Informatik: GUI-Programmierung mit Swing

Ziele

- Die verschiedenen Fensterklassen mit ihren grundlegenden Möglichkeiten unterscheiden können.
- Komponenten in Fenster ohne Verwendung eine Layoutmanagers integrieren können.
- Das Delegation-Ereignis-Modell von Java zur Behandlung von Ereignissen anwenden können.
- Private Klassen als Ereignisabhörer programmieren können.
- Maus-, Tastatur- und Fensterereignisse bearbeiten können.
- Den Unterschied zwischen einem Listener-Interface und der entsprechenden Adapter-Klasse verstehen.
- Einen Abhörer bei mehreren Ereignisquellen registrieren können.
- Mit Meldungsfenstern arbeiten können.
- Ereignisabhörer als anonyme Klasse programmieren können.
- Mit modalen Dialogfenstern arbeiten können.

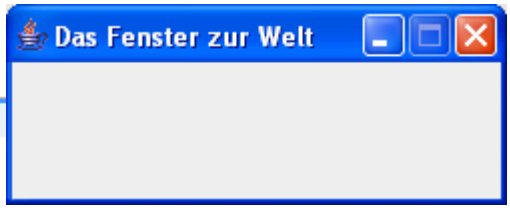
AWT (Abstract Window Toolkit) `java.awt.*`

sehr einfach gehaltene API zum Aufbau grafischer Benutzeroberflächen auf Basis der Möglichkeiten der Zielmaschine.

Swing `javax.swing.*`

vollständig in Java implementierte API mit einer Vielzahl von unterschiedlichen Komponenten.

Fenster mit `JFrame`

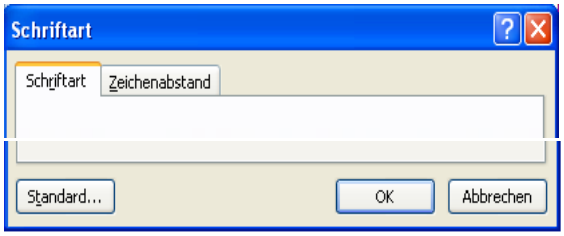


TestFenster.java

```
import javax.swing.*;

public class TestFenster extends JFrame
{
  public static void main(String[] args) {
    TestFenster f = new TestFenster();
    f.setVisible(true);
  }

  public TestFenster() {
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.setSize(250, 100);
    this.setLocation(50,50);
    this.setResizable(false);
    this.setTitle("Das Fenster zur Welt");
  }
}
```

Fenster mit `JDialog`

`JDialog`-Fenster muss geschlossen werden, um im Hauptfenster weiter arbeiten zu können (siehe hinten).



Fenster ohne Rahmen mit `JWindow`

- Ohne Titel, Rahmen, Menü und Symbol.
- Größe nicht änderbar.

Komponenten im Fenster integrieren

- Komponenten müssen zum Container des Fensters (Content-Pane) hinzugefügt werden.
- Die Position der Komponenten im Fenster verwaltet der *Layoutmanager* des Content-Pane.

ACHTUNG: Layoutmanager wird zunächst nicht verwendet!!!

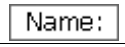
TestFenster.java


```
import javax.swing.*;

public class TestFenster extends JFrame
{
  private JLabel jLabel = null;
  private JTextField jTextField = null;
  private JButton jButton = null;

  public static void main(String[] args) {
    TestFenster f = new TestFenster();
    f.setVisible(true);
  }

  public TestFenster() {
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.setSize(250, 100);
    this.setLocation(50,50);
    this.setResizable(false);
    this.setTitle("Das Fenster zur Welt");

    this.getContentPane().setLayout(null);
    jLabel = new JLabel();
    jLabel.setBounds(10,15,50,30);
    jLabel.setText("Wurzel:");
    this.getContentPane().add(jLabel);
    jTextField = new JTextField();
    jTextField.setBounds(65,15,70,30);
    this.getContentPane().add(jTextField);
    jButton = new JButton();
    jButton.setText("Rechne");
    jButton.setBounds(140,15,90,30);
    this.getContentPane().add(jButton);
  }
```

