Universität Bremen

Praktische Informatik 1 Graphische Benutzungsoberflächen 1

Thomas Röfer

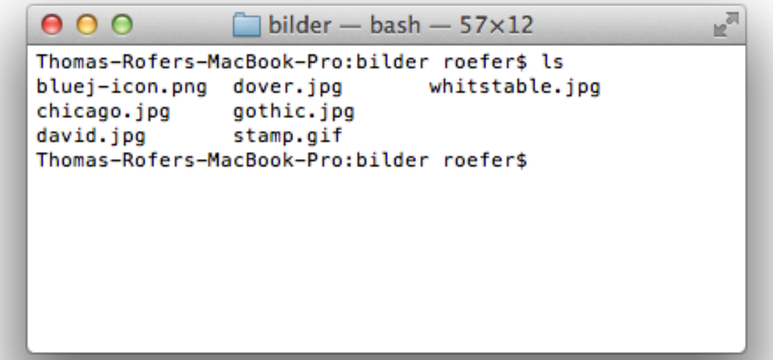
Cyber-Physical Systems
Deutsches Forschungszentrum für Künstliche Intelligenz

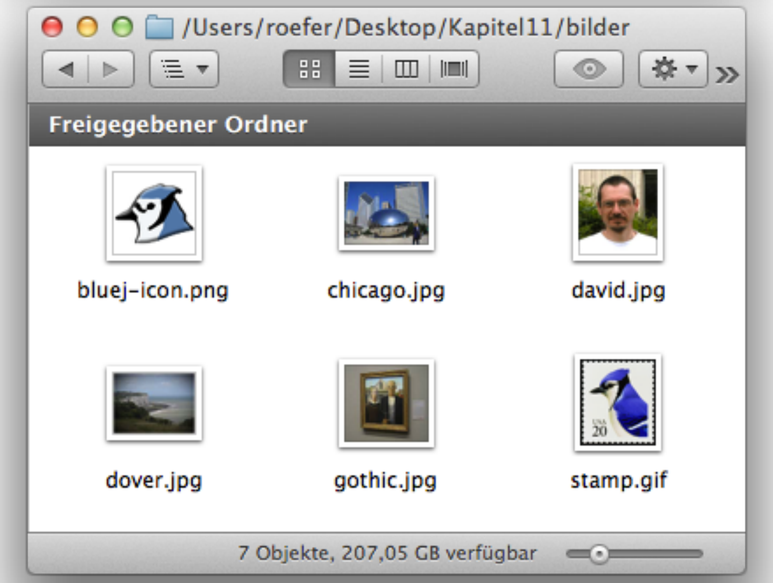
Multisensorische Interaktive Systeme
Fachbereich 3, Universität Bremen

Bild: ©Michael Ihle

Graphische Benutzungsoberflächen (GUI)

- Sollen die Benutzung eines Rechners erleichtern
- Bedienen sich **Metaphern**, d.h. sie bilden reale Objekte und Abläufe nach
- Benutzer:in bestimmt, was zu tun ist, d.h. es gibt keine globale, feste Reihenfolge der Bearbeitung
  - **modal** vs. **nicht-modal**





Universität Bremen

Desktop-Metapher

- **Desktop** (Schreibtisch/Arbeitsfläche): Dinge können abgelegt, hin- und hergeschoben und in den Papierkorb geworfen werden
  - Teilweise schräg: Datenträger in Papierkorb werfen, um ihn auszuwerfen
- **Fenster**: Zeigen einen Ausschnitt der Welt
  - Die Sicht kann durch Rollbalken geändert werden
- **Buttons** (IBM-Deutsch: Schaltflächen)
- **Listen**, **Dialoge**, **Karteikarten**, **Assistenten** …

