Ereignissteuerung (*event handling*)

Grafische Benutzeroberflächen sollten auf Aktionen reagieren, die der Benutzer auslöst. Diese Aktionen werden als Ereignisse bezeichnet. Damit eine Oberfläche auf ein Ereignis reagieren kann, muss der betreffenden Komponente mitgeteilt werden, dass sie auf dieses Ereignis "hören" soll. Dazu registriert man ein Objekt bei der Komponente, die ein entsprechendes Interface – ein sgn. ***Listener***-Interface – implementiert. Welches *Listener*-Interface man verwendet hängt von der Komponente und dem gewünschten Ereignis ab.

Interface `ActionListener`

Ein Interface, das für viele Komponenten zur Verfügung steht und allgemein für eine Aktion steht, ist das `ActionListener`-Interface:

Package java.awt.event

Interface ActionListener

…

```
public interface ActionListener
extends EventListener
```

The listener interface for receiving action events. The class that is interested in processing an action event implements this interface, and the object created with that class is registered with a component, using the component's `addActionListener` method. When the action event occurs, that object's `actionPerformed` method is invoked.

`void`	**`actionPerformed`**`(`**`ActionEvent`**` e)` Invoked when an action occurs.

In diesem Interface ist eine Methode vorgegeben. Diese Methode wird (sofern ein passendes Objekt registriert ist) von der Komponente aufgerufen, wenn sie über die GUI eine Aktion erhält. Bei einem Button ist so eine Aktion zum Beispiel ein Mausklick oder das Drücken der Enter-Taste, wenn der Button ausgewählt ist. Bei einem Textfeld wäre so eine Aktion nur das Drücken der Enter-Taste, ein Anklicken alleine würde die Methode noch nicht aufrufen.

Beim Aufruf bekommt die Methode von der auslösenden Komponente auch noch ein Objekt der Klasse `ActionEvent` als Parameter mit. Dieses Objekt enthält Informationen über den Auslöser, die mit Hilfe von geeigneten Methoden abgefragt werden können. So kann das `ActionEvent`-Objekt zum Beispiel folgende Informationen geben:

`String`	**`getActionCommand`**`()` Returns the command string associated with this action.
`Object`	**`getSource`**`()` The object on which the Event initially occurred
`long`	**`getWhen`**`()` Returns the timestamp of when this event occurred.

Vor allem die Methode `getSource` wird oft verwendet, um herauszufinden von welcher Komponente das Ereignis eigentlich ausgelöst wurde.

`ActionListener` implementieren

Damit man festlegen kann, welche Anweisungen ausgeführt werden sollen, wenn die Aktion ausgelöst wird, muss man für die Methode einen eigenen Methodenkörper schreiben. Das bedeutet man benötigt eine Klasse, die das Interface `ActionListener` implementiert und damit die Methode `actionPerformed` überschreibt.

Damit man trotzdem noch einfachen Zugriff auf die GUI-Komponenten hat, um sie steuern zu können, kann man diese Klasse als innere Klasse umsetzen. Eine innere Klasse ist eine Klasse, die innerhalb des Klassenblocks (d.h. zwischen den geschwungenen Klammern) definiert wird. Eine innere Klasse kann von außen verwendbar sein und damit public oder aber von außen unsichtbar und damit private. Für eine Ereignissteuerungsklasse genügt es, wenn die Klasse nicht sichtbar ist, d.h. `private`.

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

public class EinPanel extends JPanel {
  private JLabel lAusgabe;
  private JButton bOk, bReset;

  public EinPanel() {
    this.lAusgabe = new JLabel("Hallo!");
    this.add(this.lAusgabe);
    this.bOk = new JButton("Ok!");
    this.add(this.bOk);
    this.bReset = new JButton("Zurück!");
    this.add(this.bReset);
  }