Ausschalten Layoutmanager

Hinzufügen Content-Pane

HIDE_ON_CLOSE ist standard

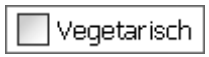


JLabel

JProgressBar

JButton

JList

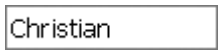


JCheckBox

JTabbedPane

JRadioButton

JToolBar



JTextField

JMenu

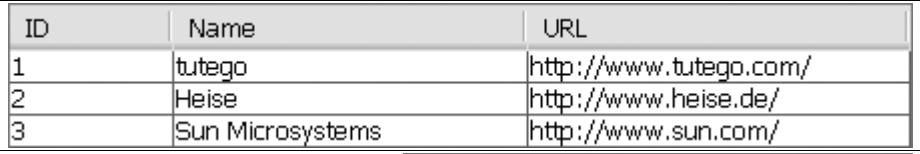
JPasswordField

JScrollPane

JComboBox

JTree

JScrollBar



JTable

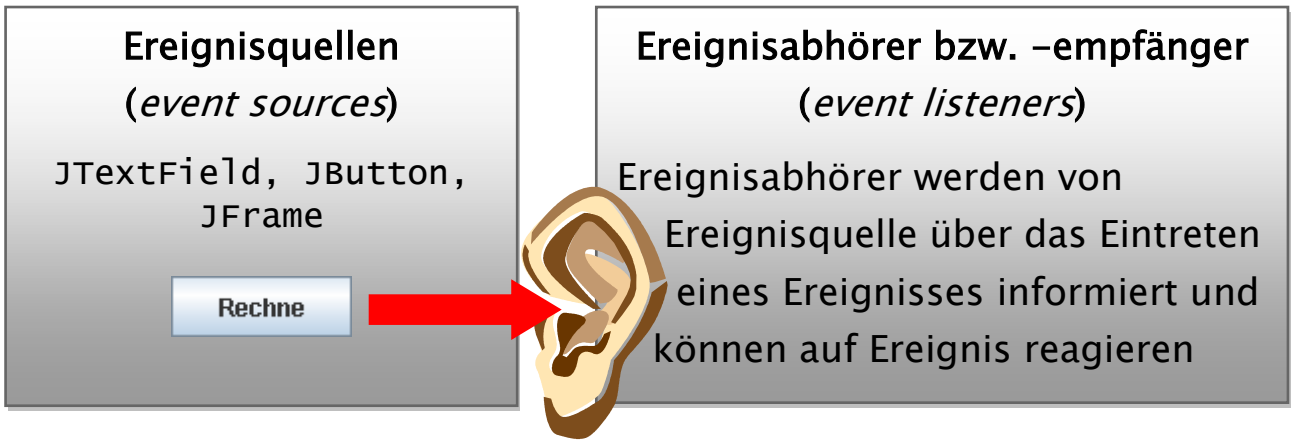
JSlider

JEditorPane

JSpinner

JColorChooser, JFileChooser

Einführung in die Ereignisverarbeitung, Delegations-Ereignis-Modell



- Damit Ereignisquelle weiß, welche Ereignisabhörer sie informieren muss, müssen sich die Ereignisabhörer bei der Ereignisquelle registrieren:
  `addActionListener(ActionListener l)` für `ActionEvent`
  `addMouseListener(MouseListener l)` für `MouseEvent`
  `addKeyListener(KeyListener l)` für `KeyEvent`,
  `removeActionListener(ActionListener l)`
- Ein Ereignisabhörer (der z. B. auf Tastendruck reagiert) muss das Listener-Interface (`KeyListener`) implementieren oder von der Adapter-Klasse (`KeyAdapter`) abgeleitet sein.
- Ereignisabhörer stellen *Ereignismethoden* (z. B. `keyPressed`) bereit, die automatisch aufgerufen werden, wenn passendes Ereignis eintritt.
- Bei einer Ereignisquelle können sich auch mehrere Ereignisabhörer registrieren.
- Ein Ereignisabhörer kann sich bei mehreren Ereignisquellen registrieren.
- Ereignisobjekte (z. B. `KeyEvent`), Listener-Interface und Adapter-Klasse sind im Paket `java.awt.event` deklariert.

Abhörer und ihre Benennungen

Registrierung	Interface	Adapter	Ereignis	Ereignismethoden
addActionListener	ActionListener	-	ActionEvent	actionPerformed
addMouseListener	MouseListener	MouseAdapter	MouseEvent	mouseClicked mouseEntered mouseExited mousePressed mouseReleased
addMouseMotionListener	MouseMotionListener	MouseMotionAdapter	MouseEvent	mouseDragged mouseMoved
addKeyListener	KeyListener	KeyAdapter	KeyEvent	keyPressed keyReleased keyTyped
addWindowListener	WindowListener	WindowAdapter	WindowEvent	windowActivated windowClosed windowClosing windowDeactivated windowDeiconified windowIconified windowOpened