Universität Bremen

Komponenten, Layout und Ereignisbehandlung

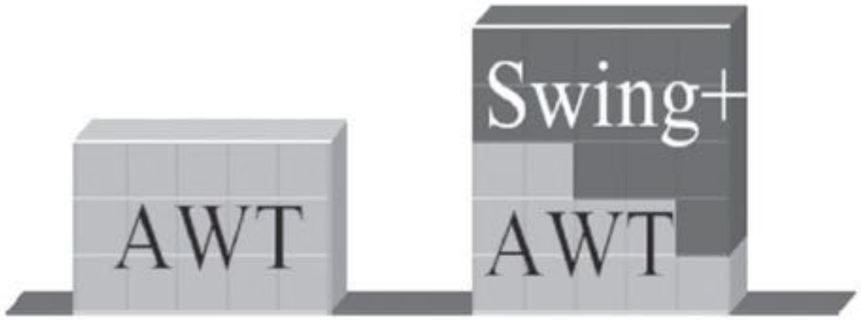
- **Komponenten**: Bausteine, aus denen eine GUI erzeugt wird, indem sie auf dem Bildschirm arrangiert werden
  - Werden durch Objekte repräsentiert
- **Layout**: Anordnung der Komponenten auf dem Bildschirm
  - Wird durch **Layout-Manager** kontrolliert
- **Ereignisverarbeitung**: Reagieren auf Benutzer:innen-Eingaben wie Mausklicks oder Tastendrücke

Universität Bremen

Abstract Window Toolkit (AWT)

- Bildet Komponenten auf Betriebssystem ab
  - Peer-Klassen
  - Oberfläche wirkt nativ
  - Aus Java-Sicht „Schwergewichtige“ Komponenten
- Kleine Zahl von Komponenten
- Gelegentlich unerwünschte Effekte
- Professionelle Oberflächen mühsam
- Findet sich unter **java.awt.***

