

# Woche 3

## Modul 404

# Agenda

## Input

- Projekt Klassenstruktur
- Java Interface

## Aufgaben Swing

- Buttons aktivieren
- Strings in Zahlen umwandeln
- Account Applikation in Swing

---

### Optional

-  Weitere Swing-Aufgaben
-  Vertieftes Wissen zu Interfaces

# Projekt Klassenstruktur

**Starter** - started das GUI ( `main` )

```
Gui g = new Gui();  
g.dialog();
```

**Gui** - rendert die Grafik

- `extends JFrame` (erbt)
- `implements ActionListener`

```
Logic logic = new Logic()  
logic.do();
```

**Logik** - beinhaltet die Spielregeln



# Java Interface

- Definiert **Methoden-Signaturen**
- Schafft **gemeinsame Basis**
- **Interaktion zwischen Aktoren** wird ermöglicht

Über die **Schnittstelle** vom **Audio Ausgang**, kann nur auf Audio zugegriffen werden. Dafür funktioniert ein Kopfhörer der 80er Jahre auch heute noch! 🎧





# Das Interface ActionListener \*

Konzept ActionListener <-- Lesen!

```
package java.awt.event;  
  
import java.util.EventListener;  
  
public interface ActionListener extends EventListener {  
    public void actionPerformed(ActionEvent e);  
  
}
```

\* ActionListener JavaDoc

# Beispiel ActionListener mit JButton

```
public class GUI extends JFrame implements ActionListener {  
    private JButton button = new JButton("Drücke mich!")  
  
    public void dialog() {  
        // Registriert das Objekt von sich selber (`this`) dem Button als Ziel,  
        // welches informiert werden soll, sobald der Button gedrückt wird.  
        // Signatur: public void addActionListener(ActionListener l) {  
        button.addActionListener(this);  
    }  
  
    // Diese Methode muss durch `interface ActionListener` implementiert werden  
    public void actionPerformed(ActionEvent e) {  
        System.out.println("ich wurde gedrückt");  
    }  
}
```

## Was geht da vor sich?

- Die Entwickler der Klasse  `JButton`  wissen, dass jede Klasse, welche das  `interface`   `ActionListener`  implementiert, die Methode  `public void actionPerformed(ActionEvent e)`  implementiert haben muss.
- Sie können also im Code vom  `JButton`  fest implementieren, dass dem durch die Methode  `addActionListener`  registrierten Objekt die Methode  `actionPerformed`  aufgerufen werden kann.
- Mit  `button.addActionListener(this)`  wird somit  `this`  als Objekt registriert, welches beim drücken durch  `actionPerformed(ActionEvent e)`  informiert werden soll.
- `this`  bezieht sich auf sich selbst, in diesem Beispiel das Objekt der Klasse  `GUI` . Theoretisch könnte dies aber auch ein anderes Objekt sein.

# **Nun seid Ihr dran!**

## Aufgaben

- **Buttons aktivieren**

`implements ActionListener`

anwenden

- **Strings in Zahlen umwandeln**

Braucht man um Zahlen von  
Inputfelder einzulesen

- **Account Applikation in Swing**

Die bereits bestehende App nun in  
Swing 😊



Löst auf der Modulwebseite die Aufgaben  
selbständig weiter.

Ihr dürft den "Konstruktor" und "Easy  
Dice Game" überspringen, diese werden  
nächste Woche genauer betrachtet.





# JFrames mit mehreren Buttons

```
public class GUI extends JFrame implements ActionListener {
    JButton button1 = new JButton("Button 1");
    JButton button2 = new JButton("Button 2");

    public void dialog() {
        button1.addActionListener(this);
        button2.addActionListener(this);
    }

    public void actionPerformed(ActionEvent event) {
        if (event.getSource() == button1) {
            // button1 gedrückt
        } else if (event.getSource() == button2) {
            // button2 gedrückt
        }
    }
}
```

# Abschluss / Lernjournal

# Ab hier ist alles Freiwillig!



**Blättert erst weiter, wenn Ihr alle Aufgaben gelöst habt**

## **Weitere Swing Aufgaben**

Programmieren lernt man nur durch Übung!

Ich kann euch nur wärmstens empfehlen diese Aufgaben zu machen. Die Erste Seite könnt ihr ignorieren. Die Aufgaben werden nicht bewertet. Es zählt der LB1 und die Projektarbeit.

## **Natürlich wird die Projektarbeit davon profitieren!**

-  Weitere Swing-Aufgaben

## **Tieferes Wissen zu Interfaces**

- Das Interface **definiert einen Typ**
- Implementiert eine Klasse ein Interface kann man von dieser Klasse auch ein Objekt vom Typ des Interface erstellen
- Dieses Objekt besitzt jedoch **nur die Methoden, welche vom Interface definiert werden!**
- Beispiel auf der nächsten Seite

# Beispiel Interface als Typ

```
public class GUI implements ActionListener {  
    // Diese Methode muss durch das Interface  
    // implementiert werden  
    public void actionPerformed(ActionEvent e) {  
        System.out.println("ich wurde gedrückt");  
    }  
  
    // Eine zusätzliche Methode, die nicht  
    // vom Interface vorgegeben wird  
    public void halloWelt() {  
        System.out.println("hallo welt");  
    }  
}
```

```
public class Starter {  
    public static void main(String[] args) {  
        GUI gui = new GUI();  
        gui.actionPerformed(new ActionEvent()) // Existiert  
        gui.halloWelt(); // Existiert!  
  
        // Die gleiche Klasse `GUI` als `ActionListener`  
        ActionListener actionListener = (ActionListener) gui;  
  
        // actionPerformed kann immer noch aufgerufen werden  
        // INFO: `new ActionEvent(...)` ist dummy-code und  
        // wird nicht kompilieren  
        actionListener.actionPerformed(new ActionEvent(...))  
  
        // halloWelt existiert nicht mehr!  
        actionListener.halloWelt(); // ERROR!  
    }  
}
```

- JavaDoc [ActionListener](#)

## Was macht nun `button.addActionListener` ?

```
public void addActionListener(ActionListener l) {  
    listenerList.add(ActionListener.class, l);  
}
```

- JavaDoc `JButton.addActionListener`
- Die Methode `addActionListener` erwartet einen Typ `ActionListener` als **Parameter**
- Mit `button.addActionListener(this)`, wird im vorherigen Beispiel das aktuelle Objekt der eigenen Klasse `GUI` übergeben.
- Die Klasse `GUI` **muss** also `implements ActionListener` implementieren