Beispiel

TestFenster.java

```
public class TestFenster extends JFrame
{
  private JLabel jLabel = null;
  private JTextField jTextField = null;
  private JButton jButton = null;

   ...
  public TestFenster() {

     ...
    jButton = new JButton();
    jButton.setText("Rechne");
    jButton.setBounds(140,15,90,30);
    MeinAktionsAbhoerer a = new MeinAktionsAbhoerer();
    jButton.addActionListener(a);
    this.getContentPane().add(jButton);
  }

  private class MeinAktionsAbhoerer implements ActionListener
  {
    public void actionPerformed(ActionEvent e) {
      try {
        double wurzel = Double.parseDouble(jTextField.getText());
        wurzel = Math.sqrt(wurzel);
        jTextField.setText(String.valueOf(wurzel));
      } catch (NumberFormatException ex) {
        jTextField.setText("Fehler");
      }
    }
  }
}
```

- *Private Klasse* kann auf Membervariablen der umschließenden Klasse direkt zugreifen.
- Da `MeinAktionsAbhoerer` das Interface `ActionListener` implementiert, muss **`implements`** geschrieben werden.

Verarbeiten von Tastaturereignissen

Ereignisabhörer müssen ***alle*** Ereignismethoden ihres Interfaces implementieren:

```
public class MeinTastaturAbhoerer implements KeyListener
{
  public void keyPressed(KeyEvent e) { … }
  public void keyReleased(KeyEvent e) { … }
  public void keyTyped(KeyEvent e) { … }
}
```

Um nur die tatsächlich verwendete Ereignismethode implementieren zu müssen, werden die sogenannten *Adapterklassen* bereitgestellt:

```
private class MeinTastaturAbhoerer extends KeyAdapter
{
  public void keyReleased(KeyEvent e) { … }
}
```

TestFenster.java

```
public class TestFenster extends JFrame
{
  private JLabel jLabel = null;
  private JTextField jTextField = null;
  private JButton jButton = null;

   ...
  public TestFenster() {

     ...
    jTextField = new JTextField();
    jTextField.setBounds(65,15,70,30);
    jTextField.addKeyListener(new MeinTastaturAbhoerer());
    this.getContentPane().add(jTextField);
  }

  private class MeinTastaturAbhoerer extends KeyAdapter
  {
    public void keyReleased(KeyEvent e) {
      System.out.print("Text: " + jTextField.getText());
      System.out.print(", Taste: " + e.getKeyChar());
      System.out.println(", Objekt: " + e.getSource().toString());
    }
  }
}
```

`getKeyChar()` liefert die gedrückte Taste.

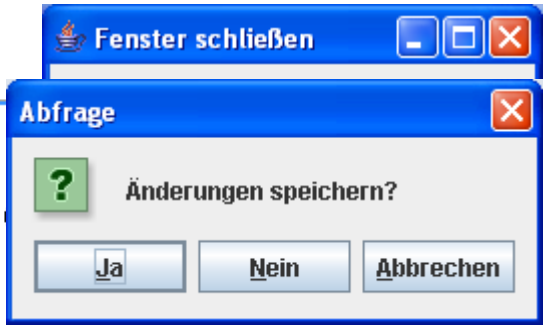
`getSource()` liefert Objekt welches Ereignis hervorgerufen hat.

Mehrere Ereignisquellen, ein Abhörer

EinAktionsAbhoererMehrereQuellen.java

```
public class EinAktionsAbhoererMehrereQuellen extends JFrame
{
  private JTextField jTextField1 = null;
  private JTextField jTextField2 = null;

    ...

  public EinAktionsAbhoererMehrereQuellen() {

      ...
    this.getContentPane().setLayout(null);
    jTextField1 = new JTextField();
    jTextField1.setBounds(10,15,100,30);
    MeinMultiTastaturAbhoerer a = new MeinMultiTastaturAbhoerer();
    jTextField1.addKeyListener(a);
    this.getContentPane().add(jTextField1);

    jTextField2 = new JTextField();
    jTextField2.setBounds(130,15,100,30);
    jTextField2.addKeyListener(a);
    this.getContentPane().add(jTextField2);
  }

  private class MeinMultiTastaturAbhoerer extends KeyAdapter
  {
    public void keyReleased(KeyEvent e) {
      if (e.getSource() == jTextField1)
        jTextField2.setText(jTextField1.getText());
      if (e.getSource() == jTextField2)
        jTextField1.setText(jTextField2.getText());
    }
  }
}
```