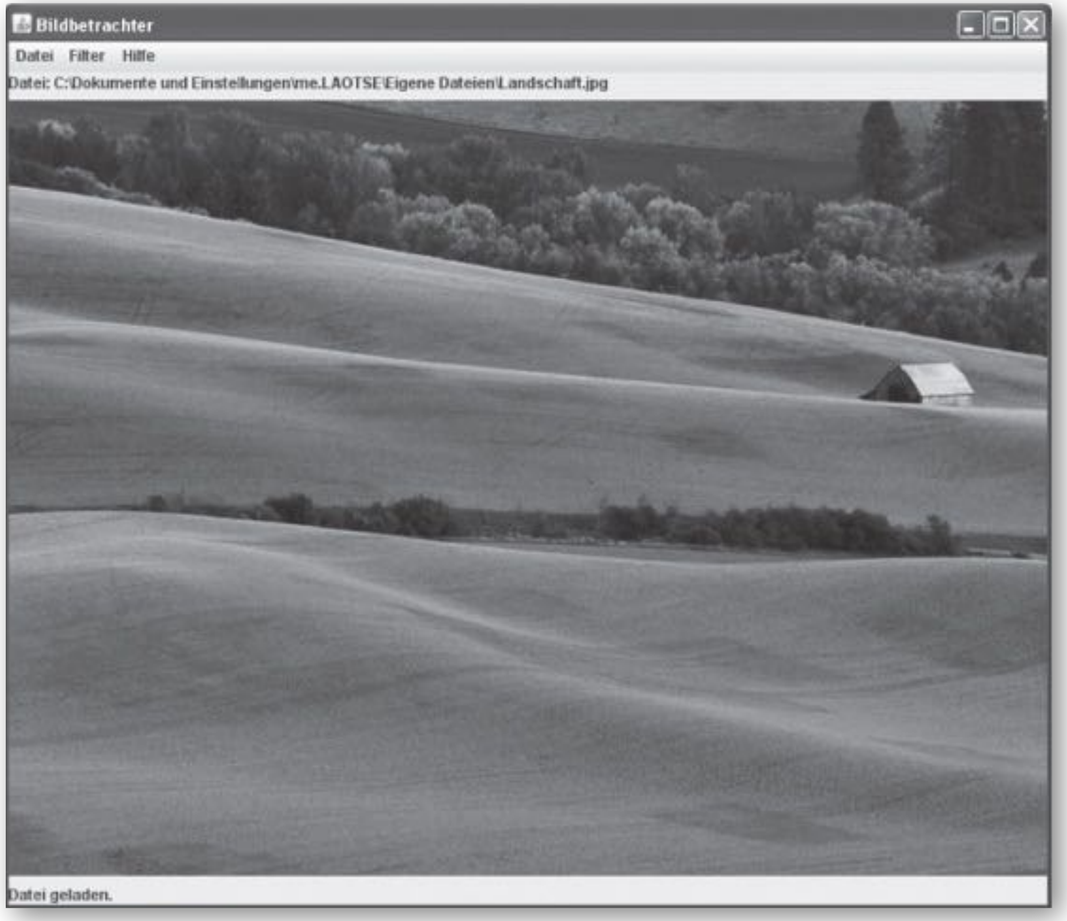
Swing



- Teil der **Java Foundation Classes**
- Aus Java-Sicht „Leichtgewichtige“ Komponenten
- Nutzt Teile von AWT mit
- Der Leistungsumfang ist deutlich größer, kann aber Betriebssystem-Besonderheiten nicht nutzen
- Unterstützt unterschiedliche „Look & Feels“
- Findet sich unter **javax.swing.***
- Weitere GUI-Bibliotheken: **JavaFX**, **Standard Widget Toolkit (SWT)**, **Android UI**…

