvent e ) {
      setBackground( Color.red );
      repaint();
    }
  });

  blueButton.addActionListener( new ActionListener() {
    public void actionPerformed( ActionEvent e ) {
      setBackground( Color.blue );
      repaint();
    }
  });

  greenButton.addActionListener( new ActionListener() {
    public void actionPerformed( ActionEvent e ) {
      setBackground( Color.green );
      repaint();
    }
  });
  }
}
```


10.5 Anonymous Inner Classes

◆ Compilieren mit ***javac***:



- Es wird für jede ***Anonyme Innere Klasse*** eine *class-Datei* erzeugt, die zur Unterscheidung mit Ziffern versehen werden.

- Durch Einführung der ***Anonymen Inneren Klassen*** wird der Code kürzer und übersichtlicher, durch deutliche Trennung von der Erzeugung der *Buttons* und der Registrierung (Anmeldung) der ***ActionListeners*** bei den *Buttons*.

10.5 Anonymous Inner Classes

◆ Ergebnis mit Appletviewer:

10.6 *Mouse-* und *Key - Eventhandling*

◆ *Mouse- und Key-Listener*:

Listener Interface	Events	Method Definition
MouseListener	mouse down	public void mousePressed(MouseEvent e)
	mouse up	public void mouseReleased(MouseEvent e)
	mouse enter	public void mouseEntered(MouseEvent e)
	mouse exit	public void mouseExited(MouseEvent e)
	mouse clicks	public void mouseClicked(MouseEvent e)
		(a mouse click is a press followed by a release in the same location)
MouseMotionListener	mouse move	public void mouseMoved(MouseMotionEvent e)
	mouse drag	public void mouseDragged(MouseMotionEvent e)
KeyListener	key down	public void keyPressed(KeyEvent e)
	key up	public void keyReleased(KeyEvent e)
	key typed	public void keyTyped(KeyEvent e)
		(a key typed is a key down followed by the same key up)

10.6 *Mouse-* und *Key - Eventhandling*

◆ Ein Beispiel mit vollständiger Implementation des Interfaces im Applet:

```
// draw lines at each click and drag

import java.awt.Graphics;
import java.awt.Color;
import java.awt.Point;
import java.awt.event.*;

public class Lines extends java.applet.Applet
    implements MouseListener, MouseMotionListener {

  final int MAXLINES = 10;
  private Point starts[] = new Point[ MAXLINES ]; // starting points
  private Point ends[] = new Point[ MAXLINES ];   // endingpoints
  private Point anchor;         // start of current line
  private Point currentPoint;   // current end of line
  private int currline = 0;     // number of lines

public void init() {
    setBackground( Color.white );
      // register event listeners
    addMouseListener( this );
    addMouseMotionListener( this );
}
```


10.6 *Mouse-* und *Key - Eventhandling*

```
// needed to satisfy listener interfaces

public void mouseMoved( MouseEvent e ) {}
public void mouseClicked( MouseEvent e ) {}
public void mouseEntered( MouseEvent e ) {}
public void mouseExited( MouseEvent e ) {}

public void mousePressed( MouseEvent e ) {
  if ( currline < MAXLINES )
    anchor = new Point( e.getX(), e.getY() );
  else
    System.out.println( "Too many lines." );
  }

public void mouseReleased( MouseEvent e ) {
  if ( currline < MAXLINES )
    addline( e.getX(), e.getY() );
  }

public void mouseDragged( MouseEvent e ) {
  if ( currline < MAXLINES ) {
    currentPoint = new Point( e.getX(), e.getY() );
    repaint();
  }
}
```

10.6 *Mouse-* und *Key - Eventhandling*

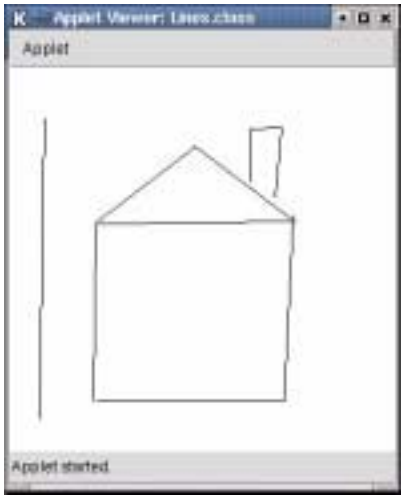
```
void addline( int x, int y ) {
    starts[ currline ] = anchor;
    ends[ currline ] = new Point( x, y );
    currline++;
    currentPoint = null;
    anchor = null;
    repaint();
}