Fensterereignisse



FensterSchliessen.java

```
public class FensterSchliessen extends JFrame
{
  public static void main(String[] args) {
    FensterSchliessen f = new FensterSchliessen(
    f.setVisible(true);
  }

  public FensterSchliessen() {
    this.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
    this.setSize(250,100);
    this.setTitle("Fenster schließen");
    this.addWindowListener(new MeinFensterSchliessenAbhoerer());
  }

  private class MeinFensterSchliessenAbhoerer extends WindowAdapter {
    public void windowClosing(WindowEvent e) {
      int ergebnis = JOptionPane.showConfirmDialog(
        FensterSchliessen.this,
        "Änderungen speichern?","Abfrage",JOptionPane.YES_NO_CANCEL_OPTION);
      if (ergebnis == JOptionPane.YES_OPTION)
        speichern();
      if (ergebnis == JOptionPane.YES_OPTION || ergebnis == JOptionPane.NO_OPTION) {
        setVisible(false);
        dispose();
        System.exit(0);
      }
    }
  }
```

- `FensterSchliessen.this`
  legt den Besitzer (engl. *owner*) des Dialoges fest. Dieser ist das umschließende Fensterobjekt in welchem der Abhörer definiert ist. Dieses wird durch *Klassenname*.`this` angesprochen.
- `dispose()`
  gibt alle vom Fenster benötigten Bildschirmressourcen frei, und der Speicherplatz wird dem BS zurückgegeben.
- `System.exit(0)`
  Beendet das laufende Programm.

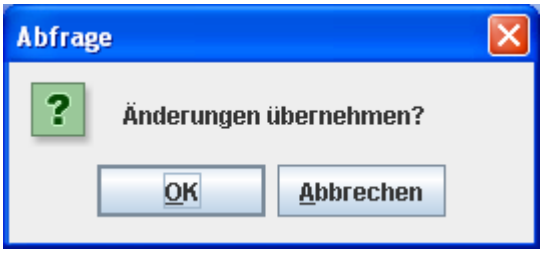
Meldungsfenster

JOptionPane.showMessageDialog()

```
JOptionPane.showMessageDialog(
  FensterSchliessen.this,
  "Warnmeldung",
  "Warnung",
  JOptionPane.WARNING_MESSAGE);
```

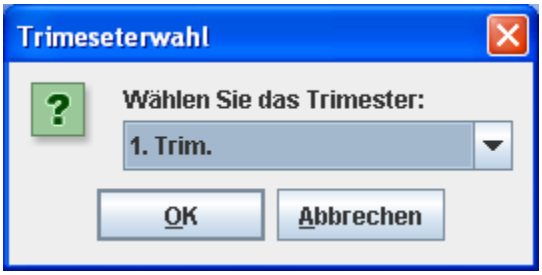
JOptionPane.showConfirmDialog()

```
JOptionPane.showConfirmDialog(
  FensterSchliessen.this,
  "Änderungen übernehmen?",
  "Abfrage",
  JOptionPane.OK_CANCEL_OPTION);
```



JOptionPane.showInputDialog()

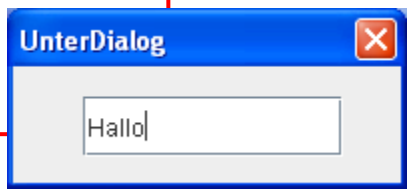
```
String[] moeglichkeiten =
  {"1. Trim.","2. Trim.","3. Trim."};

String auswahl =
  (String)JOptionPane.showInputDialog(
  FensterSchliessen.this,                 // Besitzerfenster
  "Wählen Sie das Trimester:",
  "Trimesterwahl",
  JOptionPane.QUESTION_MESSAGE,           // Art der Meldung
  null,                                   // Icon der Meldung
  moeglichkeiten,
  "1. Trim.");                            // Vorauswahl

if (auswahl == null)
  System.out.println("Nichts ausgewählt");
else
  if (auswahl.equals("1. Trim."))
    System.out.println("1. Trimester ausgewählt");
```