Universität Bremen

Bildbetrachter-Hauptfenster: Demo



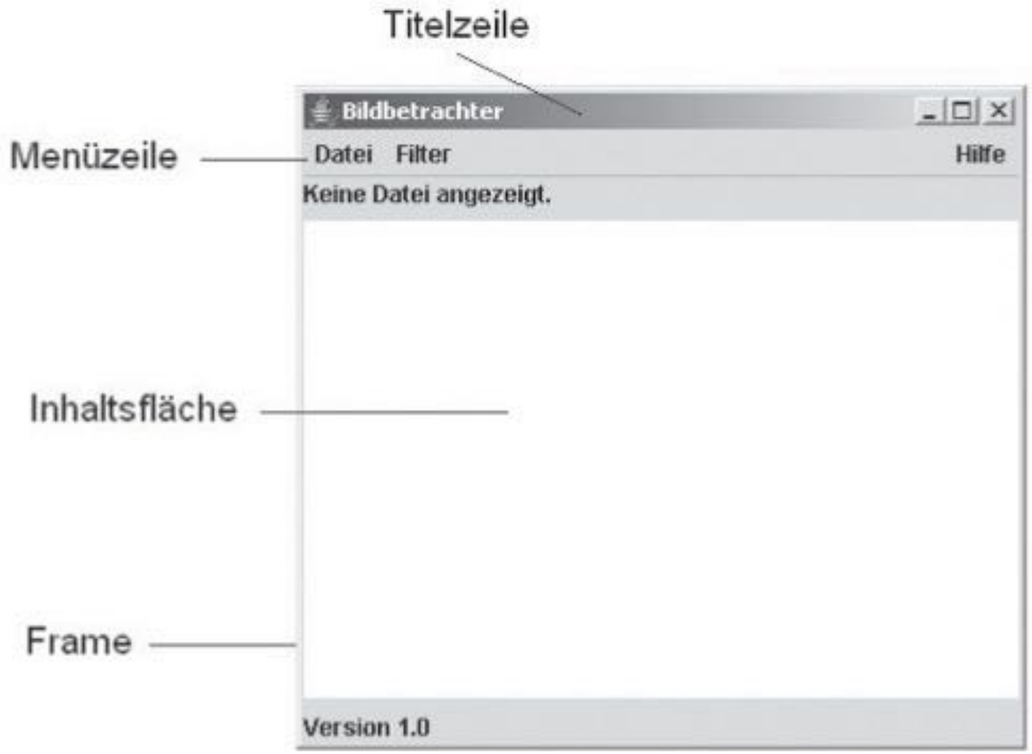
Hauptfenster

- Fenster erzeugen

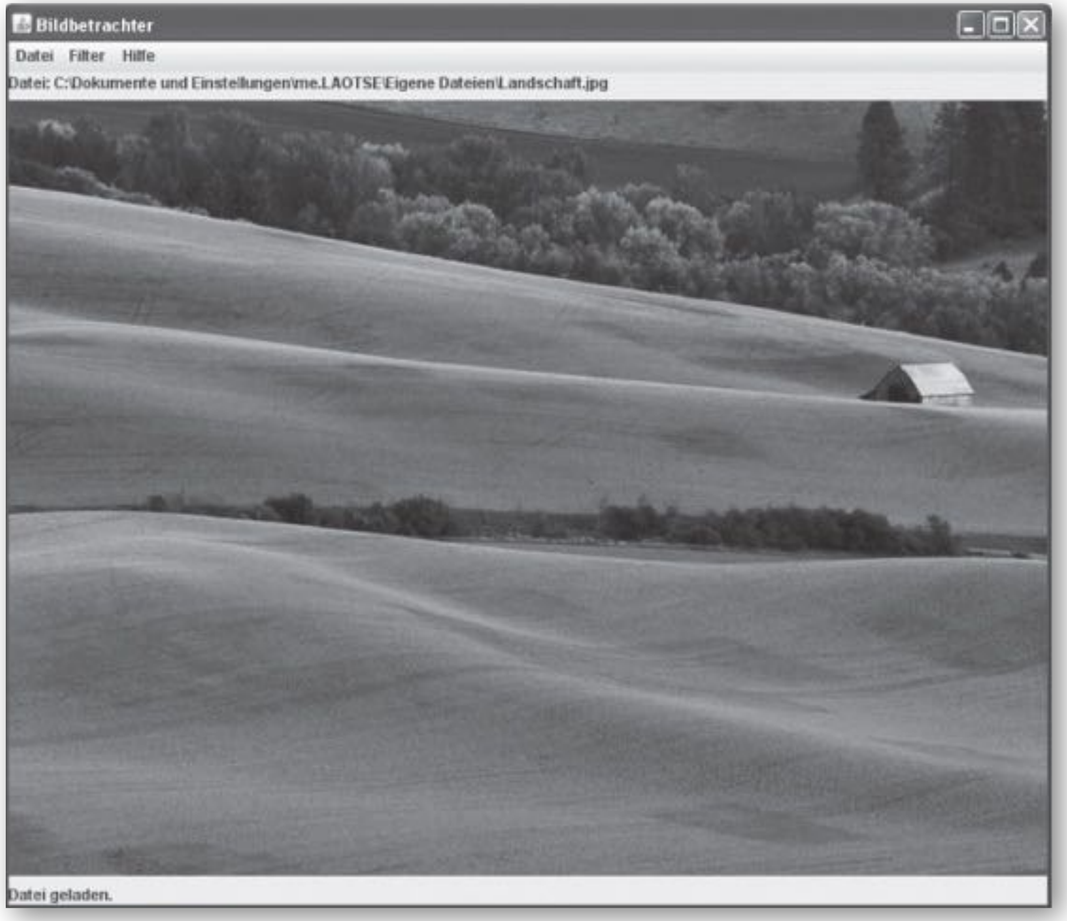
```
fenster = new JFrame("Bildbetrachter");
```

- Inhalt einfügen
  - In Inhaltsfläche und/oder Menüzeile
- Layout berechnen und Fenster anzeigen

```
fenster.pack();
fenster.setVisible(true);
```