  private class JButtonHandler implements ActionListener {

    @Override
    public void actionPerformed(ActionEvent e) {
      lAusgabe.setText("Geklickt: ");
    }
  }
}
```

import für die Ereignissteuerungsklassen bzw. -interfaces (ActionListener, ActionEvent)

Komponenten, auf die mit einer Ereignissteuerung zugegriffen werden soll, müssen als Attribute in der ganzen Klasse zur Verfügung stehen (trotzdem private)

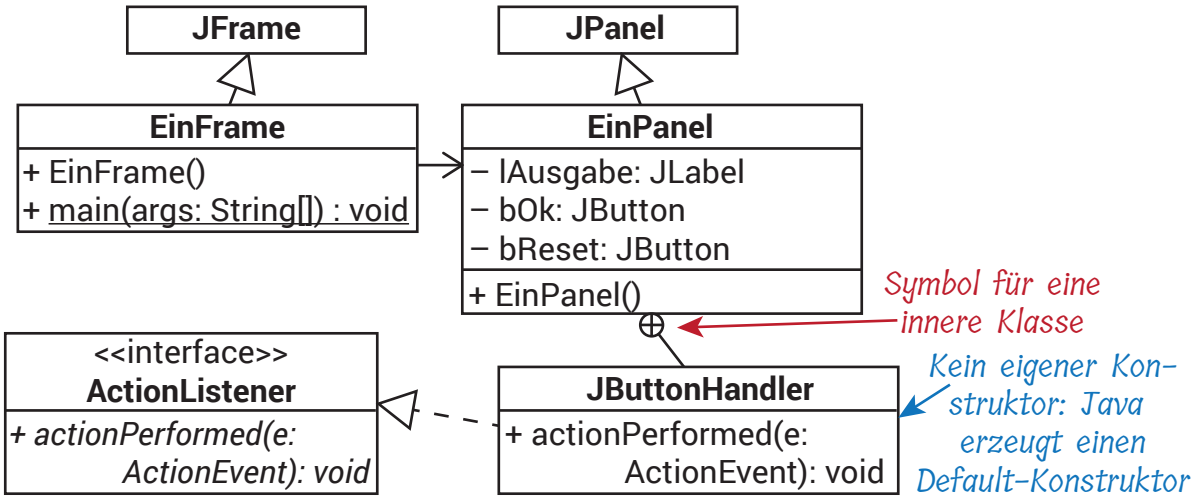
Hier wird das Standardlayout von JPanel verwendet (FlowLayout)

Private innere Klasse *(innerhalb des Klammernpaars der umschließenden Klasse), die das Interface ActionListener implementiert*

Überschreiben der Methode, die im Interface vorgegeben ist

Zugriff auf die GUI-Komponente der umschließenden Klasse. ***Achtung: nicht*** `this` *verwenden, da* `this` *auf die eigene Klasse (in diesem Fall JButtonHandler) zeigt - ohne* `this` *„sucht" Java die Variable zuerst in der Methode, dann in der inneren Klasse und schließlich in der äußeren Klasse.*

Die Frame-Klasse zum Anzeigen des Panels ist hier nicht gegeben, ist aber für das Ausführen notwendig. Damit hat das vollständige Programm folgende Struktur



Ein ActionListener-Objekt registrieren

Ein Ausführen des obigen Codes zeigt noch immer keine Reaktion beim Anklicken des Buttons, da dieser ActionListener noch nicht registriert wurde. Zu diesem Zweck muss ein Objekt aus der Klasse, die den ActionListener implementiert, erzeugt werden. Dieses Objekt wiederum muss mit der Methode addActionListener zu den Buttons hinzugefügt werden:

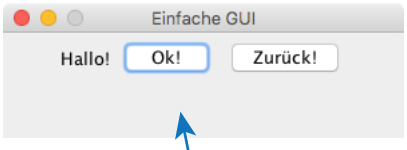
```
import javax.swing.*;
import java.awt.event.*;

public class EinPanel extends JPanel {
  private JLabel lAusgabe;
  private JButton bOk, bReset;

  public EinPanel() {
    this.lAusgabe = new JLabel("Hallo!");
    this.add(this.lAusgabe);
    this.bOk = new JButton("Ok!");
    this.add(this.bOk);
    this.bReset = new JButton("Zurück!");
    this.add(this.bReset);

    JButtonHandler jbh = new JButtonHandler();
    this.bOk.addActionListener(jbh);
    this.bReset.addActionListener(jbh);
  }

  private class JButtonHandler implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
      lAusgabe.setText("Geklickt: ");
    }
  }
}
```

Ein neues Objekt aus der inneren Klasse mit Hilfe des Default-Konstruktors erzeugen

*Dieses Objekt bei den Buttons für die Ereignis-steuerung registrieren**

**(möglich, weil die Klasse des Objekts das Interface ActionListener implementiert)*

Bei diesem Programm wird nun der Text "Geklickt: " angezeigt, sobald einer der beiden Buttons angeklickt wurde:



GUI direkt nach dem Starten

GUI nachdem der Button OK angeklickt wurde

GUI-Komponenten innerhalb von actionPerformed unterscheiden

Im obigen Beispiel passiert immer das gleiche, egal welchen Button man anklickt. Um das zu ändern gibt es zwei Möglichkeiten:

- Für jeden Button eine eigene Klasse schreiben, die ActionListener implementiert. Dieses Prinzip kann allerdings zu einer großen Menge an Klassen führen
- Unterscheidung innerhalb der actionPerformed-Methode

Ich gebe hier das Beispiel mit einer Unterscheidung innerhalb der `actionPerformed`-Methode an, da diese Technik auch später bei der Trennung von GUI und Logik gut eingesetzt werden kann.

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

public class EinPanel extends JPanel {
  private JLabel lAusgabe;
  private JButton bOk, bReset;

  public EinPanel() {
    this.lAusgabe = new JLabel("Hallo!");
    this.add(this.lAusgabe);
    this.bOk = new JButton("Ok!");
    this.add(this.bOk);
    this.bReset = new JButton("Zurück!");
    this.add(this.bReset);

    JButtonHandler jbh = new JButtonHandler();
    this.bOk.addActionListener(jbh);
    this.bReset.addActionListener(jbh);
  }

  private class JButtonHandler implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {

      Object o = e.getSource();

      if (o == bOk) {
        lAusgabe.setText("Geklickt: ");
      } else if (o == bReset) {
        lAusgabe.setText("Hallo!");
      }
    }
  }
}
```