Mit anonymen Klassen arbeiten

HauptFenster.java

```
public class HauptFenster extends JFrame
 {
   private JButton jButton = null;

   public static void main(String[] arg) {
     HauptFenster f = new HauptFenster();
     f.setVisible(true);
   }
   public HauptFenster() {
     this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     this.setSize(220,100);
     this.setLocation(50,50);
     this.setResizable(false);
     this.setTitle("Hauptfenster");

     this.getContentPane().setLayout(null);
     jButton = new JButton();
     jButton.setText("Öffne");
     jButton.setBounds(65,15,90,30);
     jButton.addActionListener(
       new ActionListener() {
         public void actionPerformed(ActionEvent e) {
           UnterDialog u = new UnterDialog(HauptFenster.this);
           u.setVisible(true);
           System.out.println(u.getJTextField());
           u.dispose();
         }
       }
     );
     this.getContentPane().add(jButton);
   }
 }
```



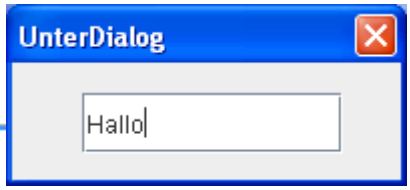
Anonyme Klasse

Klasse welche keinen Namen hat, von einer Klasse oder Interface (z. B. `ActionListener`) abgeleitet und bei ihrer Instanziierung erst programmiert wird:

```
jButton.addActionListener(
    new Basisklasse oder Interface() {
      methodenDerAnonymenKlasse() {
        ...
      }
    }
);
```

ACHTUNG: Klammersetzung

Mit `JDialog` ein modales Fenster öffnen



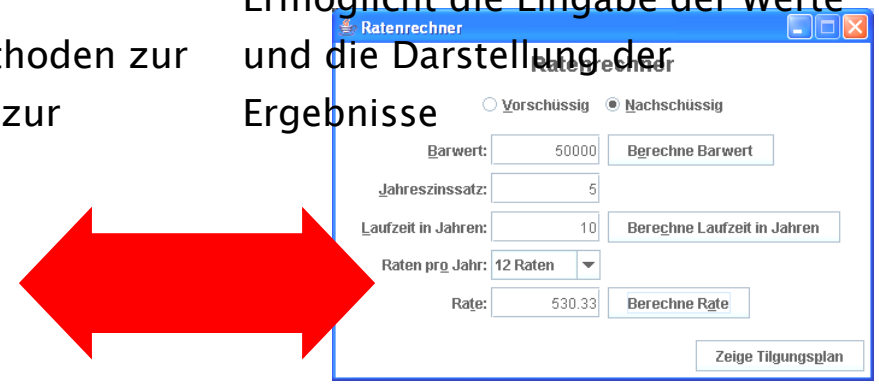
UnterDialog.java

```
public class UnterDialog extends JDialog
 {
   private JTextField jTextField = null;

   public UnterDialog(JFrame owner) {
     super(owner);
     this.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
     this.setModal(true);
     this.setResizable(false);
     this.setSize(200,90);
     this.setTitle("UnterDialog");

     this.getContentPane().setLayout(null);
     jTextField = new JTextField();
     jTextField.setBounds(35,15,130,30);
     this.getContentPane().add(jTextField);
   }

   public String getJTextField() {
     return jTextField.getText();
   }
 }
```

- Das Dialogfenster soll ein Hauptfenster als *Besitzer* (engl. owner) haben.
- `JFrame.HIDE_ON_CLOSE`
  Dialogfenster soll beim Schließen nur unsichtbar gemacht aber nicht geschlossen werden, damit Besitzer noch auf Membervariablen (`jTextField`) zugreifen kann.
- `setModal(true)`
  Dialogfenster muss zuerst unsichtbar ( ) gemacht werden, bevor im Hauptfenster weitergearbeitet werden kann.
- `setVisible(true)` im Hauptfenster
  sorgt dafür, dass die Kontrolle an das Dialogfenster übergeben wird. Erst nachdem Dialogfenster unsichtbar gemacht wurde, wird im Hauptfenster die nächste Programmzeile abgearbeitet.

Trennung zwischen Verarbeitungs- und Darstellungslogik

Klasse `RatenRechner`	Klasse `RatenRechnerGUI`
Stellt die notwendigen Eigenschaften und Methoden zur Berechnung der Raten zur Verfügung	Ermöglicht die Eingabe der Werte und die Darstellung der Ergebnisse



- Wiederverwendbarkeit von Klassen
- Übersichtlichkeit

Nützliches

WindowBuilder

ist ein SWT Designer (Abk. Standard Widget Toolkit)
Bibliothek zur Erstellung grafischer Oberflächen in Java
für Eclipse als Plugin verfügbar
`http://www.eclipse.org/windowbuilder`

Openbook Java ist auch eine Insel

mit ausführlichem Kapitel Grafische Oberflächen mit Swing
`http://www.galileocomputing.de/openbook`