public void paint( Graphics g ) {

   // draw existing lines
  for ( int i = 0; i < currline; i++ ) {
    g.drawLine( starts[i].x, starts[i].y,ends[i].x, ends[i].y );
  }
   // draw current line
  g.setColor( Color.blue );
  if ( currentPoint != null )
    g.drawLine( anchor.x, anchor.y, currentPoint.x, currentPoint.y );
  }
}
```


10.6 *Mouse-* und *Key - Eventhandling*

- Ergebnis mit Appletviewer:



10.6 *Mouse-* und *Key - Eventhandling*

◆ Ein Beispiel mit Testausgaben auf die Java-Console des Browsers:

```
/* press a key then use arrows to move it around */

import java.awt.Graphics;
import java.awt.Color;
import java.awt.Font;
import java.awt.event.*;

public class Keys extends java.applet.Applet
  implements KeyListener {
  private char currkey;
  private int currx;
  private int curry;
  private boolean notmove = true;

  public void init() {
    currx = ( this.size().width / 2 ) -8;
    curry = ( this.size().height / 2 ) -16;
    setBackground( Color.white );
    setFont( new Font( "Helvetica", Font.BOLD, 36 ) );
    addKeyListener( this );
  }
```


10.6 *Mouse-* und *Key - Eventhandling*

◆ Ein Beispiel mit Testausgaben auf die Java-Console des Browsers (cont.):

```
  public void keyReleased( KeyEvent e ) { }
  public void keyTyped( KeyEvent e ) { }

  public void keyPressed( KeyEvent e ) {

    if ( e.getKeyCode() == KeyEvent.VK_DOWN ) {
      curry += 5;
      notmove = false;
    }
    if ( e.getKeyCode() == KeyEvent.VK_UP ) {
      curry -= 5;
      notmove = false;
    }
    if ( e.getKeyCode() == KeyEvent.VK_LEFT ) {
      currx -= 5;
      notmove = false;
    }
    if ( e.getKeyCode() == KeyEvent.VK_RIGHT ) {
      currx += 5;
      notmove = false;
    }
```

10.6 *Mouse-* und *Key - Eventhandling*

◆ Ein Beispiel mit Testausgaben auf die Java-Console des Browsers (cont.):

```
  if ( notmove ) {
       currkey = e.getKeyChar();
    }
    notmove = true;
    repaint();
  }

  public void paint( Graphics g ) {
    if ( currkey != 0 ) {
      g.drawString( String.valueOf( currkey ), currx, curry );
    }
  }
}
```


10.6 *Mouse-* und *Key - Eventhandling*

◆ Ergebnis mit Appletviewer:

10.7 Listener (Übersicht)

Listener	Methods
ActionListener	actionPerformed(ActionEvent e)
AdjustmentListener	adjustmentValueChanged(AdjustmentE-vent e)
ComponentListener	componentResized(ComponentEvent e)
	componentMoved(ComponentEvent e)
	componentShown(ComponentEvent e)
	componentHidden(ComponentEvent e)
ContainerListener	componenentAdded(ContainerEvent e)
	componenentRemoved(ContainerEvent e)
FocusListener	focusGained(FocusEvent e)
	focusLost(FocusEvent e)
ItemListener	itemStateChanged(ItemEvent e)
TextListener	textValueChanged(TextEvent e)

10.7 Listener (Übersicht)

Listener	Methods
WindowListener	windowOpened(WindowEvent e)
	windowClosed(WindowEvent e)
	windowClosing(WindowEvent e)
	windowActivatet(WindowEvent e)
	windowDeactivatet(WindowEvent e)
	windowIconified(WindowEvent e)
	windowDeiconified(WindowEvent e)

◆ Die Methoden der Event-classes finden Sie unter: ***java.awt.event***