Für beide Buttons wird nach wie vor das gleiche Ereignissteuerungsobjekt verwendet

Nachfragen, von welchem Objekt die Aktion ausgelöst wurde (welches Objekt die Quelle ist)

*Der Referenzvergleich ergibt nur dann `true`, wenn das Objekt, das als Auslöser geholt wurde, das **selbe** Objekt im Speicher ist, wie jenes das von `bOk` referenziert wird.*

... wenn die Aktion von `bOk` kommt

wenn die Aktion von `bReset` ausgelöst, wurde, wird ein anderer Text gesetzt

Nun wird beim Anklicken von Ok! der Text "Geklickt:" angezeigt und beim Anklicken von Zurück! wieder "Hallo!".

GUI direkt nach dem Starten

GUI nachdem der Button Ok! angeklickt wurde

GUI nachdem der Button Zurück! angeklickt wurde

Weitere Attribute und Methoden in `Listener`-Klassen

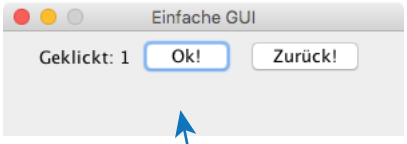
Die inneren Klassen sind vollwertige Klassen und können genauso auch noch Attribute, Konstruktoren und andere Methoden haben.

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

public class EinPanel extends JPanel {
  private JLabel lAusgabe;
  private JButton bOk, bReset;

  public EinPanel() {
    this.lAusgabe = new JLabel("Hallo!");
    this.add(this.lAusgabe);
    this.bOk = new JButton("Ok!");
    this.add(this.bOk);
    this.bReset = new JButton("Zurück!");
    this.add(this.bReset);

    JButtonHandler jbh = new JButtonHandler();
    this.bOk.addActionListener(jbh);
    this.bReset.addActionListener(jbh);
  }

  private class JButtonHandler implements ActionListener {
    private int zaehler;

    public JButtonHandler() {
      zaehler = 0;
    }

    @Override
    public void actionPerformed(ActionEvent e) {
      Object o = e.getSource();
      if (o == bOk) {
        lAusgabe.setText("Geklickt: " + (++zaehler));
      } else if (o == bReset) {
        lAusgabe.setText("Hallo!");
        zaehler = 0;
      }
    }
  }
}
```

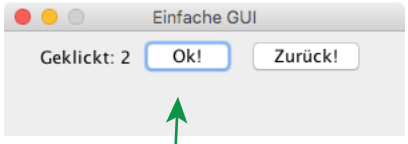
Dieses Attribut wird verwendet, um die Anzahl an Klicks auf den Button Ok! zu zählen.

*Bei jedem Klick wird **zuerst** `zaehler` inkrementiert (weil ++ vor der Variablen steht) und dieser Wert dann ins Label geschrieben*

Beim Klick auf Reset! wird `zaehler` wieder auf 0 gesetzt



GUI nachdem Ok! das 1. Mal angeklickt wurde.



GUI nachdem Ok! zum 2. Mal angeklickt wurde.

GUI nachdem der Button Zurück! angeklickt wurde

Habe ich es verstanden

- Ich weiß, dass `Listener`-Interfaces für die Ereignissteuerung verwendet werden
- Ich kenne das Interface `ActionListener` und kann seine Bedeutung sowie seine Methode erklären.
- Ich kann erklären wann und durch wen die Methode actionPerformed aufgerufen wird.
- Ich kenne den Parameter der Methode `actionPerformed` und kann die wichtigsten Methoden dieses Parameters erklären
- Ich kann eine innere Klasse schreiben, die das Interface `ActionListener` implementiert
- Ich kann die Methode `actionPerformed` so überschreiben, dass sie für eine Ereignissteuerung sinnvoll eingesetzt werden kann.
- Ich kann ein Objekt einer selbst geschriebenen Klasse, die das Interface ActionListener implementiert, als Ereignissteuerungsobjekt für Buttons registrieren.
- Ich kann innerhalb der Methode `actionPerformed` unterscheiden, von welcher Komponente die Methode ausgelöst wurde.
- Ich kann innerhalb der Ereignissteuerungsklasse auch andere Attribute und Methoden einsetzen, um weitere Aufgaben zu erledigen.