Listener 

Stephanie Böhning

Programmieren 2 - Media Systems

Hochschule für Angewandte Wissenschaften Hamburg

Wir wollen User Interaktion!

- Ein User kann auf verschiedene Art mit einem Programm kommunizieren (Mausklick, Button, Häkchen bei einer Checkbox setzen)
- Das Programm soll auf die Eingaben vom User reagieren
- Listener horchen, ob ein User z.B auf den Button klickt und wir sagen dem Listener was dann passieren soll.

Beispiel Button-Klick

```
private class ButtonClick implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        button.setText("ich wurde geklickt");
    }
}
```

Dieses Beispiel ändert den Text des Buttons auf „ich wurde geklickt“

Komponenten und ihre Listener

JButton	ActionListener
JTextField	ActionListener
JSlider	ChangeListener
JCheckBox	ItemListener
JRadioButton	ItemListener
JMenuItem	ItemListener
JComboBox	ItemListener
JList	JListSelectionListener

Listener implementieren

Man kann Listener auf 3 Arten implementieren:

- In einer inneren Klasse (s. Beispielfolie)
- In einer inneren anonymen Klasse
- Mit Lambda-Expression

Innere Klasse

- Man kann eine Klasse in einer Klasse definieren, die von außen aber nicht zugreifbar ist:

```
public class AeussereKlasse {
  ...
  private class InnereKlasse{
     /* ... Definition der inneren Klasse */
  }
} // Ende AeussereKlasse
```

```
public class ButtonClick extends JFrame{

  private JButton button = new JButton("klick mich");

  private class Clicked implements ActionListener {
     @Override
     public void actionPerformed(ActionEvent e) {
        button.setText("ich wurde geklickt");
     }
   }

   public ButtonClick() {
      button.addActionListener(new Clicked());
      add (button);
      setSize(200, 200);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setVisible(true);
}
```

Nachteile

Angenommen man möchte einen Taschenrechner programmieren, dann muss man für jeden einzelnen Button eine Klasse erstellen und das wird sehr schnell unübersichtlich.

Es ist viel Tipparbeit für wenig Inhalt und Informatiker sind faul ;-)

Deswegen gibt es anonyme innere Klassen :-)

Anonyme innere Klassen

- Die Deklaration und Instantiierung werden zusammen gefasst
- Damit steht im Code alles übersichtlich an einer Stelle

```
public class ButtonClick extends JFrame{

    public ButtonClick() {
        JButton button = new JButton("klick mich");

        ActionListener listener = new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent e) {
                button.setText("ich wurde geklickt");
            }
        };

        button.addActionListener(listener);
        add (button);

        setSize(200, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
```

Oder noch kürzer:

```
public class ButtonClick extends JFrame{

    public ButtonClick() {
        JButton button = new JButton("klick mich");

        button.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent e) {
                button.setText("ich wurde geklickt");
            }
        });
        add (button);

        setSize(200, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
```

Lambda-Ausdrücke

```
public class ButtonClick extends JFrame{

public ButtonClick() {
      JButton button = new JButton("klick mich");

      button.addActionListener(listener -> button.setText(
                                          "ich wurde geklickt"));
      add (button);

      setSize(200, 200);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setVisible(true);
   }
```

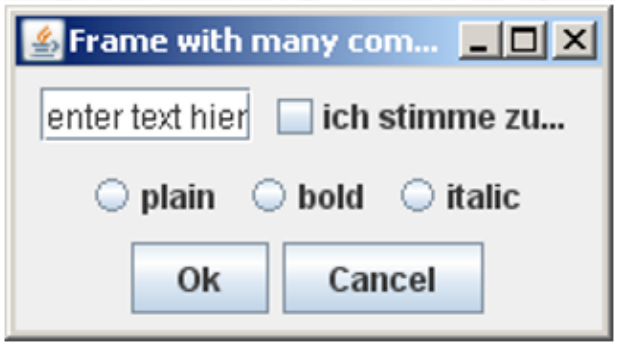
Lambda-Ausdrücke

Achtung:

Lambda-Ausdrücke funktionieren auf diese Weise nur, wenn das Interface nur eine Methode beinhaltet.

Bei WindowsListener, MouseListener, FocusListener... funktioniert das nicht!

Aufgabe 1



Programmiert die GUI oben, dabei soll der Text „*enter text here*“ im Textfield verschwinden, sobald man auf das Textfeld geht.
Klickt man auf *Ok*, sollen die Einstellungen ***Bold*** etc. in der Konsole ausgegeben werden. Bei *Cancel* soll das Fenster wieder in den ursprünglichen Zustand zurück versetzt werden.

Aufgabe 2

Es sollen sowohl bold und italic auswählbar sein, aber plain soll beide Einstellung wegnehmen. Überlegt euch wie ihr das mit Hilfe von ButtonGroups realisiert.

Aufgabe 2

Wenn der Style ***bold*** ausgewählt wurde, soll der Text im Textfeld fett geschrieben sein, bei ***italic*** kursiv. ***Plain*** nimmt beide Einstellungen wieder weg.
Dazu holt man sich vom Textfeld den ***Font***, über den Font kann man die Schriftgröße und Style (fett = 1, kursiv = 2, normal = 0) bestimmen. Verwendet die Font-Konstanten.

`myFont.deriveFont(int style) setzt den gewünschten Style.`

Achtung: es gibt auch ein deriveFont (float size)zum Verändern der Schriftgröße

Aufgabe TicTacToe

Nehmt die Tic Tac Toe Aufgabe und erweitert sie um eine GUI. Dabei sollen die Tic Tac Toe-Felder aus Button bestehen. Zeigt dem User welches Symbol gerade am Zug ist.

View:
Erstellt dazu eine weitere Klasse View, sie soll von JFrame erben und besteht aus einem Konstruktor, der die GUI aufbaut (Layouts setzt, Komponenten erstellt etc) und einer Getter-Methode für die Buttons und einer Setter-Methode für das Symbol.

Aufgabe TicTacToe

Controller:
Der Controller holt sich von der View die Button und fügt ihnen Listener hinzu.

Tipp:
Wenn ihr die Button in einem Array speichert: Man kann jedem Button sagen wie das ActionCommand sein soll, wenn der Button gedrückt wird. Man kann an der Stelle den Index vom Button setzen und der Controller kann ihn mit getActionCommand abfragen und weiß dadurch den Index vom Button.

Gibt es

Fragen ? ? ?