Bildbetrachter mit Menü: Demo



Universität Bremen

Menüs hinzufügen

- **JMenuBar**: Menüzeile

```
final JMenuBar menüzeile = new JMenuBar();
fenster.setJMenuBar(menüzeile);
```

- **JMenu**: Einzelnes Menü

```
final JMenu dateiMenü = new JMenu("Datei");
menüzeile.add(dateiMenü);
```

- **JMenuItem**: Einzelner Menüeintrag (Ereignisbehandlung mit **ActionListener**)

```
final JMenuItem öffnenEintrag = new JMenuItem("Öffnen");
öffnenEintrag.addActionListener(this);
dateiMenü.add(öffnenEintrag);
```

Universität Bremen

Ereignisbehandlung

- **Listener** können an Komponenten gehängt werden, die über Ereignisse (…**Event**) informiert werden
- Die Mitteilung erfolgt über den Aufruf von Objekt-Methoden

ActionEvent/Listener
AdjustmentEvent/Listener
ComponentEvent/Listener/Adapter
ContainerEvent/Listener/Adapter
FocusEvent/Listener/Adapter
ItemEvent/Listener
KeyEvent/Listener/Adapter
MouseEvent/Listener/Adapter
TextEvent/Listener
WindowEvent/Listener/Adapter

- Interfaces (…**Listener**) definieren, welche Ereignisse belauschbar sind
- Die Ereignisbehandler heißen oft **process**…**Event**(…**Event** e)
- Für viele **Listener** gibt es **Adapter** als Standardimplementierung

Universität Bremen

Auswahl von Menübefehle behandeln

- Interface **ActionListener** implementieren

```
class Bildbetrachter implements ActionListener
{ ...
```

- Methode **actionPerformed** definieren

```
public void actionPerformed(final ActionEvent event)
{
    System.out.println("Eintrag: " + event.getActionCommand());
}
```

- Ereignisbehandler anmelden

```
öffnenEintrag.addActionListener(this);
```

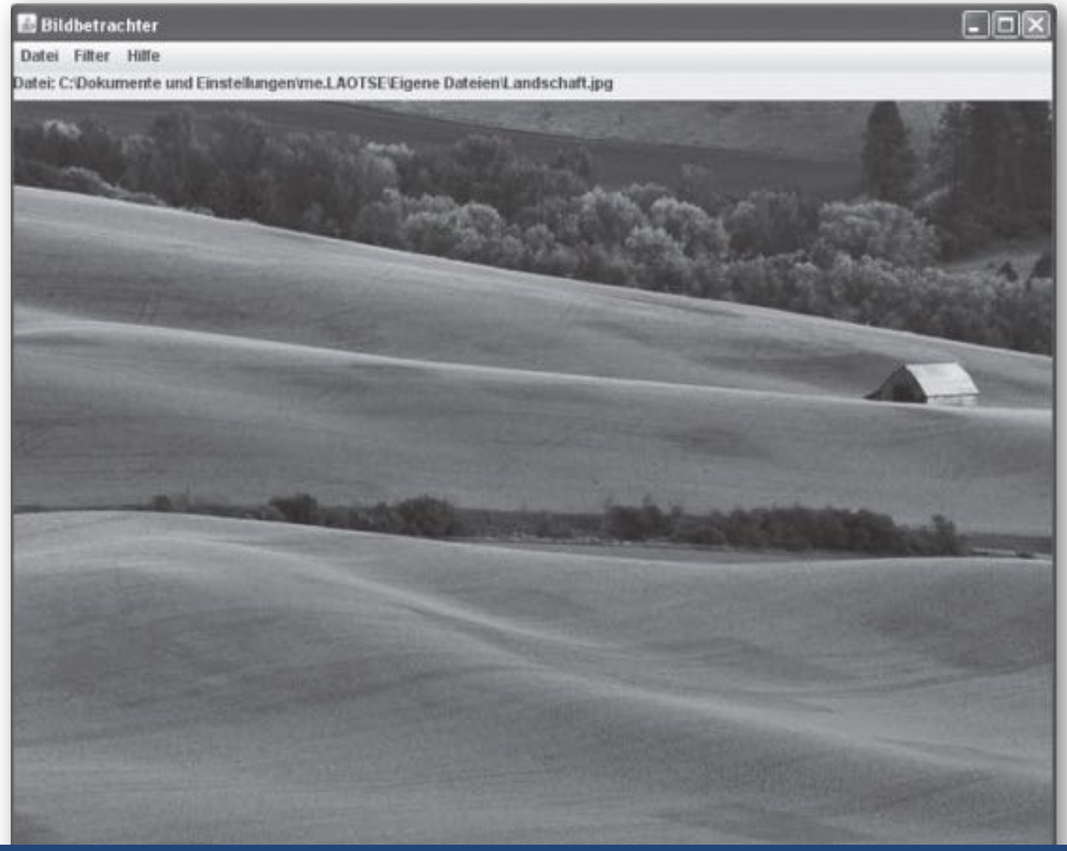
Universität Bremen

Zentrale Ereignisverarbeitung

- Ein Objekt belauscht mehrere Komponenten und verarbeitet alle Ereignisse
- Problem: Objekt muss Ereignis identifizieren können
  - Anhand von Beschriftung? Fehleranfällig, z.B. bei Lokalisierung
  - Methode muss Ereignisse weiter verteilen
- Besser: Eine Methode pro belauschter Komponente

```
String befehl = e.getActionCommand();
if (befehl.equals("Öffnen")) {
    ...
}
else if (befehl.equals("Beenden")) {
    ...
} ...
```

Dezentrale Ereignisbehandlung: Demo




```
fenster.setDefaultCloseOperation(EXIT_ON_CLOSE);
```

Universität Bremen

Innere Klassen

```
class UmschliessendeKlasse
{ ...
    private class InnereKlasse
    { ...
    } ...
}
```

Hat nichts mit Vererbung zu tun!

- Werden textuell innerhalb einer umschließenden Klasse deklariert
- Objekte können nur aus (Objekt-) Methoden der umschließenden Klasse erzeugt werden
- Objekte haben Zugriff auf alle **Methoden** und **Attribute** des **Objekts** der umschließenden Klasse, **das sie erzeugt hat**
- Werden innere Klassen mit **static** deklariert, können ihre Objekte von überall aus erzeugt werden
  - Diese können aber nur auf Klassenmethoden und -attribute der umschließenden Klasse zugreifen (und alles, was normale Klassen auch können)

Universität Bremen

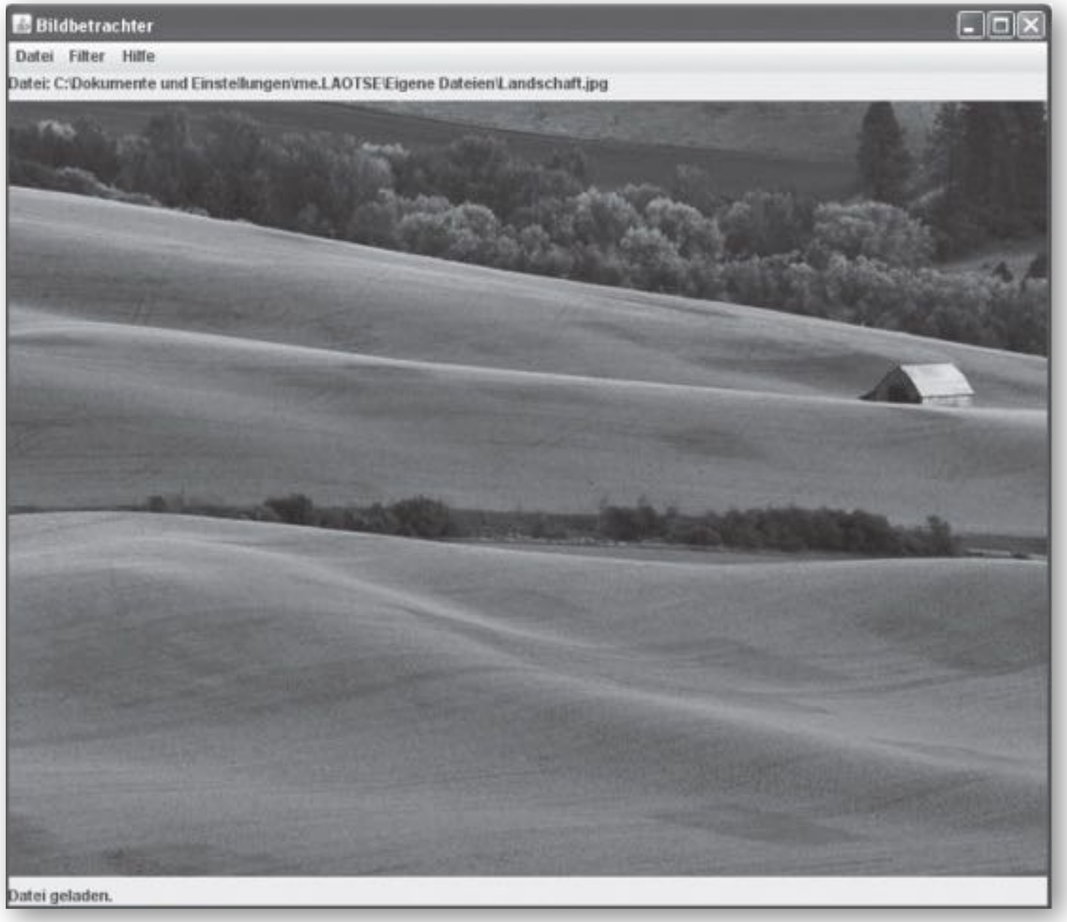
Anonyme innere Klassen

```
öffnenEintrag.addActionListener(
    new ActionListener() {
        public void actionPerformed(
                final ActionEvent e) {
            dateiÖffnen();
        }
    });
```

- Werden definiert, wo eine Instanz erzeugt wird
- Statt eines eigenen Namens wird der Name der **Superklasse** bzw. eines **Interfaces** angegeben
- Haben Zugriff auf lokale (effektiv **final**e) Konstanten der erzeugenden Methode
- Verwendung oft mit **Interfaces** oder **abstrakten Klassen**, die als Parameter verlangt werden
- Wegen erschwerter Lesbarkeit ist Einsatz nur für sehr kurze Klassen und etablierte Programmiermuster geeignet
- **Lambda-Ausdrücke** sind eigentlich anonyme innere Klassen, bei denen nur eine Methode implementiert werden kann

Universität Bremen

Tastenkürzel: Demo



Universität Bremen

Tastenkürzel

```
final KeyStroke keyStroke = KeyStroke.getKeyStroke(
        KeyEvent.VK_F4, InputEvent.ALT_MASK);
beendenEintrag.setAccelerator(keyStroke);
```

- **Virtuelle Tasten-Codes** bezeichnen Tasten auf der Tastatur
  - **KeyEvent**: **VK_A** ... **VK_Z**, **VK_SPACE** ...
- **Umschalter** bezeichnen Tasten, die oft gemeinsam mit anderen Tasten gedrückt werden
  - **InputEvent**: **SHIFT_MASK**, **CTRL_MASK** ...
  - Umschalter können addiert werden (eigentlich mit | ver-odert)
- **KeyStroke**: Tastaturaktion, z.B. Drücken einer Tastenkombination
- **setAccelerator** setzt Tastenkürzel

Universität Bremen

Zusammenfassung der Konzepte

- **Komponenten** und **Ereignisverarbeitung**
- **Menüzeile** und **Inhaltsfläche**
- (**Anonyme**) **innere Klassen**
- **Tastenkürzel**